



**EBook Gratis**

# APRENDIZAJE Ruby on Rails

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

#ruby-on-  
rails

# Tabla de contenido

Acerca de.....	1
<b>Capítulo 1: Empezando con Ruby on Rails.....</b>	<b>2</b>
Observaciones.....	2
Versiones.....	2
Examples.....	3
Creando una aplicación Ruby on Rails.....	3
Cree una nueva aplicación Rails con su base de datos de su elección e incluyendo la herram.....	5
Generando un controlador.....	6
Generar un recurso con andamios.....	7
Cree una nueva aplicación Rails con un adaptador de base de datos no estándar.....	7
Creando APIs de Rails en JSON.....	8
Instalación de rieles.....	9
<b>Capítulo 2: ActionCable.....</b>	<b>12</b>
Observaciones.....	12
Examples.....	12
[Básico] Server Side.....	12
[Básico] Lado del cliente (Coffeescript).....	12
<b>app / asset / javascripts / channels / notifications.coffee.....</b>	<b>12</b>
<b>app / asset / javascripts / application.js # generalmente se genera de esta manera.....</b>	<b>12</b>
<b>La aplicación / asset / javascripts / cable.js # generalmente se genera de esta manera.....</b>	<b>13</b>
Autenticacion de usuario.....	13
<b>Capítulo 3: ActionMailer.....</b>	<b>14</b>
Introducción.....	14
Observaciones.....	14
Examples.....	14
Correo Básico.....	14
<b>user_mailer.rb.....</b>	<b>14</b>
<b>usuario.rb.....</b>	<b>14</b>
<b>homologado.html.erb.....</b>	<b>15</b>
<b>validado.erb.....</b>	<b>15</b>

Generando un nuevo correo.....	15
Agregando Adjuntos.....	15
ActionMailer Callbacks.....	16
Generar un boletín programado.....	16
Interceptor ActionMailer.....	23
<b>Capítulo 4: ActiveJob.....</b>	<b>25</b>
Introducción.....	25
Examples.....	25
Crear el trabajo.....	25
Encolar el trabajo.....	25
<b>Capítulo 5: ActiveRecord.....</b>	<b>26</b>
Observaciones.....	26
Examples.....	26
Utilizando ActiveRecord :: Validaciones.....	26
<b>Capítulo 6: ActiveRecord.....</b>	<b>27</b>
Examples.....	27
Creando un modelo manualmente.....	27
Creando un modelo vía generador.....	27
Creando una migración.....	28
<b>Añadir / eliminar campos en tablas existentes.....</b>	<b>28</b>
<b>Crear una tabla.....</b>	<b>28</b>
<b>Crear una tabla de unión.....</b>	<b>29</b>
<b>Precedencia.....</b>	<b>29</b>
Introducción a las devoluciones de llamada.....	30
Crear una tabla de unión usando migraciones.....	31
Probando manualmente tus modelos.....	31
Usando una instancia de modelo para actualizar una fila.....	32
<b>Capítulo 7: ActiveRecord Locking.....</b>	<b>33</b>
Examples.....	33
Bloqueo optimista.....	33
Bloqueo pesimista.....	33

<b>Capítulo 8: ActiveSupport</b> .....	<b>34</b>
Observaciones.....	34
Examples.....	34
Extensiones de núcleo: String Access.....	34
<b>Cadena # en</b> .....	<b>34</b>
<b>Cadena # de</b> .....	<b>34</b>
<b>Cadena # a</b> .....	<b>34</b>
<b>Cadena # primero</b> .....	<b>35</b>
<b>Cadena # última</b> .....	<b>35</b>
Extensiones de núcleo: cadena a fecha / hora de conversión.....	35
<b>Cadena # to_time</b> .....	<b>35</b>
<b>Cadena # to_date</b> .....	<b>35</b>
<b>Cadena # to_datetime</b> .....	<b>36</b>
Extensiones de núcleo: Exclusión de cadenas.....	36
<b>Cadena # ¿excluir?</b> .....	<b>36</b>
Extensiones de núcleo: Filtros de cadena.....	36
<b>Cuerda # squish</b> .....	<b>36</b>
<b>Cadena # eliminar</b> .....	<b>36</b>
<b>Cadena # truncar</b> .....	<b>37</b>
<b>Cadena # truncate_words</b> .....	<b>37</b>
<b>Cadena # strip_heredoc</b> .....	<b>37</b>
Extensiones de núcleo: Inflexión de cuerdas.....	38
<b>Cadena # pluralizar</b> .....	<b>38</b>
<b>String # singularize</b> .....	<b>38</b>
<b>String # constantize</b> .....	<b>39</b>
<b>Cadena # safe_constantize</b> .....	<b>39</b>
<b>Cuerda # camelize</b> .....	<b>39</b>
<b>Cadena # título</b> .....	<b>39</b>
<b>Cadena # subrayado</b> .....	<b>39</b>
<b>Cadena # dasherizar</b> .....	<b>40</b>

<b>Cadena # demodulizar</b> .....	<b>40</b>
<b>Cadena # desconstantizar</b> .....	<b>40</b>
<b>Cadena # parametrizar</b> .....	<b>40</b>
<b>String # tableize</b> .....	<b>41</b>
<b>Cadena # clasifica</b> .....	<b>41</b>
<b>Cadena # humanizar</b> .....	<b>41</b>
<b>Cadena # upcase_first</b> .....	<b>41</b>
<b>Cadena # foreign_key</b> .....	<b>42</b>
<b>Capítulo 9: Actualización de rieles</b> .....	<b>43</b>
Examples .....	43
Actualización de Rails 4.2 a Rails 5.0 .....	43
<b>Capítulo 10: Agregar panel de administración</b> .....	<b>45</b>
Introducción .....	45
Sintaxis .....	45
Observaciones .....	45
Examples .....	45
Así que aquí hay algunas capturas de pantalla desde el panel de administración usando rail .....	45
<b>Capítulo 11: Agregar un Amazon RDS a su aplicación de rieles</b> .....	<b>49</b>
Introducción .....	49
Examples .....	49
Considera que estamos conectando MYSQL RDS con tu aplicación de rieles .....	49
<b>Capítulo 12: Almacenamiento en caché</b> .....	<b>51</b>
Examples .....	51
Muñeca rusa caching .....	51
SQL Caching .....	51
Almacenamiento en caché de fragmentos .....	52
Almacenamiento en caché de páginas .....	53
Almacenamiento en caché HTTP .....	53
Almacenamiento en caché de acciones .....	54
<b>Capítulo 13: Almacenamiento seguro de claves de autenticación</b> .....	<b>55</b>
Introducción .....	55

Examples.....	55
Almacenando claves de autenticación con Figaro.....	55
<b>Capítulo 14: API de Rails.....</b>	<b>57</b>
Examples.....	57
Creando una aplicación solo para API.....	57
<b>Capítulo 15: Aplicaciones de carriles de prueba.....</b>	<b>58</b>
Examples.....	58
Prueba de unidad.....	58
Solicitud de prueba.....	58
<b>Capítulo 16: Asociaciones ActiveRecord.....</b>	<b>59</b>
Examples.....	59
pertenece a.....	59
Tiene uno.....	59
tiene muchos.....	60
Asociación polimórfica.....	60
El has_many: a través de la asociación.....	61
El has_one: a través de la asociación.....	61
La asociación has_and_belongs_to_many.....	61
Asociación auto-referencial.....	62
<b>Capítulo 17: Autenticación API Rails 5.....</b>	<b>63</b>
Examples.....	63
Autenticación con Rails authenticate_with_http_token.....	63
<b>Capítulo 18: Autenticar Api utilizando Devise.....</b>	<b>64</b>
Introducción.....	64
Examples.....	64
Empezando.....	64
Token de Autenticación.....	64
<b>Capítulo 19: Autenticación de usuario en rieles.....</b>	<b>67</b>
Introducción.....	67
Observaciones.....	67
Examples.....	67
Autenticación utilizando Devise.....	67

Vistas personalizadas .....	68
Diseñar filtros de control y ayudantes .....	68
Omniauth .....	68
has_secure_password .....	69
Crear modelo de usuario .....	69
Agregar el módulo has_secure_password al modelo de usuario .....	69
has_secure_token .....	69
<b>Capítulo 20: Autorización con CanCan .....</b>	<b>71</b>
Introducción .....	71
Observaciones .....	71
Examples .....	71
Empezando con CanCan .....	71
Definiendo habilidades .....	72
Manejando gran cantidad de habilidades .....	72
Prueba rápidamente una habilidad .....	74
<b>Capítulo 21: Ayudantes de formulario .....</b>	<b>75</b>
Introducción .....	75
Observaciones .....	75
Examples .....	75
Crear un formulario .....	75
Creación de un formulario de búsqueda .....	75
Ayudantes para elementos de forma .....	76
Casillas de verificación .....	76
Botones de radio .....	76
Area de texto .....	76
Campo de número .....	77
Campo de contraseña .....	77
Campo de correo electrónico .....	77
Campo telefonico .....	77
Fecha ayudantes .....	77
Desplegable .....	78
<b>Capítulo 22: Cambiar un entorno de aplicación de Rails predeterminado .....</b>	<b>79</b>

Introducción.....	79
Examples.....	79
Corriendo en una máquina local.....	79
Corriendo en un servidor.....	79
<b>Capítulo 23: Cambiar zona horaria predeterminada.....</b>	<b>80</b>
Observaciones.....	80
Examples.....	80
Cambie la zona horaria de Rails, pero continúe guardando Active Record en la base de datos.....	80
Cambiar la zona horaria de Rails Y tener tiempos de almacenamiento Active Record en esta z.....	81
<b>Capítulo 24: Característica de pago en rieles.....</b>	<b>82</b>
Introducción.....	82
Observaciones.....	82
Examples.....	82
Cómo integrar con Stripe.....	82
<b>Cómo crear un nuevo cliente para Stripe.....</b>	<b>82</b>
<b>Cómo recuperar un plan de Stripe.....</b>	<b>83</b>
<b>Cómo crear una suscripción.....</b>	<b>83</b>
<b>Cómo cobrar a un usuario con un solo pago.....</b>	<b>83</b>
<b>Capítulo 25: Cargas de archivos.....</b>	<b>84</b>
Examples.....	84
Carga de un solo archivo usando Carrierwave.....	84
Modelo anidado - subidas múltiples.....	84
<b>Capítulo 26: Carriles 5.....</b>	<b>86</b>
Examples.....	86
Creando una API de Ruby on Rails 5.....	86
Cómo instalar Ruby on Rails 5 en RVM.....	88
<b>Capítulo 27: Chica de fábrica.....</b>	<b>89</b>
Examples.....	89
Definiendo fábricas.....	89
<b>Capítulo 28: Columnas multiusos de ActiveRecord.....</b>	<b>90</b>
Sintaxis.....	90

Examples.....	90
Guardar un objeto.....	90
Cómo.....	90
<b>En tu migración.....</b>	<b>90</b>
<b>En tu modelo.....</b>	<b>90</b>
<b>Capítulo 29: Configuración.....</b>	<b>92</b>
Examples.....	92
Configuración personalizada.....	92
<b>Capítulo 30: Configuración.....</b>	<b>94</b>
Examples.....	94
Entornos en rieles.....	94
Configuración de la base de datos.....	94
Configuración general de rieles.....	95
Configurando activos.....	95
Configurando generadores.....	96
<b>Capítulo 31: Configurar Angular con Rieles.....</b>	<b>97</b>
Examples.....	97
Angular con rieles 101.....	97
<b>Paso 1: Crea una nueva aplicación Rails.....</b>	<b>97</b>
<b>Paso 2: Eliminar Turbolinks.....</b>	<b>97</b>
<b>Paso 3: Agregar AngularJS a la tubería de activos.....</b>	<b>97</b>
<b>Paso 4: Organiza la aplicación Angular.....</b>	<b>98</b>
<b>Paso 5: Bootstrap la aplicación Angular.....</b>	<b>98</b>
<b>Capítulo 32: Constantize seguro.....</b>	<b>100</b>
Examples.....	100
Éxito seguro_constantizar.....	100
Safe_constantize sin éxito.....	100
<b>Capítulo 33: Controlador de acción.....</b>	<b>101</b>
Introducción.....	101
Examples.....	101
Salida JSON en lugar de HTML.....	101

Controladores (básicos).....	101
Parámetros.....	102
Parámetros de filtrado (Básico).....	102
Redirigiendo.....	103
Usando vistas.....	103
404 cuando no se encuentra el registro.....	105
Controlador REST básico.....	105
Mostrar páginas de error para excepciones.....	106
Filtros.....	107
Generando un controlador.....	109
Rescatando ActiveRecord :: RecordNotFound con redirect_to.....	111
<b>Capítulo 34: Convenciones de nombres.....</b>	<b>112</b>
Examples.....	112
Controladores.....	112
Modelos.....	112
Vistas y diseños.....	112
Nombres de archivos y carga automática.....	113
Clase de modelos del nombre del controlador.....	113
<b>Capítulo 35: Depuración.....</b>	<b>115</b>
Examples.....	115
Aplicación de depuración de rieles.....	115
Depurando en tu IDE.....	115
Depuración de Ruby on Rails rápidamente + Consejo para principiantes.....	117
<b>Depurando Ruby / Rails rápidamente:.....</b>	<b>117</b>
1. Método rápido: .inspect una Exception e .inspect su resultado.....	117
2. Fallback: use un depurador IRB de rubí como byebug o pry.....	117
<b>Consejos generales para principiantes.....</b>	<b>117</b>
Por ejemplo, un mensaje de error de Ruby que confunde a muchos principiantes:.....	118
Depuración de la aplicación ruby-on-rails con palanca.....	119
<b>Capítulo 36: Despliegue de una aplicación Rails en Heroku.....</b>	<b>122</b>
Examples.....	122
Desplegando su aplicación.....	122

Gestión de entornos de producción y puesta en escena para un Heroku.....	125
<b>Capítulo 37: Elasticsearch.....</b>	<b>127</b>
Examples.....	127
Instalación y pruebas.....	127
Configuración de herramientas para el desarrollo.....	127
Introducción.....	128
Searchkick.....	128
<b>Capítulo 38: Enrutamiento.....</b>	<b>130</b>
Introducción.....	130
Observaciones.....	130
Examples.....	130
Enrutamiento de recursos (básico).....	130
Restricciones.....	132
Rutas de alcance.....	134
Preocupaciones.....	137
Redirección.....	138
Miembro y rutas de recogida.....	138
URLs params con un punto.....	139
Ruta de la raíz.....	139
Acciones adicionales de REST.....	140
Ámbito local disponible.....	140
Montar otra aplicación.....	141
Redirecciones y rutas de comodines.....	141
Dividir rutas en múltiples archivos.....	141
Rutas anidadas.....	142
<b>Capítulo 39: Enrutamiento superficial.....</b>	<b>143</b>
Examples.....	143
1. Uso de poca profundidad.....	143
<b>Capítulo 40: Estados del modelo: AASM.....</b>	<b>144</b>
Examples.....	144
Estado básico con AASM.....	144
<b>Capítulo 41: Estructuras de rieles a lo largo de los años.....</b>	<b>146</b>

Introducción.....	146
Examples.....	146
¿Cómo encontrar qué marcos están disponibles en la versión actual de Rails?.....	146
Versiones de rieles en rieles 1.x.....	146
Estructuras de rieles en rieles 2.x.....	146
Estructuras de rieles en rieles 3.x.....	146
<b>Capítulo 42: Forma anidada en Ruby on Rails.....</b>	<b>148</b>
Examples.....	148
Cómo configurar un formulario anidado en Ruby on Rails.....	148
<b>Capítulo 43: Gemas.....</b>	<b>150</b>
Observaciones.....	150
Documentación de Gemfile.....	150
Examples.....	150
¿Qué es una gema?.....	150
<b>En tu proyecto Rails.....</b>	<b>150</b>
Gemfile.....	150
Gemfile.lock.....	150
<b>Desarrollo.....</b>	<b>151</b>
Bundler.....	151
Gemfiles.....	151
Gemas.....	152
<b>Capítulo 44: Herencia de una sola mesa.....</b>	<b>155</b>
Introducción.....	155
Examples.....	155
Ejemplo basico.....	155
Columna de herencia personalizada.....	156
Modelo de rieles con columna tipo y sin STI.....	156
<b>Capítulo 45: Herramientas para la optimización y limpieza del código de Ruby on Rails.....</b>	<b>157</b>
Introducción.....	157
Examples.....	157
Si desea mantener su código mantenible, seguro y optimizado, mire algunas gemas para la op.....	157

<b>Capítulo 46: I18n - Internacionalización</b>	<b>159</b>
Syntaxis	159
Examples	159
Usa I18n en vistas	159
I18n con argumentos	159
Pluralización	160
Establecer la configuración regional a través de solicitudes	160
<b>Basado en URL</b>	<b>161</b>
<b>Sesión basada o basada en la persistencia</b>	<b>161</b>
<b>Configuración regional predeterminada</b>	<b>162</b>
Obtener la configuración regional de la solicitud HTTP	162
<b>Limitaciones y alternativas</b>	<b>163</b>
1. Una solución fuera de línea	163
2. Utilice CloudFlare	163
Traducir los atributos del modelo ActiveRecord	164
Utilice I18n con etiquetas HTML y símbolos	166
<b>Capítulo 47: ID amigable</b>	<b>167</b>
Introducción	167
Examples	167
Inicio rápido de rieles	167
<b>Gemfile</b>	<b>167</b>
<b>editar aplicación / modelos / usuario.rb</b>	<b>167</b>
<b>h11</b>	<b>167</b>
<b>h12</b>	<b>167</b>
<b>Capítulo 48: Importar archivos CSV completos desde una carpeta específica</b>	<b>169</b>
Introducción	169
Examples	169
Cargas CSV desde comando de consola	169
<b>Capítulo 49: Integración de React.js con Rails usando Hyperloop</b>	<b>171</b>
Introducción	171
Observaciones	171

Examples.....	171
Agregar un componente de reacción simple (escrito en ruby) a su aplicación Rails.....	171
Declaración de parámetros de componentes (props).....	172
Etiquetas HTML.....	172
Controladores de eventos.....	172
Estados.....	173
Devoluciones de llamada.....	173
<b>Capítulo 50: Interfaz de consulta ActiveRecord.....</b>	<b>174</b>
Introducción.....	174
Examples.....	174
.dónde.....	174
.where con una matriz.....	175
Alcances.....	175
donde no.....	176
Ordenando.....	176
Métodos ActiveRecord Bang (!).....	177
.find_by.....	178
.eliminar todos.....	178
ActiveRecord caso de búsqueda insensible.....	178
Obtener primer y último registro.....	179
.group y .count.....	180
.distinto (o .uniq).....	180
Se une.....	180
Incluye.....	181
Límite y compensación.....	181
<b>Capítulo 51: Los rieles generan comandos.....</b>	<b>183</b>
Introducción.....	183
Parámetros.....	183
Observaciones.....	183
Examples.....	184
Generar rieles modelo.....	184
Rails Generar Migración.....	184

Rieles Generan Andamios.....	185
Rails Generate Controller.....	186
<b>Capítulo 52: Mejores Prácticas de Rieles.....</b>	<b>188</b>
Examples.....	188
No te repitas (SECO).....	188
Convención sobre configuración.....	188
Modelo gordo, flaco controlador.....	189
Cuidado con default_scope.....	190
default_scope y order.....	190
<b>default_scope y modelo de inicialización.....</b>	<b>190</b>
<b>unscoped.....</b>	<b>191</b>
<b>unscoped Asociaciones y Modelo.....</b>	<b>191</b>
<b>Un ejemplo de caso de uso para default_scope.....</b>	<b>192</b>
No lo vas a necesitar (YAGNI).....	192
<b>Problemas.....</b>	<b>192</b>
Sobreingeniería.....	192
Código Inflado.....	193
Característica de arrastramiento.....	193
Largo tiempo de desarrollo.....	193
<b>Soluciones.....</b>	<b>193</b>
KISS - Que sea simple, estúpido.....	193
YAGNI - No lo vas a necesitar.....	193
Refactorización continua.....	193
Objetos de dominio (no más modelos de grasa).....	193
<b>Capítulo 53: Migraciones ActiveRecord.....</b>	<b>197</b>
Parámetros.....	197
Observaciones.....	197
Examples.....	198
Ejecutar migración específica.....	198
Crear una tabla de unión.....	198
Ejecución de migraciones en diferentes entornos.....	198

Agregar una nueva columna a una tabla.....	199
Añadir una nueva columna con un índice.....	199
Eliminar una columna existente de una tabla.....	199
Agregar una columna de referencia a una tabla.....	200
Crear una nueva tabla.....	201
Añadiendo múltiples columnas a una tabla.....	201
Ejecutando migraciones.....	201
Migraciones de retroceso.....	202
Deshacer las últimas 3 migraciones.....	202
Deshacer todas las migraciones.....	202
Mesas cambiantes.....	203
Agregar una columna única a una tabla.....	203
Cambiar el tipo de una columna existente.....	203
Un método más largo pero más seguro.....	204
Rehacer migraciones.....	204
Añadir columna con valor por defecto.....	204
Prohibir valores nulos.....	205
Comprobando el estado de la migración.....	205
Crear una columna hstore.....	206
Añadir una referencia propia.....	206
Crear una columna de matriz.....	206
Agregar una restricción NOT NULL a los datos existentes.....	207
<b>Capítulo 54: Mongoide.....</b>	<b>208</b>
Examples.....	208
Instalación.....	208
Creando un modelo.....	208
Campos.....	209
Asociaciones clásicas.....	209
Asociaciones incrustadas.....	210
Llamadas a bases de datos.....	210
<b>Capítulo 55: Motor de rieles - Rieles modulares.....</b>	<b>211</b>
Introducción.....	211
Sintaxis.....	211

Examples.....	211
Crear una aplicación modular.....	211
<b>Construyendo la lista de Todo.....</b>	<b>212</b>
<b>Capítulo 56: Oleoducto de activos.....</b>	<b>214</b>
Introducción.....	214
Examples.....	214
Tareas de rastrillo.....	214
Archivos y directivas de manifiesto.....	214
Uso básico.....	215
<b>Capítulo 57: Organización de la clase.....</b>	<b>216</b>
Observaciones.....	216
Examples.....	216
Clase de modelo.....	216
Clase de servicio.....	217
<b>Capítulo 58: Palabras reservadas.....</b>	<b>220</b>
Introducción.....	220
Examples.....	220
Lista de palabras reservadas.....	220
<b>Capítulo 59: Patrón decorador.....</b>	<b>227</b>
Observaciones.....	227
Examples.....	227
Decorar un modelo utilizando SimpleDelegator.....	227
Decorando un modelo usando Draper.....	228
<b>Capítulo 60: Prawn PDF.....</b>	<b>229</b>
Examples.....	229
Ejemplo avanzado.....	229
Ejemplo básico.....	230
<b>Esta es la tarea básica.....</b>	<b>230</b>
<b>Podemos hacerlo con Implícito Bloque.....</b>	<b>230</b>
<b>Con bloque explícito.....</b>	<b>230</b>
<b>Capítulo 61: Puntos de vista.....</b>	<b>231</b>

Examples.....	231
Parciales.....	231
<b>Objetos parciales.....</b>	<b>231</b>
<b>Parciales globales.....</b>	<b>231</b>
AssetTagHelper.....	232
<b>Ayudantes de imagen.....</b>	<b>232</b>
ruta de la imagen.....	232
URL de la imagen.....	232
etiqueta_imagen.....	232
<b>Ayudantes de JavaScript.....</b>	<b>232</b>
javascript_include_tag.....	232
javascript_path.....	233
javascript_url.....	233
<b>Ayudantes de hojas de estilo.....</b>	<b>233</b>
stylesheet_link_tag.....	233
stylesheet_path.....	233
stylesheet_url.....	233
<b>Ejemplo de uso.....</b>	<b>233</b>
Estructura.....	234
Reemplazar el código HTML en las vistas.....	234
HAML - una forma alternativa de usar en tus vistas.....	235
<b>Capítulo 62: Rails Cookbook - Recetas / aprendizajes avanzados y técnicas de codificación... 237</b>	
Examples.....	237
Jugando con mesas utilizando la consola de rieles.....	237
Métodos de rieles - devolviendo valores booleanos.....	238
Manejo del error - método indefinido `donde` para #.....	238
<b>Capítulo 63: Reacciona con los rieles usando la gema reaccion-rails..... 239</b>	
Examples.....	239
Reactuar la instalación para los rieles usando rails_react_gema.....	239
Usando react_rails dentro de tu aplicación.....	239
Renderizado y montaje.....	240

<b>Capítulo 64: Registrador de carriles</b>	<b>242</b>
Examples	242
Rails.logger	242
<b>Capítulo 65: Rieles - motores</b>	<b>244</b>
Introducción	244
Sintaxis	244
Parámetros	244
Observaciones	244
Examples	244
Ejemplos famosos son	244
<b>Capítulo 66: Rieles en docker</b>	<b>246</b>
Introducción	246
Examples	246
Docker y docker-componer	246
<b>Capítulo 67: RSpec y Ruby on Rails</b>	<b>248</b>
Observaciones	248
Examples	248
Instalando RSpec	248
<b>Capítulo 68: Serializadores de modelos activos</b>	<b>249</b>
Introducción	249
Examples	249
Usando un serializador	249
<b>Capítulo 69: Trabajos activos</b>	<b>250</b>
Examples	250
Introducción	250
Trabajo de muestra	250
Creación de un trabajo activo a través del generador	250
<b>Capítulo 70: Transacciones ActiveRecord</b>	<b>251</b>
Observaciones	251
Examples	251
Ejemplo basico	251

Diferentes clases de ActiveRecord en una sola transacción.....	251
Conexiones de base de datos múltiples.....	252
guardar y destruir se envuelven automáticamente en una transacción.....	252
Devoluciones de llamada.....	252
Deshacer una transacción.....	253
<b>Capítulo 71: Transacciones ActiveRecord.....</b>	<b>254</b>
Introducción.....	254
Examples.....	254
Comenzando con transacciones de registro activo.....	254
<b>Capítulo 72: Turbolinks.....</b>	<b>256</b>
Introducción.....	256
Observaciones.....	256
<b>Puntos clave:.....</b>	<b>256</b>
Examples.....	256
Enlace al concepto de turbolink de una carga de página.....	256
Desactivar turbolinks en enlaces específicos.....	257
<b>Ejemplos:.....</b>	<b>257</b>
Entender las visitas de aplicaciones.....	257
Cancelando visitas antes de que comiencen.....	258
<b>NOTA:.....</b>	<b>258</b>
Elementos persistentes a través de cargas de página.....	258
<b>Capítulo 73: Usando GoogleMaps con Rails.....</b>	<b>260</b>
Examples.....	260
Agregue la etiqueta javascript de google maps al encabezado de diseño.....	260
Geocodificar el modelo.....	261
Mostrar direcciones en un mapa de Google en la vista de perfil.....	261
Establecer los marcadores en el mapa con javascript.....	262
Inicialice el mapa usando una clase de script de café.....	263
Inicialice los marcadores de mapa usando una clase de script de café.....	264
Zoom automático de un mapa usando una clase de script de café.....	265
Exponiendo las propiedades del modelo como json.....	265
<b>Atributos regulares de la base de datos.....</b>	<b>265</b>

<b>Otros atributos</b> .....	<b>266</b>
<b>Posición</b> .....	<b>266</b>
<b>Capítulo 74: Validación de ActiveRecord</b> .....	<b>268</b>
Examples.....	268
Validando la numericalidad de un atributo.....	268
Validar la singularidad de un atributo.....	268
Validar la presencia de un atributo.....	269
Saltando validaciones.....	269
Validación de la longitud de un atributo.....	270
Validación de agrupación.....	270
Validaciones personalizadas.....	270
ActiveModel::Validator and validates_with.....	271
ActiveModel::EachValidator y validate.....	271
Valida el formato de un atributo.....	271
Valida la inclusión de un atributo.....	272
Validación condicional.....	272
Confirmación de atributo.....	273
Usando: en la opción.....	273
<b>Creditos</b> .....	<b>274</b>

---

## Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [ruby-on-rails](#)

It is an unofficial and free Ruby on Rails ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Ruby on Rails.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

---

# Capítulo 1: Empezando con Ruby on Rails

## Observaciones



Ruby on Rails (RoR), o Rails, es un marco de aplicación web popular de código abierto. Rails utiliza Ruby, HTML, CSS y JavaScript para crear una aplicación web que se ejecuta en un servidor web. Rails utiliza el patrón model-view-controller (MVC) y proporciona una pila completa de bibliotecas desde la base de datos hasta la vista.

## Versiones

Versión	Fecha de lanzamiento
5.1.2	2017-06-26
5.0	2016-06-30
4.2	2014-12-19
4.1	2014-04-08
4.0	2013-06-25
3.2	2012-01-20
3.1	2011-08-31
3.0	2010-08-29
2.3	2009-03-16
2.0	2007-12-07
1.2	2007-01-19
1.1	2006-03-28

Versión	Fecha de lanzamiento
1.0	2005-12-13

## Examples

### Creando una aplicación Ruby on Rails

Este ejemplo asume que *Ruby* y *Ruby on Rails* ya se han instalado correctamente. Si no, puedes encontrar como hacerlo [aquí](#).

Abre una línea de comando o terminal. Para generar una nueva aplicación de rieles, use el [nuevo](#) comando de [rieles](#) seguido del nombre de su aplicación:

```
$ rails new my_app
```

Si desea crear su aplicación Rails con una versión específica de Rails, puede especificarla al momento de generar la aplicación. Para hacerlo, use `rails _version_ new` seguido del nombre de la aplicación:

```
$ rails _4.2.0_ new my_app
```

Esto creará una aplicación de Rails llamada `MyApp` en un directorio `my_app` e instalará las dependencias de gemas que ya se mencionan en `Gemfile` usando la `bundle install`.

Para cambiar al directorio de la aplicación que acaba de crear, use el comando `cd`, que significa `change directory`.

```
$ cd my_app
```

El directorio `my_app` tiene una serie de archivos y carpetas generados automáticamente que conforman la estructura de una aplicación Rails. A continuación se muestra una lista de archivos y carpetas que se crean de forma predeterminada:

Archivo / Carpeta	Propósito
<code>app /</code>	Contiene los controladores, modelos, vistas, ayudantes, correos y activos para su aplicación.
<code>compartimiento/</code>	Contiene el script de Rails que inicia su aplicación y puede contener otros scripts que usa para configurar, actualizar, implementar o ejecutar su aplicación.
<code>config /</code>	Configure las rutas de su aplicación, la base de datos y más.
<code>config.ru</code>	Configuración de rack para servidores basados en Rack utilizados para

Archivo / Carpeta	Propósito
	iniciar la aplicación.
db /	Contiene su esquema de base de datos actual, así como las migraciones de base de datos.
Gemfile Gemfile.lock	Estos archivos le permiten especificar qué dependencias de gemas son necesarias para su aplicación Rails. Estos archivos son utilizados por la gema Bundler.
lib /	Módulos extendidos para su aplicación.
Iniciar sesión/	Archivos de registro de la aplicación.
público/	La única carpeta vista por el mundo tal como es. Contiene archivos estáticos y activos compilados.
Rakefile	Este archivo localiza y carga tareas que pueden ejecutarse desde la línea de comandos. Las definiciones de tareas se definen a través de los componentes de Rails.
README.md	Este es un breve manual de instrucciones para su aplicación. Debería editar este archivo para decir a otros qué hace su aplicación, cómo configurarlo, etc.
prueba/	Pruebas unitarias, accesorios, y otros aparatos de prueba.
temperatura/	Archivos temporales (como caché y archivos pid).
vendedor/	Un lugar para todos los códigos de terceros. En una aplicación típica de Rails esto incluye gemas vendidas.

Ahora necesita crear una base de datos desde su archivo `database.yml` :

5.0

```
rake db:create
# OR
rails db:create
```

5.0

```
rake db:create
```

Ahora que hemos creado la base de datos, necesitamos ejecutar migraciones para configurar las tablas:

5.0

```
rake db:migrate
# OR
rails db:migrate
```

## 5.0

```
rake db:migrate
```

Para iniciar la aplicación, necesitamos iniciar el servidor:

```
$ rails server
# OR
$ rails s
```

De forma predeterminada, los rieles iniciarán la aplicación en el puerto 3000. Para iniciar la aplicación con un número de puerto diferente, debemos iniciar el servidor como,

```
$ rails s -p 3010
```

Si navega a <http://localhost:3000> en su navegador, verá una página de bienvenida de Rails, que muestra que su aplicación se está ejecutando.

Si se produce un error, puede haber varios problemas posibles:

- Hay un problema con la `config/database.yml`
- Tiene dependencias en su `Gemfile` que no se han instalado.
- Tienes migraciones pendientes. Ejecutar `rails db:migrate`
- En caso de que se mueva a los `rails db:rollback` migración anteriores `rails db:rollback`

Si eso sigue generando un error, entonces debería verificar su `config/database.yml`

## Cree una nueva aplicación Rails con su base de datos de su elección e incluyendo la herramienta de prueba RSpec

Rails usa `sqlite3` como la base de datos predeterminada, pero puede generar una nueva aplicación de rails con una base de datos de su elección. Solo agregue la opción `-d` seguida del nombre de la base de datos.

```
$ rails new MyApp -T -d postgresql
```

Esta es una lista (no exhaustiva) de opciones de base de datos disponibles:

- `mysql`
- `oráculo`
- `postgresql`
- `sqlite3`
- `base frontal`
- `ibm_db`

- servidor SQL
- jdbcmysql
- jdbcsqlite3
- jdbcpostgresql
- jdbc

El comando `-T` indica omitir la instalación de minitest. Para instalar un conjunto de pruebas alternativo como [RSpec](#) , edite el Gemfile y agregue

```
group :development, :test do
  gem 'rspec-rails',
end
```

Luego ejecuta el siguiente comando desde la consola:

```
rails generate rspec:install
```

## Generando un controlador

Para generar un controlador (por ejemplo, `Posts` ), navegue hasta el directorio de su proyecto desde una línea de comandos o terminal, y ejecute:

```
$ rails generate controller Posts
```

Puede acortar este código reemplazando `generate` con `g` , por ejemplo:

```
$ rails g controller Posts
```

Si abre la aplicación / controllers / **posts\_controller.rb** recién generada, verá un controlador sin acciones:

```
class PostsController < ApplicationController
  # empty
end
```

Es posible crear métodos predeterminados para el controlador pasando los argumentos de nombre del controlador.

```
$ rails g controller ControllerName method1 method2
```

Para crear un controlador dentro de un módulo, especifique el nombre del controlador como una ruta como `parent_module/controller_name` . Por ejemplo:

```
$ rails generate controller CreditCards open debit credit close
# OR
$ rails g controller CreditCards open debit credit close
```

Esto generará los siguientes archivos:

```
Controller: app/controllers/credit_cards_controller.rb
Test:      test/controllers/credit_cards_controller_test.rb
Views:    app/views/credit_cards/debit.html.erb [...etc]
Helper:   app/helpers/credit_cards_helper.rb
```

Un controlador es simplemente una clase que se define para heredar de `ApplicationController`.

Es dentro de esta clase que definirá métodos que se convertirán en las acciones para este controlador.

## Generar un recurso con andamios

De [guias.rubyonrails.org](https://guides.rubyonrails.org):

En lugar de generar un modelo directamente. . . vamos a configurar un andamio Un andamio en Rails es un conjunto completo de modelos, migración de base de datos para ese modelo, controlador para manipularlo, vistas para ver y manipular los datos y un conjunto de pruebas para cada uno de los anteriores.

Aquí hay un ejemplo de andamiaje de un recurso llamado `Task` con un nombre de cadena y una descripción de texto:

```
rails generate scaffold Task name:string description:text
```

Esto generará los siguientes archivos:

```
Controller: app/controllers/tasks_controller.rb
Test:      test/models/task_test.rb
           test/controllers/tasks_controller_test.rb
Routes:    resources :tasks added in routes.rb
Views:    app/views/tasks
           app/views/tasks/index.html.erb
           app/views/tasks/edit.html.erb
           app/views/tasks/show.html.erb
           app/views/tasks/new.html.erb
           app/views/tasks/_form.html.erb
Helper:   app/helpers/tasks_helper.rb
JS:       app/assets/javascripts/tasks.coffee
CSS:     app/assets/stylesheets/tasks.scss
           app/assets/stylesheets/scaffolds.scss
```

Ejemplo para eliminar los archivos generados por el andamio para el recurso llamado `Task`

```
rails destroy scaffold Task
```

## Cree una nueva aplicación Rails con un adaptador de base de datos no estándar

Rails se envía de forma predeterminada con `ActiveRecord`, un ORM (Mapeo relacional de objetos) derivado del patrón con el [mismo nombre](#).

Como ORM, está diseñado para manejar el mapeo relacional, y más precisamente manejando las solicitudes de SQL para usted, de ahí la limitación a las bases de datos de SQL solamente.

Sin embargo, aún puede crear una aplicación Rails con otro sistema de administración de base de datos:

1. simplemente crea tu aplicación sin registro activo

```
$ rails app new MyApp --skip-active-record
```

2. agrega tu propio sistema de gestión de base de datos en `Gemfile`

```
gem 'mongoid', '~> 5.0'
```

3. `bundle install` y siga los pasos de instalación de la base de datos deseada.

En este ejemplo, `mongoid` es un mapeo de objetos para `MongoDB` y, como muchas otras gemas de base de datos creadas para rieles, también se hereda de `ActiveModel` la misma forma que `ActiveRecord`, que proporciona una interfaz común para muchas funciones como validaciones, devoluciones de llamadas, traducciones, etc. .

Otros adaptadores de base de datos incluyen, pero no se limitan a:

- intercambio de datos
- secuelas-carriles

## Creando APIs de Rails en JSON

Este ejemplo asume que tiene experiencia en la creación de aplicaciones Rails.

Para crear una aplicación solo para API en Rails 5, ejecute

```
rails new name-of-app --api
```

Agregar `active_model_serializers` en `Gemfile`

```
gem 'active_model_serializers'
```

instalar paquete en la terminal

```
bundle install
```

Configure el adaptador `ActiveModelSerializer` para que use `:json_api`

```
# config/initializers/active_model_serializer.rb
ActiveModelSerializers.config.adapter = :json_api
Mime::Type.register "application/json", :json, %w( text/x-json application/jsonrequest
application/vnd.api+json )
```

## Genera un nuevo andamio para tu recurso

```
rails generate scaffold Task name:string description:text
```

Esto generará los siguientes archivos:

Controlador: app / controllers / tasks\_controller.rb

```
Test:          test/models/task_test.rb
               test/controllers/tasks_controller_test.rb
Routes:        resources :tasks added in routes.rb
Migration:     db/migrate/_create_tasks.rb
Model:         app/models/task.rb
Serializer:   app/serializers/task_serializer.rb
Controller:   app/controllers/tasks_controller.rb
```

## Instalación de rieles

### Instalando Rails en Ubuntu

En una ubuntu limpia, la instalación de Rails debería ser sencilla

### Actualizando paquetes de Ubuntu

```
sudo apt-get update
sudo apt-get upgrade
```

### Instalar dependencias de Ruby y Rails

```
sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev libreadline-dev
libyaml-dev libsqlite3-dev sqlite3 libxml2-dev libxslt1-dev libcurl4-openssl-dev python-
software-properties libffi-dev
```

Instalación del gestor de versiones ruby. En este caso lo fácil es usar rbenv.

```
git clone https://github.com/rbenv/rbenv.git ~/.rbenv
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(rbenv init -)"' >> ~/.bashrc
```

### Instalando Ruby Build

```
git clone https://github.com/rbenv/ruby-build.git ~/.rbenv/plugins/ruby-build
echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >> ~/.bashrc
```

### Reiniciar shell

```
exec $SHELL
```

### Instalar ruby

```
rbenv install 2.3.1
rbenv global 2.3.1
rbenv rehash
```

## Instalación de rieles

```
gem install rails
```

## Instalación de rieles en Windows

### Paso 1: *Instalando Ruby*

Necesitamos el lenguaje de programación Ruby instalado. Podemos usar una versión precompilada de Ruby llamada RubyInstaller.

- Descarga y ejecuta Ruby Installer desde [rubyinstaller.org](http://rubyinstaller.org) .
- Ejecuta el instalador. Marque "Agregar ejecutables de Ruby a su RUTA", luego instálelo.
- Para acceder a Ruby, vaya al menú de Windows, haga clic en Todos los programas, desplácese hacia abajo hasta Ruby y haga clic en "Iniciar solicitud de comando con Ruby". Se abrirá un terminal de línea de comandos. Si escribe `ruby -v` y presiona Entrar, debería ver el número de versión de Ruby que instaló.

### Paso 2: *Kit de desarrollo de rubí*

Después de instalar Ruby, podemos intentar instalar Rails. Pero algunas de las bibliotecas de Rails dependen de la necesidad de algunas herramientas de compilación para compilarse, y Windows carece de esas herramientas de forma predeterminada. Puede identificar esto si ve un error al intentar instalar Rails `Gem::InstallError: The '[gem name]' native gem requires installed build tools`. Para solucionar esto, necesitamos instalar el kit de desarrollo de Ruby.

- Descarga el [DevKit](#)
- Ejecuta el instalador.
- Necesitamos especificar una carpeta donde vamos a instalar permanentemente el DevKit. Recomiendo instalarlo en la raíz de su disco duro, en `C:\RubyDevKit` . (No use espacios en el nombre del directorio).

Ahora necesitamos que las herramientas DevKit estén disponibles para Ruby.

- En su símbolo del sistema, cambie al directorio DevKit. `cd C:\RubyDevKit` o cualquier directorio en el que lo haya instalado.
- Necesitamos ejecutar un script Ruby para inicializar la configuración de DevKit. Escribe `ruby dk.rb init` . Ahora le diremos a ese mismo script que agregue el DevKit a nuestra instalación de Ruby. Escriba `ruby dk.rb install` .

El DevKit ahora debería estar disponible para que las herramientas de Ruby lo usen al instalar bibliotecas nuevas.

### Paso 3: *Rieles*

Ahora podemos instalar rieles. Los rieles vienen como una gema de rubí. En su símbolo del sistema, escriba:

```
gem install rails
```

Una vez que presione Entrar, el programa `gem` descargará e instalará esa versión de la gema Rails, junto con todas las otras gemas de las que Rails depende.

#### Paso 4: **Node.js**

Algunas bibliotecas de las que depende Rails requieren que se instale un tiempo de ejecución de JavaScript. Instalemos Node.js para que esas bibliotecas funcionen correctamente.

- Descargue el instalador Node.js desde [aquí](#) .
- Cuando se complete la descarga, visite la carpeta de descargas y ejecute el instalador `node-v4.4.7.pkg` .
- Lea el acuerdo de licencia completo, acepte los términos y haga clic en Siguiente en el resto del asistente, dejando todo en el valor predeterminado.
- Aparecerá una ventana preguntándole si desea permitir que la aplicación realice cambios en su computadora. Haga clic en "Sí".
- Cuando finalice la instalación, deberá reiniciar su computadora para que Rails pueda acceder a Node.js.

Una vez que se reinicie la computadora, no olvide ir al menú de Windows, haga clic en "Todos los programas", desplácese hacia abajo hasta Ruby y haga clic en "Iniciar línea de comando con Ruby".

Lea [Empezando con Ruby on Rails en línea](https://riptutorial.com/es/ruby-on-rails/topic/225/empezando-con-ruby-on-rails): <https://riptutorial.com/es/ruby-on-rails/topic/225/empezando-con-ruby-on-rails>

---

# Capítulo 2: ActionCable

## Observaciones

**ActionCable** estaba disponible para Rails 4.x, y se incluía en Rails 5. Permite un fácil uso de websockets para la comunicación en tiempo real entre el servidor y el cliente.

## Examples

### [Básico] Server Side

```
# app/channels/appearance_channel.rb
class NotificationsChannel < ApplicationCable::Channel
  def subscribed
    stream_from "notifications"
  end

  def unsubscribed
  end

  def notify(data)
    ActionCable.server.broadcast "notifications", { title: 'New things!', body: data }
  end
end
```

### [Básico] Lado del cliente (Coffeescript)

---

## app / asset / javascripts / channels / notifications.coffee

```
App.notifications = App.cable.subscriptions.create "NotificationsChannel",
  connected: ->
    # Called when the subscription is ready for use on the server
    $(document).on "change", "input", (e)=>
      @notify(e.target.value)

  disconnected: ->
    # Called when the subscription has been terminated by the server
    $(document).off "change", "input"

  received: (data) ->
    # Called when there's incoming data on the websocket for this channel
    $('body').append(data)

  notify: (data)->
    @perform('notify', data: data)
```

## app / asset / javascripts / application.js # generalmente se genera de esta manera

```
//= require jquery
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

## La aplicación / asset / javascripts / cable.js # generalmente se genera de esta manera

```
//= require action_cable
//= require_self
//= require_tree ./channels

(function() {
  this.App || (this.App = {});

  App.cable = ActionCable.createConsumer();

}).call(this);
```

## Autenticación de usuario

```
# app/channels/application_cable/connection.rb
module ApplicationCable
  class Connection < ActionCable::Connection::Base
    identified_by :current_user

    def connect
      self.current_user = find_verified_user
      logger.add_tags 'ActionCable', current_user.id
      # Can replace current_user.id with usernames, ids, emails etc.
    end

    protected

    def find_verified_user
      if verified_user = env['warden'].user
        verified_user
      else
        reject_unauthorized_connection
      end
    end
  end
end
```

Lea ActionCable en línea: <https://riptutorial.com/es/ruby-on-rails/topic/1498/actioncable>

---

# Capítulo 3: ActionMailer

## Introducción

Action Mailer le permite enviar correos electrónicos desde su aplicación usando clases y vistas de correo. Los mailers funcionan de manera muy similar a los controladores. Heredan de `ActionMailer::Base` y viven en aplicaciones / correos, y tienen vistas asociadas que aparecen en aplicaciones / vistas.

## Observaciones

Es recomendable procesar el envío de correo electrónico de forma asíncrona para no bloquear su servidor web. Esto se puede hacer a través de varios servicios, como `delayed_job`.

## Examples

### Correo Básico

Este ejemplo utiliza cuatro archivos diferentes:

- El modelo de usuario
- El usuario de correo
- La plantilla html para el correo electrónico.
- La plantilla de texto plano para el correo electrónico.

En este caso, el modelo de usuario llama al método `approved` en la aplicación de correo y pasa la `post` que ha sido aprobada (el método `approved` en el modelo puede llamarse mediante una devolución de llamada, desde un método de controlador, etc.). Luego, el remitente genera el correo electrónico desde la plantilla html o de texto sin formato utilizando la información de la `post` aprobada (por ejemplo, el título). De manera predeterminada, la aplicación de correo usa la plantilla con el mismo nombre que el método en la aplicación de correo (por lo que tanto el método de envío como las plantillas tienen el nombre "aprobado").

---

## user\_mailer.rb

```
class UserMailer < ActionMailer::Base
  default from: "donotreply@example.com"

  def approved(post)
    @title = post.title
    @user = post.user
    mail(to: @user.email, subject: "Your Post was Approved!")
  end
end
```

# usuario.rb

```
def approved(post)
  UserMailer.approved(post)
end
```

---

# homologado.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Post Approved</title>
  </head>
  <body>
    <h2>Congrats <%= @user.name %>! Your post (#<%= @title %>) has been approved!</h2>
    <p>We look forward to your future posts!</p>
  </body>
</html>
```

---

# validado.erb

```
Congrats <%= @user.name %>! Your post (#<%= @title %>) has been approved!
We look forward to your future posts!
```

## Generando un nuevo correo

Para generar un nuevo correo, ingrese el siguiente comando

```
rails generate mailer PostMailer
```

Esto generará un archivo de plantilla en blanco en `app/mailers/post_mailer.rb` llamado *PostMailer*

```
class PostMailer < ApplicationMailer
end
```

También se generarán dos archivos de diseño para la vista de correo electrónico, uno para el formato html y otro para el formato de texto.

Si prefiere no utilizar el generador, puede crear sus propios correos. Asegúrese de que heredan de `ActionMailer::Base`

## Agregando Adjuntos

`ActionMailer` también permite adjuntar archivos.

```
attachments['filename.jpg'] = File.read('/path/to/filename.jpg')
```

De forma predeterminada, los archivos adjuntos se codificarán con `Base64` . Para cambiar esto, puede agregar un hash al método de adjuntos.

```
attachments['filename.jpg'] = {  
  mime_type: 'application/gzip',  
  encoding: 'SpecialEncoding',  
  content: encoded_content  
}
```

También puede agregar archivos adjuntos en línea

```
attachments.inline['image.jpg'] = File.read('/path/to/image.jpg')
```

## ActionMailer Callbacks

ActionMailer soporta tres devoluciones de llamada

- `antes_acción`
- `after_action`
- `alrededor de la acción`

Proporcionar estos en su clase de correo

```
class UserMailer < ApplicationMailer  
  after_action :set_delivery_options, :prevent_delivery_to_guests, :set_business_headers
```

Luego crea estos métodos bajo la palabra clave `private`

```
private  
  def set_delivery_options  
  end  
  
  def prevent_delivery_to_guests  
  end  
  
  def set_business_headers  
  end  
end
```

## Generar un boletín programado

Crear el modelo de **boletín** :

```
rails g model Newsletter name:string email:string  
  
subl app/models/newsletter.rb  
  
validates :name, presence: true  
validates :email, presence: true
```

## Crear el controlador del boletín :

```
rails g controller Newsletters create

class NewslettersController < ApplicationController
  skip_before_action :authenticate_user!
  before_action :set_newsletter, only: [:destroy]

  def create
    @newsletter = Newsletter.create(newsletter_params)
    if @newsletter.save
      redirect_to blog_index_path
    else
      redirect_to root_path
    end
  end

  private

  def set_newsletter
    @newsletter = Newsletter.find(params[:id])
  end

  def newsletter_params
    params.require(:newsletter).permit(:name, :email)
  end
end
```

Después de eso, cambie la vista **create.html.erb** al nombre **nex**. Convertiremos este archivo en una **vista parcial** que se almacenará dentro del **Pie de página** . El nombre será **\_form.html.erb** .

Cambiar archivo de nombre de:	A:
app / views / newsletters / create.html.erb	app / views / newsletters / _form.html.erb

Después de eso se establecen las rutas:

```
subl app/config/routes.rb

resources :newsletters
```

Más adelante, necesitamos configurar el formulario que usaremos para guardar cada correo:

```
subl app/views/newsletters/_form.html.erb

<%= form_for (Newsletter.new) do |f| %>
  <div class="col-md-12" style="margin: 0 auto; padding: 0;">
    <div class="col-md-6" style="padding: 0;">
      <%= f.text_field :name, class: 'form-control', placeholder:'Nombre' %>
    </div>
    <div class="col-md-6" style="padding: 0;">
      <%= f.text_field :email, class: 'form-control', placeholder:'Email' %>
    </div>
  </div>
  <div class="col-md-12" style="margin: 0 auto; padding:0;">
```

```
<%= f.submit class:"col-md-12 tran3s s-color-bg hvr-shutter-out-horizontal",
style:'border: none; color: white; cursor: pointer; margin: 0.5em auto; padding: 0.75em;
width: 100%;' %>
</div>
<% end %>
```

Y después de eso, inserte en el pie de página:

```
subl app/views/layouts/_footer.html.erb

<%= render 'newsletters/form' %>
```

Ahora, instale - **letter\_opener** - para obtener una vista previa del correo electrónico en el navegador predeterminado en lugar de enviarlo. Esto significa que no necesita configurar la entrega de correo electrónico en su entorno de desarrollo, y ya no tiene que preocuparse por enviar accidentalmente un correo electrónico de prueba a la dirección de otra persona.

