Kostenloses eBook

LERNEN

Ruby on Rails

Free unaffiliated eBook created from
Stack Overflow contributors.

#ruby-on-rails

Inhaltsverzeichnis

Über ... 1

Kapitel 1: Erste Schritte mit Ruby on Rails ... 2

Bemerkungen ... 2

Versionen ... 2

Examples ... 3

Erstellen einer Ruby on Rails-Anwendung ... 3

Erstellen Sie eine neue Rails-App mit Ihrer Datenbankauswahl und dem RSpec Testing Tool ... 5

Einen Controller generieren ... 6

Generieren Sie eine Ressource mit Gerüsten ... 7

Erstellen Sie eine neue Rails-App mit einem nicht standardmäßigen Datenbankadapter ... 7

Erstellen von Rails-APIs in JSON ... 8

Schienen installieren ... 9

Kapitel 2: ActionCable ... 12

Bemerkungen ... 12

Examples ... 12

[Basic] Serverseite ... 12

[Basic] Client-Seite (Coffeescript) ... 12

app / assets / javascripts / channels / notifications.coffee ... 12

app / assets / javascripts / application.js # wird normalerweise wie folgt generiert ... 12

app / assets / javascripts / cable.js # wird normalerweise so generiert ... 13

Benutzerauthentifizierung ... 13

Kapitel 3: ActionController ... 14

Einführung ... 14

Examples ... 14

Ausgabe von JSON anstelle von HTML ... 14

Steuerungen (Basic) ... 15

Parameter ... 15

Filterparameter (Basic) ... 15

Umleitung ... 16

Ansichten verwenden ... 16

404 wenn Datensatz nicht gefunden wurde ... 18
Grundlegender REST-Controller ... 18
Fehlerseiten für Ausnahmen anzeigen ... 19
Filter ... 20
Controller generieren ... 22
ActiveRecord :: RecordNotFound mit redirect_to retten ... 24
Kapitel 4: ActionMailer ... 25
Einführung ... 25
Bemerkungen ... 25
Examples ... 25
Basic Mailer ... 25
user_mailer.rb ... 25
user.rb ... 25
approved.html.erb ... 26
approved.text.erb ... 26
Neuen Mailer generieren ... 26
Anhänge hinzufügen ... 26
ActionMailer-Rückrufe ... 27
Generieren Sie einen geplanten Newsletter ... 27
ActionMailer-Interceptor ... 34
Kapitel 5: ActiveJob ... 36
Einführung ... 36
Examples ... 36
Erstellen Sie den Job ... 36
Den Job in die Warteschlange stellen ... 36
Kapitel 6: ActiveModel ... 37
Bemerkungen ... 37
Examples ... 37
ActiveModel :: Validierungen verwenden ... 37
Kapitel 7: ActiveRecord-Abfrage-Schnittstelle ... 38
Einführung ... 38

Examples 38
.woher 38
.wo mit einem Array 39
Bereiche 39
wo nicht 40
Bestellung 40
ActiveRecord Bang (!) -Methoden 41
.find_by 42
.alles löschen 42
Bei ActiveRecord wird die Groß- / Kleinschreibung nicht berücksichtigt 42
Holen Sie sich den ersten und letzten Datensatz 43
.group und .count 44
.distinct (oder .uniq) 44
Schließt sich an 44
Enthält 45
Limit und Offset 45
Kapitel 8: ActiveRecord-Migrationen 47
Parameter 47
Bemerkungen 47
Examples 47
Führen Sie eine bestimmte Migration aus 48
Erstellen Sie eine Join-Tabelle 48
Migrationen in verschiedenen Umgebungen ausführen 48
Fügen Sie einer Tabelle eine neue Spalte hinzu 49
Fügen Sie eine neue Spalte mit einem Index hinzu 49
Entfernen Sie eine vorhandene Spalte aus einer Tabelle 49
Fügen Sie einer Tabelle eine Referenzspalte hinzu 50
Erstellen Sie eine neue Tabelle 51
Mehrere Spalten zu einer Tabelle hinzufügen 51
Migrationen ausführen 51
Rollback-Migrationen 52
Rollback der letzten 3 Migrationen 52
Alle Migrationen rückgängig machen 52

Tabellen wechseln ... 53
Fügen Sie einer Tabelle eine eindeutige Spalte hinzu ... 53
Ändern Sie den Typ einer vorhandenen Spalte ... 53
Eine längere aber sicherere Methode ... 54
Migrationen wiederholen ... 54
Spalte mit Standardwert hinzufügen ... 54
Nullwerte verbieten ... 55
Migrationsstatus überprüfen ... 55
Erstellen Sie eine Hstore-Spalte ... 56
Fügen Sie einen Selbstverweis hinzu ... 56
Erstellen Sie eine Array-Spalte ... 56
Hinzufügen einer NOT NULL-Einschränkung zu vorhandenen Daten ... 57
Kapitel 9: ActiveRecord-Sperrung ... 58
Examples ... 58
Optimistisches Sperren ... 58
Pessimistisches Sperren ... 58
Kapitel 10: ActiveRecord-Transaktionen ... 59
Bemerkungen ... 59
Examples ... 59
Grundlegendes Beispiel ... 59
Verschiedene ActiveRecord-Klassen in einer einzelnen Transaktion ... 59
Mehrere Datenbankverbindungen ... 60
Speichern und Zerstören werden automatisch in eine Transaktion eingeschlossen ... 60
Rückrufe ... 60
Transaktion rückgängig machen ... 61
Kapitel 11: ActiveRecord-Transaktionen ... 62
Einführung ... 62
Examples ... 62
Erste Schritte mit aktiven Datensatztransaktionen ... 62
Kapitel 12: ActiveRecord-Überprüfungen ... 63
Examples ... 63
Überprüfung der Numerizität eines Attributs ... 63

Überprüfen Sie die Eindeutigkeit eines Attributs ... 63
Vorhandensein eines Attributs prüfen ... 64
Validierungen überspringen ... 64
Länge eines Attributs überprüfen ... 65
Gruppierungsüberprüfung ... 65
Benutzerdefinierte Validierungen ... 65
ActiveModel::Validator und validates_with ... 66
ActiveModel::EachValidator und validate ... 66
Überprüft das Format eines Attributs ... 66
Überprüft die Aufnahme eines Attributs ... 67
Bedingte Validierung ... 67
Bestätigung des Attributs ... 68
Verwendung: bei Option ... 68
Kapitel 13: ActiveRecord-Verknüpfungen ... 70
Examples ... 70
gehört ... 70
has_one ... 70
hat viele ... 71
Polymorphe Verbindung ... 71
Die has_many: durch Vereinigung ... 72
Die has_one: durch Assoziation ... 72
Die has_and_belongs_to_many-Verbindung ... 73
Selbstreferentielle Vereinigung ... 73
Kapitel 14: ActiveSupport ... 74
Bemerkungen ... 74
Examples ... 74
Core-Erweiterungen: String-Zugriff ... 74
String # at ... 74
String # von ... 74
Zeichenfolge # bis ... 74
String # zuerst ... 75
String # last ... 75

Core-Erweiterungen: Konvertierung von Zeichenfolgen in Datum / Uhrzeit 75
String # to_time 75
String # to_date 75
Zeichenfolge # bis_datetime 76
Core-Erweiterungen: Zeichenkettenausschluss 76
Zeichenfolge # ausschließen? 76
Core-Erweiterungen: String-Filter 76
String # squish 76
Zeichenfolge # entfernen 76
String # wird abgeschnitten 77
String # truncate_words 77
Zeichenfolge # strip_heredoc 77
Core-Erweiterungen: String Flexion 78
String # pluralize 78
String # singularize 78
String # konstante 79
String # safe_constantize 79
Zeichenfolge # camelize 79
String # titleize 79
String # unterstreichen 80
String # dasherize 80
String # demodulize 80
String # dekonstantisieren 80
String # parametrisieren 80
String # tableize 81
String # klassifizieren 81
String # humanize 81
String # upcase_first 81
String # foreign_key 82

Kapitel 15: Admin-Panel hinzufügen 83
Einführung 83
Syntax 83
Bemerkungen 83
Examples 83
Hier sind also ein paar Screenshots aus dem Admin-Panel, die die Schienen "schienen_Admin" 83
Kapitel 16: Aktive Jobs 87
Examples 87
Einführung 87
Musterjob 87
Aktiven Job über den Generator erstellen 87
Kapitel 17: Aktive Modell-Serialisierer 88
Einführung 88
Examples 88
Verwenden eines Serializers 88
Kapitel 18: Aktiver Rekord 89
Examples 89
Modell manuell erstellen 89
Modell über Generator erstellen 89
Eine Migration erstellen 90
Felder in vorhandenen Tabellen hinzufügen / entfernen 90
Erstellen Sie eine Tabelle 91
Erstellen Sie eine Join-Tabelle 91
Vorrang 91
Einführung in Rückrufe 92
Erstellen Sie eine Join-Tabelle mithilfe von Migrationen 93
Manuelles Testen Ihrer Modelle 93
Verwenden einer Modellinstanz zum Aktualisieren einer Zeile 94
Kapitel 19: Ändern Sie eine Standardumgebung für Rails-Anwendungen 95
Einführung 95
Examples 95

Läuft auf einem lokalen Rechner ... 95
Läuft auf einem Server ... 95
Kapitel 20: Ansichten ... 96
Examples ... 96
Teilstücke ... 96
Objektpartials ... 96
Globale Partials ... 96
AssetTagHelper ... 97
Bildhelfer ... 97
Bildpfad ... 97
Bild URL ... 97
image_tag ... 97
JavaScript-Helfer ... 97
javascript_include_tag ... 97
Javascript_Pfad ... 98
javascript_url ... 98
Stylesheet-Helfer ... 98
stylesheet_link_tag ... 98
stylesheet_path ... 98
stylesheet_url ... 98
Verwendungsbeispiel ... 98
Struktur ... 99
Ersetzen Sie HTML-Code in Ansichten ... 99
HAML - eine alternative Möglichkeit, Ihre Ansichten zu verwenden ... 100
Kapitel 21: Asset-Pipeline ... 102
Einführung ... 102
Examples ... 102
Rechenaufgaben ... 102
Manifestdateien und -richtlinien ... 102
Grundlegende Verwendung ... 103
Kapitel 22: Aufbau ... 104

Examples ... 104
Benutzerdefinierte Konfiguration ... 104
Kapitel 23: Aufbau ... 106
Examples ... 106
Umgebungen in Schienen ... 106
Datenbankkonfiguration ... 106
Rails Allgemeine Konfiguration ... 107
Assets konfigurieren ... 107
Generatoren konfigurieren ... 108
Kapitel 24: Authentifizieren Sie die API mit Devise ... 109
Einführung ... 109
Examples ... 109
Fertig machen ... 109
Authentifizierungs-Token ... 109
Kapitel 25: Autorisierung mit CanCan ... 112
Einführung ... 112
Bemerkungen ... 112
Examples ... 112
Erste Schritte mit CanCan ... 112
Fähigkeiten definieren ... 113
Umgang mit einer großen Anzahl von Fähigkeiten ... 113
Testen Sie schnell eine Fähigkeit ... 115
Kapitel 26: Benutzerauthentifizierung in Rails ... 116
Einführung ... 116
Bemerkungen ... 116
Examples ... 116
Authentifizierung mit Devise ... 116
Benutzerdefinierte Ansichten ... 117
Entwickeln Sie Controller-Filter und Helfer ... 117
Omniauth ... 117
has_secure_password ... 118
Erstellen Sie ein Benutzermodell ... 118

Fügen Sie dem Benutzermodell das Modul has_secure_password hinzu ... 118
has_secure_token ... 118
Kapitel 27: Bereitstellung einer Rails-App auf Heroku ... 120
Examples ... 120
Bereitstellung Ihrer Anwendung ... 120
Produktions- und Inszenierungsumgebungen für ein Heroku verwalten ... 123
Kapitel 28: Bezahlfunktion in Schienen ... 125
Einführung ... 125
Bemerkungen ... 125
Examples ... 125
So integrieren Sie Stripe ... 125
So erstellen Sie einen neuen Kunden für Stripe ... 125
So rufen Sie einen Plan von Stripe ab ... 126
So erstellen Sie ein Abonnement ... 126
So belasten Sie einen Benutzer mit einer einzigen Zahlung ... 126
Kapitel 29: Caching ... 128
Examples ... 128
Russisches Puppen-Caching ... 128
SQL-Caching ... 128
Fragment-Caching ... 129
Zwischenspeicherung der Seite ... 130
HTTP-Caching ... 130
Action-Caching ... 131
Kapitel 30: Datei-Uploads ... 132
Examples ... 132
Einzelnes Datei-Upload mit Carrierwave ... 132
Geschachteltes Modell - mehrere Uploads ... 132
Kapitel 31: Debuggen ... 134
Examples ... 134
Debuggen der Rails-Anwendung ... 134
Debuggen in Ihrer IDE ... 134

Schnelles Debuggen von Ruby on Rails + Beratung für Anfänger 136
Ruby / Rails schnell debuggen: 136
1. Fast-Methode: Dann eine Exception .inspect und das Ergebnis .inspect 136
2. Fallback: Verwenden Sie einen Rubin- IRB- Debugger wie byebug oder pry 136
Allgemeine Anfängerhinweise 137
ZB eine Ruby-Fehlermeldung, die viele Anfänger verwirrt: 137
Debuggen der Ruby-on-Rails-Anwendung mit pry 138
Kapitel 32: Dekorateur-Muster 141
Bemerkungen 141
Examples 141
Dekorieren eines Modells mit SimpleDelegator 141
Dekorieren eines Modells mit Draper 142
Kapitel 33: Edelsteine 143
Bemerkungen 143
Gemfile-Dokumentation 143
Examples 143
Was ist ein Juwel? 143
In Ihrem Rails-Projekt 143
Gemfile 143
Gemfile.lock 143
Entwicklung 144
Bundler 144
Gemfiles 144
Gemsets 145
Kapitel 34: Elasticsearch 148
Examples 148
Installation und Prüfung 148
Werkzeuge für die Entwicklung einrichten 148
Einführung 149
Suchkick 149
Kapitel 35: Fabrikmädchen 151

Examples ... 151
Fabriken definieren ... 151
Kapitel 36: Flaches Routing ... 152
Examples ... 152
1. Verwendung von Flachwasser ... 152
Kapitel 37: Freundliche ID ... 153
Einführung ... 153
Examples ... 153
Rails Schnellstart ... 153
Gemfile ... 153
Bearbeiten Sie App / Modelle / Benutzer.rb ... 153
h11 ... 153
h12 ... 153
Kapitel 38: Garnelen-PDF ... 155
Examples ... 155
Erweitertes Beispiel ... 155
Basisbeispiel ... 156
Dies ist die Grundaufgabe ... 156
Wir können das mit Implicit Block tun ... 156
Mit explizitem Block ... 156
Kapitel 39: GoogleMaps mit Rails verwenden ... 157
Examples ... 157
Fügen Sie den Javascript-Tag von Google Maps zum Layout-Header hinzu ... 157
Geokodieren Sie das Modell ... 158
Zeigen Sie Adressen auf einer Google-Karte in der Profilansicht an ... 158
Setzen Sie die Markierungen auf der Karte mit Javascript ... 159
Initialisieren Sie die Karte mit einer Kaffeeskriptklasse ... 160
Initialisieren Sie die Kartenmarkierungen mithilfe einer Kaffeeskriptklasse ... 161
Auto-Zoom einer Karte mit einer Kaffeeskriptklasse ... 162
Darstellen der Modelleigenschaften als Json ... 162
Regelmäßige Datenbankattribute ... 162

Andere Attribute ... 163
Position ... 163
Kapitel 40: Helfer bilden ... 165
Einführung ... 165
Bemerkungen ... 165
Examples ... 165
Erstellen Sie ein Formular ... 165
Suchformular erstellen ... 165
Helfer für Formularelemente ... 166
Ankreuzfelder ... 166
Radio Knöpfe ... 166
Textbereich ... 166
Zahlenfeld ... 167
Passwortfeld ... 167
E-Mail-Feld ... 167
Telefonfeld ... 167
Datum Helfer ... 167
Dropdown-Liste ... 168
Kapitel 41: Hinzufügen eines Amazon RDS zu Ihrer Rails-Anwendung ... 169
Einführung ... 169
Examples ... 169
Stellen Sie sich vor, wir verbinden MYSQL RDS mit Ihrer Schienenanwendung ... 169
Kapitel 42: I18n - Internationalisierung ... 171
Syntax ... 171
Examples ... 171
Verwenden Sie I18n in Ansichten ... 171
I18n mit Argumenten ... 171
Pluralisierung ... 172
Festlegen des Gebietsschemas durch Anforderungen ... 172
URL-basiert ... 173
Sitzungsbasiert oder persistenzbasiert ... 173

Standardgebietsschema ... 174
Holen Sie sich das Gebietsschema von der HTTP-Anforderung ... 175
Einschränkungen und Alternativen ... 175
1. Eine Offline-Lösung ... 175
2. Verwenden Sie CloudFlare ... 175
Attribute des ActiveRecord-Modells übersetzen ... 176
Verwenden Sie I18n mit HTML-Tags und -Symbolen ... 178
Kapitel 43: Importieren Sie ganze CSV-Dateien aus einem bestimmten Ordner ... 179
Einführung ... 179
Examples ... 179
Lädt CSV vom Konsolenbefehl hoch ... 179
Kapitel 44: Klassenorganisation ... 181
Bemerkungen ... 181
Examples ... 181
Modellklasse ... 181
Serviceklasse ... 182
Kapitel 45: Mehrzweck-ActiveRecord-Spalten ... 185
Syntax ... 185
Examples ... 185
Objekt speichern ... 185
Wie man ... 185
In deiner Migration ... 185
In deinem Modell ... 185
Kapitel 46: Modellzustände: AASM ... 186
Examples ... 186
Grundzustand bei AASM ... 186
Kapitel 47: Mongoid ... 189
Examples ... 189
Installation ... 189
Ein Modell erstellen ... 189
Felder ... 190

Klassische Assoziationen ... 190
Eingebettete Assoziationen ... 191
Datenbankaufrufe ... 191
Kapitel 48: Rails 5-API-Authentifizierung ... 192
Examples ... 192
Authentifizierung mit Rails authenticate_with_http_token ... 192
Kapitel 49: Rails auf Docker ... 193
Einführung ... 193
Examples ... 193
Docker und Docker komponieren ... 193
Kapitel 50: Rails aufrüsten ... 195
Examples ... 195
Upgrade von Rails 4.2 auf Rails 5.0 ... 195
Kapitel 51: Rails Best Practices ... 197
Examples ... 197
Wiederholen Sie sich nicht (TROCKEN) ... 197
Konvention über Konfiguration ... 197
Fettes Modell, dünner Controller ... 198
Hüten Sie sich vor default_scope ... 199
default_scope und order ... 199
default_scope und Modellinitialisierung ... 199
unscoped ... 200
unscoped und Model Associations ... 200
Ein Anwendungsbeispiel für default_scope ... 201
Du wirst es nicht brauchen (YAGNI) ... 201
Probleme ... 201
Übertechnik ... 201
Code aufgebläht ... 202
Feature Kriechen ... 202
Lange Entwicklungszeit ... 202
Lösungen ... 202

KISS - Mach es einfach, dumm ... 202
YAGNI - Du wirst es nicht brauchen ... 202
Kontinuierliches Refactoring ... 202
Domänenobjekte (keine fetten Modelle) ... 202
Kapitel 52: Rails Engine - Modulare Schienen ... 206
Einführung ... 206
Syntax ... 206
Examples ... 206
Erstellen Sie eine modulare App ... 206
Erstellen der Todo-Liste ... 207
Kapitel 53: Rails erzeugen Befehle ... 209
Einführung ... 209
Parameter ... 209
Bemerkungen ... 209
Examples ... 210
Schienen erzeugen Modell ... 210
Schienen erzeugen Migration ... 210
Schienen erzeugen Gerüst ... 211
Schienen generieren Controller ... 212
Kapitel 54: Rails-API ... 213
Examples ... 213
Nur-API-Anwendung erstellen ... 213
Kapitel 55: React.js mithilfe von Hyperloop in Rails integrieren ... 214
Einführung ... 214
Bemerkungen ... 214
Examples ... 214
Hinzufügen einer einfachen Reaktionskomponente (in Ruby geschrieben) zu Ihrer Rails-App ... 214
Komponentenparameter deklarieren (Requisiten) ... 215
HTML-Tags ... 215
Event-Handler ... 215
Zustände ... 216
Rückrufe ... 216

Kapitel 56: Reagiere mit Rails mit Reaktiver Gleis gem 217
Examples 217
Reaktiver Einbau für Rails mithilfe von schienen_reakt gem 217
Verwenden von react_rails in Ihrer Anwendung 217
Rendern & Montieren 218
Kapitel 57: Regeln der Namensgebung 220
Examples 220
Controller 220
Modelle 220
Ansichten und Layouts 220
Dateinamen und automatisches Laden 221
Modellklasse vom Controller-Namen 222
Kapitel 58: Reservierte Wörter 223
Einführung 223
Examples 223
Reservierte Wortliste 223
Kapitel 59: Routing 230
Einführung 230
Bemerkungen 230
Examples 230
Ressourcenrouting (Basic) 230
Einschränkungen 232
Scoping-Routen 234
Sorgen 237
Umleitung 238
Mitglieder- und Sammelrouten 238
URL-Params mit einem Punkt 239
Wurzelroute 239
Zusätzliche RESTful-Aktionen 240
Verfügbare Gebietsschemas 240
Mounten Sie eine andere Anwendung 241
Weiterleitungen und Wildcard-Routen 241

Trennen Sie die Routen in mehrere Dateien ... 241
Verschachtelte Routen ... 242
Kapitel 60: RSpec und Ruby on Rails ... 243
Bemerkungen ... 243
Examples ... 243
RSpec installieren ... 243
Kapitel 61: Schienen 5 ... 244
Examples ... 244
Erstellen einer Ruby on Rails 5-API ... 244
So installieren Sie Ruby on Rails 5 auf RVM ... 246
Kapitel 62: Schienengerüste im Laufe der Jahre ... 247
Einführung ... 247
Examples ... 247
Wie finde ich heraus, welche Frameworks in der aktuellen Version von Rails verfügbar sind? ... 247
Rails-Versionen in Rails 1.x ... 247
Schienengerüste in Schienen 2.x ... 247
Schienengerüste in Schienen 3.x ... 247
Kapitel 63: Schienen-Kochbuch - Fortgeschrittene Schienen-Rezepte / -Lernen und Kodierungs 249
Examples ... 249
Spielen mit Tischen unter Verwendung der Schienenkonsole ... 249
Rails-Methoden - Rückgabe boolescher Werte ... 250
Umgang mit dem Fehler - undefined Methode `where 'für # ... 250
Kapitel 64: Schienenlogger ... 251
Examples ... 251
Rails.logger ... 251
Kapitel 65: Schienen-Motoren ... 253
Einführung ... 253
Syntax ... 253
Parameter ... 253
Bemerkungen ... 253
Examples ... 253

Berühmte Beispiele sind ... 253
Kapitel 66: Sichere Speicherung von Authentifizierungsschlüsseln ... 255
Einführung ... 255
Examples ... 255
Authentifizierungsschlüssel mit Figaro speichern ... 255
Kapitel 67: Sicheres Konstantisieren ... 257
Examples ... 257
Erfolgreiche safe_constantize ... 257
Nicht erfolgreich safe_constantize ... 257
Kapitel 68: Standardzeitzone ändern ... 258
Bemerkungen ... 258
Examples ... 258
Ändern Sie die Zeitzone von Rails, speichern Sie jedoch weiterhin Active Record in der Dat ... 258
Rails-Zeitzone ändern UND in dieser Zeitzone Active Record-Speicherzeiten festlegen ... 259
Kapitel 69: Testen von Rails-Anwendungen ... 260
Examples ... 260
Gerätetest ... 260
Test anfordern ... 260
Kapitel 70: Tools für die Codeoptimierung und -bereinigung von Ruby on Rails ... 261
Einführung ... 261
Examples ... 261
Wenn Sie Ihren Code wartungsfähig, sicher und optimiert halten möchten, sehen Sie sich ein ... 261
Kapitel 71: Turbolinks ... 263
Einführung ... 263
Bemerkungen ... 263
Die zentralen Thesen: ... 263
Examples ... 263
Bindung an das Konzept von turbolink zum Laden einer Seite ... 263
Deaktivieren Sie Turbolinks für bestimmte Links ... 264
Beispiele: ... 264
Anwendungsbesuche verstehen ... 264

Besuche werden abgebrochen, bevor sie beginnen 265
HINWEIS: 265
Bestehende Elemente beim Laden von Seiten 265
Kapitel 72: Vererbung einzelner Tabellen 267
Einführung 267
Examples 267
Grundlegendes Beispiel 267
Benutzerdefinierte Vererbungsspalte 268
Schienenmodell mit Typsäule und ohne STI 268
Kapitel 73: Verschachtelte Form in Ruby on Rails 269
Examples 269
So erstellen Sie ein verschachteltes Formular in Ruby on Rails 269
Kapitel 74: Winkel mit Schienen konfigurieren 271
Examples 271
Winkel mit Schienen 101 271
Schritt 1: Erstellen Sie eine neue Rails-App 271
Schritt 2: Turbolinks entfernen 271
Schritt 3: Fügen Sie AngularJS zur Asset-Pipeline hinzu 271
Schritt 4: Organisieren Sie die Angular App 272
Schritt 5: Starten Sie die Angular App 272
Credits 274

Über

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.

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

Kapitel 1: Erste Schritte mit Ruby on Rails

Bemerkungen

Ruby on Rails (RoR) oder Rails ist ein beliebtes Open-Source-Framework für Webanwendungen. Rails verwendet Ruby, HTML, CSS und JavaScript, um eine Webanwendung zu erstellen, die auf einem Webserver ausgeführt wird. Rails verwendet das Model-View-Controller-Muster (MVC) und bietet eine Fülle von Bibliotheken von der Datenbank bis zur Ansicht.

Versionen

Ausführung	Veröffentlichungsdatum
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

Ausführung	Veröffentlichungsdatum
1,0	2005-12-13

Examples

Erstellen einer Ruby on Rails-Anwendung

In diesem Beispiel wird davon *ausgegangen,* dass *Ruby* und *Ruby on Rails* bereits ordnungsgemäß installiert wurden. Wenn nicht, können Sie **hier** nachlesen, wie es **geht** .

Öffnen Sie eine Befehlszeile oder ein Terminal. Um eine neue Schienenanwendung zu generieren, verwenden Sie den neuen Befehl schienen gefolgt vom Namen Ihrer Anwendung:

```
$ rails new my_app
```

Wenn Sie Ihre Rails-Anwendung mit einer bestimmten Rails-Version erstellen möchten, können Sie diese zum Zeitpunkt der Erstellung der Anwendung angeben. Verwenden Sie dazu `rails _version_ new` gefolgt vom Namen der Anwendung:

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

Dadurch wird eine Rails-Anwendung namens `MyApp` in einem `my_app` Verzeichnis erstellt und die `my_app` Abhängigkeiten, die bereits in `Gemfile` mithilfe der `Gemfile bundle install` .

Um zum Verzeichnis Ihrer neu erstellten App zu wechseln, verwenden Sie den Befehl `cd` (`change directory` .

```
$ cd my_app
```

Das Verzeichnis `my_app` enthält eine Reihe automatisch generierter Dateien und Ordner, aus denen sich die Struktur einer Rails-Anwendung zusammensetzt. Es folgt eine Liste der Dateien und Ordner, die standardmäßig erstellt werden:

Aktenordner	Zweck
App /	Enthält die Controller, Modelle, Ansichten, Helfer, Mailer und Assets für Ihre Anwendung.
Behälter/	Enthält das Rails-Skript, das Ihre App startet, und kann andere Skripts enthalten, die Sie zum Einrichten, Aktualisieren, Bereitstellen oder Ausführen Ihrer Anwendung verwenden.
config /	Konfigurieren Sie die Routen, die Datenbank usw. Ihrer Anwendung.
config.ru	Rack-Konfiguration für Rack-basierte Server, die zum Starten der Anwendung verwendet werden.

Aktenordner	Zweck
db /	Enthält Ihr aktuelles Datenbankschema sowie die Datenbankmigrationen.
Gemfile Gemfile.lock	Mit diesen Dateien können Sie angeben, welche Edelsteinabhängigkeiten für Ihre Rails-Anwendung erforderlich sind. Diese Dateien werden vom Bundler-Gem verwendet.
lib /	Erweiterte Module für Ihre Anwendung.
Log/	Anwendungsprotokolldateien.
Öffentlichkeit/	Der einzige Ordner, der von der Welt so gesehen wird. Enthält statische Dateien und kompilierte Assets.
Rakefile	Diese Datei sucht und lädt Aufgaben, die von der Befehlszeile aus ausgeführt werden können. Die Aufgabendefinitionen werden in allen Komponenten von Rails definiert.
README.md	Dies ist eine kurze Anleitung für Ihre Anwendung. Sie sollten diese Datei bearbeiten, um anderen Benutzern mitzuteilen, was Ihre Anwendung tut, wie sie eingerichtet werden soll
Prüfung/	Komponententests, Vorrichtungen und andere Testgeräte.
Temp /	Temporäre Dateien (wie Cache- und PID-Dateien).
Verkäufer/	Ein Ort für den Code von Drittanbietern. In einer typischen Rails-Anwendung gehören dazu auch Edelsteine.

Jetzt müssen Sie eine Datenbank aus Ihrer `database.yml` Datei erstellen:

5,0

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

5,0

```
rake db:create
```

Nachdem wir die Datenbank erstellt haben, müssen wir Migrationen ausführen, um die Tabellen einzurichten:

5,0

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

5,0

```
rake db:migrate
```

Um die Anwendung zu starten, müssen wir den Server hochfahren:

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

Rails startet die Anwendung standardmäßig an Port 3000. Um die Anwendung mit einer anderen Portnummer zu starten, müssen wir den Server hochfahren, z.

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

Wenn Sie in Ihrem Browser zu http: // localhost: 3000 navigieren, wird eine Willkommensseite von Rails angezeigt, auf der angezeigt wird, dass Ihre Anwendung jetzt ausgeführt wird.

Wenn ein Fehler ausgegeben wird, können mehrere Probleme auftreten:

- Es gibt ein Problem mit der `config/database.yml`
- Sie haben Abhängigkeiten in Ihrer `Gemfile` , die nicht installiert wurden.
- Sie haben ausstehende Migrationen. `rails db:migrate`
- Falls Sie zu den vorherigen Migrationsschienen wechseln, `rails db:rollback`

Wenn das immer noch einen Fehler ausgibt, sollten Sie Ihre `config/database.yml` überprüfen

Erstellen Sie eine neue Rails-App mit Ihrer Datenbankauswahl und dem RSpec Testing Tool

Rails verwendet `sqlite3` als Standarddatenbank. Sie können jedoch eine neue Rails-Anwendung mit einer Datenbank Ihrer Wahl erstellen. Fügen Sie einfach die Option `-d` gefolgt vom Namen der Datenbank.

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

Dies ist eine (nicht erschöpfende) Liste der verfügbaren Datenbankoptionen:

- Mysql
- Orakel
- postgresql
- sqlite3
- Frontbase
- ibm_db
- SQL Server
- jdbcmysql
- jdbcsqlite3
- jdbcpostgresql
- JDBC

Der Befehl -T zeigt an, dass die Installation von Minitest übersprungen wird. Um eine alternative Testsuite wie RSpec zu installieren, bearbeiten Sie die Gemfile und fügen Sie sie hinzu

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

Starten Sie dann den folgenden Befehl von der Konsole aus:

```
rails generate rspec:install
```

Einen Controller generieren

Um einen Controller zu generieren (z. B. `Posts`), navigieren Sie von einer Befehlszeile oder einem Terminal zu Ihrem Projektverzeichnis und führen Sie Folgendes aus:

```
$ rails generate controller Posts
```

Sie können diesen Code verkürzen, indem Sie `generate` durch `g` ersetzen. Beispiel:

```
$ rails g controller Posts
```

Wenn Sie die neu generierte app / controller / **posts_controller.rb öffnen** , wird ein Controller ohne Aktionen **angezeigt** :

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

Es ist möglich, Standardmethoden für den Controller zu erstellen, indem Sie Controller-Namensargumente übergeben.

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

Um einen Controller in einem Modul zu erstellen, geben Sie den Controller-Namen als Pfad an, wie `parent_module/controller_name` . Zum Beispiel:

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

Dadurch werden die folgenden Dateien generiert:

```
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
```

Ein Controller ist einfach eine Klasse, die definiert ist, um von `ApplicationController` zu erben.

In dieser Klasse definieren Sie Methoden, die zu den Aktionen für diesen Controller werden.

Generieren Sie eine Ressource mit Gerüsten

Von guides.rubyonrails.org:

> Anstatt ein Modell direkt zu generieren. . . Lassen Sie uns ein Gerüst aufstellen. Ein Gerüst in Rails umfasst einen vollständigen Satz von Modellen, die Datenbankmigration für dieses Modell, den Controller, der es manipuliert, Ansichten zum Anzeigen und Bearbeiten der Daten und eine Testsuite für jedes der oben genannten.

Hier ein Beispiel für ein Gerüst einer Ressource namens `Task` mit einem String-Namen und einer Textbeschreibung:

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

Dadurch werden die folgenden Dateien generiert:

```
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
```

Beispiel zum Löschen von Dateien, die vom Scaffold für die Ressource `Task` generiert wurden

```
rails destroy scaffold Task
```

Erstellen Sie eine neue Rails-App mit einem nicht standardmäßigen Datenbankadapter

Rails ist standardmäßig mit ausgeliefert `ActiveRecord` ein ORM (Object Relational Mapping) aus dem Muster abgeleitet mit dem gleichen Namen .

Als ORM ist es darauf ausgelegt, relationales Mapping zu handhaben, und genauer durch die Verarbeitung von SQL-Anforderungen für Sie, daher die Beschränkung auf SQL-Datenbanken.

Sie können jedoch weiterhin eine Rails-App mit einem anderen Datenbankverwaltungssystem

erstellen:

1. Erstellen Sie einfach Ihre App ohne Active-Record

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

2. `Gemfile` Sie Ihr eigenes Datenbankverwaltungssystem in `Gemfile`

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

3. `bundle install` und befolgen Sie die Installationsschritte in der gewünschten Datenbank.

In diesem Beispiel handelt es sich bei `mongoid` um eine Objektzuordnung für `MongoDB` Wie viele andere für Rails gebaute Datenbank- `ActiveModel` erbt auch `ActiveModel` die gleiche Methode wie `ActiveRecord` , die eine gemeinsame Schnittstelle für viele Funktionen wie Validierungen, Rückrufe, Übersetzungen usw. bietet .

Zu anderen Datenbankadaptern gehören unter anderem:

- Datamapper
- Folgeschienen

Erstellen von Rails-APIs in JSON

In diesem Beispiel wird davon ausgegangen, dass Sie Erfahrung mit der Erstellung von Rails-Anwendungen haben.

Führen Sie den folgenden Befehl aus, um eine API-only-App in Rails 5 zu erstellen

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

Fügen Sie active_model_serializers in Gemfile hinzu

```
gem 'active_model_serializers'
```

Installieren Sie das Bundle im Terminal

```
bundle install
```

`ActiveModelSerializer` Sie den `ActiveModelSerializer` Adapter so ein, dass er verwendet werden soll `: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 )
```

Generieren Sie ein neues Gerüst für Ihre Ressource

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

Dadurch werden die folgenden Dateien generiert:

Controller: App / Controller / Aufgabencontroller.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
```

Schienen installieren

Rails auf Ubuntu installieren

Auf einem sauberen Ubuntu sollte die Installation von Rails einfach sein

Aktualisieren von Ubuntu-Paketen

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

Installieren Sie Ruby und Rails Abhängigkeiten

```
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
```

Ruby-Versionsmanager installieren In diesem Fall verwendet rbenv das einfachste

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

Ruby Build installieren

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

Shell neu starten

```
exec $SHELL
```

Ruby installieren

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

```
rbenv rehash
```

Schienen montieren

```
gem install rails
```

Rails unter Windows installieren

Schritt 1: ***Ruby installieren***

Wir benötigen die Programmiersprache Ruby. Wir können eine vorkompilierte Version von Ruby mit dem Namen RubyInstaller verwenden.

- Laden Sie Ruby Installer von rubyinstaller.org herunter und führen Sie es aus.
- Führen Sie das Installationsprogramm aus. Aktivieren Sie das Kontrollkästchen "Ruby-ausführbare Dateien zu Ihrem PATH hinzufügen" und installieren Sie dann.
- Um auf Ruby zuzugreifen, öffnen Sie das Windows-Menü, klicken Sie auf Alle Programme, gehen Sie zu Ruby und klicken Sie auf "Eingabeaufforderung mit Ruby starten". Ein Eingabeaufforderungsterminal wird geöffnet. Wenn Sie `ruby -v` und die Eingabetaste drücken, sollte die installierte Ruby-Versionsnummer angezeigt werden.

Schritt 2: ***Ruby Development Kit***

Nach der Installation von Ruby können wir versuchen, Rails zu installieren. Einige Bibliotheken, die Rails benötigt, benötigen einige Build-Tools, um kompiliert zu werden, und Windows fehlen diese Tools standardmäßig. Sie können dies feststellen, wenn beim Versuch, Rails `Gem::InstallError: The '[gem name]' native gem requires installed build tools.` zu installieren, ein Fehler `Gem::InstallError: The '[gem name]' native gem requires installed build tools.` Um dies zu beheben, müssen wir das Ruby Development Kit installieren.

- Laden Sie das DevKit herunter
- Führen Sie das Installationsprogramm aus.
- Wir müssen einen Ordner angeben, in dem das DevKit dauerhaft installiert wird. Ich empfehle, es im Stammverzeichnis Ihrer Festplatte unter `C:\RubyDevKit` . (Verwenden Sie keine Leerzeichen im Verzeichnisnamen.)

Jetzt müssen wir die DevKit-Tools für Ruby verfügbar machen.

- Wechseln Sie an Ihrer Eingabeaufforderung in das DevKit-Verzeichnis. `cd C:\RubyDevKit` oder das Verzeichnis, in dem Sie es installiert haben.
- Wir müssen ein Ruby-Skript ausführen, um das DevKit-Setup zu initialisieren. `ruby dk.rb init` . Jetzt sagen wir das gleiche Skript, um das DevKit unserer Ruby-Installation hinzuzufügen. `ruby dk.rb install` .

Das DevKit sollte jetzt für Ihre Ruby-Tools zur Installation neuer Bibliotheken verfügbar sein.

Schritt 3: ***Schienen***

Jetzt können wir Rails installieren. Rails ist ein Rubin-Juwel. Geben Sie in Ihre

Eingabeaufforderung Folgendes ein:

```
gem install rails
```

Wenn Sie die Eingabetaste drücken, lädt das `gem` Programm diese Version des Rails-Gem und alle anderen Edelsteine herunter, auf die Rails angewiesen ist.

Schritt 4: ***Node.js***

Für einige Bibliotheken, von denen Rails abhängig ist, muss eine JavaScript-Laufzeitumgebung installiert sein. Installieren wir Node.js, damit diese Bibliotheken ordnungsgemäß funktionieren.

- Laden Sie das Installationsprogramm für Node.js **hier** herunter.
- Wenn der Download abgeschlossen ist, besuchen Sie den Download-Ordner und führen Sie das Installationsprogramm `node-v4.4.7.pkg` .
- Lesen Sie die vollständige Lizenzvereinbarung, akzeptieren Sie die Bedingungen und klicken Sie im Rest des Assistenten auf Weiter, wobei die Standardeinstellungen beibehalten werden.
- Es erscheint ein Fenster, in dem Sie gefragt werden, ob die App Änderungen an Ihrem Computer vornehmen soll. Klicken Sie auf "Ja".
- Nach Abschluss der Installation müssen Sie Ihren Computer neu starten, damit Rails auf Node.js zugreifen kann.

Vergessen Sie nicht, nach dem Neustart des Computers das Windows-Menü aufzurufen, klicken Sie auf "Alle Programme", blättern Sie zu Ruby und klicken Sie auf "Eingabeaufforderung mit Ruby starten".

Erste Schritte mit Ruby on Rails online lesen: https://riptutorial.com/de/ruby-on-rails/topic/225/erste-schritte-mit-ruby-on-rails

Kapitel 2: ActionCable

Bemerkungen

ActionCable war für Rails 4.x verfügbar und wurde in Rails 5 zusammengefasst. Es ermöglicht die einfache Verwendung von Websockets für die Echtzeitkommunikation zwischen Server und Client.

Examples

[Basic] Serverseite

```
# 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
```

[Basic] Client-Seite (Coffeescript)

app / assets / 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 / assets / javascripts / application.js # wird normalerweise wie folgt generiert

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

app / assets / javascripts / cable.js # wird normalerweise so generiert

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

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

  App.cable = ActionCable.createConsumer();

}).call(this);
```

Benutzerauthentifizierung

```
# 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
```

ActionCable online lesen: https://riptutorial.com/de/ruby-on-rails/topic/1498/actioncable

Kapitel 3: ActionController

Einführung

Action Controller ist das C in MVC. Nachdem der Router festgelegt hat, welcher Controller für eine Anfrage verwendet werden soll, ist der Controller dafür verantwortlich, die Anfrage zu verstehen und die Ausgabe zu erzeugen.

Der Controller empfängt die Anforderung, holt oder speichert Daten von einem Modell und verwendet eine Ansicht, um eine Ausgabe zu erstellen. Ein Controller kann als Zwischenhändler zwischen Modellen und Ansichten betrachtet werden. Es macht die Modelldaten für die Ansicht verfügbar, sodass sie dem Benutzer angezeigt werden kann, und es speichert oder aktualisiert Benutzerdaten im Modell.

Examples

Ausgabe von JSON anstelle von 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
```

Zusätzlich benötigen Sie die Route:

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

Dies wird auf zwei verschiedene Arten auf Anfragen an `/users` reagieren:

- Wenn Sie `/users` oder `/users.html` , wird eine HTML-Seite mit dem Inhalt `Hello World /users.html`
- Wenn Sie `/users.json` besuchen, wird ein JSON-Objekt `/users.json` , das `/users.json` enthält:

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

Wenn Sie sicherstellen möchten, dass Ihre Route nur auf JSON-Anforderungen antwortet, können Sie `format.html { render inline: "Hello World" }` **weglassen** .

Steuerungen (Basic)

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

Dies ist ein Basiscontroller mit dem Hinzufügen der folgenden Route (in routes.rb):

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

Zeigt die `Hello World` Nachricht auf einer Webseite an, wenn Sie auf die URL `/users` zugreifen

Parameter

Controller haben Zugriff auf HTTP-Parameter (Sie kennen sie vielleicht als `?name=foo` in URLs, aber Ruby on Rails verarbeiten auch unterschiedliche Formate!) Und geben darauf basierend unterschiedliche Antworten aus. Es gibt keine Möglichkeit, zwischen GET- und POST-Parametern zu unterscheiden, aber Sie sollten dies auf keinen Fall tun.

```
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
```

Wie üblich unsere Route:

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

Greifen Sie auf die URL `/users?name=john` und die Ausgabe wird `Hello John` , der Zugriff auf `/users?name=whatever` und die Ausgabe wird `Hello someone`

Filterparameter (Basic)

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html do
        render html: "Hello #{ user_params[:name] } user_params[:sentence]"
```

```
      end
    end
  end

  private

  def user_params
    if params[:name] == "john"
      params.permit(:name, :sentence)
    else
      params.permit(:name)
    end
  end
end
```

Sie können einige Parameter zulassen (oder ablehnen), so dass nur das, was Sie möchten *, durchgeht* und Sie keine bösen Überraschungen wie Benutzereinstellungen haben, die nicht geändert werden sollen.

Besuch `/users?name=john&sentence=developer` zeigt `Hello john developer` , aber Besuch `/users?name=smith&sentence=spy` zeigt nur `Hello smith` , weil `:sentence` nur zulässig ist, wenn Sie als `john` zugreifen

Umleitung

Annahme der Route:

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

Sie können zu einer anderen URL umleiten mit:

```
class UsersController
  def index
    redirect_to "http://stackoverflow.com/"
  end
end
```

Sie können zur vorherigen Seite zurückkehren, die der Benutzer besucht hat:

```
redirect_to :back
```

Beachten Sie, dass in *Rails 5* die Syntax für das Zurückleiten anders ist:

```
redirect_back fallback_location: "http://stackoverflow.com/"
```

Dadurch wird versucht, auf die vorherige Seite umzuleiten, und falls dies nicht möglich ist (der Browser blockiert den HTTP_REFERRER-Header), wird es weitergeleitet an `:fallback_location`

Ansichten verwenden

Annahme der Route:

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

Und der Controller:

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

Die Ansichtsanwendung `app/users/index.html.erb` wird gerendert. Wenn die Ansicht ist:

```
Hello <strong>World</strong>
```

Die Ausgabe wird eine Webseite mit dem Text "Hello **World** " sein.

Wenn Sie eine andere Ansicht rendern möchten, können Sie Folgendes verwenden:

```
render "pages/home"
```

`app/views/pages/home.html.erb` wird die Datei `app/views/pages/home.html.erb` verwendet.

Sie können Variablen mithilfe von Controller-Instanzvariablen an Ansichten übergeben:

```
class UsersController < ApplicationController
  def index
    @name = "john"

    respond_to do |format|
      format.html { render }
    end
  end
end
```

Und in der Datei `app/views/users/index.html.erb` Sie `@name` :

```
Hello <strong><%= @name %></strong>
```

Und die Ausgabe wird sein: "Hallo **John** "

Ein wichtiger Hinweis rund um die Render-Syntax. Sie können die `render` Syntax vollständig weglassen. So:

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

Kann stattdessen geschrieben werden als:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html
    end
  end
end
```

Rails ist intelligent genug, um herauszufinden, dass es die Datei `app/views/users/index.html.erb` .

404 wenn Datensatz nicht gefunden wurde

Fehler beim Aufheben des Datensatzes nicht gefunden, anstatt eine Ausnahme oder eine weiße Seite anzuzeigen:

```
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
```

Grundlegender REST-Controller

```
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

  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
```

Fehlerseiten für Ausnahmen anzeigen

Wenn Sie Ihren Benutzern sinnvolle Fehler anzeigen möchten, anstatt nur "Entschuldigung, es ist ein Fehler aufgetreten", bietet Rails ein nützliches Hilfsprogramm.

Öffnen Sie die Datei `app/controllers/application_controller.rb` und Sie sollten etwa Folgendes finden:

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
end
```

Wir können jetzt ein `rescue_from` hinzufügen, um bestimmte Fehler zu beheben:

```
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
```

Es wird empfohlen, nicht vor `Exception` oder `StandardError` zu retten, da Rails im Fehlerfall keine hilfreichen Seiten anzeigen kann.

Filter

Filter sind Methoden, die "vor", "nach" oder "um" eine Controller-Aktion ausgeführt werden. Sie werden vererbt. Wenn Sie also in Ihrem `ApplicationController` Einstellung festlegen, werden sie für jede Anforderung ausgeführt, die Ihre Anwendung empfängt.

Vor dem Filter

Bevor Filter vor der Controller-Aktion ausgeführt werden, kann die Anforderung (und / oder die Weiterleitung) angehalten werden. Häufig wird überprüft, ob ein Benutzer angemeldet ist:

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
    redirect_to some_path unless user_signed_in?
  end
end
```

Vor dem Ausführen von Filtern für Anforderungen, bevor die Anforderung die Aktion des Controllers erhält. Es kann eine Antwort selbst zurückgeben und die Aktion vollständig umgehen.

