eBook Gratuit

APPRENEZ Ruby on Rails

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#ruby-on-rails

Table des matières

À propos

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

Chapitre 1: Démarrer avec Ruby on Rails

Remarques

Ruby on Rails (RoR), ou Rails, est un framework d'applications Web populaire open-source. Rails utilise Ruby, HTML, CSS et JavaScript pour créer une application Web exécutée sur un serveur Web. Rails utilise le modèle MVC (model-view-controller) et fournit un ensemble complet de bibliothèques de la base de données à la vue.

Versions

Version	Date de sortie
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

Version	Date de sortie
1.0	2005-12-13

Examples

Création d'une application Ruby on Rails

Cet exemple suppose que *Ruby* et *Ruby on Rails* ont déjà été installés correctement. Sinon, vous pouvez trouver comment le faire **ici** .

Ouvrez une ligne de commande ou un terminal. Pour générer une nouvelle application rails, utilisez la nouvelle commande rails suivie du nom de votre application:

```
$ rails new my_app
```

Si vous souhaitez créer votre application Rails avec une version Rails spécifique, vous pouvez le spécifier au moment de la génération de l'application. Pour ce faire, utilisez `rails _version_ new` suivi du nom de l'application:

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

Cela créera une application Rails appelée `MyApp` dans un répertoire `my_app` et installera les dépendances gem déjà mentionnées dans `Gemfile` utilisant une `bundle install` .

Pour basculer vers le répertoire de votre application nouvellement créée, utilisez la commande `cd` , qui signifie « `change directory` .

```
$ cd my_app
```

Le répertoire `my_app` contient un certain nombre de fichiers et de dossiers générés automatiquement qui constituent la structure d'une application Rails. Voici une liste des fichiers et dossiers créés par défaut:

Dossier de fichiers	Objectif
app /	Contient les contrôleurs, les modèles, les vues, les assistants, les mailers et les ressources de votre application.
poubelle/	Contient le script de rails qui démarre votre application et peut contenir d'autres scripts que vous utilisez pour configurer, mettre à jour, déployer ou exécuter votre application.
config /	Configurez les routes, la base de données et plus encore de votre application.

Dossier de fichiers	Objectif
config.ru	Configuration en rack pour les serveurs basés sur Rack utilisés pour démarrer l'application.
db /	Contient votre schéma de base de données actuel, ainsi que les migrations de base de données.
Gemfile Gemfile.lock	Ces fichiers vous permettent de spécifier les dépendances de gem requises pour votre application Rails. Ces fichiers sont utilisés par la gemme Bundler.
lib /	Modules étendus pour votre application.
bûche/	Fichiers journaux d'application.
Publique/	Le seul dossier vu par le monde tel quel. Contient des fichiers statiques et des actifs compilés.
Rakefile	Ce fichier localise et charge les tâches pouvant être exécutées à partir de la ligne de commande. Les définitions de tâches sont définies dans les composants de Rails.
README.md	Ceci est un bref manuel d'instructions pour votre application. Vous devez éditer ce fichier pour indiquer aux autres ce que fait votre application, comment la configurer, etc.
tester/	Tests unitaires, appareils et autres appareils de test.
temp /	Fichiers temporaires (comme les fichiers de cache et de pid).
vendeur/	Un lieu pour tout code tiers. Dans une application Rails classique, cela inclut les gemmes vendues.

Maintenant, vous devez créer une base de données à partir de votre fichier `database.yml` :

5.0

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

5.0

```
rake db:create
```

Maintenant que nous avons créé la base de données, nous devons exécuter des migrations pour configurer les tables:

5.0

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

5.0

```
rake db:migrate
```

Pour démarrer l'application, nous devons lancer le serveur:

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

Par défaut, les rails démarreront l'application sur le port 3000. Pour lancer l'application avec un numéro de port différent, nous devons lancer le serveur comme suit:

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

Si vous accédez à http: // localhost: 3000 dans votre navigateur, vous verrez une page d'accueil Rails indiquant que votre application est en cours d'exécution.

Si une erreur se produit, plusieurs problèmes peuvent survenir:

- Il y a un problème avec `config/database.yml`
- Vous avez des dépendances dans votre `Gemfile` qui n'ont pas été installées.
- Vous avez des migrations en attente. Run `rails db:migrate`
- Si vous passez aux `rails db:rollback` migration précédents, `rails db:rollback`

Si cela génère toujours une erreur, alors vous devriez vérifier votre `config/database.yml`

Créez une nouvelle application Rails avec votre choix de base de données et l'outil de test RSpec inclus.

Rails utilise `sqlite3` comme base de données par défaut, mais vous pouvez générer une nouvelle application rails avec une base de données de votre choix. Ajoutez simplement l'option `-d` suivie du nom de la base de données.

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

Ceci est une liste (non exhaustive) des options de base de données disponibles:

- mysql
- oracle
- postgresql
- sqlite3
- base avant
- ibm_db

- serveur SQL
- jdbcmysql
- jdbcsqlite3
- jdbcpostgresql
- jdbc

La commande -T indique de sauter l'installation de minitest. Pour installer une suite de tests alternative comme RSpec , éditez le fichier Gemfile et ajoutez

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

Puis lancez la commande suivante depuis la console:

```
rails generate rspec:install
```

Générer un contrôleur

Pour générer un contrôleur (par exemple, `Posts`), accédez à votre répertoire de projet à partir d'une ligne de commande ou d'un terminal, puis exécutez:

```
$ rails generate controller Posts
```

Vous pouvez raccourcir ce code en remplaçant `generate` avec `g` , par exemple:

```
$ rails g controller Posts
```

Si vous ouvrez la nouvelle application / controllers / **posts_controller.rb générée,** vous verrez un contrôleur sans action:

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

Il est possible de créer des méthodes par défaut pour le contrôleur en transmettant les arguments du nom du contrôleur.

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

Pour créer un contrôleur dans un module, spécifiez le nom du contrôleur comme chemin d'accès tel que `parent_module/controller_name` . Par exemple:

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

Cela générera les fichiers suivants:

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

Un contrôleur est simplement une classe définie pour hériter d' `ApplicationController` .

C'est à l'intérieur de cette classe que vous définirez des méthodes qui deviendront les actions de ce contrôleur.