Primero agregue la gema a su entorno de desarrollo y ejecute el comando bundle para instalarlo.

```
subl your_project/Gemfile

gem "letter_opener", :group => :development
```

A continuación, establezca el método de entrega en el entorno de desarrollo:

```
subl your_project/app/config/environments/development.rb

config.action_mailer.delivery_method = :letter_opener
```

Ahora, cree una **estructura de Mailer** para administrar todos los correos con los que trabajaremos. En la terminal

```
rails generate mailer UserMailer newsletter_mailer
```

Y dentro del **UserMailer** , tenemos que crear un método llamado **Newsletter Mailer** que se creará para contener dentro de la última publicación del blog y se activará con una acción de rake. Asumiremos que antes tenías una estructura de blog creada.

```
subl your_project/app/mailers/user_mailer.rb

class UserMailer < ActionMailer::Base
  default_url_options { { host: "localhost:3000" } }

  def newsletter_mailer
    @newsletter = Newsletter.all
    @post = Post.last(3)
    emails = @newsletter.collect(&:email).join(", ")
    mail(to: emails, subject: "Hi, this is a test mail.")
  end
end

end
```

## Después de eso, crea la plantilla de Mailer :

```
subl your_project/app/views/user_mailer/newsletter_mailer.html.erb

<p> Dear Followers: </p>
<p> Those are the latest entries to our blog. We invite you to read and share everything we
did on this week. </p>

<br/>
<table>
<% @post.each do |post| %>
  <%#= link_to blog_url(post) do %>
    <tr style="display:flex; float:left; clear:both;">
      <td style="display:flex; float:left; clear:both; height: 80px; width: 100px;">
        <% if post.cover_image.present? %>
          <%= image_tag post.cover_image.fullsize.url, class:"principal-home-image-slider"
%>
        <%# else %>
          <%#= image_tag 'http://your_site_project.com' + post.cover_video,
class:"principal-home-image-slider" %>
          <%#= raw(video_embed(post.cover_video)) %>
        <% end %>
      </td>
      <td>
        <h3>
          <%= link_to post.title, :controller => "blog", :action => "show", :only_path =>
false, :id => post.id %>
        </h3>
        <p><%= post.subtitle %></p>
      </td>
      <td style="display:flex; float:left; clear:both;">

    </td>
  </tr>
<%# end %>
<% end %>
</table>
```

Como queremos enviar el correo electrónico como un proceso separado, creamos una tarea de Rake para activar el correo electrónico. Agregue un nuevo archivo llamado `email_tasks.rake` al directorio `lib / tasks` de su aplicación Rails:

```
touch lib/tasks/email_tasks.rake

desc 'weekly newsletter email'
task weekly_newsletter_email: :environment do
  UserMailer.newsletter_mailer.deliver!
end
```

El entorno `send_digest_email::` significa cargar el entorno Rails antes de ejecutar la tarea, para que pueda acceder a las clases de la aplicación (como `UserMailer`) dentro de la tarea.

Ahora, al ejecutar el comando `rake -T` se listará la tarea Rake recién creada. Probar todo funciona ejecutando la tarea y verificando si el correo electrónico se envía o no.

Para probar si el método de envío de correo funciona, ejecute el comando `rake`:

```
rake weekly_newsletter_email
```

En este punto, tenemos una tarea de rastreo de trabajo que se puede programar usando **crontab** . Así que instalaremos la **gema cuando** se use para proporcionar una sintaxis clara para escribir y desplegar trabajos cron.

```
subl your_project/Gemfile

gem 'whenever', require: false
```

Después de eso, ejecute el siguiente comando para crear un archivo config / schedule.rb inicial para usted (siempre que la carpeta de configuración ya esté presente en su proyecto).

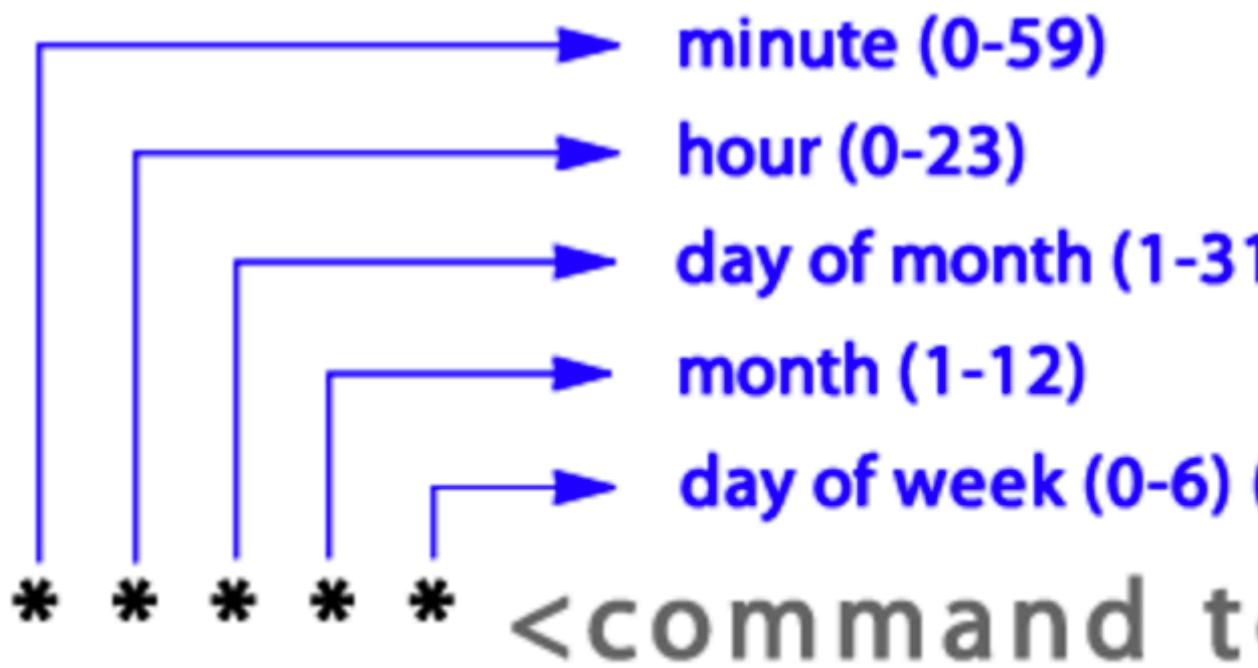
```
wheneverize .

[add] writing `./config/schedule.rb'
[done] wheneverized!
```

Ahora, dentro del archivo de programación, tenemos que crear nuestro **TRABAJO CRON** y llamar al método de correo dentro de la determinación del TRABAJO CRON para operar algunas tareas sin asistencia y en un intervalo de tiempo seleccionado. Puede usar diferentes tipos de sintaxis como se explica en este [enlace](#) .

```
subl your_project/config/schedule.rb

every 1.day, :at => '4:30 am' do
  rake 'weekly_newsletter_email'
end
```



```
every 3.hours do # 1.minute 1.day 1.week 1.m
  runner "MyModel.some_process"
  rake "my:rake:task"
  command "/usr/bin/my_great_command"
end

every 1.day, :at => '4:30 am' do
  runner "MyModel.task_to_run_at_four_thirty"
end

every :hour do # Many shortcuts available: :
  runner "SomeModel.ladeeda"
end

every :sunday, :at => '12pm' do # Use any da
  runner "Task.do_something_great"
end

every '0 0 27-31 * *' do
  command "echo 'you can use raw cron syntax'"
end

# run this task only on servers with the :ap
# see Capistrano roles section below
every :day, :at => '12:20am', :roles => [:ap
  rake "app_server:task"
end
```

fue creado con éxito, podemos usar el siguiente comando para leer desde la terminal, nuestro trabajo programado en CRON SYNTAX:

```
your_project your_mac_user$ whenever

30 4 * * * /bin/bash -l -c 'cd /Users/your_mac_user/Desktop/your_project &&
RAILS_ENV=production bundle exec rake weekly_newsletter_email --silent'
```

Ahora, para ejecutar la prueba en un entorno de desarrollo, es aconsejable establecer la siguiente línea en el archivo principal **application.rb** para que la aplicación sepa dónde están los modelos que utilizará.

```
subl your_project/config/application.rb

config.action_mailer.default_url_options = { :host => "http://localhost:3000/" }
```

Ahora para permitir que **Capistrano V3** guarde el nuevo **trabajo Cron** dentro del servidor y el desencadenante que activará la ejecución de esta tarea, tenemos que agregar el siguiente requisito:

```
subl your_project/Capfile

require 'whenever/capistrano'
```

E inserte en el archivo de **implementación** el identificador que **CRON JOB** usará sobre el **entorno** y el nombre de la **aplicación** .

```
subl your_project/config/deploy.rb

set :whenever_identifier, ->{ "#{fetch(:application)}_#{fetch(:rails_env)}" }
```

Y listo, después de guardar los cambios en cada archivo, ejecute el comando de implementación de capistrano:

```
cap production deploy
```

Y ahora su TRABAJO se creó y se calendario para ejecutar el Método Mailer, que es lo que quiero y en el intervalo de tiempo que establecemos en estos archivos.

## Interceptor ActionMailer

Action Mailer proporciona enlaces a los métodos de interceptor. Estos le permiten registrar clases que se llaman durante el ciclo de vida de la entrega de correo.

Una clase de interceptor debe implementar el método: `deliver_email (message)` que se llamará antes de que se envíe el correo electrónico, permitiéndole realizar modificaciones en el correo electrónico antes de que llegue a los agentes de entrega. Su clase debe realizar las modificaciones necesarias directamente en la instancia de `Mail :: Message`.

Puede ser útil para los desarrolladores enviar correos electrónicos a ellos mismos, no a usuarios reales.

Ejemplo de registro de un interceptor actionmailer:

```
# config/initializers/override_mail_recipient.rb

if Rails.env.development? or Rails.env.test?
  class OverrideMailRecipient
    def self.delivering_email(mail)
      mail.subject = 'This is dummy subject'
      mail.bcc = 'test_bcc@noemail.com'
      mail.to = 'test@noemail.com'
    end
  end
  ActionMailer::Base.register_interceptor(OverrideMailRecipient)
end
```

Lea ActionMailer en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2481/actionmailer>

---

# Capítulo 4: ActiveJob

## Introducción

Trabajo activo es un marco para declarar trabajos y hacer que se ejecuten en una variedad de backends de cola. Estos trabajos pueden ser desde limpiezas programadas regularmente, cargos de facturación y envíos por correo. Cualquier cosa que se pueda dividir en pequeñas unidades de trabajo y correr en paralelo, de verdad.

## Examples

### Crear el trabajo

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

  def perform(*guests)
    # Do something later
  end
end
```

### Encolar el trabajo

```
# Enqueue a job to be performed as soon as the queuing system is free.
GuestsCleanupJob.perform_later guest
```

Lea ActiveJob en línea: <https://riptutorial.com/es/ruby-on-rails/topic/8996/activejob>

---

# Capítulo 5: ActiveRecord

## Observaciones

ActiveModel fue creado para extraer el comportamiento del modelo de ActiveRecord en una preocupación separada. Esto nos permite usar el comportamiento de ActiveModel en cualquier objeto, no solo en los modelos ActiveRecord.

Los objetos ActiveRecord incluyen todo este comportamiento por defecto.

## Examples

### Utilizando ActiveModel :: Validaciones

Puedes validar cualquier objeto, incluso rubí liso.

```
class User
  include ActiveModel::Validations

  attr_reader :name, :age

  def initialize(name, age)
    @name = name
    @age = age
  end

  validates :name, presence: true
  validates :age, numericality: { only_integer: true, greater_than: 12 }
end
```

```
User.new('John Smith', 28).valid? #=> true
User.new('Jane Smith', 11).valid? #=> false
User.new(nil, 30).valid?          #=> false
```

Lea ActiveModel en línea: <https://riptutorial.com/es/ruby-on-rails/topic/1773/activemodel>

---

# Capítulo 6: ActiveRecord

## Examples

### Creando un modelo manualmente

Si bien el uso de andamios es rápido y fácil si usted es nuevo en Rails o si está creando una nueva aplicación, puede ser útil hacerlo solo para evitar la necesidad de pasar por el código generado por el andamio para reducirlo. (retire las piezas no utilizadas, etc.).

Crear un modelo puede ser tan simple como crear un archivo en `app/models`.

El modelo más simple, en `ActiveRecord`, es una clase que extiende `ActiveRecord::Base`.

```
class User < ActiveRecord::Base
end
```

Los archivos de modelo se almacenan en `app/models/`, y el nombre del archivo corresponde al nombre singular de la clase:

```
# user
app/models/user.rb

# SomeModel
app/models/some_model.rb
```

La clase heredará todas las características de `ActiveRecord`: métodos de consulta, validaciones, devoluciones de llamada, etc.

```
# Searches the User with ID 1
User.find(1)
```

Nota: Asegúrese de que existe la tabla para el modelo correspondiente. Si no, puede crear la tabla creando una [Migración](#)

Puede generar un modelo y su migración por terminal desde el siguiente comando

```
rails g model column_name1:data_type1, column_name2:data_type2, ...
```

y también puede asignar una clave externa (relación) al modelo siguiendo el comando

```
rails g model column_name:data_type, model_name:references
```

### Creando un modelo vía generador

Ruby on Rails proporciona un generador de `model` que puede utilizar para crear modelos `ActiveRecord`. Simplemente use los `rails generate model` y proporcionar el nombre del modelo

```
$ rails g model user
```

Además del archivo de modelo en `app/models` , el generador también creará:

- La prueba en `test/models/user_test.rb`
- los accesorios en `test/fixtures/users.yml`
- la migración de la base de datos en `db/migrate/XXX_create_users.rb`

También puede generar algunos campos para el modelo cuando lo genere.

```
$ rails g model user email:string sign_in_count:integer birthday:date
```

Esto creará las columnas `email`, `sign_in_count` y `birthday` en su base de datos, con los tipos adecuados.

## Creando una migración

# Añadir / eliminar campos en tablas existentes

Creando una migración ejecutando:

```
rails generate migration AddTitleToCategories title:string
```

Esto creará una migración que agrega una columna de `title` a una tabla de `categories` :

```
class AddTitleToCategories < ActiveRecord::Migration[5.0]
  def change
    add_column :categories, :title, :string
  end
end
```

De forma similar, puede generar una migración para eliminar una columna: los `rails generate migration RemoveTitleFromCategories title:string`

Esto creará una migración que elimina una columna de `title` de la tabla de `categories` :

```
class RemoveTitleFromCategories < ActiveRecord::Migration[5.0]
  def change
    remove_column :categories, :title, :string
  end
end
```

Aunque, estrictamente hablando, especificar el **tipo** ( `:string` en este caso) **no es necesario** para eliminar una columna, **es útil** , ya que proporciona la información necesaria para **revertirla** .

## Crear una tabla

Crea una migración ejecutando:

```
rails g CreateUsers name bio
```

Rails reconoce la intención de crear una tabla a partir del prefijo `Create`, el resto del nombre de la migración se utilizará como un nombre de tabla. El ejemplo dado genera lo siguiente:

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :bio
    end
  end
end
```

Observe que el comando de creación no especificó los tipos de columnas y se usó la `string` predeterminada.

---

## Crear una tabla de unión

Crea una migración ejecutando:

```
rails g CreateJoinTableParticipation user:references group:references
```

Rails detecta la intención de crear una tabla de `JoinTable` encontrando `JoinTable` en el nombre de la migración. Todo lo demás se determina a partir de los nombres de los campos que da después del nombre.

```
class CreateJoinTableParticipation < ActiveRecord::Migration
  def change
    create_join_table :users, :groups do |t|
      # t.index [:user_id, :group_id]
      # t.index [:group_id, :user_id]
    end
  end
end
```

Descomente las declaraciones de `index` necesarias y elimine el resto.

---

## Precedencia

Observe que el nombre de migración de ejemplo `CreateJoinTableParticipation` coincide con la regla para la creación de tablas: tiene un prefijo `Create`. Pero no generó un simple `create_table`. Esto se debe a que el generador de migración ( [código fuente](#) ) utiliza una **primera coincidencia** de la siguiente lista:

- (Add|Remove) <ignored> (To|From) <table\_name>

- <ignored>JoinTable<ignored>
- Create<table\_name>

## Introducción a las devoluciones de llamada

Una devolución de llamada es un método que recibe llamadas en momentos específicos del ciclo de vida de un objeto (justo antes o después de la creación, eliminación, actualización, validación, guardado o carga desde la base de datos).

Por ejemplo, supongamos que tiene un listado que caduca dentro de los 30 días de la creación.

Una forma de hacerlo es así:

```
class Listing < ApplicationRecord
  after_create :set_expiry_date

  private

  def set_expiry_date
    expiry_date = Date.today + 30.days
    self.update_column(:expires_on, expiry_date)
  end
end
```

Todos los métodos disponibles para las devoluciones de llamada son los siguientes, en el mismo orden en que se llaman durante la operación de cada objeto:

### Creando un objeto

- validación anterior
- after\_validation
- antes\_save
- alrededor\_save
- antes\_crear
- alrededor\_crear
- after\_create
- after\_save
- after\_commit / after\_rollback

### Actualizando un objeto

- validación anterior
- after\_validation
- antes\_save
- alrededor\_save
- before\_update
- around\_update
- después de la actualización
- after\_save
- after\_commit / after\_rollback

## Destruyendo un objeto

- `antes_destroy`
- `alrededor_destroy`
- `after_destroy`
- `after_commit` / `after_rollback`

**NOTA:** `after_save` se ejecuta en crear y actualizar, pero siempre después de las devoluciones de llamada más específicas `after_create` y `after_update`, sin importar el orden en que se ejecutaron las llamadas de macro.

## Crear una tabla de unión usando migraciones

Especialmente útil para la relación `has_and_belongs_to_many`, puede crear manualmente una tabla de unión usando el método `create_table`. Supongamos que tiene dos modelos de `Tags` y `Projects`, y le gustaría asociarlos utilizando una relación `has_and_belongs_to_many`. Necesita una tabla de unión para asociar instancias de ambas clases.

```
class CreateProjectsTagsJoinTableMigration < ActiveRecord::Migration
  def change
    create_table :projects_tags, id: false do |t|
      t.integer :project_id
      t.integer :tag_id
    end
  end
end
```

El nombre real de la tabla debe seguir esta convención: el modelo que precede alfabéticamente al otro debe ir primero. **P**roject precede a **T**ags por lo que el nombre de la tabla es `projects_tags`.

Además, dado que el propósito de esta tabla es enrutar la asociación entre las instancias de dos modelos, la identificación real de cada registro en esta tabla no es necesaria. Usted especifica esto pasando `id: false`

Finalmente, como es convencional en Rails, el nombre de la tabla debe ser la forma plural compuesta de los modelos individuales, pero la columna de la tabla debe estar en forma singular.

## Probando manualmente tus modelos

La prueba de sus modelos de Active Record a través de su interfaz de línea de comandos es simple. Vaya al directorio de aplicaciones en su terminal y escriba la `rails console` de Rails para iniciar la consola de Rails. Desde aquí, puede ejecutar métodos de registro activo en su base de datos.

Por ejemplo, si tuviera un esquema de base de datos con una tabla de Usuarios con un `name:string` columna de `name:string` y `email:string`, podría ejecutar:

```
User.create name: "John", email: "john@example.com"
```

Entonces, para mostrar ese registro, podrías correr:

```
User.find_by email: "john@example.com"
```

O si este es su primer o único registro, simplemente puede obtener el primer registro ejecutando:

```
User.first
```

## Usando una instancia de modelo para actualizar una fila

Digamos que tienes un modelo de `User`

```
class User < ActiveRecord::Base
end
```

Ahora para actualizar el `first_name` y `last_name` de un usuario con `id = 1`, se puede escribir el siguiente código.

```
user = User.find(1)
user.update(first_name: 'Kashif', last_name: 'Liaqat')
```

La `update` llamada intentará actualizar los atributos dados en una sola transacción, devolviendo `true` si es exitoso y `false` si no.

Lea `ActiveRecord` en línea: <https://riptutorial.com/es/ruby-on-rails/topic/828/activerecord>

---

# Capítulo 7: ActiveRecord Locking

## Examples

### Bloqueo optimista

```
user_one = User.find(1)
user_two = User.find(1)

user_one.name = "John"
user_one.save
# Run at the same instance
user_two.name = "Doe"
user_two.save # Raises a ActiveRecord::StaleObjectError
```

### Bloqueo pesimista

```
appointment = Appointment.find(5)
appointment.lock!
#no other users can read this appointment,
#they have to wait until the lock is released
appointment.save!
#lock is released, other users can read this account
```

Lea ActiveRecord Locking en línea: <https://riptutorial.com/es/ruby-on-rails/topic/3866/activerecord-locking>

---

# Capítulo 8: ActiveSupport

## Observaciones

ActiveSupport es una gema de utilidad de herramientas de uso general utilizadas por el resto del marco de Rails.

Una de las formas principales en que proporciona estas herramientas es mediante la aplicación de los tipos nativos de Ruby. Estos se conocen como **Extensiones Core**.

## Examples

### Extensiones de núcleo: String Access

---

## Cadena # en

Devuelve una subcadena de un objeto de cadena. La misma interfaz que `String#[]`.

```
str = "hello"
str.at(0)      # => "h"
str.at(1..3)   # => "ell"
str.at(-2)     # => "l"
str.at(-2..-1) # => "lo"
str.at(5)      # => nil
str.at(5..-1)  # => ""
```

---

## Cadena # de

Devuelve una subcadena desde la posición dada hasta el final de la cadena.

```
str = "hello"
str.from(0)   # => "hello"
str.from(3)   # => "lo"
str.from(-2)  # => "lo"
```

---

## Cadena # a

Devuelve una subcadena desde el principio de la cadena a la posición dada. Si la posición es negativa, se cuenta desde el final de la cadena.

```
str = "hello"
str.to(0)    # => "h"
str.to(3)    # => "hell"
```

```
str.to(-2) # => "hell"
```

from `y` to se puede utilizar en tándem.

```
str = "hello"
str.from(0).to(-1) # => "hello"
str.from(1).to(-2) # => "ell"
```

---

## Cadena # primero

Devuelve el primer carácter, o un número dado de caracteres hasta la longitud de la cadena.

```
str = "hello"
str.first # => "h"
str.first(1) # => "h"
str.first(2) # => "he"
str.first(0) # => ""
str.first(6) # => "hello"
```

---

## Cadena # última

Devuelve el último carácter, o un número dado de caracteres desde el final de la cadena que cuenta hacia atrás.

```
str = "hello"
str.last # => "o"
str.last(1) # => "o"
str.last(2) # => "lo"
str.last(0) # => ""
str.last(6) # => "hello"
```

Extensiones de núcleo: cadena a fecha / hora de conversión

---

## Cadena # to\_time

Convierte una cadena en un valor de tiempo. El parámetro de `form` puede ser `:utc` o `:local`, por defecto es `:local`.

```
"13-12-2012".to_time # => 2012-12-13 00:00:00 +0100
"06:12".to_time # => 2012-12-13 06:12:00 +0100
"2012-12-13 06:12".to_time # => 2012-12-13 06:12:00 +0100
"2012-12-13T06:12".to_time # => 2012-12-13 06:12:00 +0100
"2012-12-13T06:12".to_time(:utc) # => 2012-12-13 06:12:00 UTC
"12/13/2012".to_time # => ArgumentError: argument out of range
```

# Cadena # to\_date

Convierte una cadena en un valor de fecha.

```
"1-1-2012".to_date # => Sun, 01 Jan 2012
"01/01/2012".to_date # => Sun, 01 Jan 2012
"2012-12-13".to_date # => Thu, 13 Dec 2012
"12/13/2012".to_date # => ArgumentError: invalid date
```

---

# Cadena # to\_datetime

Convierte una cadena a un valor DateTime.

```
"1-1-2012".to_datetime # => Sun, 01 Jan 2012 00:00:00 +0000
"01/01/2012 23:59:59".to_datetime # => Sun, 01 Jan 2012 23:59:59 +0000
"2012-12-13 12:50".to_datetime # => Thu, 13 Dec 2012 12:50:00 +0000
"12/13/2012".to_datetime # => ArgumentError: invalid date
```

Extensiones de núcleo: Exclusión de cadenas

---

# Cadena # ¿excluir?

La inversa de `String#include?`

```
"hello".exclude? "lo" # => false
"hello".exclude? "ol" # => true
"hello".exclude? ?h # => false
```

Extensiones de núcleo: Filtros de cadena

---

# Cuerda # squish

Devuelve una versión de la cadena dada sin espacios en blanco iniciales o finales, y combina todos los espacios en blanco consecutivos en el interior en espacios individuales. Squish versión destructiva `squish!` opera directamente en la instancia de cadena.

Maneja tanto los espacios en blanco ASCII como Unicode.

```
%{ Multi-line
  string }.squish # => "Multi-line string"
"foo bar \n \t boo".squish # => "foo bar boo"
```

## Cadena # eliminar

Devuelve una nueva cadena con todas las apariciones de los patrones eliminados. Versión destructiva `remove!` opera directamente en la cadena dada.

```
str = "foo bar test"
str.remove(" test")           # => "foo bar"
str.remove(" test", /bar/)   # => "foo "
```

---

## Cadena # truncar

Devuelve una copia de una cadena dada truncada en una longitud dada si la cadena es más larga que la longitud.

```
'Once upon a time in a world far far away'.truncate(27)
# => "Once upon a time in a wo..."
```

Pase una cadena o expresión regular `:separator` para truncar en un corte natural

```
'Once upon a time in a world far far away'.truncate(27, separator: ' ')
# => "Once upon a time in a..."

'Once upon a time in a world far far away'.truncate(27, separator: /\s/)
# => "Once upon a time in a..."
```

---

## Cadena # truncate\_words

Devuelve una cadena truncada después de un número dado de palabras.

```
'Once upon a time in a world far far away'.truncate_words(4)
# => "Once upon a time..."
```

Pase una cadena o expresión regular para especificar un separador de palabras diferente

```
'Once<br>upon<br>a<br>time<br>in<br>a<br>world'.truncate_words(5, separator: '<br>')
# => "Once<br>upon<br>a<br>time<br>in..."
```

Los últimos caracteres se reemplazarán con la cadena de `:omission` (por defecto es "...")

```
'And they found that many people were sleeping better.'.truncate_words(5, omission: '...
(continued)')
# => "And they found that many... (continued)"
```

---

## Cadena # strip\_heredoc

Tiras de sangrado en heredocs. Busca la línea no vacía menos sangrada y elimina esa cantidad de espacios en blanco iniciales.

```
if options[:usage]
  puts <<-USAGE.strip_heredoc
  This command does such and such.

  Supported options are:
    -h          This message
    ...
  USAGE
end
```

el usuario vería

```
This command does such and such.

Supported options are:
-h          This message
...
```

Extensiones de núcleo: Inflexión de cuerdas

## Cadena # pluralizar

Devoluciones de forma plural de la cadena. Opcionalmente toma un parámetro de `count` y devuelve una forma singular si `count == 1`. También acepta un parámetro de `locale` para la pluralización específica del idioma.

```
'post'.pluralize           # => "posts"
'octopus'.pluralize        # => "octopi"
'sheep'.pluralize          # => "sheep"
'words'.pluralize          # => "words"
'the blue mailman'.pluralize # => "the blue mailmen"
'CamelOctopus'.pluralize   # => "CamelOctopi"
'apple'.pluralize(1)       # => "apple"
'apple'.pluralize(2)       # => "apples"
'ley'.pluralize(:es)       # => "leyes"
'ley'.pluralize(1, :es)    # => "ley"
```

## String # singularize

Devuelve la forma singular de la cadena. Acepta un parámetro de `locale` opcional.

```
'posts'.singularize        # => "post"
'octopi'.singularize       # => "octopus"
'sheep'.singularize        # => "sheep"
'word'.singularize         # => "word"
'the blue mailmen'.singularize # => "the blue mailman"
'CamelOctopi'.singularize  # => "CamelOctopus"
```

```
'leyes'.singularize(:es) # => "ley"
```

---

## String # constantize

Intenta encontrar una constante declarada con el nombre especificado en la cadena. `NameError` un `NameError` nombre cuando el nombre no está en CamelCase o no está inicializado.

```
'Module'.constantize # => Module
'Class'.constantize # => Class
'blargle'.constantize # => NameError: wrong constant name blargle
```

---

## Cadena # safe\_constantize

Realiza una `constantize` pero devuelve `nil` lugar de elevar `NameError`.

```
'Module'.safe_constantize # => Module
'Class'.safe_constantize # => Class
'blargle'.safe_constantize # => nil
```

---

## Cuerda # camelize

Convierte cadenas a UpperCamelCase de manera predeterminada, si `:lower` se da como param se convierte a lowerCamelCase en su lugar.

alias: `camelcase`

**Nota:** también convertirá / a `::` que es útil para convertir rutas a espacios de nombres.

```
'active_record'.camelize # => "ActiveRecord"
'active_record'.camelize(:lower) # => "activeRecord"
'active_record/errors'.camelize # => "ActiveRecord::Errors"
'active_record/errors'.camelize(:lower) # => "activeRecord::Errors"
```

---

## Cadena # titleize

Pone en mayúscula todas las palabras y reemplaza algunos caracteres en la cadena para crear un título de mejor apariencia.

alias: `titlecase`

```
'man from the boondocks'.titleize # => "Man From The Boondocks"
'x-men: the last stand'.titleize # => "X Men: The Last Stand"
```

## Cadena # subrayado

Hace una forma subrayada, minúscula de la expresión en la cadena. El reverso de `camelize`.

**Nota:** el `underscore` también cambiará `::` a `/` para convertir espacios de nombres en rutas.

```
'ActiveModel'.underscore # => "active_model"
'ActiveModel::Errors'.underscore # => "active_model/errors"
```

## Cadena # dasherizar

Reemplaza los guiones bajos con guiones en la cadena.

```
'puni_puni'.dasherize # => "puni-puni"
```

## Cadena # demodulizar

Elimina la parte del módulo de la expresión constante en la cadena.

```
'ActiveRecord::CoreExtensions::String::Inflections'.demodulize # => "Inflections"
'Inflections'.demodulize # => "Inflections"
'::Inflections'.demodulize # => "Inflections"
''.demodulize # => ''
```

## Cadena # desconstantizar

Elimina el segmento más a la derecha de la expresión constante en la cadena.

```
'Net::HTTP'.deconstantize # => "Net"
'::Net::HTTP'.deconstantize # => "::Net"
'String'.deconstantize # => ""
'::String'.deconstantize # => ""
''.deconstantize # => ""
```

## Cadena # parametrizar

Reemplaza los caracteres especiales en una cadena para que pueda usarse como parte de una URL "bonita".

```
"Donald E. Knuth".parameterize # => "donald-e-knuth"
```

Conserve el caso de los caracteres en una cadena con el argumento `:preserve_case`.

```
"Donald E. Knuth".parameterize(preserve_case: true) # => "Donald-E-Knuth"
```

Un caso de uso muy común para la `parameterize` es anular el método `to_param` de un modelo ActiveRecord para admitir slugs de URL más descriptivos.

```
class Person < ActiveRecord::Base
  def to_param
    "#{id}-#{name.parameterize}"
  end
end

Person.find(1).to_param # => "1-donald-e-knuth"
```

---

## String # tableize

Crea el nombre de una tabla como lo hace Rails para modelos a nombres de tablas. Pluraliza la última palabra en la cadena.

```
'RawScaledScorer'.tableize # => "raw_scaled_scorers"
'ham_and_egg'.tableize    # => "ham_and_eggs"
'fancyCategory'.tableize  # => "fancy_categories"
```

---

## Cadena # clasifica

Devuelve una cadena de nombre de clase de un nombre de tabla plural como lo hace Rails para los nombres de tabla a los modelos.

```
'ham_and_eggs'.classify # => "HamAndEgg"
'posts'.classify        # => "Post"
```

---

## Cadena # humanizar

`_id` mayúscula la primera palabra, convierte los guiones bajos en espacios y `_id` un `_id` final si está presente.

```
'employee_salary'.humanize # => "Employee salary"
'author_id'.humanize       # => "Author"
'author_id'.humanize(capitalize: false) # => "author"
'_id'.humanize              # => "Id"
```

---

## Cadena # upcase\_first

Convierte solo el primer carácter a mayúsculas.

```
'what a Lovely Day'.upcase_first # => "What a Lovely Day"
'w'.upcase_first                 # => "W"
''.upcase_first                  # => ""
```

---

## Cadena # foreign\_key

Crea un nombre de clave externa a partir de un nombre de clase. Pase `false` param para deshabilitar la adición de `_` entre nombre e `id`.

```
'Message'.foreign_key           # => "message_id"
'Message'.foreign_key(false)   # => "messageid"
'Admin::Post'.foreign_key      # => "post_id"
```

Lea ActiveSupport en línea: <https://riptutorial.com/es/ruby-on-rails/topic/4490/activesupport>

# Capítulo 9: Actualización de rieles

## Examples

### Actualización de Rails 4.2 a Rails 5.0

*Nota: Antes de actualizar su aplicación Rails, asegúrese siempre de guardar su código en un sistema de control de versiones, como Git.*

Para actualizar de Rails 4.2 a Rails 5.0, debe usar Ruby 2.2.2 o más reciente. Después de actualizar su versión de Ruby si es necesario, vaya a su Gemfile y cambie la línea:

```
gem 'rails', '4.2.X'
```

a:

```
gem 'rails', '~> 5.0.0'
```

y en la línea de comandos ejecute:

```
$ bundle update
```

Ahora ejecuta la tarea de actualización usando el comando:

```
$ rake rails:update
```

Esto le ayudará a actualizar los archivos de configuración. Se le pedirá que sobrescriba los archivos y tiene varias opciones para ingresar:

- Y - si, sobrescribir
- n - no, no sobrescribir
- a - todos, sobrescriben esto y todos los demás
- q - renunciar, abortar
- d - dif, muestra las diferencias entre lo antiguo y lo nuevo.
- h - ayuda

Por lo general, debe verificar las diferencias entre los archivos antiguos y nuevos para asegurarse de que no esté recibiendo cambios no deseados.

Los modelos `ActiveRecord 5.0 ActiveRecord` heredan de `ApplicationRecord`, en lugar de `ActiveRecord::Base`. `ApplicationRecord` es la superclase de todos los modelos, similar a cómo `ApplicationController` es la superclase de los controladores. Para tener en cuenta esta nueva forma en que se manejan los modelos, debe crear un archivo en su `app/models/` carpeta llamada `application_record.rb` y luego editar el contenido de ese archivo para que sea:

```
class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

Rails 5.0 también maneja callbacks ligeramente diferentes. Las devoluciones de llamada que devuelven `false` no detendrán la cadena de devolución de llamada, lo que significa que las devoluciones de llamada subsiguientes se ejecutarán, a diferencia de Rails 4.2. Cuando actualices, el comportamiento de Rails 4.2 se mantendrá, aunque puedes cambiar al comportamiento de Rails 5.0 agregando:

```
ActiveSupport.halt_callback_chains_on_return_false = false
```

al archivo `config/application.rb` . Puede detener explícitamente la cadena de devolución de llamada llamando a `throw(:abort)` .

En Rails 5.0, `ApplicationJob` heredará de `ApplicationJob` , en lugar de `ActiveJob::Base` como en Rails 4.2. Para actualizar a Rails 5.0, cree un archivo llamado `application_job.rb` en la carpeta `app/jobs/` . Edite los contenidos de ese archivo para que sean:

```
class ApplicationJob < ActiveJob::Base
end
```

Luego, debe cambiar todos sus trabajos para heredar de `ApplicationJob` lugar de `ActiveJob::Base` .

Uno de los otros cambios más importantes de Rails 5.0 no requiere ningún cambio de código, pero cambiará la forma en que usa la línea de comandos con sus aplicaciones Rails. Podrá usar `bin/rails` , o solo `rails` , para ejecutar tareas y pruebas. Por ejemplo, en lugar de usar `$ rake db:migrate` , ahora puede hacer `$ rails db:migrate` . Si ejecuta `$ bin/rails` , puede ver todos los comandos disponibles. Tenga en cuenta que muchas de las tareas que ahora se pueden ejecutar con `bin/rails` aún funcionan con `rake` .

Lea Actualización de rieles en línea: <https://riptutorial.com/es/ruby-on-rails/topic/3496/actualizacion-de-rieles>

---

# Capítulo 10: Agregar panel de administración

## Introducción

Si desea agregar un panel de administración a su aplicación de rieles, solo es cuestión de minutos.

## Sintaxis

1. Abra el archivo gema y el escritor gema 'rails\_admin', '~> 1.0'
2. paquete de instalación
3. rieles g rails\_admin: instalar
4. le preguntará acerca de la ruta del administrador si desea ir con la opción predeterminada, presione Entrar.
5. Ahora ve a app / config / initializers / rails\_admin.rb y pega este código:  
config.authorize\_with do redirect\_to main\_app.root\_path a menos que current\_user.try (:admin?) Final Los usuarios serán redirigidos a la ruta de raíz.
6. Para más detalles verifica la documentación de esta gema.  
[https://github.com/sferik/rails\\_admin/wiki](https://github.com/sferik/rails_admin/wiki)

## Observaciones

Úselo si desea tener Admin en su sitio web, de lo contrario no será necesario. Es más fácil y potente que la gema active\_admin. Puede agregar esto en cualquier etapa después de crear usuarios y no se olvide de hacer que cualquier usuario sea administrador antes del cuarto paso. Usa cancan para otorgar roles.

## Examples

Así que aquí hay algunas capturas de pantalla desde el panel de administración usando rails\_admin gem.

Como se puede ver, el diseño de esta gema es muy atractivo y fácil de usar.

NAVIGATION

[Blogs](#)

[Users](#)

# Site Administration

Dashboard

 Dashboard

Model name	Last created	Records
<a href="#">Blogs</a>	about 7 hours ago	
<a href="#">Users</a>	about 23 hours ago	

NAVIGATION

Blogs

Users

# List of Users

Dashboard / Users

List

+ Add new

Export

Filter

Refresh



<input type="checkbox"/>	Id	Email	Reset password sent at	Remember c
<input type="checkbox"/>	2	2@gmail.com	-	-
<input type="checkbox"/>	1	1@gmail.com	-	-

2 users

## NAVIGATION

Blogs

Users

# List of Blogs

[Dashboard](#) / [Blogs](#)

☰ List

+ Add new

📄 Export

Filter

↻ Refresh

✕

<input type="checkbox"/>	Id	Title	Content	Created at
<input type="checkbox"/>	7	Post 3	Test content	December 07, 2016 08:19
<input type="checkbox"/>	6	Post 2	test content	December 06, 2016 16:16
<input type="checkbox"/>	5	Post 1	test content	December 06, 2016 16:16

3 blogs

Lea Agregar panel de administración en línea: <https://riptutorial.com/es/ruby-on-rails/topic/8128/agregar-panel-de-administracion>

---

# Capítulo 11: Agregar un Amazon RDS a su aplicación de rieles

## Introducción

Pasos para crear una instancia de AWS RDS y configurar su archivo `database.yml` instalando los conectores necesarios.

## Examples

Considera que estamos conectando **MYSQL RDS** con tu aplicación de rieles.

### Pasos para crear la base de datos **MYSQL**

1. Inicie sesión en la cuenta de Amazon y seleccione el servicio RDS
2. Seleccione `Launch DB Instance` de base de datos en la pestaña de instancia
3. De forma predeterminada, se seleccionará **MYSQL Community Edition**, por lo tanto, haga clic en el botón `select`
4. Seleccione el propósito de la base de datos, diga `production` y haga clic en el `next step`
5. Proporcione la `mysql version`, `storage size`, `DB Instance Identifier`, `Master Username and Password` y haga clic en el `next step`
6. Introduzca el `Database Name` y haga clic en `Launch DB Instance`
7. Por favor espere hasta que toda la instancia sea creada. Una vez que se crea la instancia, encontrará un punto final, copie este punto de entrada (que se conoce como nombre de host)

### Instalacion de conectores

Agregue el adaptador de base de datos MySQL al archivo gem de su proyecto,

```
gem 'mysql2'
```

Instala tus gemas añadidas,

```
bundle install
```

Algunos otros adaptadores de base de datos son,

- `gem 'pg'` para PostgreSQL
- `gem 'activerecord-oracle_enhanced-adapter'` para Oracle
- `gem 'sql_server'` para SQL Server

**Configure el archivo `database.yml` de su proyecto** Abra su **archivo** `config / database.yml`

```
production:
  adapter: mysql2
  encoding: utf8
  database: <%= RDS_DB_NAME %> # Which you have entered you creating database
  username: <%= RDS_USERNAME %> # db master username
  password: <%= RDS_PASSWORD %> # db master password
  host: <%= RDS_HOSTNAME %> # db instance endpoint
  port: <%= RDS_PORT %> # db post. For MYSQL 3306
```

Lea Agregar un Amazon RDS a su aplicación de rieles en línea: <https://riptutorial.com/es/ruby-on-rails/topic/10922/agregar-un-amazon-rds-a-su-aplicacion-de-rieles>

# Capítulo 12: Almacenamiento en caché

## Examples

### Muñeca rusa caching

Es posible que desee anidar fragmentos en caché dentro de otros fragmentos en caché. Esto se llama `Russian doll caching`.

La ventaja del `Russian doll caching` de `Russian doll caching` es que si se actualiza un solo producto, todos los demás fragmentos internos se pueden reutilizar cuando se regenera el fragmento exterior.

Como se explicó en la sección anterior, un archivo almacenado en caché caducará si el valor de `updated_at` cambia para un registro del cual depende directamente el archivo almacenado en caché. Sin embargo, esto no caducará ningún caché en el que esté anidado el fragmento.

Por ejemplo, tome la siguiente vista:

```
<% cache product do %>
  <%= render product.games %>
<% end %>
```

Que a su vez hace que esta vista:

```
<% cache game do %>
  <%= render game %>
<% end %>
```

Si se cambia algún atributo del juego, el valor `updated_at` se establecerá en la hora actual, con lo que caducará el caché.

Sin embargo, dado que `updated_at` no se modificará para el objeto del producto, esa memoria caché no caducará y su aplicación servirá datos obsoletos. Para solucionar esto, unimos los modelos con el método táctil:

```
class Product < ApplicationRecord
  has_many :games
end

class Game < ApplicationRecord
  belongs_to :product, touch: true
end
```

### SQL Caching

El almacenamiento en caché de consultas es una característica de `Rails` que almacena en caché el conjunto de resultados devuelto por cada consulta. Si `Rails` vuelve a encontrar la misma

consulta para esa solicitud, usará el conjunto de resultados en caché en lugar de ejecutar la consulta nuevamente en la base de datos.

Por ejemplo:

```
class ProductsController < ApplicationController

  def index
    # Run a find query
    @products = Product.all

    ...

    # Run the same query again
    @products = Product.all
  end

end
```

La segunda vez que se ejecuta la misma consulta en la base de datos, en realidad no llegará a la base de datos. La primera vez que se devuelve el resultado de la consulta, se almacena en el caché de consulta (en la memoria) y la segunda vez se extrae de la memoria.

Sin embargo, es importante tener en cuenta que los cachés de consulta se crean al inicio de una acción y se destruyen al final de esa acción y, por lo tanto, persisten solo durante la duración de la acción. Si desea almacenar los resultados de la consulta de una manera más persistente, puede hacerlo con el almacenamiento en caché de bajo nivel.

## Almacenamiento en caché de fragmentos

`Rails.cache`, proporcionado por ActiveSupport, puede usarse para almacenar en caché cualquier objeto Ruby serializable en todas las solicitudes.

Para obtener un valor de la memoria caché para una clave dada, use `cache.read`:

```
Rails.cache.read('city')
# => nil
```

Use `cache.write` para escribir un valor en el caché:

```
Rails.cache.write('city', 'Duckburgh')
Rails.cache.read('city')
# => 'Duckburgh'
```

Alternativamente, use `cache.fetch` para leer un valor del caché y, opcionalmente, escriba un valor predeterminado si no hay ningún valor:

```
Rails.cache.fetch('user') do
  User.where(:is_awesome => true)
end
```

El valor de retorno del bloque pasado se asignará a la memoria caché bajo la clave dada, y luego se devolverá.

También puede especificar un vencimiento de caché:

```
Rails.cache.fetch('user', :expires_in => 30.minutes) do
  User.where(:is_awesome => true)
end
```

## Almacenamiento en caché de páginas

Puede usar la [gema page\\_caching de ActionPack](#) para almacenar en caché páginas individuales. Esto almacena el resultado de una solicitud dinámica como un archivo HTML estático, que se sirve en lugar de la solicitud dinámica en solicitudes posteriores. El archivo README contiene instrucciones de configuración completas. Una vez configurado, use el método de clase `caches_page` en un controlador para almacenar en caché el resultado de una acción:

```
class UsersController < ActionController::Base
  caches_page :index
end
```

Utilice `expire_page` para forzar la caducidad de la memoria caché eliminando el archivo HTML almacenado:

```
class UsersController < ActionController::Base
  caches_page :index

  def index
    @users = User.all
  end

  def create
    expire_page :action => :index
  end
end
```

La sintaxis de `expire_page` imita la de `url_for` y `friends`.

## Almacenamiento en caché HTTP

Rails> = 3 viene con capacidades de almacenamiento en caché HTTP fuera de la caja. Esto utiliza los encabezados `Cache-Control` y `Etag` para controlar cuánto tiempo un cliente o intermediario (como un CDN) puede almacenar en caché una página.

En una acción de controlador, use `expires_in` para establecer la duración del almacenamiento en caché para esa acción:

```
def show
  @user = User.find params[:id]
  expires_in 30.minutes, :public => true
end
```

Utilice `expires_now` para forzar la expiración inmediata de un recurso en caché en cualquier cliente o intermediario visitante:

```
def show
  @users = User.find params[:id]
  expires_now if params[:id] == 1
end
```

## Almacenamiento en caché de acciones

Al igual que el almacenamiento en caché de páginas, el almacenamiento en caché de acciones almacena en caché toda la página. La diferencia es que la solicitud llega a la pila de Rails, por lo que antes de que se ejecuten los filtros antes de que se sirva el caché. Se extrae de Rails a la [gema actionpack-action\\_caching](#) .

Un ejemplo común es el almacenamiento en caché de una acción que requiere autenticación:

```
class SecretIngredientsController < ApplicationController
  before_action :authenticate_user!, only: :index, :show
  caches_action :index

  def index
    @secret_ingredients = Recipe.find(params[:recipe_id]).secret_ingredients
  end
end
```

Las opciones incluyen `:expires_in` , un custom `:cache_path` (para acciones con múltiples rutas que deben almacenarse en caché de manera diferente) y `:if` / `:unless` que se controle cuándo se debe almacenar en caché la acción.

```
class RecipesController < ApplicationController
  before_action :authenticate_user!, except: :show
  caches_page :show
  caches_action :archive, expires_in: 1.day
  caches_action :index, unless: { request.format.json? }
end
```

Cuando el diseño tiene contenido dinámico, almacene en caché solo el contenido de la acción pasando el `layout: false` .

Lea [Almacenamiento en caché en línea](https://riptutorial.com/es/ruby-on-rails/topic/2833/almacenamiento-en-cache): <https://riptutorial.com/es/ruby-on-rails/topic/2833/almacenamiento-en-cache>

---

# Capítulo 13: Almacenamiento seguro de claves de autenticación

## Introducción

Muchas API de terceros requieren una clave, lo que les permite evitar el abuso. Si te emiten una clave, es muy importante que no la ingreses en un repositorio público, ya que esto permitirá que otros roben tu clave.

## Examples

### Almacenando claves de autenticación con Figaro

Agregue `gem 'figaro'` a su Gemfile y ejecute `bundle install`. Luego ejecute `bundle exec figaro install`; esto creará `config / application.yml` y lo agregará a su archivo `.gitignore`, evitando que se agregue al control de versiones.

Puede almacenar sus claves en `application.yml` en este formato:

```
SECRET_NAME: secret_value
```

donde `SECRET_NAME` y `secret_value` son el nombre y el valor de su clave API.

También debe nombrar estos secretos en `config / secrets.yml`. Puedes tener diferentes secretos en cada entorno. El archivo debería verse así:

```
development:
  secret_name: <%= ENV["SECRET_NAME"] %>
test:
  secret_name: <%= ENV["SECRET_NAME"] %>
production:
  secret_name: <%= ENV["SECRET_NAME"] %>
```

La forma en que utiliza estas claves varía, pero digamos, por ejemplo, que algunos `some_component` en el entorno de desarrollo necesitan acceso a `secret_name`. En `config / environment / development.rb`, pondrías:

```
Rails.application.configure do
  config.some_component.configuration_hash = {
    :secret => Rails.application.secrets.secret_name
  }
end
```

Por último, supongamos que desea crear un entorno de producción en Heroku. Este comando cargará los valores en `config / environment / production.rb` a Heroku:

```
$ figaro heroku:set -e production
```

Lea Almacenamiento seguro de claves de autenticación en línea: <https://riptutorial.com/es/ruby-on-rails/topic/9711/almacenamiento-seguro-de-claves-de-autenticacion>

---

# Capítulo 14: API de Rails

## Examples

### Creando una aplicación solo para API

Para crear una aplicación Rails que será un servidor API, puede comenzar con un subconjunto más limitado de Rails en Rails 5.

Para generar una nueva aplicación API Rails:

```
rails new my_api --api
```

Lo que hace `--api` es eliminar la funcionalidad que no es necesaria cuando se `--api` una API. Esto incluye sesiones, cookies, activos y cualquier cosa que haga que Rails funcione en un navegador.

También configurará los generadores para que no generen vistas, ayudantes y activos al generar un nuevo recurso.

Cuando comparas `ApplicationController` en una aplicación web con una aplicación API, verás que la versión web se extiende desde `ActionController::Base`, mientras que la versión API se extiende desde `ActionController::API`, que incluye un subconjunto mucho más pequeño de funcionalidad.

Lea API de Rails en línea: <https://riptutorial.com/es/ruby-on-rails/topic/4305/api-de-rails>

---

# Capítulo 15: Aplicaciones de carriles de prueba

## Examples

### Prueba de unidad

Pruebas unitarias prueba partes de la aplicación en aislamiento. Usualmente una unidad bajo prueba es una clase o módulo.

```
let(:gift) { create :gift }

describe '#find' do
  subject { described_class.find(user, Time.zone.now.to_date) }
  it { is_expected.to eq gift }
end
```

#### fuentes

Este tipo si la prueba es tan directa y específica como sea posible.

### Solicitud de prueba

Las pruebas de solicitud son pruebas de extremo a extremo que imitan el comportamiento del usuario.