Eine weitere häufige Anwendung von before-Filtern ist das Überprüfen der Authentifizierung eines Benutzers, bevor ihm der Zugriff auf die Aktion gewährt wird, die für die Bearbeitung seiner Anfrage festgelegt wurde. Ich habe auch gesehen, wie sie zum Laden einer Ressource aus der Datenbank, zum Überprüfen von Berechtigungen für eine Ressource oder zum Verwalten von Weiterleitungen unter anderen Umständen verwendet wurden.

Nach dem Filter

Nach Filtern ähneln die Filter "vor", aber wenn sie nach dem Aktionslauf ausgeführt werden, haben sie Zugriff auf das Antwortobjekt, das gesendet werden soll. Also kurz nachdem Filter ausgeführt wurden, nachdem die Aktion abgeschlossen ist. Es kann die Antwort ändern. Wenn etwas in einem Nachfilter ausgeführt wird, kann dies meistens in der Aktion selbst ausgeführt werden. Wenn jedoch nach dem Ausführen einer Reihe von Aktionen eine Logik ausgeführt werden muss, ist ein Nachfilter ein guter Ort es.

Im Allgemeinen habe ich nach und um Filter gesehen, die für die Protokollierung verwendet werden.

Um den Filter herum

Um Filter herum kann es vor und nach der ausgeführten Aktion Logik geben. Es gibt einfach die Handlung an dem Ort, wo es notwendig ist. Beachten Sie, dass es der Aktion nicht nachgeben

muss und möglicherweise wie ein vorheriger Filter ausgeführt wird.

Around-Filter sind für die Ausführung ihrer zugehörigen Aktionen verantwortlich, ähnlich wie Rack-Middlewares.

Callbacks umschließen die Ausführung von Aktionen. Sie können einen Rückruf in zwei verschiedenen Stilen schreiben. Im ersten Fall ist der Rückruf ein einzelner Code. Dieser Code wird aufgerufen, bevor die Aktion ausgeführt wird. Wenn der Rückrufcode die Fließmenge aufruft, wird die Aktion ausgeführt. Wenn die Aktion abgeschlossen ist, wird der Callback-Code weiter ausgeführt. Daher ist der Code vor der Rendite wie ein Callback vor der Aktion und der Code nach der Rendite ist der Callback nach der Aktion. Wenn der Rückrufcode niemals ertrag aufruft. Die Aktion wird nicht ausgeführt. Dies ist dasselbe, als wenn ein Callback vor der Aktion false zurückgibt.

Hier ist ein Beispiel für den Around-Filter:

```
around_filter :catch_exceptions

private
  def catch_exceptions
    begin
      yield
    rescue Exception => e
      logger.debug "Caught exception! #{e.message}"
    end
  end
```

Dies fängt die Ausnahme einer Aktion ab und fügt die Nachricht in Ihr Protokoll ein. Sie können Filter für die Ausnahmebehandlung, das Einrichten und Abbauen sowie eine Vielzahl anderer Fälle verwenden.

Nur und außer

Alle Filter können auf bestimmte Aktionen angewendet werden `:only` und `: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
```

Filter überspringen

Alle Filter (auch geerbte Filter) können für bestimmte Aktionen auch übersprungen werden:

```
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
```

Da sie geerbt sind, können Filter auch in einem übergeordneten Controller für `namespace` definiert werden. Angenommen, Sie haben einen `admin` Namespace, und Sie möchten natürlich nur Admin-Benutzer darauf zugreifen können. Sie könnten so etwas tun:

```
# 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
```

`before_filter` Sie, dass Sie in **Rails 4.x** `before_filter` zusammen mit `before_action` , aber `before_filter` ist derzeit in **Rails 5.0.0** veraltet und wird in **5.1 entfernt** .

Controller generieren

Rails bietet viele Generatoren, natürlich auch für Controller.

Sie können einen neuen Controller generieren, indem Sie diesen Befehl in Ihrem App-Ordner ausführen

```
rails generate controller NAME [action action] [options]
```

Hinweis: Sie können auch `rails g` Alias verwenden, um das `rails generate` aufzurufen

Um beispielsweise einen Controller für ein `Product` zu generieren, `#index` `#show` Aktionen `#index` und

`#show` ausführen

```
rails generate controller products index show
```

Dadurch wird der Controller in `app/controllers/products_controller.rb` mit den von Ihnen angegebenen Aktionen erstellt

```
class ProductsController < ApplicationController
  def index
  end

  def show
  end
end
```

Es wird auch ein erstellen `products` Ordner in `app/views/` , die beiden Vorlagen für Ihre Controller Aktionen enthält (dh `index.html.erb` und `show.html.erb` , *beachten* Sie, *dass die Erweiterung entsprechend Ihre Template - Engine kann variieren, so dass* , *wenn Sie Verwenden Sie* `slim` *, zum Beispiel erzeugt der Generator* `index.html.slim` *und* `show.html.slim`

Wenn Sie Aktionen angegeben haben, werden diese auch Ihrer `routes` hinzugefügt

```
# config/routes.rb
get 'products/show'
get 'products/index'
```

Rails erstellt für Sie eine Hilfedatei in `app/helpers/products_helper.rb` sowie die Bestandsdateien in `app/assets/javascripts/products.js` und `app/assets/stylesheets/products.css` . In `Gemfile` auf Ansichten ändert der Generator dieses Verhalten entsprechend den Angaben in Ihrer `Gemfile` : Wenn Sie in Ihrer Anwendung `Coffeescript` und `Sass` verwenden, `Coffeescript` der Controller-Generator stattdessen `products.coffee` und `products.sass` .

Zu guter Letzt erzeugt Rails auch Testdateien für Ihren Controller, Ihren Helfer und Ihre Ansichten.

Wenn Sie nicht möchten, dass eines davon erstellt wird, können Sie Rails mitteilen, es zu überspringen

`--no-` oder `--skip` wie `--skip` :

```
rails generate controller products index show --no-assets --no-helper
```

Und der Generator überspringt sowohl `assets` als auch `helper`

Wenn Sie einen Controller für einen bestimmten **`namespace`** erstellen müssen, fügen Sie ihn vor `NAME` :

```
rails generate controller admin/products
```

Dadurch wird Ihre Steuerung in `app/controllers/admin/products_controller.rb`

Rails kann auch einen kompletten RESTful-Controller für Sie generieren:

```
rails generate scaffold_controller MODEL_NAME # available from Rails 4
rails generate scaffold_controller Product
```

ActiveRecord :: RecordNotFound mit redirect_to retten

Sie können eine RecordNotFound-Ausnahme mit einer Weiterleitung retten, anstatt eine Fehlerseite anzuzeigen:

```
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
```

ActionController online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2838/actioncontroller

Kapitel 4: ActionMailer

Einführung

Mit Action Mailer können Sie E-Mails aus Ihrer Anwendung mithilfe von Mailer-Klassen und -Ansichten senden. Mailer arbeiten sehr ähnlich wie Controller. Sie erben von ActionMailer :: Base und leben in App / Mailern. Sie haben dazugehörige Ansichten, die in App / Ansichten angezeigt werden.

Bemerkungen

Es wird empfohlen, das Versenden von E-Mails asynchron zu verarbeiten, um Ihren Webserver nicht zu blockieren. Dies kann durch verschiedene Dienste wie `delayed_job` .

Examples

Basic Mailer

In diesem Beispiel werden vier verschiedene Dateien verwendet:

- Das Benutzermodell
- Der User Mailer
- Die HTML-Vorlage für die E-Mail
- Die Nur-Text-Vorlage für die E-Mail

In diesem Fall ruft das Benutzermodell die `approved` Methode im Mailer auf und übergibt die genehmigte `post` (die `approved` Methode im Modell kann durch einen Rückruf, von einer Controller-Methode usw. aufgerufen werden). Anschließend generiert der Mailer die E-Mail entweder aus der HTML- oder der Nur-Text-Vorlage anhand der Informationen aus dem übergebenen `post` (z. B. dem Titel). Standardmäßig verwendet der Mailer die Vorlage mit demselben Namen wie die Methode im Mailer (weshalb sowohl die Mailer-Methode als auch die Vorlage den Namen "Genehmigt" haben).

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
```

user.rb

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

approved.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>
```

approved.text.erb

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

Neuen Mailer generieren

Geben Sie den folgenden Befehl ein, um einen neuen Mailer zu erstellen

```
rails generate mailer PostMailer
```

Dadurch wird eine leere Vorlagendatei in `app/mailers/post_mailer.rb` Namen *PostMailer generiert*

```
class PostMailer < ApplicationMailer
end
```

Es werden auch zwei Layoutdateien für die E-Mail-Ansicht erstellt, eine für das HTML-Format und eine für das Textformat.

Wenn Sie den Generator nicht verwenden möchten, können Sie eigene Mailer erstellen. `ActionMailer::Base` Sie sicher, dass sie von `ActionMailer::Base` erben

Anhänge hinzufügen

`ActionMailer` erlaubt auch das Anhängen von Dateien.

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

Anhänge werden standardmäßig mit `Base64` codiert. Um dies zu ändern, können Sie der Attachments-Methode einen Hash hinzufügen.

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

Sie können auch Inline-Anhänge hinzufügen

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

ActionMailer-Rückrufe

ActionMailer unterstützt drei Rückrufe

- before_action
- after_action
- around_action

Stellen Sie diese in Ihrer Mailer-Klasse bereit

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

Erstellen Sie dann diese Methoden unter dem `private` Schlüsselwort

```
private
  def set_delivery_options
  end

  def prevent_delivery_to_guests
  end

  def set_business_headers
  end
end
```

Generieren Sie einen geplanten Newsletter

Erstellen Sie das **Newsletter-** Modell:

```
  rails g model Newsletter name:string email:string

  subl app/models/newsletter.rb

  validates :name, presence: true
  validates :email, presence: true
```

Erstellen Sie den **Newsletter-** Controller:

```
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
```

Ändern Sie danach die Ansicht " **create.html.erb"** in den Namen " **Nex"** . Wir konvertieren diese Datei in eine **Teilansicht** , die in der **Fußzeile** gespeichert wird. Der Name **lautet _form.html.erb** .

Namensdatei ändern von:	Zu:
app / views / **newsletters / create.html.erb**	app / views / **newsletters / _form.html.erb**

Danach legen Sie die Routen fest:

```
subl app/config/routes.rb

resources :newsletters
```

Später müssen wir das Formular festlegen, das zum Speichern jeder Mail verwendet wird:

```
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 %>
```

Und danach in die Fußzeile einfügen:

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

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

Installieren Sie nun - **letter_opener** - to, um die E-Mail im Standardbrowser als Vorschau **anzuzeigen** , anstatt sie zu senden. Das bedeutet, dass Sie die E-Mail-Zustellung nicht in Ihrer Entwicklungsumgebung einrichten müssen, und Sie müssen sich keine Sorgen mehr machen, dass Sie versehentlich eine Test-E-Mail an die Adresse eines anderen Benutzers gesendet haben.

Fügen Sie zuerst den Edelstein Ihrer Entwicklungsumgebung hinzu und führen Sie den Befehl bundle aus, um ihn zu installieren.

```
  subl your_project/Gemfile

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

Legen Sie dann die Bereitstellungsmethode in der Entwicklungsumgebung fest:

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

  config.action_mailer.delivery_method = :letter_opener
```

Erstellen Sie nun eine **Mailer-Struktur** , um die gesamten **Mailer** zu verwalten, an denen wir arbeiten werden. Im Terminal

```
  rails generate mailer UserMailer newsletter_mailer
```

Und im **UserMailer** müssen wir eine Methode namens **Newsletter Mailer** erstellen, die erstellt wird, um den neuesten Blogeintrag zu enthalten und mit einer Rake-Aktion ausgelöst zu werden. Wir gehen davon aus, dass Sie zuvor eine Blogstruktur erstellt haben.

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

class UserMailer 'your_gmail_account@gmail.com'


  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
```

Danach erstellen Sie die **Mailer-Vorlage** :

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

<p> Dear Followers: </p>
<p> Those are the lastest 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>
```

Da wir die E-Mail als separaten Vorgang senden möchten, erstellen wir eine Rake-Aufgabe, um die E-Mail auszulösen. Fügen Sie dem lib / task-Verzeichnis Ihrer Rails-Anwendung eine neue Datei mit dem Namen email_tasks.rake hinzu:

```
touch lib/taks/email_tasks.rake

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

Die send_digest_email:: -Umgebung bedeutet, die Rails-Umgebung vor dem Ausführen der Task zu laden, sodass Sie auf die Anwendungsklassen (wie UserMailer) innerhalb der Task zugreifen können.

Wenn Sie nun den Befehl rake -T ausführen, wird die neu erstellte Rake-Task aufgelistet. Testen Sie alles, indem Sie die Aufgabe ausführen und prüfen, ob die E-Mail gesendet wird oder nicht.

Führen Sie den Befehl rake aus, um zu testen, ob die Mailer-Methode funktioniert:

```
  rake weekly_newsletter_email
```

Zu diesem Zeitpunkt haben wir eine Arbeitsrechenaufgabe, die mit **Crontab** geplant werden kann. Also installieren wir den **Whenever Gem,** der eine klare Syntax für das Schreiben und Bereitstellen von Cron-Jobs bereitstellt.

```
subl your_project/Gemfile

  gem 'whenever', require: false
```

Führen Sie danach den nächsten Befehl aus, um eine ursprüngliche config / schedule.rb-Datei für Sie zu erstellen (sofern der config-Ordner bereits in Ihrem Projekt vorhanden ist).

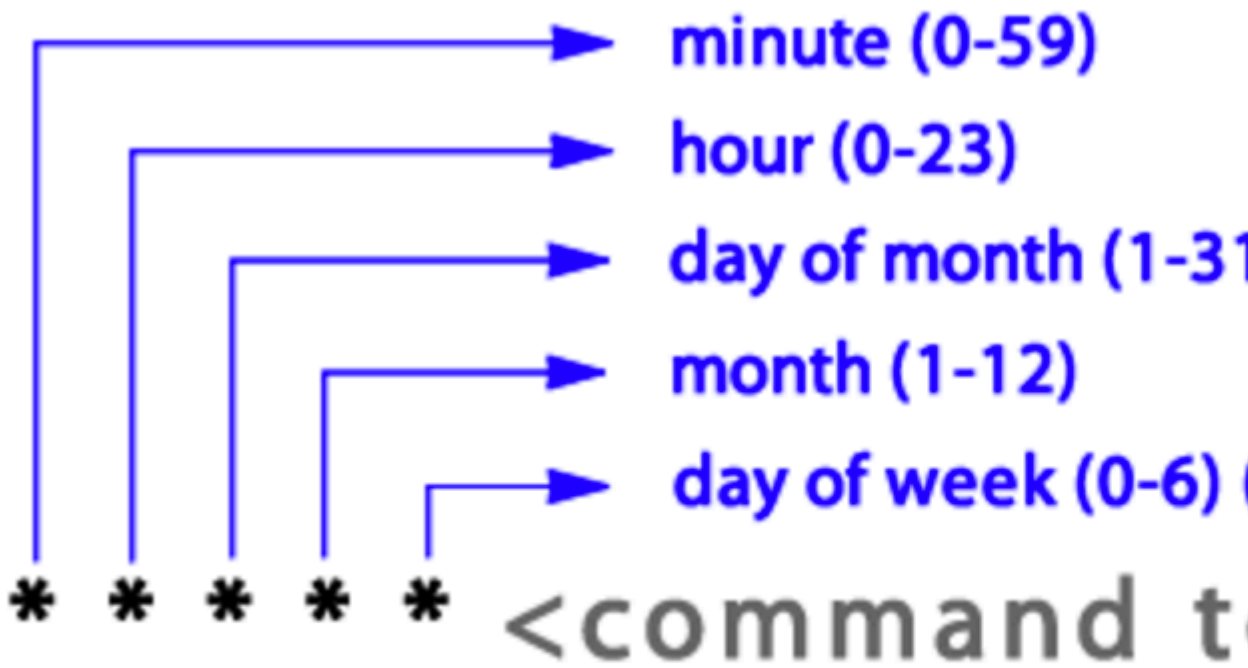
```
  wheneverize .

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

Jetzt müssen wir in der Zeitplandatei unseren **CRON-Job** erstellen und die Mailer-Methode aufrufen, um den CRON-Job zu bestimmen, um einige Aufgaben ohne Hilfe und in einem bestimmten Zeitraum auszuführen. Sie können verschiedene Syntaxtypen verwenden, wie auf diesem Link erläutert.

```
subl your_project/config/schedule.rb

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


minute (0-59)
hour (0-23)
day of month (1-31
month (1-12)
day of week (0-6)
* * * * *
<command t

```
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
```

erfolgreich erstellt wurde, können wir den nächsten Befehl verwenden, um seit dem Terminal unseren geplanten Job in CRON SYNTAX zu lesen:

```
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'
```

Um den Test in der Entwicklungsumgebung auszuführen, ist es ratsam, die nächste Zeile in der Datei " **application.rb** " zu setzen, um der Anwendung mitzuteilen, wo welche Modelle verwendet werden.

```
  subl your_project/config/application.rb

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

Damit **Capistrano V3** den neuen **Cron-Job** im Server und den Auslöser speichern kann, der die Ausführung dieser Aufgabe auslöst, müssen wir die nächste Anforderung hinzufügen:

```
  subl your_project/Capfile

  require 'whenever/capistrano'
```

Und in die **deploy** - Datei die Kennung , die **Cron** - **Job** über die **Umwelt** und den Namen der **Anwendung** verwenden.

```
subl your_project/config/deploy.rb

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

Führen Sie nach dem Speichern der Änderungen für jede Datei den Befehl capistrano deploy aus:

```
  cap production deploy
```

Und jetzt wurde Ihr Job erstellt und kalenderisiert, um die Mailer-Methode auszuführen, die ich will und in dem Zeitraum, in dem wir diese Dateien festlegen.

ActionMailer-Interceptor

Action Mailer stellt Hooks für die Interceptor-Methoden bereit. Auf diese Weise können Sie Klassen registrieren, die während des Lebenszyklus der Postzustellung aufgerufen werden.

Eine Interceptor-Klasse muss die: delivering_email (message) -Methode implementieren, die vor dem Senden der E-Mail aufgerufen wird, sodass Sie Änderungen an der E-Mail vornehmen können, bevor sie die Zustellungsagenten treffen. Ihre Klasse sollte alle erforderlichen Änderungen direkt an der in Mail :: Message übergebenen Instanz vornehmen.

Für Entwickler kann es nützlich sein, E-Mails an sich selbst und nicht an echte Benutzer zu senden.

Beispiel für die Registrierung eines Actionmailer-Interceptors:

```
# 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
```

ActionMailer online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2481/actionmailer

Kapitel 5: ActiveJob

Einführung

Active Job ist ein Rahmen zum Deklarieren von Jobs und deren Ausführung auf einer Vielzahl von Backends für Warteschlangen. Diese Jobs können alles umfassen, von regelmäßig geplanten Aufräumarbeiten über Rechnungsgebühren bis hin zu Mailings. Alles, was in kleine Arbeitseinheiten zerhackt werden kann und parallel läuft, wirklich.

Examples

Erstellen Sie den Job

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

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

Den Job in die Warteschlange stellen

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

ActiveJob online lesen: https://riptutorial.com/de/ruby-on-rails/topic/8996/activejob

Kapitel 6: ActiveModel

Bemerkungen

ActiveModel wurde erstellt, um das Modellverhalten von ActiveRecord in eine separate Betrachtung zu extrahieren. Dies ermöglicht uns die Verwendung des ActiveModel-Verhaltens in jedem Objekt, nicht nur in ActiveRecord-Modellen.

ActiveRecord-Objekte enthalten dieses Verhalten standardmäßig.

Examples

ActiveModel :: Validierungen verwenden

Sie können jedes Objekt überprüfen, auch reinen Rubin.

```
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
```

ActiveModel online lesen: https://riptutorial.com/de/ruby-on-rails/topic/1773/activemodel

Kapitel 7: ActiveRecord-Abfrage-Schnittstelle

Einführung

ActiveRecord ist das M in MVC, dh die Schicht des Systems, die für die Darstellung von Geschäftsdaten und -logik zuständig ist. Die Technik , die die reichen Objekte einer Anwendung Tabellen in einer relationalen Datenbank - Management - System verbindet , ist **O** bject **R** elational **M** apper **(ORM).**

ActiveRecord führt für Sie Abfragen in der Datenbank durch und ist mit den meisten Datenbanksystemen kompatibel. Unabhängig davon, welches Datenbanksystem Sie verwenden, das Format der ActiveRecord-Methode ist immer dasselbe.

Examples

.woher

Die `where` Methode ist für jedes `ActiveRecord` Modell verfügbar und ermöglicht das Abfragen der Datenbank nach einem Datensatz, der den angegebenen Kriterien entspricht.

Die `where` Methode akzeptiert einen Hash, bei dem die Schlüssel den Spaltennamen in der Tabelle entsprechen, die das Modell darstellt.

Als einfaches Beispiel verwenden wir folgendes Modell:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end
```

Um alle Personen mit dem Vornamen von `Sven` :

```
people = Person.where(first_name: 'Sven')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven'"
```

Um alle Personen mit dem Vornamen von `Sven` und dem Nachnamen von `Schrodinger` :

```
people = Person.where(first_name: 'Sven', last_name: 'Schrodinger')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven' AND last_name='Schrodinger'"
```

Im obigen Beispiel zeigt die SQL-Ausgabe, dass Datensätze nur zurückgegeben werden, wenn sowohl der `first_name` als auch der `last_name` übereinstimmen.

Abfrage mit ODER-Bedingung

So suchen Sie Datensätze mit `first_name == 'Bruce'` ODER `last_name == 'Wayne'`

```
User.where('first_name = ? or last_name = ?', 'Bruce', 'Wayne')
# SELECT "users".* FROM "users" WHERE (first_name = 'Bruce' or last_name = 'Wayne')
```

.wo mit einem Array

Die `where` Methode für ein beliebiges ActiveRecord-Modell kann verwendet werden, um SQL der Form `WHERE column_name IN (a, b, c, ...)` zu generieren. Dies wird erreicht, indem ein Array als Argument übergeben wird.

Als einfaches Beispiel verwenden wir folgendes Modell:

```
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')"
```

Wenn das Array eine `nil` enthält, wird das SQL geändert, um zu überprüfen, ob die Spalte `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"
```

Bereiche

Bereiche dienen als vordefinierte Filter für `ActiveRecord` Modelle.

Ein Gültigkeitsbereich wird mithilfe der `scope` definiert.

Als einfaches Beispiel verwenden wir folgendes Modell:

```
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
```

Bereiche können direkt aus der Modellklasse abgerufen werden:

```
minors = Person.minors
```

Bereiche können verkettet werden:

```
peters_children = Person.minors.with_last_name('Peters')
```

Die `where` Methode und andere Abfragetypmethoden können auch verkettet werden:

```
mary_smith = Person.with_last_name('Smith').where(first_name: 'Mary')
```

Hinter den Kulissen sind Geltungsbereiche einfach syntaktischer Zucker für eine Standardklassenmethode. Zum Beispiel sind diese Methoden funktional identisch:

```
scope :with_last_name, ->(name) { where(name: name) }

# This ^ is the same as this:

def self.with_last_name(name)
  where(name: name)
end
```

Standardumfang

in Ihrem Modell, um einen Standardbereich für alle Vorgänge am Modell festzulegen.

> Es gibt einen bemerkenswerten Unterschied zwischen der `scope` und einer Klassenmethode: `scope` -definierte `scope` geben *immer* ActiveRecord `ActiveRecord::Relation` , selbst wenn die Logik in nil zurückgibt. Klassenmethoden haben jedoch kein solches Sicherheitsnetz und können die Kettenfähigkeit brechen, wenn sie etwas anderes zurückgeben.

wo nicht

`where` Klauseln können mit der `where.not` Syntax negiert werden:

```
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')
```

Unterstützt von ActiveRecord 4.0 und höher.

Bestellung

Sie können mit **Activeabfrageergebnisse** bestellen **`.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">]
```

Wenn nicht angegeben, erfolgt die Bestellung in aufsteigender Reihenfolge. Sie können es angeben, indem Sie Folgendes tun:

```
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` akzeptiert auch eine Zeichenfolge, also können Sie dies auch tun

```
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">]
```

Da es sich bei der Zeichenfolge um Raw-SQL handelt, können Sie auch eine Tabelle und nicht nur ein Attribut angeben. Angenommen , Sie bestellen möchten `users` entsprechend ihrer `role` Namen, können Sie dies tun:

```
Class User < ActiveRecord::Base
    belongs_to :role
end

Class Role < ActiveRecord::Base
  has_many :users
end

User.includes(:role).order("roles.name ASC")
```

Der `order` kann auch einen Arel-Knoten akzeptieren:

```
User.includes(:role).order(User.arel_table[:name].asc)
```

ActiveRecord Bang (!) -Methoden

Wenn Sie eine **ActiveRecord-** Methode benötigen, um im **Fehlerfall** eine Ausnahme anstelle eines `false` Werts auszulösen, können Sie hinzufügen `!` zu ihnen. Dies ist sehr wichtig. Da einige Ausnahmen / Ausfälle schwer zu fassen sind, wenn Sie sie nicht verwenden! auf sie. Ich habe empfohlen, dies in Ihrem Entwicklungszyklus zu tun, um Ihren gesamten ActiveRecord-Code auf diese Weise zu schreiben, um Zeit und Ärger zu sparen.

```
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
```

Die **ActiveRecord-** Methoden, die einen *Knall* (`!`) Akzeptieren, sind:

- `.create!`
- `.take!`
- `.first!`

- `.last!`
- `.find_by!`
- `.find_or_create_by!`
- `#save!`
- `#update!`
- alle AR dynamischen Sucher

.find_by

Mit `find_by` können Sie Datensätze nach beliebigen Feldern in Ihrer Tabelle `find_by` .

Wenn Sie ein `User` mit einem `first_name` Attribut haben, können Sie `first_name` tun:

```
User.find_by(first_name: "John")
#=> #<User id: 2005, first_name: "John", last_name: "Smith">
```

`find_by` dass `find_by` standardmäßig keine Ausnahme `find_by` . Wenn das Ergebnis ein leerer Satz ist, wird `nil` anstelle von `find` .

Wenn die Ausnahme benötigt wird, kann `find_by!` das verursacht einen `ActiveRecord::RecordNotFound` Fehler wie `find` .

.alles löschen

Wenn Sie eine Menge von Datensätzen schnell löschen müssen, gibt **Active** **`.delete_all`** Methode. um direkt in einem Modell aufgerufen zu werden, um alle Datensätze in dieser Tabelle oder eine Auflistung zu löschen. **`.delete_all`** Sie jedoch, da **`.delete_all`** kein Objekt instanziiert und daher keinen Rückruf `before_*` (`before_*` und `after_destroy` nicht ausgelöst).

```
User.delete_all
#=> 39  <-- .delete_all return the number of rows deleted

User.where(name: "John").delete_all
```

Bei ActiveRecord wird die Groß- / Kleinschreibung nicht berücksichtigt

Wenn Sie ein ActiveRecord-Modell nach ähnlichen Werten suchen müssen, werden Sie möglicherweise versucht, `LIKE` oder `ILIKE` , dies ist jedoch zwischen Datenbank-Engines nicht übertragbar. In ähnlicher Weise kann der Rückgriff auf immer Downcasing oder Upcasing zu Performance-Problemen führen.

Sie können die zugrunde liegende Arel- `matches` Methode von ActiveRecord verwenden, um dies auf eine sichere Weise durchzuführen:

```
addresses = Address.arel_table
Address.where(addresses[:address].matches("%street%"))
```

Arel wendet das entsprechende LIKE- oder ILIKE-Konstrukt für das konfigurierte Datenbankmodul an.

Holen Sie sich den ersten und letzten Datensatz

Rails haben eine sehr einfache Möglichkeit, den `first` und `last` Datensatz aus der Datenbank abzurufen.

Um den `first` Datensatz aus der `users` , müssen Sie den folgenden Befehl eingeben:

```
User.first
```

Es wird folgende `sql` Abfrage generiert:

```
SELECT  `users`.* FROM `users`  ORDER BY `users`.`id` ASC LIMIT 1
```

Und wird folgende Aufzeichnung zurückgeben:

```
#<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 >
```

Um den `last` Datensatz aus der `users` , müssen Sie den folgenden Befehl eingeben:

```
User.last
```

Es wird folgende `sql` Abfrage generiert:

```
SELECT  `users`.* FROM `users`  ORDER BY `users`.`id` DESC LIMIT 1
```

Und wird folgende Aufzeichnung zurückgeben:

```
#<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 >
```

Durch Übergeben einer Ganzzahl an die **erste** und **letzte** Methode wird eine **LIMIT-** Abfrage erstellt und ein Array von Objekten zurückgegeben.

```
User.first(5)
```

Es wird folgende `sql` Abfrage generiert.

```
SELECT  "users".* FROM "users"  ORDER BY "users"."id" ASC LIMIT 5
```

Und

```
User.last(5)
```

Es wird folgende `sql` Abfrage generiert.

```
SELECT  "users".* FROM "users"  ORDER BY "users"."id" DESC LIMIT 5
```

.group und .count

Wir haben ein `Product` und möchten sie nach ihrer `category` gruppieren.

```
Product.select(:category).group(:category)
```

Dadurch wird die Datenbank wie folgt abgefragt:

```
SELECT "product"."category" FROM "product" GROUP BY "product"."category"
```

Stellen Sie sicher, dass das gruppierte Feld ebenfalls ausgewählt ist. Die Gruppierung eignet sich besonders zum Zählen des Vorkommens - in diesem Fall - von `categories` .

```
Product.select(:category).group(:category).count
```

Wie die Abfrage zeigt, wird die Datenbank zum Zählen verwendet, was wesentlich effizienter ist, als zuerst alle Datensätze abzurufen und im Code zu zählen:

```
SELECT COUNT("products"."category") AS count_categories, "products"."category" AS
products_category FROM "products" GROUP BY "products"."category"
```

.distinct (oder .uniq)

Wenn Sie Duplikate aus einem Ergebnis entfernen möchten, können Sie `.distinct()` :

```
Customers.select(:country).distinct
```

Dadurch wird die Datenbank wie folgt abgefragt:

```
SELECT DISTINCT "customers"."country" FROM "customers"
```

`.uniq()` hat den gleichen Effekt. Mit Rails 5.0 wurde es veraltet und mit Version 5.1 von Rails entfernt. Der Grund ist, dass das Wort " `unique` nicht die gleiche Bedeutung wie "different" hat und es kann irreführend sein. Weiterhin `distinct` ist näher an der SQL - Syntax.

Schließt sich an

`joins()` können Sie Tabellen mit Ihrem aktuellen Modell verknüpfen. Für ex.

```
User.joins(:posts)
```

erzeugt die folgende SQL-Abfrage:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id""
```

Nachdem Sie den Tisch verbunden haben, können Sie darauf zugreifen:

```
User.joins(:posts).where(posts: { title: "Hello world" })
```

Achten Sie auf die Pluralform. Wenn Ihre Beziehung `:has_many` , sollte das Argument `joins()` pluralisiert werden. Ansonsten verwenden Sie Singular.

Verschachtelte `joins` :

```
User.joins(posts: :images).where(images: { caption: 'First post' })
```

was wird produzieren:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id" INNER
JOIN "images" ON "images"."post_id" = "images"."id""
```

Enthält

ActiveRecord with `includes` stellt sicher, dass alle angegebenen Zuordnungen mit der minimal möglichen Anzahl von Abfragen geladen werden. Wenn Sie also eine Tabelle nach Daten mit einer zugeordneten Tabelle abfragen, werden beide Tabellen in den Speicher geladen.

```
@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 } )` lädt nur **Autoren** mit Bedingungen in den Speicher, **ohne Bücher zu laden** . Verwenden Sie `joins` wenn keine zusätzlichen Informationen zu verschachtelten Zuordnungen erforderlich sind.

```
@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
```

Limit und Offset

Sie können mit `limit` die Anzahl der abzurufenden Datensätze angeben und mit `offset` die Anzahl der zu überspringenden Datensätze angeben, bevor Sie die Datensätze zurückgeben.

Zum Beispiel

```
User.limit(3) #returns first three records
```

Es wird folgende SQL-Abfrage generiert.

```
"SELECT  `users`.* FROM `users` LIMIT 3"
```

Da der Offset in der obigen Abfrage nicht erwähnt wird, werden die ersten drei Datensätze zurückgegeben.

```
User.limit(5).offset(30) #returns 5 records starting from 31th i.e from 31 to 35
```

Es wird folgende SQL-Abfrage generiert.

```
"SELECT  `users`.* FROM `users` LIMIT 5 OFFSET 30"
```

ActiveRecord-Abfrage-Schnittstelle online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2154/activerecord-abfrage-schnittstelle

Kapitel 8: ActiveRecord-Migrationen

Parameter

Säulentyp	Beschreibung
`:primary_key`	Primärschlüssel
`:string`	Kürzerer String-Datentyp. Ermöglicht die `limit` für die maximale Anzahl von Zeichen.
`:text`	Längere Textmenge Erlaubt die `limit` für die maximale Anzahl von Bytes.
`:integer`	Ganze Zahl. Erlaubt die `limit` für die maximale Anzahl von Bytes.
`:bigint`	Größere ganze Zahl
`:float`	Schweben
`:decimal`	Dezimalzahl mit variabler Genauigkeit. Ermöglicht `precision` und `scale` .
`:numeric`	Ermöglicht `precision` und `scale` .
`:datetime`	DateTime-Objekt für Datum und Uhrzeit.
`:time`	Zeitobjekt für Zeiten.
`:date`	Datumsobjekt für Datumsangaben
`:binary`	Binärdaten. Erlaubt die `limit` für die maximale Anzahl von Bytes.
`:boolean`	Boolean

Bemerkungen

- Die meisten Migrationsdateien befinden sich im Verzeichnis `db/migrate/` . Sie werden durch einen UTC-Zeitstempel am Anfang ihres Dateinamens identifiziert: `YYYYMMDDHHMMSS_create_products.rb` .
- Der Befehl zum `rails generate` kann auf `rails g` gekürzt werden.
- Wenn ein `:type` nicht an ein Feld übergeben wird, wird standardmäßig eine Zeichenfolge verwendet.

Examples

Führen Sie eine bestimmte Migration aus

Verwenden Sie `db:migrate:up` oder `db:migrate:down` um eine bestimmte Migration nach oben oder unten auszuführen.

Eine bestimmte Migration durchführen:

5,0

```
rake db:migrate:up VERSION=20090408054555
```

5,0

```
rails db:migrate:up VERSION=20090408054555
```

Hinunter eine bestimmte Migration:

5,0

```
rake db:migrate:down VERSION=20090408054555
```

5,0

```
rails db:migrate:down VERSION=20090408054555
```

Die Versionsnummer in den obigen Befehlen ist das numerische Präfix im Dateinamen der Migration. Um beispielsweise zur Migration `20160515085959_add_name_to_users.rb` zu migrieren, verwenden Sie `20160515085959` als Versionsnummer.

Erstellen Sie eine Join-Tabelle

Führen Sie den Befehl aus, um eine Verbindungstabelle zwischen `students` und `courses` zu erstellen

```
$ rails g migration CreateJoinTableStudentCourse student course
```

Dadurch wird die folgende Migration generiert:

```
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
```

Migrationen in verschiedenen Umgebungen ausführen

Führen Sie den Shell-Befehl aus, um Migrationen in der `test` auszuführen:

```
rake db:migrate RAILS_ENV=test
```

5,0

Ab Rails 5.0 können Sie anstelle von `rake rails` verwenden:

```
rails db:migrate RAILS_ENV=test
```

Fügen Sie einer Tabelle eine neue Spalte hinzu

Um eine neue Spalte hinzufügen `name` an die `users` - Tabelle, führen Sie den Befehl:

```
rails generate migration AddNameToUsers name
```

Dies generiert die folgende Migration:

```
class AddNameToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :name, :string
  end
end
```

Wenn der Migrationsname die Form `AddXXXToTABLE_NAME` gefolgt von einer Liste von Spalten mit Datentypen hat, enthält die generierte Migration die entsprechenden Anweisungen `add_column` .

Fügen Sie eine neue Spalte mit einem Index hinzu

Führen Sie den Befehl aus, um eine neue *indizierte* Spalten- `email` zur `users` hinzuzufügen:

```
rails generate migration AddEmailToUsers email:string:index
```

Dadurch wird die folgende Migration generiert:

```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :email, :string
    add_index :users, :email
  end
end
```

Entfernen Sie eine vorhandene Spalte aus einer Tabelle

Zum Entfernen der vorhandenen `name` von `users` Tabelle, führen Sie den Befehl:

```
rails generate migration RemoveNameFromUsers name:string
```

Dadurch wird die folgende Migration generiert:

```
class RemoveNameFromUsers < ActiveRecord::Migration[5.0]
```

```
  def change
    remove_column :users, :name, :string
  end
end
```

Wenn der Migrationsname die Form `RemoveXXXFromYYY` gefolgt von einer Liste von Spalten mit Datentypen, enthält die generierte Migration die entsprechenden Anweisungen `remove_column` .

Zwar ist es nicht erforderlich, den Datentyp (z. B `:string`) als Parameter für `remove_column` , es wird jedoch dringend empfohlen. Wenn der Datentyp *nicht* angegeben ist, ist die Migration nicht reversibel.

Fügen Sie einer Tabelle eine Referenzspalte hinzu

Führen Sie den folgenden Befehl aus, um einem `team` einen Verweis auf die `users` hinzuzufügen:

```
$ rails generate migration AddTeamRefToUsers team:references
```

Dies generiert die folgende Migration:

```
class AddTeamRefToUsers < ActiveRecord::Migration[5.0]
  def change
    add_reference :users, :team, foreign_key: true
  end
end
```

Diese Migration erstellt eine `team_id` Spalte in der `users` .

Wenn Sie für die hinzugefügte Spalte einen geeigneten `index` und ein `foreign_key` hinzufügen möchten, ändern Sie den Befehl in `rails generate migration AddTeamRefToUsers team:references:index` zu `rails generate migration AddTeamRefToUsers team:references:index` . Dadurch wird die folgende Migration generiert:

```
class AddTeamRefToUsers < ActiveRecord::Migration
  def change
    add_reference :users, :team, index: true
    add_foreign_key :users, :teams
  end
end
```

Wenn Sie Ihre Referenzspalte anders benennen möchten als die, die Rails automatisch generiert, fügen Sie der Migration Folgendes hinzu: (ZB: Sie möchten möglicherweise den `User` aufrufen, der den `Post` als `Author` in der Tabelle " `Post`).

```
class AddAuthorRefToPosts < ActiveRecord::Migration
  def change
    add_reference :posts, :author, references: :users, index: true
  end
end
```

Erstellen Sie eine neue Tabelle

Um eine neue zu erstellen `users` Tabelle mit den Spalten `name` und `salary` , führen Sie den Befehl:

```
rails generate migration CreateUsers name:string salary:decimal
```

Dadurch wird die folgende Migration generiert:

```
class CreateUsers < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      t.string :name
      t.decimal :salary
    end
  end
end
```

Wenn der Migrationsname die Form `CreateXXX` gefolgt von einer Liste von Spalten mit Datentypen, wird eine Migration generiert, die die Tabelle `XXX` mit den aufgelisteten Spalten erstellt.

Mehrere Spalten zu einer Tabelle hinzufügen

Wenn Sie einer Tabelle mehrere Spalten hinzufügen möchten, müssen Sie ein separates `field:type` Geben Sie bei Verwendung von `rails generate migration` Leerzeichen ein, um `rails generate migration` .

Die allgemeine Syntax lautet:

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

Die folgende Tabelle fügt der `users` beispielsweise `name` , `salary` und `email` Felder hinzu:

```
rails generate migration AddDetailsToUsers name:string salary:decimal email:string
```

Was erzeugt die folgende Migration:

```
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
```

Migrationen ausführen

Führen Sie den Befehl aus:

5,0

```
rake db:migrate
```

5,0

```
rails db:migrate
```

Durch die Angabe der Zielversion werden die erforderlichen Migrationen (Up, Down, Change) ausgeführt, bis die angegebene Version erreicht ist. Die `version number` ist hier das numerische Präfix des Dateinamens der Migration.

5,0

```
rake db:migrate VERSION=20080906120000
```

5,0

```
rails db:migrate VERSION=20080906120000
```

Rollback-Migrationen

Um ein `rollback` die letzte Migration, entweder durch das Rückgriff `change` Verfahren oder indem Sie die `down` Methode. Führen Sie den Befehl aus:

5,0

```
rake db:rollback
```

5,0

```
rails db:rollback
```

Rollback der letzten 3 Migrationen

5,0

```
rake db:rollback STEP=3
```

5,0

```
rails db:rollback STEP=3
```

`STEP` gibt die Anzahl der Migrationen an, die zurückgesetzt werden sollen.

Alle Migrationen rückgängig machen

5,0

```
rake db:rollback VERSION=0
```

5,0

```
rails db:rollback VERSION=0
```

Tabellen wechseln

Wenn Sie eine Tabelle mit einem falschen Schema erstellt haben, können Sie die Spalten und ihre Eigenschaften am einfachsten mit `change_table` . Überprüfen Sie das folgende Beispiel:

```
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
```

Die obige Migration ändert eine Tabelle `orders` . Hier ist eine zeilenweise Beschreibung der Änderungen:

1. `t.remove :ordered_at` entfernt die Spalte `ordered_at` aus der Tabelle `orders` .
2. `t.string :skew_number` fügt eine neue Zeichenfolgenspalte namens `skew_number` in die `orders` .
3. `t.index :skew_number` fügt einen Index für die Spalte `skew_number` in der `orders` .
4. `t.rename :location, :state` benennt die `location` in der `orders` in `state` .

Fügen Sie einer Tabelle eine eindeutige Spalte hinzu

Führen Sie den folgenden Befehl aus, um `users` eine neue *eindeutige* Spalten- `email` hinzuzufügen:

```
rails generate migration AddEmailToUsers email:string:uniq
```

Dadurch wird die folgende Migration erstellt:

```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :email, :string
    add_index :users, :email, unique: true
  end
end
```

Ändern Sie den Typ einer vorhandenen Spalte

Führen Sie den folgenden Befehl aus, um eine vorhandene Spalte in Rails mit einer Migration zu ändern:

```
rails g migration change_column_in_table
```

Dadurch wird eine neue Migrationsdatei im Verzeichnis `db/migration` (sofern noch nicht vorhanden), die die Datei mit dem Präfix timestamp und migrationsdatei enthält, die den folgenden Inhalt enthält:

```
def change
```

```
  change_column(:table_name, :column_name, :new_type)
end
```

Rails Guide - Spalten wechseln

Eine längere aber sicherere Methode

Der obige Code verhindert, dass der Benutzer die Migration jemals rückgängig macht. Sie können dieses Problem vermeiden, indem Sie die `change` in getrennte `up` und `down` aufteilen:

```
def up
  change_column :my_table, :my_column, :new_type
end

def down
  change_column :my_table, :my_column, :old_type
end
```

Migrationen wiederholen

Sie können mit dem Befehl " `redo` einen Rollback ausführen und dann erneut migrieren. Dies ist im Grunde eine Verknüpfung, die `rollback` und `migrate` kombiniert.

Führen Sie den Befehl aus:

5,0

```
rake db:migrate:redo
```

5,0

```
rails db:migrate:redo
```

Mit dem `STEP` Parameter können Sie mehrere Versionen zurückgeben.

So gehen Sie beispielsweise 3 Migrationen zurück:

5,0

```
rake db:migrate:redo STEP=3
```

5,0

```
rails db:migrate:redo STEP=3
```

Spalte mit Standardwert hinzufügen

Im folgenden Beispiel wird der `users` eine Spalte `admin` , und dieser Spalte wird der Standardwert `false` .

```
class AddDetailsToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :admin, :boolean, default: false
  end
end
```

Migrationen mit Standardwerten können in großen Tabellen mit beispielsweise PostgreSQL sehr lange dauern. Dies liegt daran, dass jede Zeile mit dem Standardwert für die neu hinzugefügte Spalte aktualisiert werden muss. Um dies zu umgehen (und Ausfallzeiten während der Bereitstellung zu reduzieren), können Sie die Migration in drei Schritte unterteilen:

1. Fügen Sie eine `add_column` -migration ähnlich der obigen hinzu, legen Sie jedoch keinen Standard fest
2. Stellen Sie die Spalte in einer Rake-Task oder auf der Konsole bereit, und aktualisieren Sie sie, während Ihre App ausgeführt wird. Stellen Sie sicher, dass Ihre Anwendung bereits Daten für neue / aktualisierte Zeilen in diese Spalte schreibt.
3. Fügen Sie eine weitere `change_column` Migration hinzu, die dann den Standardwert dieser Spalte in den gewünschten Standardwert ändert

Nullwerte verbieten

Um `null` in Ihren Tabellenspalten zu verbieten, fügen Sie Ihrer Migration den `:null` Parameter hinzu:

```
class AddPriceToProducts < ActiveRecord::Migration
  def change
    add_column :products, :float, null: false
  end
end
```

Migrationsstatus überprüfen

Wir können den Status von Migrationen durch Ausführen überprüfen

3,0 5,0

```
rake db:migrate:status
```

5,0

```
rails db:migrate:status
```

Die Ausgabe sieht folgendermaßen aus:

```
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 **********
```

Unter dem Statusfeld `up` bedeutet , dass die Migration ausgeführt wurde und `down` bedeutet , dass wir die Migration ausgeführt werden müssen.

Erstellen Sie eine Hstore-Spalte

`Hstore` Spalten können zum Speichern von Einstellungen nützlich sein. Sie sind in PostgreSQL-Datenbanken verfügbar, nachdem Sie die Erweiterung aktiviert haben.

```
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
```

Fügen Sie einen Selbstverweis hinzu

Eine Selbstreferenz kann hilfreich sein, um einen hierarchischen Baum aufzubauen. Dies kann mit `add_reference` in einer Migration erreicht werden.

```
class AddParentPages < ActiveRecord::Migration[5.0]
  def change
    add_reference :pages, :pages
  end
end
```

Die Fremdschlüsselspalte `pages_id` . Wenn Sie den Namen der Fremdschlüsselspalte festlegen möchten, müssen Sie zuerst die Spalte erstellen und anschließend die Referenz hinzufügen.