Générer une ressource avec des échafaudages

De guides.rubyonrails.org:

> Au lieu de générer directement un modèle. . . mettons en place un échafaud. Un échafaudage dans Rails est un ensemble complet de modèles, la migration de la base de données pour ce modèle, le contrôleur pour le manipuler, les vues pour afficher et manipuler les données et une suite de tests pour chacun des éléments ci-dessus.

Voici un exemple d'échafaudage d'une ressource appelée `Task` avec un nom de chaîne et une description textuelle:

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

Cela générera les fichiers suivants:

```
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
```

exemple pour supprimer les fichiers générés par échafaudage pour la ressource appelée `Task`

```
rails destroy scaffold Task
```

Créer une nouvelle application Rails avec un adaptateur de base de données non standard

Rails est livré par défaut avec `ActiveRecord` , un ORM (Object Relational Mapping) dérivé du motif du même nom .

En tant qu'ORM, il est conçu pour gérer le mappage relationnel, et plus précisément en gérant les requêtes SQL pour vous, d'où la limitation aux bases de données SQL uniquement.

Cependant, vous pouvez toujours créer une application Rails avec un autre système de gestion de base de données:

1. créez simplement votre application sans enregistrement actif

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

2. ajoutez votre propre système de gestion de base de données dans `Gemfile`

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

3. `bundle install` et suivre les étapes d'installation de la base de données souhaitée.

Dans cet exemple, `mongoid` est un mappage d'objet pour `MongoDB` et, comme beaucoup d'autres gems de base de données construits pour les rails, il hérite également d' `ActiveModel` la même manière `ActiveRecord` , qui fournit une interface commune à de nombreuses fonctionnalités telles que validations, rappels, traductions, etc. .

Les autres adaptateurs de base de données incluent, mais ne sont pas limités à:

- datamapper
- sequel-rails

Création d'API Rails dans JSON

Cet exemple suppose que vous avez de l'expérience dans la création d'applications Rails.

Pour créer une application API uniquement dans Rails 5, exécutez

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

Ajouter active_model_serializers dans Gemfile

```
gem 'active_model_serializers'
```

installer le paquet dans le terminal

```
bundle install
```

Définissez l'adaptateur `ActiveModelSerializer` pour utiliser `: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 )
```

Générer un nouvel échafaudage pour votre ressource

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

Cela générera les fichiers suivants:

Contrôleur: app / controllers / tasks_controller.rb

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

Installation de rails

Installation de Rails sur Ubuntu

Sur un Ubuntu propre, l'installation de Rails doit être simple

Mise à niveau des paquets ubuntu

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

Installer les dépendances Ruby et Rails

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

Installation du gestionnaire de version de Ruby. Dans ce cas, le facile utilise rbenv

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

Installation de Ruby Build

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

Redémarrer la coque

```
exec $SHELL
```

Installer Ruby

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

Installation de rails

```
gem install rails
```

Installation de Rails sous Windows

Étape 1: ***Installation de Ruby***

Nous avons besoin du langage de programmation Ruby. Nous pouvons utiliser une version précompilée de Ruby appelée RubyInstaller.

- Téléchargez et exécutez Ruby Installer à partir de rubyinstaller.org .
- Exécutez le programme d'installation. Cochez "Ajouter des exécutables Ruby à votre PATH", puis installez-le.
- Pour accéder à Ruby, accédez au menu Windows, cliquez sur Tous les programmes, faites défiler jusqu'à Ruby, puis cliquez sur «Démarrer l'invite de commandes avec Ruby». Un terminal d'invite de commande s'ouvre. Si vous tapez `ruby -v` et appuyez sur Entrée, vous devriez voir le numéro de version Ruby que vous avez installé.

Étape 2: ***Kit de développement Ruby***

Après l'installation de Ruby, nous pouvons essayer d'installer Rails. Mais certaines des bibliothèques dont Rails a besoin ont besoin de quelques outils de compilation pour être compilées, et Windows ne dispose pas de ces outils par défaut. Vous pouvez identifier ceci si vous voyez une erreur en essayant d'installer Rails `Gem::InstallError: The '[gem name]' native gem requires installed build tools.` Pour résoudre ce problème, nous devons installer le kit de développement Ruby.

- Téléchargez le DevKit
- Exécutez le programme d'installation.
- Nous devons spécifier un dossier dans lequel nous allons installer définitivement le DevKit. Je recommande de l'installer dans la racine de votre disque dur, à `C:\RubyDevKit` . (N'utilisez pas d'espaces dans le nom du répertoire.)

Nous devons maintenant mettre les outils DevKit à la disposition de Ruby.

- Dans votre invite de commande, accédez au répertoire DevKit. `cd C:\RubyDevKit` ou tout autre répertoire dans `cd C:\RubyDevKit` vous l'avez installé
- Nous devons exécuter un script Ruby pour initialiser la configuration de DevKit. Tapez `ruby dk.rb init` . Nous allons maintenant demander au même script d'ajouter DevKit à notre installation Ruby. Tapez `ruby dk.rb install` .

DevKit devrait maintenant être disponible pour vos outils Ruby lors de l'installation de nouvelles bibliothèques.

Étape 3: ***Rails***

Maintenant, nous pouvons installer Rails. Rails est un joyau Ruby. Dans votre invite de commande, tapez:

```
gem install rails
```

Une fois que vous appuyez sur Entrée, le programme `gem` va télécharger et installer cette version de la gemme Rails, avec tous les autres gemmes dont dépend Rails.

Étape 4: ***Node.js***

Certaines bibliothèques dont dépend Rails nécessitent l'installation d'un script JavaScript. Installons Node.js pour que ces bibliothèques fonctionnent correctement.

- Téléchargez le programme d'installation de Node.js à partir d' **ici** .
- Une fois le téléchargement terminé, visitez votre dossier de téléchargements et exécutez le `node-v4.4.7.pkg` installation `node-v4.4.7.pkg` .
- Lisez l'intégralité du contrat de licence, acceptez les termes et cliquez sur Suivant dans le reste de l'assistant, en laissant tout à la valeur par défaut.
- Une fenêtre peut apparaître pour vous demander si vous souhaitez autoriser l'application à apporter des modifications à votre ordinateur. Cliquez sur "Oui".
- Une fois l'installation terminée, vous devrez redémarrer votre ordinateur pour que Rails puisse accéder à Node.js.

Une fois que votre ordinateur redémarre, n'oubliez pas d'aller dans le menu Windows, cliquez sur «Tous les programmes», faites défiler jusqu'à Ruby et cliquez sur «Démarrer l'invite de commandes avec Ruby».

Lire Démarrer avec Ruby on Rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/225/demarrer-avec-ruby-on-rails

Chapitre 2: ActionCable

Remarques

ActionCable était disponible pour Rails 4.x et était intégré à Rails 5. Il permet une utilisation facile des websockets pour une communication en temps réel entre le serveur et le client.

Examples

[Basique] Côté serveur

```
# 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
```

[Basique] Côté client (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 # généralement généré comme ceci

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

app / assets / javascripts / cable.js # généralement généré comme ceci

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

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

  App.cable = ActionCable.createConsumer();

}).call(this);
```

Authentification d'utilisateur

```
# 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
```

Lire ActionCable en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/1498/actioncable

Chapitre 3: ActionController

Introduction

Action Controller est le C dans MVC. Une fois que le routeur a déterminé quel contrôleur utiliser pour une demande, le contrôleur est responsable de donner un sens à la demande et de produire la sortie.