```
it 'allows the user to set their preferences' do
  check 'Ruby'
  click_on 'Save and Continue'
  expect(user.languages).to eq ['Ruby']
end
```

#### fuentes

Este tipo de prueba se enfoca en los flujos de usuarios y se ejecuta a través de todas las capas del sistema, a veces incluso procesando javascript.

Lea Aplicaciones de carriles de prueba en línea: <https://riptutorial.com/es/ruby-on-rails/topic/7853/aplicaciones-de-carriles-de-prueba>

# Capítulo 16: Asociaciones ActiveRecord

## Examples

### pertenece a

A `belongs_to` asociación establece una conexión de uno a uno con otro modelo, por lo que cada instancia del modelo declarando "pertenece a" una instancia del otro modelo.

Por ejemplo, si su aplicación incluye usuarios y publicaciones, y cada publicación puede asignarse a exactamente un usuario, declararía el modelo de publicación de esta manera:

```
class Post < ApplicationRecord
  belongs_to :user
end
```

En la estructura de tu tabla, puedes tener

```
create_table "posts", force: :cascade do |t|
  t.integer "user_id", limit: 4
end
```

### Tiene uno

Una asociación `has_one` establece una conexión uno a uno con otro modelo, pero con una semántica diferente. Esta asociación indica que cada instancia de un modelo contiene o posee una instancia de otro modelo.

Por ejemplo, si cada usuario en su aplicación tiene solo una cuenta, declararía el modelo de usuario así:

```
class User < ApplicationRecord
  has_one :account
end
```

En Registro activo, cuando tiene una relación `has_one`, el registro activo garantiza que solo exista un registro con la clave externa.

Aquí en nuestro ejemplo: En la tabla de cuentas, solo puede haber un registro con un `user_id` particular. Si intenta asociar una cuenta más para el mismo usuario, hace que la clave externa de la entrada anterior sea nula (quedando huérfana) y crea una nueva automáticamente. Hace que la entrada anterior sea nula, incluso si el guardado falla para que la nueva entrada mantenga la coherencia.

```
user = User.first
user.build_account(name: "sample")
user.save #Saves it successfully, and creates an entry in accounts table with user_id 1
```

```
user.build_account(name: "sample1") [automatically makes the previous entry's foreign key null]
user.save [creates the new account with name sample 1 and user_id 1]
```

## tiene muchos

Una asociación `has_many` indica una conexión de uno a varios con otro modelo. Esta asociación generalmente se encuentra en el otro lado de una asociación `belongs_to`.

Esta asociación indica que cada instancia del modelo tiene cero o más instancias de otro modelo.

Por ejemplo, en una aplicación que contiene usuarios y publicaciones, el modelo de usuario podría declararse así:

```
class User < ApplicationRecord
  has_many :posts
end
```

La estructura de la tabla de `Post` seguiría siendo el mismo que en el `belongs_to` ejemplo; en contraste, el `User` no requeriría ningún cambio de esquema.

Si desea obtener la lista de todas las publicaciones publicadas para el `User`, puede agregar lo siguiente (es decir, puede agregar ámbitos a sus objetos de asociación):

```
class User < ApplicationRecord
  has_many :published_posts, -> { where("posts.published IS TRUE") }, class_name: "Post"
end
```

## Asociación polimórfica

Este tipo de asociación permite que un modelo ActiveRecord pertenezca a más de un tipo de registro modelo. Ejemplo común:

```
class Human < ActiveRecord::Base
  has_one :address, :as => :addressable
end

class Company < ActiveRecord::Base
  has_one :address, :as => :addressable
end

class Address < ActiveRecord::Base
  belongs_to :addressable, :polymorphic => true
end
```

Sin esta asociación, tendría todas estas claves externas en su tabla de direcciones, pero solo tendría un valor para una de ellas porque una dirección, en este escenario, solo puede pertenecer a una entidad (humana o empresa). Aquí es como se vería:

```
class Address < ActiveRecord::Base
  belongs_to :human
```

```
  belongs_to :company
end
```

## El has\_many: a través de la asociación

A `has_many :through` asociación se utiliza a menudo para configurar una conexión de `many-to-many` con otro modelo. Esta asociación indica que el modelo declarante puede compararse con cero o más instancias de otro modelo al proceder a través de un tercer modelo.

Por ejemplo, considere una práctica médica donde los pacientes hacen citas para ver a los médicos. Las declaraciones de asociación relevantes podrían verse así:

```
class Physician < ApplicationRecord
  has_many :appointments
  has_many :patients, through: :appointments
end

class Appointment < ApplicationRecord
  belongs_to :physician
  belongs_to :patient
end

class Patient < ApplicationRecord
  has_many :appointments
  has_many :physicians, through: :appointments
end
```

## El has\_one: a través de la asociación.

A `has_one :through` asociación configura `one-to-one` conexión `one-to-one` con otro modelo. Esta asociación indica que el modelo declarante se puede hacer coincidir con una instancia de otro modelo al proceder a través de un tercer modelo.

Por ejemplo, si cada `supplier` tiene una `account` y cada cuenta está asociada con un historial de cuenta, entonces el modelo de proveedor podría tener este aspecto:

```
class Supplier < ApplicationRecord
  has_one :account
  has_one :account_history, through: :account
end

class Account < ApplicationRecord
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ApplicationRecord
  belongs_to :account
end
```

## La asociación has\_and\_belongs\_to\_many

Una asociación `has_and_belongs_to_many` crea una conexión directa de `many-to-many` con otro

modelo, sin un modelo intermedio.

Por ejemplo, si su aplicación incluye `assemblies` y `parts`, cada ensamblaje tiene muchas piezas y cada pieza aparece en muchos ensamblajes, podría declarar los modelos de esta manera:

```
class Assembly < ApplicationRecord
  has_and_belongs_to_many :parts
end

class Part < ApplicationRecord
  has_and_belongs_to_many :assemblies
end
```

## Asociación auto-referencial

La asociación autorreferencial se utiliza para asociar un modelo consigo mismo. El ejemplo más frecuente sería administrar la asociación entre un amigo y su seguidor.

ex.

```
rails g model friendship user_id:references friend_id:integer
```

ahora puedes asociar modelos como;

```
class User < ActiveRecord::Base
  has_many :friendships
  has_many :friends, :through => :friendships
  has_many :inverse_friendships, :class_name => "Friendship", :foreign_key => "friend_id"
  has_many :inverse_friends, :through => :inverse_friendships, :source => :user
end
```

y el otro modelo se verá como

```
class Friendship < ActiveRecord::Base
  belongs_to :user
  belongs_to :friend, :class_name => "User"
end
```

Lea Asociaciones ActiveRecord en línea: <https://riptutorial.com/es/ruby-on-rails/topic/1820/asociaciones-activerecord>

---

# Capítulo 17: Autenticación API Rails 5

## Examples

### Autenticación con Rails `authenticate_with_http_token`

```
authenticate_with_http_token do |token, options|  
  @user = User.find_by(auth_token: token)  
end
```

Puede probar este punto final con `curl` haciendo una solicitud como

```
curl -IH "Authorization: Token token=my-token" http://localhost:3000
```

Lea Autenticación API Rails 5 en línea: <https://riptutorial.com/es/ruby-on-rails/topic/7852/autenticacion-api-rails-5>

---

# Capítulo 18: Autenticar Api utilizando Devise

## Introducción

Devise es una solución de autenticación para Rails. Antes de seguir adelante me gustaría agregar una nota rápida sobre la API. Por lo tanto, la API no maneja sesiones (es sin estado), lo que significa que proporciona una respuesta después de su solicitud, y luego no requiere más atención, lo que significa que no se requiere un estado anterior o futuro para que el sistema funcione, por lo tanto, cada vez que solicitemos al servidor necesitamos Pase los detalles de autenticación con todas las API y debe decirle a Devise que no almacene los detalles de autenticación.

## Examples

### Empezando

Así que primero crearemos rieles de proyecto y dispositivo de configuración.

crear una aplicación de rieles

```
rails new devise_example
```

ahora agregue el dispositivo a la lista de gemas

Puede encontrar un archivo llamado 'Gemfile' en la raíz del proyecto Rails

A continuación, ejecute `bundle install`

A continuación, necesita ejecutar el generador:

```
rails generate devise:install
```

Ahora en la consola puedes encontrar algunas instrucciones solo síguelo.

Generar modelo de dispositivo.

```
rails generate devise MODEL
```

A continuación, ejecute `rake db:migrate`

Para más detalles ir a: [Devise Gem](#)

---

## Token de Autenticación

El token de autenticación se usa para autenticar a un usuario con un token único. Por lo tanto, antes de continuar con la lógica, primero debemos agregar el campo `auth_token` a un modelo de Devise

Por lo tanto,

```
rails g migration add_authentication_token_to_users

class AddAuthenticationTokenToUsers < ActiveRecord::Migration
  def change
    add_column :users, :auth_token, :string, default: ""
    add_index :users, :auth_token, unique: true
  end
end
```

A continuación, ejecute `rake db:migrate`

Ahora estamos listos para realizar la autenticación usando `auth_token`

En `app/controllers/application_controllers.rb`

Primero esta linea a ella

```
respond_to :html, :json
```

Esto ayudará a que la aplicación Rails responda con html y json.

Entonces

```
protect_from_forgery with: :null
```

cambiará esto `:null` ya que no estamos tratando con sesiones.

Ahora agregaremos el método de autenticación en `application_controller`.

Por lo tanto, de forma predeterminada, Devise usa el correo electrónico como campo único, también podemos usar campos personalizados, en este caso nos autenticaremos usando `user_email` y `auth_token`.

```
before_filter do
  user_email = params[:user_email].presence
  user       = user_email && User.find_by_email(user_email)

  if user && Devise.secure_compare(user.authentication_token, params[:auth_token])
    sign_in user, store: false
  end
end
```

Nota: el código anterior se basa únicamente en su lógica, solo trato de explicar el ejemplo de trabajo

En la línea 6 del código anterior puede ver que he configurado `store: false` lo que evitará que se

crea una sesión en cada solicitud, por lo tanto, logramos un estado sin estado.

Lea Autenticar Api utilizando Devise en línea: <https://riptutorial.com/es/ruby-on-rails/topic/9787/autenticar-api-utilizando-devise>

---

# Capítulo 19: Autenticación de usuario en rieles

## Introducción

Devise es una gema muy poderosa, te permite registrarte, iniciar sesión y cerrar sesión justo después de la instalación. Además el usuario puede agregar autenticaciones y restricciones a sus aplicaciones. Devise también viene con sus propios puntos de vista, si el usuario desea utilizar. Un usuario también puede personalizar los formularios de registro e inicio de sesión según sus necesidades y requisitos. Debe tenerse en cuenta que Devise recomienda que implementes tu propio inicio de sesión si eres nuevo en los rieles.

## Observaciones

En el momento de generar configuraciones de dispositivos, los `rails generate devise:install`, el dispositivo mostrará una lista de instrucciones en la terminal a seguir.

Si ya tiene un modelo de `USER`, al ejecutar este comando, los `rails generate devise USER` agregará las columnas necesarias a su modelo de `USER` existente.

Utilice este método auxiliar `before_action :authenticate_user!` en la parte superior de su controlador para verificar si el `user` ha iniciado sesión o no. Si no, entonces serán redirigidos a la página de inicio de sesión.

## Examples

### Autenticación utilizando Devise

Añadir gema al Gemfile:

```
gem 'devise'
```

A continuación, ejecute el comando de `bundle install`.

Use el comando `$ rails generate devise:install` para generar el archivo de configuración requerido.

Configure las opciones de URL predeterminadas para la aplicación de correo de Devise en cada entorno En el entorno de desarrollo agregue esta línea:

```
config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
```

a su `config/environments/development.rb`

de manera similar en producción, este archivo de `config/environments/production.rb` edición y

agrega

```
config.action_mailer.default_url_options = { host: 'your-site-url' }
```

Luego cree un modelo usando: `$ rails generate devise USER` Donde `USER` es el nombre de la clase para la que desea implementar la autenticación.

Finalmente, ejecute: `rake db:migrate` y ya está todo listo.

## Vistas personalizadas

Si necesita configurar sus vistas, puede usar los `rails generate devise:views` que copiarán todas las vistas a su aplicación. A continuación, puede editarlos como desee.

Si tiene más de un modelo de Devise en su aplicación (por ejemplo, Usuario y Administrador), notará que Devise utiliza las mismas vistas para todos los modelos. Devise ofrece una manera fácil de personalizar las vistas. Establezca `config.scoped_views = true` dentro del archivo `config/initializers/devise.rb`.

También puede usar el generador para crear vistas con ámbito: los `rails generate devise:views users`

Si desea generar solo unos pocos conjuntos de vistas, como las del módulo registrable y confirmable, use el indicador `-v`: los `rails generate devise:views -v registrations confirmations`

## Diseñar filtros de control y ayudantes

Para configurar un controlador con autenticación de usuario usando un dispositivo, agregue esto antes de la acción: (asumiendo que su modelo de dispositivo es 'Usuario'):

```
before_action :authenticate_user!
```

Para verificar si un usuario ha iniciado sesión, use la siguiente ayuda:

```
user_signed_in?
```

Para el usuario que inició sesión actualmente, use este ayudante:

```
current_user
```

Puede acceder a la sesión para este ámbito:

```
user_session
```

- Tenga en cuenta que si su modelo de Devise se llama `Member` lugar de `User`, reemplace el `user` anterior por `member`

## Omniauth

Primero elige tu estrategia de autenticación y `Gemfile` a tu `Gemfile`. Puede encontrar una lista de estrategias aquí: <https://github.com/intridea/omniauth/wiki/List-of-Strategies>

```
gem 'omniauth-github', :github => 'intridea/omniauth-github'  
gem 'omniauth-openid', :github => 'intridea/omniauth-openid'
```

Puedes añadir esto a tu middleware de rieles así:

```
Rails.application.config.middleware.use OmniAuth::Builder do  
  require 'openid/store/filesystem'  
  provider :github, ENV['GITHUB_KEY'], ENV['GITHUB_SECRET']  
  provider :openid, :store => OpenID::Store::Filesystem.new('/tmp')  
end
```

De forma predeterminada, OmniAuth agregará el `/auth/:provider` a sus rutas y puede comenzar usando estas rutas.

De forma predeterminada, si hay un error, omniauth redireccionará a `/auth/failure`

## has\_secure\_password

### Crear modelo de usuario

```
rails generate model User email:string password_digest:string
```

### Agregar el módulo `has_secure_password` al modelo de usuario

```
class User < ActiveRecord::Base  
  has_secure_password  
end
```

Ahora puedes crear un nuevo usuario con contraseña.

```
user = User.new email: 'bob@bob.com', password: 'Password1', password_confirmation:  
'Password1'
```

Verifique la contraseña con el método de autenticación

```
user.authenticate('somepassword')
```

## has\_secure\_token

### Crear modelo de usuario

```
# Schema: User(token:string, auth_token:string)  
class User < ActiveRecord::Base  
  has_secure_token  
  has_secure_token :auth_token  
end
```

Ahora, cuando creas un nuevo usuario, un token y `auth_token` se generan automáticamente

```
user = User.new
user.save
user.token # => "pX27zsMN2ViQKta1bGfLmVJE"
user.auth_token # => "77TMHrHJFvFDwodq8w7Ev2m7"
```

Puedes actualizar los tokens usando `regenerate_token` y `regenerate_auth_token`

```
user.regenerate_token # => true
user.regenerate_auth_token # => true
```

Lea Autenticación de usuario en rieles en línea: <https://riptutorial.com/es/ruby-on-rails/topic/1794/autenticacion-de-usuario-en-rieles>

---

# Capítulo 20: Autorización con CanCan

## Introducción

[CanCan](#) es una estrategia de autorización simple para Rails que se desacopla de los roles de usuario. Todos los permisos se almacenan en una sola ubicación.

## Observaciones

Antes de usar CanCan, no olvide crear Usuarios por gema o manualmente. Para obtener la máxima funcionalidad de CanCan puede crear un usuario administrador.

## Examples

### Empezando con CanCan

[CanCan](#) es una popular biblioteca de autorizaciones para Ruby on Rails que restringe el acceso de los usuarios a recursos específicos. La última gema (CanCanCan) es una continuación del proyecto muerto [CanCan](#).

Los permisos se definen en la clase de `Ability` y se pueden usar desde controladores, vistas, ayudantes o cualquier otro lugar en el código.

Para agregar soporte de autorización a una aplicación, agregue la gema CanCanCan al `Gemfile`:

```
gem 'cancancan'
```

Luego define la clase de habilidad:

```
# app/models/ability.rb
class Ability
  include CanCan::Ability

  def initialize(user)
    end
  end
end
```

Luego, verifique la autorización usando `load_and_authorize_resource` para cargar modelos autorizados en el controlador:

```
class ArticlesController < ApplicationController
  load_and_authorize_resource

  def show
    # @article is already loaded and authorized
  end
end
```

`authorize!` para comprobar la autorización o plantear una excepción

```
def show
  @article = Article.find(params[:id])
  authorize! :read, @article
end
```

`can?` para verificar si un objeto está autorizado contra una acción particular en cualquier lugar en los controladores, vistas o ayudantes

```
<% if can? :update, @article %>
  <%= link_to "Edit", edit_article_path(@article) %>
<% end %>
```

**Nota:** Esto supone que el usuario `current_user` es proporcionado por el método `current_user`.

## Definiendo habilidades

Las habilidades se definen en la clase `Ability` utilizando los métodos de `can` y `cannot`. Considere el siguiente ejemplo comentado para referencia básica:

```
class Ability
  include CanCan::Ability

  def initialize(user)
    # for any visitor or user
    can :read, Article

    if user
      if user.admin?
        # admins can do any action on any model or action
        can :manage, :all
      else
        # regular users can read all content
        can :read, :all
        # and edit, update and destroy their own user only
        can [:edit, :destroy], User, id: user_id
        # but cannot read hidden articles
        cannot :read, Article, hidden: true
      end
    else
      # only unlogged visitors can visit a sign_up page:
      can :read, :sign_up
    end
  end
end
```

## Manejando gran cantidad de habilidades.

Una vez que el número de definiciones de habilidades comienza a crecer en número, se vuelve cada vez más difícil manejar el archivo de `Habilidad`.

La primera estrategia para manejar estos problemas es mover las habilidades a métodos significativos, según este ejemplo:

```

class Ability
  include CanCan::Ability

  def initialize(user)
    anyone_abilities

    if user
      if user.admin?
        admin_abilities
      else
        authenticated_abilities
      end
    else
      guest_abilities
    end
  end

  private

  def anyone_abilities
    # define abilities for everyone, both logged users and visitors
  end

  def guest_abilities
    # define abilities for visitors only
  end

  def authenticated_abilities
    # define abilities for logged users only
  end

  def admin_abilities
    # define abilities for admins only
  end
end

```

Una vez que esta clase crezca lo suficiente, puede intentar dividirla en diferentes clases para manejar las diferentes responsabilidades de esta manera:

```

# app/models/ability.rb
class Ability
  include CanCan::Ability

  def initialize(user)
    self.merge Abilities::Everyone.new(user)

    if user
      if user.admin?
        self.merge Abilities::Admin.new(user)
      else
        self.merge Abilities::Authenticated.new(user)
      end
    else
      self.merge Abilities::Guest.new(user)
    end
  end
end

```

y luego definir esas clases como:

```
# app/models/abilities/guest.rb
module Abilities
  class Guest
    include CanCan::Ability

    def initialize(user)
      # Abilities for anonymous visitors only
    end
  end
end
```

y así sucesivamente con `Abilities::Authenticated`, `Abilities::Admin` o cualquier otro.

## Prueba rápidamente una habilidad

Si desea probar rápidamente si una clase de habilidad está dando los permisos correctos, puede inicializar una habilidad en la consola o en otro contexto con el entorno de rieles cargados, solo pase una instancia de usuario para probar:

```
test_ability = Ability.new(User.first)
test_ability.can?(:show, Post) #=> true
other_ability = Ability.new(RestrictedUser.first)
other_ability.cannot?(:show, Post) #=> true
```

Más información: <https://github.com/ryanb/cancan/wiki/Testing-Abilities>

Lea Autorización con CanCan en línea: <https://riptutorial.com/es/ruby-on-rails/topic/3021/autorizacion-con-cancan>

# Capítulo 21: Ayudantes de formulario

## Introducción

Rails proporciona vistas de ayuda para generar el formato de formulario.

## Observaciones

- Los tipos de entrada de fecha que incluyen `date`, `date` y `datetime`, `datetime-local`, `time`, `month` y `week` no funcionan en FireFox.
- `input<type="telephone">` solo funciona con Safari 8.
- `input<type="email">` no funciona en Safari

## Examples

### Crear un formulario

Puedes crear un formulario usando el ayudante `form_tag`

```
<%= form_tag do %>
  Form contents
<% end %>
```

Esto crea el siguiente HTML

```
<form accept-charset="UTF-8" action="/" method="post">
  <input name="utf8" type="hidden" value="&#x2713;" />
  <input name="authenticity_token" type="hidden"
value="J7CBxfHalt49OSHp27hblqK20c9PgWJ108nDHX/8Cts=" />
  Form contents
</form>
```

Esta etiqueta de formulario ha creado un campo de entrada `hidden`. Esto es necesario, porque los formularios no se pueden enviar correctamente sin él.

El segundo campo de entrada, denominado `authenticity_token` agrega protección contra la `cross-site request forgery`.

### Creación de un formulario de búsqueda

Para crear un formulario de búsqueda, ingrese el siguiente código

```
<%= form_tag("/search", method: "get") do %>
  <%= label_tag(:q, "Search for:") %>
  <%= text_field_tag(:q) %>
  <%= submit_tag("Search") %>
<% end %>
```

- `form_tag` : Este es el ayudante predeterminado para crear un formulario. Es el primer parámetro, `/search` es la acción y el segundo parámetro especifica el método HTTP. Para los formularios de búsqueda, es importante utilizar siempre el método `get`
- `label_tag` : este ayudante crea una `<label>` html `<label>` .
- `text_field_tag` : Esto creará un elemento de entrada con `text` tipo
- `submit_tag` : esto crea un elemento de entrada con el tipo `submit`

## Ayudantes para elementos de forma.

### Casillas de verificación

```
<%= check_box_tag(:pet_dog) %>
<%= label_tag(:pet_dog, "I own a dog") %>
<%= check_box_tag(:pet_cat) %>
<%= label_tag(:pet_cat, "I own a cat") %>
```

Esto generará el siguiente html

```
<input id="pet_dog" name="pet_dog" type="checkbox" value="1" />
<label for="pet_dog">I own a dog</label>
<input id="pet_cat" name="pet_cat" type="checkbox" value="1" />
<label for="pet_cat">I own a cat</label>
```

### Botones de radio

```
<%= radio_button_tag(:age, "child") %>
<%= label_tag(:age_child, "I am younger than 18") %>
<%= radio_button_tag(:age, "adult") %>
<%= label_tag(:age_adult, "I'm over 18") %>
```

Esto genera el siguiente HTML

```
<input id="age_child" name="age" type="radio" value="child" />
<label for="age_child">I am younger than 18</label>
<input id="age_adult" name="age" type="radio" value="adult" />
<label for="age_adult">I'm over 18</label>
```

### Area de texto

Para crear un cuadro de texto más grande, se recomienda usar la `text_area_tag`

```
<%= text_area_tag(:message, "This is a longer text field", size: "25x6") %>
```

Esto creará el siguiente HTML

```
<textarea id="message" name="message" cols="25" rows="6">This is a longer text
field</textarea>
```

## Campo de número

Esto creará un elemento de `input<type="number">`

```
<%= number_field :product, :rating %>
```

Para especificar un rango de valores, podemos usar la opción `in`:

```
<%= number_field :product, :rating, in: 1..10 %>
```

## Campo de contraseña

A veces quieres que los caracteres escritos por el usuario sean enmascarados. Esto generará un `<input type="password">`

```
<%= password_field_tag(:password) %>
```

## Campo de correo electrónico

Esto creará un `<input type="email">`

```
<%= email_field(:user, :email) %>
```

## Campo telefonico

Esto creará un `<input type="tel">`.

```
<%= telephone_field :user, :phone %>
```

## Fecha ayudantes

- `input [type="date"]`

```
<%= date_field(:user, :reservation) %>
```

- `input [type="week"]`

```
<%= week_field(:user, :reservation) %>
```

- `input [type="year"]`

```
<%= year_field(:user, :reservation) %>
```

- `input [type="time"]`

```
<%= time_field(:user, :check_in) %>
```

## Desplegable

Ejemplo estándar: `@models = Model.all select_tag "models", options_from_collection_for_select (@models, "id", "name"), {}`

Esto generará el siguiente HTML: David

El último argumento son opciones, que aceptan lo siguiente: `{multiple: false, disabled: false, include_blank: false, prompt: false}`

Se pueden encontrar más ejemplos:

[http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select\\_tag](http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select_tag)

Lea Ayudantes de formulario en línea: <https://riptutorial.com/es/ruby-on-rails/topic/4509/ayudantes-de-formulario>

---

# Capítulo 22: Cambiar un entorno de aplicación de Rails predeterminado

## Introducción

Esto tratará sobre cómo cambiar el entorno, de modo que cuando alguien escribe `rails s` no inicia el desarrollo sino el entorno que desea.

## Examples

### Corriendo en una máquina local

Normalmente cuando el entorno de los rieles se ejecuta escribiendo. Esto solo ejecuta el entorno por defecto que normalmente es `development`

```
rails s
```

El entorno específico se puede seleccionar utilizando la bandera `-e` por ejemplo:

```
rails s -e test
```

Que ejecutará el entorno de prueba.

El entorno predeterminado se puede cambiar en el terminal editando el archivo `~/.bashrc` y agregando la siguiente línea:

```
export RAILS_ENV=production in your
```

### Corriendo en un servidor

Si se ejecuta en un servidor remoto que usa Passenger, cambie `apache.conf` al entorno que desea usar. Por ejemplo este caso ves la `RailsEnv production`.

```
<VirtualHost *:80>
  ServerName application_name.rails.local
  DocumentRoot "/Users/rails/application_name/public"
  RailsEnv production ## This is the default
</VirtualHost>
```

Lea [Cambiar un entorno de aplicación de Rails predeterminado en línea](https://riptutorial.com/es/ruby-on-rails/topic/9915/cambiar-un-entorno-de-aplicacion-de-rails-predeterminado):

<https://riptutorial.com/es/ruby-on-rails/topic/9915/cambiar-un-entorno-de-aplicacion-de-rails-predeterminado>

---

# Capítulo 23: Cambiar zona horaria predeterminada

## Observaciones

`config.active_record.default_timezone` determina si se debe usar `Time.local` (si se establece en: `local`) o `Time.utc` (si se establece en: `utc`) al extraer fechas y horas de la base de datos. El valor predeterminado es: `utc`.

<http://guides.rubyonrails.org/configuring.html>

---

Si desea cambiar la zona horaria de **Rails** , pero sigue teniendo el **registro activo** guardado en la base de datos en **UTC** , use

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
```

---

Si desea cambiar la zona horaria de **Rails** **Y** tener tiempos de almacenamiento de **Active Record** en esta zona horaria, use

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
config.active_record.default_timezone = :local
```

**Advertencia** : realmente debería pensarlo dos veces, incluso tres veces, antes de guardar los tiempos en la base de datos en un formato no UTC.

### Nota

No olvide reiniciar su servidor Rails después de modificar `application.rb` .

---

Recuerde que `config.active_record.default_timezone` puede tomar solo dos valores

- : **local** (se convierte a la zona horaria definida en `config.time_zone` )
- : **utc** (convierte a UTC)

---

Así es como puedes encontrar todas las zonas horarias disponibles

```
rake time:zones:all
```

## Examples

Cambie la zona horaria de Rails, pero continúe guardando Active Record en la

## base de datos en UTC

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
```

## Cambiar la zona horaria de Rails Y tener tiempos de almacenamiento Active Record en esta zona horaria

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
config.active_record.default_timezone = :local
```

Lea Cambiar zona horaria predeterminada en línea: <https://riptutorial.com/es/ruby-on-rails/topic/3367/cambiar-zona-horaria-predeterminada>

---

# Capítulo 24: Característica de pago en rieles.

## Introducción

Este documento pretende presentarle, con un ejemplo completo, cómo puede implementar diferentes métodos de pago con Ruby on Rails.

En el ejemplo, cubriremos Stripe y Braintree dos plataformas de pago muy conocidas.

## Observaciones

Documentación.

[Raya](#)

[Braintree](#)

## Examples

### Cómo integrar con Stripe

Agregue la gema de la raya a nuestro `Gemfile`

```
gem 'stripe'
```

Agregue el archivo `initializers/stripe.rb` . Este archivo contiene las claves necesarias para conectarse con su cuenta de banda.

```
require 'require_all'

Rails.configuration.stripe = {
  :publishable_key => ENV['STRIPE_PUBLISHABLE_KEY'],
  :secret_key      => ENV['STRIPE_SECRET_KEY']
}

Stripe.api_key = Rails.configuration.stripe[:secret_key]
```

---

## Cómo crear un nuevo cliente para Stripe

```
Stripe::Customer.create({email: email, source: payment_token})
```

Este código crea un nuevo cliente en Stripe con una dirección de correo electrónico y una fuente dadas.

`payment_token` es el token dado por el lado del cliente que contiene un método de pago como una

tarjeta de crédito o una cuenta bancaria. Más información: [Stripe.js del lado del cliente](#)

---

## Cómo recuperar un plan de Stripe

```
Stripe::Plan.retrieve(stripe_plan_id)
```

Este código recupera un plan de Stripe por su id.

---

## Cómo crear una suscripción

Cuando tenemos un cliente y un plan, podemos crear una nueva suscripción en Stripe.

```
Stripe::Subscription.create(customer: customer.id, plan: plan.id)
```

Crearé una nueva suscripción y cobrará a nuestro Usuario. Es importante saber qué sucede realmente en Stripe cuando suscribimos un usuario a un plan, encontrará más información aquí: [Ciclo de vida de la suscripción a Stripe](#) .

---

## Cómo cobrar a un usuario con un solo pago

A veces queremos cobrarles a nuestros usuarios solo una vez, por eso haremos lo siguiente.

```
Stripe::Charge.create(amount: amount, customer: customer, currency: currency)
```

En ese caso, estamos cobrando a nuestro usuario una vez por una cantidad determinada.

Errores comunes:

- La cantidad debe enviarse en forma de entero, lo que significa que 2000 serán 20 unidades de moneda. [Mira este ejemplo](#)
- No se puede cobrar a un usuario en dos monedas. Si el usuario fue cargado en EUR en algún momento en el pasado, no puede cobrar al usuario en USD.
- No puede cobrar al usuario sin fuente (método de pago).

Lea [Característica de pago en rieles](#). en línea: <https://riptutorial.com/es/ruby-on-rails/topic/10929/caracteristica-de-pago-en-rieles->

# Capítulo 25: Cargas de archivos

## Examples

### Carga de un solo archivo usando Carrierwave

Comenzar a usar Cargas de archivos en Rails es bastante simple, lo primero que debe hacer es elegir el complemento para administrar las cargas. Las más comunes son **Carrierwave** y **Paperclip**. Ambos son similares en funcionalidad y ricos en documentación en

Veamos el ejemplo con una imagen de carga de avatar simple con Carrierwave

Después de `bundle install Carrierwave`, escriba en la consola

```
$ rails generate uploader ProfileUploader
```

Esto creará un archivo de configuración ubicado en `/app/uploaders/profile_uploader.rb`

Aquí puede configurar el almacenamiento (es decir, local o en la nube), aplicar extensiones para manipulaciones de imágenes (es decir, generar pulgares a través de MiniMagick) y configurar la lista blanca de extensiones del lado del servidor

A continuación, cree una nueva migración con el tipo de cadena para `user_pic` y monte el cargador en el modelo `user.rb`.

```
mount_uploader :user_pic, ProfileUploader
```

A continuación, muestra un formulario para cargar un avatar (puede ser una vista de edición para el usuario)

```
<% form_for @user, html: { multipart: true } do |f| %>
  <%= f.file_field :user_pic, accept: 'image/png, image/jpg' %>
  <%= f.submit "update profile pic", class: "btn" %>
<% end %>
```

Asegúrese de incluir `{multipart: true}` en el formulario de pedido para procesar las cargas. Aceptar es opcional para establecer la lista blanca de extensión del lado del cliente.

Para mostrar un avatar, simplemente haga

```
<%= image_tag @user.user_pic.url %>
```

### Modelo anidado - subidas múltiples

Si desea crear múltiples subidas, lo primero que debe hacer es crear un nuevo modelo y establecer relaciones

Supongamos que desea varias imágenes para el modelo de producto. Cree un nuevo modelo y `belongs_to` su modelo principal

```
rails g model ProductPhoto

#product.rb
has_many :product_photos, dependent: :destroy
accepts_nested_attributes_for :product_photos

#product_photo.rb
belongs_to :product
mount_uploader :image_url, ProductPhotoUploader # make sure to include uploader (Carrierwave
example)
```

`accept_nested_attributes_for` es must, porque nos permite crear un formulario anidado, por lo que podemos cargar un nuevo archivo, cambiar el nombre del producto y establecer el precio desde un solo formulario

A continuación, crear formulario en una vista (editar / crear)

```
<%= form_for @product, html: { multipart: true } do |product|>

  <%= product.text_field :price # just normal type of field %>

  <%= product.fields_for :product_photos do |photo| # nested fields %>
    <%= photo.file_field :image, :multiple => true, name:
"product_photos[image_url][]" %>
  <% end %>
  <%= p.submit "Update", class: "btn" %>
<% end %>
```

El controlador no es nada especial, si no desea crear uno nuevo, simplemente cree uno nuevo dentro del controlador de su producto

```
# create an action
def upload_file
  printer = Product.find_by_id(params[:id])
  @product_photo = printer.product_photos.create(photo_params)
end

# strong params
private
def photo_params
  params.require(:product_photos).permit(:image)
end
```

Mostrar todas las imágenes en una vista

```
<% @product.product_photos.each do |i| %>
  <%= image_tag i.image.url, class: 'img-rounded' %>
<% end %>
```

Lea Cargas de archivos en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2831/cargas-de-archivos>

---

# Capítulo 26: Carriles 5

## Examples

### Creando una API de Ruby on Rails 5

Para crear una nueva API de Rails 5, abra un terminal y ejecute el siguiente comando:

```
rails new app_name --api
```

Se creará la siguiente estructura de archivos:

```
create
create  README.rdoc
create  Rakefile
create  config.ru
create  .gitignore
create  Gemfile
create  app
create  app/assets/javascripts/application.js
create  app/assets/stylesheets/application.css
create  app/controllers/application_controller.rb
create  app/helpers/application_helper.rb
create  app/views/layouts/application.html.erb
create  app/assets/images/.keep
create  app/mailers/.keep
create  app/models/.keep
create  app/controllers/concerns/.keep
create  app/models/concerns/.keep
create  bin
create  bin/bundle
create  bin/rails
create  bin/rake
create  bin/setup
create  config
create  config/routes.rb
create  config/application.rb
create  config/environment.rb
create  config/secrets.yml
create  config/environments
create  config/environments/development.rb
create  config/environments/production.rb
create  config/environments/test.rb
create  config/initializers
create  config/initializers/assets.rb
create  config/initializers/backtrace_silencers.rb
create  config/initializers/cookies_serializer.rb
create  config/initializers/filter_parameter_logging.rb
create  config/initializers/inflections.rb
create  config/initializers/mime_types.rb
create  config/initializers/session_store.rb
create  config/initializers/wrap_parameters.rb
create  config/locales
create  config/locales/en.yml
create  config/boot.rb
```

```
create config/database.yml
create db
create db/seeds.rb
create lib
create lib/tasks
create lib/tasks/.keep
create lib/assets
create lib/assets/.keep
create log
create log/.keep
create public
create public/404.html
create public/422.html
create public/500.html
create public/favicon.ico
create public/robots.txt
create test/fixtures
create test/fixtures/.keep
create test/controllers
create test/controllers/.keep
create test/mailers
create test/mailers/.keep
create test/models
create test/models/.keep
create test/helpers
create test/helpers/.keep
create test/integration
create test/integration/.keep
create test/test_helper.rb
create tmp/cache
create tmp/cache/assets
create vendor/assets/javascripts
create vendor/assets/javascripts/.keep
create vendor/assets/stylesheets
create vendor/assets/stylesheets/.keep
```

Esta estructura de archivos se creará dentro de una nueva carpeta llamada `app_name` . Contiene todos los recursos y el código necesarios para comenzar su proyecto.

Entra en la carpeta e instala las dependencias:

```
cd app_name
bundle install
```

También debe iniciar su base de datos. Rails utiliza SQLite como una base de datos predeterminada. Para crearlo, ejecute:

```
rake db:setup
```

Ahora ejecuta tu aplicación:

```
$ rails server
```

Cuando abre su navegador en `http://localhost:3000` , su API nueva y vacía (vacía) debería estar en ejecución.

## Cómo instalar Ruby on Rails 5 en RVM

RVM es una gran herramienta para administrar sus versiones de ruby y configurar su entorno de trabajo.

Suponiendo que ya tiene RVM instalado, para obtener la última versión de ruby, que es necesaria para estos ejemplos, abra un terminal y ejecute:

```
$ rvm get stable
$ rvm install ruby --latest
```

Comprueba tu versión ruby ejecutando:

```
$ ruby -v
> ruby 2.3.0p0
```

Para instalar Rails 5, primero cree un gemset nuevo con la última versión de ruby y luego instale los rieles:

```
$ rvm use ruby-2.3.0@my_app --create
$ gem install rails
```

Para verificar su versión de rieles, ejecute:

```
$ rails -v
> Rails 5.0.0
```

Lea Carriles 5 en línea: <https://riptutorial.com/es/ruby-on-rails/topic/3019/carriles-5>

---

# Capítulo 27: Chica de fábrica

## Examples

### Definiendo fábricas

Si tiene una clase de usuario de ActiveRecord con nombre y atributos de correo electrónico, puede crear una fábrica para ella haciendo que FactoryGirl lo adivine:

```
FactoryGirl.define do
  factory :user do # it will guess the User class
    name      "John"
    email     "john@example.com"
  end
end
```

O puedes hacerlo explícito e incluso cambiar su nombre:

```
FactoryGirl.define do
  factory :user_jack, class: User do
    name      "Jack"
    email     "jack@example.com"
  end
end
```

Luego, en su especificación, puede utilizar los métodos de FactoryGirl con estos, como este:

```
# To create a non saved instance of the User class filled with John's data
build(:user)
# and to create a non saved instance of the User class filled with Jack's data
build(:user_jack)
```

Los métodos más comunes son:

```
# Build returns a non saved instance
user = build(:user)

# Create returns a saved instance
user = create(:user)

# Attributes_for returns a hash of the attributes used to build an instance
attrs = attributes_for(:user)
```

Lea Chica de fábrica en línea: <https://riptutorial.com/es/ruby-on-rails/topic/8330/chica-de-fabrica>

---

# Capítulo 28: Columnas multiusos de ActiveRecord

## Sintaxis

- `serialize: <field_plural_symbol>`

## Examples

### Guardar un objeto

Si tiene un atributo que debe guardarse y recuperarse en la base de datos como un objeto, especifique el nombre de ese atributo utilizando el método de `serialize` y se manejará automáticamente.

El atributo debe ser declarado como un campo de `text` .

En el modelo debe declarar el tipo de campo ( `Hash` o `Array` )

Más información en: [serializar >> apidock.com](#)

### Cómo

---

## En tu migración

```
class Users < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      ...
      t.text :preference
      t.text :tag
      ...
      t.timestamps
    end
  end
end
```

---

## En tu modelo

```
class User < ActiveRecord::Base
  serialize :preferences, Hash
  serialize :tags, Array
end
```

Lea Columnas multiusos de ActiveRecord en línea: <https://riptutorial.com/es/ruby-on-rails/topic/7602/columnas-multiusos-de-activerecord>

---

# Capítulo 29: Configuración

## Examples

### Configuración personalizada

Cree un archivo `YAML` en el directorio `config/`, por ejemplo: `config/neo4j.yml`

El contenido de `neo4j.yml` puede ser algo como el siguiente (por simplicidad, el `default` se usa para todos los entornos):

```
default: &default
  host: localhost
  port: 7474
  username: neo4j
  password: root

development:
  <<: *default

test:
  <<: *default

production:
  <<: *default
```

en `config/application.rb`:

```
module MyApp
  class Application < Rails::Application
    config.neo4j = config_for(:neo4j)
  end
end
```

Ahora, su configuración personalizada es accesible como a continuación:

```
Rails.configuration.neo4j['host']
#=> localhost
Rails.configuration.neo4j['port']
#=> 7474
```

---

### Más información

El documento oficial de la API de Rails describe el método `config_for` como:

Conveniencia para cargar `config / foo.yml` para la versión actual de Rails.

---

**Si no quieres usar un archivo `yml`**

Puede configurar su propio código a través del objeto de configuración de Rails con una configuración personalizada bajo la propiedad `config.x`.

## Ejemplo

```
config.x.payment_processing.schedule = :daily
config.x.payment_processing.retries = 3
config.x.super_debugger = true
```

Estos puntos de configuración están disponibles a través del objeto de configuración:

```
Rails.configuration.x.payment_processing.schedule # => :daily
Rails.configuration.x.payment_processing.retries # => 3
Rails.configuration.x.super_debugger            # => true
Rails.configuration.x.super_debugger.not_set    # => nil
```

Lea Configuración en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2558/configuracion>

---

# Capítulo 30: Configuración

## Examples

### Entornos en rieles

Los archivos de configuración para los rieles se pueden encontrar en `config/environments/`. Por defecto, los rieles tienen 3 entornos, `development`, `production` y `test`. Al editar cada archivo, solo está editando la configuración para ese entorno.

Rails también tiene un archivo de configuración en `config/application.rb`. Este es un archivo de configuración común ya que cualquier configuración definida aquí se sobrescribe con la configuración especificada en cada entorno.

Usted agrega o modifica las opciones de configuración dentro de `Rails.application.configure` do block y las opciones de configuración comienzan con la `config`.

### Configuración de la base de datos

La configuración de la base de datos de un proyecto de rieles se encuentra en un archivo `config/database.yml`. Si crea un proyecto con el `rails new` comando `rails new` y no especifica un motor de base de datos para usar, rails usa `sqlite` como la base de datos predeterminada. Un archivo típico `database.yml` con configuración predeterminada se verá similar al siguiente.

```
# SQLite version 3.x
# gem install sqlite3
#
# Ensure the SQLite 3 gem is defined in your Gemfile
# gem 'sqlite3'
#
default: &default
  adapter: sqlite3
  pool: 5
  timeout: 5000

development:
  <<: *default
  database: db/development.sqlite3

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  <<: *default
  database: db/test.sqlite3

production:
  <<: *default
  database: db/production.sqlite3
```

Si desea cambiar la base de datos predeterminada al crear un nuevo proyecto, puede especificar

la base de datos: `rails new hello_world --database=mysql`

## Configuración general de rieles

Las siguientes opciones de configuración deben llamarse en un objeto `Rails::Railtie`

- **config.after\_initialize** : toma un bloque que se ejecutará después de que los rieles hayan inicializado la aplicación.
- **config.asset\_host** : Esto establece el host para los activos. Esto es útil cuando se utiliza una *red de entrega de contenido* . Esto es una abreviatura de `config.action_controller.asset_host`
- **config.autoload\_once\_paths** : esta opción acepta una matriz de rutas en las que Rails se carga automáticamente. El valor predeterminado es una matriz vacía.
- **config.autoload\_paths** : Esto acepta una matriz de rutas en las que Rails carga automáticamente las constantes. Por defecto, todos los directorios en la `app`
- **config.cache\_classes** : determina si las clases y los módulos deben recargarse en cada solicitud. En el modo de desarrollo, este valor predeterminado es `false` y en los modos de producción y prueba el valor predeterminado es `true`
- **config.action\_view.cache\_template\_loading** : Esto determina si las plantillas deben volver a cargarse en cada solicitud. El valor predeterminado es la configuración `config.cache_classes`
- **config.beginning\_of\_week** : Esto establece el inicio de semana predeterminado. Requiere un símbolo de día de la semana válido ( `:monday` )
- **config.cache\_store** : elija qué almacén de caché usar. Las opciones incluyen `:file_store` `:memory_store` , `mem_cache_store` O `null_store` .
- **config.colorize\_logging** : Esto controla si la información de registro está coloreada
- **config.eager\_load** : Eager-cargas todas las registradas
- **config.encoding** : especifica la codificación de la aplicación. El valor predeterminado es `UTF-8`
- **config.log\_level** : establece la verbosidad del registrador de Rails. El valor predeterminado es `:debug` en todos los entornos.
- **config.middleware** : use esto para configurar el middleware de la aplicación
- **config.time\_zone** : Esto establece la zona horaria predeterminada de la aplicación.

## Configurando activos

Las siguientes opciones de configuración se pueden utilizar para configurar activos

- **config.assets.enabled** : determina si la canalización de activos está habilitada. Esto por defecto es verdadero
- **config.assets.raise\_runtime\_errors** : Esto habilita la verificación de errores en tiempo de ejecución. Es útil para el `development mode`
- **config.assets.compress** : permite comprimir los activos. En modo de producción, este por defecto es verdadero
- **config.assets.js\_compressor** : especifica qué compresor JS utilizar. Las opciones incluyen `:closure` `:uglifyjs` Y `:yui`
- **config.assets.paths** : Especifica qué rutas buscar los activos.
- **config.assets.precompile** : le permite elegir activos adicionales para ser precompilados

cuando los `rake assets:precompile` se ejecuta

- **config.assets.digest** : esta opción permite el uso de huellas dactilares MD-5 en los nombres de activos. Por defecto es verdadero en modo de desarrollo
- **config.assets.compile** : Alterna la compilación de `sprockets` en vivo en modo de producción

## Configurando generadores

Los rieles le permiten configurar qué generadores se utilizan cuando los `rails generate` ejecución `rails generate` comandos. Este método, `config.generators` toma un bloque.

```
config.generators do |g|
  g.orm :active_record
  g.test_framework :test_unit
end
```

Estas son algunas de las opciones.

Opción	Descripción	Defecto
bienes	Crea activos al generar andamios.	cierto
force_plural	Permite nombres de modelos pluralizados.	falso
ayudante	Determina si generar ayudantes.	cierto
herramienta de integración	Especificar herramienta de integración.	test_unit
javascript_engine	Configura el motor JS	:js
resource_route	Genera ruta de recursos.	cierto
stylesheet_engine	Configura el motor de hojas de estilo.	:cs
scaffold_stylesheet	Crea CSS sobre andamios	cierto
test_framework	Especificar marco de prueba	Minitest
template_engine	Configura el motor de plantillas.	:erb

Lea Configuración en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2841/configuracion>

---

# Capítulo 31: Configurar Angular con Rieles

## Examples

Angular con rieles 101

---

## Paso 1: Crea una nueva aplicación Rails

```
gem install rails -v 4.1
rails new angular_example
```

---

## Paso 2: Eliminar Turbolinks

La eliminación de turbolinks requiere su eliminación de Gemfile.

```
gem 'turbolinks'
```

Elimine el `require` de `app/assets/javascripts/application.js` :

```
//= require turbolinks
```

---

## Paso 3: Agregar AngularJS a la tubería de activos

Para que Angular funcione con el canal de activos de Rails, debemos agregarlo al Gemfile:

```
gem 'angular-rails-templates'
gem 'bower-rails'
```

Ahora ejecuta el comando

```
bundle install
```

Agregue `bower` para que podamos instalar la dependencia de AngularJS:

```
rails g bower_rails:initialize json
```

Agregue Angular a `bower.json` :

```
{
```

```
"name": "bower-rails generated dependencies",

"dependencies": {

  "angular": "latest",
  "angular-resource": "latest",
  "bourbon": "latest",
  "angular-bootstrap": "latest",
  "angular-ui-router": "latest"
}
}
```

Ahora que `bower.json` está configurado con las dependencias correctas, `bower.json` :

```
bundle exec rake bower:install
```

---

## Paso 4: Organiza la aplicación Angular

Cree la siguiente estructura de carpetas en `app/assets/javascript/angular-app/` :

```
templates/
modules/
filters/
directives/
models/
services/
controllers/
```

En `app/assets/javascripts/application.js` , `add require` para Angular, el asistente de plantillas y la estructura de archivos de la aplicación Angular. Me gusta esto:

```
//= require jquery
//= require jquery_ujs

//= require angular
//= require angular-rails-templates
//= require angular-app/app

//= require_tree ./angular-app/templates
//= require_tree ./angular-app/modules
//= require_tree ./angular-app/filters
//= require_tree ./angular-app/directives
//= require_tree ./angular-app/models
//= require_tree ./angular-app/services
//= require_tree ./angular-app/controllers
```

---

## Paso 5: Bootstrap la aplicación Angular

Crear una `app/assets/javascripts/angular-app/app.js.coffee` :

```
@app = angular.module('app', [ 'templates' ])
```

```
@app.config([ '$httpProvider', ($httpProvider)->
$httpProvider.defaults.headers.common['X-CSRF-Token'] =
$('meta[name=csrf-token]').attr('content') ]) @app.run(-> console.log 'angular app running'
)
```

Cree un módulo Angular en `app/assets/javascripts/angular-app/modules/example.js.coffee.erb` :

```
@exampleApp = angular.module('app.exampleApp', [ # additional dependencies here ])
.run(-> console.log 'exampleApp running' )
```

Cree un controlador Angular para esta aplicación en `app/assets/javascripts/angular-app/controllers/exampleCtrl.js.coffee` :

```
angular.module('app.exampleApp').controller("ExampleCtrl", [ '$scope', ($scope)->
console.log 'ExampleCtrl running' $scope.exampleValue = "Hello angular and rails" ])
```

Ahora agregue una ruta a Rails para pasar el control a Angular. En `config/routes.rb` :

```
Rails.application.routes.draw do get 'example' => 'example#index' end
```

Genere el controlador Rails para responder a esa ruta:

```
rails g controller Example
```

En `app/controllers/example_controller.rb` :

```
class ExampleController < ApplicationController
  def index
    end
end
```

En la vista, debemos especificar qué aplicación Angular y qué controlador Angular manejará esta página. Así que en `app/views/example/index.html.erb` :

```
<div ng-app='app.exampleApp' ng-controller='ExampleCtrl'>

  <p>Value from ExampleCtrl:</p>
  <p>{{ exampleValue }}</p>

</div>
```

Para ver la aplicación, inicie su servidor Rails y visite <http://localhost:3000/example> .

Lea [Configurar Angular con Rieles en línea: https://riptutorial.com/es/ruby-on-rails/topic/3902/configurar-angular-con-rieles](https://riptutorial.com/es/ruby-on-rails/topic/3902/configurar-angular-con-rieles)

---

# Capítulo 32: Constantize seguro

## Examples

### Éxito seguro\_constantizar

User es una clase ActiveRecord o Mongoid . Reemplace al User con cualquier clase de Rails en su proyecto (incluso algo como Integer o Array )

```
my_string = "User" # Capitalized string
# => 'User'
my_constant = my_string.safe_constantize
# => User
my_constant.all.count
# => 18

my_string = "Array"
# => 'Array'
my_constant = my_string.safe_constantize
# => Array
my_constant.new(4)
# => [nil, nil, nil, nil]
```

### Safe\_constantize sin éxito

Este ejemplo no funcionará porque la cadena pasada no se reconoce como una constante en el proyecto. Incluso si pasa en "array" , no funcionará ya que no está en mayúsculas.