```
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
```

Erstellen Sie eine Array-Spalte

Eine `array` Spalte wird von PostgreSQL unterstützt. Rails konvertiert ein PostgreSQL-Array automatisch in ein Ruby-Array und umgekehrt.

Erstellen Sie eine Tabelle mit einer `array` :

```
create_table :products do |t|
  t.string :name
```

```
  t.text :colors, array: true, default: []
end
```

Fügen Sie ein `array` Spalte in eine vorhandene Tabelle:

```
add_column :products, :colors, array: true, default: []
```

Fügen Sie einen Index für eine `array` :

```
add_index :products, :colors, using: 'gin'
```

Hinzufügen einer NOT NULL-Einschränkung zu vorhandenen Daten

`company_id` , Sie möchten der `users` einen Fremdschlüssel `company_id` hinzufügen und eine `NOT NULL` Einschränkung haben. Wenn Sie bereits Daten in `users` haben, müssen Sie dies in mehreren Schritten tun.

```
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
```

ActiveRecord-Migrationen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/1087/activerecord-migrationen

Kapitel 9: ActiveRecord-Sperrung

Examples

Optimistisches Sperren

```
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
```

Pessimistisches Sperren

```
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
```

ActiveRecord-Sperrung online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3866/activerecord-sperrung

Kapitel 10: ActiveRecord-Transaktionen

Bemerkungen

Transaktionen sind Schutzblöcke, bei denen SQL-Anweisungen nur dauerhaft sind, wenn sie alle als eine einzige Aktion erfolgreich sein können. Das klassische Beispiel ist eine Überweisung zwischen zwei Konten, bei denen Sie nur dann eine Einzahlung vornehmen können, wenn die Auszahlung erfolgreich war, und umgekehrt. Transaktionen erzwingen die Integrität der Datenbank und schützen die Daten vor Programmfehlern oder Datenbankausfällen. Sie sollten also grundsätzlich Transaktionsblöcke verwenden, wenn Sie mehrere Anweisungen haben, die zusammen oder nicht ausgeführt werden müssen.

Examples

Grundlegendes Beispiel

Zum Beispiel:

```
ActiveRecord::Base.transaction do
  david.withdrawal(100)
  mary.deposit(100)
end
```

Dieses Beispiel nimmt nur Geld von David und gibt es an Mary, wenn weder Abhebung noch Einzahlung eine Ausnahme darstellen. Ausnahmen erzwingen einen ROLLBACK, der die Datenbank vor dem Beginn der Transaktion in den Status zurücksetzt. Beachten Sie jedoch, dass die Instanzdaten der Objekte nicht in den Zustand vor der Transaktion zurückversetzt werden.

Verschiedene ActiveRecord-Klassen in einer einzelnen Transaktion

Obwohl die Transaktionsklassenmethode für einige ActiveRecord-Klasse aufgerufen wird, müssen die Objekte im Transaktionsblock nicht alle Instanzen dieser Klasse sein. Dies liegt daran, dass Transaktionen pro Datenbankverbindung und nicht pro Modell erfolgen.

In diesem Beispiel wird ein Saldasatz transaktional gespeichert, obwohl die Transaktion für die Konto-Klasse aufgerufen wird:

```
Account.transaction do
  balance.save!
  account.save!
end
```

Die Transaktionsmethode ist auch als Modellinstanzmethode verfügbar. Sie können beispielsweise auch Folgendes tun:

```
balance.transaction do
```

```
  balance.save!
  account.save!
end
```

Mehrere Datenbankverbindungen

Eine Transaktion wirkt auf eine einzelne Datenbankverbindung. Wenn Sie über mehrere klassenspezifische Datenbanken verfügen, schützt die Transaktion die Interaktion zwischen ihnen nicht. Eine Problemumgehung besteht darin, eine Transaktion für jede Klasse zu beginnen, deren Modelle Sie ändern:

```
Student.transaction do
  Course.transaction do
    course.enroll(student)
    student.units += course.units
  end
end
```

Dies ist eine schlechte Lösung, aber vollständig verteilte Transaktionen liegen nicht im Bereich von ActiveRecord.

Speichern und Zerstören werden automatisch in eine Transaktion eingeschlossen

Sowohl #save als auch #destroy werden in einer Transaktion verpackt, die sicherstellt, dass das, was Sie bei Validierungen oder Rückrufen tun, unter ihrer geschützten Abdeckung geschieht. Sie können Validierungen verwenden, um nach Werten zu `after_*` , von denen die Transaktion abhängig ist, oder Sie können Ausnahmen in den Rückrufen für das Rollback `after_*` , einschließlich `after_*` .

Folglich werden Änderungen an der Datenbank nicht außerhalb Ihrer Verbindung sichtbar, bis der Vorgang abgeschlossen ist. Wenn Sie beispielsweise versuchen, den Index einer Suchmaschine in `after_save` aktualisieren, wird der aktualisierte Datensatz nicht `after_save` . Der `after_commit` Callback ist der einzige, der ausgelöst wird, wenn die Aktualisierung festgeschrieben ist.

Rückrufe

Es gibt zwei Arten von Rückrufen, die mit dem `after_commit` und `after_rollback` Transaktionen verbunden sind: `after_commit` und `after_rollback` .

`after_commit` Rückrufe werden für jeden Datensatz, der innerhalb einer Transaktion gespeichert oder zerstört wurde, unmittelbar nach dem `after_commit` der Transaktion aufgerufen. `after_rollback` Rückrufe werden für jeden Datensatz, der innerhalb einer Transaktion gespeichert oder zerstört wurde, unmittelbar nach dem `after_rollback` der Transaktion oder des Sicherungspunkts aufgerufen.

Diese Callbacks sind hilfreich für die Interaktion mit anderen Systemen, da Ihnen garantiert wird, dass der Callback nur ausgeführt wird, wenn sich die Datenbank in einem permanenten Zustand

befindet. `after_commit` ist beispielsweise ein guter Ort, um einen Cache zu `after_commit` da das Löschen innerhalb einer Transaktion dazu führen könnte, dass der Cache vor der Aktualisierung der Datenbank erneut generiert wird.

Transaktion rückgängig machen

`ActiveRecord::Base.transaction` verwendet die Ausnahme `ActiveRecord::Base.transaction` `ActiveRecord::Rollback` , um ein absichtliches Rollback von anderen Ausnahmesituationen zu unterscheiden. Normalerweise führt das `.transaction` einer Ausnahme dazu, dass die `.transaction` Methode die Datenbanktransaktion `.transaction` und die Ausnahme `.transaction` . Wenn Sie jedoch eine `ActiveRecord::Rollback` Ausnahme `ActiveRecord::Rollback` , wird die Datenbanktransaktion zurückgesetzt, ohne die Ausnahme zu übergeben.

Zum Beispiel könnten Sie dies in Ihrem Controller tun, um eine Transaktion rückgängig zu machen:

```
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
```

ActiveRecord-Transaktionen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/4688/activerecord-transaktionen

Kapitel 11: ActiveRecord-Transaktionen

Einführung

ActiveRecord-Transaktionen sind Schutzblöcke, bei denen die Abfolge aktiver Datensatzabfragen nur dann dauerhaft ist, wenn sie alle als eine atomare Aktion erfolgreich sind.

Examples

Erste Schritte mit aktiven Datensatztransaktionen

Aktive Datensatztransaktionen können sowohl auf Model-Klassen als auch auf Model-Instanzen angewendet werden. Die Objekte innerhalb des Transaktionsblocks müssen nicht alle Instanzen derselben Klasse sein. Dies liegt daran, dass Transaktionen pro Datenbankverbindung und nicht pro Modell erfolgen. Zum Beispiel:

```
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
```

Die Verwendung von save with a bang stellt sicher, dass die Transaktion automatisch zurückgesetzt wird, wenn die Ausnahme ausgelöst wird. Nach dem Zurücksetzen wechselt die Steuerung zum Rettungsblock für die Ausnahme. **Stellen Sie sicher, dass Sie die Ausnahmen retten, die aus dem Speicher geworfen wurden! in Transaktionsblock.**

Wenn Sie save! Nicht verwenden möchten, können Sie `raise ActiveRecord::Rollback` manuell erhöhen `raise ActiveRecord::Rollback` wenn das Speichern fehlschlägt. Sie müssen diese Ausnahme nicht behandeln. Anschließend wird die Transaktion rückgängig gemacht, und die Kontrolle wird nach dem Transaktionsblock für die nächste Anweisung übernommen.

```
   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"
```

ActiveRecord-Transaktionen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9326/activerecord-transaktionen

Kapitel 12: ActiveRecord-Überprüfungen

Examples

Überprüfung der Numerizität eines Attributs

Diese Überprüfung beschränkt die Einfügung nur numerischer Werte.

```
class Player < ApplicationRecord
  validates :points, numericality: true
  validates :games_played, numericality: { only_integer: true }
end
```

Neben `:only_integer` akzeptiert dieser Helper auch die folgenden Optionen, um den akzeptablen Werten Einschränkungen hinzuzufügen:

- `:greater_than` - Gibt an, dass der Wert größer sein muss als der angegebene Wert. Die Standardfehlermeldung für diese Option lautet "muss größer als% {count} sein".
- `:greater_than_or_equal_to` - Gibt an, dass der Wert größer oder gleich dem angegebenen Wert sein muss. Die Standardfehlermeldung für diese Option lautet "muss größer oder gleich% {count} sein".
- `:equal_to` - Gibt an, dass der Wert dem angegebenen Wert entsprechen muss. Die Standardfehlermeldung für diese Option lautet "muss gleich% {count} sein".
- `:less_than` - Gibt an, dass der Wert unter dem angegebenen Wert liegen muss. Die Standardfehlermeldung für diese Option lautet "muss kleiner als% {count} sein".
- `:less_than_or_equal_to` - Gibt an, dass der Wert kleiner oder gleich dem angegebenen Wert sein muss. Die Standardfehlermeldung für diese Option lautet "muss kleiner oder gleich% {count} sein".
- `:other_than` - Gibt an, dass der Wert vom angegebenen Wert `:other_than` muss. Die Standardfehlermeldung für diese Option lautet "muss nicht% {count} sein".
- `:odd` - Gibt an, dass der Wert eine ungerade Zahl sein muss, wenn er auf true gesetzt ist. Die Standardfehlermeldung für diese Option lautet "Muss ungerade sein".
- `:even` - Gibt an, dass der Wert eine gerade Zahl sein muss, wenn er auf true gesetzt ist. Die Standardfehlermeldung für diese Option lautet "Muss gerade sein".

  Standardmäßig erlaubt die Numerizität keine Nullwerte. Sie können die Option allow_nil: true verwenden, um dies zuzulassen.

Überprüfen Sie die Eindeutigkeit eines Attributs

Dieser Helfer überprüft, ob der Wert des Attributs eindeutig ist, bevor das Objekt gespeichert wird.

```
class Account < ApplicationRecord
  validates :email, uniqueness: true
end
```

Es gibt eine Option `:scope` , mit der Sie ein oder mehrere Attribute angeben können, mit denen die Eindeutigkeitsprüfung eingeschränkt wird:

```
class Holiday < ApplicationRecord
  validates :name, uniqueness: { scope: :year,
    message: "should happen once per year" }
end
```

Es gibt auch eine Option `:case_sensitive` , mit der Sie festlegen können, ob die Eindeutigkeitseinschränkung die Groß- und Kleinschreibung `:case_sensitive` . Diese Option ist standardmäßig auf `true` .

```
class Person < ApplicationRecord
  validates :name, uniqueness: { case_sensitive: false }
end
```

Vorhandensein eines Attributs prüfen

Dieser Helfer überprüft, ob die angegebenen Attribute nicht leer sind.

```
class Person < ApplicationRecord
  validates :name, presence: true
end

Person.create(name: "John").valid? # => true
Person.create(name: nil).valid? # => false
```

Sie können die `absence` zu überprüfen, ob die angegebenen Attribute nicht vorhanden sind. Nutzt die `present?` Methode zum Überprüfen auf null oder leere Werte.

```
class Person < ApplicationRecord
  validates :name, :login, :email, absence: true
end
```

Hinweis: Wenn das Attribut ein `boolean` Attribut ist, können Sie die übliche Anwesenheitsüberprüfung nicht verwenden (das Attribut wäre für einen `false` Wert nicht gültig). Dies können Sie mit einer Inklusionsvalidierung erreichen:

```
validates :attribute, inclusion: [true, false]
```

Validierungen überspringen

Verwenden Sie die folgenden Methoden, wenn Sie die Validierungen überspringen möchten. Diese Methoden speichern das Objekt in der Datenbank, auch wenn es ungültig ist.

- dekrementieren!
- decrement_counter
- Zuwachs!
- increment_counter

- Umschalten!
- berühren
- Alle aktualisieren
- update_attribute
- update_column
- Update-Spalten
- Update_Counters

Sie können die Validierung beim Speichern auch überspringen, indem Sie `validate` als Argument zum `save`

```
User.save(validate: false)
```

Länge eines Attributs überprüfen

```
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
```

Die möglichen Längeneinschränkungsoptionen sind:

- `:minimum` - Das Attribut darf nicht kleiner als die angegebene Länge sein.
- `:maximum` - Das Attribut darf nicht länger als die angegebene Länge sein.
- `:in` (oder `:within`) - Die Attributlänge muss in einem bestimmten Intervall enthalten sein. Der Wert für diese Option muss ein Bereich sein.
- `:is` - Die Attributlänge muss dem angegebenen Wert entsprechen.

Gruppierungsüberprüfung

In manchen Fällen ist es sinnvoll, dass mehrere Überprüfungen eine Bedingung verwenden. Dies kann leicht mit with_options erreicht werden.

```
class User < ApplicationRecord
  with_options if: :is_admin? do |admin|
    admin.validates :password, length: { minimum: 10 }
    admin.validates :email, presence: true
  end
end
```

Alle Validierungen im with_options-Block haben die Bedingung automatisch bestanden, wenn:: is_admin?

Benutzerdefinierte Validierungen

Sie können Ihre eigenen Validierungen hinzufügen, indem Sie neue Klassen hinzufügen, die von `ActiveModel::Validator` oder `ActiveModel::EachValidator` . Beide Methoden sind ähnlich,

funktionieren jedoch auf etwas unterschiedliche Weise:

`ActiveModel::Validator` und `validates_with`

Implementieren Sie die `validate` Methode, die einen Datensatz als Argument verwendet und die Validierung daran ausführt. Verwenden Sie dann `validates_with` mit der Klasse im Modell.

```
# 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` und `validate`

Wenn Sie es vorziehen, Ihren neuen Prüfer mit der allgemeinen `validate` Methode für einen einzelnen Parameter zu verwenden, erstellen Sie eine Klasse, die `ActiveModel::EachValidator` und implementieren Sie die `validate_each` Methode, die drei Argumente verwendet: `record` , `attribute` und `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
```

Weitere Informationen zu den Rails-Führungen .

Überprüft das Format eines Attributs

Stellen Sie sicher, dass der Wert eines Attributs mit einem regulären Ausdruck übereinstimmt, und zwar mit `format` und der Option `with` .

```
class User < ApplicationRecord
  validates :name, format: { with: /\A\w{6,10}\z/ }
end
```

Sie können auch eine Konstante definieren und ihren Wert auf einen regulären Ausdruck festlegen

und an die Option `with:` . Dies kann für wirklich komplexe reguläre Ausdrücke bequemer sein

```
PHONE_REGEX = /\A\(\d{3}\)\d{3}-\d{4}\z/
validates :phone, format: { with: PHONE_REGEX }
```

Die Standardfehlermeldung `is invalid` . Dies kann mit der `:message` Option geändert werden.

```
validates :bio, format: { with: /\A\D+\z/, message: "Numbers are not allowed" }
```

Die Umkehrung antwortet ebenfalls, und Sie können angeben, dass ein Wert mit der Option `without:` *keinem* regulären Ausdruck entsprechen sollte

Überprüft die Aufnahme eines Attributs

Sie können mit Hilfe des `inclusion:` einschließen `inclusion:` prüfen, ob ein Wert in einem Array enthalten ist. Die Option `:in` und ihr Alias `:within` zeigt die Menge der zulässigen Werte.

```
class Country < ApplicationRecord
  validates :continent, inclusion: { in: %w(Africa Antartica Asia Australia
                                            Europe North America South America) }
end
```

Um zu überprüfen, ob ein Wert nicht in einem Array enthalten ist, verwenden Sie den Helper `exclusion:`

```
class User < ApplicationRecord
  validates :name, exclusion: { in: %w(admin administrator owner) }
end
```

Bedingte Validierung

In manchen Fällen müssen Sie die Aufzeichnung möglicherweise nur unter bestimmten Bedingungen überprüfen.

```
class User < ApplicationRecord
  validates :name, presence: true, if: :admin?

  def admin?
    conditional here that returns boolean value
  end
end
```

Wenn Ihre Bedingung sehr klein ist, können Sie ein Proc verwenden:

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: Proc.new { |user| user.last_name.blank? }
end
```

Für negative Bedingungen können Sie verwenden, es `unless` :

```
class User < ApplicationRecord
  validates :first_name, presence: true, unless: Proc.new { |user| user.last_name.present? }
end
```

Sie können auch einen String übergeben, der über `instance_eval` ausgeführt wird:

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: 'last_name.blank?'
end
```

Bestätigung des Attributs

Sie sollten dies verwenden, wenn Sie zwei Textfelder haben, die exakt denselben Inhalt erhalten sollen. Beispielsweise möchten Sie möglicherweise eine E-Mail-Adresse oder ein Passwort bestätigen. Bei dieser Überprüfung wird ein ***virtuelles*** Attribut erstellt, dessen Name dem Namen des Felds entspricht, das mit angehängter `_confirmation` bestätigt werden `_confirmation` .

```
class Person < ApplicationRecord
  validates :email, confirmation: true
end
```

Hinweis Diese Prüfung wird nur durchgeführt, wenn `email_confirmation` nicht gleich Null ist.

Um eine Bestätigung anzufordern, fügen Sie eine Präsenzprüfung für das Bestätigungsattribut hinzu.

```
class Person < ApplicationRecord
  validates :email,      confirmation: true
  validates :email_confirmation, presence: true
end
```

Quelle

Verwendung: bei Option

Mit der Option `:on` können Sie angeben, wann die Überprüfung erfolgen soll. Das Standardverhalten für alle integrierten Validierungshelfer ist das Ausführen beim Speichern (sowohl beim Erstellen eines neuen Datensatzes als auch beim Aktualisieren).

```
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
```

ActiveRecord-Überprüfungen online lesen: https://riptutorial.com/de/ruby-on-

rails/topic/2105/activerecord-uberprufungen

Kapitel 13: ActiveRecord-Verknüpfungen

Examples

gehört

Eine `belongs_to` -to- `belongs_to` Assoziation stellt eine Eins-zu-Eins-Verbindung mit einem anderen Modell her. `belongs_to` gehört jede Instanz des deklarierenden Modells zu einer Instanz des anderen Modells.

Wenn Ihre Anwendung beispielsweise Benutzer und Beiträge enthält und jeder Beitrag genau einem Benutzer zugewiesen werden kann, müssen Sie das Beitragsmodell folgendermaßen definieren:

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

In Ihrer Tabellenstruktur könnten Sie dann haben

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

has_one

Eine `has_one` Assoziation stellt eine Eins-zu-Eins-Verbindung mit einem anderen Modell her, jedoch mit unterschiedlicher Semantik. Diese Zuordnung zeigt an, dass jede Instanz eines Modells eine Instanz eines anderen Modells enthält oder besitzt.

Wenn zum Beispiel jeder Benutzer in Ihrer Anwendung nur ein Konto hat, würden Sie das Benutzermodell folgendermaßen definieren:

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

Wenn Sie in Active Record eine `has_one` Beziehung haben, stellt Active Record sicher, dass nur ein Datensatz mit dem Fremdschlüssel vorhanden ist.

Hier in unserem Beispiel: In der Kontentabelle kann es nur einen Datensatz mit einer bestimmten user_id geben. Wenn Sie versuchen, ein weiteres Konto für denselben Benutzer zuzuordnen, wird der Fremdschlüssel des vorherigen Eintrags als NULL (verwaisen) und automatisch ein neuer erstellt. Dadurch wird der vorherige Eintrag auf Null gesetzt, auch wenn das Speichern des neuen Eintrags fehlschlägt, um die Konsistenz zu gewährleisten.

```
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]
```

hat viele

Eine `has_many` Zuordnung zeigt eine Eins-zu-Viele-Verbindung mit einem anderen Modell an. Diese Assoziation befindet sich im Allgemeinen auf der anderen Seite einer Toits_to-Assoziation.

Diese Zuordnung zeigt an, dass jede Instanz des Modells keine oder mehrere Instanzen eines anderen Modells hat.

In einer Anwendung, die Benutzer und Beiträge enthält, könnte das Benutzermodell beispielsweise folgendermaßen deklariert werden:

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

Die Tabellenstruktur von `Post` würde die gleiche bleiben wie im Beispiel " `belongs_to` . Im Gegensatz dazu erfordert der `User` keine Schemaänderungen.

Wenn Sie die Liste aller veröffentlichten Beiträge für den `User` anzeigen möchten, können Sie Folgendes hinzufügen (dh Sie können den Assoziationsobjekten Bereiche hinzufügen):

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

Polymorphe Verbindung

Durch diese Art der Zuordnung kann ein ActiveRecord-Modell zu mehr als einer Art von Modelldatensatz gehören. Allgemeines Beispiel:

```
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
```

Ohne diese Zuordnung hätten Sie alle diese Fremdschlüssel in Ihrer Adresstabelle, aber Sie hätten immer nur einen Wert für einen von ihnen, da eine Adresse in diesem Szenario nur einer

Entität (Mensch oder Firma) gehören kann. So würde es aussehen:

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

Die has_many: durch Vereinigung

Eine `has_many :through` Assoziation wird häufig verwendet, um eine `has_many :through` `many-to-many` `has_many :through` `many-to-many` Verbindung mit einem anderen Modell herzustellen. Diese Zuordnung zeigt an, dass das deklarierende Modell mit null oder mehr Instanzen eines anderen Modells abgeglichen werden kann, indem ein drittes Modell durchlaufen wird.

Stellen Sie sich beispielsweise eine medizinische Praxis vor, in der Patienten Termine für einen Arztbesuch vereinbaren. Die relevanten Verbandserklärungen könnten so aussehen:

```
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
```

Die has_one: durch Assoziation

Eine `has_one :through` Assoziation stellt eine `one-to-one` Verbindung mit einem anderen Modell her. Diese Zuordnung zeigt an, dass das deklarierende Modell mit einer Instanz eines anderen Modells abgeglichen werden kann, indem ein drittes Modell durchlaufen wird.

Wenn beispielsweise jeder `supplier` über ein `account` verfügt und jedes Konto mit einem Kontohistor verbunden ist, könnte das Lieferantenmodell folgendermaßen aussehen:

```
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
```

Die has_and_belongs_to_many-Verbindung

Eine `has_and_belongs_to_many` Assoziation stellt eine direkte `many-to-many` Verbindung mit einem anderen Modell her, ohne dass ein Modell dazwischenliegt.

Wenn Ihre Anwendung beispielsweise `assemblies` und `parts` enthält und jede Baugruppe viele Teile hat und jedes Teil in vielen Baugruppen vorkommt, können Sie die Modelle auf diese Weise deklarieren:

```
class Assembly < ApplicationRecord
  has_and_belongs_to_many :parts
end

class Part < ApplicationRecord
  has_and_belongs_to_many :assemblies
end
```

Selbstreferentielle Vereinigung

Die selbstreferenzielle Assoziation wird verwendet, um ein Modell mit sich selbst zu verknüpfen. Das häufigste Beispiel wäre, die Verbindung zwischen einem Freund und seinem Anhänger zu verwalten.

Ex.

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

Jetzt können Sie Modelle wie assoziieren.

```
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
```

und das andere Modell wird aussehen;

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

ActiveRecord-Verknüpfungen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/1820/activerecord-verknupfungen

Kapitel 14: ActiveSupport

Bemerkungen

ActiveSupport ist ein Hilfsmittel aus Universalwerkzeugen, die vom Rest des Rails-Frameworks verwendet werden.

Eine der primären Methoden, mit denen diese Tools bereitgestellt werden, besteht darin, die systemeigenen Typen von Ruby zu überwachen. Diese werden als **Core-Erweiterungen bezeichnet** .

Examples

Core-Erweiterungen: String-Zugriff

String # at

Gibt einen Teilstring eines String-Objekts zurück. Gleiche Schnittstelle wie `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)   # => ""
```

String # von

Gibt eine Teilzeichenfolge von der angegebenen Position bis zum Ende der Zeichenfolge zurück.

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

Zeichenfolge # bis

Gibt einen Teilstring vom Anfang des Strings an die angegebene Position zurück.
Wenn die Position negativ ist, wird sie vom Ende der Zeichenfolge gezählt.

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

```
str.to(3)  # => "hell"
str.to(-2) # => "hell"
```

`from` und `to` kann im Tandem verwendet werden.

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

String # zuerst

Gibt das erste Zeichen oder eine bestimmte Anzahl von Zeichen bis zur Länge der Zeichenfolge zurück.

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

String # last

Gibt das letzte Zeichen oder eine bestimmte Anzahl von Zeichen ab dem Ende der Zeichenfolge zurück und zählt rückwärts.

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

Core-Erweiterungen: Konvertierung von Zeichenfolgen in Datum / Uhrzeit

String # to_time

Konvertiert eine Zeichenfolge in einen Zeitwert. Der `form` kann entweder `:utc` oder `:local` , der `:utc` `: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
```

String # to_date

Konvertiert eine Zeichenfolge in einen Datumswert.

```
"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
```

Zeichenfolge # bis_datetime

Konvertiert eine Zeichenfolge in einen DateTime-Wert.

```
"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
```

Core-Erweiterungen: Zeichenkettenausschluss

Zeichenfolge # ausschließen?

Die Umkehrung von `String#include?`

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

Core-Erweiterungen: String-Filter

String # squish

Gibt eine Version der angegebenen Zeichenfolge ohne führenden oder nachgestellten Leerzeichen zurück und kombiniert alle aufeinander folgenden Leerzeichen im Inneren zu einzelnen Leerzeichen. Zerstörerische Version `squish!` arbeitet direkt auf der String-Instanz.

Verarbeitet sowohl ASCII- als auch Unicode-Leerzeichen.

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

Zeichenfolge # entfernen

Gibt eine neue Zeichenfolge zurück, wobei alle Vorkommen der Muster entfernt wurden. Zerstörende Version `remove!` arbeitet direkt auf der angegebenen Zeichenkette.

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

String # wird abgeschnitten

Gibt eine Kopie einer angegebenen Zeichenfolge zurück, die bei einer angegebenen Länge abgeschnitten ist, wenn die Zeichenfolge länger als die Länge ist.

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

Übergeben Sie einen String oder ein reguläres `:separator` um bei einem natürlichen Bruch abzuschneiden

```
'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..."
```

String # truncate_words

Gibt eine Zeichenfolge zurück, die nach einer bestimmten Anzahl von Wörtern abgeschnitten wurde.

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

Übergeben Sie eine Zeichenfolge oder einen regulären Ausdruck, um ein anderes Trennzeichen für Wörter anzugeben

```
'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..."
```

Die letzten Zeichen werden durch die Zeichenfolge für die `:omission` (Standardeinstellung "...").

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

Zeichenfolge # strip_heredoc

Streifen Einrückung in Heredocs. Sucht nach der am wenigsten eingerückten, nicht leeren Zeile und entfernt diese Menge an führendem Leerzeichen.

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

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

Der Benutzer würde sehen

```
This command does such and such.

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

Core-Erweiterungen: String Flexion

String # pluralize

Gibt die Pluralform der Zeichenfolge zurück. Nimmt optional einen `count` Parameter an und gibt die Singularform zurück, wenn `count == 1` . Akzeptiert auch einen `locale` Parameter für die sprachspezifische Pluralisierung.

```
'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

Gibt die Singularform der Zeichenfolge zurück. Akzeptiert einen optionalen `locale` Parameter.

```
'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 # konstante

Versucht, eine deklarierte Konstante mit dem in der Zeichenfolge angegebenen Namen zu finden. Ein `NameError` wenn der Name nicht in CamelCase enthalten ist oder nicht initialisiert ist.

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

String # safe_constantize

Führt eine `constantize` , gibt jedoch `nil` anstatt `NameError` .

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

Zeichenfolge # camelize

Konvertiert Strings standardmäßig in UpperCamelCase, wenn `:lower` als Parameter angegeben wird, stattdessen in lowerCamelCase.

Alias: `camelcase`

Hinweis: konvertiert auch `/` in `::` was zum Konvertieren von Pfaden in Namespaces nützlich ist.

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

String # titleize

Schreibt alle Wörter groß und ersetzt einige Zeichen in der Zeichenfolge, um einen schöneren Titel zu erzeugen.

Alias: `titlecase`

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

String # unterstreichen

Erzeugt aus dem Ausdruck in der Zeichenfolge ein unterstrichenes, kleingeschriebenes Formular. Die Umkehrung von `camelize` .

Hinweis: `underscore` ändern auch `::` in `/` , um Namespaces in Pfade umzuwandeln.

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

String # dasherize

Ersetzt Unterstriche durch Striche in der Zeichenfolge.

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

String # demodulize

Entfernt den Modulteil aus dem konstanten Ausdruck in der Zeichenfolge.

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

String # dekonstantisieren

Entfernt das äußerste rechte Segment aus dem konstanten Ausdruck in der Zeichenfolge.

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

String # parametrisieren

Ersetzt Sonderzeichen in einer Zeichenfolge, sodass sie als Teil einer 'hübschen' URL verwendet werden kann.

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

Behalten Sie die Groß- und Kleinschreibung der Zeichen in einer Zeichenfolge mit dem Argument `:preserve_case` .

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

Ein sehr häufiger Anwendungsfall für die `parameterize` ist das Überschreiben der `to_param` Methode eines ActiveRecord-Modells, um mehr beschreibende URL-Slugs zu unterstützen.

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

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

String # tableize

Erzeugt den Namen einer Tabelle, wie es Rails für Modelle mit Tabellennamen tut. Pluralisiert das letzte Wort in der Zeichenfolge.

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

String # klassifizieren

Gibt eine Klassennamenzeichenfolge aus einem Mehrfach-Tabellennamen zurück, wie es Rails für Tabellennamen für Modelle tut.

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

String # humanize

Schreibt das erste Wort in `_id` , verwandelt Unterstriche in Leerzeichen und `_id` ggf. eine nachgestellte `_id` .

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

String # upcase_first

Wandelt nur das erste Zeichen in Großbuchstaben um.

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

String # foreign_key

Erstellt einen Fremdschlüsselnamen aus einem Klassennamen. Übergeben Sie `false` param, um das Hinzufügen von _ zwischen Name und `id` zu deaktivieren.

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

ActiveSupport online lesen: https://riptutorial.com/de/ruby-on-rails/topic/4490/activesupport

Kapitel 15: Admin-Panel hinzufügen

Einführung

Wenn Sie Ihrer Rails-Anwendung ein Admin-Panel hinzufügen möchten, ist dies nur eine Frage von Minuten.

Syntax

1. Gem-Datei und Writer-Gem 'tracks_admin', '~> 1.0' öffnen
2. Bundle installieren
3. Schienen g schienen_admin: installieren
4. Sie werden nach der Admin-Route gefragt, wenn Sie die Standard-Eingabetaste drücken möchten.
5. Gehen Sie nun zu app / config / initializers / schienen_admin.rb und fügen Sie den folgenden Code ein: config.authorize_with, und leiten Sie den Pfad in den main_app.root_path um, sofern nicht current_user.try (: admin?) End Benutzer werden zum Root-Pfad umgeleitet.
6. Für weitere Details überprüfen Sie die Dokumentation dieses Edelsteins. https://github.com/sferik/rails_admin/wiki

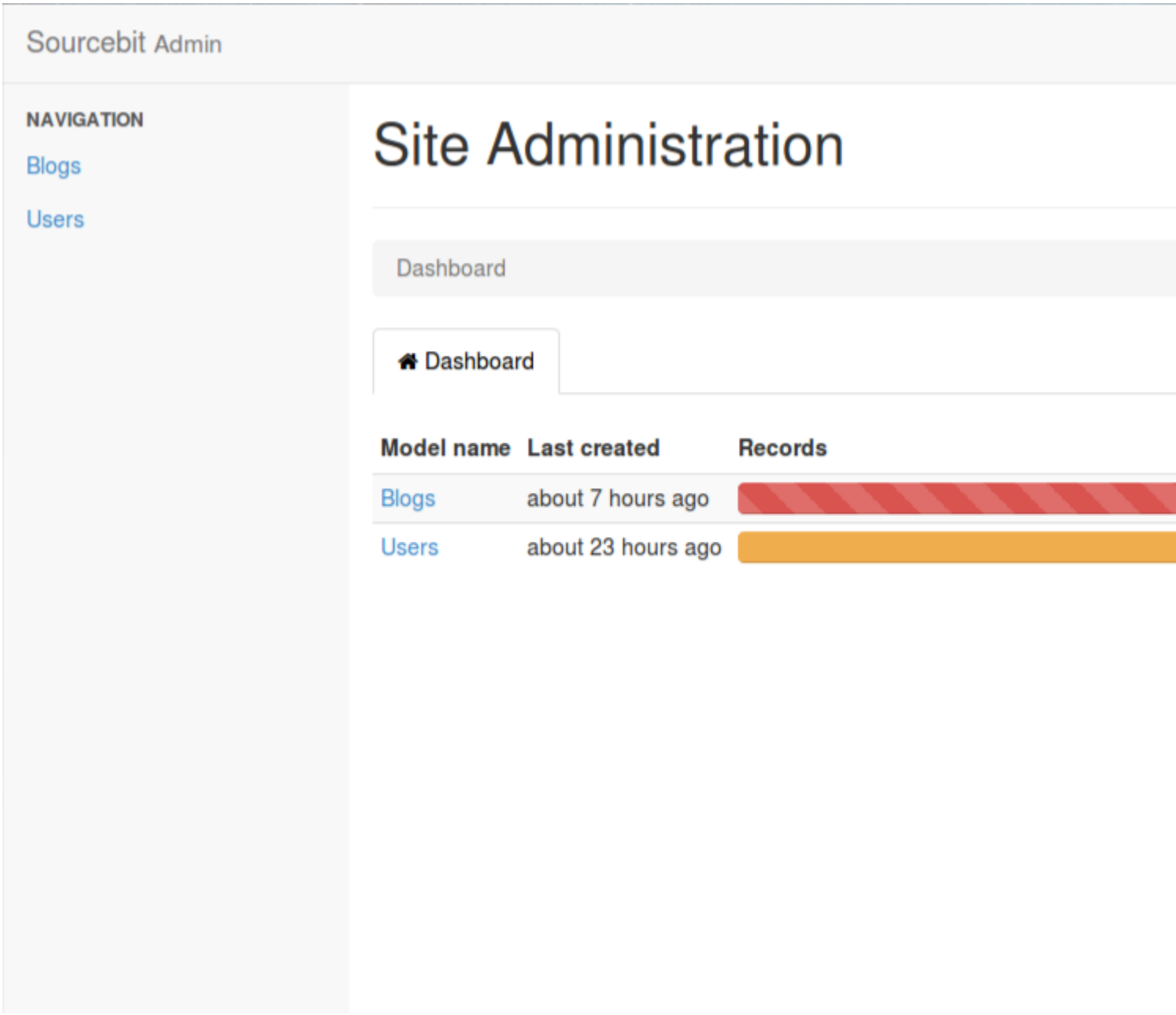
Bemerkungen

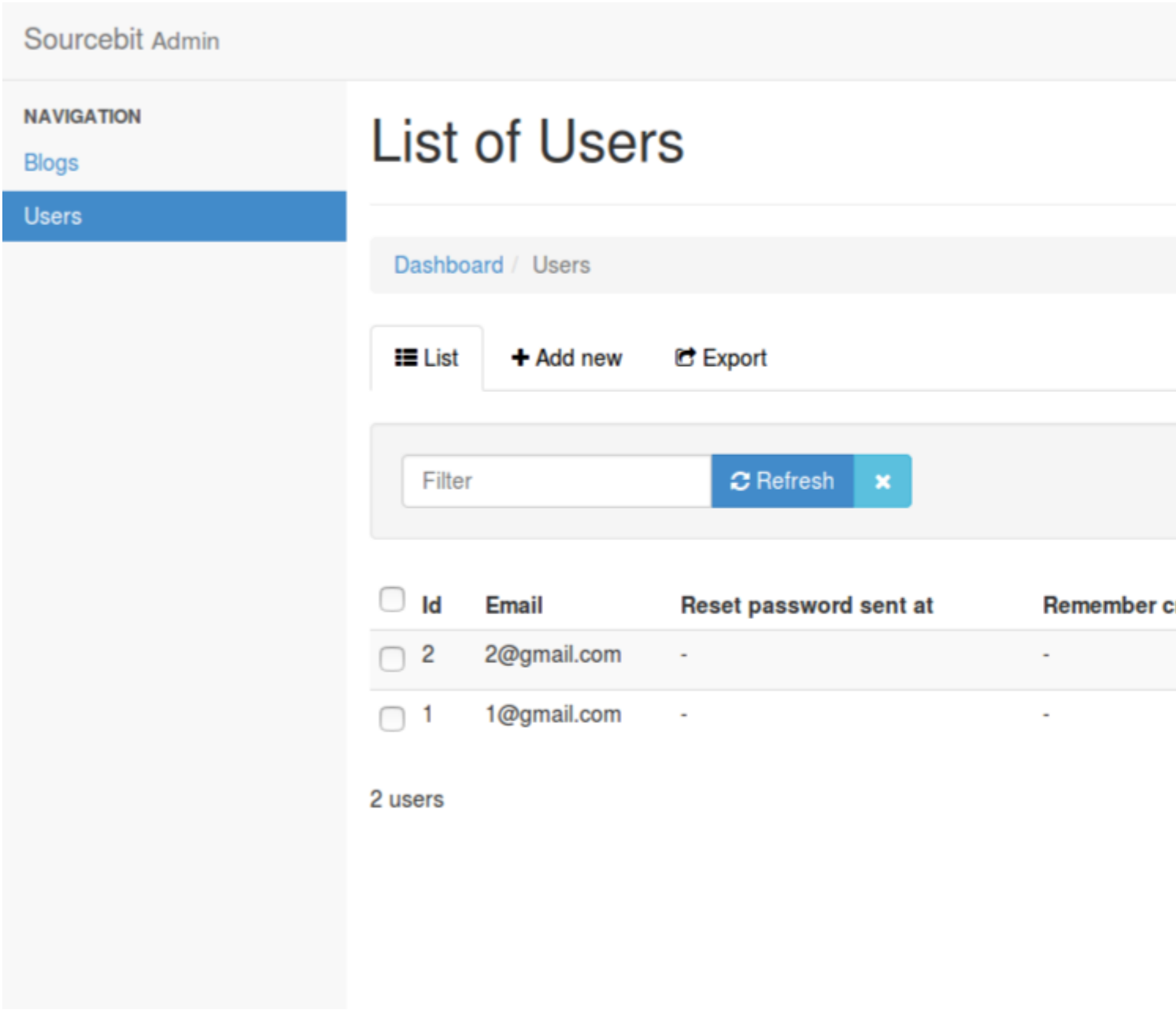
Verwenden Sie es, wenn Sie Admin für Ihre Website haben möchten. Andernfalls ist dies nicht erforderlich. Es ist einfacher und leistungsfähiger als active_admin gem. Sie können dies jederzeit nach dem Erstellen von Benutzern hinzufügen. Vergessen Sie nicht, vor dem vierten Schritt einen Benutzer als Administrator festzulegen. Verwenden Sie cancan zum Erteilen von Rollen.

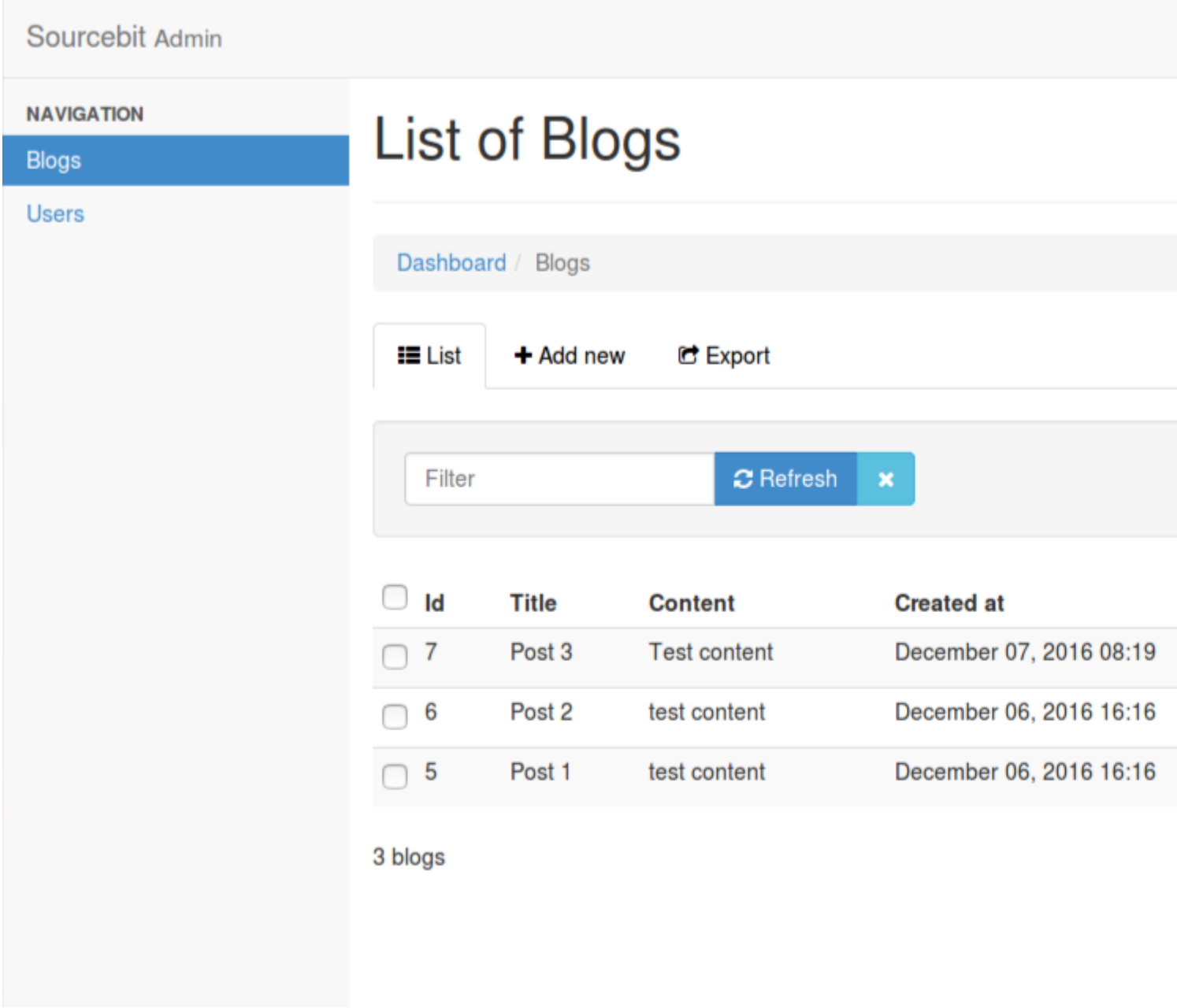
Examples

Hier sind also ein paar Screenshots aus dem Admin-Panel, die die Schienen "schienen_Admin" verwenden.

Wie Sie sehen, ist das Layout dieses Edelsteins sehr ansprechend und benutzerfreundlich.


Sourcebit Admin
NAVIGATION
Blogs
Users
Site Administration
Dashboard
Dashboard
Model name
Last created
Records
Blogs
about 7 hours ago
Users
about 23 hours ago


Sourcebit Admin
NAVIGATION
Blogs
Users
List of Users
Dashboard / Users
List
Add new
Export
Filter
Refresh
Id
Email
Reset password sent at
2
2@gmail.com
-
1
1@gmail.com
-
2 users



Admin-Panel hinzufügen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/8128/admin-panel-hinzufugen

Kapitel 16: Aktive Jobs

Examples

Einführung

Active Job ist seit Rails 4.2 verfügbar und stellt ein Framework dar, um Jobs zu deklarieren und für verschiedene Backends in der Warteschlange auszuführen. Wiederkehrende oder pünktliche Aufgaben, die nicht blockiert werden und parallel ausgeführt werden können, sind gute Anwendungsfälle für aktive Jobs.

Musterjob

```
class UserUnsubscribeJob < ApplicationJob
  queue_as :default

  def perform(user)
    # this will happen later
    user.unsubscribe
  end
end
```

Aktiven Job über den Generator erstellen

```
$ rails g job user_unsubscribe
```

Aktive Jobs online lesen: https://riptutorial.com/de/ruby-on-rails/topic/8033/aktive-jobs

Kapitel 17: Aktive Modell-Serialisierer

Einführung

ActiveModelSerializers, oder kurz AMS, bringen Konvention über Konfiguration in Ihre JSON-Generation ein. ActiveModelSerializers arbeiten mit zwei Komponenten: Serialisierern und Adaptern. Serialisierer beschreiben, welche Attribute und Beziehungen serialisiert werden sollen. Adapter beschreiben, wie Attribute und Beziehungen serialisiert werden sollen.

Examples

Verwenden eines Serializers

```
class SomeSerializer < ActiveModel::Serializer
  attribute :title, key: :name
  attributes :body
end
```

Aktive Modell-Serialisierer online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9000/aktive-modell-serialisierer

Kapitel 18: Aktiver Rekord

Examples

Modell manuell erstellen

Die Verwendung von Gerüsten ist schnell und einfach, wenn Sie noch nicht mit Rails vertraut sind oder wenn Sie eine neue Anwendung erstellen. Später kann es jedoch nützlich sein, dies nur auf eigene Faust zu tun, um zu vermeiden, dass Sie den durch Gerüst erzeugten Code durchlaufen müssen, um sie zu reduzieren (Entfernen Sie nicht verwendete Teile usw.).

Das Erstellen eines Modells kann so einfach sein wie das Erstellen einer Datei unter `app/models` .

Das einfachste Modell in `ActiveRecord` ist eine Klasse, die `ActiveRecord::Base` .

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

Modelldateien werden in `app/models/` gespeichert und der Dateiname entspricht dem einzigen Namen der Klasse:

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

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

Die Klasse übernimmt alle ActiveRecord-Funktionen: Abfragemethoden, Validierungen, Rückrufe usw.

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

Anmerkung: Stellen Sie sicher, dass die Tabelle für das entsprechende Modell vorhanden ist. Wenn nicht, können Sie die Tabelle erstellen, indem Sie eine Migration erstellen

Mit dem folgenden Befehl können Sie ein Modell und seine Migration nach Terminal generieren

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

und kann dem Modell auch einen Fremdschlüssel (Beziehung) mit folgendem Befehl zuweisen

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

Modell über Generator erstellen

Ruby on Rails bietet einen `model` , mit dem Sie ActiveRecord-Modelle erstellen können.