Le contrôleur reçoit la demande, récupère ou enregistre les données d'un modèle et utilise une vue pour créer une sortie. Un contrôleur peut être considéré comme un intermédiaire entre les modèles et les vues. Il met les données du modèle à la disposition de la vue pour les afficher à l'utilisateur et enregistre ou met à jour les données utilisateur sur le modèle.

Examples

JSON de sortie au lieu de HTML

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

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

De plus, vous aurez besoin de l'itinéraire:

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

Cela répondra de deux manières différentes aux requêtes `/users` :

- Si vous visitez `/users` ou `/users.html` , il affichera une page HTML avec le contenu `Hello World`
- Si vous visitez `/users.json` , un objet JSON contenant:

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

Vous pouvez **omettre** `format.html { render inline: "Hello World" }` si vous voulez vous assurer que votre itinéraire ne répondra qu'aux requêtes JSON.

Contrôleurs (de base)

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

Ceci est un contrôleur de base, avec l'ajout de l'itinéraire suivant (dans routes.rb):

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

Affiche le message `Hello World` dans une page Web lorsque vous accédez à l'URL `/users`

Paramètres

Les contrôleurs ont accès aux paramètres HTTP (vous les connaissez peut-être sous le `?name=foo` dans les URL, mais Ruby on Rails gère également différents formats!) Et affiche différentes réponses basées sur ces derniers. Il n'y a pas de moyen de distinguer les paramètres GET et POST, mais vous ne devriez pas le faire dans tous les cas.

```
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
```

Comme d'habitude notre itinéraire:

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

Accédez à l'URL `/users?name=john` et le résultat sera `Hello John` , access `/users?name=whatever` et le résultat sera `Hello someone`

Paramètres de filtrage (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
```

Vous pouvez autoriser (ou rejeter) certains paramètres de manière à ne laisser *passer* que ce que vous voulez et vous ne rencontrerez pas de mauvaises surprises, comme les options de paramétrage utilisateur qui ne sont pas censées être modifiées.

Visiting `/users?name=john&sentence=developer` affichera `Hello john developer` , en visitant toutefois `/users?name=smith&sentence=spy` affichera `Hello smith` only, car `:sentence` n'est autorisée que lorsque vous accédez en tant que `john`

Redirection

En supposant que l'itinéraire:

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

Vous pouvez rediriger vers une URL différente en utilisant:

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

Vous pouvez revenir à la page précédente que l'utilisateur a visitée en utilisant:

```
redirect_to :back
```

Notez que dans *Rails 5,* la syntaxe de redirection est différente:

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

Qui tentera de rediriger vers la page précédente et au cas où cela ne serait pas possible (le navigateur bloque l'en-tête HTTP_REFERRER), il sera redirigé vers `:fallback_location`

Utiliser des vues

En supposant que l'itinéraire:

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

Et le contrôleur:

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

La vue `app/users/index.html.erb` sera rendue. Si la vue est:

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

Le résultat sera une page Web avec le texte: "Hello **World** "

Si vous souhaitez afficher une vue différente, vous pouvez utiliser:

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

Et le fichier `app/views/pages/home.html.erb` sera utilisé à la place.

Vous pouvez transmettre des variables à des vues à l'aide de variables d'instance de contrôleur:

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

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

Et dans le fichier `app/views/users/index.html.erb` vous pouvez utiliser `@name` :

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

Et le résultat sera: "Bonjour **John** "

Une note importante autour de la syntaxe de rendu, vous pouvez omettre entièrement la syntaxe de `render` , Rails suppose que si vous l'omettez. Alors:

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

Peut être écrit à la place comme:

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

Rails est assez intelligent pour comprendre qu'il doit rendre le fichier
`app/views/users/index.html.erb` .

404 lorsque l'enregistrement n'a pas été trouvé

Erreur de récupération de l'enregistrement introuvable au lieu d'afficher une exception ou une page blanche:

```
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
```

Contrôleur REST de base

```
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
```

Afficher les pages d'erreur pour les exceptions

Si vous voulez afficher à vos utilisateurs des erreurs significatives au lieu de simples "désolé, quelque chose a mal tourné", Rails a un utilitaire intéressant à cet effet.

Ouvrez le fichier `app/controllers/application_controller.rb` et vous devriez trouver quelque chose comme ceci:

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

Nous pouvons maintenant ajouter un `rescue_from` pour récupérer des erreurs spécifiques:

```
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
```

Il est recommandé de ne pas utiliser `Exception` ou `StandardError` sinon Rails ne pourra pas afficher de pages utiles en cas d'erreur.

Des filtres

Les filtres sont des méthodes exécutées "avant", "après" ou "autour" d'une action de contrôleur. Ils sont hérités, donc si vous en définissez dans votre `ApplicationController` ils seront exécutés pour chaque requête reçue par votre application.

Avant filtre

Avant que les filtres ne soient exécutés avant l'action du contrôleur et peuvent arrêter la demande (et / ou la redirection). Une utilisation courante consiste à vérifier si un utilisateur est connecté:

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

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

Avant que les filtres ne soient exécutés sur les requêtes avant que la requête n'arrive à l'action du contrôleur. Il peut renvoyer une réponse elle-même et contourner complètement l'action.

D'autres utilisations courantes des filtres avant sont la validation de l'authentification d'un utilisateur avant de lui accorder l'accès à l'action désignée pour traiter leur demande. Je les ai également vus utilisés pour charger une ressource de la base de données, vérifier les autorisations sur une ressource ou gérer les redirections dans d'autres circonstances.

Après filtre

Après les filtres sont similaires à ceux "avant", mais comme ils sont exécutés après l'exécution de l'action, ils ont accès à l'objet de réponse qui est sur le point d'être envoyé. Donc, en bref, après l'exécution des filtres une fois l'action terminée. Il peut modifier la réponse. La plupart du temps, si quelque chose est fait dans un filtre après, cela peut être fait dans l'action elle-même, mais s'il y a une logique à exécuter après avoir exécuté un ensemble d'actions, alors un filtre après est un bon endroit il.

Généralement, j'ai vu après et autour des filtres utilisés pour la journalisation.

Filtre autour

Autour des filtres peuvent avoir la logique avant et après l'action en cours d'exécution. Il cède simplement à l'action dans n'importe quel endroit est nécessaire. Notez qu'il n'a pas besoin de céder à l'action et peut s'exécuter sans le faire comme un filtre avant.

Les filtres Around sont chargés d'exécuter les actions associées, tout comme les middlewares Rack fonctionnent.

Les callbacks entourent l'exécution des actions. Vous pouvez écrire un rappel autour de deux styles différents. Dans le premier, le rappel est un morceau de code unique. Ce code est appelé avant l'exécution de l'action. Si le code de rappel appelle le rendement, l'action est exécutée. Une fois l'action terminée, le code de rappel continue à s'exécuter. Ainsi, le code avant le rendement est comme un rappel d'action avant et le code après le rendement est le rappel après action. Si le code de rappel n'appelle jamais le rendement. L'action n'est pas exécutée, c'est la même chose que d'avoir un retour de rappel d'action avant false.

Voici un exemple du filtre autour:

```
around_filter :catch_exceptions

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

Cela interceptera toute exception et mettra le message dans votre journal. Vous pouvez utiliser des filtres pour la gestion des exceptions, la configuration et le démontage, ainsi que de nombreux autres cas.

Seulement et sauf

Tous les filtres peuvent être appliqués à des actions spécifiques, en utilisant `:only` and `: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
```

Filtre à sauter

Tous les filtres (ceux hérités aussi) peuvent également être ignorés pour certaines actions spécifiques:

```
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
```

Comme ils sont hérités, les filtres peuvent également être définis dans un contrôleur "parent" d' `namespace` . Dites par exemple que vous avez un espace de noms `admin` et que vous voulez bien sûr que seuls les utilisateurs administrateurs puissent y accéder. Vous pourriez faire quelque chose comme ça:

```
# 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
```

Attention, dans **Rails 4.x,** vous pouvez utiliser `before_filter` avec `before_action` , mais `before_filter` est actuellement obsolète dans **Rails 5.0.0** et sera supprimé dans **5.1** .

Générer un contrôleur

Rails fournit beaucoup de générateurs, pour les contrôleurs aussi bien sûr.

Vous pouvez générer un nouveau contrôleur en exécutant cette commande dans le dossier de votre application.

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

Remarque: Vous pouvez également utiliser des `rails g` alias pour appeler des `rails generate`

Par exemple, pour générer un contrôleur pour un modèle de `Product` , `#show` actions `#index` et `#show`

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

Cela va créer le contrôleur dans `app/controllers/products_controller.rb` , avec les deux actions que vous avez spécifiées

```
class ProductsController < ApplicationController
  def index
  end

  def show
  end
end
```

Il créera également un dossier de `products` dans `app/views/` , contenant les deux modèles pour les actions de votre contrôleur (c.-à-d `index.html.erb` `show.html.erb` et `show.html.erb` , *notez que l'extension peut varier selon votre moteur de template, donc si vous utilise `slim` , par exemple, le générateur va créer `index.html.slim` et `show.html.slim`*)

De plus, si vous avez spécifié des actions, elles seront également ajoutées à votre fichier de `routes` .

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

Rails crée un fichier d'aide dans `app/helpers/products_helper.rb` , ainsi que les fichiers de ressources dans `app/assets/javascripts/products.js` et `app/assets/stylesheets/products.css` . En ce qui concerne les vues, le générateur modifie ce comportement en fonction de ce qui est spécifié dans votre `Gemfile` : par exemple, si vous utilisez `Coffeescript` et `Sass` dans votre application, le générateur de contrôleur `Coffeescript` plutôt `products.coffee` et `products.sass` .

Enfin, Rails génère des fichiers de test pour votre contrôleur, votre assistant et vos vues.

Si vous ne voulez pas que l'un de ces éléments soit créé, vous pouvez demander à Rails de les ignorer.

`--no-` ou `--skip` , comme ceci:

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

Et le générateur ignorera les `assets` et les `helper`

Si vous devez créer un contrôleur pour un **`namespace`** spécifique, ajoutez-le devant `NAME` :

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

Cela va créer votre contrôleur dans `app/controllers/admin/products_controller.rb`

Rails peut également générer un contrôleur RESTful complet pour vous:

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

Sauver ActiveRecord :: RecordNotFound avec redirect_to

Vous pouvez récupérer une exception RecordNotFound avec une redirection au lieu d'afficher une page d'erreur:

```
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
```

Lire ActionController en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2838/actioncontroller

Chapitre 4: ActionMailer

Introduction

Action Mailer vous permet d'envoyer des e-mails depuis votre application à l'aide de classes et de vues de courrier. Les publiposteurs fonctionnent de manière très similaire aux contrôleurs. Ils héritent d'ActionMailer :: Base et vivent dans les applications / les courriers, et ils ont des vues associées qui apparaissent dans les applications / vues.

Remarques

Il est conseillé de traiter l'envoi de courrier électronique de manière asynchrone afin de ne pas bloquer votre serveur Web. Cela peut être fait par le biais de divers services tels que `delayed_job` .

Examples

Basic Mailer

Cet exemple utilise quatre fichiers différents:

- Le modèle d'utilisateur
- Le mailer de l'utilisateur
- Le modèle HTML pour l'e-mail
- Le modèle en texte brut pour l'e-mail

Dans ce cas, le modèle utilisateur appelle la méthode `approved` dans l'expéditeur et transmet le `post` approuvé (la méthode `approved` dans le modèle peut être appelée par un rappel, par une méthode de contrôleur, etc.). Ensuite, l'expéditeur génère l'e-mail à partir du modèle HTML ou texte brut en utilisant les informations du `post` transmis (par exemple, le titre). Par défaut, le publiposteur utilise le modèle avec le même nom que la méthode dans le mailer (c'est pourquoi le nom du mailer et les modèles ont le même nom).

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!
```

Générer un nouveau mailer

Pour générer un nouveau mailer, entrez la commande suivante

```
rails generate mailer PostMailer
```

Cela va générer un fichier de modèle vierge dans `app/mailers/post_mailer.rb` nommé *PostMailer*

```
class PostMailer < ApplicationMailer
end
```

Deux fichiers de mise en page seront également générés pour la vue email, un pour le format html et un pour le format texte.