```
my_string = "not_a_constant"
# => 'not_a_constant'
my_string.safe_constantize
# => nil

my_string = "array" #Not capitalized!
# => 'array'
my_string.safe_constantize
# => nil
```

Lea Constantize seguro en línea: <https://riptutorial.com/es/ruby-on-rails/topic/3015/constantize-seguro>

---

# Capítulo 33: Controlador de acción

## Introducción

Controlador de acción es la C en MVC. Una vez que el enrutador ha determinado qué controlador utilizar para una solicitud, el controlador es responsable de dar sentido a la solicitud y de producir la salida.

El controlador recibirá la solicitud, buscará o guardará datos de un modelo y usará una vista para crear resultados. Un controlador puede considerarse como un intermediario entre los modelos y las vistas. Hace que los datos del modelo estén disponibles para la vista para que puedan mostrarse al usuario, y guarda o actualiza los datos del usuario al modelo.

## Examples

### Salida JSON en lugar de HTML

```
class UsersController < ApplicationController
  def index
    hashmap_or_array = [{ name: "foo", email: "foo@example.org" }]

    respond_to do |format|
      format.html { render html: "Hello World" }
      format.json { render json: hashmap_or_array }
    end
  end
end
```

Además necesitarás la ruta:

```
resources :users, only: [:index]
```

Esto responderá de dos formas diferentes a las solicitudes de `/users` :

- Si visita `/users` o `/users.html` , mostrará una página html con el contenido `Hello World`
- Si visita `/users.json` , mostrará un objeto JSON que contiene:

```
[
  {
    "name": "foo",
    "email": "foo@example.org"
  }
]
```

Puede **omitir** `format.html { render inline: "Hello World" }` si quiere asegurarse de que su ruta solo responda a las solicitudes JSON.

## Controladores (básicos)

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render html: "Hello World" }
    end
  end
end
```

Este es un controlador básico, con la adición de la siguiente ruta (en route.rb):

```
resources :users, only: [:index]
```

Mostrará el mensaje de `Hello World` en una página web cuando acceda a la URL `/users`

## Parámetros

Los controladores tienen acceso a los parámetros HTTP (es posible que los conozcan como `?name=foo` en las URL, ¡pero Ruby on Rails también maneja diferentes formatos!) Y generan diferentes respuestas basadas en ellos. No hay una manera de distinguir entre los parámetros GET y POST, pero no debes hacer eso en ningún caso.

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html do
        if params[:name] == "john"
          render html: "Hello John"
        else
          render html: "Hello someone"
        end
      end
    end
  end
end
```

Como de costumbre nuestra ruta:

```
resources :users, only: [:index]
```

Acceda a la URL `/users?name=john` y la salida será `Hello John`, **access** `/users?name=whatever` y la salida será `Hello someone`

## Parámetros de filtrado (Básico)

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html do
        render html: "Hello #{ user_params[:name] } user_params[:sentence]"
      end
    end
  end
end
```

```

private

def user_params
  if params[:name] == "john"
    params.permit(:name, :sentence)
  else
    params.permit(:name)
  end
end
end
end

```

Puede permitir (o rechazar) algunos parámetros para que solo *pase* lo que usted desea y no tenga sorpresas desagradables, como las opciones de configuración del usuario que no deben cambiarse.

Visitar `/users?name=john&sentence=developer` mostrará el `/users?name=john&sentence=developer` Hello john developer , sin embargo visitando `/users?name=smith&sentence=spy` mostrará Hello smith , porque `:sentence` solo está permitida cuando accede como john

## Redirigiendo

Asumiendo la ruta:

```
resources :users, only: [:index]
```

Puede redirigir a una URL diferente usando:

```

class UsersController
  def index
    redirect_to "http://stackoverflow.com/"
  end
end
end

```

Puede volver a la página anterior que el usuario visitó usando:

```
redirect_to :back
```

Tenga en cuenta que en *Rails 5* la sintaxis para redirigir hacia atrás es diferente:

```
redirect_back fallback_location: "http://stackoverflow.com/"
```

El cual intentará redirigir a la página anterior y en caso de que no sea posible (el navegador está bloqueando el encabezado `HTTP_REFERER`), redirigirá a `:fallback_location`

## Usando vistas

Asumiendo la ruta:

```
resources :users, only: [:index]
```

Y el controlador:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render }
    end
  end
end
```

La vista de la `app/users/index.html.erb` será renderizada. Si la vista es:

```
Hello <strong>World</strong>
```

La salida será una página web con el texto: "Hola  **mundo** ".

Si quieres renderizar una vista diferente, puedes usar:

```
render "pages/home"
```

Y el archivo `app/views/pages/home.html.erb` se usará en su lugar.

Puede pasar variables a vistas usando variables de instancia de controlador:

```
class UsersController < ApplicationController
  def index
    @name = "john"

    respond_to do |format|
      format.html { render }
    end
  end
end
```

Y en el archivo `app/views/users/index.html.erb` puede usar `@name` :

```
Hello <strong><%= @name %></strong>
```

Y la salida será: "Hola  **John** ".

Una nota importante sobre la sintaxis de procesamiento, puede omitir completamente la sintaxis de `render` , Rails asume que si la omite. Así que:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render }
    end
  end
end
```

Se puede escribir en su lugar como:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html
    end
  end
end
```

Rails es lo suficientemente inteligente como para darse cuenta de que debe representar el archivo `app/views/users/index.html.erb`.

## 404 cuando no se encuentra el registro

Rescate del error de registro no encontrado en lugar de mostrar una excepción o una página en blanco:

```
class ApplicationController < ActionController::Base

  # ... your other stuff here

  rescue_from ActiveRecord::RecordNotFound do |exception|
    redirect_to root_path, 404, alert: 'Record not found'
  end
end
```

## Controlador REST básico

```
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]

  def index
    @posts = Post.all
  end

  def show

  end

  def new
    @post = Post.new
  end

  def edit

  end

  def create
    @post = Post.new(post_params)

    respond_to do |format|
      if @post.save
        format.html { redirect_to @post, notice: 'Post was successfully created.' }
        format.json { render :show, status: :created, location: @post }
      else
        format.html { render :new }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end
end
```

```

    end
  end
end

def update
  respond_to do |format|
    if @post.update(post_params)
      format.html { redirect_to @post.company, notice: 'Post was successfully updated.' }
      format.json { render :show, status: :ok, location: @post }
    else
      format.html { render :edit }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end

def destroy
  @post.destroy
  respond_to do |format|
    format.html { redirect_to posts_url, notice: 'Post was successfully destroyed.' }
    format.json { head :no_content }
  end
end

private

def set_post
  @post = Post.find(params[:id])
end

def post_params
  params.require(:post).permit(:title, :body, :author)
end
end

```

## Mostrar páginas de error para excepciones

Si desea mostrar a sus usuarios errores significativos en lugar de un simple "perdón, algo salió mal", Rails tiene una buena utilidad para este propósito.

Abra el archivo `app/controllers/application_controller.rb` y debería encontrar algo como esto:

```

class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
end

```

Ahora podemos agregar un `rescue_from` para recuperarse de errores específicos:

```

class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
  rescue_from ActiveRecord::RecordNotFound, with: :record_not_found

  private

  def record_not_found
    render html: "Record <strong>not found</strong>", status: 404
  end
end

```

Se recomienda no rescatar de `Exception` o `StandardError` contrario Rails no podrá mostrar páginas útiles en caso de errores.

## Filtros

Los filtros son métodos que se ejecutan "antes", "después" o "alrededor" de una acción del controlador. Se heredan, por lo que si establece alguno en su `ApplicationController`, se ejecutarán para cada solicitud que reciba su aplicación.

### Antes del filtro

Antes de que los filtros se ejecuten antes de la acción del controlador y puedan detener la solicitud (y / o redirigir). Un uso común es verificar si un usuario ha iniciado sesión:

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
    redirect_to some_path unless user_signed_in?
  end
end
```

Antes de que se ejecuten los filtros en las solicitudes antes de que la solicitud llegue a la acción del controlador. Puede devolver una respuesta por sí misma y omitir completamente la acción.

Otros usos comunes de los filtros anteriores son la validación de la autenticación de un usuario antes de otorgarles acceso a la acción designada para manejar su solicitud. También los he visto usar para cargar un recurso de la base de datos, verificar permisos en un recurso o administrar redirecciones en otras circunstancias.

### Después del filtro

Los filtros posteriores son similares a los "anteriores", pero a medida que se ejecutan después de la ejecución de la acción, tienen acceso al objeto de respuesta que está a punto de enviarse. En resumen, después de que se ejecutan los filtros después de que se completa la acción. Puede modificar la respuesta. La mayoría de las veces, si se hace algo en un filtro posterior, se puede hacer en la acción misma, pero si hay algo de lógica que ejecutar después de ejecutar cualquiera de un conjunto de acciones, entonces un filtro posterior es un buen lugar para hacer. eso.

En general, he visto después y alrededor de los filtros utilizados para el registro.

### Alrededor del filtro

Alrededor de los filtros puede haber lógica antes y después de la acción que se está ejecutando. Simplemente cede a la acción en cualquier lugar que sea necesario. Tenga en cuenta que no es necesario ceder a la acción y puede ejecutarse sin hacerlo como un filtro anterior.

Alrededor de los filtros son responsables de ejecutar sus acciones asociadas mediante el rendimiento, similar a cómo funcionan los middleware de Rack.

Alrededor de las devoluciones de llamada envuelve la ejecución de acciones. Puede escribir una vuelta de llamada en torno a dos estilos diferentes. En el primero, la devolución de llamada es una sola porción de código. Ese código se llama antes de que se ejecute la acción. Si el código de devolución de llamada invoca el rendimiento, la acción se ejecuta. Cuando la acción se completa, el código de devolución de llamada continúa ejecutándose. Por lo tanto, el código anterior al rendimiento es como una devolución de llamada de acción anterior y el código posterior al rendimiento es la devolución de llamada posterior a la acción. Si el código de devolución de llamada nunca invoca el rendimiento, la acción no se ejecuta; esto es lo mismo que tener un retorno de devolución de llamada de acción anterior falso.

Aquí hay un ejemplo del filtro de alrededor:

```
around_filter :catch_exceptions

private
  def catch_exceptions
    begin
      yield
    rescue Exception => e
      logger.debug "Caught exception! #{e.message}"
    end
  end
end
```

Esto detectará la excepción de cualquier acción y pondrá el mensaje en su registro. Puede usar alrededor de los filtros para el manejo de excepciones, la configuración y el desmontaje, y muchos otros casos.

## Solo y Excepto

Todos los filtros se pueden aplicar a acciones específicas, utilizando `:only` y `:except` :

```
class ProductsController < ApplicationController
  before_action :set_product, only: [:show, :edit, :update]

  # ... controller actions

  # Define your filters as controller private methods
  private

  def set_product
    @product = Product.find(params[:id])
  end
end
```

## Filtro de salto

También se pueden omitir todos los filtros (también los heredados) para algunas acciones específicas:

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
```

```

    redirect_to some_path unless user_signed_in?
  end
end

class HomeController < ApplicationController
  skip_before_action :authenticate_user!, only: [:index]

  def index
    end
end

```

A medida que se heredan, los filtros también se pueden definir en un controlador "primario" de namespace . Digamos, por ejemplo, que tiene un espacio de nombres de `admin` y, por supuesto, desea que solo los usuarios administradores puedan acceder a él. Podrías hacer algo como esto:

```

# config/routes.rb
namespace :admin do
  resources :products
end

# app/controllers/admin_controller.rb
class AdminController < ApplicationController
  before_action :authenticate_admin_user!

  private

  def authenticate_admin_user!
    redirect_to root_path unless current_user.admin?
  end
end

# app/controllers/admin/products_controller.rb
class Admin::ProductsController < AdminController
  # This controller will inherit :authenticate_admin_user! filter
end

```

Tenga en cuenta que en **Rails 4.x** puede usar `before_filter` junto con `before_action` , pero `before_filter` está actualmente en desuso en **Rails 5.0.0** y se eliminará en **5.1** .

## Generando un controlador

Rails proporciona muchos generadores, también para los controladores, por supuesto.

Puede generar un nuevo controlador ejecutando este comando en su carpeta de aplicaciones

```
rails generate controller NAME [action action] [options]
```

*Nota: también puede usar `rails g` alias para invocar `rails generate`*

Por ejemplo, para generar un controlador para un modelo de `Product` , con las acciones `#index` y `#show` que ejecutaría

```
rails generate controller products index show
```

Esto creará el controlador en `app/controllers/products_controller.rb` , con las dos acciones que especificó

```
class ProductsController < ApplicationController
  def index
  end

  def show
  end
end
```

También creará una carpeta de `products` dentro de `app/views/` , que contiene las dos plantillas para las acciones de su controlador (es decir, `index.html.erb` y `show.html.erb` , *tenga en cuenta que la extensión puede variar según el motor de su plantilla, por lo que si `show.html.slim` usando `slim` , por ejemplo, el generador creará `index.html.slim` y `show.html.slim` )*

Además, si especificó alguna acción, también se agregarán a su archivo de `routes`

```
# config/routes.rb
get 'products/show'
get 'products/index'
```

Rails crea un archivo de ayuda para usted, en `app/helpers/products_helper.rb` , y también los archivos de activos en `app/assets/javascripts/products.js` y `app/assets/stylesheets/products.css` . En cuanto a las vistas, el generador cambia este comportamiento de acuerdo con lo que se especifica en su `Gemfile` : es decir, si está utilizando `Coffeescript` y `Sass` en su aplicación, el generador del controlador generará `products.coffee` y `products.sass` .

Por último, pero no menos importante, Rails también genera archivos de prueba para su controlador, su ayudante y sus vistas.

Si no desea que se cree ninguno de estos, puede decirle a Rails que los omite, simplemente anteponga cualquier opción con

`--no-0 --skip` , así:

```
rails generate controller products index show --no-assets --no-helper
```

Y el generador saltará tanto los `assets` como el `helper`

Si necesita crear un controlador para un `namespace` específico, agréguelo delante de `NAME` :

```
rails generate controller admin/products
```

Esto creará su controlador dentro de `app/controllers/admin/products_controller.rb`

Rails también puede generar un controlador RESTful completo para usted:

```
rails generate scaffold_controller MODEL_NAME # available from Rails 4
rails generate scaffold_controller Product
```

## Rescatando ActiveRecord :: RecordNotFound con redirect\_to

Puede rescatar una excepción RecordNotFound con una redirección en lugar de mostrar una página de error:

```
class ApplicationController < ActionController::Base

  # your other stuff

  rescue_from ActiveRecord::RecordNotFound do |exception|
    redirect_to root_path, 404, alert: I18n.t("errors.record_not_found")
  end
end
```

Lea Controlador de acción en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2838/controlador-de-accion>

---

# Capítulo 34: Convenciones de nombres

## Examples

### Controladores

Los nombres de clase del controlador están pluralizados. La razón es que el controlador controla varias instancias de la instancia de objeto.

*Por ejemplo* : `OrdersController` sería el controlador para una tabla de `orders` . Rails buscará la definición de clase en un archivo llamado `orders_controller.rb` en el directorio `/app/controllers` .

*Por ejemplo* : `PostsController` sería el controlador para una tabla de `posts` .

Si el nombre de la clase del controlador tiene varias palabras en mayúsculas, se asume que el nombre de la tabla tiene guiones bajos entre estas palabras.

*Por ejemplo* : si un controlador se llama `PendingOrdersController` , el nombre de archivo asumido para este controlador será `pending_orders_controller.rb` .

### Modelos

El modelo se nombra utilizando la convención de nomenclatura de clases de `MixedCase` ininterrumpido y es siempre el singular del nombre de la tabla.

*Por ejemplo* : si una tabla se llamara `orders` , el modelo asociado se llamaría `Order`

*Por ejemplo* : si una tabla se llamara `posts` , el modelo asociado se llamaría `Post`

Rails buscará la definición de clase en un archivo llamado `order.rb` en el directorio `/app/models` .

Si el nombre de la clase modelo tiene múltiples palabras en mayúsculas, se asume que el nombre de la tabla tiene guiones bajos entre estas palabras.

*Por ejemplo* : si un modelo se llama `BlogPost` , el nombre de tabla asumido será `blog_posts` .

### Vistas y diseños

Cuando se representa una acción del controlador, Rails intentará encontrar un diseño y una vista coincidentes en función del nombre del controlador.

Las vistas y diseños se colocan en el directorio de `app/views` .

Dada una solicitud a la acción del `PeopleController#index` , Rails buscará:

- el diseño denominado `people` en `app/views/layouts/` (o `application` si no se encuentra ninguna coincidencia)
- una vista llamada `index.html.erb` en `app/views/people/` por defecto

- Si desea representar otro archivo llamado `index_new.html.erb` , debe escribir código para eso en `PeopleController#index` acción de `PeopleController#index` como `render 'index_new'`
- podemos establecer diferentes `layouts` para cada `action` escribiendo `render 'index_new', layout: 'your_layout_name'`

## Nombres de archivos y carga automática

Los archivos Rails, y los archivos Ruby en general, deben nombrarse con los `lower_snake_case` archivo `lower_snake_case` . P.ej

```
app/controllers/application_controller.rb
```

es el archivo que contiene la definición de la clase `ApplicationController` . Tenga en cuenta que, mientras que `PascalCase` se utiliza para nombres de clase y módulo, los archivos en los que residen deben ser `lower_snake_case` .

La nomenclatura coherente es importante ya que Rails utiliza los archivos de carga automática según sea necesario y utiliza la "inflexión" para transformar entre diferentes estilos de denominación, como la transformación del `application_controller` de `application_controller` en `ApplicationController` y viceversa.

Por ejemplo, si Rails ve que la clase `BlogPost` no existe (aún no se ha cargado), buscará un archivo llamado `blog_post.rb` e intentará cargar ese archivo.

Por lo tanto, también es importante nombrar los archivos por lo que contienen, ya que el autocargador espera que los nombres de los archivos coincidan con el contenido. Si, por ejemplo, la `blog_post.rb` lugar contiene una clase llamada solo `Post` , verá una `LoadError : Expected [some path]/blog_post.rb to define BlogPost` .

Si agrega un directorio en `app/something/` (por ejemplo, `/ models / products /`), y

- desea espacios de nombres y clases dentro de `new dir`, entonces no necesita hacer nada y se cargará solo. Por ejemplo, en la `app/models/products/` you would need to wrap your class in `módulo Productos``.
- no quiero que los espacios y las clases de espacio de nombres `config.autoload_paths += %W( #{config.root}/app/models/products )` dentro de mi nuevo directorio, entonces debes agregar `config.autoload_paths += %W( #{config.root}/app/models/products )` a tu `application.rb` para `autoload`.

Una cosa más a la que prestar atención (especialmente si el inglés no es su primer idioma) es el hecho de que Rails da cuenta de los sustantivos irregulares en plural en inglés. Por lo tanto, si tiene un modelo llamado "Foot", el controlador correspondiente debe llamarse "FeetController" en lugar de "FootController" si desea que el enrutamiento "mágico" de los rieles (y muchas de estas características) funcione.

## Clase de modelos del nombre del controlador

Puede obtener una clase de modelo de un nombre de controlador de esta manera (el contexto es

la clase de controlador):

```
class MyModelController < ActionController::Base

  # Returns corresponding model class for this controller
  # @return [ActiveRecord::Base]
  def corresponding_model_class
    # ... add some validation
    controller_name.classify.constantize
  end
end
```

Lea Convenciones de nombres en línea: <https://riptutorial.com/es/ruby-on-rails/topic/1493/convenciones-de-nombres>

---

# Capítulo 35: Depuración

## Examples

### Aplicación de depuración de rieles

Para poder depurar una aplicación es muy importante comprender el flujo de la lógica y los datos de una aplicación. Ayuda a resolver errores lógicos y agrega valor a la experiencia de programación y la calidad del código. Dos gemas populares para la depuración son el [depurador](#) (para ruby 1.9.2 y 1.9.3) y [byebug](#) (para ruby > = 2.x).

Para depurar archivos `.rb` , siga estos pasos:

1. Agregue el `debugger` o el `byebug` al grupo de `development` de `Gemfile`
2. Ejecutar `bundle install`
3. Añadir `debugger` o `byebug` como punto de interrupción
4. Ejecutar el código o realizar solicitud
5. Ver el registro del servidor de rieles detenido en el punto de interrupción especificado
6. En este punto, puede usar su terminal de servidor como la `rails console` y verificar los valores de variable y params
7. Para pasar a la siguiente instrucción, escriba `next` y presione `enter`
8. Para salir teclea `c` y presiona `enter`

Si desea depurar archivos `.html.erb` , el punto de interrupción se agregará como `<% debugger %>`

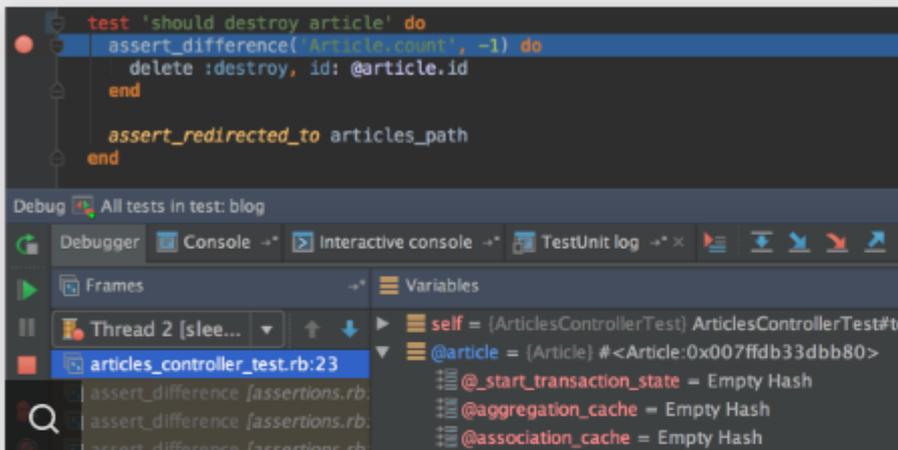
### Depurando en tu IDE

Cada [IDE](#) bueno proporciona una [GUI](#) para la depuración interactiva de aplicaciones de Ruby (y, por lo tanto, de Rails) donde puede agregar puntos de interrupción, relojes, pausas automáticas de excepción y le permite seguir la ejecución del código incluso paso a paso, línea por línea.

Por ejemplo, eche un vistazo a una de las mejores funciones de depuración de Ruby IDE, [RubyMine](#) en la imagen.

# Powerful Debugger

RubyMine brings a clever debugger with a graphical UI for Ruby, JS, and CoffeeScript. Set breakpoints and run your code step by step with all the information at your fingertips.



## Smart, flexible breakpoints

- Place a breakpoint on a line of code and define the hit conditions—a set of Boolean expressions that are evaluated to determine whether to stop the code execution or not.
- If you have multiple breakpoints in your code, you can set dependencies between them to define the order in which they can be hit.
- Setting a breakpoint is just a matter of a single mouse click on the gutter or invoking a shortcut.
- Breakpoints are also available in Rails views, so you can use them for debugging Rails code as well.

## Built-in expression evaluator

Evaluate any expression while your debugging session is paused. Type an expression or a code fragment, with coding assistance available in the dialog. All expressions are evaluated against the current context.

## Frames and call stack

When a breakpoint is hit or code execution is suspended, you can use the Frames panel to examine the current threads, their state, call stack, methods, and variables along with their values.

## Convenient user interface

- Look under the hood of any application with the Variables and Watches views.
- The UI is fully customizable: you can select toolbar commands, project code while stepping through it, and so on.
- The debugger UI is also tightly integrated with the IDE, so you can navigate between the debugger and the code editor, etc.
- You also get the complete set of debugging views.

## Debugging JavaScript

- RubyMine provides an advanced JavaScript debugger, which works with Google Chrome and Firefox.
- You can easily debug ECMAScript code running on RubyMine debugger's server.
- A full-featured debugger for JavaScript, so you can debug apps running locally or remotely.

## Dedicated Watches

Track any number of expressions and variables in the current stack frame context. The Watches view is available through your debugging session.

## Remote debugging

As you connect to a remote host, you can use the mapping between the local source code and the remote debug processes can be launched.

## Depuración de Ruby on Rails rápidamente + Consejo para principiantes

**La depuración al generar excepciones** es mucho más fácil que entrecerrar los ojos a través de `print` declaraciones del registro de `print`, y para la mayoría de los errores, generalmente es mucho más rápido que abrir un depurador irb como `pry` o `byebug`. Esas herramientas no deberían ser tu primer paso.

---

# Depurando Ruby / Rails rápidamente:

## 1. Método rápido: `.inspect` una `Exception` e `.inspect` su resultado.

La forma más rápida de depurar el código de Ruby (especialmente Rails) es `raise` una excepción a lo largo de la ruta de ejecución de su código al llamar a `.inspect` en el método u objeto (por ejemplo, `foo`):

```
raise foo.inspect
```

En el código anterior, `raise` desencadena una `Exception` que *detiene la ejecución de su código*, y devuelve un mensaje de error que contiene convenientemente `.inspect` información sobre el objeto / método (es decir, `foo`) en la línea que usted está tratando de depurar.

Esta técnica es útil para examinar *rápidamente* un objeto o método ( *por ejemplo, ¿es `nil`?* ) Y para confirmar de inmediato si una línea de código se ejecuta incluso en un contexto determinado.

## 2. Fallback: use un depurador *IRB* de rubí como `byebug` o `pry`

Solo después de que tenga información sobre el estado del flujo de ejecución de sus códigos, debe considerar pasar a un depurador irb ruby gem como `pry` o `byebug` donde puede profundizar más en el estado de los objetos dentro de su ruta de ejecución.

Para usar la gema `byebug` para la depuración en Rails:

1. Agregue `gem 'byebug'` dentro del grupo de *desarrollo* en su *Gemfile*
2. Ejecutar `bundle install`
3. Luego, para usarlo, inserte la frase `byebug` dentro de la ruta de ejecución del código que desea examinar.

Esta variable `byebug` cuando se ejecute abrirá una sesión IRB de ruby de su código, brindándole acceso directo al estado de los objetos tal como están en ese punto en la ejecución del código.

Los depuradores de IRB como `Byebug` son útiles para analizar profundamente el estado de su código a medida que se ejecuta. Sin embargo, son un procedimiento que consume más tiempo en comparación con el aumento de errores, por lo que en la mayoría de las situaciones no deberían ser su primer paso.

---

# Consejos generales para principiantes

Cuando intenta depurar un problema, un buen consejo es siempre: **Lea el mensaje de error (@F) del @ @ \$ \$ ing**

Eso significa leer los mensajes de error con *cuidado* y por *completo* antes de actuar para que *entiendas lo que está tratando de decirte*. Cuando realice la depuración, haga las siguientes preguntas mentales, *en este orden*, al leer un mensaje de error:

1. ¿A qué **clase** hace referencia el error? (es decir, ¿tengo la clase de objeto correcta o mi objeto es `nil`?)
2. ¿Qué **método** hace referencia el error? (es decir, ¿es un tipo en el método? ¿Puedo llamar a este método en este tipo / clase de objeto?)
3. Finalmente, utilizando lo que puedo deducir de mis últimas dos preguntas, ¿qué **líneas de código** debo investigar? (recuerde: la última línea de código en el seguimiento de la pila no es necesariamente donde reside el problema).

En el seguimiento de la pila, preste especial atención a las líneas de código que provienen de su proyecto (por ejemplo, líneas que comienzan con la `app/...` si está usando Rails). El 99% de las veces el problema es con tu propio código.

---

Para ilustrar por qué la interpretación *en este orden* es importante ...

## Por ejemplo, un mensaje de error de Ruby que confunde a muchos principiantes:

Ejecuta código que en algún momento se ejecuta como tal:

```
@foo = Foo.new
...
@foo.bar
```

y obtienes un error que dice:

```
undefined method "bar" for Nil:nilClass
```

Los principiantes ven este error y piensan que el problema es que la `bar` método *no* está *definida*. **No es**. En este error la parte real que importa es:

```
for Nil:nilClass
```

**for Nil:nilClass significa que @foo es Nil!** `@foo` no es una variable de instancia de `Foo`! Tienes un objeto que es `Nil`. Cuando ve este error, es simplemente ruby tratando de decirle que la `bar` método no existe para los objetos de la clase `Nil`. (bueno duh! ya que estamos tratando de usar un método para un objeto de la clase `Foo` no `Nil`).

Desafortunadamente, debido a la forma en que se escribe este error ( `undefined method "bar" for Nil:nilClass` ) es fácil `undefined method "bar" for Nil:nilClass` pensar que este error tiene que ver con que la `bar` `undefined` esté `undefined` . Cuando no se lee con atención, este error hace que los principiantes se adentren por error en los detalles del método de `bar` en `Foo` , perdiendo por completo la parte del error que insinúa que el objeto es de la clase incorrecta (en este caso: `nil`). Es un error que se evita fácilmente leyendo los mensajes de error en su totalidad.

## Resumen:

Siempre **lea** atentamente **el mensaje de error completo** antes de comenzar cualquier depuración. Eso significa que: siempre verifique *primero* el tipo de **clase** de un objeto en un mensaje de error, luego sus **métodos** , *antes de* comenzar a buscar en cualquier seguimiento de pila o línea de código donde cree que puede estar ocurriendo el error. Esos 5 segundos pueden ahorrarle 5 horas de frustración.

**tl; dr:** No entrecerrar los ojos en los registros de impresión: en su lugar, generar excepciones. Evite los agujeros de conejo leyendo los errores cuidadosamente antes de depurar.

## Depuración de la aplicación ruby-on-rails con palanca

`pry` es una herramienta poderosa que se puede usar para depurar cualquier aplicación ruby. Configurar una aplicación de rubí sobre rieles con esta gema es muy fácil y directo.

### Preparar

Para comenzar a depurar tu aplicación con palanca

- Agregue `gem 'pry'` al `Gemfile` la aplicación y `Gemfile`

```
group :development, :test do
  gem 'pry'
end
```

- Navegue hasta el directorio raíz de la aplicación en la consola de terminal y ejecute la `bundle install` . Ya está todo listo para comenzar a usarlo en cualquier parte de su aplicación.

### Utilizar

El uso de la palanca en su aplicación solo incluye el `binding.pry` en los puntos de interrupción que desea inspeccionar mientras realiza la depuración. Puede agregar `binding.pry` interrupción `binding.pry` en cualquier lugar de su aplicación que sea interpretado por el intérprete de ruby (cualquier aplicación / controlador, aplicación / modelo, aplicación / archivos de vista)

i) Depuración de un controlador

`app / controllers / users_controller.rb`

```
class UsersController < ApplicationController
  def show
    use_id = params[:id]
  end
end
```

```

// breakpoint to inspect if the action is receiving param as expected
binding.pry
@user = User.find(user_id)
respond_to do |format|
  format.html
end
end
end
end

```

En este ejemplo, el servidor de rieles se detiene con una consola de palanca en el punto de quiebre cuando intenta visitar el enrutamiento de una página para `show` acción en `UsersController`. Puede inspeccionar el objeto `params` y hacer una consulta ActiveRecord en el modelo de `User` desde ese punto de interrupción

## ii) Depurar una vista

*app / views / users / show.html.haml*

```

%table
  %tbody
    %tr
      %td ID
      %td= @user.id
    %tr
      %td email
      %td= @user.email
    %tr
      %td logged in ?
      %td
        - binding.pry
        - if @user.logged_in?
          %p= "Logged in"
        - else
          %p= "Logged out"

```

En este ejemplo, el punto de ruptura se detiene en la consola de palanca cuando la página de `users/show` se compila previamente en el servidor de rieles antes de enviarla de nuevo al navegador del cliente. Este punto de ruptura permite depurar la corrección de `@user.logged_in?` cuando se está portando mal.

## ii) Depurar un modelo

```

app/models/user.rb

class User < ActiveRecord::Base
  def full_name
    binding.pry
    "#{self.first_name} #{self.last_name}"
  end
end
end

```

En este ejemplo, el punto de ruptura se puede usar para depurar el método de instancia del modelo de `User` `full_name` cuando se llama a este método desde cualquier lugar de la aplicación.

En conclusión, pry es una poderosa herramienta de depuración para la aplicación de rieles con una configuración sencilla y una guía de depuración directa. Prueba esto.

Lea **Depuración en línea**: <https://riptutorial.com/es/ruby-on-rails/topic/3877/depuracion>

---

# Capítulo 36: Despliegue de una aplicación Rails en Heroku

## Examples

### Desplegando su aplicación

Asegúrese de estar en el directorio que contiene su aplicación Rails, luego cree una aplicación en Heroku.

```
$ heroku create example
Creating ☐ example... done
https://example.herokuapp.com/ | https://git.heroku.com/example.git
```

La primera URL de la salida, <http://example.herokuapp.com>, es la ubicación en la que está disponible la aplicación. La segunda URL, `git@heroku.com: example.git`, es la URL del repositorio git remoto.

Este comando solo debe usarse en un repositorio git inicializado. El comando `heroku create` agrega automáticamente un control remoto de git llamado "heroku" que apunta a esta URL.

El argumento del nombre de la aplicación ("ejemplo") es opcional. Si no se especifica ningún nombre de aplicación, se generará un nombre aleatorio. Dado que los nombres de las aplicaciones Heroku están en un espacio de nombres global, puede esperar que ya se tomen nombres comunes, como "blog" o "wiki". A menudo es más fácil comenzar con un nombre predeterminado y cambiar el nombre de la aplicación más tarde.

A continuación, despliegue su código:

```
$ git push heroku master
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Ruby app detected
remote: -----> Compiling Ruby/Rails
remote: -----> Using Ruby version: ruby-2.3.1
remote: -----> Installing dependencies using bundler 1.11.2
remote:           Running: bundle install --without development:test --path vendor/bundle --
binstubs vendor/bundle/bin -j4 --deployment
remote:           Warning: the running version of Bundler is older than the version that created
the lockfile. We suggest you upgrade to the latest version of Bundler by running `gem install
bundler`.
remote:           Fetching gem metadata from https://rubygems.org/.....
remote:           Fetching version metadata from https://rubygems.org/...
remote:           Fetching dependency metadata from https://rubygems.org/..
remote:           Installing concurrent-ruby 1.0.2
remote:           Installing i18n 0.7.0
remote:           Installing rake 11.2.2
remote:           Installing minitest 5.9.0
remote:           Installing thread_safe 0.3.5
```

```
remote:      Installing builder 3.2.2
remote:      Installing mini_portile2 2.1.0
remote:      Installing erubis 2.7.0
remote:      Installing pkg-config 1.1.7
remote:      Installing rack 2.0.1
remote:      Installing nio4r 1.2.1 with native extensions
remote:      Installing websocket-extensions 0.1.2
remote:      Installing mime-types-data 3.2016.0521
remote:      Installing arel 7.0.0
remote:      Installing coffee-script-source 1.10.0
remote:      Installing execjs 2.7.0
remote:      Installing method_source 0.8.2
remote:      Installing thor 0.19.1
remote:      Installing multi_json 1.12.1
remote:      Installing puma 3.4.0 with native extensions
remote:      Installing pg 0.18.4 with native extensions
remote:      Using bundler 1.11.2
remote:      Installing sass 3.4.22
remote:      Installing tilt 2.0.5
remote:      Installing turbolinks-source 5.0.0
remote:      Installing tzinfo 1.2.2
remote:      Installing nokogiri 1.6.8 with native extensions
remote:      Installing rack-test 0.6.3
remote:      Installing sprockets 3.6.3
remote:      Installing websocket-driver 0.6.4 with native extensions
remote:      Installing mime-types 3.1
remote:      Installing coffee-script 2.4.1
remote:      Installing uglifier 3.0.0
remote:      Installing turbolinks 5.0.0
remote:      Installing activesupport 5.0.0
remote:      Installing mail 2.6.4
remote:      Installing globalid 0.3.6
remote:      Installing activemodel 5.0.0
remote:      Installing jbuilder 2.5.0
remote:      Installing activejob 5.0.0
remote:      Installing activerecord 5.0.0
remote:      Installing loofah 2.0.3
remote:      Installing rails-dom-testing 2.0.1
remote:      Installing rails-html-sanitizer 1.0.3
remote:      Installing actionview 5.0.0
remote:      Installing actionpack 5.0.0
remote:      Installing actionmailer 5.0.0
remote:      Installing railties 5.0.0
remote:      Installing actioncable 5.0.0
remote:      Installing sprockets-rails 3.1.1
remote:      Installing coffee-rails 4.2.1
remote:      Installing jquery-rails 4.1.1
remote:      Installing rails 5.0.0
remote:      Installing sass-rails 5.0.5
remote:      Bundle complete! 15 Gemfile dependencies, 54 gems now installed.
remote:      Gems in the groups development and test were not installed.
remote:      Bundled gems are installed into ./vendor/bundle.
remote:      Bundle completed (31.86s)
remote:      Cleaning up the bundler cache.
remote:      Warning: the running version of Bundler is older than the version that created
remote:      the lockfile. We suggest you upgrade to the latest version of Bundler by running `gem install
remote:      bundler`.
remote:      -----> Preparing app for Rails asset pipeline
remote:      Running: rake assets:precompile
remote:      I, [2016-07-08T17:08:57.046245 #1222] INFO -- : Writing
remote:      /tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
```

```

1bf5315c71171ad5f9cbef00193d56b7e45263ddc64caf676ce988cfbb6570bd.js
remote:      I, [2016-07-08T17:08:57.046951 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
1bf5315c71171ad5f9cbef00193d56b7e45263ddc64caf676ce988cfbb6570bd.js.gz
remote:      I, [2016-07-08T17:08:57.060208 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855.css
remote:      I, [2016-07-08T17:08:57.060656 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855.css.gz
remote:      Asset precompilation completed (4.06s)
remote:      Cleaning assets
remote:      Running: rake assets:clean
remote:
remote: ##### WARNING:
remote:      No Procfile detected, using the default web server.
remote:      We recommend explicitly declaring how to boot your server process via a
Procfile.
remote:      https://devcenter.heroku.com/articles/ruby-default-web-server
remote:
remote: -----> Discovering process types
remote:      Procfile declares types      -> (none)
remote:      Default types for buildpack -> console, rake, web, worker
remote:
remote: -----> Compressing...
remote:      Done: 29.2M
remote: -----> Launching...
remote:      Released v5
remote:      https://example.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/example.git
 * [new branch]      master -> master

```

Si está utilizando la base de datos en su aplicación, necesita migrar manualmente la base de datos ejecutando:

```
$ heroku run rake db:migrate
```

Cualquier comando después de la `heroku run` Heroku se ejecutará en un dino Heroku. Puede obtener una sesión de shell interactiva ejecutando:

```
$ heroku run bash
```

Asegúrese de tener un dinamómetro que ejecuta el tipo de proceso web:

```
$ heroku ps:scale web=1
```

El comando `heroku ps` enumera los días de ejecución de su aplicación:

```

$ heroku ps
=== web (Standard-1X): bin/rails server -p $PORT -e $RAILS_ENV (1)
web.1: starting 2016/07/08 12:09:06 -0500 (~ 2s ago)

```

Ahora puedes visitar la aplicación en nuestro navegador con `heroku open`.

```
$ heroku open
```

Heroku te da una URL web predeterminada en el dominio `herokuapp.com`. Cuando esté listo para escalar para la producción, puede agregar su propio dominio personalizado.

## Gestión de entornos de producción y puesta en escena para un Heroku.

Cada aplicación Heroku se ejecuta en al menos dos entornos: en Heroku (llamaremos a esa producción) y en su máquina local (desarrollo). Si más de una persona está trabajando en la aplicación, entonces tiene múltiples entornos de desarrollo, uno por máquina, generalmente. Por lo general, cada desarrollador también tendrá un entorno de prueba para ejecutar pruebas. Desafortunadamente, este enfoque se rompe a medida que los entornos se vuelven menos similares. Windows y Mac, por ejemplo, proporcionan entornos diferentes a la pila de Linux en Heroku, por lo que no siempre puede estar seguro de que el código que funciona en su entorno de desarrollo local funcionará de la misma manera cuando lo implemente en producción.

La solución es tener un entorno de prueba que sea tan similar a la producción como sea posible. Esto se puede lograr creando una segunda aplicación de Heroku que aloja su aplicación de estadificación. Con la puesta en escena, puede verificar su código en una configuración similar a la producción antes de que afecte a sus usuarios reales.

### Empezando desde cero

Suponga que tiene una aplicación ejecutándose en su máquina local y que está listo para enviarla a Heroku. Tendremos que crear tanto entornos remotos, puesta en escena y producción. Para adquirir el hábito de empujar a la puesta en escena primero, comenzaremos con esto:

```
$ heroku create --remote staging
Creating strong-river-216.... done
http://strong-river-216.herokuapp.com/ | https://git.heroku.com/strong-river-216.git
Git remote staging added
```

De forma predeterminada, la CLI de heroku crea proyectos con un control remoto de heroku git. Aquí, estamos especificando un nombre diferente con la marca `--remote`, por lo que presionar el código a Heroku y ejecutar los comandos contra la aplicación parece un poco diferente al `git push heroku master` normal:

```
$ git push staging master
...
$ heroku ps --remote staging
=== web: `bundle exec puma -C config/puma.rb`
web.1: up for 21s
```

Una vez que su aplicación de pruebas está funcionando correctamente, puede crear su aplicación de producción:

```
$ heroku create --remote production
Creating fierce-ice-327.... done
http://fierce-ice-327.herokuapp.com/ | https://git.heroku.com/fierce-ice-327.git
```

```
Git remote production added
$ git push production master
...
$ heroku ps --remote production
=== web: `bundle exec puma -C config/puma.rb
web.1: up for 16s
```

Y con eso, tiene el mismo código base que se ejecuta como dos aplicaciones Heroku separadas: una puesta en escena y una producción, configuradas de manera idéntica. Solo recuerde que tendrá que especificar qué aplicación va a operar en su trabajo diario. Puede usar la bandera '--remote' o usar git config para especificar una aplicación predeterminada.

Lea **Despliegue de una aplicación Rails en Heroku en línea**: <https://riptutorial.com/es/ruby-on-rails/topic/4485/despliegue-de-una-aplicacion-rails-en-heroku>

# Capítulo 37: Elasticsearch

## Examples

### Instalación y pruebas

Lo primero que desea hacer para el desarrollo local es instalar Elasticsearch en su máquina y probarlo para ver si se está ejecutando. Requiere que Java esté instalado. La instalación es bastante sencilla:

- **Mac OS X:** `brew install elasticsearch`
- **Ubuntu:** `sudo apt-get install elasticsearch`

Entonces empieza:

- **Mac OS X:** `brew services start elasticsearch`
- **Ubuntu:** `sudo service elasticsearch start`

Para probarlo, la forma más fácil es con `curl`. Puede tardar unos segundos en comenzar, así que no se asuste si no recibe ninguna respuesta al principio.

```
curl localhost:9200
```

Ejemplo de respuesta:

```
{
  "name" : "Hydro-Man",
  "cluster_name" : "elasticsearch_gkbonetti",
  "version" : {
    "number" : "2.3.5",
    "build_hash" : "90f439ff60a3c0f497f91663701e64ccd01edbb4",
    "build_timestamp" : "2016-07-27T10:36:52Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```

### Configuración de herramientas para el desarrollo.

Cuando esté comenzando con Elasticsearch (ES), puede ser bueno tener una herramienta gráfica que lo ayude a explorar sus datos. Un plugin llamado `elasticsearch-head` hace justamente eso. Para instalarlo, haga lo siguiente:

- **Averigüe en qué carpeta está instalada ES:** `ls -l $(which elasticsearch)`
- **cd en esta carpeta y ejecute el binario de instalación del complemento:**  
`elasticsearch/bin/plugin -install mobz/elasticsearch-head`
- **Abra `http://localhost:9200/_plugin/head/` en su navegador**

Si todo funcionó como se esperaba, debería ver una buena interfaz gráfica de usuario donde

puede explorar sus datos.

## Introducción

ElasticSearch tiene una API JSON bien documentada, pero probablemente querrá usar algunas bibliotecas que lo manejen por usted:

- [Elasticsearch](#) - el envoltorio oficial de bajo nivel para la API HTTP
- [Elasticsearch-rails](#) : la integración oficial de alto nivel de Rails que le ayuda a conectar sus modelos de Rails con ElasticSearch utilizando ActiveRecord o Repository pattern.
- [Chewy](#) : una integración de Rails de alto nivel alternativa y no oficial que es muy popular y posiblemente tiene mejor documentación

Usemos la primera opción para probar la conexión:

```
gem install elasticsearch
```

Luego enciende la terminal de rubíes y pruébalo:

```
require 'elasticsearch'

client = Elasticsearch::Client.new log: true
# by default it connects to http://localhost:9200

client.transport.reload_connections!
client.cluster.health

client.search q: 'test'
```

## Searchkick

Si quieres configurar rápidamente elasticsearch puedes usar la gema de búsqueda:

```
gem 'searchkick'
```

Añadir searchkick a los modelos que desea buscar.

```
class Product < ActiveRecord::Base
  searchkick
end
```

Añadir datos al índice de búsqueda.

```
Product.reindex
```

Y para consultar, usar:

```
products = Product.search "apples"
products.each do |product|
```

```
puts product.name  
end
```

Bastante rápido, no se requiere conocimiento de elasticsearch ;-)

Más información aquí: <https://github.com/ankane/searchkick>

Lea Elasticsearch en línea: <https://riptutorial.com/es/ruby-on-rails/topic/6500/elasticsearch>

# Capítulo 38: Enrutamiento

## Introducción

El enrutador Rails reconoce las URL y las envía a la acción de un controlador. También puede generar rutas y direcciones URL, evitando la necesidad de cadenas de código en sus vistas.

## Observaciones

En general, "enrutamiento" es cómo las URL son "manejadas" por su aplicación. En el caso de Rails, generalmente es qué controlador y qué acción de ese controlador manejará una URL entrante en particular. En las aplicaciones de Rails, las rutas generalmente se ubican en el archivo `config/routes.rb`.

## Examples

### Enrutamiento de recursos (básico)

Las rutas se definen en `config/routes.rb`. A menudo se definen como un grupo de rutas relacionadas, utilizando los `resources` o `resource` métodos de `resource`.

`resources :users` crean las siguientes siete rutas, todas `UserController` a las acciones de `UserController`:

```
get      '/users',      to: 'users#index'
post     '/users',      to: 'users#create'
get      '/users/new',   to: 'users#new'
get      '/users/:id/edit', to: 'users#edit'
get      '/users/:id',   to: 'users#show'
patch/put '/users/:id',   to: 'users#update'
delete   '/users/:id',   to: 'users#destroy'
```

Nombres de las acciones se muestran después de la # en el `to` parámetro anterior. Los métodos con esos mismos nombres se deben definir en `app/controllers/users_controller.rb` siguiente manera:

```
class UsersController < ApplicationController
  def index
  end

  def create
  end

  # continue with all the other methods...
end
```

Puede limitar las acciones que se generan con `only` o `except`:

```
resources :users, only: [:show]
resources :users, except: [:show, :index]
```

Puede ver todas las rutas de su aplicación en cualquier momento ejecutando:

5.0

```
$ rake routes
```

5.0

```
$ rake routes
# OR
$ rails routes
```

users	GET	/users(.:format)	users#index
	POST	/users(.:format)	users#create
new_user	GET	/users/new(.:format)	users#new
edit_user	GET	/users/:id/edit(.:format)	users#edit
user	GET	/users/:id(.:format)	users#show
	PATCH	/users/:id(.:format)	users#update
	PUT	/users/:id(.:format)	users#update
	DELETE	/users/:id(.:format)	users#destroy

Para ver solo las rutas que se asignan a un controlador en particular:

5.0

```
$ rake routes -c static_pages
static_pages_home GET /static_pages/home(.:format) static_pages#home
static_pages_help GET /static_pages/help(.:format) static_pages#help
```

5.0

```
$ rake routes -c static_pages
static_pages_home GET /static_pages/home(.:format) static_pages#home
static_pages_help GET /static_pages/help(.:format) static_pages#help

# OR

$ rails routes -c static_pages
static_pages_home GET /static_pages/home(.:format) static_pages#home
static_pages_help GET /static_pages/help(.:format) static_pages#help
```

Puedes buscar a través de rutas usando la opción `-g`. Esto muestra cualquier ruta que coincida parcialmente con el nombre del método auxiliar, la ruta de la URL o el verbo HTTP:

5.0

```
$ rake routes -g new_user # Matches helper method
$ rake routes -g POST # Matches HTTP Verb POST
```

5.0

```

$ rake routes -g new_user      # Matches helper method
$ rake routes -g POST          # Matches HTTP Verb POST
# OR
$ rails routes -g new_user     # Matches helper method
$ rails routes -g POST         # Matches HTTP Verb POST

```

Además, al ejecutar el servidor de `rails` en modo de desarrollo, puede acceder a una página web que muestra todas sus rutas con un filtro de búsqueda, con prioridad de arriba a abajo, en `<hostname>/rails/info/routes` . Se verá así:

Ayudante	Verbo HTTP	Camino	Controlador # Acción
Camino / Url		[Match Path]	
ruta_usuario	OBTENER	/users(.:format)	índice de usuarios #
	ENVIAR	/users(.:format)	usuarios # crear
new_user_path	OBTENER	/users/new(.:format)	usuarios # nuevo
edit_user_path	OBTENER	/users/:id/edit(.:format)	usuarios # editar
ruta_usuario	OBTENER	/users/:id(.:format)	usuarios # show
	PARCHE	/users/:id(.:format)	usuarios # actualizar
	PONER	/users/:id(.:format)	usuarios # actualizar
	BORRAR	/users/:id(.:format)	usuarios # destruir

Las rutas se pueden declarar disponibles solo para miembros (no para colecciones) utilizando el `resource` del método en lugar de los `resources` en `routes.rb` . Con `resource` , una ruta de `index` no se crea de forma predeterminada, sino solo cuando se solicita explícitamente una como esta:

```
resource :orders, only: [:index, :create, :show]
```

## Restricciones

Puede filtrar qué rutas están disponibles usando restricciones.

Hay varias formas de usar restricciones, incluyendo:

- [restricciones de segmento](#) ,
- [restricciones basadas en solicitudes](#)
- [restricciones avanzadas](#)

Por ejemplo, una restricción basada solicitada para permitir solo una dirección IP específica para acceder a una ruta:

```
constraints(ip: /127\.0\.0\.1$/) do
  get 'route', to: "controller#action"
end
```

[Vea otros ejemplos similares ActionDispatch :: Routing :: Mapper :: Scoping](#) .

Si desea hacer algo más complejo, puede usar restricciones más avanzadas y crear una clase para envolver la lógica:

```
# lib/api_version_constraint.rb
class ApiVersionConstraint
  def initialize(version:, default:)
    @version = version
    @default = default
  end

  def version_header
    "application/vnd.my-app.v#{@version}"
  end

  def matches?(request)
    @default || request.headers["Accept"].include?(version_header)
  end
end

# config/routes.rb
require "api_version_constraint"

Rails.application.routes.draw do
  namespace :v1, constraints: ApiVersionConstraint.new(version: 1, default: true) do
    resources :users # Will route to app/controllers/v1/users_controller.rb
  end

  namespace :v2, constraints: ApiVersionConstraint.new(version: 2) do
    resources :users # Will route to app/controllers/v2/users_controller.rb
  end
end
```

## Una forma, varios botones de enviar

También puede utilizar el valor de las etiquetas de envío de un formulario como una restricción para enrutar a una acción diferente. Si tiene un formulario con varios botones de envío (por ejemplo, "vista previa" y "envío"), podría capturar esta restricción directamente en su `routes.rb`, en lugar de escribir javascript para cambiar la URL de destino del formulario. Por ejemplo, con la gema [commit\\_param\\_routing](#) puede aprovechar las ventajas de rails `submit_tag`

`submit_tag` primer parámetro de Rails `submit_tag` permite cambiar el valor de su parámetro de confirmación de formulario

```
# app/views/orders/mass_order.html.erb
<%= form_for(@orders, url: mass_create_order_path do |f| %>
  <!-- Big form here -->
  <%= submit_tag "Preview" %>
  <%= submit_tag "Submit" %>
  # => <input name="commit" type="submit" value="Preview" />
  # => <input name="commit" type="submit" value="Submit" />
```

```

...
<% end %>

# config/routes.rb
resources :orders do
  # Both routes below describe the same POST URL, but route to different actions
  post 'mass_order', on: :collection, as: 'mass_order',
    constraints: CommitParamRouting.new('Submit'), action: 'mass_create' # when the user
  presses "submit"
  post 'mass_order', on: :collection,
    constraints: CommitParamRouting.new('Preview'), action: 'mass_create_preview' # when the
  user presses "preview"
  # Note the `as:` is defined only once, since the path helper is mass_create_order_path for
  the form url
  # CommitParamRouting is just a class like ApiVersionConstraint
end

```

## Rutas de alcance

Rails te ofrece varias formas de organizar tus rutas.