Verwenden Sie einfach `rails generate model` und geben Sie den Modellnamen an.

```
$ rails g model user
```

Neben der Modelldatei in `app/models` der Generator außerdem:

- den Test in `test/models/user_test.rb`
- die Fixtures in `test/fixtures/users.yml`
- die Datenbank Migration in `db/migrate/XXX_create_users.rb`

Sie können beim Generieren auch einige Felder für das Modell generieren.

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

Dadurch werden die Spalten email, sign_in_count und birthday in Ihrer Datenbank mit den entsprechenden Typen erstellt.

Eine Migration erstellen

Felder in vorhandenen Tabellen hinzufügen / entfernen

Erstellen Sie eine Migration, indem Sie Folgendes ausführen:

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

Dies erzeugt eine Migration , die eine fügt `title` zu einer `categories` - Tabelle:

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

Auf ähnliche Weise können Sie eine Migration zum Entfernen einer Spalte `rails generate migration RemoveTitleFromCategories title:string` : `rails generate migration RemoveTitleFromCategories title:string`

Dies erzeugt eine Migration , die einen entfernt `title` Spalt aus den `categories` Tabelle:

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

Während streng genommen Angabe **Typ** (`:string` in diesem Fall) zum Entfernen eine Spalte

nicht notwendig ist, **ist es hilfreich,** da es die notwendigen Informationen zum **Walzen zurück** zur Verfügung stellt.

Erstellen Sie eine Tabelle

Erstellen Sie eine Migration, indem Sie Folgendes ausführen:

```
rails g CreateUsers name bio
```

Rails erkennt die Absicht, eine Tabelle mit dem Präfix " `Create` , der Rest des Migrationsnamens wird als Tabellenname verwendet. Das gegebene Beispiel generiert Folgendes:

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

Beachten Sie, dass die Schaffung Befehl nicht Spaltentypen angeben und das Standard - `string` verwendet wurde.

Erstellen Sie eine Join-Tabelle

Erstellen Sie eine Migration, indem Sie Folgendes ausführen:

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

Rails erkennt die Absicht, eine Join-Tabelle zu erstellen, indem er `JoinTable` im Migrationsnamen findet. Alles andere wird aus den Namen der Felder bestimmt, die Sie nach dem Namen angeben.

```
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
```

Kommentar- der erforderlichen `index` - Anweisungen und den Rest löschen.

Vorrang

Beachten Sie, dass der Beispiel-Migrationsname `CreateJoinTableParticipation` der Regel für die Tabellenerstellung entspricht: Er hat das Präfix `Create` . Es wurde jedoch keine einfache `create_table` generiert. Dies liegt daran, dass der Migrationsgenerator (Quellcode) eine **erste Übereinstimmung** der folgenden Liste verwendet:

- `(Add|Remove)<ignored>(To|From)<table_name>`
- `<ignored>JoinTable<ignored>`
- `Create<table_name>`

Einführung in Rückrufe

Ein Rückruf ist eine Methode, die zu bestimmten Zeitpunkten des Lebenszyklus eines Objekts aufgerufen wird (direkt vor oder nach dem Erstellen, Löschen, Aktualisieren, Überprüfen, Speichern oder Laden aus der Datenbank).

Angenommen, Sie haben eine Auflistung, die innerhalb von 30 Tagen nach der Erstellung abläuft.

Eine Möglichkeit, dies zu tun, ist wie folgt:

```
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
```

Alle verfügbaren Methoden für Callbacks lauten in der Reihenfolge, in der sie während des Betriebs jedes Objekts aufgerufen werden:

Ein Objekt erstellen

- before_validation
- after_validation
- before_save
- around_save
- vor_erstellen
- around_create
- after_create
- after_save
- after_commit / after_rollback

Objekt aktualisieren

- before_validation
- after_validation

- before_save
- around_save
- vor_update
- around_update
- nach dem Update
- after_save
- after_commit / after_rollback

Objekt zerstören

- vor_destroy
- around_destroy
- after_destroy
- after_commit / after_rollback

*HINWEIS **:** after_save wird sowohl beim Erstellen als auch beim Update ausgeführt, jedoch immer nach den spezifischeren Rückrufen after_create und after_update, unabhängig von der Reihenfolge, in der die Makroaufrufe ausgeführt wurden.*

Erstellen Sie eine Join-Tabelle mithilfe von Migrationen

Besonders nützlich für `has_and_belongs_to_many` Beziehung, können Sie manuell eine Join - Tabelle mit der erstellen `create_table` Methode. Angenommen, Sie verfügen über zwei Modelle, `Tags` und `Proyects` , und `Proyects` diese mithilfe einer `has_and_belongs_to_many` Beziehung `has_and_belongs_to_many` . Sie benötigen eine Join-Tabelle, um Instanzen beider Klassen zu verknüpfen.

```
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
```

Der tatsächliche Name der Tabelle muss dieser Konvention folgen: Das Modell, das dem anderen alphabetisch vorangeht, muss zuerst gehen. **P** roject steht vor **T** ags, so dass der Name der Tabelle projects_tags lautet.

Da diese Tabelle auch dazu dient, die Zuordnung zwischen den Instanzen von zwei Modellen zu routen, ist die tatsächliche ID jedes Datensatzes in dieser Tabelle nicht erforderlich. Sie geben dies an, indem Sie `id: false`

Wie in Rails üblich, muss der Tabellenname schließlich die zusammengesetzte Pluralform der einzelnen Modelle sein, aber die Spalte der Tabelle muss in Singularform sein.

Manuelles Testen Ihrer Modelle

Das Testen Ihrer Active Record-Modelle über Ihre Befehlszeilenschnittstelle ist einfach. Navigieren Sie zu dem App - Verzeichnis in Ihrem Terminal und geben Sie in `rails console` die Rails - Konsole zu starten. Von hier aus können Sie aktive Record-Methoden in Ihrer Datenbank ausführen.

Wenn Sie beispielsweise ein Datenbankschema mit einer Users-Tabelle mit einem `name:string` und `email:string` , könnten Sie Folgendes ausführen:

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

Um diesen Datensatz anzuzeigen, können Sie Folgendes ausführen:

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

Wenn dies Ihre erste oder einzige Aufnahme ist, können Sie einfach die erste Aufnahme abrufen, indem Sie Folgendes ausführen:

```
User.first
```

Verwenden einer Modellinstanz zum Aktualisieren einer Zeile

Angenommen, Sie haben ein `User`

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

Jetzt können Sie zum Aktualisieren des `first_name` und `last_name` eines Benutzers mit der `id = 1` den folgenden Code schreiben.

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

Beim Aufruf von `update` wird versucht, die angegebenen Attribute in einer einzelnen Transaktion zu aktualisieren. Bei Erfolg wird `true` andernfalls `false` .

Aktiver Rekord online lesen: https://riptutorial.com/de/ruby-on-rails/topic/828/aktiver-rekord

Kapitel 19: Ändern Sie eine Standardumgebung für Rails-Anwendungen

Einführung

In diesem Abschnitt wird beschrieben, wie die Umgebung geändert werden kann. Wenn jemand `rails s` startet er nicht in der Entwicklung, sondern in der gewünschten Umgebung.

Examples

Läuft auf einem lokalen Rechner

Normalerweise, wenn die Schienenumgebung durch Tippen ausgeführt wird. Dies führt nur die Standardumgebung aus, die normalerweise in der `development`

```
rails s
```

Die spezifische Umgebung kann beispielsweise mit dem Flag `-e` ausgewählt werden:

```
rails s -e test
```

Welches wird die Testumgebung ausführen.

Die Standardumgebung kann in Terminal `~/.bashrc` werden, indem Sie die Datei `~/.bashrc` und die folgende Zeile hinzufügen:

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

Läuft auf einem Server

Wenn Sie auf einem Remote-Server laufen, der Passenger verwendet, ändern Sie die Datei apache.conf in die Umgebung, die Sie verwenden möchten. In diesem Fall sehen Sie `RailsEnv production` .

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

Ändern Sie eine Standardumgebung für Rails-Anwendungen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9915/andern-sie-eine-standardumgebung-fur-rails-anwendungen

Kapitel 20: Ansichten

Examples

Teilstücke

Teilvorlagen (Partials) sind eine Möglichkeit, den Rendering-Prozess in überschaubarere Abschnitte zu unterteilen. Mit Partials können Sie Code-Teile aus Ihren Vorlagen extrahieren, um Dateien zu trennen und sie auch in Ihren Vorlagen wiederzuverwenden.

Um einen Teil *zu erstellen,* erstellen Sie eine neue Datei, die mit einem Unterstrich beginnt: `_form.html.erb`

Verwenden Sie zum *Rendern* eines Teils als Teil einer Ansicht die Render-Methode in der Ansicht: `<%= render "form" %>`

- Beachten Sie, dass der Unterstrich beim Rendern ausgelassen wird
- Ein Teil muss mit seinem Pfad gerendert werden, wenn es sich in einem anderen Ordner befindet

Um eine Variable in den Teil als lokaler Variable, verwenden Sie diese Notation *übergeben:*

```
<%= render :partial => 'form', locals: { post: @post } %>
```

Partials sind auch nützlich, wenn Sie genau denselben Code *wiederverwenden* müssen (**DRY-Philosophie**).

Um beispielsweise `<head>` -Code wiederzuverwenden, erstellen Sie einen partiellen Namen namens `_html_header.html.erb` , geben Sie Ihren `<head>` -Code ein, der wiederverwendet werden soll, und rendern Sie den partiellen Code bei Bedarf durch: `<%= render 'html_header' %>` .

Objektpartials

Objekte, die auf `to_partial_path` antworten, können auch gerendert werden, wie in `<%= render @post %>` . Standardmäßig handelt es sich bei ActiveRecord-Modellen so etwas wie `posts/post @post` tatsächlich rendern, werden die Dateien `views/posts/_post.html.erb` gerendert.

Ein lokal benannter `post` wird automatisch zugewiesen. Am Ende ist `<%= render @post %>` eine kurze Hand für `<%= render 'posts/post', post: @post %>` .

`to_partial_path` können auch Sammlungen von Objekten bereitgestellt werden, die auf `to_partial_path` antworten, beispielsweise `<%= render @posts %>` . Jedes Element wird nacheinander gerendert.

Globale Partials

Um ein globales Partial zu erstellen, das überall verwendet werden kann, ohne auf den genauen Pfad zu verweisen, muss das Partial in den `views/application` . Das vorige Beispiel wurde unten geändert, um diese Funktion zu veranschaulichen.

Dies ist beispielsweise ein Pfad zu einem globalen Teil von `app/views/application/_html_header.html.erb`:

Verwenden Sie `<%= render 'html_header' %>` um dieses globale Teil überall `<%= render 'html_header' %>`

AssetTagHelper

In den meisten Fällen, in denen Sie integrierte Helfer verwenden möchten, lassen Sie Assets (css / js / images) automatisch und korrekt verknüpfen. (Offizielle Dokumentation)

Bildhelfer

Bildpfad

Dadurch wird der Pfad zu einem Bildasset in `app/assets/images` .

```
image_path("edit.png") # => /assets/edit.png
```

Bild URL

Dadurch wird die vollständige URL zu einem Bildasset in `app/assets/images` .

```
image_url("edit.png") # => http://www.example.com/assets/edit.png
```

image_tag

Verwenden Sie diesen Helfer, wenn Sie einen `<img src="" />` -Tag in die Quellgruppe aufnehmen möchten.

```
image_tag("icon.png") # => <img src="/assets/icon.png" alt="Icon" />
```

JavaScript-Helfer

javascript_include_tag

Wenn Sie eine JavaScript-Datei in Ihre Ansicht aufnehmen möchten.

```
javascript_include_tag "application" # => <script src="/assets/application.js"></script>
```

Javascript_Pfad

Dies gibt den Pfad Ihrer JavaScript-Datei zurück.

```
javascript_path "application" # => /assets/application.js
```

javascript_url

Dies gibt die vollständige URL Ihrer JavaScript-Datei zurück.

```
javascript_url "application" # => http://www.example.com/assets/application.js
```

Stylesheet-Helfer

stylesheet_link_tag

Wenn Sie eine CSS-Datei in Ihre Ansicht aufnehmen möchten.

```
stylesheet_link_tag "application" # => <link href="/assets/application.css" media="screen"
rel="stylesheet" />
```

stylesheet_path

Dies gibt den Pfad Ihres Stylesheet-Assets zurück.

```
stylesheet_path "application" # => /assets/application.css
```

stylesheet_url

Dies gibt die vollständige URL Ihres Stylesheet-Assets zurück.

```
stylesheet_url "application" # => http://www.example.com/assets/application.css
```

Verwendungsbeispiel

Beim Erstellen einer neuen Schienen-App stehen Ihnen automatisch zwei dieser Helfer in `app/views/layouts/application.html.erb`

```
<%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track': 'reload' %>
<%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
```

Dies gibt aus:

```
// 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>
```

Struktur

Als Rails die M **V** C Muster folgt `Views` sind , wo Ihre „Vorlagen“ für Ihre Handlungen sind.

Nehmen wir an, Sie haben einen Controller `articles_controller.rb` . Für diesen Controller haben Sie einen Ordner in Ansichten, der als `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
    |
    '-- [...]
```

Diese Struktur ermöglicht es Ihnen, für jeden Controller einen Ordner zu erstellen. Beim Aufruf einer Aktion in Ihrem Controller wird die entsprechende Ansicht automatisch gerendert.

```
// articles_controller.rb
class ArticlesController < ActionController::Base
  def show
  end
end

// show.html.erb
<h1>My show view</h1>
```

Ersetzen Sie HTML-Code in Ansichten

Wenn Sie schon einmal den HTML-Inhalt ermitteln wollten, der zur Laufzeit auf einer Seite gedruckt werden soll, bietet rail eine sehr gute Lösung dafür. Es hat etwas namens **content_for** , mit dem wir einen Block an eine Schienenansicht übergeben können. Bitte überprüfen Sie das Beispiel unten.

Deklarieren Sie content_for

```
<div>
  <%= yield :header %>
</div>

<% content_for :header do %>
 <ul>
   <li>Line Item 1</li>
   <li>Line Item 2</li>
 </ul>
<% end %>
```

HAML - eine alternative Möglichkeit, Ihre Ansichten zu verwenden

HAML (HTML Abstraction Markup Language) ist eine schöne und elegante Möglichkeit, den HTML-Code Ihrer Ansichten zu beschreiben und zu gestalten. Anstatt Tags zu öffnen und zu schließen, verwendet HAML die Einrückung für die Struktur Ihrer Seiten. Wenn etwas in ein anderes Element eingefügt werden soll, wird es einfach mit einem Tabulator eingerückt. In HAML sind Registerkarten und Leerzeichen wichtig. Stellen Sie daher sicher, dass Sie immer dieselbe Anzahl von Registerkarten verwenden.

Beispiele:

```
#myview.html.erb
<h1><%= @the_title %></h1>
<p>This is my form</p>
<%= render "form" %>
```

Und in HAML:

```
#myview.html.haml
%h1= @the_title
%p
    This is my form
= render 'form'
```

Sie sehen, die Struktur des Layouts ist viel klarer als die Verwendung von HTML und ERB.

Installation

Installieren Sie einfach den Edelstein mit

```
gem install haml
```

und füge den Edelstein zum Gemfile hinzu

```
gem "haml"
```

Für HAML anstelle von HTML / ERB verwenden, ersetzen nur die Dateierweiterungen Ihrer Ansichten von `something.html.erb` `something.html.haml` .

Schnelle Tipps

Gemeinsame Elemente wie divs können kurz geschrieben werden

HTML

```
<div class="myclass">My Text</div>
```

HAML

```
%div.myclass
```

HAML, Abkürzung

```
.myclass
```

Attribute

HTML

```
<p class="myclass" id="myid">My paragraph</p>
```

HAML

```
%p{:class => "myclass", :id => "myid"} My paragraph
```

Ruby-Code einfügen

Sie können Ruby-Code mit den = und - Zeichen einfügen.

```
= link_to "Home", home_path
```

Code, der mit **=** beginnt, wird ausgeführt und in das Dokument eingebettet.

Code, der mit **-** beginnt, wird ausgeführt, aber nicht in das Dokument eingefügt.

Vollständige Dokumentation

HAML ist sehr einfach zu beginnen, aber auch sehr komplex. Ich empfehle Ihnen, die Dokumentation zu lesen .

Ansichten online lesen: https://riptutorial.com/de/ruby-on-rails/topic/850/ansichten

Kapitel 21: Asset-Pipeline

Einführung

Die Asset-Pipeline bietet ein Framework zum Verketten und Minimieren oder Komprimieren von JavaScript- und CSS-Assets. Es bietet außerdem die Möglichkeit, diese Assets in anderen Sprachen und Vorprozessoren wie CoffeeScript, Sass und ERB zu schreiben. Dadurch können Assets in Ihrer Anwendung automatisch mit Assets anderer Edelsteine kombiniert werden. Zum Beispiel enthält jquery-rail eine Kopie von jquery.js und aktiviert AJAX-Funktionen in Rails.

Examples

Rechenaufgaben

Standardmäßig werden `sprockets-rails` mit den folgenden Rake-Aufgaben ausgeliefert:

- `assets:clean[keep]` : Entferne alte kompilierte Assets
- `assets:clobber` : Kompilierte Assets entfernen
- `assets:environment` Laden von Assets
- `assets:precompile` : Kompiliert alle in `config.assets.precompile` genannten `config.assets.precompile`

Manifestdateien und -richtlinien

Im `assets` Initalizer (`config/initializers/assets.rb`) sind einige Dateien explizit als vorkompiliert definiert.

```
# 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 )
```

In diesem Beispiel sind `application.coffee` und `application.scss` sogenannte 'Manifest-Dateien'. Diese Dateien sollten verwendet werden, um andere JavaScript- oder CSS-Elemente einzuschließen. Der folgende Befehl ist verfügbar:

- `require <path>` : Die `require` Richtlinie funktioniert ähnlich wie Rubys eigene `require` . Es bietet eine Möglichkeit, eine Abhängigkeit von einer Datei in Ihrem Pfad zu erklären, und stellt sicher, dass diese nur einmal vor der Quelldatei geladen wird.
- `require_directory <path>` : erfordert alle Dateien in einem einzigen Verzeichnis. Es ist dem `path/*` ähnlich, da er nicht auf verschachtelte Verzeichnisse folgt.
- `require_tree <path>` : erfordert alle geschachtelten Dateien in einem Verzeichnis. Sein glob-Äquivalent ist `path/**/*` .
- `require_self` : bewirkt , dass der Körper der aktuellen Datei eingefügt werden , bevor eine nachfolgende `require` Richtlinien. Nützlich in CSS-Dateien, in denen die Indexdatei häufig globale Stile enthält, die definiert werden müssen, bevor andere Abhängigkeiten geladen

werden.

- `stub <path>` : Entfernen Sie eine Datei aus der Aufnahme
- `depend_on <path>` : Ermöglicht das `depend_on <path>` einer Abhängigkeit von einer Datei, ohne sie `depend_on <path>` . Dies wird zum Zwischenspeichern verwendet. Durch Änderungen an der Abhängigkeitsdatei wird der Cache der Quelldatei ungültig.

Eine `application.scss` Datei könnte folgendermaßen aussehen:

```
/*
 *= require bootstrap
 *= require_directory .
 *= require_self
 */
```

Ein anderes Beispiel ist die Datei `application.coffee` . Hier mit `jquery` und `Turbolinks` :

```
#= require jquery2
#= require jquery_ujs
#= require turbolinks
#= require_tree .
```

Wenn Sie nicht CoffeeScript verwenden, sondern nur JavaScript, lautet die Syntax:

```
//= require jquery2
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

Grundlegende Verwendung

Die Asset-Pipeline kann auf zwei Arten verwendet werden:

1. Wenn Sie einen Server im Entwicklungsmodus ausführen, werden Ihre Assets automatisch im Voraus vorverarbeitet und vorbereitet.

2. Im Produktionsmodus verwenden Sie sie wahrscheinlich zur Vorverarbeitung, Versionierung sowie Komprimierung und Kompilierung Ihrer Assets. Führen Sie dazu den folgenden Befehl aus:

   ```
   bundle exec rake assets:precompile
   ```

Asset-Pipeline online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3386/asset-pipeline

Kapitel 22: Aufbau

Examples

Benutzerdefinierte Konfiguration

Erstellen Sie eine `YAML` Datei im Verzeichnis `config/` , zum Beispiel: `config/neo4j.yml`

Der Inhalt von `neo4j.yml` kann wie `neo4j.yml` aussehen (zur Vereinfachung wird der `default` für alle Umgebungen verwendet):

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

development:
  <<: *default

test:
  <<: *default

production:
  <<: *default
```

in `config/application.rb` :

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

Nun ist Ihre benutzerdefinierte Konfiguration wie folgt zugänglich:

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

Mehr Info

Das offizielle API-Dokument von Rails beschreibt die Methode `config_for` :

> Bequemes Laden von config / foo.yml für die aktuelle Rails-Umgebung.

Wenn Sie keine `yaml` Datei verwenden `yaml`

Sie können Ihren eigenen Code über das Rails-Konfigurationsobjekt mit benutzerdefinierter Konfiguration unter der Eigenschaft `config.x` konfigurieren.

Beispiel

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

Diese Konfigurationspunkte stehen dann über das Konfigurationsobjekt zur Verfügung:

```
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
```

Aufbau online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2558/aufbau

Kapitel 23: Aufbau

Examples

Umgebungen in Schienen

Konfigurationsdateien für Rails finden Sie in `config/environments/` . In der Standardeinstellung besitzt rail 3 Umgebungen: `development` , `production` und `test` . Indem Sie jede Datei bearbeiten, bearbeiten Sie nur die Konfiguration für diese Umgebung.

Rails verfügt auch über eine Konfigurationsdatei in `config/application.rb` . Dies ist eine allgemeine Konfigurationsdatei, da alle hier definierten Einstellungen von der in jeder Umgebung angegebenen Konfiguration überschrieben werden.

Sie können Konfigurationsoptionen innerhalb der `Rails.application.configure do` Blockierungs- und Konfigurationsoptionen hinzufügen oder ändern, wobei die Konfigurationsoptionen mit `config`.

Datenbankkonfiguration

Die Datenbankkonfiguration eines Schienenprojekts liegt in einer Datei `config/database.yml` . Wenn Sie ein Projekt mit dem `rails new` Befehl schienen erstellen und keine Datenbank-Engine angeben, die verwendet werden soll, verwendet `sqlite` als Standarddatenbank. Eine typische `database.yml` Datei mit Standardkonfiguration sieht folgendermaßen aus:

```
# 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
```

Wenn Sie die Standarddatenbank beim Erstellen eines neuen Projekts ändern möchten, können

Sie die Datenbank angeben: `rails new hello_world --database=mysql`

Rails Allgemeine Konfiguration

Die folgenden Konfigurationsoptionen sollten für ein `Rails::Railtie` Objekt `Rails::Railtie` werden

- **config.after_initialize** : **Übernimmt** einen Block, der ausgeführt wird, nachdem **rail** die Anwendung initialisiert hat.
- **config.asset_host** : Hiermit wird der Host für die Assets festgelegt. Dies ist nützlich, wenn Sie ein *Content Delivery Network verwenden* . Dies ist eine Abkürzung für `config.action_controller.asset_host`
- **config.autoload_once_paths** : Diese Option akzeptiert ein Array von Pfaden, in denen Rails Konstanten **automatisch lädt** . Der Standardwert ist ein leeres Array
- **config.autoload_paths** : Dies akzeptiert ein Array von Pfaden, bei denen Rails Konstanten **automatisch lädt** . Standardmäßig werden alle Verzeichnisse unter `app`
- **config.cache_classes** : Bestimmt, ob Klassen und Module bei jeder Anforderung erneut geladen werden sollen. Im Entwicklungsmodus ist dies standardmäßig auf " `false` und im Produktions- und Testmodus auf " `true`
- **config.action_view.cache_template_loading** : Hiermit wird festgelegt, ob Vorlagen für jede Anforderung erneut geladen werden sollen. Die `config.cache_classes`
- **config.beginning_of_week** : Hier wird der Standardbeginn der Woche festgelegt. Es erfordert ein gültiges Wochentagesymbol (`:monday`).
- **config.cache_store** : Wählen Sie den zu verwendenden Cache-Speicher aus. Zu den Optionen gehören `:file_store` `:memory_store` , `mem_cache_store` oder `null_store` .
- **config.colorize_logging** : Dies steuert, ob die Protokollinformationen **eingefärbt werden**
- **config.eager_load** : Eager lädt alle registrierten
- **config.encoding** : Gibt die Anwendungskodierung an. Der Standardwert ist `UTF-8`
- **config.log_level** : Legt die Ausführlichkeit des Rails Logger fest. Der Standardwert ist `:debug` in allen Umgebungen.
- **config.middleware** : **Hiermit** konfigurieren Sie die Middleware der Anwendung
- **config.time_zone** : Hiermit wird die Standardzeitzone der Anwendung festgelegt.

Assets konfigurieren

Die folgenden Konfigurationsoptionen können zur Konfiguration von Assets verwendet werden

- **config.assets.enabled** : Bestimmt, ob die Asset-Pipeline aktiviert ist. Dies ist standardmäßig true
- **config.assets.raise_runtime_errors** : Dies ermöglicht die Laufzeitfehlerprüfung. Es ist nützlich für den `development mode`
- **config.assets.compress** : Ermöglicht die Komprimierung von Assets. Im Produktionsmodus ist dies standardmäßig true
- **config.assets.js_compressor** : Gibt an, welcher JS-Kompressor verwendet werden soll. Zu den Optionen gehören `:closure` `:uglifier` und `:yui`
- **config.assets.paths** : Gibt an, welche Pfade nach Assets durchsucht werden sollen.
- **config.assets.precompile** : Hier können Sie zusätzliche Assets auswählen, die vorkompiliert werden sollen, wenn `rake assets:precompile` wird ausgeführt

- **config.assets.digest** : Diese Option ermöglicht die Verwendung von `MD-5` Fingerabdrücken in den Asset-Namen. Im Entwicklungsmodus ist der Standardwert "true"
- **config.assets.compile** : Schaltet die Live-Erstellung von `Sprockets` im Produktionsmodus um

Generatoren konfigurieren

Rails können Sie konfigurieren , welche Generatoren verwendet werden , wenn `rails generate` Befehle. Diese Methode, `config.generators` nimmt einen Block

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

Hier sind einige Optionen

Möglichkeit	Beschreibung	Standard
Vermögenswerte	Erzeugt Assets beim Generieren des Gerüsts	wahr
force_plural	Erlaubt mehrfache Modellnamen	falsch
Helfer	Legt fest, ob Helfer generiert werden sollen	wahr
integration_tool	Integrationswerkzeug angeben	`test_unit`
javascript_engine	Konfiguriert die JS-Engine	`:js`
Ressourcenroute	Erzeugt eine Ressourcenroute	wahr
stylesheet_engine	Konfiguriert die Stylesheet-Engine	`:cs`
scaffold_stylesheet	Erzeugt CSS beim Scaffolding	wahr
test_framework	Geben Sie das Test-Framework an	`Minitest`
template_engine	Konfiguriert die Template Engine	`:erb`

Aufbau online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2841/aufbau

Kapitel 24: Authentifizieren Sie die API mit Devise

Einführung

Devise ist eine Authentifizierungslösung für Rails. Bevor ich weiter gehe, möchte ich noch eine kurze Anmerkung zur API hinzufügen. Daher behandelt die API keine Sitzungen (ist zustandslos), dh eine, die nach Ihrer Anforderung eine Antwort liefert, und erfordert dann keine weitere Aufmerksamkeit. Das bedeutet, dass kein früherer oder zukünftiger Status erforderlich ist, damit das System funktioniert, wenn wir eine Anfrage an den Server stellen Authentifizierungsdetails mit allen APIs übergeben und sollte Devise anweisen, Authentifizierungsdetails nicht zu speichern.

Examples

Fertig machen

Zuerst werden wir ein Schienenprojekt und ein Setup-Gerät erstellen

Erstellen Sie eine Schienenanwendung

```
rails new devise_example
```

füge jetzt der Edelsteinliste Gerät hinzu

> Sie finden eine Datei mit dem Namen 'Gemfile' im Stammverzeichnis des rail-Projekts

Führen Sie dann die `bundle install`

Als Nächstes müssen Sie den Generator ausführen:

```
rails generate devise:install
```

Nun können Sie auf der Konsole einige Anweisungen finden, die Sie befolgen.

Erstellen Sie das Modellmodell

```
rails generate devise MODEL
```

Führen Sie dann `rake db:migrate`

Weitere Informationen finden Sie unter: Devise Gem

Authentifizierungs-Token

Das Authentifizierungstoken wird verwendet, um einen Benutzer mit einem eindeutigen Token zu authentifizieren. Bevor wir mit der Logik fortfahren, müssen wir einem Devise-Modell das Feld `auth_token` hinzufügen

Daher,

```
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
```

Führen Sie dann `rake db:migrate`

Jetzt sind wir `auth_token` , die Authentifizierung mit `auth_token`

In `app/controllers/application_controllers.rb`

Zuerst diese Linie dazu

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

Dies hilft Rails-Anwendungen, sowohl mit HTML als auch mit Json zu reagieren

Dann

```
protect_from_forgery with: :null
```

wird dies ändern `:null` da wir uns nicht mit Sitzungen beschäftigen.

Jetzt fügen wir in application_controller eine Authentifizierungsmethode hinzu

Standardmäßig verwendet Devise email als eindeutiges Feld. Wir können auch benutzerdefinierte Felder verwenden. In diesem Fall authentifizieren wir uns mit user_email und 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
```

> Hinweis: Der obige Code basiert rein auf Ihrer Logik. Ich versuche nur, das Arbeitsbeispiel zu erklären

In Zeile 6 im obigen Code können Sie sehen, dass ich `store: false` wodurch verhindert werden kann, dass für jede Anforderung eine Sitzung erstellt wird

Authentifizieren Sie die API mit Devise online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9787/authentifizieren-sie-die-api-mit-devise

Kapitel 25: Autorisierung mit CanCan

Einführung

CanCan ist eine einfache Berechtigungsstrategie für Rails, die von Benutzerrollen entkoppelt ist. Alle Berechtigungen werden an einem einzigen Ort gespeichert.

Bemerkungen

Bevor Sie CanCan verwenden, vergessen Sie nicht, Benutzer entweder nach einem erfundenen Juwel oder manuell zu erstellen. Um die maximale Funktionalität von CanCan zu erhalten, erstellen Sie einen Admin-Benutzer.

Examples

Erste Schritte mit CanCan

CanCan ist eine beliebte Berechtigungsbibliothek für Ruby on Rails, die den Benutzerzugriff auf bestimmte Ressourcen einschränkt. Das neueste Juwel (CanCanCan) ist eine Fortsetzung des toten Projekts CanCan .

Berechtigungen werden in der `Ability` Klasse definiert und können von Controllern, Ansichten, Helfern oder anderen Stellen im Code verwendet werden.

Um einer App Autorisierungsunterstützung hinzuzufügen, fügen Sie der Gemfile den CanCanCan-Edelstein `Gemfile` :

```
gem 'cancancan'
```

Dann definieren Sie die Fähigkeitsklasse:

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

  def initialize(user)
  end
end
```

Überprüfen Sie dann die Autorisierung mit `load_and_authorize_resource` , um autorisierte Modelle in den Controller zu laden:

```
class ArticlesController < ApplicationController
  load_and_authorize_resource

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

```
  end
end
```

`authorize!` um die Berechtigung zu prüfen oder eine Ausnahme zu erheben

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

`can?` um zu prüfen, ob ein Objekt gegen eine bestimmte Aktion in den Steuerungen, Ansichten oder Helfern autorisiert ist

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

Hinweis: Dies setzt voraus, dass der signierte Benutzer von der Methode `current_user` bereitgestellt wird.

Fähigkeiten definieren

Fähigkeiten werden in der `Ability` mit den Methoden `can` und `cannot` . Betrachten Sie das folgende kommentierte Beispiel als Basisreferenz:

```
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
```

Umgang mit einer großen Anzahl von Fähigkeiten

Sobald die Anzahl der Definitionen der Fähigkeiten zunimmt, wird es immer schwieriger, mit der

Fähigkeitsdatei umzugehen.

Die erste Strategie, um mit diesem Problem umzugehen, besteht darin, Fähigkeiten in sinnvolle Methoden umzuwandeln, wie in diesem Beispiel gezeigt:

```
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
```

Sobald diese Klasse groß genug ist, können Sie versuchen, sie in verschiedene Klassen aufzuteilen, um die verschiedenen Verantwortlichkeiten wie folgt zu handhaben:

```
# 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
```

und definieren Sie dann diese Klassen als:

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

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

und so weiter mit `Abilities::Authenticated` , `Abilities::Admin` oder einem anderen.

Testen Sie schnell eine Fähigkeit

Wenn Sie schnell testen möchten, ob eine Berechtigungsklasse die richtigen Berechtigungen erteilt, können Sie eine Fähigkeit in der Konsole oder in einem anderen Kontext mit geladener Schienenumgebung initialisieren.

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

Weitere Informationen: https://github.com/ryanb/cancan/wiki/Testing-Abilities

Autorisierung mit CanCan online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3021/autorisierung-mit-cancan

Kapitel 26: Benutzerauthentifizierung in Rails

Einführung

Devise ist ein sehr mächtiges Juwel, mit dem Sie sich direkt nach der Installation anmelden, an- und abmelden können. Darüber hinaus kann der Benutzer seinen Anwendungen Authentifizierungen und Einschränkungen hinzufügen. Devise enthält auch eigene Ansichten, wenn der Benutzer dies verwenden möchte. Ein Benutzer kann die Anmelde- und Anmeldeformulare je nach Bedarf und Anforderung anpassen. Es wird darauf hingewiesen, dass Devise empfiehlt, dass Sie Ihr eigenes Login implementieren, wenn Sie noch nicht mit Rail vertraut sind.

Bemerkungen

Zum Zeitpunkt der Generierung von devise-Konfigurationen mithilfe von schienen `rails generate devise:install` , listet devise eine Reihe von Anweisungen auf dem Terminal auf, die zu befolgen sind.

Wenn Sie bereits über ein `USER` Modell verfügen, können Sie mit diesem Befehlsschienengenerator `rails generate devise USER` erforderlichen Spalten an Ihr vorhandenes `USER` Modell anhängen.

Verwenden Sie diese `before_action :authenticate_user!` oben auf dem Controller, um zu überprüfen, ob der `user` angemeldet ist oder nicht. Andernfalls werden sie auf die Anmeldeseite umgeleitet.

Examples

Authentifizierung mit Devise

Edelstein zum Gemfile hinzufügen:

```
gem 'devise'
```

Führen Sie dann den Befehl zum `bundle install` Pakets aus.

Verwenden Sie den Befehl `$ rails generate devise:install` , um die erforderliche Konfigurationsdatei zu generieren.

Richten Sie die Standard-URL-Optionen für den Devise-Mailer in jeder Umgebung ein. Fügen Sie in der Entwicklungsumgebung folgende Zeile hinzu:

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

in Ihre `config/environments/development.rb`

In ähnlicher Weise bearbeiten Sie diese Datei in der Datei `config/environments/production.rb` und

fügen Sie hinzu

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

Erstellen Sie anschließend ein Modell mit folgendem `$ rails generate devise USER` : `$ rails generate devise USER` Wobei `USER` der Klassenname ist, für den Sie die Authentifizierung implementieren möchten.

Zum Schluss: `rake db:migrate` und du bist fertig.

Benutzerdefinierte Ansichten

Wenn Sie Ihre Ansichten konfigurieren müssen, können Sie den Generatoren für die `rails generate devise:views` , der alle Ansichten in Ihre Anwendung kopiert. Sie können sie dann wie gewünscht bearbeiten.

Wenn sich in Ihrer Anwendung mehr als ein Devise-Modell befindet (z. B. Benutzer und Administrator), werden Sie feststellen, dass Devise für alle Modelle die gleichen Ansichten verwendet. Devise bietet eine einfache Möglichkeit, Ansichten anzupassen. `config.scoped_views = true` Sie `config.scoped_views = true` in der Datei `config/initializers/devise.rb` .

Sie können den Generator auch verwenden, um Bereichsansichten zu erstellen: `rails generate devise:views users`

Wenn Sie nur einige Ansichten erstellen möchten, beispielsweise die für das registrierbare und bestätigbare Modul, verwenden Sie das Flag -v: `rails generate devise:views –v registrations confirmations`

Entwickeln Sie Controller-Filter und Helfer

Um einen Controller mit Benutzerauthentifizierung mit devise einzurichten, fügen Sie Folgendes vor before_action hinzu: (vorausgesetzt, Ihr Devise-Modell ist 'User'):

```
before_action :authenticate_user!
```

Um zu überprüfen, ob ein Benutzer angemeldet ist, verwenden Sie den folgenden Helfer:

```
user_signed_in?
```

Verwenden Sie für den aktuell angemeldeten Benutzer diesen Helfer:

```
current_user
```

Sie können auf die Sitzung für diesen Bereich zugreifen:

```
user_session
```

- Wenn Ihr Devise-Modell als `Member` statt `User` , ersetzen Sie den obigen `user` durch `member`

Omniauth

Wählen Sie zuerst Ihre Auth-Strategie und fügen Sie sie Ihrem `Gemfile` . Eine Liste der Strategien finden Sie hier: https://github.com/intridea/omniauth/wiki/List-of-Strategies

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

Sie können dies Ihrer Rail-Middleware wie folgt hinzufügen:

```
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
```

Standardmäßig fügt OmniAuth den Routen `/auth/:provider` , und Sie können mit diesen Pfaden beginnen.

Wenn ein Fehler auftritt, wird Omniauth standardmäßig zu `/auth/failure` umgeleitet

has_secure_password

Erstellen Sie ein Benutzermodell

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

Fügen Sie dem Benutzermodell das Modul has_secure_password hinzu

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

Jetzt können Sie einen neuen Benutzer mit Passwort erstellen

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

Überprüfen Sie das Kennwort mit der Authentifizierungsmethode

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

has_secure_token

Erstellen Sie ein Benutzermodell

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

Wenn Sie nun einen neuen Benutzer erstellen, werden automatisch ein Token und ein auth_token generiert

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

Sie können die Token mit `regenerate_token` und `regenerate_auth_token` aktualisieren

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

Benutzerauthentifizierung in Rails online lesen: https://riptutorial.com/de/ruby-on-rails/topic/1794/benutzerauthentifizierung-in-rails

Kapitel 27: Bereitstellung einer Rails-App auf Heroku

Examples

Bereitstellung Ihrer Anwendung

Stellen Sie sicher, dass Sie sich in dem Verzeichnis befinden, in dem sich Ihre Rails-App befindet, und erstellen Sie dann eine App auf Heroku.

```
$ heroku create example
Creating  example... done
https://example.herokuapp.com/ | https://git.heroku.com/example.git
```

Die erste URL der Ausgabe, http://example.herokuapp.com , ist der Ort, an dem die App verfügbar ist. Die zweite URL, git@heroku.com: example.git, ist die Remote-git-Repository-URL.

Dieser Befehl sollte nur für ein initialisiertes git-Repository verwendet werden. Der Befehl heroku create fügt automatisch eine git-Fernbedienung namens "heroku" hinzu, die auf diese URL zeigt.

Das App-Namensargument ("Beispiel") ist optional. Wenn kein Anwendungsname angegeben wird, wird ein zufälliger Name generiert. Da sich die Heroku-App-Namen in einem globalen Namespace befinden, können Sie davon ausgehen, dass allgemeine Namen wie „Blog“ oder „Wiki“ bereits vergeben sind. Es ist oft einfacher, mit einem Standardnamen zu beginnen und die App später umzubenennen.

Stellen Sie als Nächstes Ihren Code bereit:

```
$ 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
the lockfile. We suggest you upgrade to the latest version of Bundler by running `gem install
bundler`.
remote: -----> Preparing app for Rails asset pipeline
remote:        Running: rake assets:precompile
remote:        I, [2016-07-08T17:08:57.046245 #1222]  INFO -- : Writing
/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
```

Wenn Sie die Datenbank in Ihrer Anwendung verwenden, müssen Sie die Datenbank manuell migrieren, indem Sie Folgendes ausführen:

```
$ heroku run rake db:migrate
```

Alle Befehle nach dem `heroku run` werden auf einem Heroku-Dyno ausgeführt. Sie können eine interaktive Shell-Sitzung abrufen, indem Sie Folgendes ausführen:

```
$ heroku run bash
```

Stellen Sie sicher, dass Sie über einen Dyno den Webprozesstyp ausführen:

```
$ heroku ps:scale web=1
```

Der Befehl heroku ps listet die ausgeführten Dynamiken Ihrer Anwendung auf:

```
$ 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)
```

Sie können die App jetzt in unserem Browser mit `heroku open` .

```
$ heroku open
```

Heroku gibt Ihnen eine Standard-Web-URL in der Domäne `herokuapp.com` . Wenn Sie für die Produktion bereit sind, können Sie Ihre eigene benutzerdefinierte Domäne hinzufügen.

Produktions- und Inszenierungsumgebungen für ein Heroku verwalten

Jede Heroku-App kann in mindestens zwei Umgebungen ausgeführt werden: auf Heroku (wir nennen diese Produktion) und auf Ihrem lokalen Computer (Entwicklung). Wenn mehr als eine Person an der App arbeitet, gibt es mehrere Entwicklungsumgebungen - in der Regel eine pro Maschine. Normalerweise verfügt jeder Entwickler auch über eine Testumgebung für die Ausführung von Tests. Dieser Ansatz schlägt leider fehl, da die Umgebungen weniger ähnlich sind. Windows und Macs bieten beispielsweise unterschiedliche Umgebungen als der Linux-Stack von Heroku. Sie können also nicht immer sicher sein, dass in Ihrer lokalen Entwicklungsumgebung funktionierender Code auf die gleiche Weise funktioniert, wenn Sie ihn für die Produktion bereitstellen.

Die Lösung ist eine Staging-Umgebung, die der Produktion so ähnlich wie möglich ist. Dies kann erreicht werden, indem eine zweite Heroku-Anwendung erstellt wird, die Ihre Staging-Anwendung hostet. Mit Staging können Sie Ihren Code in einer produktionsähnlichen Einstellung überprüfen, bevor er sich auf Ihre tatsächlichen Benutzer auswirkt.

Von Anfang an anfangen

Angenommen, Sie haben eine Anwendung, die auf Ihrem lokalen Computer ausgeführt wird, und Sie sind bereit, sie auf Heroku zu übertragen. Wir müssen sowohl Remote-Umgebungen als auch Staging und Produktion erstellen. Um sich daran zu gewöhnen, zuerst auf die Inszenierung zu drängen, beginnen wir damit:

```
$ heroku create --remote staging
Creating strong-river-216.... done
http://strong-river-216.heroku.com/ | https://git.heroku.com/strong-river-216.git
Git remote staging added
```

Standardmäßig erstellt die Heroku-CLI Projekte mit einer Heroku-Fernbedienung. Hier geben wir einen anderen Namen mit der Flagge --remote an. Wenn Sie also Code in Heroku eingeben und Befehle gegen die App ausführen, sehen Sie etwas anders aus als der normale Git Push Heroku Master:

```
 $ git push staging master
...
 $ heroku ps --remote staging
=== web: `bundle exec puma -C config/puma.rb``
web.1: up for 21s
```

Sobald Ihre Staging-App ordnungsgemäß läuft, können Sie Ihre Produktions-App erstellen:

```
$ heroku create --remote production
Creating fierce-ice-327.... done
```

```
http://fierce-ice-327.heroku.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
```

Und damit haben Sie die gleiche Codebase wie zwei separate Heroku-Apps - eine Inszenierung und eine Produktion, die identisch eingerichtet sind. Denken Sie daran, dass Sie angeben müssen, welche App Sie bei Ihrer täglichen Arbeit verwenden werden. Sie können entweder das Flag "--remote" oder Ihre git config verwenden, um eine Standard-App anzugeben.

Bereitstellung einer Rails-App auf Heroku online lesen: https://riptutorial.com/de/ruby-on-rails/topic/4485/bereitstellung-einer-rails-app-auf-heroku

Kapitel 28: Bezahlfunktion in Schienen

Einführung

Dieses Dokument gibt vor, Ihnen anhand eines vollständigen Beispiels vorzustellen, wie Sie mit Ruby on Rails verschiedene Zahlungsmethoden implementieren können.

In diesem Beispiel werden zwei bekannte Zahlungsplattformen von Stripe und Braintree behandelt.

Bemerkungen

Dokumentation.

Streifen

Braintree

Examples

So integrieren Sie Stripe

`Gemfile` Stripe `Gemfile` unserem `Gemfile`

```
gem 'stripe'
```

Fügen `initializers/stripe.rb` Datei `initializers/stripe.rb` . Diese Datei enthält die erforderlichen Schlüssel für die Verbindung mit Ihrem Stripe-Konto.

```
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]
```

So erstellen Sie einen neuen Kunden für Stripe

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

Dieser Code erstellt einen neuen Kunden in Stripe mit der angegebenen E-Mail-Adresse und

Quelle.

`payment_token` ist das Token des Kunden, das eine Zahlungsmethode wie eine Kreditkarte oder ein Bankkonto enthält. Weitere Informationen: Stripe.js clientseitig

So rufen Sie einen Plan von Stripe ab

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

Dieser Code ruft einen Plan anhand seiner ID von Stripe ab.

So erstellen Sie ein Abonnement

Wenn wir einen Kunden und einen Plan haben, können wir ein neues Abonnement für Stripe erstellen.

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

Es wird ein neues Abonnement erstellt und unser Benutzer wird belastet. Es ist wichtig zu wissen, was wirklich in Stripe passiert, wenn wir einen Benutzer für einen Plan abonnieren. Weitere Informationen finden Sie hier: Stripe-Abonnement-Lebenszyklus .

So belasten Sie einen Benutzer mit einer einzigen Zahlung

Manchmal möchten wir unsere Benutzer nur einmal berechnen, da wir das nächste tun.

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

In diesem Fall belasten wir unseren Benutzer einmalig für den angegebenen Betrag.

Häufige Fehler:

- Der Betrag muss in ganzzahliger Form gesendet werden, dh 2000 werden 20 Währungseinheiten sein. Überprüfen Sie dieses Beispiel
- Sie können einen Benutzer nicht in zwei Währungen berechnen. Wenn der Benutzer in der Vergangenheit zu irgendeinem Zeitpunkt in EUR berechnet wurde, können Sie ihn nicht in USD berechnen.
- Sie können dem Benutzer keine Kosten berechnen (Zahlungsmethode).

Bezahlfunktion in Schienen online lesen: https://riptutorial.com/de/ruby-on-

rails/topic/10929/bezahlfunktion-in-schienen

Kapitel 29: Caching

Examples

Russisches Puppen-Caching

Sie können zwischengespeicherte Fragmente in anderen zwischengespeicherten Fragmenten verschachteln. Dies wird als `Russian doll caching` .

Das `Russian doll caching` bietet den Vorteil, dass bei der Aktualisierung eines einzelnen Produkts alle anderen inneren Fragmente beim Regenerieren des äußeren Fragments wiederverwendet werden können.