Si vous préférez ne pas utiliser le générateur, vous pouvez créer vos propres mailers. Assurez-vous qu'ils héritent d' `ActionMailer::Base`

Ajout de pièces jointes

`ActionMailer` permet également de joindre des fichiers.

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

Par défaut, les pièces jointes seront encodées avec `Base64` . Pour changer cela, vous pouvez ajouter un hachage à la méthode des pièces jointes.

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

Vous pouvez également ajouter des pièces jointes en ligne

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

ActionMailer Callbacks

ActionMailer prend en charge trois rappels

- avant_action
- après_action
- around_action

Fournissez-les dans votre classe Mailer

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

Ensuite, créez ces méthodes sous le mot-clé `private`

```
private
  def set_delivery_options
  end

  def prevent_delivery_to_guests
  end

  def set_business_headers
  end
end
```

Générer une newsletter programmée

Créez le modèle de **newsletter** :

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

  subl app/models/newsletter.rb

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

Créer le contrôleur de **newsletter** :

```
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
```

Après cela, changez la vue **create.html.erb** pour le nom nex. Nous allons convertir ce fichier en **vue partielle** qui sera stockée dans le **pied de page** . Le nom sera **_form.html.erb** .

Changer le fichier de nom de:	À:
app / views / **newsletters / create.html.erb**	app / views / **newsletters / _form.html.erb**

Après cela, définissez les itinéraires:

```
subl app/config/routes.rb

resources :newsletters
```

Plus tard, nous devons définir le formulaire que nous utiliserons pour enregistrer chaque courrier:

```
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 %>
```

Et après cela, insérer sur le pied de page:

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

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

Maintenant, installez le - **letter_opener** - to peut prévisualiser le courrier électronique dans le navigateur par défaut au lieu de l'envoyer. Cela signifie que vous n'avez pas besoin de configurer la remise des e-mails dans votre environnement de développement et que vous n'avez plus besoin d'envoyer accidentellement un e-mail de test à l'adresse de quelqu'un d'autre.

Ajoutez d'abord la gemme à votre environnement de développement et exécutez la commande bundle pour l'installer.

```
  subl your_project/Gemfile

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

Ensuite, définissez la méthode de livraison dans l'environnement de développement:

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

  config.action_mailer.delivery_method = :letter_opener
```

Maintenant, créez une **structure de mailer** pour gérer l'ensemble des mailers que nous allons travailler. Dans le terminal

```
  rails generate mailer UserMailer newsletter_mailer
```

Et dans le **UserMailer** , nous devons créer une méthode appelée **Newsletter Mailer** qui sera créée pour contenir le dernier article du blog et sera déclenchée par une action de rake. Nous supposerons que vous aviez une structure de blog créée auparavant.

```
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
```

Après cela, créez le **modèle de mailer** :

```
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>
```

Puisque nous voulons envoyer le courrier électronique en tant que processus distinct, créons une tâche Rake pour envoyer le courrier électronique. Ajoutez un nouveau fichier appelé email_tasks.rake au répertoire lib / tasks de votre application Rails:

```
touch lib/taks/email_tasks.rake

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

L'environnement send_digest_email:: signifie charger l'environnement Rails avant d'exécuter la tâche, vous pouvez donc accéder aux classes d'application (comme UserMailer) dans la tâche.

A présent, l'exécution de la commande rake -T listera la tâche Rake nouvellement créée. Tout tester fonctionne en exécutant la tâche et en vérifiant si l'e-mail est envoyé ou non.

Pour tester si la méthode mailer fonctionne, exécutez la commande rake:

```
  rake weekly_newsletter_email
```

À ce stade, nous avons une tâche de rake qui peut être programmée en utilisant **crontab** . Nous allons donc installer le **Gemme Whenever** qui est utilisé pour fournir une syntaxe claire pour écrire et déployer des tâches cron.

```
subl your_project/Gemfile

  gem 'whenever', require: false
```

Ensuite, exécutez la commande suivante pour créer un fichier config / schedule.rb initial pour vous (tant que le dossier de configuration est déjà présent dans votre projet).

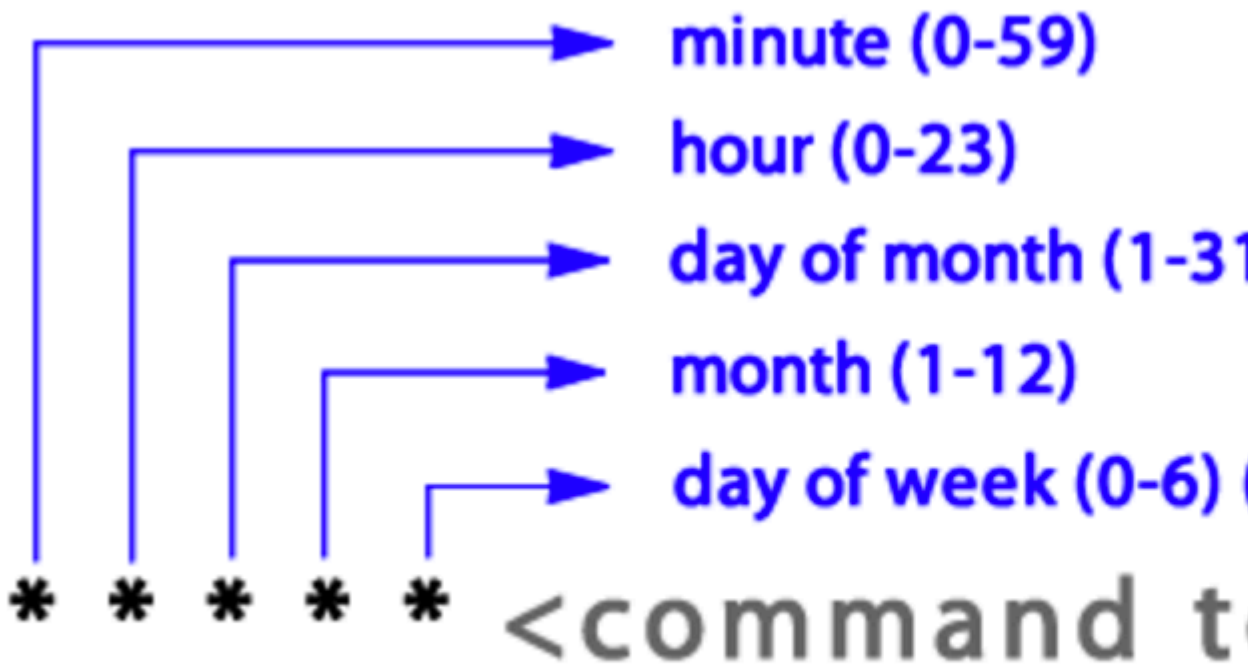
```
  wheneverize .