### Alcance por URL :

```

scope 'admin' do
  get 'dashboard', to: 'administration#dashboard'
  resources 'employees'
end

```

Esto genera las siguientes rutas.

```

get      '/admin/dashboard',      to: 'administration#dashboard'
post     '/admin/employees',   to: 'employees#create'
get      '/admin/employees/new', to: 'employees#new'
get      '/admin/employees/:id/edit', to: 'employees#edit'
get      '/admin/employees/:id', to: 'employees#show'
patch/put '/admin/employees/:id', to: 'employees#update'
delete   '/admin/employees/:id', to: 'employees#destroy'

```

Puede tener más sentido, en el lado del servidor, mantener algunas vistas en una subcarpeta diferente, para separar las vistas de administrador de las vistas de los usuarios.

### Alcance por módulo

```

scope module: :admin do
  get 'dashboard', to: 'administration#dashboard'
end

```

`module` busca los archivos del controlador en la subcarpeta del nombre dado

```

get      '/dashboard',      to: 'admin/administration#dashboard'

```

Puede cambiar el nombre del prefijo de los ayudantes de ruta agregando un parámetro `as`

```
scope 'admin', as: :administration do
  get 'dashboard'
end

# => administration_dashboard_path
```

Rails proporciona una manera conveniente de hacer todo lo anterior, utilizando el método de `namespace` . Las siguientes declaraciones son equivalentes.

```
namespace :admin do
end

scope 'admin', module: :admin, as: :admin
```

## Alcance por controlador

```
scope controller: :management do
  get 'dashboard'
  get 'performance'
end
```

Esto genera estas rutas.

```
get    '/dashboard',      to: 'management#dashboard'
get    '/performance',   to: 'management#performance'
```

## Anidación poco profunda

Las rutas de recursos aceptan una opción `:shallow` que ayuda a acortar las URL cuando sea posible. Los recursos no deben estar anidados a más de un nivel de profundidad. Una forma de evitar esto es mediante la creación de rutas poco profundas. El objetivo es dejar de lado los segmentos de URL de colección principal donde no son necesarios. El resultado final es que las únicas rutas anidadas generadas son para `:index` `:create` y `:new` . El resto se mantienen en su propio contexto de URL superficial. Hay dos opciones de alcance para personalizar rutas poco profundas:

- **`:shallow_path`**: Prefiere las rutas de los miembros con un parámetro específico

```
scope shallow_path: "sekret" do
  resources :articles do
    resources :comments, shallow: true
  end
end
```

- **`:shallow_prefix`** : Agrega parámetros especificados a los ayudantes nombrados

```
scope shallow_prefix: "sekret" do
  resources :articles do
    resources :comments, shallow: true
  end
end
```

También podemos ilustrar rutas `shallow` más por:

```
resources :auctions, shallow: true do
  resources :bids do
    resources :comments
  end
end
```

codificado alternativamente de la siguiente manera (si está satisfecho con el bloque):

```
resources :auctions do
  shallow do
    resources :bids do
      resources :comments
    end
  end
end
```

Las rutas resultantes son:

Prefijo	Verbo	Patrón de URI
bid_comments	OBTENER	/bids/:bid_id/comments(.:format)
	ENVIAR	/bids/:bid_id/comments(.:format)
new_bid_comment	OBTENER	/bids/:bid_id/comments/new(.:format)
Editar comentario	OBTENER	/comments/:id/edit(.:format)
comentario	OBTENER	/comments/:id(.:format)
	PARCHE	/comments/:id(.:format)
	PONER	/comments/:id(.:format)
	BORRAR	/comments/:id(.:format)
remate de ofertas	OBTENER	/auctions/:auction_id/bids(.:format)
	ENVIAR	/auctions/:auction_id/bids(.:format)
new_auction_bid	OBTENER	/auctions/:auction_id/bids/new(.:format)
edit_bid	OBTENER	/bids/:id/edit(.:format)
oferta	OBTENER	/bids/:id(.:format)
	PARCHE	/bids/:id(.:format)
	PONER	/bids/:id(.:format)

Prefijo	Verbo	Patrón de URI
	BORRAR	/bids/:id(.:format)
subastas	OBTENER	/subastas(.formato)
	ENVIAR	/subastas(.formato)
nueva_aucción	OBTENER	/auctions/new(.:format)
edición_aucción	OBTENER	/auctions/:id/edit(.:format)
subasta	OBTENER	/auctions/:id(.:format)
	PARCHE	/auctions/:id(.:format)
	PONER	/auctions/:id(.:format)
	BORRAR	/auctions/:id(.:format)

Si analiza las rutas generadas con cuidado, notará que las partes anidadas de la URL solo se incluyen cuando son necesarias para determinar qué datos mostrar.

## Preocupaciones

Para evitar la repetición en rutas anidadas, las preocupaciones proporcionan una excelente manera de compartir recursos comunes que son reutilizables. Para crear una preocupación utilice el método de `concern` dentro de la `routes.rb` archivo. El método espera un símbolo y un bloque:

```
concern :commentable do
  resources :comments
end
```

Aunque no crea ninguna ruta en sí, este código permite usar el atributo `:concerns` en un recurso. El ejemplo más simple sería:

```
resource :page, concerns: :commentable
```

El recurso anidado equivalente se vería así:

```
resource :page do
  resource :comments
end
```

Esto construiría, por ejemplo, las siguientes rutas:

```
/pages/#{page_id}/comments
/pages/#{page_id}/comments/#{comment_id}
```

Para que las preocupaciones sean significativas, debe haber múltiples recursos que utilicen la

preocupación. Los recursos adicionales podrían usar cualquiera de la siguiente sintaxis para llamar a la inquietud:

```
resource :post, concerns: %i(commentable)
resource :blog do
  concerns :commentable
end
```

## Redirección

Puede realizar la redirección en las rutas de Rails de la siguiente manera:

4.0

```
get '/stories', to: redirect('/posts')
```

4.0

```
match "/abc" => redirect("http://example.com/abc")
```

También puede redirigir todas las rutas desconocidas a una ruta determinada:

4.0

```
match '*path' => redirect('/'), via: :get
# or
get '*path' => redirect('/')
```

4.0

```
match '*path' => redirect('/')
```

## Miembro y rutas de recogida

La definición de un bloque miembro dentro de un recurso crea una ruta que puede actuar sobre un miembro individual de esa ruta basada en recursos:

```
resources :posts do
  member do
    get 'preview'
  end
end
```

Esto genera la siguiente ruta de miembro:

```
get '/posts/:id/preview', to: 'posts#preview'
# preview_post_path
```

Las rutas de recolección permiten crear rutas que pueden actuar sobre una colección de objetos de recursos:

```
resources :posts do
  collection do
    get 'search'
  end
end
```

Esto genera la siguiente ruta de recolección:

```
get '/posts/search', to: 'posts#search'
# search_posts_path
```

Una sintaxis alternativa:

```
resources :posts do
  get 'preview', on: :member
  get 'search', on: :collection
end
```

## URLs params con un punto

Si desea admitir un parámetro url más complejo que un número de identificación, puede tener problemas con el analizador si el valor contiene un punto. Se asumirá que todo lo que sigue a un período es un formato (es decir, json, xml).

Puede solucionar esta limitación utilizando una restricción para *ampliar* la entrada aceptada.

Por ejemplo, si desea hacer referencia a un registro de usuario por dirección de correo electrónico en la url:

```
resources :users, constraints: { id: /.*/ }
```

## Ruta de la raíz

Puede agregar una ruta de página de inicio a su aplicación con el método `root` .

```
# config/routes.rb
Rails.application.routes.draw do
  root "application#index"
  # equivalent to:
  # get "/", "application#index"
end

# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  def index
    render "homepage"
  end
end
```

Y en la terminal, las `rake routes` ( `rails routes` en Rails 5) producirán:

```
root    GET    /    application#index
```

Debido a que la página principal suele ser la ruta más importante, y las rutas se priorizan en el orden en que aparecen, la ruta `root` debería ser la primera en su archivo de rutas.

## Acciones adicionales de REST

```
resources :photos do
  member do
    get 'preview'
  end
  collection do
    get 'dashboard'
  end
end
```

Esto crea las siguientes rutas **además de las 7 rutas RESTful predeterminadas** :

```
get    '/photos/:id/preview',    to: 'photos#preview'
get    '/photos/dashboards', to: 'photos#dashboard'
```

Si desea hacer esto para líneas simples, puede usar:

```
resources :photos do
  get 'preview', on: :member
  get 'dashboard', on: :collection
end
```

También puede agregar una acción a la `/new` ruta:

```
resources :photos do
  get 'preview', on: :new
end
```

Que creará:

```
get    '/photos/new/preview',    to: 'photos#preview'
```

Tenga en cuenta al agregar acciones a sus rutas RESTful, ¡es probable que le falte otro recurso!

## Ámbito local disponible

Si su aplicación está disponible en diferentes idiomas, normalmente muestra la configuración regional actual en la URL.

```
scope '(/:locale)', locale: /#{I18n.available_locales.join('|')}/ do
  root 'example#root'
  # other routes
end
```

Se podrá acceder a su raíz a través de las configuraciones regionales definidas en

```
I18n.available_locales .
```

## Montar otra aplicación

el montaje se utiliza para montar otra aplicación (básicamente, la aplicación en bastidor) o los rieles de los motores que se utilizarán dentro de la aplicación actual

### sintaxis:

```
mount SomeRackApp, at: "some_route"
```

Ahora puede acceder a la aplicación montada anteriormente usando el ayudante de ruta

```
some_rack_app_path o some_rack_app_url .
```

Pero si desea cambiar el nombre de este ayudante, puede hacerlo como:

```
mount SomeRackApp, at: "some_route", as: :myapp
```

Esto generará los ayudantes `myapp_path` y `myapp_url` que se pueden usar para navegar a esta aplicación montada.

## Redirecciones y rutas de comodines

Si desea proporcionar una URL por conveniencia para su usuario pero asignarla directamente a otra que ya esté usando. Utilice una redirección:

```
# config/routes.rb
TestApp::Application.routes.draw do
  get 'courses/:course_name' => redirect('/courses/{course_name}/lessons'), :as => "course"
end
```

Bueno, eso se puso interesante rápido. El principio básico aquí es simplemente usar el método `#redirect` para enviar una ruta a otra. Si su ruta es bastante simple, es un método muy sencillo. Pero si también desea enviar los parámetros originales, necesita hacer un poco de gimnasia capturando el parámetro dentro de `{here}`. Tenga en cuenta las comillas simples alrededor de todo.

En el ejemplo anterior, también hemos cambiado el nombre de la ruta por conveniencia utilizando un alias con el: como parámetro. Esto nos permite usar ese nombre en métodos como los ayudantes `#_path`. De nuevo, prueba tus `$ rake routes` con preguntas.

## Dividir rutas en múltiples archivos

Si su archivo de rutas es abrumadoramente grande, puede poner sus rutas en múltiples archivos e incluir cada uno de los archivos con el método `require_relative` de Ruby:

```
config/routes.rb :
```

```
YourAppName::Application.routes.draw do
  require_relative 'routes/admin_routes'
  require_relative 'routes/sidekiq_routes'
  require_relative 'routes/api_routes'
  require_relative 'routes/your_app_routes'
end
```

config/routes/api\_routes.rb :

```
YourAppName::Application.routes.draw do
  namespace :api do
    # ...
  end
end
```

## Rutas anidadas

Si desea agregar rutas anidadas, puede escribir el siguiente código en el archivo `routes.rb` .

```
resources :admins do
  resources :employees
end
```

Esto generará las siguientes rutas:

admin_employees	GET	/admins/:admin_id/employees(.:format)	employees#index
	POST	/admins/:admin_id/employees(.:format)	
employees#create			
new_admin_employee	GET	/admins/:admin_id/employees/new(.:format)	employees#new
edit_admin_employee	GET	/admins/:admin_id/employees/:id/edit(.:format)	employees#edit
admin_employee	GET	/admins/:admin_id/employees/:id(.:format)	employees#show
	PATCH	/admins/:admin_id/employees/:id(.:format)	
employees#update			
	PUT	/admins/:admin_id/employees/:id(.:format)	
employees#update			
	DELETE	/admins/:admin_id/employees/:id(.:format)	
employees#destroy			
admins	GET	/admins(.:format)	admins#index
	POST	/admins(.:format)	admins#create
new_admin	GET	/admins/new(.:format)	admins#new
edit_admin	GET	/admins/:id/edit(.:format)	admins#edit
admin	GET	/admins/:id(.:format)	admins#show
	PATCH	/admins/:id(.:format)	admins#update
	PUT	/admins/:id(.:format)	admins#update
	DELETE	/admins/:id(.:format)	admins#destroy

Lea Enrutamiento en línea: <https://riptutorial.com/es/ruby-on-rails/topic/307/enrutamiento>

---

# Capítulo 39: Enrutamiento superficial

## Examples

### 1. Uso de poca profundidad.

Una forma de evitar el anidamiento profundo (como se recomienda anteriormente) es generar las acciones de recopilación dentro del ámbito primario, para tener una idea de la jerarquía, pero no anidar las acciones de los miembros. En otras palabras, solo construir rutas con la cantidad mínima de información para identificar de forma única el recurso, como esto:

```
resources :articles, shallow: true do
  resources :comments
  resources :quotes
  resources :drafts
end
```

El método superficial del DSL crea un ámbito dentro del cual cada anidación es superficial. Esto genera las mismas rutas que el ejemplo anterior:

```
shallow do
  resources :articles do
    resources :comments
    resources :quotes
    resources :drafts
  end
end
```

Existen dos opciones de alcance para personalizar rutas poco profundas. : shallow\_path prefija las rutas de los miembros con el parámetro especificado:

```
scope shallow_path: "sekret" do
  resources :articles do
    resources :comments, shallow: true
  end
end
```

Utilice el comando Rake para obtener las rutas generadas como se define a continuación:

```
rake routes
```

Lea Enrutamiento superficial en línea: <https://riptutorial.com/es/ruby-on-rails/topic/7775/enrutamiento-superficial>

# Capítulo 40: Estados del modelo: AASM

## Examples

### Estado básico con AASM

Por lo general, terminará creando modelos que contendrán un estado, y ese estado cambiará durante la vida útil del objeto.

[AASM](#) es una biblioteca de habilitación de máquina de estado finito que puede ayudarlo a lidiar con tener un paso fácil a través del proceso de diseño de sus objetos.

Tener algo como esto en su modelo va bastante alineado con la idea de [Fat Model](#), [Skinny Controller](#), una de las mejores prácticas de Rails. El modelo es el único responsable de gestionar su estado, sus cambios y de generar los eventos desencadenados por esos cambios.

Para instalar, en Gemfile

```
gem 'aasm'
```

Considere una aplicación donde el usuario cite un producto por un precio.

```
class Quote

  include AASM

  aasm do
    state :requested, initial: true # User sees a product and requests a quote
    state :priced # Seller sets the price
    state :payed # Buyer pays the price
    state :canceled # The buyer is not willing to pay the price
    state :completed # The product has been delivered.

    event :price do
      transitions from: :requested, to: :priced
    end

    event :pay do
      transitions from: :priced, to: :payed, success: :set_payment_date
    end

    event :complete do
      transitions from: :payed, to: :completed, guard: product_delivered?
    end

    event :cancel do
      transitions from: [:requested, :priced], to: :canceled
      transitions from: :payed, to: :canceled, success: :reverse_charges
    end

  end

end
```

```
private

def set_payment_date
  update payed_at: Time.zone.now
end
end
```

Los estados de la clase de cotización pueden irse sin embargo es mejor para su proceso.

Puede pensar que los estados son pasados, como en el ejemplo anterior o algo en otro tiempo, por ejemplo: precios, pagos, entregas, etc. El nombre de los estados depende de usted. Desde un punto de vista personal, los estados pasados funcionan mejor porque su estado final seguramente será una acción pasada y se enlaza mejor con los nombres de los eventos, que se explicarán más adelante.

**NOTA:** tenga cuidado con los nombres que utiliza, debe preocuparse por no usar las palabras clave reservadas de Ruby o Ruby on Rails, como `valid`, `end`, `being`, etc.

Habiendo definido los estados y las transiciones, ahora podemos acceder a algunos métodos creados por AASM.

Por ejemplo:

```
Quote.priced # Shows all Quotes with priced events
quote.priced? # Indicates if that specific quote has been priced
quote.price! # Triggers the event the would transition from requested to priced.
```

Como puede ver, el evento tiene transiciones, estas transiciones determinan la forma en que cambiará el estado en la llamada del evento. Si el evento no es válido debido al estado actual, se generará un error.

Los eventos y transiciones también tienen algunas otras devoluciones de llamada, por ejemplo

```
guard: product_delivered?
```

Llamará al `product_delivered?` Método que devolverá un booleano. Si resulta falso, la transición no se aplicará y si no hay otras transiciones disponibles, el estado no cambiará.

```
success: :reverse_charges
```

Si esa traducción se realiza correctamente, se `:reverse_charges` método `:reverse_charges`.

Hay varios otros métodos en AASM con más devoluciones de llamada en el proceso, pero esto te ayudará a crear tus primeros modelos con estados finitos.

Lea Estados del modelo: AASM en línea: <https://riptutorial.com/es/ruby-on-rails/topic/7826/estados-del-modelo--aasm>

---

# Capítulo 41: Estructuras de rieles a lo largo de los años.

## Introducción

Cuando es nuevo en Rails y trabaja en aplicaciones de Rails heredadas, puede ser confuso entender qué marco de trabajo se introdujo cuando. Este tema está diseñado para ser la lista *definitiva* de todos los marcos en las versiones de Rails.

## Examples

¿Cómo encontrar qué marcos están disponibles en la versión actual de Rails?

Utilizar el

```
config.frameworks
```

Opción para obtener una matriz de `Symbol` que representan cada marco.

### Versiones de rieles en rieles 1.x

- ActionMailer
- Paquete de acción
- ActionWebService
- ActiveRecord
- ActiveSupport
- Railties

### Estructuras de rieles en rieles 2.x

- ActionMailer
- Paquete de acción
- ActiveRecord
- ActiveResource ( *ActiveWebService* fue reemplazado por *ActiveResource*, y con eso, *Rails* se movió de SOAP a REST de forma predeterminada )
- ActiveSupport
- Railties

### Estructuras de rieles en rieles 3.x

- ActionMailer
- Paquete de acción
- ActiveModel
- ActiveRecord

- ActiveRecord
- ActiveSupport
- Railties

Lea Estructuras de rieles a lo largo de los años. en línea: <https://riptutorial.com/es/ruby-on-rails/topic/8107/estructuras-de-rieles-a-lo-largo-de-los-anos->

# Capítulo 42: Forma anidada en Ruby on Rails

## Examples

### Cómo configurar un formulario anidado en Ruby on Rails

Lo primero que hay que tener: un modelo que contiene una relación `has_many` con otro modelo.

```
class Project < ApplicationRecord
  has_many :todos
end

class Todo < ApplicationRecord
  belongs_to :project
end
```

En `ProjectsController`:

```
class ProjectsController < ApplicationController
  def new
    @project = Project.new
  end
end
```

En una forma anidada, puede crear objetos secundarios con un objeto principal al mismo tiempo.

```
<%= nested_form_for @project do |f| %>
  <%= f.label :name %>
  <%= f.text_field :name %>

  <% # Now comes the part for `Todo` object %>
  <%= f.fields_for :todo do |todo_field| %>
    <%= todo_field.label :name %>
    <%= todo_field.text_field :name %>
  <% end %>
<% end %>
```

Cuando iniciamos `@project` con `Project.new` para tener algo para crear un nuevo objeto `Project`, de la misma manera para crear un objeto `Todo`, tenemos que tener algo como esto, y hay varias formas de hacerlo:

1. En `Projectscontroller`, en el `new` método, puede escribir: `@todo = @project.todos.build` o `@todo = @project.todos.new` para crear una instancia de un nuevo objeto `Todo`.
2. También puede hacer esto en la vista: `<%= f.fields_for :todos, @project.todos.build %>`

Para parámetros fuertes, puede incluirlos de la siguiente manera:

```
def project_params
  params.require(:project).permit(:name, todo_attributes: [:name])
end
```

```
end
```

Dado que, los objetos de `Todo` se crearán a través de la creación de un objeto de `Project`, por lo que debe especificar esto en el modelo de `Project` agregando la siguiente línea:

```
accepts_nested_attributes_for :todos
```

Lea **Forma anidada en Ruby on Rails en línea**: <https://riptutorial.com/es/ruby-on-rails/topic/8203/forma-anidada-en-ruby-on-rails>

---

# Capítulo 43: Gemas

## Observaciones

### Documentación de Gemfile

Para los proyectos que se espera que crezcan, es una buena idea agregar comentarios a su `Gemfile`. De esa manera, incluso en configuraciones grandes, aún sabrás qué hace cada gema, incluso si el nombre no se explica por sí mismo y lo agregaste hace 2 años.

Esto también puede ayudarlo a recordar por qué eligió una determinada versión y, por lo tanto, volver a evaluar el requisito de la versión más adelante.

Ejemplos:

```
# temporary downgrade for TeamCity
gem 'rake', '~> 10.5.0'
# To upload invoicing information to payment provider
gem 'net-sftp'
```

## Examples

### ¿Qué es una gema?

Una gema es el equivalente a un complemento o una extensión para el lenguaje de programación ruby.

Para ser exactos, los rieles no son más que una gema. Muchas gemas se construyen sobre rieles u otras gemas (dependen de dicha gema) o son independientes.

---

## En tu proyecto Rails

### Gemfile

Para tu proyecto de Rails tienes un archivo llamado `Gemfile`. Aquí puede agregar gemas que desee incluir y usar en su proyecto. Una vez agregado, debe instalar la gema utilizando `bundler` (vea la sección `bundler`).

### Gemfile.lock

Una vez que hayas hecho esto, tu `Gemfile.lock` se actualizará con tus gemas recién agregadas y sus dependencias. Este archivo bloquea las gemas usadas para que usen esa versión específica declarada en ese archivo.

```
GEM
remote: https://rubygems.org/
specs:
devise (4.0.3)
bcrypt (~> 3.0)
orm_adapter (~> 0.1)
railties (>= 4.1.0, < 5.1)
responders
warden (~> 1.2.3)
```

Este ejemplo es para el `devise` gema. En el `Gemfile.lock` la versión `4.0.3`, para indicar cuándo se instala su proyecto en otra máquina o en su servidor de producción qué versión específica se debe usar.

## Desarrollo

Una sola persona, un grupo o toda una comunidad trabaja y mantiene una gema. El trabajo realizado generalmente se libera después de `issues` se hayan solucionado ciertos `issues` o se hayan agregado `features`.

Por lo general, los lanzamientos siguen el principio de [versión semántica 2.0.0](#).

## Bundler

La forma más fácil de manejar y administrar gemas es mediante el uso de `bundler`. [Bundler](#) es un gestor de paquetes comparable a `bower`.

Para usar `bundler` primero necesitas instalarlo.

```
gem install bundler
```

Después de tener el empaquetador en funcionamiento, todo lo que necesita hacer es agregar gemas a su `Gemfile` y ejecutar

```
bundle
```

en tu terminal Esto instala sus gemas recién agregadas a su proyecto. Si surgiera un problema, obtendría un aviso en su terminal.

Si está interesado en obtener más detalles, le sugiero que eche un vistazo a los [documentos](#).

## Gemfiles

Para comenzar, `gemfiles` requiere al menos una fuente, en la forma de la URL para un servidor RubyGems.

Genere un `Gemfile` con la fuente predeterminada `rubygems.org` ejecutando `bundle init`. Use `https` para que su conexión al servidor se verifique con SSL.

```
source 'https://rubygems.org'
```

A continuación, declare las gemas que necesita, incluidos los números de versión.

```
gem 'rails', '4.2.6'  
gem 'rack', '>=1.1'  
gem 'puma', '~>3.0'
```

La mayoría de los especificadores de versión, como `> = 1.0`, se explican por sí mismos. El especificador `~>` tiene un significado especial. `~> 2.0.3` es idéntico a `> = 2.0.3` y `<2.1`. `~> 2.1` es idéntico a `> = 2.1` y `<3.0`. `~> 2.2.beta` coincidirá con versiones preliminares como `2.2.beta.12`.

Los repositorios de Git también son fuentes de gemas válidas, siempre que el repositorio contenga una o más gemas válidas. Especifique qué comprobar con `:tag`, `:branch`, o `:ref`. El valor predeterminado es la rama `master`.

```
gem 'nokogiri', :git => 'https://github.com/sparklemotion/nokogiri', :branch => 'master'
```

Si desea utilizar una gema desempaquetada directamente desde el sistema de archivos, simplemente configure la opción: ruta a la ruta que contiene los archivos de la gema.

```
gem 'extracted_library', :path => './vendor/extracted_library'
```

Las dependencias se pueden colocar en grupos. Los grupos pueden ignorarse en el momento de la instalación (usando `--without`) o requerirse de una sola vez (usando `Bundler.require`).

```
gem 'rails_12factor', group: :production  
  
group :development, :test do  
  gem 'byebug'  
  gem 'web-console', '~> 2.0'  
  gem 'spring'  
  gem 'dotenv-rails'  
end
```

Puede especificar la versión requerida de Ruby en el Gemfile con `ruby`. Si el Gemfile se carga en una versión diferente de Ruby, Bundler generará una excepción con una explicación.

```
ruby '2.3.1'
```

## Gemas

Si está utilizando RVM (Ruby Version Manager) entonces es una buena idea usar un `gemset` de `gemset` para cada proyecto. Un `gemset` es solo un contenedor que puedes usar para mantener las gemas separadas unas de otras. Crear un conjunto de `gemset` por proyecto le permite cambiar gemas (y versiones de gemas) para un proyecto sin romper todos los otros proyectos. Cada proyecto solo necesita preocuparse por sus propias gemas.

RVM proporciona (`> = 0.1.8`) un `@global gemset` por intérprete de rubíes. Las gemas que instala en el

conjunto de `@global gemset` para un rubí determinado están disponibles para todos los demás conjuntos de gemas que cree en asociación con ese rubí. Esta es una buena manera de permitir que todos sus proyectos compartan la misma gema instalada para una instalación específica de un intérprete de Ruby.

## Creando gemsets

Supongamos que ya tiene instalado `ruby-2.3.1` y lo ha seleccionado con este comando:

```
rvm use ruby-2.3.1
```

Ahora para crear gemset para esta versión rubí:

```
rvm gemset create new_gemset
```

donde el `new_gemset` es el nombre de gemset. Para ver la lista de gemsets disponibles para una versión ruby:

```
rvm gemset list
```

para enumerar las gemas de todas las versiones de ruby:

```
rvm gemset list_all
```

para usar un gemset de la lista (supongamos que `new_gemset` es el gemset que quiero usar):

```
rvm gemset use new_gemset
```

También puede especificar la versión de ruby con el conjunto de gemas si desea cambiar a otra versión de ruby:

```
rvm use ruby-2.1.1@new_gemset
```

para especificar un conjunto de gemas predeterminado para una versión particular de ruby:

```
rvm use 2.1.1@new_gemset --default
```

para eliminar todas las gemas instaladas de un conjunto de gemas, puede vaciarlo de la siguiente manera:

```
rvm gemset empty new_gemset
```

para copiar un conjunto de gemas de un rubí a otro puedes hacerlo de la siguiente manera:

```
rvm gemset copy 2.1.1@rails4 2.1.2@rails4
```

para eliminar un gemset:

```
rvm gemset delete new_gemset
```

para ver el nombre actual de gemset:

```
rvm gemset name
```

para instalar una gema en el conjunto global de gemas:

```
rvm @global do gem install ...
```

## Inicializando Gemsets durante Ruby Instalaciones

Cuando instala un nuevo ruby, RVM no solo crea dos gemsets (el gemset vacío y el gemset global), sino que también utiliza un conjunto de archivos editables por el usuario para determinar qué gemas instalar.

Al trabajar en `~/.rvm/gemsets`, rvm busca `global.gems` y `default.gems` usando una jerarquía de árbol basada en la cadena ruby que se está instalando. Usando el ejemplo de `ree-1.8.7-p2010.02`, rvm verificará (e importará) los siguientes archivos:

```
~/.rvm/gemsets/ree/1.8.7/p2010.02/global.gems
~/.rvm/gemsets/ree/1.8.7/p2010.02/default.gems
~/.rvm/gemsets/ree/1.8.7/global.gems
~/.rvm/gemsets/ree/1.8.7/default.gems
~/.rvm/gemsets/ree/global.gems
~/.rvm/gemsets/ree/default.gems
~/.rvm/gemsets/global.gems
~/.rvm/gemsets/default.gems
```

Por ejemplo, si editó `~/.rvm/gemsets/global.gems` agregando estas dos líneas:

```
bundler
awesome_print
```

Cada vez que instalas un nuevo rubí, estas dos gemas se instalan en tu conjunto global de gemas. `global.gems` archivos `default.gems` y `global.gems` generalmente se sobrescriben durante la actualización de rvm.

Lea Gemas en línea: <https://riptutorial.com/es/ruby-on-rails/topic/3130/gemas>

---

# Capítulo 44: Herencia de una sola mesa

## Introducción

La herencia de una sola tabla (STI) es un patrón de diseño que se basa en la idea de guardar los datos de varios modelos que se heredan del mismo modelo base, en una sola tabla en la base de datos.

## Examples

### Ejemplo basico

Primero necesitamos una tabla para guardar nuestros datos.

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :password
      t.string :type # <- This makes it an STI

      t.timestamps
    end
  end
end
```

Entonces vamos a crear algunos modelos.

```
class User < ActiveRecord::Base
  validates_presence_of :password
  # This is a parent class. All shared logic goes here
end

class Admin < User
  # Admins must have more secure passwords than regular users
  # We can add it here
  validates :custom_password_validation
end

class Guest < User
  # Lets say that we have a guest type login.
  # It has a static password that cannot be changed
  validates_inclusion_of :password, in: ['guest_password']
end
```

Cuando hagas un `Guest.create(name: 'Bob')` ActiveRecord lo traducirá para crear una entrada en la tabla de Usuarios con el `type: 'Guest'`.

Cuando recupera el registro `bob = User.where(name: 'Bob').first` el objeto devuelto será una instancia de `Guest`, que se puede tratar a la fuerza como un usuario con `bob.becomes(User)`

es más útil cuando se trata de parciales compartidos o rutas / controladores de la superclase en lugar de la subclase.

## Columna de herencia personalizada

Por defecto, el nombre de la clase del modelo STI se almacena en una columna denominada `type`. Pero su nombre se puede cambiar al reemplazar el valor de `inheritance_column` en una clase base. P.ej:

```
class User < ActiveRecord::Base
  self.inheritance_column = :entity_type # can be string as well
end

class Admin < User; end
```

La migración en este caso se verá como sigue:

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :password
      t.string :entity_type

      t.timestamps
    end
  end
end
```

Cuando haga `Admin.create`, este registro se guardará en la tabla de usuarios con `entity_type = "Admin"`

## Modelo de rieles con columna tipo y sin STI.

Tener `type` columna en un modelo de rieles sin invocar STI se puede lograr mediante la asignación de `:_type_disabled` a `inheritance_column`:

```
class User < ActiveRecord::Base
  self.inheritance_column = :_type_disabled
end
```

Lea Herencia de una sola mesa en línea: <https://riptutorial.com/es/ruby-on-rails/topic/9125/herencia-de-una-sola-mesa>

---

# Capítulo 45: Herramientas para la optimización y limpieza del código de Ruby on Rails

## Introducción

Mantener su código limpio y organizado mientras desarrolla una gran aplicación de Rails puede ser todo un desafío, incluso para un desarrollador experimentado. Afortunadamente, hay toda una categoría de gemas que facilitan mucho este trabajo.

## Examples

Si desea mantener su código mantenible, seguro y optimizado, mire algunas gemas para la optimización y limpieza del código:

### Bala

Este en particular me dejó perplejo. La gema de bala te ayuda a eliminar todas las consultas de N + 1, así como las relaciones cargadas innecesariamente ansiosas. Una vez que lo instale y comience a visitar varias rutas en desarrollo, aparecerán recuadros de alerta con advertencias que indican las consultas de la base de datos que deben optimizarse. Funciona de inmediato y es extremadamente útil para optimizar su aplicación.

### Mejores Prácticas de Rieles

Analizador de código estático para encontrar olores de códigos específicos de Rails. Ofrece una variedad de sugerencias; use el acceso al alcance, restrinja las rutas generadas automáticamente, agregue índices de bases de datos, etc. Sin embargo, contiene muchas sugerencias agradables que le brindarán una mejor perspectiva sobre cómo redefinir su código y aprender algunas de las mejores prácticas.

### Rubocop

Un analizador de código estático Ruby que puede usar para verificar si su código cumple con las pautas del código comunitario de Ruby. La gema informa sobre las violaciones de estilo a través de la línea de comando, con una gran cantidad de elementos útiles de refactorización de códigos, como la asignación de variables inútiles, el uso redundante del Objeto # to\_s en la interpolación o incluso el argumento de método no utilizado.

Una cosa buena es que es altamente configurable, ya que el analizador puede ser bastante irritante si no está siguiendo la guía de estilo de Ruby al 100% (es decir, tiene muchos espacios en blanco al final o cita dos veces sus cadenas incluso cuando no está interpolando, etc.) .

Se divide en 4 subanálisis (llamados policías): estilo, pelusa, métricas y rieles.

Lea Herramientas para la optimización y limpieza del código de Ruby on Rails en línea:  
<https://riptutorial.com/es/ruby-on-rails/topic/8713/herramientas-para-la-optimizacion-y-limpieza-del-codigo-de-ruby-on-rails>

# Capítulo 46: I18n - Internacionalización

## Sintaxis

- `I18n.t("clave")`
- `I18n.translate("clave")` # equivalente a `I18n.t("key")`
- `I18n.t("clave", cuenta: 4)`
- `I18n.t("key", param1: "Something", param2: "Else")`
- `I18n.t("doesnt_exist", predeterminado: "clave")` # especifique un valor predeterminado si falta la clave
- `I18n.locale # =>: en`
- `I18n.locale =: en`
- `I18n.default_locale # =>: en`
- `I18n.default_locale =: en`
- `t(". clave")` # igual que `I18n.t("key")` , pero con el alcance de la acción / plantilla desde la que se llama

## Examples

### Usa I18n en vistas

Suponiendo que tiene este archivo de configuración regional YAML:

```
# config/locales/en.yml
en:
  header:
    title: "My header title"
```

y desea mostrar su cadena de título, puede hacer esto

```
# in ERB files
<%= t('header.title') %>

# in SLIM files
= t('header.title')
```

### I18n con argumentos

Puede pasar parámetros al método **I18n t** :

```
# Example config/locales/en.yml
en:
  page:
    users: "%{users_count} users currently online"

# In models, controller, etc...
I18n.t('page.users', users_count: 12)
```

```

# In views

# ERB
<%= t('page.users', users_count: 12) %>

#SLIM
= t('page.users', users_count: 12)

# Shortcut in views - DRY!
# Use only the dot notation
# Important: Consider you have the following controller and view page#users

# ERB Example app/views/page/users.html.erb
<%= t('.users', users_count: 12) %>

```

Y consigue la siguiente salida:

```
"12 users currently online"
```

## Pluralización

Puedes dejar que **I18n** maneje la pluralización por ti, solo usa el argumento `count` .

Necesitas configurar tu archivo de configuración regional de esta manera:

```

# config/locales/en.yml
en:
  online_users:
    one: "1 user is online"
    other: "%{count} users are online"

```

Y luego use la clave que acaba de crear pasando el argumento de `count` a `I18n.t` helper:

```

I18n.t("online_users", count: 1)
#=> "1 user is online"

I18n.t("online_users", count: 4)
#=> "4 users are online"

```

## Establecer la configuración regional a través de solicitudes

En la mayoría de los casos, es posible que desee establecer la `I18n` regional `I18n` . Es posible que desee establecer la configuración regional para la sesión actual, el usuario actual o en función de un parámetro de URL. Esto se puede lograr fácilmente implementando una `before_action` en uno de sus controladores, o en `ApplicationController` para tenerlo en todos sus controladores.

```

class ApplicationController < ActionController::Base
  before_action :set_locale

  protected

  def set_locale

```

```

# Remove inappropriate/unnecessary ones
I18n.locale = params[:locale] || # Request parameter
  session[:locale] || # Current session
  (current_user.preferred_locale if user_signed_in?) || # Model saved configuration
  extract_locale_from_accept_language_header || # Language header - browser
config
  I18n.default_locale # Set in your config files, english by super-default
end

# Extract language from request header
def extract_locale_from_accept_language_header
  if request.env['HTTP_ACCEPT_LANGUAGE']
    lg = request.env['HTTP_ACCEPT_LANGUAGE'].scan(/^[a-z]{2}/).first.to_sym
    lg.in?(:en, YOUR_AVAILABLE_LANGUAGES) ? lg : nil
  end
end
end

```

## Basado en URL

El parámetro `locale` podría provenir de una URL como esta

```
http://yourapplication.com/products?locale=en
```

O

```
http://yourapplication.com/en/products
```

Para lograr esto último, necesita editar sus `routes` , agregando un `scope` :

```

# config/routes.rb
scope "(:locale)", locale: /en|fr/ do
  resources :products
end

```

Al hacer esto, visitar `http://yourapplication.com/en/products` establecerá su configuración regional en `:en` . En su lugar, visitar `http://yourapplication.com/fr/products` lo configurará en `:fr` . Además, no obtendrá un error de enrutamiento cuando falte el **parámetro** `:locale` , ya que al visitar `http://yourapplication.com/products` se cargará la configuración regional predeterminada **I18n** .

## Sesión basada o basada en la persistencia

Esto supone que el usuario puede hacer clic en un botón / marca de idioma para cambiar el idioma. La acción puede dirigirse a un controlador que establece la sesión en el idioma actual (y eventualmente persistir los cambios en una base de datos si el usuario está conectado)

```

class SetLanguageController < ApplicationController
  skip_before_filter :authenticate_user!
  after_action :set_preferred_locale
end

```

```
# Generic version to handle a large list of languages
def change_locale
  I18n.locale = sanitize_language_param
  set_session_and_redirect
end
```

Debe definir `sanitize_language_param` con su lista de idiomas disponibles y, finalmente, manejar los errores en caso de que el idioma no exista.

Si tiene muy pocos idiomas, puede valer la pena definirlos así:

```
def fr
  I18n.locale = :fr
  set_session_and_redirect
end

def en
  I18n.locale = :en
  set_session_and_redirect
end

private

def set_session_and_redirect
  session[:locale] = I18n.locale
  redirect_to :back
end

def set_preferred_locale
  if user_signed_in?
    current_user.preferred_locale = I18n.locale.to_s
    current_user.save if current_user.changed?
  end
end

end
```

*Nota: no olvide agregar algunas rutas a sus acciones `change_language`*

## Configuración regional predeterminada

Recuerde que debe establecer la configuración regional predeterminada de su aplicación. Puede hacerlo configurándolo en `config/application.rb` :

```
config.i18n.default_locale = :de
```

o creando un inicializador en la carpeta `config/initializers` :

```
# config/initializers/locale.rb
I18n.default_locale = :it
```

## Obtener la configuración regional de la solicitud HTTP

A veces puede ser útil establecer la configuración regional de su aplicación en función de la solicitud de IP. Puedes lograrlo fácilmente usando `Geocoder`. Entre las muchas cosas que hace `Geocoder`, también puede indicar la `location` de una `request`.

Primero, agrega `Geocoder` a tu `Gemfile`

```
# Gemfile
gem 'geocoder'
```

`Geocoder` agrega `location` métodos de `location` y `location safe_location` al objeto estándar `Rack::Request` para que pueda buscar fácilmente la ubicación de cualquier solicitud HTTP por dirección IP. Puede usar estos métodos en una `before_action` en su `before_action` de

`ApplicationController`:

```
class ApplicationController < ActionController::Base
  before_action :set_locale_from_request

  def set_locale_from_request
    country_code = request.location.data["country_code"] #=> "US"
    country_sym = country_code.underscore.to_sym #=> :us

    # If request locale is available use it, otherwise use I18n default locale
    if I18n.available_locales.include? country_sym
      I18n.locale = country_sym
    end
  end
end
```

Tenga en cuenta que esto no funcionará en entornos de `development` y `test`, ya que cosas como `0.0.0.0` y `localhost` son direcciones IP de Internet válidas.

---

## Limitaciones y alternativas

`Geocoder` es muy potente y flexible, pero debe configurarse para funcionar con un *servicio de geocodificación* (ver [más detalles](#)); Muchos de los cuales ponen límites en el uso. También vale la pena tener en cuenta que llamar a un servicio externo en cada solicitud podría afectar el rendimiento.

Para abordar esto, también puede valer la pena considerar:

### 1. Una solución fuera de línea

El uso de una gema como `GeoIP` (ver [aquí](#)) permite que se realicen búsquedas en un archivo de datos local. Puede haber una compensación en términos de precisión, ya que estos archivos de datos deben mantenerse actualizados.

### 2. Utilice CloudFlare

Las páginas servidas a través de CloudFlare tienen la opción de geocodificarse de forma transparente, con el código del país que se agrega al encabezado ( `HTTP_CF_IPCOUNTRY` ). Más detalles se pueden encontrar [aquí](#) .

## Traducir los atributos del modelo ActiveRecord

`globalize` gem es una gran solución para agregar traducciones a sus modelos de `ActiveRecord` . Puedes instalarlo agregando esto a tu `Gemfile` :

```
gem 'globalize', '~> 5.0.0'
```

Si está utilizando `Rails 5` también necesitará agregar `activemodel-serializers-xml`

```
gem 'activemodel-serializers-xml'
```

Las traducciones de modelos le permiten traducir los valores de los atributos de sus modelos, por ejemplo:

```
class Post < ActiveRecord::Base
  translates :title, :text
end

I18n.locale = :en
post.title # => Globalize rocks!

I18n.locale = :he
post.title # => טלוש זייילאבולג!
```

Después de definir los atributos de su modelo que deben traducirse, debe crear una tabla de traducción a través de una migración. `globalize` proporciona `create_translation_table!` y `drop_translation_table!` .

Para esta migración, debe usar `up` y `down` , y **no** `change` . Además, para ejecutar esta migración correctamente, primero debe definir los atributos traducidos en su modelo, como se muestra arriba. Una migración adecuada para el modelo de `Post` anterior es esta:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table! title: :string, text: :text
  end

  def down
    Post.drop_translation_table!
  end
end
```

También puede pasar opciones para opciones específicas, como:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table! title: :string,
```

```

      text: { type: :text, null: false, default: "Default text" }
    end

    def down
      Post.drop_translation_table!
    end
  end
end

```

En caso de que ya tenga **datos** en las columnas de traducción que necesita, puede migrarlos fácilmente a la tabla de traducciones, ajustando su migración:

```

class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table!({
      title: :string,
      text: :text
    }, {
      migrate_data: true
    })
  end

  def down
    Post.drop_translation_table! migrate_data: true
  end
end

```

**Asegúrese de eliminar las columnas traducidas de la tabla principal después de que todos sus datos se hayan migrado de forma segura.** Para eliminar automáticamente las columnas traducidas de la tabla principal después de la migración de datos, agregue la opción `remove_source_columns` a la migración:

```

class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table!({
      title: :string,
      text: :text
    }, {
      migrate_data: true,
      remove_source_columns: true
    })
  end

  def down
    Post.drop_translation_table! migrate_data: true
  end
end

```

También puede agregar nuevos campos a una tabla de traducción creada anteriormente:

```

class Post < ActiveRecord::Base
  # Remember to add your attribute here too.
  translates :title, :text, :author
end

```

```
class AddAuthorToPost < ActiveRecord::Migration
  def up
    Post.add_translation_fields! author: :text
  end

  def down
    remove_column :post_translations, :author
  end
end
```

## Utilice I18n con etiquetas HTML y símbolos

```
# config/locales/en.yml
en:
  stackoverflow:
    header:
      title_html: "Use <strong>I18n</strong> with Tags & Symbols"
```

Tenga en cuenta la adición de `_html` adicional después del `title` nombre.

Y en vistas,

```
# ERB
<h2><%= t(:title_html, scope: [:stackoverflow, :header]) %></h2>
```

Lea I18n - Internacionalización en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2772/i18n---internacionalizacion>

---

# Capítulo 47: ID amigable

## Introducción

FriendlyId es el "bulldozer del ejército suizo" de complementos de slugging y permalink para Active Record. Te permite crear URL bonitas y trabajar con cadenas amigables para los humanos como si fueran identificadores numéricos. Con FriendlyId, es fácil hacer que su aplicación use URL como:

<http://example.com/states/washington>

## Examples

### Inicio rápido de rieles

```
rails new my_app
cd my_app
```

---

## Gemfile

```
gem 'friendly_id', '~> 5.1.0' # Note: You MUST use 5.0.0 or greater for Rails 4.0+
rails generate friendly_id
rails generate scaffold user name:string slug:string:uniq
rake db:migrate
```

---

## editar aplicación / modelos / usuario.rb

```
class User < ApplicationRecord
  extend FriendlyId
  friendly_id :name, use: :slugged
end

User.create! name: "Joe Schmoe"

# Change User.find to User.friendly.find in your controller
User.friendly.find(params[:id])
```

---

```
rails server
GET http://localhost:3000/users/joe-schmoe
```

---

```
# If you're adding FriendlyId to an existing app and need
```

```
# to generate slugs for existing users, do this from the
# console, runner, or add a Rake task:
User.find_each(&:save)

Finders are no longer overridden by default. If you want to do friendly finds, you must do
Model.friendly.find rather than Model.find. You can however restore FriendlyId 4-style finders
by using the :finders addon

friendly_id :foo, use: :slugged # you must do MyClass.friendly.find('bar')
#or...
friendly_id :foo, use: [:slugged, :finders] # you can now do MyClass.find('bar')
```

Una nueva funcionalidad "candidata" que facilita la configuración de una lista de bugs alternativos que se pueden usar para distinguir registros de forma única, en lugar de agregar una secuencia. Por ejemplo:

```
class Restaurant < ActiveRecord::Base
  extend FriendlyId
  friendly_id :slug_candidates, use: :slugged

  # Try building a slug based on the following fields in
  # increasing order of specificity.
  def slug_candidates
    [
      :name,
      [:name, :city],
      [:name, :street, :city],
      [:name, :street_number, :street, :city]
    ]
  end
end
```

¿Establecer la longitud del límite de babosa usando la gema friendly\_id?

```
def normalize_friendly_id(string)
  super[0..40]
end
```

Lea ID amigable en línea: <https://riptutorial.com/es/ruby-on-rails/topic/9664/id-amigable>

---

# Capítulo 48: Importar archivos CSV completos desde una carpeta específica

## Introducción

En este ejemplo, digamos que tenemos muchos archivos CSV de productos en una carpeta. Cada archivo CSV necesita cargar nuestra base de datos desde nuestra consola, escriba un comando. Ejecute el siguiente comando en un proyecto nuevo o existente para crear este modelo.

## Examples

### Cargas CSV desde comando de consola

Comandos de la terminal:

```
rails g model Product name:string quantity:integer price:decimal{12,2}
rake db:migrate
```

Lates crea el controlador.

Comandos de la terminal:

```
rails g controller Products
```

Código del controlador:

```
class HistoriesController < ApplicationController
  def create
    file = Dir.glob("#{Rails.root}/public/products/**/*.csv") #=> This folder directory
    where read the CSV files
    file.each do |file|
      Product.import(file)
    end
  end
end
```

Modelo:

```
class Product < ApplicationRecord
  def self.import(file)
    CSV.foreach(file.path, headers: true) do |row|
      Product.create! row.to_hash
    end
  end
end
```

rutas.rb

```
resources :products
```

app / config / application.rb

```
require 'csv'
```

Ahora abre tu `console` desarrollo y `run`

```
=> ProductsController.new.create #=> Uploads your whole CSV files from your folder directory
```

Lea [Importar archivos CSV completos desde una carpeta específica en línea](https://riptutorial.com/es/ruby-on-rails/topic/8658/importar-archivos-csv-completos-desde-una-carpeta-especifica):

<https://riptutorial.com/es/ruby-on-rails/topic/8658/importar-archivos-csv-completos-desde-una-carpeta-especifica>

---

# Capítulo 49: Integración de React.js con Rails usando Hyperloop

## Introducción

Este tema trata sobre la integración de React.js con Rails usando la gema [Hyperloop](#)

Otros enfoques no cubiertos aquí son usar las gemas `react-rails` o `react_on_rails`.

## Observaciones

Las clases de componentes simplemente generan las clases de componentes javascript equivalentes.

También puede acceder a los componentes y bibliotecas de javascript directamente desde sus clases de componentes ruby.

Hyperloop "pre-rendera" el lado del servidor de la vista, por lo que su vista inicial se cargará como las plantillas ERB o HAML. Una vez que se carga en el cliente, la reacción asume el control y actualizará de forma incremental el DOM a medida que el estado cambia debido a las entradas del usuario, las solicitudes HTTP o los datos de socket web entrantes.

Además de los Componentes, Hyperloop tiene Almacenes para administrar el estado compartido, Operaciones para encapsular la lógica empresarial isomórfica y Modelos que le dan acceso directo a sus modelos ActiveRecord en el cliente usando la sintaxis estándar de AR.