Wie im vorherigen Abschnitt erläutert, `updated_at` eine zwischengespeicherte Datei, wenn sich der Wert von `updated_at` für einen Datensatz ändert, von dem die zwischengespeicherte Datei direkt abhängt. Dadurch verfällt jedoch kein Cache, in dem das Fragment verschachtelt ist.

Nehmen Sie zum Beispiel die folgende Ansicht:

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

Was wiederum diese Ansicht wiedergibt:

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

Wenn ein Attribut des Spiels geändert wird, wird der `updated_at` Wert auf die aktuelle Uhrzeit gesetzt, wodurch der Cache abläuft.

Da jedoch `updated_at` für das Produktobjekt nicht geändert wird, ist der Cache nicht abgelaufen, und Ihre App `updated_at` veraltete Daten. Um dies zu beheben, binden wir die Modelle mit der Touch-Methode zusammen:

```
class Product < ApplicationRecord
  has_many :games
end

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

SQL-Caching

Abfrage-Caching ist eine `Rails` Funktion, die die von jeder Abfrage zurückgegebene Ergebnismenge zwischenspeichert. Wenn `Rails` dieselbe Abfrage erneut für diese Anforderung

findet, verwendet es die zwischengespeicherte Ergebnismenge, `Rails` die Abfrage erneut für die Datenbank auszuführen.

Zum Beispiel:

```
class ProductsController < ApplicationController

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

    ...

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

end
```

Wenn die gleiche Abfrage zum zweiten Mal für die Datenbank ausgeführt wird, wird sie nicht wirklich die Datenbank treffen. Wenn das Ergebnis zum ersten Mal von der Abfrage zurückgegeben wird, wird es im Abfrage-Cache (im Speicher) gespeichert, und das zweite Mal wird es aus dem Speicher abgerufen.

Beachten Sie jedoch, dass Abfrage-Caches zu Beginn einer Aktion erstellt und am Ende dieser Aktion gelöscht werden und daher nur für die Dauer der Aktion bestehen bleiben. Wenn Sie die Abfrageergebnisse dauerhaft speichern möchten, können Sie die Zwischenspeicherung auf niedriger Ebene verwenden.

Fragment-Caching

`Rails.cache` , bereitgestellt von ActiveSupport, kann verwendet werden, um jedes serialisierbare Ruby-Objekt zwischen Anforderungen zwischenzuspeichern.

Verwenden Sie `cache.read` um einen Wert für einen bestimmten Schlüssel aus dem Cache `cache.read` .

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

Verwenden Sie `cache.write` , um einen Wert in den Cache zu schreiben:

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

Alternativ können Sie `cache.fetch` , um einen Wert aus dem Cache zu lesen und optional einen Standardwert zu schreiben, wenn kein Wert vorhanden ist:

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

```
end
```

Der Rückgabewert des übergebenen Blocks wird dem Cache unter dem angegebenen Schlüssel zugewiesen und anschließend zurückgegeben.

Sie können auch einen Cache-Ablauf angeben:

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

Zwischenspeicherung der Seite

Mit dem ActionPack page_caching können Sie einzelne Seiten zwischenspeichern. Dadurch wird das Ergebnis einer dynamischen Anforderung als statische HTML-Datei gespeichert, die bei nachfolgenden Anforderungen anstelle der dynamischen Anforderung bereitgestellt wird. Die Readme-Datei enthält vollständige Anweisungen zur Installation. Verwenden Sie nach dem `caches_page` Klassenmethode `caches_page` in einem Controller, um das Ergebnis einer Aktion zwischenzuspeichern:

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

Verwenden Sie `expire_page` , um den Ablauf des Caches durch Löschen der gespeicherten HTML-Datei zu erzwingen:

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

  def index
    @users = User.all
  end

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

Die Syntax von `expire_page` ahmt die von `url_for` und friends nach.

HTTP-Caching

Rails> = 3 verfügt über HTTP-Caching-Funktionen. Dies verwendet die Header `Cache-Control` und `ETag` zu steuern, wie lange ein Client oder ein Vermittler (z. B. ein CDN) eine Seite zwischenspeichern kann.

Verwenden `expires_in` in einer Controller-Aktion `expires_in` , um die Länge der Zwischenspeicherung für diese Aktion `expires_in` :

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

Verwenden Sie `expires_now` , um den sofortigen Ablauf einer zwischengespeicherten Ressource für jeden Besucherkunden oder `expires_now` zu erzwingen:

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

Action-Caching

Wie beim Seiten-Caching wird beim Action-Caching die gesamte Seite zwischengespeichert. Der Unterschied besteht darin, dass die Anforderung den Rails-Stack trifft, bevor Filter ausgeführt werden, bevor der Cache bereitgestellt wird. Es wird aus Rails zu actionpack-action_caching gem extrahiert.

Ein häufiges Beispiel ist das Zwischenspeichern einer Aktion, für die eine Authentifizierung erforderlich ist:

```
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
```

Zu den Optionen gehören `:expires_in` , ein custom `:cache_path` (für Aktionen mit mehreren Routen, die unterschiedlich zwischengespeichert werden sollten) und `:if` / `:unless` gesteuert, wann die Aktion zwischengespeichert werden soll.

```
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
```

Wenn das Layout über dynamischen Inhalt verfügt, zwischenspeichern Sie nur den Aktionsinhalt, indem Sie das `layout: false` .

Caching online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2833/caching

Kapitel 30: Datei-Uploads

Examples

Einzelnes Datei-Upload mit Carrierwave

Die Verwendung von Datei-Uploads in Rails ist ziemlich einfach. Als Erstes müssen Sie ein Plugin für die Verwaltung der Uploads auswählen. Die häufigsten Einflüsse sind **Carrierwave** und **Paperclip** . Beide sind in ihrer Funktionalität ähnlich und reich an Dokumentation

Schauen wir uns das Beispiel mit einem einfachen Avatar-Upload-Bild mit Carrierwave an

Nach der `bundle install` Carrierwave geben Sie die Konsole ein

```
$ rails generate uploader ProfileUploader
```

Dadurch wird eine Konfigurationsdatei unter */app/uploaders/profile_uploader.rb erstellt*

Hier können Sie Speicher einrichten (z. B. lokal oder in der Cloud), Erweiterungen für Bildmanipulationen anwenden (z. B. Thumbnails über MiniMagick generieren) und serverseitige Erweiterungs-Whitelist festlegen

Erstellen Sie als Nächstes eine neue Migration mit dem String-Tipe für user_pic und stellen Sie den *Upload*- Uploader im *user.rb*- Modell *bereit* .

```
mount_uploader :user_pic, ProfileUploader
```

Zeigen Sie als Nächstes ein Formular zum Hochladen des Avatars an (möglicherweise eine Bearbeitungsansicht für den Benutzer).

```
<% 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 %>
```

Stellen Sie sicher, dass Sie {multipart: true} in das Bestellformular einschließen, um Uploads zu verarbeiten. Accept ist eine Option zum Festlegen einer clientseitigen Erweiterungs-Whitelist.

Um einen Avatar anzuzeigen, machen Sie einfach

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

Geschachteltes Modell - mehrere Uploads

Wenn Sie mehrere Uploads erstellen möchten, sollten Sie zunächst ein neues Modell erstellen und Beziehungen einrichten

Angenommen, Sie möchten mehrere Bilder für das Produktmodell. Erstellen Sie ein neues Modell und machen Sie `belongs_to` Ihrem übergeordneten Modell

```
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 ist ein Muss, da wir damit geschachtelte Formulare erstellen können, sodass wir eine neue Datei hochladen, den Produktnamen ändern und den Preis aus einem einzigen Formular festlegen können

Als Nächstes erstellen Sie ein Formular in einer Ansicht (Bearbeiten / Erstellen).

```
    <%= 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 %>
```

Controller ist nichts Besonderes. Wenn Sie keinen neuen erstellen möchten, erstellen Sie einfach einen neuen Controller in Ihrem Produktcontroller

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

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

Alle Bilder in einer Ansicht anzeigen

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

Datei-Uploads online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2831/datei-uploads

Kapitel 31: Debuggen

Examples

Debuggen der Rails-Anwendung

Um eine Anwendung debuggen zu können, ist es sehr wichtig, den Ablauf der Logik und der Daten einer Anwendung zu verstehen. Es hilft bei der Behebung logischer Fehler und erhöht die Programmierkenntnisse und die Codequalität. Zwei beliebte Edelsteine zum Debuggen sind der Debugger (für Ruby 1.9.2 und 1.9.3) und Byebug (für Ruby> = 2.x).

`.rb` , um `.rb` Dateien zu debuggen:

1. Fügen `byebug` der `development` von `Gemfile debugger` oder `byebug Gemfile`
2. Führen Sie die `bundle install`
3. Fügen `byebug` als Haltepunkt `debugger` oder `byebug`
4. Führen Sie den Code aus oder stellen Sie eine Anfrage
5. Das Rail-Serverprotokoll wird am angegebenen Haltepunkt angehalten
6. An diesem Punkt können Sie Ihr Serverterminal wie eine `rails console` und die Werte von Variablen und Parametern überprüfen
7. Um zur nächsten Anweisung zu gelangen, geben Sie `next` und drücken Sie die `enter`
8. Geben Sie `c` und drücken Sie die `enter`

Wenn Sie `.html.erb` Dateien debuggen `.html.erb` , wird der Haltepunkt als `<% debugger %>` hinzugefügt

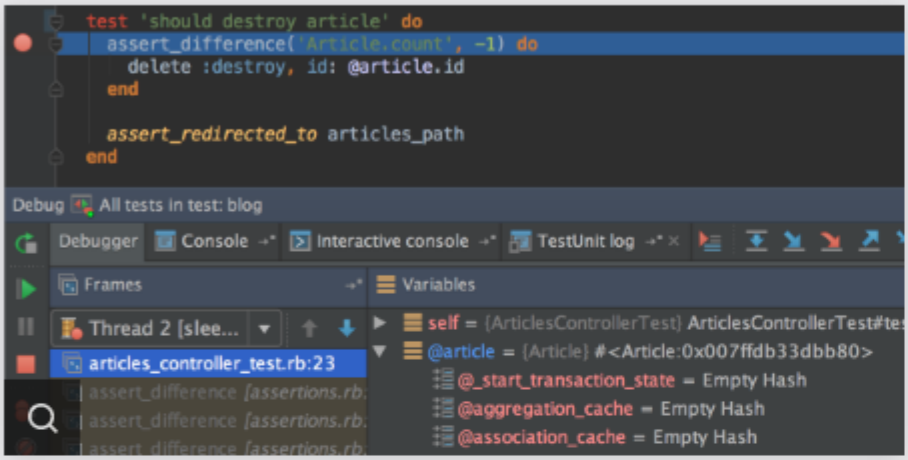
Debuggen in Ihrer IDE

Jede gute IDE bietet eine grafische Benutzeroberfläche für das interaktive Debuggen von Ruby-Anwendungen (und damit auch von Rails-Anwendungen), in der Sie Haltepunkte hinzufügen, überwachen und bei Ausnahmebedingungen automatisch pausieren können. Außerdem können Sie die Codeausführung Schritt für Schritt und Zeile für Zeile verfolgen.

Sehen Sie sich beispielsweise eine der besten Ruby-IDEs an, die Debugging-Funktionen von RubyMine

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.



Convenient user in

- Look under the hood of ar Variables and Watches vie
- The UI is fully customizabl select toolbar commands, project code while steppir
- The debugger UI is also tig navigate between the deb etc.
- You also get the complete views.

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.

Debugging JavaSc

- RubyMine provides an adv which works with Google
- You can easily debug ECM on RubyMine debugger's
- A full-featured debugger f to debug apps running loc

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.

Dedicated Watches

Track any number of expressio current stack frame context. T through your debugging sessi

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.

Remote debugging

As you connect to a remote ho mapping between the local so debug processes can be launc

Das Debuggen durch Auslösen von Ausnahmen ist *viel einfacher* als das Schielen durch `print` Bei den meisten Fehlern ist es `byebug` *viel schneller* als das Öffnen eines irb-Debuggers wie `pry` oder `byebug` . Diese Tools sollten nicht der erste Schritt sein.

Ruby / Rails schnell debuggen:

1. Fast-Methode: Dann eine `Exception` `.inspect` und das Ergebnis `.inspect`

Der *schnellste* Weg zum Debuggen von Ruby-Code (insbesondere von Rails-Code) besteht darin `raise` eine Ausnahme entlang des Ausführungspfads Ihres Codes `.inspect` während `.inspect` für die Methode oder das Objekt (z. B. `foo`) `.inspect` :

```
raise foo.inspect
```

Im obigen Code löst *`raise`* eine *`Exception`* , die die *Ausführung Ihres Codes* `.inspect` , und gibt eine Fehlermeldung aus, die zweckmäßigerweise Informationen über das Objekt / die Methode (dh `foo`) in der Zeile enthält, die Sie debuggen möchten.

Diese Technik ist nützlich, um ein Objekt oder eine Methode *schnell zu* untersuchen (*z. B. ist es `nil` ?*) Und sofort zu bestätigen, ob eine Codezeile überhaupt innerhalb eines gegebenen Kontextes ausgeführt wird.

2. Fallback: Verwenden Sie einen Rubin- *IRB*- Debugger wie `byebug` oder `pry`

Erst nachdem Sie Informationen über den Status des Codeausführungsablaufs erhalten haben, sollten Sie in Erwägung ziehen, zu einem Ruby-Edelstein-Debugger wie `pry` oder `byebug` in dem Sie tiefer in den Status von Objekten innerhalb Ihres Ausführungspfads `byebug` können.

So verwenden Sie den `byebug` Edelstein zum Debuggen in Rails:

1. Fügen Sie den `gem 'byebug'` innerhalb der *Entwicklungsgruppe* in Ihrem *Gemfile hinzu*
2. Führen Sie die `bundle install`
3. `byebug` die Phrase `byebug` in den Ausführungspfad des Codes ein, den Sie untersuchen möchten.

`byebug` diese `byebug` Variable ausgeführt wird, wird eine Ruby-IRB-Sitzung Ihres Codes geöffnet, sodass Sie direkt auf den Status der Objekte zugreifen können, wie sie sich zu diesem Zeitpunkt in der Ausführung des Codes befinden.

IRB-Debugger wie Byebug sind nützlich, um den Status Ihres Codes bei der Ausführung genau zu

analysieren. Sie sind jedoch zeitaufwändiger als das Erheben von Fehlern. Daher sollten sie in den meisten Situationen nicht der erste Schritt sein.

Allgemeine Anfängerhinweise

Wenn Sie versuchen, ein Problem zu debuggen, sollten Sie immer folgendes **tun** : **Lesen Sie die! @ # $ lng-Fehlermeldung (RTFM).**

Das bedeutet, dass Sie *die* Fehlermeldungen *sorgfältig* und *vollständig* lesen, bevor Sie handeln, damit Sie *verstehen, was Sie zu sagen versuchen.* Stellen Sie beim Debuggen beim Lesen einer Fehlermeldung die folgenden mentalen Fragen *in dieser Reihenfolge* :

1. Welche ***Klasse*** bezieht sich der Fehler? (dh *ich habe die richtige Objektklasse oder ist mein Objekt gleich* `nil` *?*)
2. Welche ***Methode*** bezieht sich der Fehler? (dh *ist ihr Typ in der Methode; kann ich diese Methode für diesen Objekttyp / diese Objektklasse aufrufen?*)
3. Schließlich, was ich aus meinen letzten beiden Fragen entnehmen kann, welche ***Codezeilen*** sollte ich untersuchen? (Denken Sie daran: Die letzte Codezeile in der Stack-Ablaufverfolgung muss nicht unbedingt das Problem sein.)

Achten Sie im Stack-Trace besonders auf Codezeilen, die aus Ihrem Projekt stammen (z. B. Zeilen, die mit `app/...` wenn Sie Rails verwenden). In 99% der Fälle liegt das Problem bei Ihrem eigenen Code.

Um zu veranschaulichen, warum Dolmetschen *in dieser Reihenfolge* wichtig ist ...

ZB eine Ruby-Fehlermeldung, die viele Anfänger verwirrt:

Sie führen Code aus, der zu einem bestimmten Zeitpunkt als solcher ausgeführt wird:

```
@foo = Foo.new

...

@foo.bar
```

und Sie erhalten einen Fehler, der besagt:

```
undefined method "bar" for Nil:nilClass
```

Anfänger sehen diesen Fehler und denken , das Problem ist , dass die Methode `bar` ist *nicht definiert.* **Es ist nicht.** In diesem Fehler ist der Realteil, auf den es ankommt, Folgendes:

```
for Nil:nilClass
```

`for Nil:nilClass` bedeutet, dass `@foo` ist! `@foo` ist keine `Foo` Instanzvariable! Sie haben ein Objekt, das `Nil` . Wenn dieser Fehler auftritt, ist es einfach rubin versuchen , Ihnen zu sagen , dass die

Methode `bar` nicht für Objekte der Klasse existiert `Nil` . (na ja, da wir versuchen, eine Methode für ein Objekt der Klasse `Foo` nicht `Nil`).

Aufgrund dieses Fehlers (`undefined method "bar" for Nil:nilClass`) ist es `undefined method "bar" for Nil:nilClass` leicht zu `undefined method "bar" for Nil:nilClass` , dass dieser Fehler damit zusammenhängt, dass der `bar undefined` . Wenn dieser Fehler nicht sorgfältig gelesen wird, fangen die Anfänger fälschlicherweise an, die Details der `bar` Methode auf `Foo` , wobei der Teil des Fehlers, der darauf hinweist, dass das Objekt die falsche Klasse ist (in diesem Fall: nil), völlig fehlt. Es ist ein Fehler, der durch das Lesen von Fehlermeldungen vollständig vermieden werden kann.

Zusammenfassung:

Lesen Sie immer sorgfältig **die *gesamte* Fehlermeldung,** bevor Sie mit dem Debuggen beginnen. Das bedeutet: Immer den ***Klassentyp*** eines Objekts in einer Fehlermeldung überprüfen Sie *zuerst,* dann seine ***Methoden,*** *bevor* Sie sleuthing in jede Stacktrace oder Codezeile beginnen , wo Sie den Fehler denken auftreten kann. Diese 5 Sekunden können Ihnen 5 Stunden Frustration ersparen.

tl; dr: Blinzeln Sie nicht bei Druckprotokollen: Erheben Sie stattdessen Ausnahmen. Vermeiden Sie Kaninchenlöcher, indem Sie die Fehler vor dem Debuggen sorgfältig lesen.

Debuggen der Ruby-on-Rails-Anwendung mit pry

pry ist ein leistungsfähiges Werkzeug, mit dem Sie jede Ruby-Anwendung debuggen können. Das Einrichten einer Ruby-on-Rail-Anwendung mit diesem Edelstein ist sehr einfach und unkompliziert.

Konfiguration

So starten Sie das Debuggen Ihrer Anwendung mit pry

- In `gem 'pry'` , um die Anwendung des `Gemfile` und bündeln

```
group :development, :test do
  gem 'pry'
end
```

- Navigieren Sie zum Stammverzeichnis der Anwendung auf der Terminal-Konsole und führen Sie die `bundle install` . Sie können es überall in Ihrer Anwendung einsetzen.

Benutzen

Die Verwendung von pry in Ihrer Anwendung umfasst lediglich die `binding.pry` von `binding.pry` an den Haltepunkten, die Sie beim Debuggen überprüfen möchten. Sie können `binding.pry` Haltepunkte an beliebiger Stelle in Ihrer Anwendung hinzufügen, die von Ruby Interpreter interpretiert werden (alle App / Controller, App / Modelle, App / Views-Dateien).

i) Controller debuggen

app / controller / users_controller.rb

```
class UsersController < ApplicationController
  def show
    use_id = params[:id]
    // 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
```

In diesem Beispiel wird der Rails-Server am Haltepunkt mit einer `UsersController` wenn Sie versuchen, ein `UsersController` , `show` Aktion auf `UsersController` . Sie können das `params` prüfen und die ActiveRecord-Abfrage für das `User` von diesem Haltepunkt aus durchführen

ii) Einen View debuggen

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"
```

In diesem Beispiel wird der Haltepunkt mit der Hebelkonsole angehalten, wenn die `users/show` Seite im Rail-Server vorkompiliert ist, bevor sie an den Browser des Clients zurückgesendet wird. Dieser Haltepunkt ermöglicht es, die Richtigkeit von `@user.logged_in?` zu debuggen `@user.logged_in?` wenn es sich schlecht benimmt

ii) Modell debuggen

```
app/models/user.rb

class User < ActiveRecord::Base
  def full_name
    binding.pry
    "#{self.first_name} #{self.last_name}"
  end
end
```

In diesem Beispiel kann der Haltepunkt zum Debuggen der Instanzmethode `full_name` `User` verwendet werden, wenn diese Methode an einer beliebigen Stelle in der Anwendung aufgerufen

wird.

Zusammenfassend ist pry ein leistungsfähiges Debugging-Tool für Schienenanwendungen mit einfachem Setup und einfacher Debugging-Richtlinie. Probieren Sie es aus.

Debuggen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3877/debuggen

Kapitel 32: Dekorateur-Muster

Bemerkungen

Mit dem **Decorator-Muster** können Sie das Verhalten von Objekten situationsabhängig hinzufügen oder ändern, ohne das Basisobjekt zu beeinflussen.

Dies kann durch einfaches Ruby mit der stdlib oder durch beliebte Edelsteine wie Draper erreicht werden .

Examples

Dekorieren eines Modells mit SimpleDelegator

Die meisten Rails-Entwickler beginnen mit der Modifizierung ihrer Modellinformationen in der Vorlage selbst:

```
<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>
```

Bei Modellen mit vielen Daten kann dies schnell mühsam werden und zu einer Kopierlogik von einer Vorlage in eine andere führen.

In diesem Beispiel wird `SimpleDelegator` aus der stdlib verwendet.

Alle Anforderungen an ein `SimpleDelegator` Objekt werden standardmäßig an das übergeordnete Objekt übergeben. Sie können jede Methode mit der Präsentationslogik überschreiben oder neue Methoden hinzufügen, die für diese Ansicht spezifisch sind.

`SimpleDelegator` bietet zwei Methoden: `__setobj__` , um `__setobj__` an welches Objekt delegiert wird, und `__getobj__` , um das Objekt `__getobj__` .

```
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
```

Einige Dekorateure verlassen sich auf Magie, um dieses Verhalten zu verknüpfen. Sie können jedoch deutlich machen, woher die Präsentationslogik kommt, indem Sie das Objekt auf der Seite initialisieren.

```
<% user = UserDecorator.new(@user, self) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

Durch die Übergabe eines Verweises auf das Ansichtsobjekt in den Dekorateur können wir trotzdem auf alle übrigen Ansichtshilfen zugreifen, während Sie die Präsentationslogik erstellen, ohne sie hinzufügen zu müssen.

Die Ansichtsvorlage befasst sich jetzt nur noch mit dem Einfügen von Daten in die Seite und ist deutlich übersichtlicher.

Dekorieren eines Modells mit Draper

Draper passt Modelle automatisch an ihre Dekorateure an.

```
# 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
```

Bei einer `@user` Variablen, die ein ActiveRecord-Objekt enthält, können Sie auf Ihren Dekorateur zugreifen, indem Sie `#decorate` für den `@user` oder die Draper-Klasse `@user` , wenn Sie spezifisch sein möchten.

```
<% user = @user.decorate %><!-- OR -->
<% user = UserDecorator.decorate(@user) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

Dekorateur-Muster online lesen: https://riptutorial.com/de/ruby-on-rails/topic/5694/dekorateur-muster

Kapitel 33: Edelsteine

Bemerkungen

Gemfile-Dokumentation

Für Projekte, die voraussichtlich wachsen, ist es eine gute Idee, Kommentare zu Ihrem `Gemfile` . Auf diese Weise wissen Sie auch in großen Setups immer noch, was jeder Edelstein tut, auch wenn der Name nicht selbsterklärend ist und Sie ihn vor zwei Jahren hinzugefügt haben.

Auf diese Weise können Sie sich auch daran erinnern, warum Sie sich für eine bestimmte Version entschieden haben, und die Versionsanforderung später erneut prüfen.

Beispiele:

```
# temporary downgrade for TeamCity
gem 'rake', '~> 10.5.0'
# To upload invoicing information to payment provider
gem 'net-sftp'
```

Examples

Was ist ein Juwel?

Ein gem ist das Äquivalent zu einem Plugin oder einer Erweiterung für die Programmiersprache Ruby.

Um genau zu sein, sogar Schienen sind nichts weiter als ein Juwel. Viele Edelsteine werden auf Schienen oder anderen Edelsteinen gebaut (sie sind von diesem Edelstein abhängig) oder sind eigenständig.

In Ihrem Rails-Projekt

Gemfile

Für Ihr Rails-Projekt haben Sie eine Datei namens `Gemfile` . Hier können Sie Edelsteine hinzufügen, die Sie in Ihr Projekt aufnehmen und verwenden möchten. Nach dem Hinzufügen Sie benötigen den Edelstein installieren , indem Sie `bundler` (siehe Abschnitt Bündler).

Gemfile.lock

Sobald Sie dies getan haben, wird Ihr `Gemfile.lock` mit Ihren neu hinzugefügten Edelsteinen und ihren Abhängigkeiten aktualisiert. Diese Datei sperrt Ihre verwendeten Edelsteine, so dass sie die in dieser Datei deklarierte Version verwenden.

```
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)
```

Dieses Beispiel ist für das Juwel `devise` . In der `Gemfile.lock` die Version `4.0.3` deklariert, um bei der Installation Ihres Projekts auf einem anderen Computer oder auf Ihrem Produktionsserver anzugeben, welche Version verwendet werden soll.

Entwicklung

Entweder arbeitet eine einzelne Person, eine Gruppe oder eine ganze Gemeinschaft an einem Juwel. Die geleistete Arbeit wird normalerweise freigegeben, nachdem bestimmte `issues` behoben oder `features` hinzugefügt wurden.

In der Regel folgen die Releases dem Semantic Versioning 2.0.0- Prinzip.

Bundler

Der einfachste Weg, Edelsteine zu handhaben und zu verwalten, ist der `bundler` . Bundler ist ein Paketmanager, der mit Bower vergleichbar ist.

Um den Bundler zu verwenden, müssen Sie ihn zunächst installieren.

```
gem install bundler
```

Nachdem Sie den Bundler in Betrieb genommen haben, müssen Sie nur noch Edelsteine zu Ihrem `Gemfile` und ausführen

```
bundle
```

in Ihrem Terminal. Dadurch werden Ihre neu hinzugefügten Edelsteine in Ihrem Projekt installiert. Sollte ein Problem auftreten, erhalten Sie eine Aufforderung in Ihrem Terminal.

Wenn Sie an weiteren Details interessiert sind, schlage ich vor, die Dokumente zu lesen .

Gemfiles

Zum Starten benötigen gemfiles mindestens eine Quelle in Form der URL für einen RubyGems-Server.

Generieren Sie eine Gemfile mit der Standardquelle rubygems.org, indem Sie das `bundle init` ausführen. Verwenden Sie https, damit Ihre Verbindung zum Server mit SSL überprüft wird.

```
source 'https://rubygems.org'
```

Als nächstes deklarieren Sie die benötigten Edelsteine einschließlich der Versionsnummern.

```
gem 'rails', '4.2.6'
gem 'rack',  '>=1.1'
gem 'puma',  '~>3.0'
```

Die meisten Versionsbezeichner, wie> = 1.0, sind selbsterklärend. Der Spezifizierer ~> hat eine besondere Bedeutung. ~> 2.0.3 ist identisch mit> = 2.0.3 und <2.1. ~> 2.1 ist identisch mit> = 2.1 und <3.0. ~> 2.2.beta wird mit Vorabversionen wie 2.2.beta.12 übereinstimmen.

Git-Repositories sind ebenfalls gültige Edelsteinquellen, sofern das Repo einen oder mehrere gültige Edelsteine enthält. Legen Sie fest, was mit dem `:tag` Befehl überprüft werden soll `:tag` ,: `:branch` oder `:ref` . Die Standardeinstellung ist der `master` - Zweig.

```
gem 'nokogiri', :git => 'https://github.com/sparklemotion/nokogiri', :branch => 'master'
```

Wenn Sie einen entpackten Edelstein direkt aus dem Dateisystem verwenden möchten, setzen Sie die Option: path einfach auf den Pfad, der die Dateien des Edelsteins enthält.

```
gem 'extracted_library', :path => './vendor/extracted_library'
```

Abhängigkeiten können in Gruppen platziert werden. Gruppen können bei der Installation Zeit (unter Verwendung ignoriert werden `--without`) oder erforderlich , um alle auf einmal (mit `Bundler.require`).

```
gem 'rails_12factor', group: :production

group :development, :test do
  gem 'byebug'
  gem 'web-console', '~> 2.0'
  gem 'spring'
  gem 'dotenv-rails'
end
```

Sie können die gewünschte Version von Ruby in der Gemfile mit `ruby` angeben. Wenn die Gemfile-Datei in einer anderen Ruby-Version geladen wird, gibt Bundler eine Ausnahme mit einer Erklärung aus.

```
ruby '2.3.1'
```

Gemsets

Wenn Sie `RVM(Ruby Version Manager)` ist die Verwendung eines `gemset` für jedes Projekt eine gute Idee. Ein `gemset` ist nur ein Container, mit dem Sie die Edelsteine voneinander trennen können. `gemset` Sie ein `gemset` pro Projekt `gemset` , können Sie Edelsteine (und Edelsteinversionen) für ein Projekt ändern, ohne alle anderen Projekte zu `gemset` . Jedes Projekt muss sich nur um seine

eigenen Edelsteine kümmern.

`RVM` bietet (> = 0.1.8) ein `@global gemset` pro Ruby-Interpreter. Gems, die `@global gemset` für ein bestimmtes Rubin in das `@global gemset` installieren, stehen allen anderen Gemsets zur Verfügung, die Sie in Verbindung mit diesem Rubin erstellen. Dies ist ein guter Weg, um zuzulassen, dass alle Ihre Projekte den gleichen installierten Edelstein für eine bestimmte Ruby-Interpreterinstallation verwenden.

Gemsets erstellen

Angenommen, Sie haben bereits `ruby-2.3.1` installiert und Sie haben es mit folgendem Befehl ausgewählt:

```
rvm use ruby-2.3.1
```

Um nun ein Gemset für diese Ruby-Version zu erstellen:

```
rvm gemset create new_gemset
```

Dabei ist das `new_gemset` der Name von gemset. So zeigen Sie die Liste der verfügbaren Edelsteine für eine Ruby-Version an:

```
rvm gemset list
```

Um die Edelsteine aller Ruby-Versionen aufzulisten:

```
rvm gemset list_all
```

um ein gemset aus der Liste zu verwenden (angenommen, `new_gemset` ist das gemset, das ich verwenden möchte):

```
rvm gemset use new_gemset
```

Sie können die Ruby-Version auch mit dem Gemset angeben, wenn Sie zu einer anderen Ruby-Version wechseln möchten:

```
rvm use ruby-2.1.1@new_gemset
```

So legen Sie ein Standard-Gemset für eine bestimmte Ruby-Version fest:

```
rvm use 2.1.1@new_gemset --default
```

Um alle installierten Edelsteine aus einem Gemset zu entfernen, können Sie es leeren, indem Sie:

```
rvm gemset empty new_gemset
```

Um ein Gemset von einem Rubin in einen anderen zu kopieren, können Sie es tun, indem Sie:

```
rvm gemset copy 2.1.1@rails4 2.1.2@rails4
```

ein gemset löschen:

```
rvm gemset delete new_gemset
```

um den aktuellen gemsetnamen zu sehen:

```
rvm gemset name
```

So installieren Sie einen Edelstein im globalen Gemset:

```
rvm @global do gem install ...
```

Initialisieren von Gemsets während der Ruby-Installation

Wenn Sie einen neuen Ruby installieren, erstellt RVM nicht nur zwei Gemsets (das standardmäßige, leere Gemset und das globale Gemset), sondern verwendet auch eine Reihe von vom Benutzer bearbeitbaren Dateien, um zu bestimmen, welche Gems installiert werden sollen.

Arbeiten in `~/.rvm/gemsets` , rvm searchs für `global.gems` und `default.gems` eine Baum-Hierachie auf dem Rubin - String basierend Verwendung installiert. Am Beispiel von `ree-1.8.7-p2010.02` überprüft und importiert rvm die folgenden Dateien:

```
~/.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
```

Wenn Sie beispielsweise `~/.rvm/gemsets/global.gems` bearbeitet haben, indem Sie diese beiden Zeilen hinzufügen:

```
bundler
awesome_print
```

Jedes Mal, wenn Sie einen neuen Rubin installieren, werden diese beiden Edelsteine in Ihrem globalen Gemset installiert. `default.gems` und `global.gems` Dateien werden in der Regel während der Aktualisierung von rvm überschrieben.

Edelsteine online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3130/edelsteine

Kapitel 34: Elasticsearch

Examples

Installation und Prüfung

Das erste, was Sie für die lokale Entwicklung tun möchten, ist, ElasticSearch auf Ihrem Computer zu installieren und zu testen, ob es läuft. Es muss Java installiert sein. Die Installation ist ziemlich unkompliziert:

- Mac OS X: `brew install elasticsearch`
- Ubuntu: `sudo apt-get install elasticsearch`

Dann starte es:

- Mac OS X: `brew services start elasticsearch`
- Ubuntu: `sudo service elasticsearch start`

Am einfachsten geht es mit `curl` . Es kann einige Sekunden dauern, bis es startet, also keine Panik, wenn Sie zuerst keine Antwort erhalten.

```
curl localhost:9200
```

Beispielantwort:

```
{
  "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"
}
```

Werkzeuge für die Entwicklung einrichten

Wenn Sie mit ElasticSearch (ES) beginnen, ist es möglicherweise sinnvoll, ein grafisches Werkzeug zu haben, mit dem Sie Ihre Daten untersuchen können. Ein Plugin namens `elasticsearch-head` macht genau das. Gehen Sie folgendermaßen vor, um es zu installieren:

- Finden Sie heraus, in welchem Ordner ES installiert ist: `ls -l $(which elasticsearch)`
- `cd` in diesen Ordner und führen Sie das Plugin-Installations-Binärprogramm aus: `elasticsearch/bin/plugin -install mobz/elasticsearch-head`
- Öffnen Sie `http://localhost:9200/_plugin/head/` in Ihrem Browser

Wenn alles wie erwartet funktioniert, sollten Sie eine nette GUI sehen, auf der Sie Ihre Daten

untersuchen können.

Einführung

ElasticSearch verfügt über eine gut dokumentierte JSON-API, aber Sie möchten wahrscheinlich einige Bibliotheken verwenden, die das für Sie erledigen:

- `Elasticsearch` - der offizielle Low-Level-Wrapper für die HTTP-API
- `Elasticsearch-rails` - die offizielle `Elasticsearch-rails` Rails-Integration, mit der Sie Ihre Rails-Modelle mit ElasticSearch mithilfe des ActiveRecord- oder Repository-Musters verbinden können
- `Chewy` - Eine alternative, nicht offizielle High-End-Rails-Integration, die sehr beliebt ist und eine bessere Dokumentation bietet

Verwenden wir die erste Option zum Testen der Verbindung:

```
gem install elasticsearch
```

Starten Sie dann das Ruby-Terminal und probieren Sie es aus:

```
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'
```

Suchkick

Wenn Sie schnell elasticsearch einrichten möchten, können Sie den searchkick gem verwenden:

```
gem 'searchkick'
```

Fügen Sie den Modellen, nach denen Sie suchen möchten, einen Suchkick hinzu.

```
class Product < ActiveRecord::Base
  searchkick
end
```

Fügen Sie dem Suchindex Daten hinzu.

```
Product.reindex
```

Und zur Abfrage verwenden Sie:

```
products = Product.search "apples"
products.each do |product|
  puts product.name
end
```

Ziemlich schnell, keine Elasticsearch-Kenntnisse erforderlich ;-)

Weitere Informationen finden Sie hier: https://github.com/ankane/searchkick

Elasticsearch online lesen: https://riptutorial.com/de/ruby-on-rails/topic/6500/elasticsearch

Kapitel 35: Fabrikmädchen

Examples

Fabriken definieren

Wenn Sie über eine ActiveRecord-Benutzerklasse mit Namen und E-Mail-Attributen verfügen, können Sie eine Factory dafür erstellen, indem Sie FactoryGirl dazu raten lassen:

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

Oder Sie können es explizit machen und sogar seinen Namen ändern:

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

Dann können Sie in Ihrer Spezifikation die FactoryGirl-Methoden wie folgt verwenden:

```
# 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)
```

Die gebräuchlichsten Methoden sind:

```
# 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)
```

Fabrikmädchen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/8330/fabrikmadchen

Kapitel 36: Flaches Routing

Examples

1. Verwendung von Flachwasser

Eine Möglichkeit, eine tiefe Verschachtelung (wie oben empfohlen) zu vermeiden, besteht darin, die unter dem übergeordneten Bereich angegebenen Auflistungsaktionen zu generieren, um ein Gefühl für die Hierarchie zu erhalten, die Elementaktionen jedoch nicht zu verschachteln. Mit anderen Worten, nur Routen mit einer minimalen Menge an Informationen erstellen, um die Ressource eindeutig zu identifizieren:

```
resources :articles, shallow: true do
  resources :comments
  resources :quotes
  resources :drafts
end
```

Die flache Methode des DSL schafft einen Bereich, in dem jede Schachtelung flach ist. Dadurch werden die gleichen Routen wie im vorherigen Beispiel generiert:

```
shallow do
  resources :articles do
    resources :comments
    resources :quotes
    resources :drafts
  end
end
```

Es gibt zwei Optionen für den Umfang, um flache Routen anzupassen. : shallow_path stellt Mitgliedspfade den angegebenen Parameter voran:

```
scope shallow_path: "sekret" do
  resources :articles do
    resources :comments, shallow: true
  end
end
```

Verwenden Sie den Rake-Befehl, um generierte Routen wie folgt zu erhalten:

```
rake routes
```

Flaches Routing online lesen: https://riptutorial.com/de/ruby-on-rails/topic/7775/flaches-routing

Kapitel 37: Freundliche ID

Einführung

FriendlyId ist der "Swiss Army Bulldozer" von Plugins für Plug-Ins und Permalink für Active Record. Sie können hübsche URLs erstellen und mit benutzerfreundlichen Zeichenfolgen arbeiten, als wären sie numerische IDs. Mit FriendlyId können Sie ganz einfach festlegen, dass in Ihrer Anwendung URLs verwendet werden:

http://example.com/states/washington

Examples

Rails Schnellstart

```
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
```

Bearbeiten Sie App / Modelle / Benutzer.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')
```

Eine neue "Kandidaten" -Funktionalität, die es einfach macht, eine Liste alternativer Slugs zu erstellen, die zur eindeutigen Unterscheidung von Datensätzen verwendet werden können, anstatt eine Sequenz anzuhängen. Zum Beispiel:

```
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
```

Slug-Grenzlänge mit friendly_id gem einstellen?

```
def normalize_friendly_id(string)
   super[0..40]
end
```

Freundliche ID online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9664/freundliche-id

Kapitel 38: Garnelen-PDF

Examples

Erweitertes Beispiel

Dies ist der fortgeschrittene Ansatz mit Beispiel

```
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
```

Ich oben genannten Code haben wir diese Zeile `FundsPdf.new(@fund, view_context)` . Hier initialisieren wir die Klasse FundsPdf mit der Instanz @fund und view_context, um Hilfsmethoden in FundsPdf zu verwenden. FundsPdf würde so aussehen

```
 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
```

Dies ist eine der besten Methoden, um PDFs mit Klassen unter Verwendung von Prawn-Edelsteinen zu erzeugen.

Basisbeispiel

Sie müssen Gem und PDF MIME hinzufügen: Geben Sie mime_types.rb ein, da wir die Rails über den PDF-Mime-Typ informieren müssen.

Danach können wir Pdf mit Prawn auf folgende grundlegende Arten erzeugen

Dies ist die Grundaufgabe

```
pdf = Prawn::Document.new
pdf.text "Hello World"
pdf.render_file "assignment.pdf"
```

Wir können das mit Implicit Block tun

```
Prawn::Document.generate("implicit.pdf") do
 text "Hello World"
end
```

Mit explizitem Block

```
Prawn::Document.generate("explicit.pdf") do |pdf|
 pdf.text "Hello World"
end
```

Garnelen-PDF online lesen: https://riptutorial.com/de/ruby-on-rails/topic/4163/garnelen-pdf

Kapitel 39: GoogleMaps mit Rails verwenden

Examples

Fügen Sie den Javascript-Tag von Google Maps zum Layout-Header hinzu

Damit Google Maps ordnungsgemäß mit Turbolinks funktioniert, fügen Sie das Javascript-Tag direkt zum Layout-Header hinzu, anstatt es in eine Ansicht aufzunehmen.

```
# app/views/layouts/my_layout.html.haml
!!!
%html{:lang => 'en'}
  %head
    - # ...
    = google_maps_api_script_tag
```

Der `google_maps_api_script_tag` sich am besten in einem Helfer definieren.

```
# 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
```

Sie können Ihre Anwendung bei Google registrieren und Ihren API-Schlüssel in der Google API-Konsole abrufen . Google hat eine kurze Anleitung, wie Sie einen API-Schlüssel für die Google Maps-JavaScript-API anfordern .

Der API-Schlüssel wird in der Datei `secrets.yml` gespeichert:

```
# config/secrets.yml
development:
  google_maps_api_key: '...'
  # ...
production:
  google_maps_api_key: '...'
  # ...
```

Vergessen Sie nicht, `config/secrets.yml` zu Ihrer `.gitignore` Datei `.gitignore` und sicherzustellen, dass Sie den API-Schlüssel nicht in das Repository einbinden.

Geokodieren Sie das Modell

Angenommen, Ihre Benutzer und / oder Gruppen verfügen über Profile, und Sie möchten Adressprofilfelder auf einer Google-Map anzeigen.

```
# 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
```

Eine gute Möglichkeit, die Adressen zu geokodieren, dh `longitude` und `latitude` anzugeben, ist der Geocoder-Edelstein .

Fügen Sie Geocoder zu Ihrem `Gemfile` und führen Sie ein `bundle` , um es zu installieren.

```
# Gemfile
gem 'geocoder', '~> 1.3'
```

Fügen Sie Datenbankspalten für `latitude` und `longitude` , um den Standort in der Datenbank zu speichern. Dies ist effizienter, als den Geokodierungsdienst jedes Mal abzufragen, wenn Sie den Standort benötigen. Es ist schneller und Sie erreichen das Abfragelimit nicht so schnell.

```
➜ 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
```

Fügen Sie Ihrem Modell den Geokodierungsmechanismus hinzu. In diesem Beispiel wird die Adresszeichenfolge im `value` Attribut gespeichert. Konfigurieren Sie die Geokodierung so, dass sie ausgeführt wird, wenn sich der Datensatz geändert hat und nur ein Wert vorhanden ist:

```
# 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
```

Geocoder verwendet standardmäßig Google als Suchdienst. Es hat viele interessante Features wie Entfernungsberechnungen oder Annäherungssuche. Weitere Informationen finden Sie in der README des Geocoders .

Zeigen Sie Adressen auf einer Google-Karte in der Profilansicht an

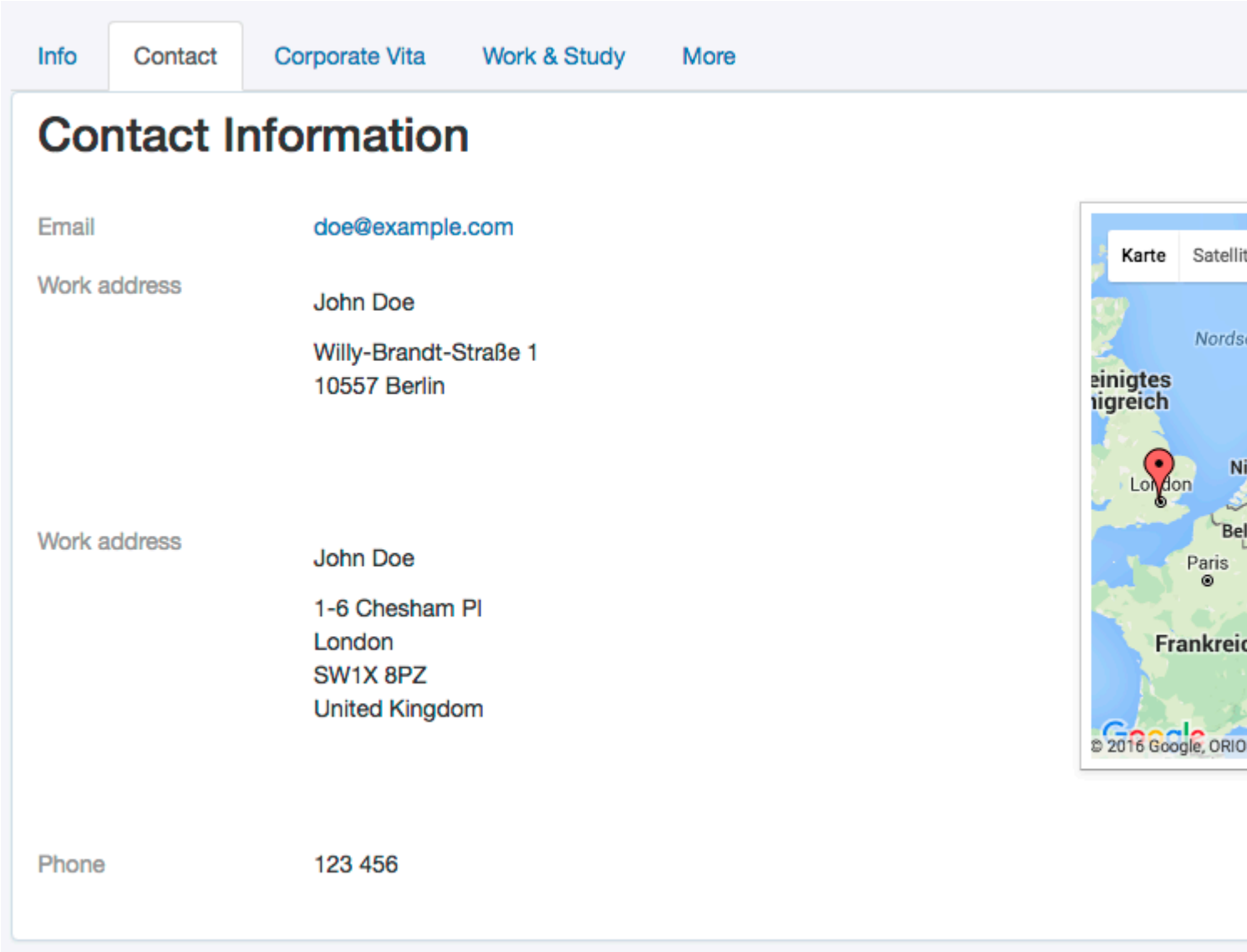
Zeigen Sie in der Profilansicht die Profilfelder eines Benutzers oder einer Gruppe in einer Liste sowie die Adressfelder in einer Google-Map an.

```
- # app/views/profiles/show.html.haml
%h1 Contact Information
.profile_fields
  = render @profile_fields
.google_map{data: address_fields: @address_fields.to_json }
```

Die entsprechenden `@profile_fields` und `@address_fields` werden im Controller festgelegt:

```
# 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
```

Initialisieren Sie die Karte, setzen Sie die Markierungen, stellen Sie den Zoom und andere Karteneinstellungen mit Javascript ein.