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

Maintenant, à l'intérieur du fichier de programme, nous devons créer notre **JOB CRON** et appeler la méthode mailer pour déterminer le JOB CRON afin d'exécuter certaines tâches sans assistance et dans une plage de temps sélectionnée. Vous pouvez utiliser différents types de syntaxe, comme expliqué sur ce lien .

```
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
```

été créé avec succès, nous pouvons utiliser la commande suivante pour lire depuis le terminal, notre travail planifié dans CRON SYNTAX:

```
your_project your_mac_user$ whenever

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

Maintenant, pour exécuter le test dans l'environnement de développement, il est judicieux de définir la ligne suivante du fichier principal **application.rb** pour que l'application sache où sont les modèles qu'elle utilisera.

```
  subl your_project/config/application.rb

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

Maintenant, pour laisser **Capistrano V3** enregistrer le nouveau **Job Cron** à l'intérieur du serveur et le déclencheur qui va déclencher l'exécution de cette tâche, nous devons ajouter la condition suivante:

```
  subl your_project/Capfile

  require 'whenever/capistrano'
```

Et insérez dans le fichier de **déploiement** l'identifiant que **CRON JOB** utilisera sur l' **environnement** et le nom de l' **application** .

```
subl your_project/config/deploy.rb

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

Et prêt, après avoir enregistré les modifications sur chaque fichier, lancez la commande capistrano deploy:

```
  cap production deploy
```

Et maintenant votre JOB a été créé et calendarize pour exécuter la méthode Mailer qui est ce que je veux et dans la plage de temps que nous définissons sur ces fichiers.

Intercepteur ActionMailer

Action Mailer fournit des points d'ancrage aux méthodes d'intercepteur. Celles-ci vous permettent d'enregistrer les classes appelées pendant le cycle de vie de la distribution du courrier.

Une classe d'intercepteur doit implémenter la méthode: deliver_email (message) qui sera appelée avant l'envoi du courrier électronique, vous permettant de modifier le courrier électronique avant qu'il ne touche les agents de distribution. Votre classe doit apporter les modifications nécessaires directement à l'instance transmise dans Mail :: Message.

Il peut être utile que les développeurs envoient des e-mails à eux-mêmes et non à de vrais utilisateurs.

Exemple d'enregistrement d'un intercepteur actionmailer:

```
# config/initializers/override_mail_recipient.rb

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

Lire ActionMailer en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2481/actionmailer

Chapitre 5: ActiveJob

Introduction

Active Job est une structure permettant de déclarer des travaux et de les exécuter sur différents serveurs principaux. Ces tâches peuvent aller du nettoyage régulier à la facturation, en passant par les mailings. Tout ce qui peut être découpé en petites unités de travail et exécuté en parallèle, vraiment.

Examples

Créer le job

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

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

Mettre le job en file d'attente

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

Lire ActiveJob en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/8996/activejob

Chapitre 6: ActiveModel

Remarques

ActiveModel a été créé pour extraire le comportement du modèle ActiveRecord dans un souci distinct. Cela nous permet d'utiliser le comportement ActiveModel dans n'importe quel objet, pas seulement les modèles ActiveRecord.

Les objets ActiveRecord incluent tout ce comportement par défaut.

Examples

Utiliser ActiveModel :: Validations

Vous pouvez valider n'importe quel objet, même un simple rubis.

```
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
```

Lire ActiveModel en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/1773/activemodel

Chapitre 7: ActiveRecord

Examples

Créer un modèle manuellement

Bien que l'utilisation de l'échafaudage soit rapide et facile si vous débutez dans Rails ou si vous créez une nouvelle application, il peut être utile de le faire tout seul pour éviter le besoin de parcourir le code généré par l'échafaudage (retirez les pièces inutilisées, etc.).

Créer un modèle peut être aussi simple que de créer un fichier sous `app/models` .

Le modèle le plus simple, dans `ActiveRecord` , est une classe qui étend `ActiveRecord::Base` .

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

Les fichiers modèles sont stockés dans `app/models/` et le nom du fichier correspond au nom singulier de la classe:

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

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

La classe héritera de toutes les fonctionnalités d'ActiveRecord: méthodes de requête, validations, rappels, etc.

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

Remarque: Assurez-vous que la table du modèle correspondant existe. Sinon, vous pouvez créer la table en créant une migration

Vous pouvez générer un modèle et sa migration par terminal à partir de la commande suivante

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

et peut également affecter une clé étrangère (relation) au modèle en suivant la commande

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

Créer un modèle via un générateur

Ruby on Rails fournit un générateur de `model` vous pouvez utiliser pour créer des modèles ActiveRecord. Utilisez simplement des `rails generate model` et indiquez le nom du modèle.

```
$ rails g model user
```

En plus du fichier modèle dans les `app/models` , le générateur créera également:

- le test dans `test/models/user_test.rb`
- les Fixtures dans `test/fixtures/users.yml`
- la migration de la base de données dans `db/migrate/XXX_create_users.rb`

Vous pouvez également générer des champs pour le modèle lors de sa génération.

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

Cela créera les colonnes email, sign_in_count et birthday dans votre base de données, avec les types appropriés.

Créer une migration

Ajouter / supprimer des champs dans des tables existantes

Créez une migration en exécutant:

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

Cela créera une migration qui ajoutera une colonne de `title` à une table de `categories` :

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

De même, vous pouvez générer une migration pour supprimer une colonne: `rails generate migration RemoveTitleFromCategories title:string`

Cela créera une migration qui supprime une colonne de `title` de la table des `categories` :

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

Bien que, à proprement parler, la spécification du **type** (`:string` dans ce cas) **ne** soit **pas nécessaire** pour supprimer une colonne, **elle est utile** , car elle fournit les informations nécessaires pour la **restaurer** .