Más información aquí: <http://ruby-hyperloop.io/>

## Examples

### Agregar un componente de reacción simple (escrito en ruby) a su aplicación Rails

1. Agregue la gema `hyperloop` a sus rieles (4.0 - 5.1) Gemfile
2. `bundle install`
3. Agregue el manifiesto `hyperloop` al archivo `application.js`:

```
// app/assets/javascripts/application.js
...
//= hyperloop-loader
```

4. Cree sus componentes de reacción y colóquelos en el directorio `hyperloop/components`

```
# app/hyperloop/components/hello_world.rb
class HelloWorld < Hyperloop::Component
```

```

after_mount do
  every(1.second) { mutate.current_time(Time.now) }
end
render do
  "Hello World! The time is now: #{state.current_time}"
end
end

```

5. Los componentes actúan como las vistas. Se "montan" utilizando el método `render_component` en un controlador:

```

# somewhere in a controller:
...
def hello_world
  render_component # renders HelloWorld based on method name
end

```

## Declaración de parámetros de componentes (props)

```

class Hello < Hyperloop::Component
  # params (= react props) are declared using the param macro
  param :guest
  render do
    "Hello there #{params.guest}"
  end
end

# to "mount" Hello with guest = "Matz" say
Hello(guest: 'Matz')

# params can be given a default value:
param guest: 'friend' # or
param :guest, default: 'friend'

```

## Etiquetas HTML

```

# HTML tags are built in and are UPPERCASE
class HTMLExample < Hyperloop::Component
  render do
    DIV do
      SPAN { "Hello There" }
      SPAN { "Welcome to the Machine!" }
    end
  end
end

```

## Controladores de eventos

```

# Event handlers are attached using the 'on' method
class ClickMe < Hyperloop::Component
  render do
    DIV do
      SPAN { "Hello There" }
      A { "Click Me" }.on(:click) { alert('you did it!') }
    end
  end
end

```

```
end
end
end
```

## Estados

```
# States are read using the 'state' method, and updated using 'mutate'
# when states change they cause re-render of all dependent dom elements

class StateExample < Hyperloop::Component
  state count: 0 # by default states are initialized to nil
  render do
    DIV do
      SPAN { "Hello There" }
      A { "Click Me" }.on(:click) { mutate.count(state.count + 1) }
    DIV do
      "You have clicked me #{state.count} #{'time'.pluralize(state.count)}"
    end unless state.count == 0
  end
end
end
end
```

Tenga en cuenta que los estados se pueden compartir entre componentes utilizando [Hyperloop :: Stores](#)

## Devoluciones de llamada

```
# all react callbacks are supported using active-record-like syntax

class SomeCallbacks < Hyperloop::Component
  before_mount do
    # initialize stuff - replaces normal class initialize method
  end
  after_mount do
    # any access to actual generated dom node, or window behaviors goes here
  end
  before_unmount do
    # any cleanups (i.e. cancel intervals etc)
  end

  # you can also specify a method the usual way:
  before_mount :do_some_more_initialization
end
```

Lea Integración de React.js con Rails usando Hyperloop en línea: <https://riptutorial.com/es/ruby-on-rails/topic/9809/integracion-de-react-js-con-rails-usando-hyperloop>

---

# Capítulo 50: Interfaz de consulta ActiveRecord

## Introducción

ActiveRecord es la M en MVC, que es la capa del sistema responsable de representar los datos y la lógica de negocios. La técnica que conecta los objetos ricos de una aplicación a las tablas en un sistema de administración de base de datos relacional es **O** bject **R** elational **M** apper ( **ORM** ).

ActiveRecord realizará consultas en la base de datos por usted y es compatible con la mayoría de los sistemas de bases de datos. Independientemente del sistema de base de datos que esté utilizando, el formato del método ActiveRecord siempre será el mismo.

## Examples

### .dónde

El método `where` está disponible en cualquier modelo ActiveRecord y permite consultar la base de datos para un conjunto de registros que coincidan con los criterios dados.

El método `where` acepta un hash donde las claves corresponden a los nombres de columna en la tabla que representa el modelo.

Como ejemplo simple, usaremos el siguiente modelo:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end
```

Para encontrar a todas las personas con el nombre de `Sven` :

```
people = Person.where(first_name: 'Sven')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven'"
```

Para encontrar a todas las personas con el nombre de `Sven` y el apellido de `Schrodinger` :

```
people = Person.where(first_name: 'Sven', last_name: 'Schrodinger')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven' AND last_name='Schrodinger'"
```

En el ejemplo anterior, la salida sql muestra que sólo los registros serán devueltos si tanto el `first_name` y el `last_name` partido.

### consulta con condición OR

Para buscar registros con el primer nombre `first_name == 'Bruce'` O el `last_name == 'Wayne'`

```
User.where('first_name = ? or last_name = ?', 'Bruce', 'Wayne')
# SELECT "users".* FROM "users" WHERE (first_name = 'Bruce' or last_name = 'Wayne')
```

## .where con una matriz

El método `where` en cualquier modelo ActiveRecord se puede usar para generar SQL de la forma `WHERE column_name IN (a, b, c, ...)`. Esto se logra pasando una matriz como argumento.

Como ejemplo simple, usaremos el siguiente modelo:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end

people = Person.where(first_name: ['Mark', 'Mary'])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary')"
```

Si la matriz contiene un valor `nil`, el SQL se modificará para verificar si la columna es `null`:

```
people = Person.where(first_name: ['Mark', 'Mary', nil])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary') OR first_name IS NULL"
```

## Alcances

Los ámbitos actúan como filtros predefinidos en los modelos ActiveRecord.

Un alcance se define utilizando el método de clase de `scope`.

Como ejemplo simple, usaremos el siguiente modelo:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
  #attribute :age, :integer

  # define a scope to get all people under 17
  scope :minors, -> { where(age: 0..17) }

  # define a scope to search a person by last name
  scope :with_last_name, ->(name) { where(last_name: name) }
end
```

Los ámbitos se pueden llamar directamente desde la clase modelo:

```
minors = Person.minors
```

Los alcances pueden ser encadenados:

```
peters_children = Person.minors.with_last_name('Peters')
```

El método `where` y otros métodos de tipo de consulta también se pueden encadenar:

```
mary_smith = Person.with_last_name('Smith').where(first_name: 'Mary')
```

Detrás de las escenas, los ámbitos son simplemente azúcar sintáctica para un método de clase estándar. Por ejemplo, estos métodos son funcionalmente idénticos:

```
scope :with_last_name, ->(name) { where(name: name) }

# This ^ is the same as this:

def self.with_last_name(name)
  where(name: name)
end
```

## Alcance predeterminado

en su modelo para establecer un alcance predeterminado para todas las operaciones en el modelo.

Hay una diferencia notable entre el `scope` método y un método de clase: `scope` -definida alcances *siempre* devolverá un `ActiveRecord::Relation`, incluso si la lógica dentro devuelve `nil`. Los métodos de clase, sin embargo, no tienen dicha red de seguridad y pueden romper la capacidad de cadena si devuelven algo más.

## donde no

`where` cláusulas pueden ser negadas usando la sintaxis de `where.not` :

```
class Person < ApplicationRecord
  #attribute :first_name, :string
end

people = Person.where.not(first_name: ['Mark', 'Mary'])
# => SELECT "people".* FROM "people" WHERE "people"."first_name" NOT IN ('Mark', 'Mary')
```

Compatible con ActiveRecord 4.0 y versiones posteriores.

## Ordenando

Puede ordenar los resultados de la consulta de **ActiveRecord** usando `.order` :

```
User.order(:created_at)
#=> => [#<User id: 2, created_at: "2015-08-12 21:36:23">, #<User id: 11, created_at: "2015-08-15 10:21:48">]
```

Si no se especifica, el orden se realizará en orden ascendente. Puedes especificarlo haciendo:

```
User.order(created_at: :asc)
#=> => [#<User id: 2, created_at: "2015-08-12 21:36:23">, #<User id: 11, created_at: "2015-08-15 10:21:48">]

User.order(created_at: :desc)
#=> [#<User id: 7585, created_at: "2016-07-13 17:15:27">, #<User id: 7583, created_at: "2016-07-13 16:51:18">]
```

**.order** también acepta una cadena, por lo que también podría hacer

```
User.order("created_at DESC")
#=> [#<User id: 7585, created_at: "2016-07-13 17:15:27">, #<User id: 7583, created_at: "2016-07-13 16:51:18">]
```

Como la cadena es SQL sin formato, también puede especificar una tabla y no solo un atributo. Suponiendo que desea ordenar a los `users` según el nombre de su `role`, puede hacer esto:

```
Class User < ActiveRecord::Base
  belongs_to :role
end

Class Role < ActiveRecord::Base
  has_many :users
end

User.includes(:role).order("roles.name ASC")
```

El alcance del `order` también puede aceptar un nodo Arel:

```
User.includes(:role).order(User.arel_table[:name].asc)
```

## Métodos ActiveRecord Bang (!)

Si necesita un método **ActiveRecord** para generar una excepción en lugar de un valor `false` en caso de fallo, ¡puede agregar ! a ellos. Esto es muy importante. ¡Como algunas excepciones / fallas son difíciles de detectar si no las usa! en ellos. Recomendé hacer esto en su ciclo de desarrollo para escribir todo su código ActiveRecord de esta manera para ahorrarle tiempo y problemas.

```
Class User < ActiveRecord::Base
  validates :last_name, presence: true
end

User.create!(first_name: "John")
#=> ActiveRecord::RecordInvalid: Validation failed: Last name can't be blank
```

Los métodos de **ActiveRecord** que aceptan un *bang* ( ! ) Son:

- `.create!`
- `.take!`
- `.first!`
- `.last!`

- `.find_by!`
- `.find_or_create_by!`
- `#save!`
- `#update!`
- todos los buscadores dinámicos AR

## `.find_by`

Puede encontrar registros por cualquier campo en su tabla usando `find_by`.

Por lo tanto, si tiene un modelo de `User` con un atributo `first_name`, puede hacerlo:

```
User.find_by(first_name: "John")
#=> #<User id: 2005, first_name: "John", last_name: "Smith">
```

`find_by` cuenta que `find_by` no lanza ninguna excepción de forma predeterminada. Si el resultado es un conjunto vacío, devuelve `nil` lugar de `find`.

Si la excepción es necesaria puede usar `find_by!` que plantea un error

`ActiveRecord::RecordNotFound` como `find`.

## `.eliminar todos`

Si necesita eliminar muchos registros rápidamente, **ActiveRecord** le da el método `.delete_all` para ser llamado directamente en un modelo, para eliminar todos los registros en esa tabla, o una colección. Sin embargo, `.delete_all` cuidado, ya que `.delete_all` no `.delete_all` una instancia de ningún objeto, por lo tanto, no proporciona ninguna devolución de llamada (`before_*` y `after_destroy` no se activan)

```
User.delete_all
#=> 39 <-- .delete_all return the number of rows deleted

User.where(name: "John").delete_all
```

## ActiveRecord caso de búsqueda insensible

Si necesita buscar valores similares en un modelo ActiveRecord, es posible que tenga la tentación de usar `LIKE` o `ILIKE` pero esto no es portátil entre los motores de base de datos. Del mismo modo, recurrir siempre a la reducción o mejora de la situación puede crear problemas de rendimiento.

Puede utilizar Arel subyacente de ActiveRecord `matches` método para hacer esto de una manera segura:

```
addresses = Address.arel_table
Address.where(addresses[:address].matches("%street%"))
```

Arel aplicará la construcción `LIKE` o `ILIKE` apropiada para el motor de base de datos configurado.

## Obtener primer y último registro

Los rieles tienen una forma muy sencilla de obtener el `first` y `last` registro de la base de datos.

Para obtener el `first` registro de la tabla de `users` , debemos escribir el siguiente comando:

```
User.first
```

Se generará la siguiente consulta `sql` :

```
SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC LIMIT 1
```

Y volverá el siguiente registro:

```
#<User:0x007f8a6db09920 id: 1, first_name: foo, created_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00, updated_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00 >
```

Para obtener el `last` registro de la tabla de `users` , debemos escribir el siguiente comando:

```
User.last
```

Se generará la siguiente consulta `sql` :

```
SELECT `users`.* FROM `users` ORDER BY `users`.`id` DESC LIMIT 1
```

Y volverá el siguiente registro:

```
#<User:0x007f8a6db09920 id: 10, first_name: bar, created_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00, updated_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00 >
```

Pasar un entero al **primer** y **último** método crea una consulta **LIMIT** y devuelve una matriz de objetos.

```
User.first(5)
```

Se generará la siguiente consulta `sql` .

```
SELECT "users".* FROM "users" ORDER BY "users"."id" ASC LIMIT 5
```

Y

```
User.last(5)
```

Se generará la siguiente consulta `sql` .

```
SELECT "users".* FROM "users" ORDER BY "users"."id" DESC LIMIT 5
```

## .group y .count

Tenemos un modelo de `Product` y queremos agruparlos por `category`.

```
Product.select(:category).group(:category)
```

Esto consultará la base de datos de la siguiente manera:

```
SELECT "product"."category" FROM "product" GROUP BY "product"."category"
```

Asegúrese de que el campo agrupado también esté seleccionado. La agrupación es especialmente útil para contar la ocurrencia, en este caso, de las `categories`.

```
Product.select(:category).group(:category).count
```

Como muestra la consulta, utilizará la base de datos para el conteo, que es mucho más eficiente, que recuperar todo el registro primero y hacer el conteo en el código:

```
SELECT COUNT("products"."category") AS count_categories, "products"."category" AS products_category FROM "products" GROUP BY "products"."category"
```

## .distinto (o .uniq)

Si desea eliminar duplicados de un resultado, puede usar `.distinct()`:

```
Customers.select(:country).distinct
```

Esto consulta la base de datos de la siguiente manera:

```
SELECT DISTINCT "customers"."country" FROM "customers"
```

`.uniq()` tiene el mismo efecto. Con Rails 5.0 quedó en desuso y se eliminará de Rails con la versión 5.1. La razón es que la palabra `unique` no tiene el mismo significado que `distinta` y puede ser engañosa. Además `distinct` está más cerca de la sintaxis de SQL.

## Se une

`joins()` permite unir tablas a su modelo actual. Por ej.

```
User.joins(:posts)
```

producirá la siguiente consulta SQL:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id"
```

Teniendo mesa unida, tendrás acceso a ella:

```
User.joins(:posts).where(posts: { title: "Hello world" })
```

Preste atención en forma plural. Si su relación es `:has_many`, entonces el argumento `joins()` debe estar pluralizado. De lo contrario, utilice singular.

Anidada `joins`:

```
User.joins(posts: :images).where(images: { caption: 'First post' })
```

que producirá:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id" INNER JOIN "images" ON "images"."post_id" = "images"."id"
```

## Incluye

`ActiveRecord` con `includes` garantiza que todas las asociaciones especificadas se carguen utilizando el número mínimo posible de consultas. Por lo tanto, al consultar datos de una tabla con una tabla asociada, ambas tablas se cargan en la memoria.

```
@authors = Author.includes(:books).where(books: { bestseller: true } )

# this will print results without additional db hitting
@authors.each do |author|
  author.books.each do |book|
    puts book.title
  end
end
```

`Author.joins(:books).where(books: { bestseller: true } )` cargará solo a los **autores** con condiciones en la memoria **sin cargar libros**. Utilice `joins` cuando no se requiera información adicional sobre asociaciones anidadas.

```
@authors = Author.joins(:books).where(books: { bestseller: true } )

# this will print results without additional queries
@authors.each { |author| puts author.name }

# this will print results with additional db queries
@authors.each do |author|
  author.books.each do |book|
    puts book.title
  end
end
```

## Límite y compensación

Puede usar el `limit` para indicar el número de registros que se van a recuperar, y usar la `offset` para indicar el número de registros que se deben omitir antes de comenzar a devolver los registros.

Por ejemplo

```
User.limit(3) #returns first three records
```

Se generará la siguiente consulta sql.

```
"SELECT `users`.* FROM `users` LIMIT 3"
```

Como el desplazamiento no se menciona en la consulta anterior, devolverá los primeros tres registros.

```
User.limit(5).offset(30) #returns 5 records starting from 31th i.e from 31 to 35
```

Se generará la siguiente consulta sql.

```
"SELECT `users`.* FROM `users` LIMIT 5 OFFSET 30"
```

Lea Interfaz de consulta ActiveRecord en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2154/interfaz-de-consulta-activerecord>

# Capítulo 51: Los rieles generan comandos.

## Introducción

Uso: `rails generate GENERATOR_NAME [args] [options]` .

Use `rails generate` para listar generadores disponibles. Alias: `rails g` .

## Parámetros

Parámetro	Detalles
<code>-h / --help</code>	Obtenga ayuda en cualquier comando del generador
<code>-p / - --pretend</code>	Modo de simulación: ejecute el generador pero no creará ni cambiará ningún archivo
<code>field:type</code>	'nombre-campo' es el nombre de la columna que se creará y 'tipo' es el tipo de datos de la columna. Los valores posibles para 'tipo' en el <code>field:type</code> se dan en la sección Comentarios.

## Observaciones

Los valores posibles para 'tipo' en el `field:type` son:

Tipo de datos	Descripción
<code>:string</code>	Para fragmentos de texto más pequeños (generalmente tiene un límite de caracteres de 255)
<code>:text</code>	Para textos más largos, como un párrafo.
<code>:binary</code>	Almacenamiento de datos incluyendo imágenes, audios y videos
<code>:boolean</code>	Almacenar valores verdaderos o falsos
<code>:date</code>	Solo la fecha
<code>:time</code>	Solo el tiempo
<code>:datetime</code>	Fecha y hora
<code>:float</code>	Almacenando flotadores sin precisión
<code>:decimal</code>	Almacenando flotadores con precisión

Tipo de datos	Descripción
:integer	Almacenando números enteros

## Examples

### Generar rieles modelo

Para generar un modelo `ActiveRecord` que genere automáticamente las migraciones db correctas y los archivos de prueba para su modelo, ingrese este comando

```
rails generate model NAME column_name:column_type
```

'NOMBRE' es el nombre del modelo. 'campo' es el nombre de la columna en la tabla de base de datos y 'tipo' es el tipo de columna (por ejemplo, `name:string` o `body:text` ). Consulte la sección de Comentarios para obtener una lista de los tipos de columna admitidos.

Para configurar las claves foráneas, `belongs_to:model_name` .

Entonces, digamos que desea configurar un modelo de `User` que tenga un `username` , `email` y pertenezca a una `School` , escriba lo siguiente

```
rails generate model User username:string email:string school:belongs_to
```

`rails g` son taquigrafía para `rails generate` . Esto produciría el mismo resultado.

```
rails g model User username:string email:string school:belongs_to
```

### Rails Generar Migración

Puede generar un archivo de migración de rieles desde el terminal usando el siguiente comando:

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

Para obtener una lista de todas las opciones admitidas por el comando, puede ejecutar el comando sin ningún argumento, ya que los `rails generate migration` .

Por ejemplo, si desea agregar `first_name` y `last_name` campos a `users` la tabla, que puede hacer:

```
rails generate migration AddNamesToUsers last_name:string first_name:string
```

Rails creará el siguiente archivo de migración:

```
class AddNamesToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :last_name, :string
  end
end
```

```
    add_column :users, :first_name, :string
  end
end
```

Ahora, aplique las migraciones pendientes a la base de datos ejecutando lo siguiente en el terminal:

5.0

```
rake db:migrate
```

5.0

```
rails db:migrate
```

**Nota:** Para escribir menos, puede reemplazar `generate` con `g`.

## Rieles Generan Andamios

**NEGACIÓN DE RESPONSABILIDAD:** No se recomienda el uso de andamios a menos que sea para aplicaciones / pruebas CRUD muy convencionales. Esto puede generar una gran cantidad de archivos (vistas / modelos / controladores) que no son necesarios en su aplicación web, causando así dolores de cabeza (mal :()).

Para generar un andamio en pleno funcionamiento para un nuevo objeto, incluidos el modelo, el controlador, las vistas, los activos y las pruebas, use el comando `rails g scaffold`.

```
$ rails g scaffold Widget name:string price:decimal
  invoke  active_record
  create  db/migrate/20160722171221_create_widgets.rb
  create  app/models/widget.rb
  invoke  test_unit
  create  test/models/widget_test.rb
  create  test/fixtures/widgets.yml
  invoke  resource_route
  route   resources :widgets
  invoke  scaffold_controller
  create  app/controllers/widgets_controller.rb
  invoke  erb
  create  app/views/widgets
  create  app/views/widgets/index.html.erb
  create  app/views/widgets/edit.html.erb
  create  app/views/widgets/show.html.erb
  create  app/views/widgets/new.html.erb
  create  app/views/widgets/_form.html.erb
  invoke  test_unit
  create  test/controllers/widgets_controller_test.rb
  invoke  helper
  create  app/helpers/widgets_helper.rb
  invoke  jbuilder
  create  app/views/widgets/index.json.jbuilder
  create  app/views/widgets/show.json.jbuilder
  invoke  assets
  invoke  javascript
```

```
create    app/assets/javascripts/widgets.js
invoke   scss
create    app/assets/stylesheets/widgets.scss
```

Luego puede ejecutar `rake db:migrate` para configurar la tabla de la base de datos.

Luego puede visitar <http://localhost:3000/widgets> y verá un andamio CRUD completamente funcional.

## Rails Generate Controller

Podemos crear un nuevo controlador con el comando `rails g controller`.

```
$ bin/rails generate controller controller_name
```

El generador del controlador espera parámetros en forma de `generate controller ControllerName action1 action2`.

Lo siguiente crea un controlador de saludos con una acción de hola.

```
$ bin/rails generate controller Greetings hello
```

Verás la siguiente salida.

```
create  app/controllers/greetings_controller.rb
route   get "greetings/hello"
invoke  erb
create  app/views/greetings
create  app/views/greetings/hello.html.erb
invoke  test_unit
create  test/controllers/greetings_controller_test.rb
invoke  helper
create  app/helpers/greetings_helper.rb
invoke  assets
invoke  coffee
create  app/assets/javascripts/greetings.coffee
invoke  scss
create  app/assets/stylesheets/greetings.scss
```

Esto genera lo siguiente

Expediente	Ejemplo
Archivo controlador	<code>greetings_controller.rb</code>
Ver archivo	<code>hello.html.erb</code>
Archivo de prueba funcional	<code>greetings_controller_test.rb</code>
Ver Ayudante	<code>greetings_helper.rb</code>
Archivo de JavaScript	<code>greetings.coffee</code>

También agregará rutas para cada acción en `routes.rb`

Lea [Los rieles generan comandos](https://riptutorial.com/es/ruby-on-rails/topic/2540/los-rieles-generan-comandos-). en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2540/los-rieles-generan-comandos->

---

# Capítulo 52: Mejores Prácticas de Rieles

## Examples

### No te repitas (SECO)

Para ayudar a mantener el código limpio, Rails sigue el principio de DRY.

Implica siempre que sea posible, reutilizar tanto código como sea posible en lugar de duplicar código similar en múltiples lugares (por ejemplo, usar parciales). Esto reduce los *errores*, mantiene su código *limpio* y aplica el principio de *escribir código una vez* y luego reutilizarlo. También es más fácil y más eficiente actualizar el código en un solo lugar que actualizar varias partes del mismo código. Haciendo así tu código más modular y robusto.

También *Fat Model*, *Skinny Controller* está SECO, porque usted escribe el código en su modelo y en el controlador solo hace la llamada, como:

```
# Post model
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at > ?', timestamp) }

# Any controller
def index
  ....
  @unpublished_posts = Post.unpublished
  ....
end

def others
  ...
  @unpublished_posts = Post.unpublished
  ...
end
```

Esto también ayuda a conducir a una estructura basada en API donde los métodos internos están ocultos y los cambios se logran a través de pasar parámetros en forma de API.

### Convención sobre configuración

En Rails, te encuentras mirando *controladores*, *vistas* y *modelos* para tu base de datos.

Para reducir la necesidad de una configuración pesada, Rails implementa reglas para facilitar el trabajo con la aplicación. Puede definir sus propias reglas, pero para el principio (y para más adelante) es una buena idea atenerse a las convenciones que ofrece Rails.

Estas convenciones acelerarán el desarrollo, mantendrán su código conciso y legible, y le permitirán una fácil navegación dentro de su aplicación.

Las convenciones también reducen las barreras de entrada para los principiantes. Hay tantas

convenciones en Rails que un principiante ni siquiera necesita conocer, pero que solo pueden beneficiarse de la ignorancia. Es posible crear grandes aplicaciones sin saber por qué todo es como es.

## Por ejemplo

Si tiene una tabla de base de datos llamada `orders` con el `id` clave principal, el modelo coincidente se llama `order` y el controlador que maneja toda la lógica se denomina `orders_controller`. La vista se divide en diferentes acciones: si el controlador tiene una acción `new` y de `edit`, también hay una vista `new` y de `edit`.

## Por ejemplo

Para crear una aplicación, simplemente ejecuta `rails new app_name`. Esto generará aproximadamente 70 archivos y carpetas que conforman la infraestructura y la base de su aplicación Rails.

Incluye:

- Carpetas para guardar sus modelos (capa de base de datos), controladores y vistas
- Carpetas para realizar pruebas unitarias para su aplicación.
- Carpetas para guardar sus activos web como archivos Javascript y CSS
- Archivos predeterminados para respuestas HTTP 400 (es decir, archivo no encontrado)
- Muchos otros

## Modelo gordo, flaco controlador

"Modelo gordo, controlador delgado" se refiere a cómo las partes M y C de MVC trabajan idealmente juntas. Es decir, cualquier lógica no relacionada con la respuesta debe ir en el modelo, idealmente en un método agradable y comprobable. Mientras tanto, el controlador "delgado" es simplemente una interfaz agradable entre la vista y el modelo.

En la práctica, esto puede requerir un rango de diferentes tipos de refactorización, pero todo se reduce a una idea: al mover cualquier lógica que no sea sobre la respuesta al modelo (en lugar del controlador), no solo ha promovido la reutilización siempre que sea posible, pero también ha hecho posible probar su código fuera del contexto de una solicitud.

Veamos un ejemplo simple. Digamos que tienes un código como este:

```
def index
  @published_posts = Post.where('published_at <= ?', Time.now)
  @unpublished_posts = Post.where('published_at IS NULL OR published_at > ?', Time.now)
end
```

Puedes cambiarlo a esto:

```
def index
  @published_posts = Post.published
  @unpublished_posts = Post.unpublished
end
```

Luego, puede mover la lógica a su modelo de publicación, donde podría verse así:

```
scope :published, ->(timestamp = Time.now) { where('published_at <= ?', timestamp) }
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at >
?', timestamp) }
```

## Cuidado con default\_scope

ActiveRecord incluye `default_scope`, para `default_scope` automáticamente un modelo de forma predeterminada.

```
class Post
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

El código anterior servirá publicaciones que ya están publicadas cuando realiza cualquier consulta en el modelo.

```
Post.all # will only list published posts
```

Ese alcance, aunque tiene un aspecto inocuo, tiene múltiples efectos secundarios ocultos que tal vez no desee.

`default_scope` **y** `order`

Dado que declaró un `order` en el `default_scope`, el `order` llamada en `Post` se agregará como pedidos adicionales en lugar de anular el valor predeterminado.

```
Post.order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."created_at"
DESC, "posts"."updated_at" DESC
```

Probablemente este no sea el comportamiento que querías; puede anular esto excluyendo el `order` del alcance primero

```
Post.except(:order).order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."updated_at"
DESC
```

---

## `default_scope` y modelo de inicialización

Al igual que con cualquier otro `ActiveRecord::Relation`, `default_scope` alterará el estado predeterminado de los modelos que se inicializaron a partir de él.

En el ejemplo anterior, la `Post` tiene `where(published: true)` establece de forma predeterminada, por lo que los nuevos modelos de la `Post` también lo tendrán configurado.

```
Post.new # => <Post published: true>
```

## unscoped

`default_scope` se puede borrar nominalmente llamando primero sin `unscoped`, pero esto también tiene efectos secundarios. Tomemos, por ejemplo, un modelo de STI:

```
class Post < Document
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

De forma predeterminada, las consultas contra `Post` tendrán un alcance para `type` columnas que contengan `'Post'`. Sin embargo, `unscoped` lo `unscoped` junto con su propio `default_scope`, por lo que si usa `unscoped` también debe recordar tenerlo en cuenta.

```
Post.unscoped.where(type: 'Post').order(updated_at: :desc)
```

## unscoped Asociaciones y Modelo

Considera una relación entre `Post` y `User`

```
class Post < ApplicationRecord
  belongs_to :user
  default_scope ->{ where(published: true).order(created_at: :desc) }
end

class User < ApplicationRecord
  has_many :posts
end
```

Al obtener un `User` individual, puede ver las publicaciones relacionadas con él:

```
user = User.find(1)
user.posts
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' AND "posts"."user_id" = ? ORDER BY "posts"."created_at" DESC [["user_id", 1]]
```

Pero desea borrar el `default_scope` de la relación de `posts`, por lo que utiliza sin `unscoped`

```
user.posts.unscoped
```

```
SELECT "posts".* FROM "posts"
```

Esto elimina la condición `user_id` así como el `default_scope`.

## Un ejemplo de caso de uso para `default_scope`

A pesar de todo eso, hay situaciones en las que el uso de `default_scope` es justificable.

Considere un sistema de múltiples inquilinos donde se sirven múltiples subdominios desde la misma aplicación pero con datos aislados. Una forma de lograr este aislamiento es a través de `default_scope`. Las desventajas en otros casos se convierten en ventajas aquí.

```
class ApplicationRecord < ActiveRecord::Base
  def self.inherited(subclass)
    super

    return unless subclass.superclass == self
    return unless subclass.column_names.include? 'tenant_id'

    subclass.class_eval do
      default_scope ->{ where(tenant_id: Tenant.current_id) }
    end
  end
end
```

Todo lo que necesita hacer es establecer `Tenant.current_id` en algo al principio de la solicitud, y cualquier tabla que contenga `tenant_id` se convertirá automáticamente en un ámbito sin ningún código adicional. Los registros de creación de instancias heredarán automáticamente la identificación del inquilino con la que se crearon.

Lo importante de este caso de uso es que el alcance se establece una vez por solicitud y no cambia. Los únicos casos que necesitará sin `unscoped` aquí son casos especiales, como los trabajadores en segundo plano que se ejecutan fuera del alcance de una solicitud.

### No lo vas a necesitar (YAGNI)

Si puede decir "YAGNI" (No lo va a necesitar) sobre una característica, es mejor que no la implemente. Puede ahorrarse mucho tiempo de desarrollo al centrarse en la simplicidad. Implementar tales características de todas formas puede llevar a problemas:

## Problemas

### Sobreingeniería

Si un producto es más complicado de lo que tiene que ser, está sobre diseñado. Por lo general, estas características "aún no utilizadas" nunca se utilizarán de la forma prevista en que fueron escritas y deben ser refactorizadas si alguna vez se usan. Las optimizaciones prematuras, especialmente las optimizaciones de rendimiento, a menudo conducen a decisiones de diseño que resultarán incorrectas en el futuro.

## Código Inflado

Código Bloat significa código complicado innecesario. Esto puede ocurrir, por ejemplo, por abstracción, redundancia o aplicación incorrecta de patrones de diseño. El código base se vuelve difícil de entender, confuso y costoso de mantener.

## Característica de arrastramiento

Feature Creep se refiere a agregar nuevas funciones que van más allá de la funcionalidad central del producto y conducen a una complejidad innecesariamente alta del producto.

## Largo tiempo de desarrollo

El tiempo que podría usarse para desarrollar las características necesarias se emplea para desarrollar características innecesarias. El producto tarda más en entregarse.

---

## Soluciones

### KISS - Que sea simple, estúpido.

Según KISS, la mayoría de los sistemas funcionan mejor si están diseñados de manera simple. La simplicidad debe ser un objetivo principal de diseño para reducir la complejidad. Se puede lograr siguiendo el "Principio de Responsabilidad Única", por ejemplo.

### YAGNI - No lo vas a necesitar

Menos es más. Piense en todas las características, ¿es realmente necesario? Si puedes pensar en alguna forma de que sea YAGNI, déjalo. Es mejor desarrollarlo cuando es necesario.

## Refactorización continua

El producto se está mejorando constantemente. Con la refactorización, podemos asegurarnos de que el producto se está realizando de acuerdo con las mejores prácticas y no se degenerará en un trabajo de parche.

### Objetos de dominio (no más modelos de grasa)

"Fat Model, Skinny Controller" es un muy buen primer paso, pero no se escala bien una vez que tu base de código comienza a crecer.

Pensemos en la [Responsabilidad Única](#) de los modelos. ¿Cuál es la única responsabilidad de los modelos? ¿Es para mantener la lógica empresarial? ¿Es para mantener la lógica no relacionada con la respuesta?

No. Su responsabilidad es manejar la capa de persistencia y su abstracción.

La lógica de negocios, así como cualquier lógica no relacionada con la respuesta y la lógica no relacionada con la persistencia, deben incluirse en los objetos de dominio.

Los objetos de dominio son clases diseñadas para tener una sola responsabilidad en el dominio del problema. Deja que tus clases " [griten su arquitectura](#) " para los problemas que resuelven.

En la práctica, debes esforzarte por conseguir modelos delgados, vistas delgadas y controladores delgados. La arquitectura de su solución no debe verse influida por el marco que elija.

## Por ejemplo

Digamos que usted es un mercado que cobra una comisión fija del 15% a sus clientes a través de Stripe. Si cobra una comisión fija del 15%, eso significa que su comisión varía según el monto del pedido porque Stripe cobra 2.9% + 30 ¢.

El monto que cobra como comisión debe ser:  $\text{amount} * 0.15 - (\text{amount} * 0.029 + 0.30)$  .

No escriba esta lógica en el modelo:

```
# app/models/order.rb
class Order < ActiveRecord::Base
  SERVICE_COMMISSION = 0.15
  STRIPE_PERCENTAGE_COMMISSION = 0.029
  STRIPE_FIXED_COMMISSION = 0.30

  ...

  def commission
    amount * SERVICE_COMMISSION - stripe_commission
  end

  private

  def stripe_commission
    amount * STRIPE_PERCENTAGE_COMMISSION + STRIPE_FIXED_COMMISSION
  end
end
```

Tan pronto como se integre con un nuevo método de pago, no podrá escalar esta funcionalidad dentro de este modelo.

Además, tan pronto como empiece a integrar más lógica de negocios, su objeto `Order` comenzará a perder [cohesión](#) .

Prefiera los objetos de dominio, con el cálculo de la comisión totalmente abstraído de la responsabilidad de las órdenes persistentes:

```
# app/models/order.rb
class Order < ActiveRecord::Base
  ...
  # No reference to commission calculation
end
```

```

# lib/commission.rb
class Commission
  SERVICE_COMMISSION = 0.15

  def self.calculate(payment_method, model)
    model.amount*SERVICE_COMMISSION - payment_commission(payment_method, model)
  end

  private

  def self.payment_commission(payment_method, model)
    # There are better ways to implement a static registry,
    # this is only for illustration purposes.
    Object.const_get("#{payment_method}Commission").calculate(model)
  end
end

# lib/stripe_commission.rb
class StripeCommission
  STRIPE_PERCENTAGE_COMMISSION = 0.029
  STRIPE_FIXED_COMMISSION = 0.30

  def self.calculate(model)
    model.amount*STRIPE_PERCENTAGE_COMMISSION
    + STRIPE_PERCENTAGE_COMMISSION
  end
end

# app/controllers/orders_controller.rb
class OrdersController < ApplicationController
  def create
    @order = Order.new(order_params)
    @order.commission = Commission.calculate("Stripe", @order)
    ...
  end
end

```

El uso de objetos de dominio tiene las siguientes ventajas arquitectónicas:

- es extremadamente fácil de realizar una prueba unitaria, ya que no se requieren accesorios ni fábricas para instanciar los objetos con la lógica.
- Funciona con todo lo que acepta el `amount` mensaje.
- mantiene cada objeto de dominio pequeño, con responsabilidades claramente definidas y con mayor cohesión.
- Se escala fácilmente con nuevos métodos de pago por [adición, no por modificación](#) .
- detiene la tendencia a tener un objeto de `User` cada vez mayor en cada aplicación de Ruby on Rails.

Personalmente me gusta poner objetos de dominio en `lib` . Si lo hace, recuerde agregarlo a `autoload_paths` :

```

# config/application.rb
config.autoload_paths << Rails.root.join('lib')

```

También puede preferir crear objetos de dominio más orientados a la acción, siguiendo el patrón

de Comando / Consulta. En tal caso, colocar estos objetos en la `app/commands` podría ser un lugar mejor, ya que todos `app` subdirectorios de la `app` se agregan automáticamente a la ruta de carga automática.

Lea Mejores Prácticas de Rieles en línea: <https://riptutorial.com/es/ruby-on-rails/topic/1207/mejores-practicas-de-rieles>

# Capítulo 53: Migraciones ActiveRecord

## Parámetros

Tipo de columna	Descripción
<code>:primary_key</code>	Clave primaria
<code>:string</code>	Cadena de datos más corta. Permite la opción de <code>limit</code> para el número máximo de caracteres.
<code>:text</code>	Mayor cantidad de texto. Permite la opción de <code>limit</code> para el número máximo de bytes.
<code>:integer</code>	Entero. Permite la opción de <code>limit</code> para el número máximo de bytes.
<code>:bigint</code>	Entero mayor
<code>:float</code>	Flotador
<code>:decimal</code>	Número decimal con precisión variable. Permite opciones de <code>precision</code> y <code>scale</code> .
<code>:numeric</code>	Permite opciones de <code>precision</code> y <code>scale</code> .
<code>:datetime</code>	Objeto DateTime para fechas / horas.
<code>:time</code>	Objeto de tiempo por tiempos.
<code>:date</code>	Objeto de fecha para las fechas.
<code>:binary</code>	Datos binarios. Permite la opción de <code>limit</code> para el número máximo de bytes.
<code>:boolean</code>	Booleano

## Observaciones

- La mayoría de los archivos de migración viven en `db/migrate/` directorio `db/migrate/`. Están identificados por una marca de tiempo UTC al principio de su nombre de archivo:  
`YYYYMMDDHHMMSS_create_products.rb`.
- El comando de `rails generate` se puede acortar a `rails g`.
- Si a `:type` no se pasa a un campo, por defecto es una cadena.

# Examples

## Ejecutar migración específica

Para ejecutar una migración específica hacia arriba o hacia abajo, use `db:migrate:up` o `db:migrate:down`.

Hasta una migración específica:

5.0

```
rake db:migrate:up VERSION=20090408054555
```

5.0

```
rails db:migrate:up VERSION=20090408054555
```

Abajo una migración específica:

5.0

```
rake db:migrate:down VERSION=20090408054555
```

5.0

```
rails db:migrate:down VERSION=20090408054555
```

El número de versión en los comandos anteriores es el prefijo numérico en el nombre de archivo de la migración. Por ejemplo, para migrar a la migración `20160515085959_add_name_to_users.rb`, usaría `20160515085959` como número de versión.

## Crear una tabla de unión

Para crear una tabla de unión entre `students` y `courses`, ejecute el comando:

```
$ rails g migration CreateJoinTableStudentCourse student course
```

Esto generará la siguiente migración:

```
class CreateJoinTableStudentCourse < ActiveRecord::Migration[5.0]
  def change
    create_join_table :students, :courses do |t|
      # t.index [:student_id, :course_id]
      # t.index [:course_id, :student_id]
    end
  end
end
```

## Ejecución de migraciones en diferentes entornos.

Para ejecutar migraciones en el entorno de `test` , ejecute este comando de shell:

```
rake db:migrate RAILS_ENV=test
```

## 5.0

Comenzando en Rails 5.0, puedes usar `rails` lugar de `rake` :

```
rails db:migrate RAILS_ENV=test
```

## Agregar una nueva columna a una tabla

Para agregar un nuevo `name` columna a la tabla de `users` , ejecute el comando:

```
rails generate migration AddNameToUsers name
```

Esto genera la siguiente migración:

```
class AddNameToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :name, :string
  end
end
```

Cuando el nombre de la migración tiene el formato `AddXXXXToTABLE_NAME` seguido de una lista de columnas con tipos de datos, la migración generada contendrá las declaraciones `add_column` apropiadas.

## Añadir una nueva columna con un índice.

Para agregar un nuevo `email` columna *indexada* a la tabla de `users` , ejecute el comando:

```
rails generate migration AddEmailToUsers email:string:index
```

Esto generará la siguiente migración:

```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :email, :string
    add_index :users, :email
  end
end
```

## Eliminar una columna existente de una tabla

Para eliminar el `name` columna existente de la tabla de `users` , ejecute el comando:

```
rails generate migration RemoveNameFromUsers name:string
```

Esto generará la siguiente migración:

```
class RemoveNameFromUsers < ActiveRecord::Migration[5.0]
  def change
    remove_column :users, :name, :string
  end
end
```

Cuando el nombre de la migración tiene el formato `RemoveXXXXFromYYY` seguido de una lista de columnas con tipos de datos, la migración generada contendrá las declaraciones de `remove_column` apropiadas.

Si bien no es necesario especificar el tipo de datos (por ejemplo, `:string`) como parámetro para `remove_column`, es altamente recomendable. Si *no* se especifica el tipo de datos, la migración no será reversible.

## Agregar una columna de referencia a una tabla

Para agregar una referencia a un `team` en la tabla de `users`, ejecute este comando:

```
$ rails generate migration AddTeamRefToUsers team:references
```

Esto genera la siguiente migración:

```
class AddTeamRefToUsers < ActiveRecord::Migration[5.0]
  def change
    add_reference :users, :team, foreign_key: true
  end
end
```

Esa migración creará una columna `team_id` en la tabla de `users`.

Si desea agregar un `index` apropiado y la `foreign_key` en la columna agregada, cambie el comando para que los `rails generate migration AddTeamRefToUsers team:references:index`. Esto generará la siguiente migración:

```
class AddTeamRefToUsers < ActiveRecord::Migration
  def change
    add_reference :users, :team, index: true
    add_foreign_key :users, :teams
  end
end
```

Si desea asignar un nombre a su columna de referencia que no sea la que Rails genera automáticamente, agregue lo siguiente a su migración: (Por ejemplo, puede llamar al `User` que creó la `Post` como `Author` en la tabla de `Post`)

```
class AddAuthorRefToPosts < ActiveRecord::Migration
  def change
    add_reference :posts, :author, references: :users, index: true
  end
end
```

```
end
```

## Crear una nueva tabla

Para crear una nueva tabla de `users` con el `name` y el `salary` las columnas, ejecute el comando:

```
rails generate migration CreateUsers name:string salary:decimal
```

Esto generará la siguiente migración:

```
class CreateUsers < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      t.string :name
      t.decimal :salary
    end
  end
end
```

Cuando el nombre de la migración tiene el formato `CreateXXX` seguido de una lista de columnas con tipos de datos, se generará una migración que crea la tabla `xxx` con las columnas enumeradas.

## Añadiendo múltiples columnas a una tabla

Para agregar varias columnas a una tabla, separe el `field:type` pares con espacios cuando use `rails generate migration comando de rails generate migration .`

La sintaxis general es:

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

Por ejemplo, lo siguiente agregará los campos de `name` , `salary` y `email` a la tabla de `users` :

```
rails generate migration AddDetailsToUsers name:string salary:decimal email:string
```

Lo que genera la siguiente migración:

```
class AddDetailsToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :name, :string
    add_column :users, :salary, :decimal
    add_column :users, :email, :string
  end
end
```

## Ejecutando migraciones

Ejecutar comando:

## 5.0

```
rake db:migrate
```

## 5.0

```
rails db:migrate
```

La especificación de la versión de destino ejecutará las migraciones necesarias (arriba, abajo, cambiar) hasta que haya alcanzado la versión especificada. Aquí, el `version number` es el prefijo numérico en el nombre de archivo de la migración.

## 5.0

```
rake db:migrate VERSION=20080906120000
```

## 5.0

```
rails db:migrate VERSION=20080906120000
```

## Migraciones de retroceso

Para `rollback` la migración más reciente, ya sea revirtiendo el método de `change` o ejecutando el método `down` . Ejecutar comando:

## 5.0

```
rake db:rollback
```

## 5.0

```
rails db:rollback
```

## Deshacer las últimas 3 migraciones

## 5.0

```
rake db:rollback STEP=3
```

## 5.0

```
rails db:rollback STEP=3
```

`STEP` proporcionar el número de migraciones para revertir.

## Deshacer todas las migraciones

## 5.0

```
rake db:rollback VERSION=0
```

## 5.0

```
rails db:rollback VERSION=0
```

### Mesas cambiantes

Si ha creado una tabla con algún esquema incorrecto, la forma más fácil de cambiar las columnas y sus propiedades es `change_table` . Revise el siguiente ejemplo:

```
change_table :orders do |t|
  t.remove :ordered_at # removes column ordered_at
  t.string :skew_number # adds a new column
  t.index :skew_number #creates an index
  t.rename :location, :state #renames location column to state
end
```

La migración anterior cambia una tabla de `orders` . Aquí hay una descripción línea por línea de los cambios:

1. `t.remove :ordered_at` elimina la columna `ordered_at` de las `orders` de la tabla.
2. `t.string :skew_number` agrega una nueva columna de tipo cadena llamada `skew_number` en la tabla de `orders` .
3. `t.index :skew_number` agrega un índice en la columna `skew_number` en la tabla de `orders` .
4. `t.rename :location, :state` cambia el nombre de la columna de `location` en la tabla de `orders` a `state` .

### Agregar una columna única a una tabla

Para agregar un nuevo `email` columna *única* a los `users` , ejecute el siguiente comando:

```
rails generate migration AddEmailToUsers email:string:uniq
```

Esto creará la siguiente migración:

```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :email, :string
    add_index :users, :email, unique: true
  end
end
```

### Cambiar el tipo de una columna existente

Para modificar una columna existente en Rails con una migración, ejecute el siguiente comando:

```
rails g migration change_column_in_table
```

Esto creará un nuevo archivo de `db/migration` directorio `db/migration` (si aún no existe), que contendrá el archivo con el prefijo con la marca de tiempo y el nombre del archivo de migración que contiene el contenido a continuación:

```
def change
  change_column(:table_name, :column_name, :new_type)
end
```

## Guía de rieles - Cambio de columnas

### Un método más largo pero más seguro.

El código anterior evita que el usuario pueda revertir la migración. Puede evitar este problema dividiendo el método de `change` en métodos separados hacia `up` y `down` :

```
def up
  change_column :my_table, :my_column, :new_type
end

def down
  change_column :my_table, :my_column, :old_type
end
```

### Rehacer migraciones

Puede revertir y luego migrar nuevamente usando el comando `redo` . Básicamente, se trata de un acceso directo que combina las tareas de `rollback` y `migrate` .

Ejecutar comando:

5.0

```
rake db:migrate:redo
```

5.0

```
rails db:migrate:redo
```

Puede usar el parámetro `STEP` para retroceder más de una versión.

Por ejemplo, para retroceder 3 migraciones:

5.0

```
rake db:migrate:redo STEP=3
```

5.0

```
rails db:migrate:redo STEP=3
```

### Añadir columna con valor por defecto

El siguiente ejemplo agrega una columna `admin` a la tabla de `users` y le da a esa columna el valor predeterminado `false` .

```
class AddDetailsToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :admin, :boolean, default: false
  end
end
```

Las migraciones con valores predeterminados pueden tardar mucho tiempo en tablas grandes, por ejemplo, con PostgreSQL. Esto se debe a que cada fila deberá actualizarse con el valor predeterminado para la columna recién agregada. Para evitar esto (y reducir el tiempo de inactividad durante las implementaciones), puede dividir su migración en tres pasos:

1. Agregue un `add_column` -migration similar al anterior, pero no establezca ningún valor predeterminado
2. Implemente y actualice la columna en una tarea de rake o en la consola mientras se ejecuta su aplicación. Asegúrese de que su aplicación ya escribe datos en esa columna para las filas nuevas / actualizadas.
3. Agregue otra migración de `change_column`, que luego cambia el valor predeterminado de esa columna al valor predeterminado deseado

## Prohibir valores nulos

Para prohibir `null` valores `null` en las columnas de su tabla, agregue el parámetro `:null` a su migración, como esto:

```
class AddPriceToProducts < ActiveRecord::Migration
  def change
    add_column :products, :float, null: false
  end
end
```

## Comprobando el estado de la migración

Podemos comprobar el estado de las migraciones ejecutando.

### 3.0 5.0

```
rake db:migrate:status
```

### 5.0

```
rails db:migrate:status
```

La salida se verá así:

Status	Migration ID	Migration Name
up	20140711185212	Create documentation pages
up	20140724111844	Create nifty attachments table
up	20140724114255	Create documentation screenshots
up	20160213170731	Create owners
up	20160218214551	Create users

```
up      20160221162159 ***** NO FILE *****
up      20160222231219 ***** NO FILE *****
```

Bajo el campo de estado, `up` significa que la migración se ha ejecutado y `down` significa que necesitamos ejecutar la migración.

## Crear una columna hstore

`Hstore` columnas de `Hstore` pueden ser útiles para almacenar configuraciones. Están disponibles en las bases de datos PostgreSQL después de habilitar la extensión.

```
class CreatePages < ActiveRecord::Migration[5.0]
  def change
    create_table :pages do |t|
      enable_extension 'hstore' unless extension_enabled?('hstore')
      t.hstore :settings
      t.timestamps
    end
  end
end
```

## Añadir una referencia propia

Una auto referencia puede ser útil para construir un árbol jerárquico. Esto se puede lograr con `add_reference` en una migración.

```
class AddParentPages < ActiveRecord::Migration[5.0]
  def change
    add_reference :pages, :pages
  end
end
```

La columna de clave externa será `pages_id`. Si desea decidir sobre el nombre de la columna de clave externa, primero debe crear la columna y luego agregar la referencia.

```
class AddParentPages < ActiveRecord::Migration[5.0]
  def change
    add_column :pages, :parent_id, :integer, null: true, index: true
    add_foreign_key :pages, :pages, column: :parent_id
  end
end
```

## Crear una columna de matriz

PostgreSQL admite una columna de `array`. Rails convertirá automáticamente una matriz PostgreSQL en una matriz Ruby, y viceversa.

Crear una tabla con una columna de `array`:

```
create_table :products do |t|
  t.string :name
```

```
t.text :colors, array: true, default: []
end
```

Agregue una columna de `array` a una tabla existente:

```
add_column :products, :colors, array: true, default: []
```

Agregue un índice para una columna de `array` :

```
add_index :products, :colors, using: 'gin'
```

## Agregar una restricción NOT NULL a los datos existentes

Supongamos que desea agregar una clave externa `company_id` a la tabla de `users` y desea tener una restricción `NOT NULL` en ella. Si ya tiene datos en los `users` , tendrá que hacerlo en varios pasos.

```
class AddCompanyIdToUsers < ActiveRecord::Migration
  def up
    # add the column with NULL allowed
    add_column :users, :company_id, :integer

    # make sure every row has a value
    User.find_each do |user|
      # find the appropriate company record for the user
      # according to your business logic
      company = Company.first
      user.update!(company_id: company.id)
    end

    # add NOT NULL constraint
    change_column_null :users, :company_id, false
  end

  # Migrations that manipulate data must use up/down instead of change
  def down
    remove_column :users, :company_id
  end
end
```

Lea Migraciones ActiveRecord en línea: <https://riptutorial.com/es/ruby-on-rails/topic/1087/migraciones-activerecord>

# Capítulo 54: Mongoide

## Examples

### Instalación

Primero, agregue `Mongoid` a su `Gemfile` :

```
gem "mongoid", "~> 4.0.0"
```

y luego ejecute `bundle install` . O simplemente ejecuta:

```
$ gem install mongoid
```

Después de la instalación, ejecute el generador para crear el archivo de configuración:

```
$ rails g mongoid:config
```

que creará el archivo `(myapp)/config/mongoid.yml` .

### Creando un modelo

Crea un modelo (llamémoslo `User` ) ejecutando:

```
$ rails g model User
```

que generará el archivo `app/models/user.rb` :

```
class User
  include Mongoid::Document

end
```

Esto es todo lo que necesita para tener un modelo (aunque nada más que un campo de `id` ). A diferencia de `ActiveRecord` , no hay archivos de migración. Toda la información de la base de datos para el modelo está contenida en el archivo del modelo.

Las marcas de tiempo no se incluyen automáticamente en su modelo cuando lo genera. Para agregar `created_at` y `updated_at` a su modelo, agregue

```
include Mongoid::Timestamps
```

debajo de tu modelo `include Mongoid::Document` como:

```
class User
  include Mongoid::Document
```

```
include Mongoid::Timestamps
end
```

## Campos

Según la [documentación de Mongoid](#) , hay 16 tipos de campos válidos:

- Formación
- BigDecimal
- Booleano
- Fecha
- Fecha y hora
- Flotador
- Picadillo
- Entero
- BSON :: ObjectId
- BSON :: Binario
- Distancia
- Regexp
- Cuerda
- Símbolo
- Hora
- TimeWithZone

Para agregar un campo (llamémoslo `name` y que sea una `String` ), agregue esto a su archivo modelo:

```
field :name, type: String
```

Para establecer un valor predeterminado, simplemente pase la opción `default` :

```
field :name, type: String, default: ""
```

## Asociaciones clásicas

Mongoid permite las asociaciones clásicas de `ActiveRecord` :

- Uno a uno: `has_one / belongs_to`
- Uno a muchos: `has_many / belongs_to`
- Muchos a muchos: `has_and_belongs_to_many`

Para agregar una asociación (digamos que el usuario tiene `has_many` publicaciones), puede agregar esto a su archivo de modelo de `User` :

```
has_many :posts
```

y esto a tu archivo de modelo de `Post` :

```
belongs_to :user
```

Esto agregará un campo `user_id` en su modelo de `Post` , agregará un método de `user` a su clase de `Post` y agregará un método de `posts` a su clase de `User` .

## Asociaciones incrustadas

Mongoid permite Asociaciones Incrustadas:

- Uno a uno: `embeds_one / embedded_in`
- Uno a muchos: `embeds_many / embedded_in`

Para agregar una asociación (digamos el usuario `embeds_many` direcciones), agregue esto a su archivo de `User` :

```
embeds_many :addresses
```

Y esto a su archivo de modelo de `Address` :

```
embedded_in :user
```

Esto incorporará la `Address` en su modelo de `User` , agregando un método de `addresses` a su clase de `User` .

## Llamadas a bases de datos

Mongoid intenta tener una sintaxis similar a `ActiveRecord` cuando puede. Soporta estas llamadas (y muchas más).