Setzen Sie die Markierungen auf der Karte mit Javascript

Nehmen wir an , es gibt eine `.google_map` div, die die Karte mit sich, und die hat die Adressfelder als Marker zu zeigen , `data`

Zum Beispiel:

```
<!-- 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>
```

Um das `$(document).ready` mit Turbolinks zu nutzen, ohne die Turbolinks-Ereignisse von Hand zu verwalten, verwenden Sie den Edelstein jquery.turbolinks .

Wenn Sie später noch andere Vorgänge mit der Karte ausführen möchten, z. B. Filter- oder Infofenster, ist es sinnvoll, die Karte von einer Kaffeeskriptklasse verwalten zu lassen.

```
# 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
```

Wenn Sie mehrere Kaffeeskriptdateien verwenden, die standardmäßig einen Namensraum haben, ist es zweckmäßig, einen globalen `App` Namensraum zu definieren, der von allen Kaffeeskriptdateien gemeinsam genutzt wird.

Dann durchlaufen Sie (möglicherweise mehrere) `.google_map` und erstellen `App.GoogleMap` für jede einzelne eine Instanz der `App.GoogleMap` Klasse.

```
# 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
```

Initialisieren Sie die Karte mit einer Kaffeeskriptklasse.

Eine bereitgestellt `App.GoogleMap` Kaffee Script - Klasse kann die Google - Karte wie folgt initialisiert werden:

```
# 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
  }
```

Weitere `map_configuration` zu den möglichen Optionen für map_configuration finden Sie in der Google MapOptions-Dokumentation und in deren Anleitung zum Hinzufügen von Steuerelementen .

Als Referenz ist die Klasse `google.maps.Map` hier ausführlich dokumentiert .

Initialisieren Sie die Kartenmarkierungen mithilfe einer Kaffeeskriptklasse

GEWäHR `App.GoogleMap` Kaffee Skript - `data-address-fields` `.google_map` Klasse und die Markierungsinformation in dem gespeichert werden `data-address-fields` Attribut des `.google_map` div können die Kartenmarkierungen auf der Karte wie folgt initialisiert werden:

```
# 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
```

Weitere Informationen zu Markierungsoptionen finden Sie in der Google MarkerOptions-Dokumentation und in deren Leitfaden zu Markern .

Auto-Zoom einer Karte mit einer Kaffeeskriptklasse

Eine bereitgestellt `App.GoogleMap` Kaffee Skript - `google.maps.Map @map google.maps.Marker @markers` Klasse mit dem `google.maps.Map` als gespeicherte `@map` und die `google.maps.Marker` s als gespeicherte `@markers` , die Karte kann automatisch gezoomt, dh eingestellt sein , dass alle Markierungen sichtbar sind, wie dies : auf der Karte wie folgt:

```
# 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
```

Weitere Informationen zu Grenzen finden Sie in der LatLngBounds-Dokumentation von Google .

Darstellen der Modelleigenschaften als Json

Um Adressprofilfelder als Markierungen auf einer Google-Map anzuzeigen, müssen die Adressfeldobjekte als Json-Objekte an Javascript übergeben werden.

Regelmäßige Datenbankattribute

Beim Aufruf von `to_json` für ein `ApplicationRecord` Objekt werden die Datenbankattribute automatisch `to_json` .

Wenn ein `ProfileFields::Address` Modell mit Attributen für `label` , `value` , `longitude` und `latitude` , führt `address_field.as_json` zu einem `Hash` , z.

```
address_field.as_json  # =>
  {label: "Work address", value: "Willy-Brandt-Straße 1\n10557 Berlin",
    longitude: ..., latitude: ...}
```

welches von `to_json` in einen json-String `to_json` :

```
address_field.to_json  # =>
  "{\"label\":\"Work address\",\"value\":\"Willy-Brandt-Straße 1\\n
    10557 Berlin\",\"longitude\":...,\"latitude\":...}"
```

Dies ist nützlich, da Sie später in Javascript `label` und `value` , um beispielsweise Tooltipps für die Kartenmarkierungen anzuzeigen.

Andere Attribute

Andere virtuelle Attribute können durch Überschreiben der `as_json` Methode `as_json` werden.

Um beispielsweise ein `title` Attribut `as_json` zu machen, `as_json` es in den zusammengeführten `as_json` Hash ein:

```
# 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
```

Das obige Beispiel verwendet `super` , um die ursprüngliche `as_json` Methode `as_json` , die den ursprünglichen Attributhash des Objekts zurückgibt und ihn mit dem erforderlichen Positionshash zusammenführt.

Um den Unterschied zwischen `as_json` und `to_json` zu verstehen, `as_json` `to_json` einen Blick auf diesen Blogbeitrag von jjulian .

Position

Um Marker zu rendern, erfordert die Google Maps-API standardmäßig einen `position` der Längen- und Breitengrad als `lng` bzw. `lat` .

Dieser Positionshash kann in Javascript erstellt werden, später oder hier, wenn Sie die Json-Darstellung des Adressfelds definieren:

Um diese `position` als json-Attribut des Adressfelds `as_json` , überschreiben `as_json` einfach die `as_json` Methode im Modell.

```
# 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
```

GoogleMaps mit Rails verwenden online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2828/googlemaps-mit-rails-verwenden

Kapitel 40: Helfer bilden

Einführung

Rails bietet Ansichtshilfen zum Generieren von Formularmarkierungen.

Bemerkungen

- Die `datetime` einschließlich `date` , `date` und `datetime` , `datetime-local` und `time` sowie `month` und `week` funktionieren in FireFox nicht.
- `input<type="telephone">` funktioniert nur mit Safari 8.
- `input<type="email">` funktioniert nicht in Safari

Examples

Erstellen Sie ein Formular

Sie können ein Formular mit dem Helfer `form_tag` erstellen

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

Dadurch wird folgender HTML-Code erstellt

```
<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>
```

Dieses Formular-Tag hat ein `hidden` Eingabefeld erstellt. Dies ist notwendig, da Formulare ohne diese nicht erfolgreich übermittelt werden können.

Das zweite Eingabefeld mit dem Namen " `authenticity_token` bietet zusätzlichen Schutz vor `cross-site request forgery` .

Suchformular erstellen

Um ein Suchformular zu erstellen, geben Sie den folgenden Code ein

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

- `form_tag` : Dies ist der Standard-Helfer zum Erstellen eines Formulars. Der erste Parameter ist `/search` ist die Aktion und der zweite Parameter gibt die HTTP-Methode an. Für Suchformulare ist es wichtig, immer die Methode `get`
- `label_tag` : Dieser Helfer erstellt ein html `<label>` -Tag.
- `text_field_tag` : Dadurch wird ein Eingabeelement mit dem Typ `text`
- `submit_tag` : Dies erzeugt ein Eingabeelement mit Typ `submit`

Helfer für Formularelemente

Ankreuzfelder

```
<%= 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") %>
```

Dadurch wird das folgende HTML generiert

```
<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>
```

Radio Knöpfe

```
<%= 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") %>
```

Dies erzeugt den folgenden HTML-Code

```
<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>
```

Textbereich

Um ein größeres Textfeld zu erstellen, wird empfohlen, den `text_area_tag`

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

Dadurch wird der folgende HTML-Code erstellt

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

Zahlenfeld

Dadurch wird ein `input<type="number">`

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

Um einen Wertebereich anzugeben, können Sie die Option `in:`

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

Passwortfeld

Manchmal möchten Sie, dass die vom Benutzer eingegebenen Zeichen maskiert werden. Dadurch wird ein `<input type="password">` generiert

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

E-Mail-Feld

Dadurch wird eine `<input type="email">`

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

Telefonfeld

Dadurch wird ein `<input type="tel">` .

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

Datum Helfer

- `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) %>
```

Dropdown-Liste

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

Dadurch wird der folgende HTML-Code generiert: David

Das letzte Argument sind Optionen, die Folgendes akzeptieren: {multiple: false, disabled: false, include_blank: false, Eingabeaufforderung: false}

Weitere Beispiele finden Sie unter:
http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select_tag

Helfer bilden online lesen: https://riptutorial.com/de/ruby-on-rails/topic/4509/helfer-bilden

Kapitel 41: Hinzufügen eines Amazon RDS zu Ihrer Rails-Anwendung

Einführung

Schritte zum Erstellen einer AWS RDS-Instanz und zum Konfigurieren der Datei database.yml durch Installieren der erforderlichen Connectors.

Examples

Stellen Sie sich vor, wir verbinden MYSQL RDS mit Ihrer Schienenanwendung.

Schritte zum Erstellen der MYSQL-Datenbank

1. Melden Sie sich bei Amazon an und wählen Sie den RDS-Dienst
2. Wählen Sie auf der Registerkarte "Instanz" die Option " `Launch DB Instance` starten" aus
3. Standardmäßig wird MYSQL Community Edition ausgewählt. Klicken Sie daher auf die `select`
4. Wählen Sie den Datenbankzweck aus, sagen Sie `production` und klicken Sie auf den `next step`
5. Geben Sie die `mysql version, storage size, DB Instance Identifier, Master Username and Password` und klicken Sie auf den `next step`
6. Geben Sie den `Database Name` und klicken Sie auf `Launch DB Instance`
7. Bitte warten Sie, bis die gesamte Instanz erstellt wurde. Sobald die Instanz erstellt wurde, finden Sie einen Endpunkt. Kopieren Sie diesen Einstiegspunkt (der als Hostname bezeichnet wird).

Anschlüsse installieren

Fügen Sie den MySQL-Datenbankadapter der gemfile Ihres Projekts hinzu.

```
gem 'mysql2'
```

Installiere deine hinzugefügten Edelsteine,

```
bundle install
```

Einige andere Datenbankadapter sind

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

Konfigurieren Sie die Datei database.yml Ihres Projekts. Öffnen Sie **die Datei** 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 entrypoint
  port: <%= RDS_PORT %>         # db post. For MYSQL 3306
```

Hinzufügen eines Amazon RDS zu Ihrer Rails-Anwendung online lesen: https://riptutorial.com/de/ruby-on-rails/topic/10922/hinzufugen-eines-amazon-rds-zu-ihrer-rails-anwendung

Kapitel 42: I18n - Internationalisierung

Syntax

- I18n.t ("Schlüssel")
- I18n.translate ("key") # entspricht `I18n.t("key")`
- I18n.t ("Schlüssel", Anzahl: 4)
- I18n.t ("key", param1: "Something", param2: "Else")
- I18n.t ("doesnt_exist", Standard: "Schlüssel") # Gibt einen Standardwert an, wenn der Schlüssel fehlt
- I18n.locale # =>: en
- I18n.locale =: en
- I18n.default_locale # =>: de
- I18n.default_locale =: en
- t (". key") # Wie `I18n.t("key")` , jedoch auf die Aktion / Vorlage, von der es aufgerufen wird

Examples

Verwenden Sie I18n in Ansichten

Angenommen, Sie haben diese YAML-Sprachdatei:

```
# config/locales/en.yml
en:
  header:
    title: "My header title"
```

Wenn Sie Ihre Titelzeichenfolge anzeigen möchten, können Sie dies tun

```
# in ERB files
<%= t('header.title') %>

# in SLIM files
= t('header.title')
```

I18n mit Argumenten

Sie können Parameter **I18n** passieren `t` - Methode:

```
# 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) %>
```

Und erhalte folgende Ausgabe:

```
"12 users currently online"
```

Pluralisierung

Sie können **I18n für** die Pluralisierung verantwortlich machen, verwenden Sie einfach das Argument `count` .

Sie müssen Ihre Ländereinstellungsdatei folgendermaßen einrichten:

```
# config/locales/en.yml
en:
  online_users:
    one: "1 user is online"
    other: "%{count} users are online"
```

Verwenden Sie dann den soeben erstellten Schlüssel, indem Sie das `count` Argument an `I18n.t` Helfer `I18n.t` :

```
I18n.t("online_users", count: 1)
#=> "1 user is online"

I18n.t("online_users", count: 4)
#=> "4 users are online"
```

Festlegen des Gebietsschemas durch Anforderungen

In den meisten Fällen möchten Sie möglicherweise das Gebietsschema **`I18n`** . Möglicherweise möchten Sie das Gebietsschema für die aktuelle Sitzung, den aktuellen Benutzer oder basierend auf einem URL-Parameter `before_action` Dies kann leicht durch Implementierung einer `before_action` in einem Ihrer Controller oder in `ApplicationController` , um es in allen Controllern zu haben.

```
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
```

URL-basiert

Das `locale` param könnte von einer solchen URL stammen

```
http://yourapplication.com/products?locale=en
```

Oder

```
http://yourapplication.com/en/products
```

Um letzteres zu erreichen, müssen Sie Ihre `routes` bearbeiten und einen `scope` hinzufügen:

```
# config/routes.rb
scope "(:locale)", locale: /en|fr/ do
  resources :products
end
```

Durch das `http://yourapplication.com/en/products` wird Ihr Gebietsschema auf `http://yourapplication.com/en/products` festgelegt `:en` . `http://yourapplication.com/fr/products` Sie stattdessen `http://yourapplication.com/fr/products` besuchen, wird `http://yourapplication.com/fr/products` festgelegt `:fr` . Außerdem erhalten Sie keinen Routing-Fehler, wenn Sie das `:locale` -Param fehlen. Beim Besuch von `http://yourapplication.com/products` wird das Standard- **I18n-** Gebietsschema **geladen** .

Sitzungsbasiert oder persistenzbasiert

Dies setzt voraus, dass der Benutzer auf eine Schaltfläche / eine Sprachmarkierung klicken kann, um die Sprache zu ändern. Die Aktion kann an einen Controller weiterleiten, der die Sitzung auf die aktuelle Sprache einstellt (und die Änderungen an einer Datenbank eventuell beibehalten, wenn der Benutzer verbunden ist).

```
class SetLanguageController < ApplicationController
  skip_before_filter :authenticate_user!
  after_action :set_preferred_locale

  # Generic version to handle a large list of languages
  def change_locale
    I18n.locale = sanitize_language_param
    set_session_and_redirect
  end
```

> Sie müssen sanitize_language_param mit Ihrer Liste der verfügbaren Sprachen definieren und eventuell Fehler behandeln, falls die Sprache nicht vorhanden ist.

Wenn Sie nur wenige Sprachen haben, können Sie sie stattdessen wie folgt definieren:

```
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
```

Hinweis: Vergessen Sie nicht, Ihren `change_language` *Aktionen einige Routen* `change_language`

Standardgebietsschema

Denken Sie daran, dass Sie das Standardgebietsschema der Anwendung festlegen müssen. Sie können dies tun, indem Sie es entweder in `config/application.rb` einstellen:

```
config.i18n.default_locale = :de
```

oder durch Erstellen eines Initialisierers im Ordner `config/initializers` :

```
# config/initializers/locale.rb
I18n.default_locale = :it
```

Holen Sie sich das Gebietsschema von der HTTP-Anforderung

Manchmal kann es nützlich sein, das Anwendungsgebietsschema basierend auf der Anforderungs-IP festzulegen. Sie können dies leicht mit **`Geocoder`** . **`Geocoder`** kann unter anderem die `location` einer `request` **`Geocoder`** .

`Geocoder` Sie `Gemfile` **`Geocoder`** zu Ihrem `Gemfile`

```
# Gemfile
gem 'geocoder'
```

`Geocoder` fügt dem standardmäßigen `Rack::Request` Objekt `location` und `safe_location` Methoden hinzu, sodass Sie den Speicherort jeder HTTP-Anfrage nach IP-Adresse ermitteln können. Sie können diese Methoden in einer `before_action` in Ihrem `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
```

Beachten Sie, dass dies in `development` und `test` nicht funktioniert, da Dinge wie `0.0.0.0` und `localhost` gültige Internet-IP-Adressen sind.

Einschränkungen und Alternativen

`Geocoder` ist sehr leistungsfähig und flexibel, muss jedoch für die Arbeit mit einem *Geokodierungsdienst* konfiguriert *werden* (weitere Informationen). viele davon setzen der Nutzung Grenzen. Beachten Sie auch, dass das Anrufen eines externen Dienstes bei jeder Anforderung die Leistung beeinträchtigen kann.

Um diese anzusprechen, kann es auch eine Überlegung wert sein:

1. Eine Offline-Lösung

Bei Verwendung eines **`GeoIP`** wie **`GeoIP`** (siehe hier) können Lookups anhand einer lokalen Datei durchgeführt werden. In Bezug auf die Genauigkeit kann es zu einem Kompromiss kommen, da diese Dateien aktuell gehalten werden müssen.

2. Verwenden Sie CloudFlare

Über CloudFlare `HTTP_CF_IPCOUNTRY` Seiten können transparent geokodiert werden, wobei der Ländercode zum Header hinzugefügt wird (`HTTP_CF_IPCOUNTRY`). Weitere Details finden Sie hier .

Attribute des ActiveRecord-Modells übersetzen

`globalize` gem ist eine großartige Lösung, um Ihren `ActiveRecord` Modellen Übersetzungen hinzuzufügen. Sie können es installieren, indem Sie dies Ihrem `Gemfile` :

```
gem 'globalize', '~> 5.0.0'
```

Wenn Sie `Rails 5` Sie auch `activemodel-serializers-xml` hinzufügen

```
gem 'activemodel-serializers-xml'
```

Mit Modellübersetzungen können Sie die Attributwerte Ihrer Modelle übersetzen, zum Beispiel:

```
class Post < ActiveRecord::Base
  translates :title, :text
end

I18n.locale = :en
post.title # => Globalize rocks!

I18n.locale = :he
post.title # => גלובאלייז 2 טלוש!
```

Nachdem Sie Ihre Modellattribute definiert haben, die übersetzt werden müssen, müssen Sie über eine Migration eine Übersetzungstabelle erstellen. **`globalize`** bietet `create_translation_table!` und `drop_translation_table!` .

Für diese Migration müssen Sie `up` und `down` und **nicht** `change` . Damit diese Migration erfolgreich ausgeführt werden kann, müssen Sie zunächst die übersetzten Attribute in Ihrem Modell definieren, wie oben gezeigt. Eine korrekte Migration für das vorherige `Post` Modell lautet:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table! title: :string, text: :text
  end

  def down
    Post.drop_translation_table!
  end
end
```

Sie können auch Optionen für bestimmte Optionen übergeben, z.

```
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
```

Falls Sie bereits über **vorhandene Daten** in den benötigten Übersetzungsspalten verfügen, können Sie diese leicht in die Übersetzungstabelle migrieren, indem Sie Ihre Migration anpassen:

```
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
```

Stellen Sie sicher, dass Sie die übersetzten Spalten aus der übergeordneten Tabelle löschen, nachdem alle Ihre Daten sicher migriert wurden. Um die übersetzten Spalten nach der Datenmigration automatisch aus der übergeordneten Tabelle zu entfernen, fügen `remove_source_columns` der Migration die Option `remove_source_columns` :

```
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
```

Sie können auch neue Felder zu einer zuvor erstellten Übersetzungstabelle hinzufügen:

```
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
```

Verwenden Sie I18n mit HTML-Tags und -Symbolen

```
# config/locales/en.yml
en:
  stackoverflow:
    header:
      title_html: "Use <strong>I18n</strong> with Tags &amp; Symbols"
```

Beachten Sie den Zusatz von zusätzlicher `_html` nach dem Namen des `title` .

Und in Ansichten,

```
# ERB
<h2><%= t(:title_html, scope: [:stackoverflow, :header]) %></h2>
```

I18n - Internationalisierung online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2772/i18n---internationalisierung

Kapitel 43: Importieren Sie ganze CSV-Dateien aus einem bestimmten Ordner

Einführung

Nehmen wir an, wir haben in diesem Beispiel viele Produkt-CSV-Dateien in einem Ordner. Jede CSV-Datei muss von unserer Konsole aus unsere Datenbank hochladen und einen Befehl schreiben. Führen Sie den folgenden Befehl in einem neuen oder vorhandenen Projekt aus, um dieses Modell zu erstellen.

Examples

Lädt CSV vom Konsolenbefehl hoch

Terminalbefehle:

```
rails g model Product name:string quantity:integer price:decimal{12,2}
rake db:migrate
```

Lates erstellen Controller.

Terminalbefehle:

```
rails g controller Products
```

Controller Code:

```
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
```

Modell:

```
class Product< ApplicationRecord
  def self.import(file)
      CSV.foreach(file.path, headers: true) do |row|
          Product.create! row.to_hash
      end
  end
end
```

routen.rb

```
resources :products
```

app / config / application.rb

```
require 'csv'
```

Öffnen Sie nun Ihre Entwicklung `console` & `run`

```
=> ProductsController.new.create #=> Uploads your whole CSV files from your folder directory
```

Importieren Sie ganze CSV-Dateien aus einem bestimmten Ordner online lesen: https://riptutorial.com/de/ruby-on-rails/topic/8658/importieren-sie-ganze-csv-dateien-aus-einem-bestimmten-ordner

Kapitel 44: Klassenorganisation

Bemerkungen

Dies scheint eine einfache Sache zu sein, aber wenn der Unterricht anfängt, in der Größe zu steigen, sind Sie dankbar, dass Sie sich die Zeit genommen haben, um sie zu organisieren.

Examples

Modellklasse

```
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
```

Modelle sind typischerweise verantwortlich für:

- Beziehungen aufbauen
- Daten validieren
- Bereitstellung des Zugriffs auf Daten über Geltungsbereiche und Methoden
- Durchführen von Aktionen zur Persistenz von Daten.

Auf höchster Ebene beschreiben Modelle Domänenkonzepte und verwalten deren Persistenz.

Serviceklasse

Controller ist ein Einstiegspunkt in unsere Anwendung. Es ist jedoch nicht der einzige mögliche Einstiegspunkt. Ich hätte gerne meine Logik von:

- Rechenaufgaben
- Hintergrundjobs
- Konsole
- Tests

Wenn ich meine Logik in eine Steuerung stecke, ist sie von all diesen Stellen nicht zugänglich. Versuchen wir also, den Ansatz „Skinny Controller, Fat Model“ auszuprobieren und die Logik auf ein Modell zu übertragen. Aber welcher? Wenn eine bestimmte Logik ein `User` , `Cart` und `Product` - wo sollte sie leben?

Eine Klasse, die von `ActiveRecord::Base` erbt, hat bereits viele Verantwortlichkeiten. Es behandelt die Abfrage-Schnittstelle, Zuordnungen und Validierungen. Wenn Sie Ihrem Modell noch mehr Code hinzufügen, wird es mit Hunderten öffentlicher Methoden schnell zu einem unauffälligen Durcheinander.

Ein Service ist nur ein reguläres Ruby-Objekt. Ihre Klasse muss nicht von einer bestimmten Klasse erben. Sein Name ist eine Verbphrase, z. B. `CreateUserAccount` anstelle von `UserCreation` oder `UserCreationService` . Es lebt im App / Services-Verzeichnis. Sie müssen dieses Verzeichnis selbst erstellen, Rails lädt jedoch automatisch Klassen für Sie.

Ein Serviceobjekt macht eine Sache

Ein Serviceobjekt (alias Methodenobjekt) führt eine Aktion aus. Es enthält die Geschäftslogik, um diese Aktion auszuführen. Hier ist ein Beispiel:

```
# app/services/accept_invite.rb
class AcceptInvite
  def self.call(invite, user)
    invite.accept!(user)
    UserMailer.invite_accepted(invite).deliver
  end
end
```

Die drei Konventionen, die ich befolge, sind:

Services gehen in das `app/services directory` . Ich empfehle Ihnen, Unterverzeichnisse für geschäftslogikreiche Domains zu verwenden. Zum Beispiel:

- Die Datei `app/services/invite/accept.rb` definiert `Invite::Accept` während `app/services/invite/create.rb` `Invite::Create` definiert
- Services beginnen mit einem Verb (und enden nicht mit Service): `ApproveTransaction` , `SendTestNewsletter` , `ImportUsersFromCsv`
- Dienste reagieren auf die `call` . Ich fand es mit einem anderen Verb etwas überflüssig: `ApproveTransaction.approve()` liest nicht gut. Auch der `call` ist Methode der de - facto - Methode für `lambda` , `procs` und Methodenobjekte.

Leistungen

Serviceobjekte zeigen, was meine Anwendung macht

Ich kann nur einen Blick auf das `ApproveTransaction` `CancelTransaction` , `BlockAccount` zu sehen, was meine Anwendung bewirkt: `ApproveTransaction` , `CancelTransaction` , `BlockAccount` , `SendTransactionApprovalReminder` ...

Ein kurzer Blick auf ein Serviceobjekt und ich weiß, welche Geschäftslogik involviert ist. Ich muss nicht die Controller, `ActiveRecord` Modellrückrufe und Beobachter durchgehen, um zu verstehen, was das "Genehmigen einer Transaktion" bedeutet.

Aufräummodelle und Controller

Controller wandeln die Anfrage (Parameter, Sitzung, Cookies) in Argumente um, leiten sie an den Dienst weiter und leiten sie entsprechend der Antwort des Dienstes um oder rendern sie.

```
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
```

Modelle behandeln nur Assoziationen, Gültigkeitsbereiche, Validierungen und Persistenz.

```
class Invite < ActiveRecord::Base
  def accept!(user, time=Time.now)
    update_attributes!(
      accepted_by_user_id: user.id,
      accepted_at: time
    )
  end
end
```

Dies macht Modelle und Controller viel einfacher zu testen und zu warten!

Wann wird die Serviceklasse verwendet?

Reichweite für Serviceobjekte, wenn eine Aktion eines oder mehrere der folgenden Kriterien erfüllt:

- Die Aktion ist komplex (zB das Schließen der Bücher am Ende einer Abrechnungsperiode)
- Die Aktion erstreckt sich über mehrere Modelle (z. B. einen E-Commerce-Kauf mit Aufträgen, Kreditkarten und Kundenobjekten).
- Die Aktion interagiert mit einem externen Dienst (z. B. Veröffentlichung in sozialen Netzwerken).
- Die Aktion ist kein zentrales Anliegen des zugrunde liegenden Modells (z. B. das Aufholen

veralteter Daten nach einem bestimmten Zeitraum).
- Es gibt mehrere Möglichkeiten, die Aktion auszuführen (z. B. Authentifizierung mit einem Zugriffstoken oder Kennwort).

Quellen

Adam Niedzielski Blog

Brew House-Blog

Code Climate Blog

Klassenorganisation online lesen: https://riptutorial.com/de/ruby-on-rails/topic/7623/klassenorganisation

Kapitel 45: Mehrzweck-ActiveRecord-Spalten

Syntax

- `serialize: <field_plural_symbol>`

Examples

Objekt speichern

Wenn Sie über ein Attribut verfügen, das als Objekt gespeichert und in der Datenbank abgerufen werden muss, geben Sie den Namen dieses Attributs mit der `serialize` Methode an. Das Attribut wird automatisch verarbeitet.

Das Attribut muss als `text` deklariert werden.

Im Modell müssen Sie den Typ des Feldes (`Hash` oder `Array`) `Array`

Mehr Infos unter: serialize >> apidock.com

Wie man

In deiner Migration

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

In deinem Modell

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

Mehrzweck-ActiveRecord-Spalten online lesen: https://riptutorial.com/de/ruby-on-rails/topic/7602/mehrzweck-activerecord-spalten

Kapitel 46: Modellzustände: AASM

Examples

Grundzustand bei AASM

Normalerweise erstellen Sie am Ende Modelle, die einen Zustand enthalten, und dieser Zustand ändert sich während der Lebensdauer des Objekts.

AASM ist eine Finite-State-Machine-Enabler-Bibliothek, die Ihnen beim einfachen Durcharbeiten des Prozessdesigns Ihrer Objekte helfen kann.

Wenn Sie so etwas in Ihrem Modell haben, stimmt das mit der Idee von Fat Model, Skinny Controller , einer der besten Methoden von Rails, überein. Das Modell ist allein verantwortlich für die Verwaltung seines Zustands, seiner Änderungen und für das Generieren der durch diese Änderungen ausgelösten Ereignisse.

Zu installieren, in Gemfile

```
gem 'aasm'
```

Betrachten Sie eine App, in der der Benutzer ein Produkt zu einem Preis anbietet.

```
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

  private

  def set_payment_date
    update payed_at: Time.zone.now
  end
end
```

Die Statusklassen der Quote-Klasse können jedoch verwendet werden. Dies ist jedoch am besten für Ihren Prozess.

Sie können sich die Zustände als Vergangenheit vorstellen, wie im vorherigen Beispiel oder auch in einer anderen Form, zum Beispiel: Preise, Zahlung, Lieferung usw. Die Benennung der Zustände hängt von Ihnen ab. Aus persönlicher Sicht funktionieren vergangene Zustände besser, da Ihr Endzustand sicherlich eine vergangene Aktion ist und eine bessere Verknüpfung mit den Ereignisnamen herstellt, was später erläutert wird.

HINWEIS: Achten Sie darauf , welche Namen Sie verwenden, müssen Sie sich Sorgen über nicht mit Ruby oder Ruby on Rails reservierten Schlüsselwörter, wie `valid` , `end` , `being` usw.

Nachdem wir die Zustände und Übergänge definiert haben, können wir nun auf einige von AASM erstellte Methoden zugreifen.

Zum Beispiel:

```
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.
```

Wie Sie sehen können, dass das Ereignis Übergänge hat, bestimmen diese Übergänge, wie sich der Zustand beim Ereignisaufruf ändert. Wenn das Ereignis aufgrund des aktuellen Status ungültig ist, wird ein Fehler ausgegeben.

Die Ereignisse und Übergänge haben beispielsweise auch andere Rückrufe

```
guard: product_delivered?
```

`product_delivered?` das `product_delivered?` Methode, die einen Boolean zurückgibt. Wenn es sich als falsch herausstellt, wird der Übergang nicht angewendet. Wenn keine anderen Übergänge verfügbar sind, ändert sich der Status nicht.

```
success: :reverse_charges
```

Wenn diese Übersetzung erfolgreich durchgeführt wird, wird die `:reverse_charges` Methode aufgerufen.

Es gibt mehrere andere Methoden in AASM mit mehr Rückrufen im Prozess. Dies hilft Ihnen jedoch, Ihre ersten Modelle mit endlichen Zuständen zu erstellen.

Modellzustände: AASM online lesen: https://riptutorial.com/de/ruby-on-rails/topic/7826/modellzustande--aasm

Kapitel 47: Mongoid

Examples

Installation

Zuerst fügen Sie `Mongoid` zu Ihrem `Gemfile` :

```
gem "mongoid", "~> 4.0.0"
```

und dann `bundle install` ausführen. Oder einfach laufen:

```
$ gem install mongoid
```

Führen Sie nach der Installation den Generator aus, um die Konfigurationsdatei zu erstellen:

```
$ rails g mongoid:config
```

`(myapp)/config/mongoid.yml` wird die Datei `(myapp)/config/mongoid.yml` .

Ein Modell erstellen

Erstellen Sie ein Modell (nennen `User` es `User`), indem Sie `User` ausführen:

```
$ rails g model User
```

`app/models/user.rb` wird die Datei `app/models/user.rb` :

```
class User
  include Mongoid::Document

end
```

Das ist alles, was Sie brauchen, um ein Modell zu haben (allerdings nur ein `id` Feld). Im Gegensatz zu `ActiveRecord` gibt es keine Migrationsdateien. Alle Datenbankinformationen für das Modell sind in der Modelldatei enthalten.

Zeitstempel werden bei der Generierung nicht automatisch in Ihr Modell aufgenommen. `created_at` `updated_at` zum Hinzufügen von `created_at` und `updated_at` zu Ihrem Modell hinzu

```
include Mongoid::Timestamps
```

Zu deinem Modell darunter `include Mongoid::Document` so hinzu:

```
class User
  include Mongoid::Document
```

```
  include Mongoid::Timestamps

end
```

Felder

Gemäß der mongoiden Dokumentation gibt es 16 gültige Feldtypen:

- Array
- BigDecimal
- Boolean
- Datum
- Terminzeit
- Schweben
- Hash
- Ganze Zahl
- BSON :: ObjectId
- BSON :: Binär
- Angebot
- Regexp
- String
- Symbol
- Zeit
- TimeWithZone

Um ein Feld hinzuzufügen (nennen wir es den `name` und lassen Sie es einen `String`), fügen Sie es Ihrer Modelldatei hinzu:

```
field :name, type: String
```

Um einen Standardwert festzulegen, übergeben Sie einfach die `default` :

```
field :name, type: String, default: ""
```

Klassische Assoziationen

Mongoid erlaubt die klassischen `ActiveRecord` Assoziationen:

- Eins-zu-Eins: `has_one` / `belongs_to`
- Eins-zu-Viele: `has_many` / `belongs_to`
- Many-to-many: `has_and_belongs_to_many`

Um eine Assoziation hinzuzufügen (sagen `has_many` Beiträge des Benutzers `has_many`), können Sie diese zu Ihrer `User` hinzufügen:

```
has_many :posts
```

und dies zu deiner `Post` Modelldatei:

```
belongs_to :user
```

Dies fügt ein Feld `user_id` in Ihrem `Post` Modell hinzu, fügt Ihrer `Post` Klasse eine `user` hinzu und fügt Ihrer `User` Klasse eine `posts` Methode hinzu.

Eingebettete Assoziationen

Mongoid erlaubt eingebettete Assoziationen:

- Eins-zu-Eins: `embeds_one` / `embedded_in`
- Eins-zu-Viele: `embeds_many` / `embedded_in`

`embeds_many` Sie diese zu Ihrer `User` hinzu, um eine Zuordnung hinzuzufügen (sagen wir die `embeds_many` Adressen " `embeds_many`):

```
embeds_many :addresses
```

und dies in Ihre `Address` :

```
embedded_in :user
```

Dadurch wird `Address` in Ihr `User` eingebettet und der `User` Klasse eine `addresses` Methode hinzugefügt.

Datenbankaufrufe

Mongoid versucht , ähnliche Syntax haben `ActiveRecord` , wenn er kann. Es unterstützt diese Anrufe (und viele mehr)

```
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
```

Mongoid online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3071/mongoid

Kapitel 48: Rails 5-API-Authentifizierung

Examples

Authentifizierung mit Rails authenticate_with_http_token

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

Sie können diesen Endpunkt mit `curl` testen, indem Sie eine Anfrage wie

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

Rails 5-API-Authentifizierung online lesen: https://riptutorial.com/de/ruby-on-rails/topic/7852/rails-5-api-authentifizierung

Kapitel 49: Rails auf Docker

Einführung

Dieses Tutorial beginnt mit der Installation von Docker und einer Rails-App

Examples

Docker und Docker komponieren

Zunächst müssen wir unsere `Dockerfile` erstellen. Ein gutes Beispiel ist in diesem Blog von Nick Janetakis zu finden.

Dieser Code enthält das Skript, das zum Zeitpunkt des Starts auf unserer Docker-Maschine ausgeführt wird. Aus diesem Grund installieren wir alle erforderlichen Bibliotheken und enden mit dem Start von Puma (RoR dev-Server).

```
# 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
```

Außerdem werden wir `docker-compose.yml` . `docker-compose.yml` erstellen wir `docker-compose.yml` . Die Erklärung dieser Datei wird eher ein Docker-Compose-Tutorial als eine Integration mit Rails sein. Ich werde hier nicht weiter darauf eingehen.

```
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
```

Nur mit diesen beiden Dateien haben Sie genug, um `docker-compose up` auszuführen und Ihr Docker zu aktivieren

Rails auf Docker online lesen: https://riptutorial.com/de/ruby-on-rails/topic/10933/rails-auf-docker

Kapitel 50: Rails aufrüsten

Examples

Upgrade von Rails 4.2 auf Rails 5.0

Hinweis: Speichern Sie Ihren Code vor dem Upgrade Ihrer Rails-App immer auf einem Versionskontrollsystem wie Git.

Um ein Upgrade von Rails 4.2 auf Rails 5.0 durchzuführen, müssen Sie Ruby 2.2.2 oder neuer verwenden. Wechseln Sie nach dem Upgrade Ihrer Ruby-Version bei Bedarf zu Ihrem Gemfile und ändern Sie die Zeile:

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

zu:

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

und in der Befehlszeile ausführen:

```
$ bundle update
```

Führen Sie nun die Update-Aufgabe mit dem Befehl aus:

```
$ rake rails:update
```

Dies hilft Ihnen bei der Aktualisierung von Konfigurationsdateien. Sie werden aufgefordert, Dateien zu überschreiben, und Sie haben mehrere Eingabemöglichkeiten:

- Y - ja, überschreiben
- n - nein, nicht überschreiben
- a - alle, überschreibe dies und alle anderen
- q - beenden, abbrechen
- d - diff, zeigt die Unterschiede zwischen dem alten und dem neuen
- h - helfen

In der Regel sollten Sie die Unterschiede zwischen der alten und der neuen Datei überprüfen, um sicherzustellen, dass Sie keine unerwünschten Änderungen erhalten.

Rails 5.0 `ActiveRecord` Modelle erben von `ApplicationRecord` und nicht von `ActiveRecord::Base` . `ApplicationRecord` ist die Superklasse für alle Modelle, ähnlich wie `ApplicationController` die Superklasse für Controller ist. Um dieser neuen Art und Weise Rechnung zu tragen, wie Modelle behandelt werden, müssen Sie eine Datei mit dem Namen `application_record.rb` in Ihrem `app/models/` -Ordner `application_record.rb` und dann den Inhalt dieser Datei wie folgt bearbeiten:

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

Rails 5.0 behandelt auch Rückrufe, die etwas anders sind. Callbacks, die `false` , stoppen die Callback-Kette nicht, was bedeutet, dass nachfolgende Callbacks im Gegensatz zu Rails 4.2 weiterhin ausgeführt werden. Wenn Sie ein Upgrade durchführen, bleibt das Verhalten von Rails 4.2 erhalten. Sie können jedoch zum Verhalten von Rails 5.0 wechseln, indem Sie Folgendes hinzufügen:

```
ActiveSupport.halt_callback_chains_on_return_false = false
```

in die Datei `config/application.rb` . Sie können die Callback-Kette explizit anhalten, indem Sie `throw(:abort)` aufrufen.

In Rails 5.0 erbt `ActiveJob` von `ApplicationJob` und nicht von `ActiveJob::Base` wie in Rails 4.2. Erstellen Sie zum Upgrade auf Rails 5.0 eine Datei mit dem Namen `application_job.rb` im Ordner `app/jobs/` . Bearbeiten Sie den Inhalt dieser Datei wie folgt:

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

Anschließend müssen Sie alle Jobs so ändern, dass sie von `ApplicationJob` anstelle von `ActiveJob::Base` erben.

Eine der anderen größten Änderungen von Rails 5.0 erfordert keine Codeänderungen, ändert jedoch die Art und Weise, wie Sie die Befehlszeile mit Ihren Rails-Apps verwenden. Sie können `bin/rails` oder nur `rails` , um Aufgaben und Tests auszuführen. Anstatt `$ rake db:migrate` , können Sie jetzt `$ rails db:migrate` . Wenn Sie `$ bin/rails` ausführen, können Sie alle verfügbaren Befehle anzeigen. Beachten Sie, dass viele der Aufgaben, die jetzt mit `bin/rails` weiterhin mit `rake` funktionieren.

Rails aufrüsten online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3496/rails-aufrusten

Kapitel 51: Rails Best Practices

Examples

Wiederholen Sie sich nicht (TROCKEN)

Um den Code sauber zu halten, folgt Rails dem DRY-Prinzip.

Es geht darum, wann immer möglich, so viel Code wie möglich wiederzuverwenden, anstatt ähnlichen Code an mehreren Stellen zu duplizieren (z. B. unter Verwendung von Partials). Dies reduziert *Fehler* , hält Ihren Code *sauber* und setzt das Prinzip ein *, Code einmal zu schreiben* und anschließend wiederzuverwenden. Das Aktualisieren von Code an einer Stelle ist einfacher und effizienter als das Aktualisieren mehrerer Teile desselben Codes. Dadurch wird Ihr Code modularer und robuster.

Auch *Fat Model, Skinny Controller* ist DRY, weil Sie den Code in Ihr Modell schreiben und im Controller nur den Aufruf ausführen, z.

```
# 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
```

Dies führt auch zu einer API-gesteuerten Struktur, bei der interne Methoden verborgen werden und Änderungen durch das Übergeben von Parametern auf eine API-Art erzielt werden.

Konvention über Konfiguration

In Rails finden Sie *Controller, Ansichten und Modelle* für Ihre Datenbank.

Um den Bedarf an umfangreichen Konfigurationen zu reduzieren, implementiert Rails Regeln, um die Arbeit mit der Anwendung zu erleichtern. Sie können Ihre eigenen Regeln definieren, aber für den Anfang (und für später) ist es eine gute Idee, sich an die von Rails angebotenen Konventionen zu halten.

Diese Konventionen beschleunigen die Entwicklung, halten Ihren Code übersichtlich und lesbar

und ermöglichen Ihnen eine einfache Navigation in Ihrer Anwendung.

Konventionen senken auch die Eintrittsbarrieren für Anfänger. Es gibt so viele Konventionen in Rails, von denen ein Anfänger nicht einmal etwas wissen muss, sondern nur aus Unwissenheit Nutzen ziehen kann. Es ist möglich, großartige Anwendungen zu erstellen, ohne zu wissen, warum alles so ist, wie es ist.

Zum Beispiel

Wenn Sie über eine Datenbanktabelle mit der Primärschlüssel- `id` `orders` verfügen, wird das übereinstimmende Modell als `order` und der Controller, der die gesamte Logik verarbeitet, heißt `orders_controller` . Die Ansicht ist in verschiedene Aktionen aufgeteilt: Wenn der Controller eine `new` und eine `edit` hat, gibt es auch eine `new` und eine `edit` .

Zum Beispiel

Um eine App zu erstellen, führen Sie einfach `rails new app_name` . Auf diese Weise werden ungefähr 70 Dateien und Ordner generiert, die die Infrastruktur und Grundlage für Ihre Rails-App bilden.

Es enthält:

- Ordner für Ihre Modelle (Datenbankebene), Controller und Ansichten
- Ordner zum Halten von Komponententests für Ihre Anwendung
- Ordner zum Speichern von Web-Assets wie Javascript- und CSS-Dateien
- Standarddateien für HTTP 400-Antworten (dh Datei wurde nicht gefunden)
- Viele andere

Fettes Modell, dünner Controller

„Fat Model, Skinny Controller“ bezieht sich darauf, wie die M- und C-Teile von MVC ideal zusammenarbeiten. Jede nicht-antwortbezogene Logik sollte in das Modell aufgenommen werden, idealerweise in einer schönen, überprüfbaren Methode. Inzwischen ist der "Skinny" Controller einfach eine schöne Schnittstelle zwischen Ansicht und Modell.

In der Praxis kann dies eine Reihe verschiedener Arten von Refactoring erfordern, aber es kommt auf eine Idee: Durch das Verschieben einer beliebigen Logik, die nicht die Reaktion auf das Modell (anstelle des Controllers) ist, haben Sie nicht nur die Wiederverwendung befördert wo möglich, aber Sie haben es auch ermöglicht, Ihren Code außerhalb des Kontexts einer Anfrage zu testen.

Schauen wir uns ein einfaches Beispiel an. Angenommen, Sie haben folgenden Code:

```
def index
  @published_posts = Post.where('published_at <= ?', Time.now)
  @unpublished_posts = Post.where('published_at IS NULL OR published_at > ?', Time.now)
end
```

Sie können es so ändern:

```
def index
  @published_posts = Post.published
  @unpublished_posts = Post.unpublished
end
```

Dann können Sie die Logik in Ihr Post-Modell verschieben, wo es wie folgt aussehen könnte:

```
scope :published, ->(timestamp = Time.now) { where('published_at <= ?', timestamp) }
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at > ?', timestamp) }
```

Hüten Sie sich vor default_scope

`default_scope` enthält `default_scope` , um ein Modell standardmäßig automatisch `default_scope` .

```
class Post
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

Der obige Code wird Beiträge bereitstellen, die bereits veröffentlicht werden, wenn Sie eine Abfrage für das Modell ausführen.

```
Post.all # will only list published posts
```

Dieser Spielraum wirkt zwar harmlos, hat aber mehrere versteckte Nebeneffekte, die Sie möglicherweise nicht möchten.

`default_scope` und `order`

Da Sie im `default_scope` eine `order` `default_scope` , wird die aufrufende `order` bei `Post` als zusätzliche Bestellung hinzugefügt, anstatt die Standardeinstellung zu überschreiben.

```
Post.order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."created_at" DESC, "posts"."updated_at" DESC
```

Dies ist wahrscheinlich nicht das gewünschte Verhalten. Sie können dies überschreiben, indem Sie die `order` aus dem Geltungsbereich ausschließen

```
Post.except(:order).order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."updated_at" DESC
```

`default_scope` und Modellinitialisierung

Wie bei jedem anderen `ActiveRecord::Relation` `default_scope` den Standardzustand von Modellen, die von ihm initialisiert werden.

In dem obigen Beispiel hat `Post` standardmäßig `where(published: true)` festgelegt, sodass neue Modelle von `Post` ebenfalls festgelegt werden.

```
Post.new # => <Post published: true>
```

unscoped

`default_scope` kann nominell gelöscht werden, `unscoped` zuerst `unscoped` wird. Dies hat jedoch auch Nebenwirkungen. Nehmen Sie zum Beispiel ein STI-Modell:

```
class Post < Document
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

Standardmäßig Abfragen für `Post` werden scoped werden , um `type` - Spalten mit `'Post'` . `unscoped` wird dies jedoch zusammen mit Ihrem eigenen `default_scope` . Wenn Sie also `unscoped` verwenden, `unscoped` Sie dies ebenfalls berücksichtigen.

```
Post.unscoped.where(type: 'Post').order(updated_at: :desc)
```

unscoped und Model Associations

Betrachten Sie eine Beziehung zwischen `Post` und `User`

```
class Post < ApplicationRecord
  belongs_to :user
  default_scope ->{ where(published: true).order(created_at: :desc) }
end

class User < ApplicationRecord
  has_many :posts
end
```

Indem Sie einen einzelnen `User` , können Sie die dazugehörigen Beiträge sehen:

```
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]]
```

Sie möchten jedoch den `default_scope` aus der `posts` Relation `unscoped` , sodass Sie nicht mit einem Bereich versehene `unscoped`

```
user.posts.unscoped
```

```
SELECT "posts".* FROM "posts"
```

Dadurch werden die `user_id` Bedingung sowie der `default_scope` .

Ein Anwendungsbeispiel für `default_scope`

Trotzdem gibt es Situationen, in denen die Verwendung von `default_scope` vertretbar ist.