Créer une table

Créez une migration en exécutant:

```
rails g CreateUsers name bio
```

Rails reconnaît l'intention de créer une table à partir du préfixe `Create` , le reste du nom de la migration sera utilisé comme nom de table. L'exemple donné génère les éléments suivants:

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

Notez que la commande de création n'a pas spécifié les types de colonnes et que la `string` par défaut a été utilisée.

Créer une table de jointure

Créez une migration en exécutant:

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

Rails détecte l'intention de créer une table de jointure en recherchant `JoinTable` dans le nom de la migration. Tout le reste est déterminé à partir des noms des champs que vous donnez après le nom.

```
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
```

Décommentez les instructions d' `index` nécessaires et supprimez le reste.

Priorité

Notez que l'exemple de nom de migration `CreateJoinTableParticipation` correspond à la règle de création de table: il comporte un préfixe `Create` . Mais il n'a pas généré de simple `create_table` . En

effet, le générateur de migration (code source) utilise une **première correspondance** de la liste suivante:

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

Introduction aux rappels

Un rappel est une méthode appelée à des moments spécifiques du cycle de vie d'un objet (juste avant ou après la création, la suppression, la mise à jour, la validation, l'enregistrement ou le chargement depuis la base de données).

Par exemple, disons que vous avez une liste qui expire dans les 30 jours suivant sa création.

Une façon de faire est comme ceci:

```
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
```

Toutes les méthodes disponibles pour les rappels sont les suivantes, dans l'ordre dans lequel elles sont appelées pendant le fonctionnement de chaque objet:

Créer un objet

- before_validation
- after_validation
- avant_save
- autour_save
- avant_créer
- around_create
- after_create
- après_save
- after_commit / after_rollback

Mise à jour d'un objet

- before_validation
- after_validation
- avant_save
- autour_save

- avant_update
- autour_update
- after_update
- après_save
- after_commit / after_rollback

Détruire un objet

- before_destroy
- around_destroy
- after_destroy
- after_commit / after_rollback

REMARQUE: *after_save s'exécute à la fois sur create et update, mais toujours après les rappels plus spécifiques after_create et after_update, quel que soit l'ordre dans lequel les appels de macros ont été exécutés.*

Créer une table de jointure à l'aide de Migrations

Particulièrement utile pour la relation `has_and_belongs_to_many` , vous pouvez créer manuellement une table de jointure à l'aide de la méthode `create_table` . Supposons que vous ayez deux modèles, `Tags` et `Proyects` , et que vous souhaitez les associer en utilisant une relation `has_and_belongs_to_many` . Vous avez besoin d'une table de jointure pour associer des instances des deux classes.

```
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
```

Le nom réel de la table doit suivre cette convention: le modèle qui précède l'autre alphabétiquement doit passer en premier. **P** rojet preceds **T** ags de sorte que le nom de la table est projects_tags.

De plus, l'objectif de cette table étant de router l'association entre les instances de deux modèles, l'ID réel de chaque enregistrement de cette table n'est pas nécessaire. Vous spécifiez ceci en passant `id: false`

Enfin, comme c'est la convention dans Rails, le nom de la table doit être la forme plurielle composée des modèles individuels, mais la colonne de la table doit être au singulier.

Test manuel de vos modèles

Tester vos modèles Active Record via votre interface de ligne de commande est simple. Accédez au répertoire de l'application dans votre terminal et tapez dans la `rails console` pour démarrer la

console Rails. De là, vous pouvez exécuter des méthodes d'enregistrement actives sur votre base de données.

Par exemple, si vous aviez un schéma de base de données avec une table Users portant un `name:string` colonne et `email:string` , vous pourriez exécuter:

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

Ensuite, pour afficher cet enregistrement, vous pouvez exécuter:

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

Ou, s'il s'agit de votre premier ou de votre seul enregistrement, vous pouvez simplement obtenir le premier enregistrement en exécutant:

```
User.first
```

Utilisation d'une instance de modèle pour mettre à jour une ligne

Disons que vous avez un modèle d' `User`

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

Maintenant, pour mettre à jour le `first_name` et le `last_name` d'un utilisateur avec `id = 1` , vous pouvez écrire le code suivant.

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

L'appel de `update` à jour tentera de mettre à jour les attributs donnés dans une transaction unique, renvoyant `true` si l'opération réussit et `false` si ce n'est pas le cas.

Lire ActiveRecord en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/828/activerecord

Chapitre 8: ActiveSupport

Remarques

ActiveSupport est un outil utilitaire d'outils à usage général utilisé par le reste du framework Rails.

L'une des principales façons dont il fournit ces outils consiste à comparer les types natifs de Ruby. Ceux-ci sont appelés **extensions de base** .

Examples

Extensions Core: accès aux chaînes

String # à

Renvoie une sous-chaîne d'un objet chaîne. Même interface que `String#[]` .

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

Chaîne # de

Renvoie une sous-chaîne de la position donnée à la fin de la chaîne.

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

Chaîne # à

Renvoie une sous-chaîne du début de la chaîne à la position donnée.
Si la position est négative, elle est comptée à partir de la fin de la chaîne.

```
str = "hello"
str.to(0)  # => "h"
str.to(3)  # => "hell"
str.to(-2) # => "hell"
```

`from` et `to` peut être utilisé en tandem.

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

String # en premier

Renvoie le premier caractère ou un nombre donné de caractères jusqu'à la longueur de la chaîne.

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

String # last

Renvoie le dernier caractère ou un nombre donné de caractères à partir de la fin de la chaîne en comptant à rebours.

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

Core Extensions: Conversion de chaîne en date / heure

Chaîne # to_time

Convertit une chaîne en valeur Time. Le paramètre de `form` peut être `:utc` ou `:local` , par défaut `: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

Convertit une chaîne en valeur de date.

```
"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
```

String # to_datetime

Convertit une chaîne en valeur DateTime.

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

Extensions de base: exclusion de chaînes

Chaîne # exclure?

L'inverse de `String#include?`

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

Core Extensions: Filtres de chaîne

String # squish

Renvoie une version de la chaîne donnée sans espace de début ou de fin et combine tous les espaces blancs consécutifs à l'intérieur des espaces. Version destructive de `squish!` fonctionne directement sur l'instance de chaîne.