```
User.first #Gets first user from the database

User.count #Gets the count of all users from the database

User.find(params[:id]) #Returns the user with the id found in params[:id]

User.where(name: "Bob") #Returns a Mongoid::Criteria object that can be chained
                        #with other queries (like another 'where' or an 'any_in')
                        #Does NOT return any objects from database

User.where(name: "Bob").entries #Returns all objects with name "Bob" from database

User.where(:name.in => ['Bob', 'Alice']).entries #Returns all objects with name "Bob" or
" Alice" from database

User.any_in(name: ["Bob", "Joe"]).first #Returns the first object with name "Bob" or "Joe"
User.where(:name => 'Bob').exists? # will return true if there is one or more users with name
bob
```

Lea Mongoide en línea: <https://riptutorial.com/es/ruby-on-rails/topic/3071/mongoide>

---

# Capítulo 55: Motor de rieles - Rieles modulares

## Introducción

### Visión general rápida de los motores Rails

Los motores son pequeñas aplicaciones de Rails que se pueden usar para agregar funcionalidades a la aplicación que las aloja. La clase que define una aplicación de Ruby on Rails es `Rails::Application` que en realidad hereda gran parte de su comportamiento de `Rails::Engine`, la clase que define un motor. Podemos decir que una aplicación regular de Rails es simplemente un motor con más funciones.

## Sintaxis

- `plugin de rieles nuevo my_module --mountable`

## Examples

### Crear una aplicación modular

# Empezando

Primero, generemos una nueva aplicación de Ruby on Rails:

```
rails new ModularTodo
```

El siguiente paso es generar un motor!

```
cd ModularTodo && rails plugin new todo --mountable
```

También crearemos una carpeta de 'motores' para almacenar los motores (¡incluso si solo tenemos uno!).

```
mkdir engines && mv todo ./engines
```

Los motores, al igual que las gemas, vienen con un archivo `gemspec`. Pongamos algunos valores reales para evitar advertencias.

```
#ModularTodo/engines/todo/todo.gemspec
$:push File.expand_path("../lib", __FILE__)
```

```

#Maintain your gem's version:
require "todo/version"

#Describe your gem and declare its dependencies:
Gem::Specification.new do |s|
  s.name           = "todo"
  s.version        = Todo::VERSION
  s.authors        = ["Thibault Denizet"]
  s.email          = ["bo@samurails.com"]
  s.homepage       = "http://samurails.com"
  s.summary        = "Todo Module"
  s.description    = "Todo Module for Modular Rails article"
  s.license        = "MIT"

  #Moar stuff
  #...
end

```

Ahora necesitamos agregar el motor de Todo a la aplicación principal Gemfile.

```

#ModularTodo/Gemfile
#Other gems
gem 'todo', path: 'engines/todo'

```

Vamos a ejecutar `bundle install`. Deberías ver lo siguiente en la lista de gemas:

```
Using todo 0.0.1 from source at engines/todo
```

Genial, nuestro motor Todo está cargado correctamente! Antes de comenzar la codificación, tenemos una última cosa que hacer: montar el motor Todo. Podemos hacerlo en el archivo `route.rb` en la aplicación principal.

```

Rails.application.routes.draw do
  mount Todo::Engine => "/", as: 'todo'
end

```

Lo estamos montando en `/` pero también podríamos hacerlo accesible en `/todo`. Como solo tenemos un módulo, `/` está bien.

Ahora puede encender su servidor y verificarlo en su navegador. Debería ver la vista de Rails predeterminada porque aún no definimos ningún controlador / vista. ¡Vamos a hacer eso ahora!

## Construyendo la lista de Todo

Vamos a crear un modelo denominado `Task` dentro del módulo `Todo`, pero para migrar correctamente la base de datos desde la aplicación principal, debemos agregar un inicializador pequeño al archivo `engine.rb`.

```

#ModularTodo/engines/todo/lib/todo/engine.rb
module Todo

```

```

class Engine < ::Rails::Engine
  isolate_namespace Todo

  initializer :append_migrations do |app|
    unless app.root.to_s.match(root.to_s)
      config.paths["db/migrate"].expanded.each do |p|
        app.config.paths["db/migrate"] << p
      end
    end
  end
end

end
end

```

Eso es todo, ahora cuando ejecutamos migraciones desde la aplicación principal, las migraciones en el motor de Todo también se cargarán.

Vamos a crear el modelo de `Task`. El comando de `scaffold` debe ejecutarse desde la carpeta del motor.

```
cd engines/todo && rails g scaffold Task title:string content:text
```

Ejecuta las migraciones desde la carpeta padre:

```
rake db:migrate
```

Ahora, solo necesitamos definir la ruta raíz dentro del motor de Todo:

```

#ModularTodo/engines/todo/config/routes.rb
Todo::Engine.routes.draw do
  resources :tasks
  root 'tasks#index'
end

```

Puedes jugar con él, crear tareas, eliminarlas ... ¡Oh, espera, la eliminación no funciona! ¿Por qué?! Bueno, parece que JQuery no está cargado, ¡así que vamos a agregarlo al archivo `application.js` dentro del motor!

```

// ModularTodo/engines/todo/app/assets/javascripts/todo/application.js
//= require jquery
//= require jquery_ujs
//= require_tree .

```

Yay, ahora podemos destruir tareas!

Lea Motor de rieles - Rieles modulares en línea: <https://riptutorial.com/es/ruby-on-rails/topic/9080/motor-de-rieles---rieles-modulares>

---

# Capítulo 56: Oleoducto de activos

## Introducción

El flujo de activos proporciona un marco para concatenar y minimizar o comprimir los activos de JavaScript y CSS. También agrega la capacidad de escribir estos activos en otros lenguajes y preprocesadores como CoffeeScript, Sass y ERB. Permite que los activos en tu aplicación se combinen automáticamente con los activos de otras gemas. Por ejemplo, jquery-rails incluye una copia de jquery.js y habilita las funciones AJAX en Rails.

## Examples

### Tareas de rastrillo

Por defecto, las `sprockets-rails` se envían con las siguientes tareas de rake:

- `assets:clean[keep]` : eliminar activos compilados antiguos
- `assets:clobber` : eliminar los activos compilados
- `assets:environment` : cargar entorno de compilación de activos
- `assets:precompile` : compile todos los activos nombrados en `config.assets.precompile`

### Archivos y directivas de manifiesto

En el initalizer de `assets` ( `config/initializers/assets.rb` ) hay algunos archivos definidos explícitamente para ser precompilados.

```
# Precompile additional assets.
# application.coffee, application.scss, and all non-JS/CSS in app/assets folder are already
added.
# Rails.application.config.assets.precompile += %w( search.js )
```

En este ejemplo, `application.coffee` y `application.scss` se denominan 'Archivos de manifiesto'. Estos archivos deben usarse para incluir otros recursos de JavaScript o CSS. Los siguientes comandos están disponibles:

- `require <path>` : las funciones de la directiva `require` similares a las de Ruby `require` . Proporciona una forma de declarar una dependencia de un archivo en su ruta y asegura que solo se cargue una vez antes del archivo de origen.
- `require_directory <path>` : requiere todos los archivos dentro de un solo directorio. Es similar a la `path/*` ya que no sigue directorios anidados.
- `require_tree <path>` : requiere todos los archivos anidados en un directorio. Su equivalente global es `path/**/*` .
- `require_self` : hace que el cuerpo del archivo actual se inserte antes de cualquier directiva de `require` posterior. Útil en archivos CSS, donde es común que el archivo de índice contenga estilos globales que deben definirse antes de que se carguen otras dependencias.
- `stub <path>`

: elimina un archivo para que no se incluya

- `depend_on <path>` : le permite establecer una dependencia en un archivo sin incluirlo. Esto se utiliza para fines de almacenamiento en caché. Cualquier cambio realizado en el archivo de dependencia invalidará la memoria caché del archivo de origen.

Un archivo `application.scss` podría verse como:

```
/*
 *= require bootstrap
 *= require_directory .
 *= require_self
 */
```

Otro ejemplo es el archivo `application.coffee` . Aquí con la inclusión de `jquery` y `Turbolinks` :

```
#= require jquery2
#= require jquery_ujs
#= require turbolinks
#= require_tree .
```

Si no utilizas CoffeeScript, pero solo JavaScript, la sintaxis sería:

```
//= require jquery2
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

## Uso básico

Hay dos formas básicas en que se utiliza el flujo de activos:

1. Al ejecutar un servidor en modo de desarrollo, automáticamente procesa y prepara sus activos sobre la marcha.
2. En el modo de producción, probablemente lo utilice para preprocesar, versionar, comprimir y compilar sus activos. Puedes hacerlo ejecutando el siguiente comando:

```
bundle exec rake assets:precompile
```

Lea **Oleoducto de activos en línea**: <https://riptutorial.com/es/ruby-on-rails/topic/3386/oleoducto-de-activos>

---

# Capítulo 57: Organización de la clase

## Observaciones

Esto parece algo simple de hacer, pero cuando tus clases comienzan a aumentar de tamaño, estarás agradecido de que te hayas tomado el tiempo de organizarlas.

## Examples

### Clase de modelo

```
class Post < ActiveRecord::Base
  belongs_to :user
  has_many :comments

  validates :user, presence: true
  validates :title, presence: true, length: { in: 6..40 }

  scope :topic, -> (topic) { joins(:topics).where(topic: topic) }

  before_save :update_slug
  after_create :send_welcome_email

  def publish!
    update(published_at: Time.now, published: true)
  end

  def self.find_by_slug(slug)
    find_by(slug: slug)
  end

  private

  def update_slug
    self.slug = title.join('-')
  end

  def send_welcome_email
    WelcomeMailer.welcome(self).deliver_now
  end
end
```

Los modelos suelen ser responsables de:

- establecer relaciones
- validando datos
- Proporcionar acceso a datos a través de ámbitos y métodos.
- Realización de acciones en torno a la persistencia de datos.

En el nivel más alto, los modelos describen conceptos de dominio y gestionan su persistencia.

## Clase de servicio

El controlador es un punto de entrada a nuestra aplicación. Sin embargo, no es el único punto de entrada posible. Me gustaría tener mi lógica accesible desde:

- Tareas de rastrillo
- trabajos de fondo
- consola
- pruebas

Si lanzo mi lógica en un controlador, no será accesible desde todos estos lugares. Así que probemos el enfoque de "controlador delgado, modelo gordo" y pasemos la lógica a un modelo. ¿Pero cual? Si una determinada pieza de lógica involucra modelos de `User`, `Cart` y `Product`, ¿dónde debería vivir?

Una clase que hereda de `ActiveRecord::Base` ya tiene muchas responsabilidades. Maneja interfaz de consulta, asociaciones y validaciones. Si agrega aún más código a su modelo, se convertirá rápidamente en un desorden inigualable con cientos de métodos públicos.

Un servicio es solo un objeto regular de Ruby. Su clase no tiene que heredar de ninguna clase específica. Su nombre es una frase verbal, por ejemplo, `CreateUserAccount` lugar de `UserCreation` o `UserCreationService`. Vive en el directorio de aplicaciones / servicios. Tienes que crear este directorio por ti mismo, pero Rails cargará automáticamente las clases dentro de ti.

### Un objeto de servicio hace una cosa.

Un objeto de servicio (también conocido como objeto de método) realiza una acción. Mantiene la lógica empresarial para realizar esa acción. Aquí hay un ejemplo:

```
# app/services/accept_invite.rb
class AcceptInvite
  def self.call(invite, user)
    invite.accept!(user)
    UserMailer.invite_accepted(invite).deliver
  end
end
```

Las tres convenciones que sigo son:

Los servicios van bajo el `app/services` directory. Le animo a usar subdirectorios para los dominios pesados de la lógica de negocios. Por ejemplo:

- El archivo `app/services/invite/accept.rb` definirá `Invite::Accept` mientras que `app/services/invite/create.rb` definirá `Invite::Create`
- Los servicios comienzan con un verbo (y no terminan con el servicio): `ApproveTransaction`, `SendTestNewsletter`, `ImportUsersFromCsv`
- Los servicios responden al método de `call`. Descubrí que usar otro verbo lo hace un poco redundante: `ApproveTransaction.approve()` no lee bien. Además, el método de `call` es el método de facto para `lambda`, `procs` y métodos de objetos.

## Beneficios

### *Los objetos de servicio muestran lo que hace mi aplicación*

Solo puedo ver el directorio de servicios para ver qué hace mi aplicación: `ApproveTransaction` , `CancelTransaction` , `BlockAccount` , `SendTransactionApprovalReminder` ...

Una mirada rápida a un objeto de servicio y sé en qué consiste la lógica de negocios. No tengo que pasar por los controladores, las devoluciones de llamada del modelo `ActiveRecord` y los observadores para comprender lo que implica "aprobar una transacción".

### *Modelos de limpieza y controladores.*

Los controladores convierten la solicitud (parámetros, sesión, cookies) en argumentos, los transmiten al servicio y los redirigen o representan según la respuesta del servicio.

```
class InviteController < ApplicationController
  def accept
    invite = Invite.find_by_token!(params[:token])
    if AcceptInvite.call(invite, current_user)
      redirect_to invite.item, notice: "Welcome!"
    else
      redirect_to '/', alert: "Oopsy!"
    end
  end
end
```

Los modelos solo tratan con asociaciones, ámbitos, validaciones y persistencia.

```
class Invite < ActiveRecord::Base
  def accept!(user, time=Time.now)
    update_attributes!(
      accepted_by_user_id: user.id,
      accepted_at: time
    )
  end
end
```

¡Esto hace que los modelos y los controladores sean mucho más fáciles de probar y mantener!

### **Cuándo usar la clase de servicio**

Alcance los objetos de servicio cuando una acción cumpla con uno o más de estos criterios:

- La acción es compleja (p. Ej., Cerrar los libros al final de un período contable)
- La acción se extiende a través de múltiples modelos (por ejemplo, una compra de comercio electrónico utilizando los objetos `Order`, `CreditCard` y `Customer`)
- La acción interactúa con un servicio externo (p. Ej., Publicación en redes sociales)
- La acción no es una preocupación central del modelo subyacente (por ejemplo, barriendo los datos desactualizados después de un cierto período de tiempo).
- Hay varias formas de realizar la acción (por ejemplo, autenticarse con un token de acceso o contraseña).

## Fuentes

[Blog de Adam Niedzielski](#)

[Blog de Brew House](#)

[Código Clima Blog](#)

Lea **Organizacion de la clase en línea**: <https://riptutorial.com/es/ruby-on-rails/topic/7623/organizacion-de-la-clase>

---

# Capítulo 58: Palabras reservadas

## Introducción

Debe tener cuidado al usar estas palabras para la variable, nombre del modelo, nombre del método o etc.

## Examples

### Lista de palabras reservadas

- ADDITIONAL\_LOAD\_PATHS
- ARGF
- ARGV
- Controlador de acción
- ActionView
- ActiveRecord
- ArgumentError
- Formación
- BasicSocket
- Punto de referencia
- Bignum
- Unión
- CGI
- CGIMethods
- CROSS\_COMPILING
- Clase
- ClassInheritableAttributes
- Comparable
- Condiciónvariable
- Config
- Continuación
- DRb
- DRbIdConv
- DRbObject
- DRbUndumped
- Datos
- Fecha
- Fecha y hora
- Delegador
- Delegador
- Digerir
- Dir
- ENV

- EOFError
- ERB
- Enumerable
- Errno
- Excepción
- FALSO
- FalseClass
- Fcntl
- Expediente
- FileList
- FileTask
- Prueba de Archivo
- FileUtils
- Arreglo
- Flotador
- FloatDomainError
- GC
- Joya
- GetoptLong
- Picadillo
- IO
- IOError
- IPSocket
- IPsocket
- IndexError
- Inflector
- Entero
- Interrumpir
- Núcleo
- LN\_SUPPORTED
- Error de carga
- LocalJumpError
- Logger
- Mariscal
- MatchData
- Datos coincidentes
- Mates
- Método
- Módulo
- Mutex
- Mysql
- Error de MySQL
- MysqlField
- MysqlRes
- NULO
- NameError

- NilClass
- NoMemoryError
- NoMethodError
- NoWrite
- NotImplementedError
- Numérico
- OPT\_TABLE
- Objeto
- Espacio de objetos
- Observable
- Observador
- PGError
- PGconn
- Paje
- PGresult
- PLATAFORMA
- PStore
- Fecha de análisis
- Precisión
- Proc
- Proceso
- Cola
- RAKEVERSION
- FECHA DE LANZAMIENTO
- RUBÍ
- RUBY\_PLATFORM
- RUBY\_RELEASE\_DATE
- RUBY\_VERSION
- Estante
- Rastrillo
- RakeApp
- RakeFileUtils
- Distancia
- RangeError
- Racional
- Regexp
- RegexpError
- Solicitud
- Error de tiempo de ejecución
- STDERR
- STDIN
- Repartir
- Error de exploración
- Error de guión
- Error de seguridad
- Señal

- SignalException
- SimpleDelegater
- SimpleDelegator
- Semifallo
- SizedQueue
- Enchufe
- Error de socket
- Error estándar
- Cuerda
- StringScanner
- Struct
- Símbolo
- Error de sintaxis
- SystemCallError
- SystemExit
- SystemStackError
- TCPServer
- TCPsocket
- Servidor TCP
- TCPsocket
- TOPLEVEL\_BINDING
- CIERTO
- Tarea
- Texto
- Hilo
- ThreadError
- ThreadGroup
- Hora
- Transacción
- TrueClass
- Error de teclado
- UDPSocket
- UDPsocket
- Servidor UNIX
- UNIXSocket
- Servidor UNIX
- UNIXsocket
- UnboundMethod
- Url
- VERSIÓN
- Verboso
- YAML
- ZeroDivisionError
- @base\_path
- aceptar
- Acces

- Axi
- acción
- atributos
- aplicacion2
- llamar de vuelta
- categoría
- conexión
- base de datos
- transportista
- display1
- conducir
- errores
- formato
- anfitrión
- llave
- diseño
- carga
- enlazar
- nuevo
- notificar
- abierto
- público
- citar
- hacer
- solicitud
- archivos
- respuestas
- salvar
- alcance
- enviar
- sesión
- sistema
- modelo
- prueba
- se acabó el tiempo
- to\_s
- tipo
- URI
- visitas
- Observador

### **Nombres de campo de la base de datos**

- Creado en
- creado en
- updated\_at
- actualizado en

- deleted\_at
- (paranoia
- joya)
- versión de bloqueo
- tipo
- carné de identidad
- # {table\_name} \_count
- posición
- Identificación de los padres
- lft
- rgt
- cotización\_valor

## **Ruby Reserved Words**

- alias
- y
- EMPEZAR
- empezar
- descanso
- caso
- clase
- def
- definido?
- hacer
- más
- elsif
- FIN
- fin
- asegurar
- falso
- para
- Si
- módulo
- siguiente
- nulo
- no
- o
- rehacer
- rescate
- procesar de nuevo
- regreso
- yo
- súper
- entonces
- cierto
- undef

- a no ser que
- hasta
- cuando
- mientras
- rendimiento
- `_ ARCHIVO _`
- `_ LINEA _`

Lea Palabras reservadas en línea: <https://riptutorial.com/es/ruby-on-rails/topic/10818/palabras-reservadas>

# Capítulo 59: Patrón decorador

## Observaciones

El **patrón Decorator** le permite agregar o modificar el comportamiento de los objetos de una manera coyuntural sin afectar el objeto base.

Esto se puede lograr a través de Ruby usando el `stdlib`, o por medio de gemas populares como [Draper](#).

## Examples

### Decorar un modelo utilizando SimpleDelegator

La mayoría de los desarrolladores de Rails comienzan modificando la información de su modelo dentro de la propia plantilla:

```
<h1><%= "#{ @user.first_name } #{ @user.last_name }" %></h1>
<h3>joined: <%= @user.created_at.in_time_zone(current_user.timezone).strftime("%A, %d %b %Y
%l:%M %p") %></h3>
```

Para modelos con una gran cantidad de datos, esto puede volverse engorroso rápidamente y llevar a copiar y pegar la lógica de una plantilla a otra.

Este ejemplo utiliza `SimpleDelegator` de la `stdlib`.

Todas las solicitudes a un objeto `SimpleDelegator` se pasan al objeto principal de forma predeterminada. Puede anular cualquier método con lógica de presentación, o puede agregar nuevos métodos que sean específicos para esta vista.

`SimpleDelegator` proporciona dos métodos: `__setobj__` para establecer a qué objeto se está delegando, y `__getobj__` para obtener ese objeto.

```
class UserDecorator < SimpleDelegator
  attr_reader :view
  def initialize(user, view)
    __setobj__ @user
    @view = view
  end

  # new methods can call methods on the parent implicitly
  def full_name
    "#{ first_name } #{ last_name }"
  end

  # however, if you're overriding an existing method you need
  # to use __getobj__
  def created_at
    Time.use_zone(view.current_user.timezone) do
```

```
    __getobj__.created_at.strftime("%A, %d %b %Y %l:%M %p")
  end
end
end
```

Algunos decoradores confían en la magia para conectar este comportamiento, pero puede hacer que sea más obvio de dónde proviene la lógica de presentación al inicializar el objeto en la página.

```
<% user = UserDecorator.new(@user, self) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

Al pasar una referencia al objeto de vista al decorador, aún podemos acceder al resto de los asistentes de vista mientras construimos la lógica de presentación sin tener que incluirlo.

Ahora, la plantilla de vista solo se ocupa de insertar datos en la página, y es mucho más claro.

## Decorando un modelo usando Draper

Draper combina automáticamente los modelos con sus decoradores por convención.

```
# app/decorators/user_decorator.rb
class UserDecorator < Draper::Decorator
  def full_name
    "#{object.first_name} #{object.last_name}"
  end

  def created_at
    Time.use_zone(h.current_user.timezone) do
      object.created_at.strftime("%A, %d %b %Y %l:%M %p")
    end
  end
end
```

Dada una variable `@user` que contiene un objeto ActiveRecord, puede acceder a su decorador llamando a `#decorate` en `@user`, o especificando la clase Draper si desea ser específico.

```
<% user = @user.decorate %><!-- OR -->
<% user = UserDecorator.decorate(@user) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

Lea Patrón decorador en línea: <https://riptutorial.com/es/ruby-on-rails/topic/5694/patron-decorador>

# Capítulo 60: Prawn PDF

## Examples

### Ejemplo avanzado

Este es el enfoque avanzado con el ejemplo.

```
class FundsController < ApplicationController

  def index
    @funds = Fund.all_funds(current_user)
  end

  def show
    @fund = Fund.find(params[:id])
    respond_to do |format|
      format.html
      format.pdf do
        pdf = FundsPdf.new(@fund, view_context)
        send_data pdf.render, filename:
          "fund_#{@fund.created_at.strftime("%d/%m/%Y")}.pdf",
          type: "application/pdf"
      end
    end
  end
end
```

Por encima del código tenemos esta línea `FundsPdf.new(@fund, view_context)`. Aquí estamos inicializando la clase `FundsPdf` con la instancia de `@fund` y `view_context` para usar los métodos de ayuda en `FundsPdf`. `FundsPdf` se vería así

```
class FundPdf < Prawn::Document

  def initialize(fund, view)
    super()
    @fund = fund
    @view = view
    upper_half
    lower_half
  end

  def upper_half
    logopath = "#{Rails.root}/app/assets/images/logo.png"
    image logopath, :width => 197, :height => 91
    move_down 10
    draw_text "Receipt", :at => [220, 575], size: 22
    move_down 80
    text "Hello #{@invoice.customer.profile.first_name.capitalize},"
  end

  def thanks_message
    move_down 15
    text "Thank you for your order.Print this receipt as
```

```
confirmation of your order.",
  :indent_paragraphs => 40, :size => 13
end
end
```

Este es uno de los mejores métodos para generar PDF con clases usando la gema Prawn.

## Ejemplo básico

Debe agregar Gem y PDF MIME: Escriba dentro de mime\_types.rb, ya que debemos notificar a los rieles sobre el tipo de mime PDF.

Después de eso podemos generar Pdf con Prawn en las siguientes formas básicas

---

## Esta es la tarea básica.

```
pdf = Prawn::Document.new
pdf.text "Hello World"
pdf.render_file "assignment.pdf"
```

---

## Podemos hacerlo con Implícito Bloque.

```
Prawn::Document.generate("implicit.pdf") do
  text "Hello World"
end
```

---

## Con bloque explícito

```
Prawn::Document.generate("explicit.pdf") do |pdf|
  pdf.text "Hello World"
end
```

Lea Prawn PDF en línea: <https://riptutorial.com/es/ruby-on-rails/topic/4163/prawn-pdf>

---

# Capítulo 61: Puntos de vista

## Examples

### Parciales

Las plantillas parciales (parciales) son una forma de dividir el proceso de representación en partes más manejables. Los parciales le permiten extraer fragmentos de código de sus plantillas en archivos separados y también reutilizarlos en todas sus plantillas.

Para *crear* un parcial, cree un nuevo archivo que comience con un guión bajo: `_form.html.erb`

Para *representar* un parcial como parte de una vista, use el método de procesamiento dentro de la vista: `<%= render "form" %>`

- Tenga en cuenta, el subrayado se deja de lado cuando la representación
- Se debe representar un parcial usando su ruta si se encuentra en una carpeta diferente

Para *pasar* una variable al parcial como una variable local, use esta notación:

```
<%= render :partial => 'form', locals: { post: @post } %>
```

Los parciales también son útiles cuando necesita *reutilizar* exactamente el mismo código ( **filosofía DRY** ).

Por ejemplo, para reutilizar el código `<head>` , cree un parcial llamado `_html_header.html.erb` , ingrese su código `<head>` para ser reutilizado, y haga el parcial cuando sea necesario: `<%= render 'html_header' %>` .

---

## Objetos parciales

Los objetos que responden a `to_partial_path` también se pueden representar, como en `<%= render @post %>` . De forma predeterminada, para los modelos ActiveRecord, esto será algo así como `posts/post` , por lo que al renderizar `@post` , se `views/posts/_post.html.erb` las `views/posts/_post.html.erb` archivo `views/posts/_post.html.erb` .

Una `post` local con nombre será asignada automáticamente. Al final, `<%= render @post %>` es una mano corta para `<%= render 'posts/post', post: @post %>` .

También se pueden proporcionar colecciones de objetos que responden a `to_partial_path` , como `<%= render @posts %>` . Cada elemento será renderizado consecutivamente.

---

## Parciales globales

Para crear un parcial global que pueda usarse en cualquier lugar sin hacer referencia a su ruta

exacta, el parcial debe ubicarse en la ruta de las `views/application` . El ejemplo anterior se ha modificado a continuación para ilustrar esta característica.

Por ejemplo, esta es una ruta a una `app/views/application/_html_header.html.erb`: **parcial global** `app/views/application/_html_header.html.erb`:

Para representar este parcial global en cualquier lugar, use `<%= render 'html_header' %>`

## AssetTagHelper

Para permitir que los rieles vinculen de forma automática y correcta los recursos (css / js / images) en la mayoría de los casos, usted quiere usar ayudantes integrados. ( [Documentación oficial](#) )

---

# Ayudantes de imagen

## *ruta de la imagen*

Esto devuelve la ruta a un recurso de imagen en `app/assets/images` **asset** `app/assets/images` .

```
image_path("edit.png") # => /assets/edit.png
```

## *URL de la imagen*

Esto devuelve la URL completa a un activo de imagen en `app/assets/images` **asset** `app/assets/images` .

```
image_url("edit.png") # => http://www.example.com/assets/edit.png
```

## *etiqueta imagen*

Utilice este asistente si desea incluir una `<img src="" />` con el conjunto de fuentes.

```
image_tag("icon.png") # => 
```

---

# Ayudantes de JavaScript

## *javascript\_include\_tag*

Si desea incluir un archivo JavaScript en su vista.

```
javascript_include_tag "application" # => <script src="/assets/application.js"></script>
```

## ***javascript\_path***

Esto devuelve la ruta de su archivo JavaScript.

```
javascript_path "application" # => /assets/application.js
```

## ***javascript\_url***

Esto devuelve la URL completa de su archivo JavaScript.

```
javascript_url "application" # => http://www.example.com/assets/application.js
```

---

# Ayudantes de hojas de estilo

## ***stylesheet\_link\_tag***

Si desea incluir un archivo CSS en su vista.

```
stylesheet_link_tag "application" # => <link href="/assets/application.css" media="screen"
rel="stylesheet" />
```

## ***stylesheet\_path***

Esto devuelve la ruta de su activo hoja de estilo.

```
stylesheet_path "application" # => /assets/application.css
```

## ***stylesheet\_url***

Esto devuelve la URL completa de su elemento de hoja de estilo.

```
stylesheet_url "application" # => http://www.example.com/assets/application.css
```

---

# Ejemplo de uso

Al crear una nueva aplicación de Rails, automáticamente tendrá dos de estos ayudantes en `app/views/layouts/application.html.erb`

```
<%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track': 'reload' %>
<%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
```

Esto produce:

```
// CSS
<link rel="stylesheet" media="all" href="/assets/application.self-
e19d4b856cacba4f6fb0e5aa82a1ba9aa4ad616f0213a1259650b281d9cf6b20.css?body=1" data-turbolinks-
track="reload" />
// JavaScript
<script src="/assets/application.self-
619d9bf310b8eb258c67de7af745cafbf2a98f6d4c7bb6db8e1b00aed89eb0b1.js?body=1" data-turbolinks-
track="reload"></script>
```

## Estructura

A medida que Rails sigue el patrón **M V C**, las `Views` son donde están sus "plantillas" para sus acciones.

Digamos que tienes un controlador `articles_controller.rb`. Para este controlador, tendría una carpeta en vistas llamada `app/views/articles`:

```
app
|-- controllers
|   |-- articles_controller.rb
|
'-- views
    |-- articles
    |   |-- index.html.erb
    |   |-- edit.html.erb
    |   |-- show.html.erb
    |   |-- new.html.erb
    |   |-- _partial_view.html.erb
    |
    '-- [...]
```

Esta estructura le permite tener una carpeta para cada controlador. Al llamar a una acción en su controlador, la vista correspondiente se representará automáticamente.

```
// articles_controller.rb
class ArticlesController < ActionController::Base
  def show
    end
end

// show.html.erb
<h1>My show view</h1>
```

## Reemplazar el código HTML en las vistas

Si alguna vez quiso determinar el contenido html que se imprimirá en una página durante el tiempo de ejecución, Rails tiene una muy buena solución para eso. Tiene algo que se llama **content\_for**, que nos permite pasar un bloque a una vista de rieles. Por favor, consulte el siguiente ejemplo,

### Declarar el contenido

```
<div>
  <%= yield :header %>
</div>

<% content_for :header do %>
  <ul>
    <li>Line Item 1</li>
    <li>Line Item 2</li>
  </ul>
<% end %>
```

## HAML - una forma alternativa de usar en tus vistas

HAML (lenguaje de marcado de abstracción HTML) es una forma hermosa y elegante de describir y diseñar el HTML de sus vistas. En lugar de abrir y cerrar etiquetas, HAML usa sangría para la estructura de sus páginas. Básicamente, si se debe colocar algo dentro de otro elemento, solo se debe sangrar utilizando una tabulación. Las pestañas y el espacio en blanco son importantes en HAML, así que asegúrese de usar siempre la misma cantidad de pestañas.

### Ejemplos:

```
#myview.html.erb
<h1><%= @the_title %></h1>
<p>This is my form</p>
<%= render "form" %>
```

### Y en HAML:

```
#myview.html.haml
%h1= @the_title
%p
  This is my form
= render 'form'
```

Verás, la estructura del diseño es mucho más clara que usar HTML y ERB.

## Instalación

Solo instala la gema usando

```
gem install haml
```

y agregar la gema al Gemfile

```
gem "haml"
```

Para usar HAML en lugar de HTML / ERB, simplemente reemplace las extensiones de archivo de sus vistas desde `something.html.erb` a `something.html.haml` .

## Consejos rápidos

Los elementos comunes como divs se pueden escribir de una manera corta

## HTML

```
<div class="myclass">My Text</div>
```

## HAML

```
%div.myclass
```

## HAML, taquigrafía

```
.myclass
```

## Atributos

### HTML

```
<p class="myclass" id="myid">My paragraph</p>
```

### HAML

```
%p{:class => "myclass", :id => "myid"} My paragraph
```

## Insertando código rubí

Puede insertar código ruby con los signos = y -.

```
= link_to "Home", home_path
```

El código que comienza con = se ejecutará e incrustará en el documento.

Código que comienza con : se ejecutará, pero no se insertará en el documento.

## Documentación completa

Es muy fácil comenzar con HAML, pero también es muy complejo, así que recomiendo [leer la documentación](#) .

Lea **Puntos de vista en línea**: <https://riptutorial.com/es/ruby-on-rails/topic/850/puntos-de-vista>

# Capítulo 62: Rails Cookbook - Recetas / aprendizajes avanzados y técnicas de codificación

## Examples

Jugando con mesas utilizando la consola de rieles.

### Ver tablas

```
ActiveRecord::Base.connection.tables
```

### Eliminar cualquier tabla .

```
ActiveRecord::Base.connection.drop_table("users")
-----OR-----
ActiveRecord::Migration.drop_table(:users)
-----OR-----
ActiveRecord::Base.connection.execute("drop table users")
```

### Eliminar índice de la columna existente

```
ActiveRecord::Migration.remove_index(:users, :name => 'index_users_on_country')
```

donde `country` es un nombre de columna en el archivo de migración con un índice **ya** agregado en la tabla de `users` como se muestra a continuación:

```
t.string :country, add_index: true
```

### Eliminar restricción de clave externa

```
ActiveRecord::Base.connection.remove_foreign_key('food_items', 'menus')
```

donde los `menus has_many food_items` y `menus has_many food_items` y sus respectivas migraciones también.

### Añadir columna

```
ActiveRecord::Migration.remove_column :table_name, :column_name
```

por ejemplo:-

```
ActiveRecord::Migration.add_column :profiles, :profile_likes, :integer, :default => 0
```

## Métodos de rieles - devolviendo valores booleanos

Cualquier método en el modelo de Rails puede devolver un valor booleano.

### método simple

```
##this method return ActiveRecord::Relation
def check_if_user_profile_is_complete
  User.includes(:profile_pictures,:address,:contact_detail).where("user.id = ?",self)
end
```

### De nuevo, un método simple que devuelve valor booleano.

```
##this method return Boolean(NOTE THE !! signs before result)
def check_if_user_profile_is_complete
  !!User.includes(:profile_pictures,:address,:contact_detail).where("user.id = ?",self)
end
```

Entonces, el mismo método ahora devolverá booleano en lugar de cualquier otra cosa :).

## Manejo del error - método indefinido `donde` para #

A veces queremos usar una consulta `where` en una colección de registros devueltos que no es `ActiveRecord::Relation` modo, obtenemos el error anterior, ya que la cláusula `where` es conocida para `ActiveRecord` y no para `Array`.

Hay una solución precisa para esto utilizando `Joins`.

### EJEMPLO : -

Supongamos que necesito encontrar todos los perfiles de usuario (`UserProfile`) que estén activos y no sean usuarios (`User`) con un `id = 10`.

```
UserProfiles.includes(:user=>:profile_pictures).where(:active=>true).map(&:user).where.not(:id=>10)
```

Por lo tanto, la consulta anterior fallará después de que el `map` como `map` devuelva una `array` que **no** funcionará con la cláusula `where`.

### Pero usar uniones, lo hará funcionar,

```
UserProfiles.includes(:user=>:profile_pictures).where(:active=>true).joins(:user).where.not(:id=>10)
```

Dado que las `joins` generarán registros similares como `map` pero serán `ActiveRecord` y **no** un `Array`.

[Lea Rails Cookbook - Recetas / aprendizajes avanzados y técnicas de codificación en línea:](https://riptutorial.com/es/ruby-on-rails/topic/7259/rails-cookbook---recetas---aprendizajes-avanzados-y-tecnicas-de-codificacion)  
<https://riptutorial.com/es/ruby-on-rails/topic/7259/rails-cookbook---recetas---aprendizajes-avanzados-y-tecnicas-de-codificacion>

---

# Capítulo 63: Reacciona con los rieles usando la gema `react-rails`.

## Examples

### Reactuar la instalación para los rieles usando `rails_react` gema

Agregue `react-rails` a su Gemfile:

```
gem 'react-rails'
```

E instalar:

```
bundle install
```

A continuación, ejecute el script de instalación:

```
rails g react:install
```

Esta voluntad:

Cree un archivo de manifiesto `components.js` y un directorio `app / asset / javascripts / components /`, donde colocará sus componentes en su `application.js`:

```
//= require react
//= require react_ujs
//= require components
```

## Usando `react_rails` dentro de tu aplicación

### React.js construye

Puede elegir qué compilación React.js (desarrollo, producción, con o sin complementos) para servir en cada entorno agregando una configuración. Aquí están los valores por defecto:

```
# config/environments/development.rb
MyApp::Application.configure do
  config.react.variant = :development
end

# config/environments/production.rb
MyApp::Application.configure do
  config.react.variant = :production
end
```

Para incluir complementos, use esta configuración:

```
MyApp::Application.configure do
  config.react.addons = true # defaults to false
end
```

Después de reiniciar su servidor Rails, `// = require react` proporcionará la compilación de React.js que fue especificada por las configuraciones.

react-rails ofrece algunas otras opciones para versiones y compilaciones de React.js. Consulte [VERSIONS.md](#) para obtener más información sobre el uso de la gema `reaccion-source` o cómo descargar sus propias copias de React.js.

## JSX

Después de instalar react-rails, reinicie su servidor. Ahora, los archivos `.js.jsx` se transformarán en la canalización de activos.

### Opciones de BabelTransformer

Puede usar los transformadores y los complementos personalizados de Babel, y pasar las opciones al transpiler de Babel agregando las siguientes configuraciones:

```
config.react.jsx_transform_options = {
  blacklist: ['spec.functionName', 'validation.react', 'strict'], # default options
  optional: ["transformerName"], # pass extra babel options
  whitelist: ["useStrict"] # even more options[enter link description here][1]
}
```

Bajo el capó, react-rails utiliza [ruby-babel-transpiler](#), para la transformación.

## Renderizado y montaje

react-rails incluye un asistente de vista (`react_component`) y un controlador de JavaScript discreto (`react_ujs`) que trabajan juntos para poner los componentes de React en la página. Debería requerir el controlador UJS en su manifiesto después de reaccionar (y después de turbolinks si usa Turbolinks).

El asistente de visualización coloca un `div` en la página con la clase de componente y accesorios solicitados. Por ejemplo:

```
<%= react_component('HelloMessage', name: 'John') %>
<!-- becomes: -->
<div data-react-class="HelloMessage" data-react-
  props="{&quot;name&quot;:&quot;John&quot;}"></div>
```

En la carga de la página, el controlador `react_ujs` escaneará la página y montará los componentes usando `data-react-class` y `data-react-props`.

Si Turbolinks está presente, los componentes se montan en la página: evento de cambio y se

desmontan en la página: antes de la descarga. Se recomienda Turbolinks > = 2.4.0 porque expone mejores eventos.

En el caso de llamadas Ajax, el montaje UJS puede activarse manualmente llamando desde javascript:

ReactRailsUJS.mountComponents () La firma del ayudante de vista es:

```
react_component(component_class_name, props={}, html_options={})
```

`component_class_name` es una cadena que nombra una clase de componente accesible globalmente. Puede tener puntos (por ejemplo, "MyApp.Header.Menuitem").

```
`props` is either an object that responds to `#to_json` or an already-stringified JSON object (eg, made with Jbuilder, see note below).
```

`html_options` puede incluir: `tag`: para usar un elemento distinto de un div para incrustar `data-reaccion-class` y `data-reac-props`. `prerender: true` para representar el componente en el servidor. **\*\*other** Cualquier otro argumento (por ejemplo, `class :`, `id :`) se pasa a `content_tag`.

Lea [Reacciona con los rieles usando la gema reaccion-rails](https://riptutorial.com/es/ruby-on-rails/topic/7032/reacciona-con-los-rieles-usando-la-gema-reaccion-rails). en línea:

[https://riptutorial.com/es/ruby-on-rails/topic/7032/reacciona-con-los-rieles-usando-la-gema-reaccion-rails-](https://riptutorial.com/es/ruby-on-rails/topic/7032/reacciona-con-los-rieles-usando-la-gema-reaccion-rails)

# Capítulo 64: Registrador de carriles

## Examples

### Rails.logger

Siempre use `Rails.logger.{debug|info|warn|error|fatal}` lugar de `puts`. Esto permite que sus registros se ajusten al formato de registro estándar, tengan una marca de tiempo y un nivel para que elija si son lo suficientemente importantes como para mostrarlos en un entorno específico. Puede ver los archivos de registro separados para su aplicación en `log/` directorio `log/` con el nombre del entorno de su aplicación rails. como: `development.log` `production.log` `staging.log`

Puede rotar fácilmente los registros de producción de rieles con LogRotate. Solo tiene que hacer una pequeña configuración como se muestra a continuación

Abra `/etc/logrotate.conf` con su editor de linux favorito `vim` o `nano` y agregue el siguiente código en este archivo en la parte inferior.

```
/YOUR/RAILSAPP/PATH/log/*.log {
  daily
  missingok
  rotate 7
  compress
  delaycompress
  notifempty
  copytruncate
}
```

Entonces, **¿cómo funciona?** Esto es increíblemente fácil. Cada bit de la configuración hace lo siguiente:

- **diario** : rote los archivos de registro cada día. También puedes usar semanalmente o mensualmente aquí.
- **missingok** - Si el archivo de registro no existe, ignóralo
- **rotar 7** - Mantener solo 7 días de troncos alrededor
- **comprimir** - GZip el archivo de registro en rotación
- **delaycompress** : rote el archivo un día, luego **comprímalo** al día siguiente para que podamos estar seguros de que no interferirá con el servidor Rails
- **notifempty** - No rote el archivo si los registros están vacíos
- **copytruncate** - Copia el archivo de registro y luego lo vacía. Esto asegura que el archivo de registro en el que Rails está escribiendo siempre existe, por lo que no tendrá problemas porque el archivo no cambia realmente. Si no usa esto, deberá reiniciar su aplicación Rails cada vez.

**Ejecutando Logrotate** Ya que acabamos de escribir esta configuración, desea probarla.

Para ejecutar logrotate manualmente, simplemente haga lo siguiente: `sudo /usr/sbin/logrotate -f /etc/logrotate.conf`

Eso es.

Lea Registrador de carriles en línea: <https://riptutorial.com/es/ruby-on-rails/topic/3904/registrador-de-carriles>

# Capítulo 65: Rieles - motores

## Introducción

Los motores pueden considerarse aplicaciones en miniatura que proporcionan funcionalidad a sus aplicaciones host. Una aplicación de Rails es en realidad solo un motor "supercargado", con la clase `Rails :: Application` heredando gran parte de su comportamiento de `Rails :: Engine`.

Los motores son las aplicaciones / complementos de rieles reutilizables. Funciona como una gema. Los motores famosos son las gemas `Device`, `Spree` que se pueden integrar fácilmente con aplicaciones de rieles.

## Sintaxis

- `rails plugin new [engine name] --mountable`

## Parámetros

Parámetros	Propósito
<code>- mountable</code>	La opción le dice al generador que desea crear un motor "montable" y aislado en el espacio de nombres
<code>--completo</code>	La opción le dice al generador que desea crear un motor, incluida una estructura de esqueleto

## Observaciones

Los motores son muy buenas opciones para crear un complemento reutilizable para la aplicación de rieles

## Examples

### Ejemplos famosos son

Generando un sencillo motor de blog.

```
rails plugin new [engine name] --mountable
```

Ejemplos de motores famosos son

[Dispositivo](#) (gema de autenticación para rieles)

[Spree](#) (comercio electrónico)

Lea Rieles - motores en línea: <https://riptutorial.com/es/ruby-on-rails/topic/10881/rieles---motores>

---

# Capítulo 66: Rieles en docker

## Introducción

Este tutorial comenzará con Docker instalado y con una aplicación Rails

## Examples

### Docker y docker-componer

En primer lugar, necesitaremos crear nuestro `Dockerfile`. Un buen ejemplo se puede encontrar en este [blog](#) por Nick Janetakis.

Este código contiene la secuencia de comandos que se ejecutará en nuestra máquina docker en el momento del inicio. Por este motivo, estamos instalando todas las bibliotecas necesarias y finaliza con el inicio de Puma (servidor de desarrollo RoR)

```
# Use the barebones version of Ruby 2.3.
FROM ruby:2.3.0-slim

# Optionally set a maintainer name to let people know who made this image.
MAINTAINER Nick Janetakis <nick.janetakis@gmail.com>

# Install dependencies:
# - build-essential: To ensure certain gems can be compiled
# - nodejs: Compile assets
# - libpq-dev: Communicate with postgres through the postgres gem
RUN apt-get update && apt-get install -qq -y --no-install-recommends \
    build-essential nodejs libpq-dev git

# Set an environment variable to store where the app is installed to inside
# of the Docker image. The name matches the project name out of convention only.
ENV INSTALL_PATH /mh-backend
RUN mkdir -p $INSTALL_PATH

# This sets the context of where commands will be running in and is documented
# on Docker's website extensively.
WORKDIR $INSTALL_PATH

# We want binstubs to be available so we can directly call sidekiq and
# potentially other binaries as command overrides without depending on
# bundle exec.
COPY Gemfile* $INSTALL_PATH/

ENV BUNDLE_GEMFILE $INSTALL_PATH/Gemfile
ENV BUNDLE_JOBS 2
ENV BUNDLE_PATH /gembox

RUN bundle install

# Copy in the application code from your work station at the current directory
# over to the working directory.
```

```
COPY . .

# Ensure the static assets are exposed to a volume so that nginx can read
# in these values later.
VOLUME ["$INSTALL_PATH/public"]

ENV RAILS_LOG_TO_STDOUT true

# The default command that gets run will be to start the Puma server.
CMD bundle exec puma -C config/puma.rb
```

Además, usaremos docker-compose, para eso, crearemos `docker-compose.yml` . La explicación de este archivo será más un tutorial de composición acoplable que una integración con Rails y no lo cubriremos aquí.

```
version: '2'

services:
  backend:
    links:
      - #whatever you need to link like db
    build: .
    command: ./scripts/start.sh
    ports:
      - '3000:3000'
    volumes:
      - ./backend
    volumes_from:
      - gembox
    env_file:
      - .dev-docker.env
    stdin_open: true
    tty: true
```

Solo con estos dos archivos tendrás suficiente para ejecutar la `docker-compose up` y activar tu ventana acoplable.

Lea Rieles en docker en línea: <https://riptutorial.com/es/ruby-on-rails/topic/10933/rieles-en-docker>

---

# Capítulo 67: RSpec y Ruby on Rails

## Observaciones

RSpec es un marco de prueba para Ruby o, como se define en la documentación oficial, *RSpec es una herramienta de desarrollo dirigida por el comportamiento para los programadores de Ruby*.

Este tema cubre los aspectos básicos del uso de [RSpec](#) con Ruby on Rails. Para obtener información específica sobre RSpec, visite el [tema RSpec](#).

## Examples

### Instalando RSpec

Si desea utilizar RSpec para un proyecto de Rails, debe usar la gema `rspec-rails`, que puede generar ayudantes y archivos de especificaciones para usted automáticamente (por ejemplo, cuando crea modelos, recursos o andamios utilizando la `rails generate`).

Agregue `rspec-rails` a los grupos de `:development` y `:test` en el `Gemfile`:

```
group :development, :test do
  gem 'rspec-rails', '~> 3.5'
end
```

Ejecutar `bundle` para instalar las dependencias.

Inicialízalo con:

```
rails generate rspec:install
```

Esto creará una `spec/` carpeta para sus pruebas, junto con los siguientes archivos de configuración:

- `.rspec` contiene opciones predeterminadas para la herramienta `rspec` línea de comandos
- `spec/spec_helper.rb` incluye opciones de configuración básicas de RSpec
- `spec/rails_helper.rb` agrega más opciones de configuración que son más específicas para usar RSpec y Rails juntos.

Todos estos archivos se escriben con valores predeterminados razonables para comenzar, pero puede agregar funciones y cambiar configuraciones para satisfacer sus necesidades a medida que su conjunto de pruebas crezca.

Lea [RSpec y Ruby on Rails en línea](https://riptutorial.com/es/ruby-on-rails/topic/5335/rspec-y-ruby-on-rails): <https://riptutorial.com/es/ruby-on-rails/topic/5335/rspec-y-ruby-on-rails>

---

# Capítulo 68: Serializadores de modelos activos

## Introducción

ActiveModelSerializers, o AMS para abreviar, trae 'convención sobre la configuración' a su generación JSON. ActiveModelSerializers funciona a través de dos componentes: serializadores y adaptadores. Los serializadores describen qué atributos y relaciones deben ser serializados. Los adaptadores describen cómo se deben serializar los atributos y las relaciones.

## Examples

### Usando un serializador

```
class SomeSerializer < ActiveModel::Serializer
  attribute :title, key: :name
  attributes :body
end
```

Lea Serializadores de modelos activos en línea: <https://riptutorial.com/es/ruby-on-rails/topic/9000/serializadores-de-modelos-activos>

---

# Capítulo 69: Trabajos activos

## Examples

### Introducción

Disponibles desde Rails 4.2, Active Job es un marco para declarar trabajos y hacer que se ejecuten en una variedad de backends de cola. Las tareas recurrentes o puntuales que no están bloqueando y que se pueden ejecutar en paralelo son buenos casos de uso para Trabajos Activos.