Stellen Sie sich ein Mehrmandanten-System vor, bei dem mehrere Subdomains von derselben Anwendung aus mit isolierten Daten bedient werden. Eine Möglichkeit, diese Isolation zu erreichen, ist `default_scope` . Die Nachteile in anderen Fällen werden hier zu einem Nachteil.

```
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
```

Alles, was Sie tun müssen, ist, `Tenant.current_id` zu einem frühen `Tenant.current_id` in der Anforderung `Tenant.current_id` , und jede Tabelle, die `tenant_id` enthält, wird automatisch ohne zusätzlichen Code festgelegt. Durch das Instanziieren von Datensätzen wird automatisch die Mandanten-ID übernommen, unter der sie erstellt wurden.

Das Wichtigste an diesem Anwendungsfall ist, dass der Gültigkeitsbereich einmal pro Anforderung festgelegt wird und sich nicht ändert. Die einzigen Fälle , die Sie benötigen `unscoped` hier sind Spezialfälle wie Hintergrund Arbeitnehmer , die außerhalb eines Ersuchens Umfang ausgeführt werden .

Du wirst es nicht brauchen (YAGNI)

Wenn Sie über ein Feature „YAGNI“ (Sie werden es nicht brauchen) sagen können, sollten Sie es besser nicht implementieren. Durch die Fokussierung auf Einfachheit kann viel Entwicklungszeit eingespart werden. Die ohnehin vorhandene Implementierung solcher Funktionen kann zu Problemen führen:

Probleme

Übertechnik

Wenn ein Produkt komplizierter ist als es sein muss, ist es überentwickelt. Normalerweise werden

diese "noch nicht verwendeten" Funktionen niemals in der vorgesehenen Art und Weise verwendet, in der sie geschrieben wurden, und müssen umgestaltet werden, wenn sie jemals verwendet werden. Vorzeitige Optimierungen, insbesondere Leistungsoptimierungen, führen häufig zu Designentscheidungen, die sich in der Zukunft als falsch herausstellen.

Code aufgebläht

Code Bloat bedeutet unnötig komplizierten Code. Dies kann beispielsweise durch Abstraktion, Redundanz oder fehlerhafte Anwendung von Entwurfsmustern geschehen. Die Codebasis wird schwer verständlich, verwirrend und teuer in der Wartung.

Feature Kriechen

Feature Creep bezieht sich auf das Hinzufügen neuer Features, die über die Kernfunktionalität des Produkts hinausgehen und zu einer unnötig hohen Komplexität des Produkts führen.

Lange Entwicklungszeit

Die Zeit, die zum Entwickeln notwendiger Merkmale verwendet werden könnte, wird aufgewendet, um nicht benötigte Merkmale zu entwickeln. Die Lieferung dauert länger.

Lösungen

KISS - Mach es einfach, dumm

KISS zufolge funktionieren die meisten Systeme am besten, wenn sie einfach gestaltet sind. Einfachheit sollte ein vorrangiges Designziel sein, um die Komplexität zu reduzieren. Dies kann erreicht werden, indem zum Beispiel das Prinzip der Einzelverantwortung befolgt wird.

YAGNI - Du wirst es nicht brauchen

Weniger ist mehr. Denken Sie über jedes Feature nach, ist es wirklich nötig? Wenn Sie sich vorstellen können, dass es sich um YAGNI handelt, lassen Sie es weg. Es ist besser, es zu entwickeln, wenn es benötigt wird.

Kontinuierliches Refactoring

Das Produkt wird ständig verbessert. Mit Refactoring können wir sicherstellen, dass das Produkt nach bester Vorgehensweise erstellt wird und nicht zu einer Patch-Arbeit degeneriert.

Domänenobjekte (keine fetten Modelle)

"Fat Model, Skinny Controller" ist ein sehr guter erster Schritt, skaliert jedoch nicht, sobald Ihre

Codebase zu wachsen beginnt.

Denken wir über die Einzelverantwortung von Modellen nach. Was ist die alleinige Verantwortung von Modellen? Soll es Geschäftslogik geben? Soll es eine nicht antwortbezogene Logik geben?

Nein. Ihre Verantwortung besteht darin, mit der Persistenzschicht und ihrer Abstraktion umzugehen.

Geschäftslogik sowie jede nicht antwortbezogene Logik und nicht persistente Logik sollte in Domänenobjekte verankert sein.

Domänenobjekte sind Klassen, die nur eine Verantwortung in der Domäne des Problems haben. Lassen Sie Ihre Klassen für die von ihnen gelösten Probleme " Scream Their Architecture ".

In der Praxis sollten Sie nach dünnen Modellen, dünnen Ansichten und dünnen Controllern streben. Die Architektur Ihrer Lösung sollte nicht durch das von Ihnen gewählte Framework beeinflusst werden.

Zum Beispiel

Angenommen, Sie sind ein Marktplatz, der Ihren Kunden eine feste Provision von 15% über Stripe berechnet. Wenn Sie eine feste Provision von 15% berechnen, ändert sich Ihre Provision abhängig von der Höhe der Bestellung, da Stripe 2,9% + 30 ¢ berechnet.

Der Betrag, den Sie als Provision berechnen, sollte `amount*0.15 - (amount*0.029 + 0.30)` : `amount*0.15 - (amount*0.029 + 0.30)` .

Schreiben Sie diese Logik nicht in das Modell:

```
# 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
```

Sobald Sie eine neue Zahlungsmethode integriert haben, können Sie diese Funktionalität innerhalb dieses Modells nicht skalieren.

Sobald Sie beginnen, mehr Geschäftslogik zu integrieren, verliert Ihr `Order` Objekt den Zusammenhalt .

Bevorzugen Sie Domain-Objekte, wobei die Berechnung der Provision vollständig von der Verantwortung für persistierende Aufträge abhängt:

```
# 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
```

Die Verwendung von Domänenobjekten hat die folgenden architektonischen Vorteile:

- Der Komponententest ist extrem einfach, da keine Fixtures oder Factorys erforderlich sind, um die Objekte mit der Logik zu instanziieren.
- arbeitet mit allem, was die Meldung akzeptiert `amount` .
- hält jedes Domänenobjekt klein, mit klar definierten Verantwortlichkeiten und höherer Kohäsion.
- Skaliert leicht mit neuen Zahlungsmethoden durch Hinzufügen, nicht Modifizieren .
- stoppt die Tendenz, in jeder Ruby on Rails-Anwendung ein ständig wachsendes `User` zu haben.

Ich persönlich mag Domänenobjekte in `lib` . Wenn Sie dies tun, denken Sie daran, es zu `autoload_paths` hinzuzufügen:

```
# config/application.rb
config.autoload_paths << Rails.root.join('lib')
```

Sie können auch bevorzugen, Domänenobjekte nach dem Befehls- / Abfragemuster aktionsorientierter zu erstellen. In einem solchen Fall empfiehlt es sich, diese Objekte in `app/commands` zu platzieren, da alle `app` Unterverzeichnisse automatisch zum Autoload-Pfad hinzugefügt werden.

Rails Best Practices online lesen: https://riptutorial.com/de/ruby-on-rails/topic/1207/rails-best-practices

Kapitel 52: Rails Engine - Modulare Schienen

Einführung

Schneller Überblick über Rails-Motoren

Engines sind kleine Rails-Anwendungen, mit denen der Anwendung, die sie hostet, Funktionen hinzugefügt werden. Die Klasse, die eine Ruby on Rails-Anwendung definiert, ist `Rails::Application` die ein Großteil ihres Verhaltens von `Rails::Engine` erbt, der Klasse, die eine Engine definiert. Wir können sagen, dass eine normale Rails-Anwendung einfach eine Engine mit mehr Funktionen ist.

Syntax

- Schienen Plugin neues my_module --mountable

Examples

Erstellen Sie eine modulare App

Fertig machen

Zunächst generieren wir eine neue Ruby on Rails-Anwendung:

```
rails new ModularTodo
```

Der nächste Schritt ist die Erzeugung eines Motors!

```
cd ModularTodo && rails plugin new todo --mountable
```

Wir werden auch einen 'engine'-Ordner erstellen, um die Engines zu speichern (selbst wenn wir nur einen haben!).

```
mkdir engines && mv todo ./engines
```

Engines werden wie Edelsteine mit einer Gemspec-Datei geliefert. Lassen Sie uns einige reale Werte angeben, um Warnungen zu vermeiden.

```
 #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    = "//samurails.com"
  s.summary     = "Todo Module"
  s.description = "Todo Module for Modular Rails article"
  s.license     = "MIT"

  #Moar stuff
  #...
end
```

Jetzt müssen wir die Todo-Engine zur übergeordneten Anwendung Gemfile hinzufügen.

```
#ModularTodo/Gemfile
#Other gems
gem 'todo', path: 'engines/todo'
```

Lassen Sie uns `bundle install` ausführen. In der Liste der Edelsteine sollte Folgendes angezeigt werden:

```
Using todo 0.0.1 from source at engines/todo
```

Toll, unser Todo-Motor wird richtig geladen! Bevor wir mit dem Programmieren beginnen, müssen wir noch eines tun: die Todo-Engine installieren. Wir können das in der Datei routes.rb in der übergeordneten App tun.

```
Rails.application.routes.draw do
  mount Todo::Engine => "/", as: 'todo'
end
```

Wir montieren es unter `/` aber wir könnten es auch unter `/todo` zugänglich machen. Da wir nur ein Modul haben, ist `/` in Ordnung.

Jetzt können Sie Ihren Server hochfahren und in Ihrem Browser überprüfen. Sie sollten die Standardansicht von Rails sehen, da wir noch keine Controller / Ansichten definiert haben. Lass uns das jetzt machen!

Erstellen der Todo-Liste

Wir werden ein Modell mit dem Namen `Task` im Todo-Modul erstellen, aber um die Datenbank korrekt aus der übergeordneten Anwendung zu migrieren, müssen wir der `engine.rb` Datei einen kleinen Initialisierer `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
```

Das ist es, jetzt, wenn wir Migrationen von der übergeordneten Anwendung ausführen, werden die Migrationen in der Todo-Engine ebenfalls geladen.

Erstellen wir das `Task` . Der Befehl `scaffold` muss aus dem Engine-Ordner ausgeführt werden.

```
cd engines/todo && rails g scaffold Task title:string content:text
```

Führen Sie die Migrationen aus dem übergeordneten Ordner aus:

```
rake db:migrate
```

Jetzt müssen wir nur noch die Wurzelroute in der Todo-Engine definieren:

```
#ModularTodo/engines/todo/config/routes.rb
Todo::Engine.routes.draw do
  resources :tasks
  root 'tasks#index'
end
```

Sie können damit spielen, Aufgaben erstellen, löschen… Oh, warten Sie, das Löschen funktioniert nicht! Warum?! Nun, es scheint, als sei JQuery nicht geladen, also fügen wir es der Datei `application.js` in der Engine hinzu!

```
// ModularTodo/engines/todo/app/assets/javascripts/todo/application.js
//= require jquery
//= require jquery_ujs
//= require_tree .
```

Ja, jetzt können wir Aufgaben zerstören!

Rails Engine - Modulare Schienen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9080/rails-engine---modulare-schienen

Kapitel 53: Rails erzeugen Befehle

Einführung

Verwendung: `rails generate GENERATOR_NAME [args] [options]` .

Verwenden Sie `rails generate` um verfügbare Generatoren aufzulisten. Alias: `rails g` .

Parameter

Parameter	Einzelheiten
`-h` / `--help`	Hilfe zu jedem Generator-Befehl erhalten
`-p` / `--pretend`	Pretend Mode: Generator ausführen, aber keine Dateien erstellen oder ändern
`field:type`	'Feldname' ist der Name der zu erstellenden Spalte und 'Typ' ist der Datentyp der Spalte. Die möglichen Werte für 'type' in `field:type` sind im Abschnitt "Anmerkungen" angegeben.

Bemerkungen

Die möglichen Werte für 'Typ' in `field:type` sind:

Datentyp	Beschreibung
`:string`	Für kleinere Textstücke (hat normalerweise eine Zeichenbegrenzung von 255)
`:text`	Für längere Textstücke wie einen Absatz
`:binary`	Speichern von Daten einschließlich Bildern, Audios und Videos
`:boolean`	Speichern von wahren oder falschen Werten
`:date`	Nur das Datum
`:time`	Nur die zeit
`:datetime`	Datum und Uhrzeit
`:float`	Schwimmer ohne Genauigkeit lagern
`:decimal`	Schwimmer präzise lagern
`:integer`	Speichern ganzer Zahlen

Examples

Schienen erzeugen Modell

Um zu erzeugen `ActiveRecord` Modell , das automatisch die richtige db Migrationen & vorformulierten Testdateien für Ihr Modell erstellt, geben Sie diesen Befehl

```
rails generate model NAME column_name:column_type
```

'NAME' ist der Name des Modells. 'Feld' ist der Name der Spalte in der DB-Tabelle und 'Typ' ist der Spaltentyp (zB `name:string` oder `body:text`). Überprüfen Sie den Abschnitt "Bemerkungen" auf eine Liste der unterstützten Spaltentypen.

Zum Einrichten von Fremdschlüsseln fügen Sie `belongs_to:model_name` .

Angenommen, Sie wollten ein `User` einrichten, das einen `username` und eine `email` und zu einer `School` gehört. `username` Sie Folgendes ein

```
rails generate model User username:string email:string school:belongs_to
```

`rails g` ist eine Abkürzung für `rails generate` . Dies würde das gleiche Ergebnis erzielen

```
rails g model User username:string email:string school:belongs_to
```

Schienen erzeugen Migration

Sie können eine Rails-Migrationsdatei mit dem folgenden Befehl vom Terminal aus generieren:

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

Für eine Liste aller vom Befehl unterstützten Optionen können Sie den Befehl ohne Argumente ausführen, da `rails generate migration` in `rails generate migration` .

Wenn Sie zum `last_name` Felder `first_name` und `last_name` zur `users` hinzufügen möchten, können Sie `first_name` tun:

```
rails generate migration AddNamesToUsers last_name:string first_name:string
```

Rails erstellt die folgende Migrationsdatei:

```
class AddNamesToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :last_name, :string
    add_column :users, :first_name, :string
  end
end
```

Wenden Sie nun die ausstehenden Migrationen auf die Datenbank an, indem Sie Folgendes im Terminal ausführen:

5,0

```
rake db:migrate
```

5,0

```
rails db:migrate
```

> **Hinweis:** Um noch weniger zu tippen, können Sie `generate` durch `g` ersetzen.

Schienen erzeugen Gerüst

HAFTUNGSAUSSCHLUSS : Gerüste werden nicht empfohlen, es sei denn, es handelt sich um sehr herkömmliche CRUD-Apps / -Tests. Dadurch können viele Dateien (Ansichten / Modelle / Controller) generiert werden, die in Ihrer Webanwendung nicht benötigt werden. Dies führt zu Kopfschmerzen (bad :().

Um ein voll funktionsfähiges Gerüst für ein neues Objekt zu erstellen, einschließlich Modell, Controller, Ansichten, Assets und Tests, verwenden Sie den Befehl `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
```

Dann können Sie `rake db:migrate` ausführen, um die Datenbanktabelle einzurichten.

Dann können Sie die http: // localhost: 3000 / Widgets besuchen und sehen ein voll funktionsfähiges CRUD-Gerüst.

Schienen generieren Controller

Wir können einen neuen Controller mit `rails g controller` Befehl erstellen.

```
$ bin/rails generate controller controller_name
```

Der Controller-Generator erwartet Parameter in der Form `generate controller ControllerName action1 action2` .

Im Folgenden wird ein Greetings-Controller mit einer Aktion von Hallo erstellt.

```
$ bin/rails generate controller Greetings hello
```

Sie sehen die folgende Ausgabe

```
     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
```

Dies generiert Folgendes

Datei	Beispiel
Controller-Datei	`greetings_controller.rb`
Datei ansehen	`hello.html.erb`
Funktionstestdatei	`greetings_controller_test.rb`
Helfer anzeigen	`greetings_helper.rb`
JavaScript-Datei	`greetings.coffee`

Es werden auch Routen für jede Aktion in `routes.rb`

Rails erzeugen Befehle online lesen: https://riptutorial.com/de/ruby-on-rails/topic/2540/rails-erzeugen-befehle

Kapitel 54: Rails-API

Examples

Nur-API-Anwendung erstellen

Um eine Rails-Anwendung zu erstellen, bei der es sich um einen API-Server handelt, können Sie mit einer begrenzten Untermenge von Rails in Rails 5 beginnen.

So generieren Sie eine neue Rails-API-App:

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

`--api` entfernt Funktionen, die beim `--api` einer API nicht benötigt werden. Dazu gehören Sitzungen, Cookies, Assets und alles, was dazu führt, dass Rails in einem Browser funktioniert.

Außerdem werden die Generatoren so konfiguriert, dass sie beim Generieren einer neuen Ressource keine Ansichten, Helfer und Elemente generieren.

Wenn Sie `ApplicationController` in einer Web-App mit einer API-App vergleichen, werden Sie feststellen, dass die `ActionController::Base` von `ActionController::Base` , während die API-Version von `ActionController::API` , die eine viel kleinere Teilmenge an Funktionen enthält, erweitert wird.

Rails-API online lesen: https://riptutorial.com/de/ruby-on-rails/topic/4305/rails-api

Kapitel 55: React.js mithilfe von Hyperloop in Rails integrieren

Einführung

In diesem Thema wird die Integration von React.js mit Rails mithilfe des Hyperloop- Gems beschrieben

Andere Ansätze, die hier nicht behandelt werden, sind die reakts-Schienen oder die reakt_on_rails-Edelsteine.

Bemerkungen

Komponentenklassen generieren einfach die entsprechenden Javascript-Komponentenklassen.

Sie können auch direkt von Ihren Ruby-Komponentenklassen auf Javascript-Komponenten und -Bibliotheken zugreifen.

Hyperloop "prerender" die View-Server-Seite, sodass Ihre ursprüngliche Ansicht genau wie ERB- oder HAML-Vorlagen geladen wird. Nach dem Laden auf dem Client übernimmt rea und wird das DOM inkrementell aktualisieren, wenn sich der Status aufgrund von Eingaben des Benutzers, HTTP-Anforderungen oder eingehenden Web-Socket-Daten ändert.

Neben den Komponenten verfügt Hyperloop über Stores zum Verwalten des gemeinsam genutzten Zustands, Operationen zum Einkapseln von isomorpher Geschäftslogik und Modelle, die direkten Zugriff auf Ihre ActiveRecord-Modelle auf dem Client mit der Standard-AR-Syntax ermöglichen.

Mehr Infos hier: http://ruby-hyperloop.io/

Examples

Hinzufügen einer einfachen Reaktionskomponente (in Ruby geschrieben) zu Ihrer Rails-App

1. Füge den Hyperloop-Edelstein zu deinen Rails (4.0 - 5.1) hinzu
2. `bundle install`
3. Fügen Sie das Hyperloop-Manifest der Datei "application.js" hinzu:

```
// app/assets/javascripts/application.js
...
//= hyperloop-loader
```

4. Erstellen Sie Ihre Reaktionskomponenten und platzieren Sie sie im `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. Komponenten wirken wie Ansichten. Sie werden mithilfe der `render_component` Methode in einem Controller "angehängt":

```
# somewhere in a controller:
  ...
  def hello_world
    render_component # renders HelloWorld based on method name
  end
```

Komponentenparameter deklarieren (Requisiten)

```
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'
```

HTML-Tags

```
# HTML tags are built in and are UPCASE
class HTMLExample < Hyperloop::Component
  render do
    DIV do
      SPAN { "Hello There" }
      SPAN { "Welcome to the Machine!" }
    end
  end
end
```

Event-Handler

```
# 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
```

Zustände

```
# 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
```

Beachten Sie, dass Zustände mithilfe von Hyperloop :: Stores von Komponenten gemeinsam genutzt werden können

Rückrufe

```
# 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
```

React.js mithilfe von Hyperloop in Rails integrieren online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9809/react-js-mithilfe-von-hyperloop-in-rails-integrieren

Kapitel 56: Reagiere mit Rails mit Reaktiver Gleis gem

Examples

Reaktiver Einbau für Rails mithilfe von schienen_reakt gem

Fügen Sie React-Rails zu Ihrem Gemfile hinzu:

```
gem 'react-rails'
```

Und installiere:

```
bundle install
```

Führen Sie dann das Installationsskript aus:

```
rails g react:install
```

Dieser Wille:

Erstellen Sie eine Manifest-Datei components.js und ein Verzeichnis app / assets / javascripts / components /, in das Sie Ihre Komponenten einfügen und Folgendes in Ihre application.js einfügen:

```
//= require react
//= require react_ujs
//= require components
```

Verwenden von react_rails in Ihrer Anwendung

React.js baut auf

Sie können auswählen, welche React.js-Builds (Entwicklung, Produktion, mit oder ohne Add-Ons) in jeder Umgebung bereitgestellt werden sollen, indem Sie eine Konfiguration hinzufügen. Hier sind die Standardeinstellungen:

```
# 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
```

Verwenden Sie diese Konfiguration, um Add-Ons einzubinden:

```
MyApp::Application.configure do
  config.react.addons = true # defaults to false
end
```

Nach dem Neustart Ihres Rails-Servers stellt // = requir reag den Build von React.js bereit, der in den Konfigurationen angegeben wurde.

React-Rails bietet einige weitere Optionen für Versionen und Builds von React.js. In VERSIONS.md finden Sie weitere Informationen zur Verwendung des React-Source-Gem oder zum Ablegen eigener React.js-Kopien.

JSX

Starten Sie Ihren Server nach der Installation von React-Rails neu. Jetzt werden .js.jsx-Dateien in die Asset-Pipeline umgewandelt.

BabelTransformer-Optionen

Sie können die Transformatoren und benutzerdefinierten Plug-Ins von Babel verwenden und Optionen an den Babel-Transpiler übergeben, indem Sie die folgenden Konfigurationen hinzufügen:

```
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]
}
```

Reactive-Rails verwenden unter der Haube einen Ruby-Babel-Transpiler für die Transformation.

Rendern & Montieren

React `react-rails` enthalten einen View Helper `(react_component)` und einen unauffälligen JavaScript-Treiber (Reakt_Ujs), die zusammenarbeiten, um React-Komponenten auf der Seite zu platzieren. Sie sollten den UJS-Treiber in Ihrem Manifest nach dem Reaktivieren (und nach Turbolinks, wenn Sie Turbolinks verwenden) benötigen.

Der View-Helfer fügt auf der Seite ein div mit der angeforderten Komponentenklasse und den entsprechenden Requisiten hinzu. Zum Beispiel:

```
<%= react_component('HelloMessage', name: 'John') %>
<!-- becomes: -->
<div data-react-class="HelloMessage" data-react-
props="{&quot;name&quot;:&quot;John&quot;}"></div>
```

Beim Laden der Seite scannt der Treiber reagiert die Seite und stellt die Komponenten mithilfe der

Datenreaktionsklasse und der Datenreaktionspropeller bereit.

Wenn Turbolinks vorhanden ist, werden Komponenten auf der Seite installiert: Ereignis ändern und auf Seite entladen: Vor dem Entladen. Turbolinks> = 2.4.0 werden empfohlen, da hier bessere Ereignisse auftreten.

Im Falle von Ajax-Aufrufen kann das UJS-Mounten manuell durch Aufrufen von Javascript ausgelöst werden:

ReactRailsUJS.mountComponents () Die Signatur des View Helper lautet:

```
react_component(component_class_name, props={}, html_options={})
```

`component_class_name` ist eine Zeichenfolge, die eine global zugreifbare Komponentenklasse benennt. Es kann Punkte enthalten (z. B. "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` kann `html_options` umfassen: `tag:` Verwenden eines anderen Elements als div zum Einbetten von data `html_options` und data -react-props. `prerender: true` , um die Komponente auf dem Server `prerender: true` . `**other` Alle anderen Argumente (zB class:, id :) werden an content_tag übergeben.

Reagiere mit Rails mit Reaktiver Gleis gem online lesen: https://riptutorial.com/de/ruby-on-rails/topic/7032/reagiere-mit-rails-mit-reaktiver-gleis-gem

Kapitel 57: Regeln der Namensgebung

Examples

Controller

Die Namen der Controller-Klassen sind pluralisiert. Der Grund ist, dass der Controller mehrere Instanzen der Objektinstanz steuert.

Zum Beispiel : `OrdersController` wäre der Controller für eine `orders` Tabelle. Rails sucht dann nach der Klassendefinition in einer Datei namens `orders_controller.rb` im `orders_controller.rb /app/controllers` .

Zum Beispiel : `PostsController` wäre der Controller für eine `posts` Tabelle.

Wenn der Name der Controller-Klasse mehrere Wörter enthält, wird angenommen, dass der Tabellenname Unterstriche zwischen diesen Wörtern enthält.

Zum Beispiel: Wenn ein Controller `PendingOrdersController` dann wird der Dateiname für diesen Controller `pending_orders_controller.rb` .

Modelle

Das Modell wird nach der Klassennamenskonvention von MixedCase ohne Unterbrechung benannt und ist immer das Singular des Tabellennamens.

Beispiel : Wenn eine Tabelle mit dem Namen `orders` , würde das zugehörige Modell den Namen `Order`

Beispiel : Wenn eine Tabelle mit `posts` , würde das zugehörige Modell als `Post`

Rails sucht dann in einer Datei namens `order.rb` im `order.rb /app/models` nach der Klassendefinition.

Wenn der Modellklassenname mehrere Wörter mit Großschreibung enthält, wird angenommen, dass der Tabellenname Unterstriche zwischen diesen Wörtern aufweist.

Zum Beispiel: Wenn ein Modell mit dem Namen wird `BlogPost` dann angenommen Tabellenname wird `blog_posts` .

Ansichten und Layouts

Wenn eine Controller-Aktion ausgeführt wird, versucht Rails basierend auf dem Namen des Controllers, ein übereinstimmendes Layout und eine passende Ansicht zu finden.

Ansichten und Layouts werden im `app/views` Verzeichnis abgelegt.

Bei einer Anfrage an die `PeopleController#index` Aktion sucht Rails nach:

- das Layout namens `people` in `app/views/layouts/` (oder `application` wenn keine Übereinstimmung gefunden wird)
- Standardmäßig eine Ansicht namens `index.html.erb` in `app/views/people/`
- Wenn Sie eine andere Datei namens `index_new.html.erb` rendern möchten, `index_new.html.erb` Sie Code in `PeopleController#index` action wie `render 'index_new'`
- Wir können für jede `action` verschiedene `layouts render 'index_new', layout: 'your_layout_name'` indem Sie `render 'index_new', layout: 'your_layout_name'` schreiben. `render 'index_new', layout: 'your_layout_name'`

Dateinamen und automatisches Laden

Rails-Dateien - und Ruby-Dateien im Allgemeinen - sollten mit `lower_snake_case` Dateinamen `lower_snake_case` . Z.B

```
app/controllers/application_controller.rb
```

ist die Datei, die die `ApplicationController` Klassendefinition enthält. Beachten Sie, dass, während `PascalCase` für Klassen- und `PascalCase` verwendet wird, die Dateien, in denen sie sich befinden, immer noch `lower_snake_case` .

Eine konsistente Benennung ist wichtig, da Rails nach Bedarf Dateien automatisch lädt und "Flexion" verwendet, um zwischen verschiedenen Benennungsstilen zu transformieren, z. B. zum Umwandeln von `application_controller` in `ApplicationController` und wieder zurück.

Wenn Rails beispielsweise `BlogPost` dass die `BlogPost` Klasse nicht vorhanden ist (noch nicht geladen wurde), sucht es nach einer Datei namens `blog_post.rb` und versucht, diese Datei zu laden.

Es ist daher auch wichtig, Dateien nach dem zu benennen, was sie enthalten, da der Autoloader erwartet, dass Dateinamen mit dem Inhalt übereinstimmen. Wenn zum Beispiel die `blog_post.rb` stattdessen eine Klasse mit dem Namen `Post` , wird ein `LoadError` : `Expected [some path]/blog_post.rb to define BlogPost` .

Wenn Sie ein Verzeichnis unter `app/something/` hinzufügen (z. B. / models / products /) und

- Wenn Sie Module und Klassen innerhalb des neuen Verzeichnisses benennen möchten, müssen Sie nichts tun, und es wird selbst geladen. Zum Beispiel `app/models/products/ you would need to wrap your class in` in `app/models/products/ you would need to wrap your class in` Modul Products einbetten.
- `config.autoload_paths += %W( #{config.root}/app/models/products )` Sie keine Module und Klassen innerhalb meines neuen `config.autoload_paths += %W( #{config.root}/app/models/products )` möchten, müssen Sie `config.autoload_paths += %W( #{config.root}/app/models/products )` zu Ihrer `application.rb` hinzufügen, um sie `config.autoload_paths += %W( #{config.root}/app/models/products )` zu `config.autoload_paths += %W( #{config.root}/app/models/products )` .

Eine weitere Sache, auf die Sie achten sollten (vor allem, wenn Englisch nicht Ihre Muttersprache ist), ist die Tatsache, dass Rails unregelmäßige Plural-Substantive in Englisch berücksichtigt.

Wenn Sie ein Modell mit dem Namen "Foot" haben, muss der entsprechende Controller "FeetController" und nicht "FootsController" genannt werden, wenn das "magische" Routing von Rails (und viele weitere solcher Funktionen) funktionieren soll.

Modellklasse vom Controller-Namen

Sie können eine Model-Klasse von einem Controllernamen auf folgende Weise erhalten (Kontext ist Controller-Klasse):

```
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
```

Regeln der Namensgebung online lesen: https://riptutorial.com/de/ruby-on-rails/topic/1493/regeln-der-namensgebung

Kapitel 58: Reservierte Wörter

Einführung

Sie sollten vorsichtig sein, wenn Sie diese Wörter für Variablen, Modellnamen, Methodennamen usw. verwenden.

Examples

Reservierte Wortliste

- ADDITIONAL_LOAD_PATHS
- ARGF
- ARGV
- ActionController
- ActionView
- Aktiver Rekord
- ArgumentError
- Array
- BasicSocket
- Benchmark
- Bignum
- Bindung
- CGI
- CGIMethods
- CROSS_COMPILING
- Klasse
- ClassInheritableAttributes
- Vergleichbar
- ConditionVariable
- Konfig
- Fortsetzung
- DRb
- DRbIdConv
- DRbObject
- DRbUndumped
- Daten
- Datum
- Terminzeit
- Delegator
- Delegator
- Verdauen
- Dir
- ENV

- EOFError
- ERB
- Zahlreich
- Errno
- Ausnahme
- FALSCH
- FalseClass
- Fcntl
- Datei
- Dateiliste
- FileTask
- FileTest
- FileUtils
- Fixnum
- Schweben
- FloatDomainError
- GC
- Juwel
- GetoptLong
- Hash
- IO
- IOError
- IPSocket
- IPsocket
- IndexError
- Inflector
- Ganze Zahl
- Unterbrechen
- Kernel
- LN_SUPPORTED
- LoadError
- LocalJumpError
- Logger
- Marschall
- MatchData
- Übereinstimmende Daten
- Mathematik
- Methode
- Modul
- Mutex
- Mysql
- MySQL-Fehler
- MysqlField
- MysqlRes
- NULL
- NameFehler

- NilClass
- NoMemoryError
- NoMethodError
- Nicht schreiben
- NotImplementedError
- Numerisch
- OPT_TABLE
- Objekt
- ObjectSpace
- Beobachtbar
- Beobachter
- PGError
- PGconn
- PGlarge
- PGresult
- PLATTFORM
- PStore
- ParseDate
- Präzision
- Proc
- Verarbeiten
- Warteschlange
- RAKEVERSION
- VERÖFFENTLICHUNGSDATUM
- RUBIN
- RUBY_PLATFORM
- RUBY_RELEASE_DATE
- RUBY_VERSION
- Gestell
- Rechen
- RakeApp
- RakeFileUtils
- Angebot
- RangeError
- Rational
- Regexp
- RegexpError
- Anfordern
- Laufzeit Fehler
- STDERR
- STDIN
- STDOUT
- ScanError
- Skriptfehler
- Sicherheitsfehler
- Signal

- SignalException
- SimpleDelegater
- SimpleDelegator
- Singleton
- SizedQueue
- Steckdose
- SocketError
- Standart Fehler
- String
- StringScanner
- Struktur
- Symbol
- Syntax-Fehler
- SystemCallError
- SystemExit
- SystemStackError
- TCPServer
- TCPSocket
- TCPserver
- TCPsocket
- TOPLEVEL_BINDING
- WAHR
- Aufgabe
- Text
- Faden
- ThreadError
- ThreadGroup
- Zeit
- Transaktion
- TrueClass
- TypeError
- UDPSocket
- UDP-Socket
- UNIXServer
- UNIXSocket
- UNIX-Server
- UNIXsocket
- UnboundMethod
- URL
- AUSFÜHRUNG
- Ausführlich
- YAML
- ZeroDivisionError
- @Basis_Pfad
- akzeptieren
- Acces

- Axi
- Aktion
- Attribute
- application2
- Ruf zurück
- Kategorie
- Verbindung
- Datenbank
- Dispatcher
- display1
- Fahrt
- Fehler
- Format
- Wirt
- Schlüssel
- Layout
- Belastung
- Verknüpfung
- Neu
- benachrichtigen
- öffnen
- Öffentlichkeit
- Zitat
- machen
- anfordern
- Aufzeichnungen
- Antworten
- sparen
- Umfang
- senden
- Session
- System
- Vorlage
- Prüfung
- Auszeit
- to_s
- Art
- URI
- Besuche
- Beobachter

Datenbankfeldnamen

- hergestellt in
- erstellt am
- aktualisiert am
- aktualisiert am

- deleted_at
- (Paranoia
- Juwel)
- Version sperren
- Art
- Ich würde
- # {table_name} _count
- Position
- Eltern ID
- lft
- rgt
- quote_value

Ruby reservierte Wörter

- alias
- und
- START
- Start
- brechen
- Fall
- Klasse
- def
- definiert?
- tun
- sonst
- elsif
- ENDE
- Ende
- dafür sorgen
- falsch
- zum
- ob
- Modul
- Nächster
- Null
- nicht
- oder
- Wiederholen
- Rettung
- wiederholen
- Rückkehr
- selbst
- Super
- dann
- wahr
- undef

- es sei denn
- bis um
- wann
- während
- Ausbeute
- _ *DATEI* _
- _ *LINE* _

Reservierte Wörter online lesen: https://riptutorial.com/de/ruby-on-rails/topic/10818/reservierte-worter

Kapitel 59: Routing

Einführung

Der Rails-Router erkennt URLs und leitet sie an die Aktion eines Controllers weiter. Es kann auch Pfade und URLs generieren, ohne dass Strings in Ihren Ansichten hartcodiert werden müssen.

Bemerkungen

Unter "Routing" werden im Allgemeinen URLs von Ihrer App "behandelt". Im Fall von Rails ist es in der Regel der Controller und welche Aktion des Controllers für eine bestimmte eingehende URL. In Rails-Apps werden Routen normalerweise in der Datei `config/routes.rb` abgelegt.

Examples

Ressourcenrouting (Basic)

Routen sind in `config/routes.rb` . Sie werden häufig als Gruppe zusammengehöriger Routen definiert, wobei die `resources` oder `resource` werden.

`resources :users` erstellt die folgenden sieben Routen, die alle den Aktionen von `UsersController` :

```
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'
```

Aktionsnamen werden nach dem `#` im Parameter `to` oben angezeigt. Methoden mit denselben Namen müssen in `app/controllers/users_controller.rb` wie folgt definiert werden:

```
class UsersController < ApplicationController
  def index
  end

  def create
  end

  # continue with all the other methods…
end
```

Sie können die generierten Aktionen `only` `except` einschränken:

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

Sie können alle Routen Ihrer Anwendung jederzeit anzeigen, indem Sie Folgendes ausführen:

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
```

So zeigen Sie nur die Routen an, die einem bestimmten Controller zugeordnet sind:

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
```

Sie können mit der Option `-g` nach Routen suchen. Dies zeigt jede Route, die teilweise mit dem Namen der Hilfsmethode, dem URL-Pfad oder dem HTTP-Verb übereinstimmt:

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
```

Wenn Sie den `rails` server im Entwicklungsmodus ausführen, können Sie außerdem auf eine Webseite zugreifen, die alle Ihre Routen mit einem von oben nach unten übereinstimmenden `<hostname>/rails/info/routes` . Es wird so aussehen:

Helfer	HTTP-Verb	Pfad	Controller Nr. Aktion
Pfad / URL		[Pfadübereinstimmung]	
Benutzerpfad	ERHALTEN	/users(.:format)	Benutzer # Index
	POST	/users(.:format)	Benutzer # erstellen
new_user_path	ERHALTEN	/users/new(.:format)	Benutzer # neu
edit_user_path	ERHALTEN	/users/:id/edit(.:format)	Benutzer # bearbeiten
user_path	ERHALTEN	/users/:id(.:format)	Benutzer # anzeigen
	PATCH	/users/:id(.:format)	Benutzer # Update
	STELLEN	/users/:id(.:format)	Benutzer # Update
	LÖSCHEN	/users/:id(.:format)	Benutzer zerstören

Routen können für nur Mitgliedern zur Verfügung deklariert werden (nicht Sammlungen) unter Verwendung des Verfahrens `resource` anstelle von `resources` in `routes.rb` . Bei einer `resource` wird eine `index` nicht standardmäßig erstellt, sondern nur, wenn explizit nach einer der folgenden Optionen gefragt wird:

```
resource :orders, only: [:index, :create, :show]
```

Einschränkungen

Sie können mithilfe von Einschränkungen filtern, welche Routen verfügbar sind.

Es gibt verschiedene Möglichkeiten, Einschränkungen zu verwenden, darunter:

- Segmenteinschränkungen ,
- anforderungsbasierte Einschränkungen
- fortgeschrittene Einschränkungen

Beispielsweise eine angeforderte Einschränkung, die nur den Zugriff einer bestimmten IP-Adresse auf eine Route zulässt:

```
constraints(ip: /127\.0\.0\.1$/) do
  get 'route', to: "controller#action"
```

```
end
```

Weitere ähnliche Beispiele anzeigen ActionDispatch :: Routing :: Mapper :: Scoping .

Wenn Sie etwas komplexeres machen möchten, können Sie erweiterte Einschränkungen verwenden und eine Klasse erstellen, um die Logik zu umschließen:

```
# 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
```

Ein Formular, mehrere Schaltflächen zum Senden

Sie können den Wert der Übergabe-Tags eines Formulars auch als Einschränkung verwenden, um zu einer anderen Aktion weiterzuleiten. Wenn Sie über ein Formular mit mehreren Senden-Schaltflächen verfügen (z. B. "Vorschau" und " `routes.rb` "), können Sie diese Einschränkung direkt in der `routes.rb` , anstatt Javascript zu schreiben, um die Ziel-URL des Formulars zu ändern. Zum Beispiel können Sie mit dem commit_param_routing gem die Schienen `submit_tag`

`submit_tag` ersten Parameter von Rails `submit_tag` können Sie den Wert Ihres Formular-Commit-Parameters ändern

```
# 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 ApiVersionContraint
end
```

Scoping-Routen

Rails bietet mehrere Möglichkeiten, um Ihre Routen zu organisieren.

Umfang nach URL :

```
scope 'admin' do
  get 'dashboard', to: 'administration#dashboard'
  resources 'employees'
end
```

Dies generiert die folgenden Routen

```
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'
```

Auf der Serverseite kann es sinnvoller sein, einige Ansichten in einem anderen Unterordner zu speichern, um Administratoransichten von Benutzeransichten zu trennen.

Umfang nach Modul

```
scope module: :admin do
  get 'dashboard', to: 'administration#dashboard'
end
```

`module` sucht die Controller-Dateien unter dem Unterordner des angegebenen Namens

```
get       '/dashboard',          to: 'admin/administration#dashboard'
```

Sie können das Pfadhilfspräfix umbenennen, indem Sie einen `as` Parameter hinzufügen

```
scope 'admin', as: :administration do
  get 'dashboard'
```

```
end

# => administration_dashboard_path
```

Rails bietet eine bequeme Möglichkeit, all dies mit der `namespace` Methode `namespace` . Die folgenden Erklärungen sind gleichwertig

```
namespace :admin do
end

scope 'admin', module: :admin, as: :admin
```

Umfang durch Controller

```
scope controller: :management do
  get 'dashboard'
  get 'performance'
end
```

Diese generieren diese Routen

```
get       '/dashboard',          to: 'management#dashboard'
get       '/performance',        to: 'management#performance'
```

Flache Schachtelung

Ressourcenwege akzeptieren eine `:shallow` Option, mit der sich URLs nach Möglichkeit verkürzen lassen. Ressourcen sollten nicht mehr als eine Ebene tief verschachtelt sein. Eine Möglichkeit, dies zu vermeiden, ist das Erstellen von flachen Routen. Das Ziel ist, die URL-Segmente der übergeordneten Sammlung dort zu lassen, wo sie nicht benötigt werden. Das Endergebnis ist, dass die einzigen geschachtelten Routen für die `:index` Aktionen generiert werden `:index` `:create` und `:new` Aktionen. Der Rest wird in einem eigenen flachen URL-Kontext gehalten. Es gibt zwei Optionen für den Umfang der benutzerdefinierten flachen Routen:

- **: shallow_path: Stellt** einen angegebenen Parameter den Mitgliederpfaden **voran**

  ```
  scope shallow_path: "sekret" do
    resources :articles do
      resources :comments, shallow: true
    end
  end
  ```

- **: shallow_prefix: Fügt** den benannten Helfern angegebene Parameter hinzu

  ```
  scope shallow_prefix: "sekret" do
    resources :articles do
      resources :comments, shallow: true
    end
  end
  ```

Wir können `shallow` Wege auch anhand folgender `shallow` veranschaulichen:

```
resources :auctions, shallow: true do
  resources :bids do
   resources :comments
  end
end
```

alternativ wie folgt codiert (wenn Sie blockglücklich sind):

```
resources :auctions do
 shallow do
   resources :bids do
     resources :comments
   end
 end
end
```

Die resultierenden Routen sind:

Präfix	Verb	URI-Muster
bid_comments	ERHALTEN	/bids/:bid_id/comments(.:format)
	POST	/bids/:bid_id/comments(.:format)
new_bid_comment	ERHALTEN	/bids/:bid_id/comments/new(.:format)
edit_comment	ERHALTEN	/comments/:id/edit(.:format)
Kommentar	ERHALTEN	/comments/:id(.:format)
	PATCH	/comments/:id(.:format)
	STELLEN	/comments/:id(.:format)
	LÖSCHEN	/comments/:id(.:format)
auction_bids	ERHALTEN	/auctions/:auction_id/bids(.:format)
	POST	/auctions/:auction_id/bids(.:format)
new_auction_bid	ERHALTEN	/auktionen/:auction_id/bids/new(.:format)
edit_bid	ERHALTEN	/bids/:id/edit(.:format)
bieten	ERHALTEN	/bids/:id(.:format)
	PATCH	/bids/:id(.:format)
	STELLEN	/bids/:id(.:format)
	LÖSCHEN	/bids/:id(.:format)

Präfix	Verb	URI-Muster
Auktionen	ERHALTEN	/auctions(.:format)
	POST	/auctions(.:format)
new_auction	ERHALTEN	/auctions/new(.:format)
edit_auction	ERHALTEN	/auctions/:id/edit(.:format)
Versteigerung	ERHALTEN	/auctions/:id(.:format)
	PATCH	/auctions/:id(.:format)
	STELLEN	/auctions/:id(.:format)
	LÖSCHEN	/auctions/:id(.:format)

Wenn Sie die erstellten Routen sorgfältig analysieren, werden Sie feststellen, dass die verschachtelten Teile der URL nur dann enthalten sind, wenn sie zur Ermittlung der anzuzeigenden Daten benötigt werden.

Sorgen

Um Wiederholungen in verschachtelten Routen zu vermeiden, bieten Bedenken eine gute Möglichkeit, gemeinsame Ressourcen zu verwenden, die wiederverwendbar sind. Um ein Anliegen zu erstellen, verwenden Sie die Methode `concern` in der Datei `routes.rb` . Die Methode erwartet ein Symbol und einen Block:

```
concern :commentable do
  resources :comments
end
```

Obwohl keine Routen selbst erstellt werden, ermöglicht dieser Code die Verwendung des Attributs `:concerns` für eine Ressource. Das einfachste Beispiel wäre:

```
resource :page, concerns: :commentable
```

Die entsprechende verschachtelte Ressource würde folgendermaßen aussehen:

```
resource :page do
  resource :comments
end
```

Dies würde zum Beispiel die folgenden Routen erstellen:

```
/pages/#{page_id}/comments
/pages/#{page_id}/comments/#{comment_id}
```

Damit Anliegen sinnvoll sind, müssen mehrere Ressourcen vorhanden sein, die das Anliegen nutzen. Zusätzliche Ressourcen könnten eine der folgenden Syntax zum Aufrufen des Problems verwenden:

```
resource :post, concerns: %i(commentable)
resource :blog do
  concerns :commentable
end
```

Umleitung

Sie können eine Umleitung in Rails-Routen wie folgt durchführen:

4,0

```
get '/stories', to: redirect('/posts')
```

4,0

```
match "/abc" => redirect("http://example.com/abc")
```

Sie können auch alle unbekannten Routen zu einem bestimmten Pfad umleiten:

4,0

```
match '*path' => redirect('/'), via: :get
# or
get '*path' => redirect('/')
```

4,0

```
match '*path' => redirect('/')
```

Mitglieder- und Sammelrouten

Durch das Definieren eines Mitgliederblocks in einer Ressource wird eine Route erstellt, die auf ein einzelnes Mitglied dieser ressourcenbasierten Route einwirken kann:

```
resources :posts do
  member do
    get 'preview'
  end
end
```

Dadurch wird die folgende Memberroute generiert:

```
get '/posts/:id/preview', to: 'posts#preview'
# preview_post_path
```

Sammlungsrouten ermöglichen das Erstellen von Routen, die sich auf eine Sammlung von

Ressourcenobjekten auswirken können:

```
resources :posts do
  collection do
    get 'search'
  end
end
```

Dadurch wird die folgende Sammlungsroute generiert:

```
get '/posts/search', to: 'posts#search'
# search_posts_path
```

Eine alternative Syntax:

```
resources :posts do
  get 'preview', on: :member
  get 'search',  on: :collection
end
```

URL-Params mit einem Punkt

Wenn Sie einen URL-Parameter unterstützen möchten, der komplexer als eine ID-Nummer ist, können Probleme mit dem Parser auftreten, wenn der Wert einen Punkt enthält. Alles, was auf einen Punkt folgt, wird als Format angenommen (z. B. Json, XML).

Sie können diese Einschränkung umgehen, indem Sie eine Einschränkung verwenden, *um* die akzeptierte Eingabe zu *erweitern* .

Wenn Sie beispielsweise einen Benutzerdatensatz anhand der E-Mail-Adresse in der URL referenzieren möchten:

```
resources :users, constraints: { id: /.*/ }
```

Wurzelroute

Sie können Ihrer App eine Startseite-Route mit der `root` Methode hinzufügen.

```
# 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
```

Im Terminal werden auf `rake routes` (`rails routes` in Rails 5) Folgendes erzeugt:

```
root     GET    /         application#index
```

Da die Startseite normalerweise die wichtigste Route ist und die Routen in der Reihenfolge, in der sie angezeigt werden, priorisiert werden, sollte die `root` normalerweise die erste in der Routendatei sein.

Zusätzliche RESTful-Aktionen

```
resources :photos do
  member do
    get 'preview'
  end
  collection do
    get 'dashboard'
  end
end
```

Dadurch werden die folgenden Routen **zusätzlich zu den standardmäßig 7 RESTful-Routen erstellt** :

```
get       '/photos/:id/preview',          to: 'photos#preview'
get       '/photos/dashboards',           to: 'photos#dashboard'
```

Wenn Sie dies für einzelne Zeilen tun möchten, können Sie Folgendes verwenden:

```
resources :photos do
  get 'preview',   on: :member
  get 'dashboard', on: :collection
end
```

Sie können dem `/new` Pfad auch eine Aktion hinzufügen:

```
resources :photos do
  get 'preview', on: :new
end
```

Was wird schaffen:

```
get       '/photos/new/preview',          to: 'photos#preview'
```

Achten Sie beim Hinzufügen von Aktionen zu Ihren REST-Routen, wahrscheinlich fehlt Ihnen eine andere Ressource!

Verfügbare Gebietsschemas

Wenn Ihre Anwendung in verschiedenen Sprachen verfügbar ist, wird normalerweise das aktuelle Gebietsschema in der URL angezeigt.

```
scope '/(:locale)', locale: /#{I18n.available_locales.join('|')}/ do
    root 'example#root'
    # other routes
end
```

Ihr Stamm ist über die in `I18n.available_locales` definierten `I18n.available_locales` .

Mounten Sie eine andere Anwendung

mount wird verwendet, um eine andere Anwendung (hauptsächlich Rack-Anwendung) oder Schienen-Engines für die Verwendung in der aktuellen Anwendung zu installieren

Syntax:

```
mount SomeRackApp, at: "some_route"
```

Jetzt können Sie mit dem `some_rack_app_path` oder `some_rack_app_url` auf die oben eingebaute Anwendung `some_rack_app_url` .

Wenn Sie diesen Helfernamen umbenennen möchten, können Sie dies folgendermaßen tun:

```
mount SomeRackApp, at: "some_route", as: :myapp
```

Dadurch werden die Helfer `myapp_path` und `myapp_url` generiert, mit denen Sie zu dieser `myapp_url` App navigieren können.

Weiterleitungen und Wildcard-Routen

Wenn Sie Ihrem Benutzer eine URL zur Verfügung stellen möchten, die sich jedoch direkt auf eine andere bezieht, die Sie bereits verwenden. Verwenden Sie eine Weiterleitung:

```
# config/routes.rb
TestApp::Application.routes.draw do
  get 'courses/:course_name' => redirect('/courses/%{course_name}/lessons'), :as => "course"
end
```

Nun, das wurde schnell interessant. Das grundlegende Prinzip besteht darin, einfach die `#redirect` Methode zu verwenden, um eine Route an eine andere Route zu senden. Wenn Ihre Route recht einfach ist, ist dies eine sehr unkomplizierte Methode. Wenn Sie jedoch auch die ursprünglichen Parameter senden möchten, müssen Sie etwas Gymnastik machen, indem Sie den Parameter in `%{here}` erfassen. Beachten Sie die einfachen Anführungszeichen um alles herum.

Im obigen Beispiel haben wir die Route auch umbenannt, indem Sie einen Alias mit dem Parameter: as verwenden. Dadurch können wir diesen Namen in Methoden wie den #_path-Helfern verwenden. Testen Sie erneut Ihre `$ rake routes` mit Fragen.

Trennen Sie die Routen in mehrere Dateien

Wenn Ihre Routendatei überwältigend groß ist, können Sie Ihre Routen in mehreren Dateien

ablegen und jede der Dateien mit der Ruby- `require_relative` Methode aufnehmen:

`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
```

Verschachtelte Routen

Wenn Sie verschachtelte Routen hinzufügen möchten, können Sie den folgenden Code in `routes.rb` Datei `routes.rb` schreiben.

```
resources :admins do
  resources :employees
end
```

Dadurch werden folgende Routen generiert:

```
     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
```

Routing online lesen: https://riptutorial.com/de/ruby-on-rails/topic/307/routing

Kapitel 60: RSpec und Ruby on Rails

Bemerkungen

RSpec ist ein Test-Framework für Ruby oder, wie in der offiziellen Dokumentation definiert, *RSpec ist ein verhaltensgesteuertes Entwicklungstool für Ruby-Programmierer* .

Dieses Thema behandelt die Grundlagen der Verwendung von RSpec mit Ruby on Rails. Weitere Informationen zu RSpec finden Sie im RSpec-Thema .

Examples

RSpec installieren

Wenn Sie RSpec für ein Rails-Projekt verwenden möchten, sollten Sie den Gem `rspec-rails` tracks verwenden, der automatisch Hilfsprogramme und Spezifikationsdateien für Sie generieren kann (z. B. beim Erstellen von Modellen, Ressourcen oder Gerüsten mit Hilfe von `rails generate`).

Fügen Sie `rspec-rails` `Gemfile` sowohl zu `:development` als auch zu den `:test` in der `Gemfile` :

```
group :development, :test do
  gem 'rspec-rails', '~> 3.5'
end
```

Führen Sie ein `bundle` , um die Abhängigkeiten zu installieren.

Initialisieren Sie es mit:

```
rails generate rspec:install
```

Dadurch wird ein `spec/` Ordner für Ihre Tests erstellt, zusammen mit den folgenden Konfigurationsdateien:

- `.rspec` enthält Standardoptionen für das Befehlszeilen-Tool `rspec`
- `spec/spec_helper.rb` enthält grundlegende RSpec-Konfigurationsoptionen
- `spec/rails_helper.rb` fügt weitere Konfigurationsoptionen hinzu, die spezifischer für die Verwendung von RSpec und Rails sind.

Alle diese Dateien werden mit vernünftigen Standardwerten geschrieben, um Ihnen den Einstieg zu erleichtern. Sie können jedoch Funktionen hinzufügen und Konfigurationen ändern, um Ihre Anforderungen anzupassen, wenn Ihre Testsuite wächst.

RSpec und Ruby on Rails online lesen: https://riptutorial.com/de/ruby-on-rails/topic/5335/rspec-und-ruby-on-rails

Kapitel 61: Schienen 5

Examples

Erstellen einer Ruby on Rails 5-API

Um eine neue Rails 5-API zu erstellen, öffnen Sie ein Terminal und führen Sie den folgenden Befehl aus:

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

Die folgende Dateistruktur wird erstellt:

```
  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
```

Diese Dateistruktur wird in einem neuen Ordner mit dem Namen `app_name` . Es enthält alle Assets und Code, die zum Starten Ihres Projekts erforderlich sind.

Geben Sie den Ordner ein und installieren Sie die Abhängigkeiten:

```
cd app_name
bundle install
```

Sie sollten auch Ihre Datenbank starten. Rails verwendet SQLite als Standarddatenbank. Um es zu erstellen, führen Sie Folgendes aus:

```
rake db:setup
```

Führen Sie jetzt Ihre Anwendung aus:

```
$ rails server
```

Wenn Sie Ihren Browser unter `http://localhost:3000` öffnen, sollte Ihre glänzende neue (leere) API

ausgeführt werden!

So installieren Sie Ruby on Rails 5 auf RVM

RVM ist ein großartiges Werkzeug, um Ihre Ruby-Versionen zu verwalten und Ihre Arbeitsumgebung einzurichten.

Wenn Sie bereits RVM installiert haben, erhalten Sie die neueste Version von Ruby, die für diese Beispiele erforderlich ist, ein Terminal und öffnen Sie:

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

Überprüfen Sie Ihre Ruby-Version, indem Sie Folgendes ausführen:

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

Um Rails 5 zu installieren, erstellen Sie zuerst ein neues Gemset mit der neuesten Ruby-Version und installieren Sie dann die Schienen:

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

Um Ihre Schienenversion zu überprüfen, führen Sie Folgendes aus:

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

Schienen 5 online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3019/schienen-5

Kapitel 62: Schienengerüste im Laufe der Jahre

Einführung

Wenn Sie mit Rails noch nicht vertraut sind und mit älteren Rails-Anwendungen arbeiten, kann es verwirrend sein, welches Framework wann eingeführt wurde. Dieses Thema ist die *endgültige* Liste aller Frameworks in allen Rails-Versionen.

Examples

Wie finde ich heraus, welche Frameworks in der aktuellen Version von Rails verfügbar sind?

Verwenden Sie die

    config.frameworks

Option, um ein Array von `Symbol` abzurufen, die jedes Framework darstellen.

Rails-Versionen in Rails 1.x

- ActionMailer
- ActionPack
- ActionWebService
- Aktiver Rekord
- ActiveSupport
- Railties

Schienengerüste in Schienen 2.x

- ActionMailer
- ActionPack
- Aktiver Rekord
- *ActiveResource* (*ActiveWebService wurde durch ActiveResource ersetzt, und damit wurde Rails standardmäßig von SOAP nach REST verschoben*)
- ActiveSupport
- Railties

Schienengerüste in Schienen 3.x

- ActionMailer
- ActionPack
- ActiveModel

- Aktiver Rekord
- ActiveResource
- ActiveSupport
- Railties

Schienengerüste im Laufe der Jahre online lesen: https://riptutorial.com/de/ruby-on-rails/topic/8107/schienengeruste-im-laufe-der-jahre

Kapitel 63: Schienen-Kochbuch - Fortgeschrittene Schienen-Rezepte / -Lernen und Kodierungstechniken

Examples

Spielen mit Tischen unter Verwendung der Schienenkonsole

Tabellen anzeigen

```
ActiveRecord::Base.connection.tables
```

Löschen Sie eine Tabelle .

```
    ActiveRecord::Base.connection.drop_table("users")
    ------------OR----------------------
    ActiveRecord::Migration.drop_table(:users)
    ------------OR----------------------
    ActiveRecord::Base.connection.execute("drop table users")
```

Index aus vorhandener Spalte entfernen

```
 ActiveRecord::Migration.remove_index(:users, :name => 'index_users_on_country')
```

Dabei ist `country` ein Spaltenname in der Migrationsdatei mit **bereits** hinzugefügtem Index in der `users` (siehe unten): -

```
 t.string :country,add_index: true
```

Entfernen Sie die Fremdschlüsseleinschränkung

```
ActiveRecord::Base.connection.remove_foreign_key('food_items', 'menus')
```

wo die `menus has_many food_items` und ihre jeweiligen Migrationen enthalten.

Spalte hinzufügen

```
ActiveRecord::Migration.remove_column :table_name, :column_name
```

zum Beispiel:-

```
 ActiveRecord::Migration.add_column :profiles, :profile_likes, :integer, :default => 0
```

Rails-Methoden - Rückgabe boolescher Werte

Jede Methode im Rails-Modell kann einen booleschen Wert zurückgeben.

einfache Methode

```
##this method return ActiveRecord::Relation
def check_if_user_profile_is_complete
  User.includes( :profile_pictures,:address,:contact_detail).where("user.id = ?",self)
end
```

Wieder einfache Methode, die boolesche Werte zurückgibt.

```
##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
```

Die gleiche Methode gibt jetzt also boolesche Werte zurück :).

Umgang mit dem Fehler - undefined Methode `where 'für

Manchmal möchten wir eine `where` Abfrage für eine zurückgegebene Auflistung von Datensätzen verwenden, die nicht ActiveRecord `ActiveRecord::Relation` .Hence ist. Wir erhalten den obigen Fehler als `Where` Klausel in `ActiveRecord` und nicht in `Array` .

Es gibt eine präzise Lösung dafür, indem Sie `Joins` .

BEISPIEL : -

> Angenommen, ich muss alle aktiven Benutzerprofile (UserProfile) suchen, bei denen es sich nicht um einen Benutzer (Benutzer) mit einer ID = 10 handelt.

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).map(&:user).where.not(:id=>10)
```

Die obige Abfrage schlägt daher nach der `map` fehl, da die `map` ein `array` das mit der `where` Klausel **nicht** funktioniert.

Aber mit Joins wird es funktionieren,

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).joins(:user).where.not(:id=>10)
```

Bei `joins` werden ähnliche Datensätze wie `map` ausgegeben, es handelt sich jedoch um `ActiveRecord` und **nicht um** ein `Array` .

Schienen-Kochbuch - Fortgeschrittene Schienen-Rezepte / -Lernen und Kodierungstechniken online lesen: https://riptutorial.com/de/ruby-on-rails/topic/7259/schienen-kochbuch---fortgeschrittene-schienen-rezepte----lernen-und-kodierungstechniken

Kapitel 64: Schienenlogger

Examples

Rails.logger

Verwenden Sie immer `Rails.logger.{debug|info|warn|error|fatal}` eher als `puts` . Dadurch können Ihre Protokolle in das Standardprotokollformat passen, über einen Zeitstempel und eine Stufe verfügen, sodass Sie auswählen können, ob sie für die Anzeige in einer bestimmten Umgebung wichtig genug sind. Sie können die separaten Protokolldateien für Ihre Anwendung unter `log/` directory mit dem Namen der Rail-App-Umgebung anzeigen. wie: `development.log` oder `production.log` oder `staging.log`

Mit LogRotate können Sie Schienenprotokolle einfach drehen. Sie müssen nur eine kleine Konfiguration wie unten beschrieben vornehmen

Öffnen Sie `/etc/logrotate.conf` mit Ihrem bevorzugten Linux-Editor `vim` oder `nano` und fügen Sie den unten stehenden Code in diese Datei unten ein.

```
/YOUR/RAILSAPP/PATH/log/*.log {
  daily
  missingok
  rotate 7
  compress
  delaycompress
  notifempty
  copytruncate
}
```

So **funktioniert es** Das ist fantastisch einfach. Jedes Bit der Konfiguration hat folgende Funktionen:

- **täglich** - Drehen Sie die Protokolldateien täglich. Sie können stattdessen hier auch wöchentlich oder monatlich verwenden.
- **missingok** - Wenn die Protokolldatei nicht vorhanden ist, ignorieren Sie sie
- **drehen Sie 7** - Bewahren Sie nur 7 Tage Protokolle auf
- **compress** - GZip der Protokolldatei bei Rotation
- **delaycompress** - Drehen Sie die Datei um einen Tag und komprimieren Sie sie am nächsten Tag, um sicherzugehen, dass sie den Rails-Server nicht stört
- **notifempty** - Drehen Sie die Datei nicht, wenn die Protokolle leer sind
- **copytruncate** - **Kopiert** die Protokolldatei und leert sie. Dadurch wird sichergestellt, dass die Protokolldatei, in die Rails geschrieben wird, immer vorhanden ist, sodass Sie keine Probleme bekommen, da sich die Datei nicht wirklich ändert. Wenn Sie dies nicht verwenden, müssen Sie Ihre Rails-Anwendung jedes Mal neu starten.

Logrotate ausführen Da wir diese Konfiguration gerade geschrieben haben, möchten Sie sie testen.

Um Logrotate manuell auszuführen, führen Sie einfach Folgendes aus: `sudo /usr/sbin/logrotate -f /etc/logrotate.conf`

Das ist es.

Schienenlogger online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3904/schienenlogger

Kapitel 65: Schienen-Motoren

Einführung

Engines können als Miniaturanwendungen betrachtet werden, die ihren Hostanwendungen Funktionalität bieten. Eine Rails-Anwendung ist eigentlich nur eine "aufgeladene" Engine. Die Rails :: Application-Klasse erbt viel von ihrem Verhalten von Rails :: Engine.

Engines sind die wiederverwendbaren Schienenanwendungen / Plugins. Es funktioniert wie ein Juwel. Bekannte Motoren sind Device, Spree-Edelsteine, die problemlos in Schienenanwendungen integriert werden können.

Syntax

- `rails plugin new [engine name] --mountable`

Parameter

Parameter	Zweck
--montierbar	Diese Option teilt dem Generator mit, dass Sie eine "mountable" und mit Namespace isolierte Engine erstellen möchten
--voll	Diese Option teilt dem Generator mit, dass Sie eine Engine erstellen möchten, einschließlich einer Skelettstruktur

Bemerkungen

Engines sind sehr gute Optionen, um wiederverwendbare Plugins für Schienenanwendungen zu erstellen

Examples

Berühmte Beispiele sind

Einfache Blog-Engine generieren

```
rails plugin new [engine name] --mountable
```

Berühmte Motorenbeispiele sind

Gerät (Authentifizierungsstein für Schienen)

Spree (E-Commerce)

Schienen-Motoren online lesen: https://riptutorial.com/de/ruby-on-rails/topic/10881/schienen-motoren

Kapitel 66: Sichere Speicherung von Authentifizierungsschlüsseln

Einführung

Viele APIs von Drittanbietern erfordern einen Schlüssel, um Missbrauch zu verhindern. Wenn Sie einen Schlüssel ausstellen, ist es sehr wichtig, dass Sie den Schlüssel nicht in ein öffentliches Repository einbinden, da andere Benutzer Ihren Schlüssel stehlen können.

Examples

Authentifizierungsschlüssel mit Figaro speichern

Füge Gem `gem 'figaro'` zu deinem Gemfile hinzu und führe das `bundle install` . Führen Sie dann das `bundle exec figaro install` . Dadurch wird config / application.yml erstellt und zu Ihrer .gitignore-Datei hinzugefügt, sodass es nicht zur Versionskontrolle hinzugefügt wird.

Sie können Ihre Schlüssel in application.yml in diesem Format speichern:

```
SECRET_NAME: secret_value
```

Dabei sind SECRET_NAME und secret_value der Name und der Wert Ihres API-Schlüssels.

Sie müssen diese Geheimnisse auch in config / secrets.yml benennen. Sie können in jeder Umgebung unterschiedliche Geheimnisse haben. Die Datei sollte so aussehen:

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

Die Art und Weise, wie Sie diese Schlüssel verwenden, ist unterschiedlich. `some_component` in der Entwicklungsumgebung benötigt Zugriff auf `secret_name` . In config /virones / development.rb würden Sie Folgendes eingeben:

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

Angenommen, Sie möchten eine Produktionsumgebung auf Heroku einrichten. Dieser Befehl lädt die Werte in config /virones / production.rb in Heroku hoch:

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

Sichere Speicherung von Authentifizierungsschlüsseln online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9711/sichere-speicherung-von-authentifizierungsschlusseln

Kapitel 67: Sicheres Konstantisieren

Examples

Erfolgreiche safe_constantize

`User` ist ein `ActiveRecord` oder `Mongoid` Klasse. Ersetzen Sie den `User` durch eine beliebige `Rails` Klasse in Ihrem Projekt (auch etwas wie `Integer` oder `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]
```

Nicht erfolgreich safe_constantize

Dieses Beispiel funktioniert nicht, da die übergebene Zeichenfolge nicht als Konstante im Projekt erkannt wird. Selbst wenn Sie `"array"` , funktioniert es nicht, da es nicht groß geschrieben wird.

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

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

Sicheres Konstantisieren online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3015/sicheres-konstantisieren

Kapitel 68: Standardzeitzone ändern

Bemerkungen

> **config.active_record.default_timezone** legt fest, ob Time.local (wenn auf local gesetzt) oder Time.utc (wenn auf utc gesetzt) verwendet wird, um Datumsangaben und Zeiten aus der Datenbank zu ziehen. Der Standardwert ist: utc.
> http://guides.rubyonrails.org/configuring.html

Wenn Sie die Zeitzone von **Rails** ändern möchten und **Active Record weiterhin** in der Datenbank in **UTC** speichern möchten, verwenden Sie

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

Wenn Sie die Zeitzone von **Rails** ändern möchten **und Active Record-** Speicherzeiten in dieser Zeitzone haben möchten, verwenden Sie

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

Achtung : Sie sollten wirklich zweimal darüber nachdenken, bevor Sie Zeiten in der Datenbank in einem Nicht-UTC-Format speichern.

> **Hinweis**
> Vergessen Sie nicht, Ihren Rails-Server nach dem Ändern von `application.rb` neu zu starten.

Denken `config.active_record.default_timezone` daran, dass `config.active_record.default_timezone` nur zwei Werte `config.active_record.default_timezone` kann

- **: local** (konvertiert in die in `config.time_zone` definierte `config.time_zone`)
- **: utc** (konvertiert nach UTC)

So finden Sie alle verfügbaren Zeitzonen

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

Examples

Ändern Sie die Zeitzone von Rails, speichern Sie jedoch weiterhin Active

Record in der Datenbank in UTC

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

Rails-Zeitzone ändern UND in dieser Zeitzone Active Record-Speicherzeiten festlegen

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

Standardzeitzone ändern online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3367/standardzeitzone-andern

Kapitel 69: Testen von Rails-Anwendungen

Examples

Gerätetest

Unit-Tests testen Teile der Anwendung isoliert. Normalerweise ist eine zu testende Einheit eine Klasse oder ein Modul.

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

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

Quelle

Diese Art von Test ist so direkt und spezifisch wie möglich.

Test anfordern

Anforderungstests sind End-to-End-Tests, die das Verhalten eines Benutzers imitieren.

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

Quelle

Diese Art von Tests konzentriert sich auf Benutzerabläufe und durchläuft alle Schichten des Systems, manchmal sogar das Rendern von Javascript.

Testen von Rails-Anwendungen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/7853/testen-von-rails-anwendungen

Kapitel 70: Tools für die Codeoptimierung und -bereinigung von Ruby on Rails

Einführung

Wenn Sie Ihren Code während der Entwicklung einer großen Rails-Anwendung sauber und strukturiert halten, kann dies selbst für einen erfahrenen Entwickler eine Herausforderung sein. Glücklicherweise gibt es eine ganze Reihe von Edelsteinen, die diese Arbeit wesentlich erleichtern.

Examples

Wenn Sie Ihren Code wartungsfähig, sicher und optimiert halten möchten, sehen Sie sich einige Juwelen zur Codeoptimierung und -bereinigung an:

Kugel

Dieser hat mich besonders begeistert. Mit dem Bullet-Edelstein können Sie alle N + 1-Abfragen sowie unnötig belastete Beziehungen beenden. Nach der Installation und dem Start verschiedener Routen in der Entwicklung werden Warnfenster mit Warnhinweisen angezeigt, die auf Datenbankabfragen hinweisen, die optimiert werden müssen. Es funktioniert sofort und ist äußerst hilfreich bei der Optimierung Ihrer Anwendung.

Rails Best Practices

Statischer Code-Analysator zum Finden von Rails-spezifischen Codegerüchen. Es bietet eine Vielzahl von Vorschlägen; Verwenden Sie den Bereichszugriff, beschränken Sie automatisch generierte Routen, fügen Sie Datenbankindizes hinzu usw. Trotzdem enthält es viele nette Vorschläge, die Ihnen eine bessere Perspektive geben, wie Sie Ihren Code umgestalten und einige bewährte Methoden kennenlernen.

Rubocop

Ein statischer Ruby-Code-Analysator, mit dem Sie prüfen können, ob Ihr Code den Ruby-Community-Code-Richtlinien entspricht. Der gem meldet Verstöße über die Befehlszeile mit vielen nützlichen Codes, die Goodies umwandeln, wie beispielsweise die Zuweisung unbrauchbarer Variablen, die redundante Verwendung von Object # to_s in der Interpolation oder sogar ein nicht verwendetes Methodenargument.

Das Gute daran ist, dass es in hohem Maße konfigurierbar ist, da der Analysator sehr ärgerlich sein kann, wenn Sie den Ruby-Style-Guide nicht zu 100% befolgen (dh Sie haben viele nachgestellte Whitespaces oder doppelte Ihre Strings, selbst wenn Sie nicht interpolieren usw.). .

Es ist in 4 Unteranalysatoren (Cops) unterteilt: Style, Lint, Metrics und Rails.

Tools für die Codeoptimierung und -bereinigung von Ruby on Rails online lesen: https://riptutorial.com/de/ruby-on-rails/topic/8713/tools-fur-die-codeoptimierung-und--bereinigung-von-ruby-on-rails

Kapitel 71: Turbolinks

Einführung

Turbolinks ist eine Javascript-Bibliothek, die das Navigieren in Ihrer Webanwendung beschleunigt. Wenn Sie einem Link folgen, ruft Turbolinks die Seite automatisch ab, wechselt ihren <Hauptteil> und führt dessen <Kopf> zusammen, ohne dass die Kosten für das vollständige Laden der Seite entstehen.

Bemerkungen

Als Schienenentwickler interagieren Sie während der Entwicklung wahrscheinlich minimal mit Turbolinks. Es ist jedoch eine wichtige Bibliothek, mit der Sie vertraut sein sollten, da sie die Ursache für einige schwer zu findende Fehler sein kann.

Die zentralen Thesen:

- Binden Sie an die `turbolinks:load` Ereignis `turbolinks:load` anstelle des Ereignisses `document.ready`
- Verwenden Sie das Attribut `data-turbolinks-false` , um die Turbolink-Funktion auf Link-Basis zu deaktivieren.
- Verwenden Sie das Attribut " `data-turbolinks-permanent` , um Elemente beim Laden von Seiten zu erhalten und Cache-bezogene Fehler zu vermeiden.

Eine ausführlichere Behandlung von Turbolinks finden Sie im offiziellen Repository von github .

Ein Großteil dieser Dokumentation geht an die Leute, die die Turbolinks-Dokumentation für das Github-Repository entworfen haben.

Examples

Bindung an das Konzept von turbolink zum Laden einer Seite

Mit Turbolinks ist der traditionelle Ansatz zur Verwendung von:

```
$(document).ready(function() {
  // awesome code
});
```

funktioniert nicht Bei Verwendung von Turbolinks wird das Ereignis `$(document).ready()` nur einmal `$(document).ready()` : beim ersten Laden der Seite. Von diesem Zeitpunkt an werden Turbolinks jedes Mal, wenn ein Benutzer auf einen Link auf Ihrer Website klickt, das Link-Click-Ereignis abfangen und eine Ajax-Anforderung stellen, um das <body> -Tag zu ersetzen und die <head> -Tags zusammenzuführen. Der gesamte Prozess löst die Vorstellung eines "Besuchs" in

Turbolinks-Land aus. Anstatt die traditionelle `document.ready()` Syntax oben zu verwenden, müssen Sie sich daher wie folgt an das Besuchereignis von turbolink binden:

```
// pure js
document.addEventListener("turbolinks:load", function() {
  // awesome code
});

// jQuery
$(document).on('turbolinks:load', function() {
  // your code
});
```

Deaktivieren Sie Turbolinks für bestimmte Links

Es ist sehr einfach, Turbolinks für bestimmte Links zu deaktivieren. Laut der offiziellen Dokumentation zu Turbolinks :

> Turbolinks können pro Link deaktiviert werden, indem ein Link oder einer seiner Vorfahren mit data-turbolinks = "false" gekennzeichnet wird.

Beispiele:

```
// 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>
```

Anwendungsbesuche verstehen

Anwendungsbesuche werden durch Klicken auf einen Turbolinks-fähigen Link oder programmgesteuert durch Aufrufen initiiert

```
Turbolinks.visit(location)
```

Die Besuchsfunktion verwendet standardmäßig die Aktion "Weiter". Verständlicherweise ist das Standardverhalten der Besuchsfunktion, zu der durch den Parameter "location" angegebenen Seite zu gelangen. Immer wenn eine Seite besucht wird, schiebt turbolinks mit `history.pushState` einen neuen Eintrag in den Browserverlauf. Der Verlauf ist wichtig, da Turbolinks versuchen, den Verlauf zu verwenden, um Seiten aus dem Cache zu laden, wann immer dies möglich ist. Dies

ermöglicht extrem schnelles Seiten-Rendering für häufig besuchte Seiten.

Wenn Sie jedoch einen Ort besuchen möchten, ohne einen Verlauf auf den Stapel zu legen, können Sie die Aktion "Ersetzen" für die Besuchsfunktion wie folgt verwenden:

```
// using links
<a href="/edit" data-turbolinks-action="replace">Edit</a>

// programatically
Turbolinks.visit("/edit", { action: "replace" })
```

Dadurch wird der obere Teil des Protokollstapels durch die neue Seite ersetzt, sodass die Gesamtzahl der Elemente auf dem Stapel unverändert bleibt.

Es gibt auch eine Aktion zum Wiederherstellen , die Wiederherstellungsbesuchern hilft , die Besuche, die durch das Klicken des Vorwärts- oder Zurückschalters des Benutzers im Browser ausgelöst werden . Turbolinks behandelt diese Arten von Ereignissen intern und empfiehlt Benutzern, das Standardverhalten nicht manuell zu manipulieren.

Besuche werden abgebrochen, bevor sie beginnen

Turbolinks stellt einen Ereignis-Listener bereit, mit dem Besuche verhindert werden können. Hören Sie sich die `turbolinks:before-visit` , der benachrichtigt wird, wenn ein Besuch beginnt.

Im Event-Handler können Sie Folgendes verwenden:

```
// pure javascript
event.data.url
```

oder

```
// jQuery
$event.originalEvent.data.url
```

um den Ort des Besuchs abzurufen. Der Besuch kann dann storniert werden durch:

```
event.preventDefault()
```

HINWEIS:

Laut den offiziellen Turbolinks-Dokumenten :

> Wiederherstellungsbesuche können nicht abgesagt werden und keine Turbolinks abfeuern: vor dem Besuch.

Bestehende Elemente beim Laden von Seiten

Stellen Sie sich folgende Situation vor: Stellen Sie sich vor, Sie sind Entwickler einer Social-

Media-Website, auf der Benutzer mit anderen Benutzern befreundet sein können und Turbolinks verwenden, um das Laden von Seiten zu beschleunigen. Oben rechts auf jeder Seite der Website gibt es eine Zahl, die die Gesamtzahl der Freunde angibt, die ein Benutzer aktuell hat. Stellen Sie sich vor, Sie nutzen Ihre Site und haben 3 Freunde. Immer wenn ein neuer Freund hinzugefügt wird, haben Sie ein Javascript, das den Freundeszähler aktualisiert. Stellen Sie sich vor, Sie haben gerade einen neuen Freund hinzugefügt und Ihr Javascript wurde ordnungsgemäß ausgeführt und die Anzahl der Freunde oben rechts auf der Seite aktualisiert, um jetzt 4 zu rendern. Stellen Sie sich nun vor, dass Sie auf die Schaltfläche "Zurück" des Browsers klicken. Wenn die Seite geladen wird, stellen Sie fest, dass die Freundesanzeige 3 anzeigt, obwohl Sie vier Freunde haben.

Dies ist ein relativ häufiges Problem, für das Turbolinks eine Lösung gefunden haben. Das Problem tritt auf, weil Turbolinks Seiten automatisch aus dem Cache laden, wenn ein Benutzer auf die Zurück-Schaltfläche klickt. Die zwischengespeicherte Seite wird nicht immer mit der Datenbank aktualisiert.

Um dieses Problem zu lösen, stellen Sie sich vor, dass Sie die Anzahl der Freunde in einem -Tag mit der ID "friend-count" darstellen:

```
<div id="friend-count" data-turbolinks-permanent>3 friends</div>
```

Durch das Hinzufügen des Attributs " `data-turbolinks-permanent` Sie den Turbo-Links mit, dass bestimmte Elemente auch beim Laden von Seiten bestehen bleiben sollen. Die offiziellen Dokumente sagen :

> Definieren Sie permanente Elemente, indem Sie ihnen eine HTML-ID zuweisen und sie mit Daten-Turbolinks-Permanent kennzeichnen. Vor jedem Rendern gleicht Turbolinks alle permanenten Elemente anhand der ID ab und überträgt sie von der ursprünglichen Seite auf die neue Seite. Dabei bleiben ihre Daten- und Ereignis-Listener erhalten.

Turbolinks online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9331/turbolinks

Kapitel 72: Vererbung einzelner Tabellen

Einführung

Single Table Inheritance (STI) ist ein Entwurfsmuster, das auf der Idee basiert, die Daten mehrerer Modelle, die alle von demselben Basismodell erben, in einer einzigen Tabelle in der Datenbank zu speichern.

Examples

Grundlegendes Beispiel

Zuerst brauchen wir eine Tabelle, um unsere Daten zu speichern

```
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
```

Dann können wir einige Modelle erstellen

```
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
```

Wenn Sie ein `Guest.create(name: 'Bob')` ausführen, übersetzt `Guest.create(name: 'Bob')` dies, um einen Eintrag in der Tabelle Benutzer mit dem `type: 'Guest'` zu erstellen.

Wenn Sie den Datensatz abrufen `bob = User.where(name: 'Bob').first` das zurückgegebene Objekt eine Instanz von `Guest` , die zwangsweise als Benutzer mit `bob.becomes(User)`

wird am nützlichsten, wenn es sich um gemeinsam genutzte Teilstrecken oder Routen / Controller der Oberklasse anstelle der Unterklasse handelt.

Benutzerdefinierte Vererbungsspalte

Standardmäßig wird der STI-Modellklassenname in einer Spalte namens `type` gespeichert. Der Name kann jedoch durch Überschreiben des `inheritance_column` in einer Basisklasse geändert werden. Z.B:

```
class User < ActiveRecord::Base
  self.inheritance_column = :entity_type # can be string as well
end

class Admin < User; end
```

Die Migration in diesem Fall sieht wie folgt aus:

```
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
```

Wenn Sie `Admin.create` , wird dieser Datensatz mit `entity_type = "Admin"` in der Benutzertabelle `entity_type = "Admin"`

Schienenmodell mit Typsäule und ohne STI

Mit `type` - Spalte in einem Rails - Modell ohne Aufruf STI durch die Zuordnung erreicht werden kann `:_type_disabled` `inheritance_column` :

```
class User < ActiveRecord::Base
  self.inheritance_column = :_type_disabled
end
```

Vererbung einzelner Tabellen online lesen: https://riptutorial.com/de/ruby-on-rails/topic/9125/vererbung-einzelner-tabellen

Kapitel 73: Verschachtelte Form in Ruby on Rails

Examples

So erstellen Sie ein verschachteltes Formular in Ruby on Rails

Das Erste, was zu haben ist: ein Modell, das eine `has_many` Beziehung zu einem anderen Modell enthält.

```
class Project < ApplicationRecord
  has_many :todos
end

class Todo < ApplicationRecord
  belongs_to :project
end
```

In `ProjectsController` :

```
class ProjectsController < ApplicationController
  def new
    @project = Project.new
  end
end
```

In einem verschachtelten Formular können Sie gleichzeitig untergeordnete Objekte mit einem übergeordneten Objekt erstellen.

```
<%= 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 %>
```

Da wir `@project` mit `Project.new` initialisiert haben, um etwas zum Erstellen eines neuen `Project` Objekts zu haben, genauso wie zum Erstellen eines `Todo` Objekts, müssen wir etwas Ähnliches haben, und es gibt mehrere Möglichkeiten, dies zu tun:

1. In `Projectscontroller` können Sie in der `new` Methode `@todo = @project.todos.build` schreiben: `@todo = @project.todos.build` oder `@todo = @project.todos.new` , um ein neues `Todo` Objekt zu instantiieren.

2. Sie können dies auch in der Ansicht tun: `<%= f.fields_for :todos, @project.todos.build %>`

Für starke Params können Sie sie auf folgende Weise hinzufügen:

```
def project_params
  params.require(:project).permit(:name, todo_attributes: [:name])
end
```

Da die `Todo` Objekte durch die Erstellung eines `Project` werden, müssen Sie dies in `Project` model angeben, indem Sie die folgende Zeile hinzufügen:

```
accepts_nested_attributes_for :todos
```

Verschachtelte Form in Ruby on Rails online lesen: https://riptutorial.com/de/ruby-on-rails/topic/8203/verschachtelte-form-in-ruby-on-rails

Kapitel 74: Winkel mit Schienen konfigurieren

Examples

Winkel mit Schienen 101

Schritt 1: Erstellen Sie eine neue Rails-App

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

Schritt 2: Turbolinks entfernen

Das Entfernen von Turbolinks erfordert das Entfernen aus dem Gemfile.

```
gem 'turbolinks'
```

Entfernen Sie die `require` aus `app/assets/javascripts/application.js` :

```
//= require turbolinks
```

Schritt 3: Fügen Sie AngularJS zur Asset-Pipeline hinzu

Damit Angular mit der Rails-Asset-Pipeline arbeiten kann, müssen wir der Gemfile Folgendes hinzufügen:

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

Führen Sie nun den Befehl aus

```
bundle install
```

Fügen Sie `bower` damit wir die AngularJS-Abhängigkeit installieren können:

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

Hinzufügen von Angular zu `bower.json` :

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

  "dependencies": {

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

Nun, da `bower.json` mit den richtigen Abhängigkeiten eingerichtet ist, installieren wir sie:

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

Schritt 4: Organisieren Sie die Angular App

Erstellen Sie die folgende Ordnerstruktur in `app/assets/javascript/angular-app/` :

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

`app/assets/javascripts/application.js` in `app/assets/javascripts/application.js` die `require` für Angular, den Vorlagenhelfer und die Angular-App-Dateistruktur hinzu. So was:

```
//= 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
```

Schritt 5: Starten Sie die Angular App

Erstellen Sie `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=csrftoken]').attr('content') ])  @app.run(->   console.log 'angular app running'
)
```

Erstellen Sie ein Angular-Modul unter `app/assets/javascripts/angular-app/modules/example.js.coffee.erb` :

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

Erstellen Sie einen Angular-Controller für diese App unter `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"  ])
```

Fügen Sie nun eine Route zu Rails hinzu, um die Kontrolle an Angular zu übergeben. In `config/routes.rb` :

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

Generieren Sie den Rails-Controller, um auf diese Route zu reagieren:

```
rails g controller Example
```

In `app/controllers/example_controller.rb` :

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

In der Ansicht müssen wir angeben, welche Angular-App und welcher Angular-Controller diese Seite steuert. Also in `app/views/example/index.html.erb` :

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

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

</div>
```

Starten Sie Ihren Rails-Server und besuchen Sie http: // localhost: 3000 / example , um die App anzuzeigen.

Winkel mit Schienen konfigurieren online lesen: https://riptutorial.com/de/ruby-on-rails/topic/3902/winkel-mit-schienen-konfigurieren

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Ruby on Rails	Abhishek Jain, Adam Lassek, Ajay Barot, animuson, ArtOfCode, Aswathy, Community, Darpan Chhatravala, Darshan Patel, Deepak Mahakale, fybw id, Geoffroy, hschin, hvenables, Jon Wood, kfrz, Kirti Thorat, Lorenzo Baracchi, Luka Kerr, MauroPorrasP, michaelpri, nifCody, Niyanta, olive_tree, RADan, RareFever, Richard Hamilton, sa77, saadlulu, sahil, Sathishkumar Jayaraj, Simone Carletti, Stanislav Valášek, theoretisch, tpei, Undo, uzaif, Yana
2	ActionCable	Ich, Sladey, Undo
3	ActionController	Adam Lassek, Atul Khanduri, Deep, Fire-Dragon-DoL, Francesco Lupo Renzi, jackerman09, RamenChef, Sven Reuter
4	ActionMailer	Adam Lassek, Atul Khanduri, jackerman09, owahab, Phil Ross, Richard Hamilton, Rodrigo Argumedo, William Romero
5	ActiveJob	Brian, owahab
6	ActiveModel	Adam Lassek, RamenChef
7	ActiveRecord-Abfrage-Schnittstelle	Adam Lassek, Ajay Barot, Avdept, br3nt, dnsh, Fabio Ros, Francesco Lupo Renzi, giniouxe, jeffdill2, MikeAndr, Muhammad Abdullah, Niyanta, powerup7, rdnewman, Reboot, Robin, sa77, Vishal Taj PM
8	ActiveRecord-Migrationen	Adam Lassek, Aigars Cibuļskis, Alex Kitchens, buren, Deepak Mahakale, Dharam, DSimon, Francesco Lupo Renzi, giniouxe, Hardik Kanjariya ツ, hschin, jeffdill2, Kirti Thorat, KULKING, maartenvanvliet, Manish Agarwal , Milo P, Mohamad, MZaragoza, nomatteus, Reboot, Richard Hamilton, rii, Robin, Rodrigo Argumedo, rony36, Rory O'Kane, tessi, uzaif, webster
9	ActiveRecord-Sperrung	Adam Lassek, fatfrog, Muaaz Rafi
10	ActiveRecord-Transaktionen	abhas, Adam Lassek
11	ActiveRecord-	Adam Lassek, Colin Herzog, Deepak Mahakale,

	Überprüfungen	dgilperez, dodo121, giniouxe, Hai Pandu, Hardik Upadhyay, mmichael, Muhammad Abdullah, pablofullana , Richard Hamilton
12	ActiveRecord-Verknüpfungen	giniouxe, Hardik Upadhyay, Khanh Pham, Luka Kerr, Manish Agarwal, Niyanta, RareFever, Raynor Kuang, Sapna Jindal
13	ActiveSupport	Adam Lassek
14	Admin-Panel hinzufügen	Ahsan Mahmood, MSathieu
15	Aktive Jobs	tirdadc
16	Aktive Modell-Serialisierer	Flip, owahab
17	Aktiver Rekord	Adam Lassek, AnoE, Bijal Gajjar, br3nt, D-side, Francesco Lupo Renzi, glapworth, jeffdill2, Joel Drapper, Luka Kerr, maartenvanvliet, marcamillion, Mario Uher, powerup7, Sebastialonso, Simone Carletti, Sven Reuter, walid
18	Ändern Sie eine Standardumgebung für Rails-Anwendungen	Whitecat
19	Ansichten	danirod, dgilperez, elasticman, Luka Kerr, MikeC, MMachinegun, Pragash, RareFever
20	Asset-Pipeline	fybw id, Robin
21	Aufbau	Ali MasudianPour, Undo
22	Authentifizieren Sie die API mit Devise	Vishal Taj PM
23	Autorisierung mit CanCan	4444, Ahsan Mahmood, dgilperez, mlabarca, toobulkeh
24	Benutzerauthentifizierung in Rails	Abhinay, Ahsan Mahmood, Antarr Byrd, ArtOfCode, dgilperez, Kieran Andrews, Luka Kerr, Qchmqs, uzaif,
25	Bereitstellung einer Rails-App auf Heroku	B Liu, hschin
26	Bezahlfunktion in Schienen	ppascualv, Sathishkumar Jayaraj
27	Caching	ArtOfCode, Cuisine Hacker, Khanh Pham, RamenChef, tirdadc
28	Datei-Uploads	Sergey Khmelevskoy

29	Debuggen	Adam Lassek, Dénes Papp, Dharam, Kelseydh, sa77, titan
30	Dekorateur-Muster	Adam Lassek
31	Edelsteine	Deep, hschin, ma_il, MMachinegun, RamenChef
32	Elasticsearch	Don Giovanni, Luc Boissaye
33	Fabrikmädchen	Rafael Costa
34	Flaches Routing	Darpan Chhatravala
35	Freundliche ID	Thang Le Sy
36	Garnelen-PDF	Awais Shafqat
37	GoogleMaps mit Rails verwenden	fiedl
38	Helfer bilden	aisflat439, owahab, Richard Hamilton, Simon Tsang, Slava.K
39	Hinzufügen eines Amazon RDS zu Ihrer Rails-Anwendung	Sathishkumar Jayaraj
40	I18n - Internationalisierung	Cyril Duchon-Doris, Francesco Lupo Renzi, Frederik Spang, gwcodes, Jorge Najera T, Lahiru, RamenChef
41	Importieren Sie ganze CSV-Dateien aus einem bestimmten Ordner	fool
42	Klassenorganisation	Deep, hadees, HParker
43	Mehrzweck-ActiveRecord-Spalten	Fabio Ros
44	Modellzustände: AASM	Lomefin
45	Mongoid	Ryan K, tes
46	Rails 5-API-Authentifizierung	HParker
47	Rails auf Docker	ppascualv, Sathishkumar Jayaraj
48	Rails aufrüsten	hschin, michaelpri, Rodrigo Argumedo
49	Rails Best Practices	Adam Lassek, Brandon Williams, Gaston, giniouxe,

		Hardik Upadhyay, inye, Joel Drapper, Josh Caswell, Luka Kerr, ma_il, msohng, Muaaz Rafi, piton4eg, powerup7, rony36, Sri, Tom Lazar
50	Rails Engine - Modulare Schienen	Mayur Shah
51	Rails erzeugen Befehle	Adam Lassek, ann, Deepak Mahakale, Dharam, Hardik Upadhyay, jackerman09, Jeremy Green, marcamillion, Milind, Muhammad Abdullah, nomatteus, powerup7, Reub, Richard Hamilton
52	Rails-API	Adam Lassek, hschin
53	React.js mithilfe von Hyperloop in Rails integrieren	Mitch VanDuyn
54	Reagiere mit Rails mit Reaktiver Gleis gem	Kimmo Hintikka, tirdadc
55	Regeln der Namensgebung	Andrey Deineko, Atul Khanduri, br3nt, Flambino, giniouxe, hgsongra, Luka Kerr, Marko Kacanski, Muhammad Abdullah, Sven Reuter, Xinyang Li
56	Reservierte Wörter	Emre Kurt
57	Routing	Adam Lassek, advishnuprasad, Ahsan Mahmood, Alejandro Babio, Andy Gauge, AppleDash, ArtOfCode, Baldrick, cl3m, Cyril Duchon-Doris, Deepak Mahakale, Dharam, Eliot Sykes, esthervillars, Fabio Ros, Fire-Dragon-DoL, Francesco Lupo Renzi, giniouxe, Giuseppe , Hassan Akram, Hizqeel, HungryCoder, jkdev, John Slegers, Jon Wood, Kevin Sylvestre, Kieran Andrews, Kirti Thorat, KULKING, Leito, Mario Uher, Milind, Muhammad Faisal Iqbal, niklashultstrom, nuclearpidgeon , pastullo, Rahul Singh, rap-2-h, Raynor Kuang, Richard Hamilton, Robin, rogerdpack, Rory O'Kane, Ryan Hilbert, Ryan K, Silviu Simeria, Simone Carletti, sohail khalil, Stephen Leppik, TheChamp, Thor odinson, Undo, Zoran,
58	RSpec und Ruby on Rails	Ashish Bista, Scott Matthewman, Simone Carletti
59	Schienen 5	thiago araujo
60	Schienengerüste im Laufe der Jahre	Shivasubramanian A
61	Schienen-Kochbuch -	Milind

	Fortgeschrittene Schienen-Rezepte / -Lernen und Kodierungstechniken	
62	Schienenlogger	Alejandro Montilla, hgsongra
63	Schienen-Motoren	Deepak Kabbur
64	Sichere Speicherung von Authentifizierungsschlüsseln	DawnPaladin
65	Sicheres Konstantisieren	Eric Bouchut, Ryan K
66	Standardzeitzone ändern	Mihai-Andrei Dinculescu
67	Testen von Rails-Anwendungen	HParker
68	Tools für die Codeoptimierung und -bereinigung von Ruby on Rails	Akshay Borade
69	Turbolinks	Mark
70	Vererbung einzelner Tabellen	Niyanta, Ruslan, Slava.K, toobulkeh, Vishal Taj PM
71	Verschachtelte Form in Ruby on Rails	Arslan Ali
72	Winkel mit Schienen konfigurieren	B8vrede, Rory O'Kane, Umar Khan