### Trabajo de muestra

```
class UserUnsubscribeJob < ApplicationJob
  queue_as :default

  def perform(user)
    # this will happen later
    user.unsubscribe
  end
end
```

### Creación de un trabajo activo a través del generador

```
$ rails g job user_unsubscribe
```

Lea Trabajos activos en línea: <https://riptutorial.com/es/ruby-on-rails/topic/8033/trabajos-activos>

---

# Capítulo 70: Transacciones ActiveRecord

## Observaciones

Las transacciones son bloques protectores donde las sentencias de SQL solo son permanentes si todas pueden tener éxito como una sola acción atómica. El ejemplo clásico es una transferencia entre dos cuentas donde solo puede tener un depósito si el retiro fue exitoso y viceversa. Las transacciones imponen la integridad de la base de datos y protegen los datos contra los errores del programa o las fallas de la base de datos. Básicamente, debería usar bloques de transacciones cuando tenga una serie de sentencias que deben ejecutarse juntas o no deben ejecutarse.

## Examples

### Ejemplo basico

Por ejemplo:

```
ActiveRecord::Base.transaction do
  david.withdrawal(100)
  mary.deposit(100)
end
```

Este ejemplo solo tomará dinero de David y se lo dará a Mary si ni el retiro ni el depósito generan una excepción. Las excepciones forzarán un ROLLBACK que devuelve la base de datos al estado antes de que comience la transacción. Tenga en cuenta, sin embargo, que a los objetos no se les devolverán los datos de instancia a su estado previo a la transacción.

### Diferentes clases de ActiveRecord en una sola transacción

Aunque se llama al método de clase de transacción en alguna clase de ActiveRecord, no es necesario que todos los objetos dentro del bloque de transacción sean instancias de esa clase. Esto se debe a que las transacciones son por conexión de base de datos, no por modelo.

En este ejemplo, un registro de saldo se guarda de manera transaccional aunque la transacción se llame en la clase de Cuenta:

```
Account.transaction do
  balance.save!
  account.save!
end
```

El método de transacción también está disponible como un método de instancia de modelo. Por ejemplo, también puedes hacer esto:

```
balance.transaction do
```

```
balance.save!  
account.save!  
end
```

## Conexiones de base de datos múltiples

Una transacción actúa en una sola conexión de base de datos. Si tiene varias bases de datos específicas de clase, la transacción no protegerá la interacción entre ellas. Una solución es comenzar una transacción en cada clase cuyos modelos modifique:

```
Student.transaction do  
  Course.transaction do  
    course.enroll(student)  
    student.units += course.units  
  end  
end
```

Esta es una solución deficiente, pero las transacciones totalmente distribuidas están fuera del alcance de ActiveRecord.

## guardar y destruir se envuelven automáticamente en una transacción

Tanto `#save` como `#destroy` vienen envueltos en una transacción que garantiza que todo lo que haga en las validaciones o devoluciones de llamada sucederá bajo su cobertura protegida. Por lo tanto, puede usar las validaciones para verificar los valores de los que depende la transacción o puede generar excepciones en las devoluciones de llamada para revertir, incluidas las devoluciones de llamada `after_*`.

Como consecuencia, los cambios en la base de datos no se ven fuera de su conexión hasta que se completa la operación. Por ejemplo, si intenta actualizar el índice de un motor de búsqueda en `after_save` el indexador no verá el registro actualizado. La `after_commit` llamada `after_commit` es la única que se activa una vez que se confirma la actualización.

## Devoluciones de llamada

Hay dos tipos de devoluciones de llamada asociadas con las transacciones de `after_commit` y `after_rollback`: `after_commit` y `after_rollback`.

`after_commit` devoluciones de llamada `after_commit` se llaman en cada registro guardado o destruido dentro de una transacción inmediatamente después de que se confirma la transacción. `after_rollback` devoluciones de llamada `after_rollback` realizan en cada registro guardado o destruido dentro de una transacción inmediatamente después de que la transacción o el punto de salvaguarda se revierte.

Estas devoluciones de llamada son útiles para interactuar con otros sistemas, ya que se le garantizará que la devolución de llamada solo se ejecute cuando la base de datos se encuentre en un estado permanente. Por ejemplo, `after_commit` es un buen lugar para poner un gancho para borrar un caché, ya que borrarlo de una transacción podría hacer que el caché se regenere antes de actualizar la base de datos.

## Deshacer una transacción

`ActiveRecord::Base.transaction` utiliza la excepción `ActiveRecord::Rollback` para distinguir una reversión deliberada de otras situaciones excepcionales. Normalmente, al generar una excepción, el método `.transaction` revertirá la transacción de la base de datos y pasará la excepción. Pero si `ActiveRecord::Rollback` una excepción `ActiveRecord::Rollback`, la transacción de la base de datos se retrotraerá, sin pasar la excepción.

Por ejemplo, podría hacer esto en su controlador para deshacer una transacción:

```
class BooksController < ActionController::Base
  def create
    Book.transaction do
      book = Book.new(params[:book])
      book.save!
      if today_is_friday?
        # The system must fail on Friday so that our support department
        # won't be out of job. We silently rollback this transaction
        # without telling the user.
        raise ActiveRecord::Rollback, "Call tech support!"
      end
    end
    # ActiveRecord::Rollback is the only exception that won't be passed on
    # by ActiveRecord::Base.transaction, so this line will still be reached
    # even on Friday.
    redirect_to root_url
  end
end
```

Lea Transacciones ActiveRecord en línea: <https://riptutorial.com/es/ruby-on-rails/topic/4688/transacciones-activerecord>

# Capítulo 71: Transacciones ActiveRecord

## Introducción

Las transacciones ActiveRecord son bloques protectores donde la secuencia de consultas de registros activos solo son permanentes si todas pueden tener éxito como una sola acción atómica.

## Examples

### Comenzando con transacciones de registro activo

Las transacciones de registro activo se pueden aplicar a las clases de modelo así como a las instancias de modelo, los objetos dentro del bloque de transacción no necesitan ser todos instancias de la misma clase. Esto se debe a que las transacciones son por conexión de base de datos, no por modelo. Por ejemplo:

```
User.transaction do
  account.save!
  profile.save!
  print "All saves success, returning 1"
  return 1
end
rescue_from ActiveRecord::RecordInvalid do |exception|
  print "Exception thrown, transaction rolledback"
  render_error "failure", exception.record.errors.full_messages.to_sentence
end
```

El uso de guardar con una explosión asegura que la transacción se retrotraerá automáticamente cuando se lance la excepción y, después de la reversión, el control se dirige al bloque de rescate para la excepción. **¡Asegúrate de rescatar las excepciones lanzadas desde el guardado! en Bloque de Transacciones.**

Si no desea utilizar guardar, puede elevar manualmente `raise ActiveRecord::Rollback` cuando falla la `raise ActiveRecord::Rollback` guardar. No necesitas manejar esta excepción. Luego, hará retroceder la transacción y tomará el control hasta la siguiente declaración después del bloqueo de la transacción.

```
User.transaction do
  if account.save && profile.save
    print "All saves success, returning 1"
    return 1
  else
    raise ActiveRecord::Rollback
  end
end
print "Transaction Rolled Back"
```

Lea Transacciones ActiveRecord en línea: <https://riptutorial.com/es/ruby-on->



---

# Capítulo 72: Turbolinks

## Introducción

Turbolinks es una biblioteca de JavaScript que hace que la navegación de su aplicación web sea más rápida. Cuando sigue un enlace, Turbolinks obtiene automáticamente la página, intercambia su < cuerpo > y fusiona su < encabezado >, todo ello sin incurrir en el costo de una carga de página completa.

## Observaciones

Como desarrollador de rieles, es probable que interactúes mínimamente con turbolinks durante tu desarrollo. Sin embargo, es una biblioteca importante con la que hay que estar familiarizado porque puede ser la causa de algunos errores difíciles de encontrar.

---

## Puntos clave:

- Enlazar a los `turbolinks:load` evento de `turbolinks:load` lugar del evento `document.ready`
- Utilice el atributo `data-turbolinks=false` para deshabilitar la funcionalidad de turbolink por enlace.
- Utilice el atributo `data-turbolinks-permanent` para persistir los elementos en todas las cargas de la página y evitar errores relacionados con la caché.

Para un tratamiento más profundo de los enlaces turbo, visite el [repositorio oficial de github](#) .

La mayor parte de esta documentación corresponde a la gente que redactó la documentación de turbolinks en el repositorio de github.

## Examples

### Enlace al concepto de turbolink de una carga de página.

Con turbolinks, el enfoque tradicional para usar:

```
$(document).ready(function() {  
  // awesome code  
});
```

no funcionará Durante el uso de turbolinks, el evento `$(document).ready()` solo se activará una vez: en la carga de la página inicial. A partir de ese momento, cada vez que un usuario haga clic en un enlace de su sitio web, turbolinks interceptará el evento de clic de enlace y realizará una solicitud ajax para reemplazar la etiqueta < body > y combinar las etiquetas < head >. Todo el proceso desencadena la noción de una "visita" en tierra turbolinks. Por lo tanto, en lugar de usar la sintaxis de `document.ready()` anterior, tendrá que vincularse al evento de visita de turbolink, de este modo:

```
// pure js
document.addEventListener("turbolinks:load", function() {
  // awesome code
});

// jQuery
$(document).on('turbolinks:load', function() {
  // your code
});
```

## Desactivar turbolinks en enlaces específicos

Es muy fácil desactivar turbolinks en enlaces específicos. Según [la documentación oficial de turbolinks](#) :

Los Turbolinks se pueden desactivar por enlace, anotando un enlace o cualquiera de sus antepasados con `data-turbolinks = "false"`.

## Ejemplos:

```
// disables turbolinks for this one link
<a href="/" data-turbolinks="false">Disabled</a>

// disables turbolinks for all links nested within the div tag
<div data-turbolinks="false">
  <a href="/">I'm disabled</a>
  <a href="/">I'm also disabled</a>
</div>

// re-enable specific link when ancestor has disabled turbolinks
<div data-turbolinks="false">
  <a href="/">I'm disabled</a>
  <a href="/" data-turbolinks="true">I'm re-enabled</a>
</div>
```

## Entender las visitas de aplicaciones

Las visitas de la aplicación se inician haciendo clic en un enlace habilitado para Turbolinks o programáticamente llamando

```
Turbolinks.visit(location)
```

De forma predeterminada, la función de visita utiliza la acción "avance". De manera más comprensible, el comportamiento predeterminado para la función de visita es avanzar a la página indicada por el parámetro "ubicación". Cada vez que se visita una página, turbolinks inserta una nueva entrada en el historial del navegador usando `history.pushState` . El historial es importante porque turbolinks intentará usar el historial para cargar páginas desde la memoria caché siempre que sea posible. Esto permite una representación de páginas extremadamente rápida para las páginas visitadas con frecuencia.

Sin embargo, si desea visitar una ubicación sin insertar ningún historial en la pila, puede usar la acción 'reemplazar' en la función de visita de la siguiente manera:

```
// using links
<a href="/edit" data-turbolinks-action="replace">Edit</a>

// programatically
Turbolinks.visit("/edit", { action: "replace" })
```

Esto reemplazará la parte superior de la pila de historial con la nueva página para que el número total de elementos en la pila permanezca sin cambios.

También hay una acción de "restauración" que ayuda en las visitas de [restauración](#), las visitas que se producen como resultado de que el usuario haga clic en el botón de avance o atrás en su navegador. Turbolinks maneja estos tipos de eventos internamente y recomienda que los usuarios no manipulen manualmente el comportamiento predeterminado.

## Cancelando visitas antes de que comiencen

Turbolinks proporciona un detector de eventos que se puede usar para evitar que ocurran visitas. Escuche los `turbolinks:before-visit` evento `turbolinks:before-visit` para que se le notifique cuando una visita está por comenzar.

En el controlador de eventos, puede utilizar:

```
// pure javascript
event.data.url
```

o

```
// jQuery
$(event.originalEvent.data.url
```

Para recuperar la ubicación de la visita. La visita puede ser cancelada llamando al

```
event.preventDefault()
```

## NOTA:

Según los [documentos oficiales de turbolinks](#) :

Las visitas de restauración no se pueden cancelar y no se activan los turbolinks: antes de la visita.

## Elementos persistentes a través de cargas de página

Considera la siguiente situación: imagina que eres el desarrollador de un sitio web de redes

sociales que permite a los usuarios ser amigos de otros usuarios y que utiliza enlaces de enlace para hacer que la página se cargue más rápido. En la parte superior derecha de cada página del sitio, hay un número que indica el número total de amigos que un usuario tiene actualmente. Imagina que estás usando tu sitio y que tienes 3 amigos. Cada vez que se agrega un nuevo amigo, tiene un javascript que se ejecuta y actualiza el contador de amigos. Imagine que acaba de agregar un nuevo amigo y que su javascript se ejecutó correctamente y actualizó el recuento de amigos en la parte superior derecha de la página para procesar ahora 4. Ahora, imagine que hace clic en el botón Atrás del navegador. Cuando se carga la página, observa que el contador de amigos dice 3 aunque tenga cuatro amigos.

Este es un problema relativamente común y para el que Turbolinks ha proporcionado una solución. La razón por la que se produce el problema es porque turbolinks carga automáticamente las páginas de la caché cuando un usuario hace clic en el botón Atrás. La página en caché no siempre se actualizará con la base de datos.

Para resolver este problema, imagine que representa el recuento de amigos dentro de una etiqueta `<div>` con un ID de "recuento de amigos":

```
<div id="friend-count" data-turbolinks-permanent>3 friends</div>
```

Al agregar el atributo `data-turbolinks-permanent`, le está diciendo a turbolinks que persista ciertos elementos en las cargas de la página. Los [documentos oficiales dicen](#) :

Designe elementos permanentes asignándoles un ID de HTML y anotándolos con `data-turbolinks-permanent`. Antes de cada procesamiento, Turbolinks hace coincidir todos los elementos permanentes por ID y los transfiere de la página original a la nueva página, conservando sus datos y escuchas de eventos.

Lea Turbolinks en línea: <https://riptutorial.com/es/ruby-on-rails/topic/9331/turbolinks>

# Capítulo 73: Usando GoogleMaps con Rails

## Examples

### Agregue la etiqueta javascript de google maps al encabezado de diseño

Para que Google Maps funcione correctamente con [turbolinks](#) , agregue la etiqueta javascript directamente al encabezado del diseño en lugar de incluirla en una vista.

```
# app/views/layouts/my_layout.html.haml
!!!
%html{:lang => 'en'}
  %head
    - # ...
    = google_maps_api_script_tag
```

El `google_maps_api_script_tag` se define mejor en un ayudante.

```
# app/helpers/google_maps_helper.rb
module GoogleMapsHelper
  def google_maps_api_script_tag
    javascript_include_tag google_maps_api_source
  end

  def google_maps_api_source
    "https://maps.googleapis.com/maps/api/js?key=#{google_maps_api_key}"
  end

  def google_maps_api_key
    Rails.application.secrets.google_maps_api_key
  end
end
```

Puede registrar su aplicación en google y obtener su clave de api en la [consola de google api](#) . Google tiene una breve [guía sobre cómo solicitar una clave de API para la API de Google Maps javascript](#) .

La clave api se almacena en el archivo `secrets.yml` :

```
# config/secrets.yml
development:
  google_maps_api_key: '...'
  # ...
production:
  google_maps_api_key: '...'
  # ...
```

No se olvide de añadir `config/secrets.yml` a su `.gitignore` archivo y makre asegurarse de que no se compromete la clave de API en el repositorio.

## Geocodificar el modelo

Supongamos que sus usuarios y / o grupos tienen perfiles y desea mostrar los campos de perfil de dirección en un mapa de Google.

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # Attributes:
  # label, e.g. "Work address"
  # value, e.g. "Willy-Brandt-Straße 1\n10557 Berlin"
end
```

Una gran manera de geocodificar las direcciones, es decir, proporcionar `longitude` y `latitude` es la [gema del geocodificador](#) .

Agregue geocodificador a su `Gemfile` y ejecute `bundle` para instalarlo.

```
# Gemfile
gem 'geocoder', '~> 1.3'
```

Agregue columnas de la base de datos para `latitude` y `longitude` para guardar la ubicación en la base de datos. Esto es más eficiente que consultar el servicio de geocodificación cada vez que necesita la ubicación. Es más rápido y no alcanzas el límite de consulta tan rápido.

```
→ bin/rails generate migration add_latitude_and_longitude_to_profile_fields \
  latitude:float longitude:float
→ bin/rails db:migrate # Rails 5, or:
→ rake db:migrate # Rails 3, 4
```

Agregue el mecanismo de geocodificación a su modelo. En este ejemplo, la cadena de dirección se almacena en el atributo de `value` . Configure la geocodificación para que se realice cuando el registro haya cambiado, y solo cuando esté presente un valor:

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  geocoded_by :value
  after_validation :geocode, if: ->(address_field){
    address_field.value.present? and address_field.value_changed?
  }
end
```

Por defecto, el geocodificador usa google como servicio de búsqueda. Tiene muchas características interesantes como cálculos de distancia o búsqueda de proximidad. Para más información, eche un vistazo al [README del geocodificador](#) .

## Mostrar direcciones en un mapa de Google en la vista de perfil

En la vista de perfil, muestre los campos de perfil de un usuario o grupo en una lista, así como los campos de dirección en un mapa de Google.

```
- # app/views/profiles/show.html.haml
%h1 Contact Information
.profile_fields
  = render @profile_fields
.google_map{data: address_fields: @address_fields.to_json }
```

Los `@profile_fields` y `@address_fields` apropiados se configuran en el controlador:

```
# app/controllers/profiles_controller.rb
class ProfilesController < ApplicationController
  def show
    # ...
    @profile_fields = @user_or_group.profile_fields
    @address_fields = @profile_fields.where(type: 'ProfileFields::Address')
  end
end
```

Inicialice el mapa, coloque los marcadores, configure el zoom y otras configuraciones del mapa con javascript.

The screenshot shows a web interface with a navigation bar at the top containing links for 'Info', 'Contact', 'Corporate Vita', 'Work & Study', and 'More'. The 'Contact' tab is active. Below the navigation bar is a section titled 'Contact Information'. This section contains two entries for 'Work address' and one for 'Email'. The first 'Work address' entry lists 'John Doe' and 'Willy-Brandt-Straße 1, 10557 Berlin'. The second 'Work address' entry lists 'John Doe', '1-6 Chesham Pl', 'London', 'SW1X 8PZ', and 'United Kingdom'. The 'Email' entry lists 'doe@example.com'. To the right of the contact information is a map showing a portion of Europe with a red location pin over London. The map includes labels for 'London', 'Paris', and 'Frankreich' (Germany). The map interface also shows 'Karte' and 'Satellit' tabs at the top, and a copyright notice '© 2016 Google, ORIO' at the bottom.

## Establecer los marcadores en el mapa con javascript

Supongamos que hay un div `.google_map`, que se convertirá en el mapa, y que tiene los campos de dirección para mostrar como marcadores como atributo de `data`.

Por ejemplo:

```
<!-- http://localhost:3000/profiles/123 -->
<div class="google_map" data-address-fields="[
  {label: 'Work address', value: 'Willy-Brandt-Straße 1\n10557 Berlin',
  position: {lng: ..., lat: ...}},
  ...
]"></div>
```

Para utilizar el evento `$(document).ready` con [turbolinks](#) sin gestionar los eventos `turbolinks` a mano, use la [gema jquery.turbolinks](#).

Si desea realizar otras operaciones con el mapa, más adelante, por ejemplo, mediante el filtrado o las ventanas de información, es conveniente que el mapa sea administrado por una [clase de script de café](#).

```
# app/assets/javascripts/google_maps.js.coffee
window.App = {} unless App?
class App.GoogleMap
  constructor: (map_div)->
    # TODO: initialize the map
    # TODO: set the markers
```

Cuando se usan varios archivos de `script de café`, que tienen espacios de nombre por defecto, es conveniente definir un espacio de nombres de `App` global, que sea compartido por todos los archivos de `script de café`.

Luego, `.google_map` (posiblemente varios) `divs .google_map` y cree una instancia de la clase `App.GoogleMap` para cada uno de ellos.

```
# app/assets/javascripts/google_maps.js.coffee
# ...
$(document).ready ->
  App.google_maps = []
  $('.google_map').each ->
    map_div = $(this)
    map = new App.GoogleMap map_div
    App.google_maps.push map
```

## Inicialice el mapa usando una clase de `script de café`.

Siempre que haya una [clase de script de café](#) `App.GoogleMap`, el mapa de google se puede inicializar así:

```
# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  map_div: {}
  map: {}
```

```

constructor: (map_div)->
  @map_div = map_div
  @init_map()
  @reference_the_map_as_data_attribute

# To access the GoogleMap object or the map object itself
# later via the DOM, for example
#
#   $('google_map').data('GoogleMap')
#
# store references as data attribute of the map_div.
#
reference_the_map_as_data_attribute: ->
  @map_div.data 'GoogleMap', this
  @map_div.data 'map', @map

init_map: ->
  @map = new google.maps.Map(@dom_element, @map_configuration) if google?

# `@map_div` is the jquery object. But google maps needs
# the real DOM element.
#
dom_element: ->
  @map_div.get(0)

map_configuration: -> {
  scrollWheel: true
}

```

Para aprender más acerca de los posibles `map_configuration` opciones, echar un vistazo a Google de [documentación MapOptions](#) y su [guía a la adición de elementos de control](#) .

Para referencia, la clase `google.maps.Map` está [ampliamente documentada aquí](#) .

## Inicialice los marcadores de mapa usando una clase de script de café

Si se proporciona una [clase de script de café](#) `App.GoogleMap` y la información del marcador se almacena en el atributo `data-address-fields` de la div `.google_map` , los marcadores del mapa se pueden inicializar en el mapa de la siguiente manera:

```

# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  # ...
  markers: []

  constructor: (map_div)->
    # ...
    @init_markers()

  address_fields: ->
    @map_div.data('address-fields')

  init_markers: ->
    self = this # to reference the instance as `self` when `this` is redefined.
    self.markers = []

```

```

for address_field in self.address_fields()
  marker = new google.maps.Marker {
    map: self.map,
    position: {
      lng: address_field.longitude,
      lat: address_field.latitude
    },
    # # or, if `position` is defined in `ProfileFields::Address#as_json`:
    # position: address_field.position,
    title: address_field.value
  }
self.markers.push marker

```

Para obtener más información sobre las opciones de marcadores, consulte la [documentación de MarkerOptions](#) y su [guía de marcadores](#) .

## Zoom automático de un mapa usando una clase de script de café

Se proporcionó una [clase de script de café](#) `App.GoogleMap` con `google.maps.Map` almacenado como `@map` y `google.maps.Marker` s almacenado como `@markers` , el mapa puede `@markers` automáticamente, es decir, ajustado para que todos los marcadores sean visibles, de esta manera : en el mapa como este:

```

# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  # ...
  bounds: {}

  constructor: (map_div)->
    # ...
    @auto_zoom()

  auto_zoom: ->
    @init_bounds()
    # TODO: Maybe, adjust the zoom to have a maximum or
    # minimum zoom level, here.

  init_bounds: ->
    @bounds = new google.maps.LatLngBounds()
    for marker in @markers
      @bounds.extend marker.position
    @map.fitBounds @bounds

```

Para obtener más información sobre los límites, eche un vistazo a la [documentación de LatLngBounds](#) de [Google](#) .

## Exponiendo las propiedades del modelo como json.

Para mostrar los campos de perfil de dirección como marcadores en un mapa de Google, los objetos del campo de dirección deben pasarse como objetos json a javascript.

# Atributos regulares de la base de datos

Al llamar a `to_json` en un objeto `ApplicationRecord`, los atributos de la base de datos se exponen automáticamente.

Dado un `ProfileFields::Address` Modelo de `ProfileFields::Address` con atributos de `label`, `value`, `longitudo` y `latitude`, `address_field.as_json` da `address_field.as_json` resultado un Hash, por ejemplo, representación,

```
address_field.as_json # =>
  {label: "Work address", value: "Willy-Brandt-Straße 1\n10557 Berlin",
   longitude: ..., latitude: ...}
```

que se convierte en una cadena json por `to_json`:

```
address_field.to_json # =>
  "{\"label\":\"Work address\",\"value\":\"Willy-Brandt-Straße 1\\n10557 Berlin\",\"longitude\":...,\"latitude\":...}"
```

Esto es útil porque permite usar `label` y `value` más adelante en javascript, por ejemplo, para mostrar sugerencias de herramientas para los marcadores de mapa.

---

## Otros atributos

Otros atributos virtuales se pueden exponer anulando el método `as_json`.

Por ejemplo, para exponer un atributo de `title`, `as_json` hash `as_json` combinado:

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # ...

  # For example: "John Doe, Work address"
  def title
    "#{self.parent.name}, #{self.label}"
  end

  def as_json
    super.merge {
      title: self.title
    }
  end
end
```

El ejemplo anterior usa `super` para llamar al método `as_json` original, que devuelve el hash del atributo original del objeto, y lo combina con el hash de posición requerido.

Para comprender la diferencia entre `as_json` y `to_json`, `to_json` un vistazo a [esta publicación del blog de julian](#).

---

## Posición

Para representar marcadores, la API de google maps, de forma predeterminada, requiere un hash de `position` que tiene la longitud y la latitud almacenadas como `lng` y `lat` respectivamente.

Este hash de posición se puede crear en javascript, más tarde, o aquí cuando se define la representación json del campo de dirección:

Para proporcionar esta `position` como atributo json del campo de dirección, simplemente anule el método `as_json` en el modelo.

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # ...

  def as_json
    super.merge {
      # ...
      position: {
        lng: self.longitude,
        lat: self.latitude
      }
    }
  end
end
```

Lea Usando GoogleMaps con Rails en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2828/usando-googlemaps-con-rails>

# Capítulo 74: Validación de ActiveRecord

## Examples

### Validando la numericalidad de un atributo.

Esta validación restringe la inserción de solo valores numéricos.

```
class Player < ApplicationRecord
  validates :points, numericality: true
  validates :games_played, numericality: { only_integer: true }
end
```

Además `:only_integer`, este asistente también acepta las siguientes opciones para agregar restricciones a los valores aceptables:

- `:greater_than`: especifica que el valor debe ser mayor que el valor proporcionado. El mensaje de error predeterminado para esta opción es "debe ser mayor que% {count}".
- `:greater_than_or_equal_to`: especifica que el valor debe ser mayor o igual al valor suministrado. El mensaje de error predeterminado para esta opción es "debe ser mayor o igual a% {count}".
- `:equal_to`: especifica que el valor debe ser igual al valor suministrado. El mensaje de error predeterminado para esta opción es "debe ser igual a% {count}".
- `:less_than`: especifica que el valor debe ser menor que el valor proporcionado. El mensaje de error predeterminado para esta opción es "debe ser menor que% {count}".
- `:less_than_or_equal_to`: especifica que el valor debe ser menor o igual al valor proporcionado. El mensaje de error predeterminado para esta opción es "debe ser menor o igual a% {count}".
- `:other_than`: especifica que el valor debe ser distinto del valor suministrado. El mensaje de error predeterminado para esta opción es "debe ser distinto de% {count}".
- `:odd`: especifica que el valor debe ser un número impar si se establece en true. El mensaje de error predeterminado para esta opción es "debe ser impar".
- `:even`: especifica que el valor debe ser un número par si se establece en true. El mensaje de error predeterminado para esta opción es "debe ser par".

Por defecto, la numericalidad no permite valores nulos. Puede usar la opción `allow_nil: true` para permitirlo.

### Validar la singularidad de un atributo.

Este ayudante valida que el valor del atributo sea único justo antes de que se guarde el objeto.

```
class Account < ApplicationRecord
  validates :email, uniqueness: true
end
```

Hay una opción de `:scope` que puede usar para especificar uno o más atributos que se usan para limitar la verificación de unicidad:

```
class Holiday < ApplicationRecord
  validates :name, uniqueness: { scope: :year,
    message: "should happen once per year" }
end
```

También hay una opción `:case_sensitive` que puede usar para definir si la restricción de unicidad será sensible a mayúsculas o no. Esta opción por defecto es `true`.

```
class Person < ApplicationRecord
  validates :name, uniqueness: { case_sensitive: false }
end
```

## Validar la presencia de un atributo.

Este ayudante valida que los atributos especificados no estén vacíos.

```
class Person < ApplicationRecord
  validates :name, presence: true
end

Person.create(name: "John").valid? # => true
Person.create(name: nil).valid? # => false
```

Puede usar el asistente de `absence` para validar que los atributos especificados están ausentes. ¿Utiliza el `present?` Método para verificar valores nulos o vacíos.

```
class Person < ApplicationRecord
  validates :name, :login, :email, absence: true
end
```

**Nota:** En caso de que el atributo sea `boolean`, no puede hacer uso de la validación de presencia habitual (el atributo no sería válido para un valor `false`). Puede hacer esto utilizando una validación de inclusión:

```
validates :attribute, inclusion: [true, false]
```

## Saltando validaciones

Utilice los siguientes métodos si desea omitir las validaciones. Estos métodos guardarán el objeto en la base de datos incluso si no es válido.

- `¡decremento!`
- `decrement_counter`
- `¡incremento!`
- `increment_counter`
- `¡palanca!`

- toque
- actualizar todo
- atributo\_actualizacion
- update\_column
- update\_columns
- actualizar\_contadores

También puede omitir la validación mientras se guarda al pasar `validate` como un argumento para `save`

```
User.save(validate: false)
```

## Validación de la longitud de un atributo.

```
class Person < ApplicationRecord
  validates :name, length: { minimum: 2 }
  validates :bio, length: { maximum: 500 }
  validates :password, length: { in: 6..20 }
  validates :registration_number, length: { is: 6 }
end
```

Las posibles opciones de restricción de longitud son:

- `:minimum` : el atributo no puede tener menos de la longitud especificada.
- `:maximum` : el atributo no puede tener más de la longitud especificada.
- `:in` (o `:within`): la longitud del atributo debe incluirse en un intervalo determinado. El valor para esta opción debe ser un rango.
- `:is` - La longitud del atributo debe ser igual al valor dado.

## Validación de agrupación

A veces es útil tener varias validaciones para usar una condición. Se puede lograr fácilmente usando `with_options`.

```
class User < ApplicationRecord
  with_options if: :is_admin? do |admin|
    admin.validates :password, length: { minimum: 10 }
    admin.validates :email, presence: true
  end
end
```

Todas las validaciones dentro del bloque `with_options` pasarán automáticamente la condición si: `is_admin?`

## Validaciones personalizadas

Puede agregar sus propias validaciones agregando nuevas clases heredadas de `ActiveModel::Validator` o de `ActiveModel::EachValidator`. Ambos métodos son similares pero funcionan de formas ligeramente diferentes:

## ActiveModel::Validator **and** validates\_with

Implementar el método de `validate` que toma un registro como argumento y realiza la validación en él. Luego use `validates_with` con la clase en el modelo.

```
# app/validators/starts_with_a_validator.rb
class StartsWithAValidator < ActiveModel::Validator
  def validate(record)
    unless record.name.starts_with? 'A'
      record.errors[:name] << 'Need a name starting with A please!'
    end
  end
end

class Person < ApplicationRecord
  validates_with StartsWithAValidator
end
```

## ActiveModel::EachValidator **y** validate

Si prefiere usar su nuevo validador usando el método de `validate` común en un solo parámetro, cree una clase `ActiveModel::EachValidator` de `ActiveModel::EachValidator` e implemente el método `validate_each` que toma tres argumentos: `record`, `attribute` y `value`:

```
class EmailValidator < ActiveModel::EachValidator
  def validate_each(record, attribute, value)
    unless value =~ /\A([\^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\z/i
      record.errors[attribute] << (options[:message] || 'is not an email')
    end
  end
end

class Person < ApplicationRecord
  validates :email, presence: true, email: true
end
```

Más información sobre las [guías de rieles](#) .

## Valida el formato de un atributo.

Valide que el valor de un atributo coincida con una expresión regular usando el `format` y la opción `with` .

```
class User < ApplicationRecord
  validates :name, format: { with: /\A\w{6,10}\z/ }
end
```

También puede definir una constante y establecer su valor en una expresión regular y pasarla a la opción `with`: Esto podría ser más conveniente para expresiones regulares realmente complejas

```
PHONE_REGEX = /\A(\d{3})\d{3}-\d{4}\z/
```

```
validates :phone, format: { with: PHONE_REGEX }
```

El mensaje de error predeterminado es `is invalid`. Esto se puede cambiar con la opción `:message`.

```
validates :bio, format: { with: /\A\D+\z/, message: "Numbers are not allowed" }
```

El reverso también responde, y puede especificar que un valor *no* debe coincidir con una expresión regular con la opción `without`:

## Valida la inclusión de un atributo.

Puede verificar si un valor está incluido en una matriz usando la `inclusion: helper`. La opción `:in` y su alias, `:within` muestran el conjunto de valores aceptables.

```
class Country < ApplicationRecord
  validates :continent, inclusion: { in: %w(Africa Antartica Asia Australia
                                           Europe North America South America) }
end
```

Para verificar si un valor no está incluido en una matriz, use la `exclusion: helper`

```
class User < ApplicationRecord
  validates :name, exclusion: { in: %w(admin administrator owner) }
end
```

## Validación condicional

A veces es posible que necesite validar el registro solo bajo ciertas condiciones.

```
class User < ApplicationRecord
  validates :name, presence: true, if: :admin?

  def admin?
    conditional here that returns boolean value
  end
end
```

Si tu condicional es realmente pequeño, puedes usar un Proc:

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: Proc.new { |user| user.last_name.blank? }
end
```

Para condicional negativo puede usar a `unless`:

```
class User < ApplicationRecord
  validates :first_name, presence: true, unless: Proc.new { |user| user.last_name.present? }
end
```

También puede pasar una cadena, que se ejecutará a través de `instance_eval`:

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: 'last_name.blank?'
end
```

## Confirmación de atributo

Debe usar esto cuando tenga dos campos de texto que deberían recibir exactamente el mismo contenido. Por ejemplo, es posible que desee confirmar una dirección de correo electrónico o una contraseña. Esta validación crea un atributo **virtual** cuyo nombre es el nombre del campo que debe confirmarse con `_confirmation` anexada.

```
class Person < ApplicationRecord
  validates :email, confirmation: true
end
```

**Nota** Esta comprobación se realiza solo si `email_confirmation` no es nulo.

Para requerir confirmación, asegúrese de agregar una verificación de presencia para el atributo de confirmación.

```
class Person < ApplicationRecord
  validates :email,          confirmation: true
  validates :email_confirmation, presence: true
end
```

## Fuente

### Usando: en la opción

La opción `:on` permite especificar cuándo debe ocurrir la validación. El comportamiento predeterminado para todos los ayudantes de validación incorporados se debe ejecutar en guardar (tanto cuando está creando un nuevo registro como cuando lo está actualizando).

```
class Person < ApplicationRecord
  # it will be possible to update email with a duplicated value
  validates :email, uniqueness: true, on: :create

  # it will be possible to create the record with a non-numerical age
  validates :age, numericality: true, on: :update

  # the default (validates on both create and update)
  validates :name, presence: true
end
```

Lea Validación de ActiveRecord en línea: <https://riptutorial.com/es/ruby-on-rails/topic/2105/validacion-de-activerecord>

# Creditos

S. No	Capítulos	Contributors
1	Empezando con Ruby on Rails	<a href="#">Abhishek Jain</a> , <a href="#">Adam Lassek</a> , <a href="#">Ajay Barot</a> , <a href="#">animuson</a> , <a href="#">ArtOfCode</a> , <a href="#">Aswathy</a> , <a href="#">Community</a> , <a href="#">Darpan Chhatravala</a> , <a href="#">Darshan Patel</a> , <a href="#">Deepak Mahakale</a> , <a href="#">fybw id</a> , <a href="#">Geoffroy</a> , <a href="#">hschin</a> , <a href="#">hvenables</a> , <a href="#">Jon Wood</a> , <a href="#">kfrz</a> , <a href="#">Kirti Thorat</a> , <a href="#">Lorenzo Baracchi</a> , <a href="#">Luka Kerr</a> , <a href="#">MauroPorrasP</a> , <a href="#">michaelpri</a> , <a href="#">nifCody</a> , <a href="#">Niyanta</a> , <a href="#">olive_tree</a> , <a href="#">RADan</a> , <a href="#">RareFever</a> , <a href="#">Richard Hamilton</a> , <a href="#">sa77</a> , <a href="#">saadlulu</a> , <a href="#">sahil</a> , <a href="#">Sathishkumar Jayaraj</a> , <a href="#">Simone Carletti</a> , <a href="#">Stanislav Valášek</a> , <a href="#">theoretisch</a> , <a href="#">tpei</a> , <a href="#">Undo</a> , <a href="#">uzaif</a> , <a href="#">Yana</a>
2	ActionCable	<a href="#">Ich</a> , <a href="#">Sladey</a> , <a href="#">Undo</a>
3	ActionMailer	<a href="#">Adam Lassek</a> , <a href="#">Atul Khanduri</a> , <a href="#">jackerman09</a> , <a href="#">owahab</a> , <a href="#">Phil Ross</a> , <a href="#">Richard Hamilton</a> , <a href="#">Rodrigo Argumedo</a> , <a href="#">William Romero</a>
4	ActiveJob	<a href="#">Brian</a> , <a href="#">owahab</a>
5	ActiveModel	<a href="#">Adam Lassek</a> , <a href="#">RamenChef</a>
6	ActiveRecord	<a href="#">Adam Lassek</a> , <a href="#">AnoE</a> , <a href="#">Bijal Gajjar</a> , <a href="#">br3nt</a> , <a href="#">D-side</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">glapworth</a> , <a href="#">jeffdill2</a> , <a href="#">Joel Drapper</a> , <a href="#">Luka Kerr</a> , <a href="#">maartenvanvliet</a> , <a href="#">marcamillion</a> , <a href="#">Mario Uher</a> , <a href="#">powerup7</a> , <a href="#">Sebastialonso</a> , <a href="#">Simone Carletti</a> , <a href="#">Sven Reuter</a> , <a href="#">walid</a>
7	ActiveRecord Locking	<a href="#">Adam Lassek</a> , <a href="#">fatfrog</a> , <a href="#">Muaaz Rafi</a>
8	ActiveSupport	<a href="#">Adam Lassek</a>
9	Actualización de rieles	<a href="#">hschin</a> , <a href="#">michaelpri</a> , <a href="#">Rodrigo Argumedo</a>
10	Agregar panel de administración	<a href="#">Ahsan Mahmood</a> , <a href="#">MSathieu</a>
11	Agregar un Amazon RDS a su aplicación de rieles	<a href="#">Sathishkumar Jayaraj</a>
12	Almacenamiento en caché	<a href="#">ArtOfCode</a> , <a href="#">Cuisine Hacker</a> , <a href="#">Khanh Pham</a> , <a href="#">RamenChef</a> , <a href="#">tirdadc</a>
13	Almacenamiento	<a href="#">DawnPaladin</a>

	seguro de claves de autenticación	
14	API de Rails	<a href="#">Adam Lassek</a> , <a href="#">hschin</a>
15	Aplicaciones de carriles de prueba	<a href="#">HParker</a>
16	Asociaciones ActiveRecord	<a href="#">giniouxe</a> , <a href="#">Hardik Upadhyay</a> , <a href="#">Khanh Pham</a> , <a href="#">Luka Kerr</a> , <a href="#">Manish Agarwal</a> , <a href="#">Niyanta</a> , <a href="#">RareFever</a> , <a href="#">Raynor Kuang</a> , <a href="#">Sapna Jindal</a>
17	Autenticación API Rails 5	<a href="#">HParker</a>
18	Autenticar Api utilizando Devise	<a href="#">Vishal Taj PM</a>
19	Autenticación de usuario en rieles	<a href="#">Abhinay</a> , <a href="#">Ahsan Mahmood</a> , <a href="#">Antarr Byrd</a> , <a href="#">ArtOfCode</a> , <a href="#">dgilperez</a> , <a href="#">Kieran Andrews</a> , <a href="#">Luka Kerr</a> , <a href="#">Qchmq5</a> , <a href="#">uzaif</a> ,
20	Autorización con CanCan	<a href="#">4444</a> , <a href="#">Ahsan Mahmood</a> , <a href="#">dgilperez</a> , <a href="#">mlabarca</a> , <a href="#">toobulkeh</a>
21	Ayudantes de formulario	<a href="#">aisflat439</a> , <a href="#">owahab</a> , <a href="#">Richard Hamilton</a> , <a href="#">Simon Tsang</a> , <a href="#">Slava.K</a>
22	Cambiar un entorno de aplicación de Rails predeterminado	<a href="#">Whitecat</a>
23	Cambiar zona horaria predeterminada	<a href="#">Mihai-Andrei Dinculescu</a>
24	Característica de pago en rieles.	<a href="#">ppascualv</a> , <a href="#">Sathishkumar Jayaraj</a>
25	Cargas de archivos	<a href="#">Sergey Khmelevskoy</a>
26	Carriles 5	<a href="#">thiago araujo</a>
27	Chica de fábrica	<a href="#">Rafael Costa</a>
28	Columnas multiusos de ActiveRecord	<a href="#">Fabio Ros</a>
29	Configuración	<a href="#">Ali MasudianPour</a> , <a href="#">Undo</a>
30	Configurar Angular	<a href="#">B8vrede</a> , <a href="#">Rory O'Kane</a> , <a href="#">Umar Khan</a>

	con Rieles	
31	Constantize seguro	<a href="#">Eric Bouchut</a> , <a href="#">Ryan K</a>
32	Controlador de acción	<a href="#">Adam Lassek</a> , <a href="#">Atul Khanduri</a> , <a href="#">Deep</a> , <a href="#">Fire-Dragon-DoL</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">jackerman09</a> , <a href="#">RamenChef</a> , <a href="#">Sven Reuter</a>
33	Convenciones de nombres	<a href="#">Andrey Deineko</a> , <a href="#">Atul Khanduri</a> , <a href="#">br3nt</a> , <a href="#">Flambino</a> , <a href="#">giniouxe</a> , <a href="#">hgsongra</a> , <a href="#">Luka Kerr</a> , <a href="#">Marko Kacanski</a> , <a href="#">Muhammad Abdullah</a> , <a href="#">Sven Reuter</a> , <a href="#">Xinyang Li</a>
34	Depuración	<a href="#">Adam Lassek</a> , <a href="#">Dénes Papp</a> , <a href="#">Dharam</a> , <a href="#">Kelseydh</a> , <a href="#">sa77</a> , <a href="#">titan</a>
35	Despliegue de una aplicación Rails en Heroku	<a href="#">B Liu</a> , <a href="#">hschin</a>
36	Elasticsearch	<a href="#">Don Giovanni</a> , <a href="#">Luc Boissaye</a>
37	Enrutamiento	<a href="#">Adam Lassek</a> , <a href="#">advishnuprasad</a> , <a href="#">Ahsan Mahmood</a> , <a href="#">Alejandro Babio</a> , <a href="#">Andy Gauge</a> , <a href="#">AppleDash</a> , <a href="#">ArtOfCode</a> , <a href="#">Baldrick</a> , <a href="#">cl3m</a> , <a href="#">Cyril Duchon-Doris</a> , <a href="#">Deepak Mahakale</a> , <a href="#">Dharam</a> , <a href="#">Eliot Sykes</a> , <a href="#">esthervillars</a> , <a href="#">Fabio Ros</a> , <a href="#">Fire-Dragon-DoL</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">giniouxe</a> , <a href="#">Giuseppe</a> , <a href="#">Hassan Akram</a> , <a href="#">Hizqeel</a> , <a href="#">HungryCoder</a> , <a href="#">jkdev</a> , <a href="#">John Slegers</a> , <a href="#">Jon Wood</a> , <a href="#">Kevin Sylvestre</a> , <a href="#">Kieran Andrews</a> , <a href="#">Kirti Thorat</a> , <a href="#">KULKING</a> , <a href="#">Leito</a> , <a href="#">Mario Uher</a> , <a href="#">Milind</a> , <a href="#">Muhammad Faisal Iqbal</a> , <a href="#">niklashultstrom</a> , <a href="#">nuclearpidgeon</a> , <a href="#">pastullo</a> , <a href="#">Rahul Singh</a> , <a href="#">rap-2-h</a> , <a href="#">Raynor Kuang</a> , <a href="#">Richard Hamilton</a> , <a href="#">Robin</a> , <a href="#">rogerdpack</a> , <a href="#">Rory O'Kane</a> , <a href="#">Ryan Hilbert</a> , <a href="#">Ryan K</a> , <a href="#">Silviu Simeria</a> , <a href="#">Simone Carletti</a> , <a href="#">sohail khalil</a> , <a href="#">Stephen Leppik</a> , <a href="#">TheChamp</a> , <a href="#">Thor odinson</a> , <a href="#">Undo</a> , <a href="#">Zoran</a> ,
38	Enrutamiento superficial	<a href="#">Darpan Chhatravala</a>
39	Estados del modelo: AASM	<a href="#">Lomefin</a>
40	Estructuras de rieles a lo largo de los años.	<a href="#">Shivasubramanian A</a>
41	Forma anidada en Ruby on Rails	<a href="#">Arslan Ali</a>
42	Gemas	<a href="#">Deep</a> , <a href="#">hschin</a> , <a href="#">ma_il</a> , <a href="#">MMachinegun</a> , <a href="#">RamenChef</a>
43	Herencia de una sola mesa	<a href="#">Niyanta</a> , <a href="#">Ruslan</a> , <a href="#">Slava.K</a> , <a href="#">toobulkeh</a> , <a href="#">Vishal Taj PM</a>

44	Herramientas para la optimización y limpieza del código de Ruby on Rails	<a href="#">Akshay Borade</a>
45	I18n - Internacionalización	<a href="#">Cyril Duchon-Doris</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">Frederik Spang</a> , <a href="#">gwcodes</a> , <a href="#">Jorge Najera T</a> , <a href="#">Lahiru</a> , <a href="#">RamenChef</a>
46	ID amigable	<a href="#">Thang Le Sy</a>
47	Importar archivos CSV completos desde una carpeta específica	<a href="#">fool</a>
48	Integración de React.js con Rails usando Hyperloop	<a href="#">Mitch VanDuyn</a>
49	Interfaz de consulta ActiveRecord	<a href="#">Adam Lassek</a> , <a href="#">Ajay Barot</a> , <a href="#">Avdept</a> , <a href="#">br3nt</a> , <a href="#">dnsh</a> , <a href="#">Fabio Ros</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">giniouxe</a> , <a href="#">jeffdill2</a> , <a href="#">MikeAndr</a> , <a href="#">Muhammad Abdullah</a> , <a href="#">Niyanta</a> , <a href="#">powerup7</a> , <a href="#">rdnewman</a> , <a href="#">Reboot</a> , <a href="#">Robin</a> , <a href="#">sa77</a> , <a href="#">Vishal Taj PM</a>
50	Los rieles generan comandos.	<a href="#">Adam Lassek</a> , <a href="#">ann</a> , <a href="#">Deepak Mahakale</a> , <a href="#">Dharam</a> , <a href="#">Hardik Upadhyay</a> , <a href="#">jackerman09</a> , <a href="#">Jeremy Green</a> , <a href="#">marcamillion</a> , <a href="#">Milind</a> , <a href="#">Muhammad Abdullah</a> , <a href="#">nomatteus</a> , <a href="#">powerup7</a> , <a href="#">Reub</a> , <a href="#">Richard Hamilton</a>
51	Mejores Prácticas de Rieles	<a href="#">Adam Lassek</a> , <a href="#">Brandon Williams</a> , <a href="#">Gaston</a> , <a href="#">giniouxe</a> , <a href="#">Hardik Upadhyay</a> , <a href="#">inye</a> , <a href="#">Joel Drapper</a> , <a href="#">Josh Caswell</a> , <a href="#">Luka Kerr</a> , <a href="#">ma_il</a> , <a href="#">msohng</a> , <a href="#">Muaaz Rafi</a> , <a href="#">piton4eg</a> , <a href="#">powerup7</a> , <a href="#">rony36</a> , <a href="#">Sri</a> , <a href="#">Tom Lazar</a>
52	Migraciones ActiveRecord	<a href="#">Adam Lassek</a> , <a href="#">Aigars Cibulsksis</a> , <a href="#">Alex Kitchens</a> , <a href="#">buren</a> , <a href="#">Deepak Mahakale</a> , <a href="#">Dharam</a> , <a href="#">DSimon</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">giniouxe</a> , <a href="#">Hardik Kanjariya</a> ♪, <a href="#">hschin</a> , <a href="#">jeffdill2</a> , <a href="#">Kirti Thorat</a> , <a href="#">KULKING</a> , <a href="#">maartenvanvliet</a> , <a href="#">Manish Agarwal</a> , <a href="#">Milo P</a> , <a href="#">Mohamad</a> , <a href="#">MZaragoza</a> , <a href="#">nomatteus</a> , <a href="#">Reboot</a> , <a href="#">Richard Hamilton</a> , <a href="#">rii</a> , <a href="#">Robin</a> , <a href="#">Rodrigo Argumedo</a> , <a href="#">rony36</a> , <a href="#">Rory O'Kane</a> , <a href="#">tessi</a> , <a href="#">uzaif</a> , <a href="#">webster</a>
53	Mongoide	<a href="#">Ryan K</a> , <a href="#">tes</a>
54	Motor de rieles - Rieles modulares	<a href="#">Mayur Shah</a>
55	Oleoducto de activos	<a href="#">fybw id</a> , <a href="#">Robin</a>
56	Organizacion de la	<a href="#">Deep</a> , <a href="#">hadees</a> , <a href="#">HParker</a>

	clase	
57	Palabras reservadas	<a href="#">Emre Kurt</a>
58	Patrón decorador	<a href="#">Adam Lassek</a>
59	Prawn PDF	<a href="#">Awais Shafqat</a>
60	Puntos de vista	<a href="#">danirod</a> , <a href="#">dgilperez</a> , <a href="#">elasticman</a> , <a href="#">Luka Kerr</a> , <a href="#">MikeC</a> , <a href="#">MMachinegun</a> , <a href="#">Pragash</a> , <a href="#">RareFever</a>
61	Rails Cookbook - Recetas / aprendizajes avanzados y técnicas de codificación	<a href="#">Milind</a>
62	Reacciona con los rieles usando la gema reaccion-rails.	<a href="#">Kimmo Hintikka</a> , <a href="#">tirdadc</a>
63	Registrador de carriles	<a href="#">Alejandro Montilla</a> , <a href="#">hgsongra</a>
64	Rieles - motores	<a href="#">Deepak Kabbur</a>
65	Rieles en docker	<a href="#">ppascualv</a> , <a href="#">Sathishkumar Jayaraj</a>
66	RSpec y Ruby on Rails	<a href="#">Ashish Bista</a> , <a href="#">Scott Matthewman</a> , <a href="#">Simone Carletti</a>
67	Serializadores de modelos activos	<a href="#">Flip</a> , <a href="#">owahab</a>
68	Trabajos activos	<a href="#">tirdadc</a>
69	Transacciones ActiveRecord	<a href="#">abhas</a> , <a href="#">Adam Lassek</a>
70	Turbolinks	<a href="#">Mark</a>
71	Usando GoogleMaps con Rails	<a href="#">fiedl</a>
72	Validación de ActiveRecord	<a href="#">Adam Lassek</a> , <a href="#">Colin Herzog</a> , <a href="#">Deepak Mahakale</a> , <a href="#">dgilperez</a> , <a href="#">dodo121</a> , <a href="#">ginioux</a> , <a href="#">Hai Pandu</a> , <a href="#">Hardik Upadhyay</a> , <a href="#">mmichael</a> , <a href="#">Muhammad Abdullah</a> , <a href="#">pablofullana</a> , <a href="#">Richard Hamilton</a>