Gère les espaces blancs ASCII et Unicode.

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

String # supprimer

Retourne une nouvelle chaîne avec toutes les occurrences des motifs supprimés. Version destructive `remove!` fonctionne directement sur la chaîne donnée.

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

String # tronqué

Renvoie une copie d'une chaîne donnée tronquée à une longueur donnée si la chaîne est plus longue que la longueur.

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

Passer une chaîne ou une expression rationnelle `:separator` pour tronquer lors d'une rupture naturelle

```
'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

Renvoie une chaîne tronquée après un nombre de mots donné.

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

Passer une chaîne ou une expression rationnelle pour spécifier un séparateur de mots différent

```
'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..."
```

Les derniers caractères seront remplacés par la chaîne `:omission` (par défaut, "...")

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

String # strip_heredoc

Bandes d'indentation dans heredocs. Recherche la ligne non vide la moins en retrait et supprime cette quantité d'espaces principaux.

```
if options[:usage]
```

```
  puts <<-USAGE.strip_heredoc
    This command does such and such.

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

l'utilisateur verrait

```
This command does such and such.

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

Extensions Core: Inflection String

String # pluralize

Renvoie la forme plurielle de la chaîne. Prend éventuellement un paramètre de `count` et retourne une forme singulière si `count == 1` . Accepte également un `locale` paramètres `locale` pour la pluralisation spécifique à la langue.

```
'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 # singulariser

Renvoie la forme singulière de la chaîne. Accepte un `locale` paramètres `locale` facultatif.

```
'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"
```

Chaîne # constantize

Essaie de trouver une constante déclarée avec le nom spécifié dans la chaîne. Il déclenche une `NameError` lorsque le nom n'est pas dans CamelCase ou n'est pas initialisé.

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

String # safe_constantize

Effectue une `constantize` mais renvoie `nil` au lieu de `NameError` .

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

String # camelize

Convertit les chaînes en UpperCamelCase par défaut, si `:lower` est donné en tant que param convertit à lowerCamelCase.

alias: `camelcase`

Remarque: convertira également `/` vers `::` ce qui est utile pour convertir les chemins vers les espaces de noms.

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

String # titleize

Met en majuscule tous les mots et remplace certains caractères de la chaîne pour créer un titre plus joli.

alias: `titlecase`

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

String # souligné

Donne une forme soulignée, en minuscules, de l'expression de la chaîne. Le revers de `camelize` .

Note: le `underscore` changera aussi `::` to `/` pour convertir les espaces de noms en chemins.

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

String # dasherize

Remplace les traits de soulignement par des tirets dans la chaîne.

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

String # démodule

Supprime la partie module de l'expression constante de la chaîne.

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

String # déconstantize

Supprime le segment le plus à droite de l'expression constante dans la chaîne.

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

String # paramétrer

Remplace les caractères spéciaux d'une chaîne de manière à ce qu'elle puisse être utilisée dans le cadre d'une "jolie" URL.

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

Conservez la casse des caractères dans une chaîne avec l'argument `:preserve_case` .

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

Un cas d'utilisation très courant de `parameterize` est de remplacer la méthode `to_param` d'un modèle ActiveRecord pour prendre en charge des `to_param` d'URL plus descriptifs.

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

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

String # tableize

Crée le nom d'une table comme Rails fait pour les modèles aux noms de tables. Pluralise le dernier mot de la chaîne.

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

String # classifier

Renvoie une chaîne de nom de classe à partir d'un nom de table pluriel comme Rails pour les noms de table vers les modèles.

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

Chaîne # humaniser

Met en majuscule le premier mot, transforme les traits de soulignement en espaces et supprime un `_id` si présent.

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

String # upcase_first

Convertit juste le premier caractère en majuscule.

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

String # étrangères_key

Crée un nom de clé étrangère à partir d'un nom de classe. Passez `false` param pour désactiver l'ajout de _ entre nom et `id` .

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

Lire ActiveSupport en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/4490/activesupport

Chapitre 9: Ajout d'une application Amazon RDS à votre application rails

Introduction

Étapes pour créer une instance AWS RDS et configurer votre fichier database.yml en installant les connecteurs requis.

Examples

Considérez que nous connectons MYSQL RDS avec votre application de rails.

Étapes pour créer une base de données MySQL

1. Connectez-vous au compte amazon et sélectionnez le service RDS
2. Sélectionnez `Launch DB Instance` partir de l'onglet instance.
3. Par defaul MYSQL Community Edition sera sélectionné, donc cliquez sur le bouton de `select`
4. Sélectionnez l'objectif de la base de données, dites `production` et cliquez sur `next step`
5. Indiquez la `mysql version, storage size, DB Instance Identifier, Master Username and Password` puis cliquez sur `next step`
6. Entrez le `Database Name` et cliquez sur `Launch DB Instance`
7. Veuillez patienter jusqu'à ce que toute l'instance soit créée. Une fois l'instance créée, vous trouverez un noeud final, copiez ce point d'entrée (appelé nom d'hôte)

Installation de connecteurs

Ajouter l'adaptateur de base de données MySQL au gemfile de votre projet,

```
gem 'mysql2'
```

Installez vos gemmes ajoutées,

```
bundle install
```

Certains autres adaptateurs de base de données sont,

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

Configurez le fichier database.yml de votre projet Ouvrez votre **fichier** 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
```

Lire Ajout d'une application Amazon RDS à votre application rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/10922/ajout-d-une-application-amazon-rds-a-votre-application-rails

Chapitre 10: Ajouter un panneau d'administration

Introduction

Si vous souhaitez ajouter un panneau d'administration à votre application rails, il ne vous faudra que quelques minutes.

Syntaxe

1. Ouvrez le fichier gem et le gem écrivain 'rails_admin', '~> 1.0'
2. installation groupée
3. rails g rails_admin: install
4. il vous posera des questions sur le chemin d'administration si vous souhaitez accéder à la valeur par défaut, appuyez sur Entrée.
5. Maintenant, allez app / config / initializers / rails_admin.rb et collez ce code: config.authorize_with do redirect_to main_app.root_path sauf si current_user.try (: admin?) Fin les utilisateurs seront redirigés vers le rootpath.
6. Pour plus de détails, consultez la documentation de ce joyau. https://github.com/sferik/rails_admin/wiki

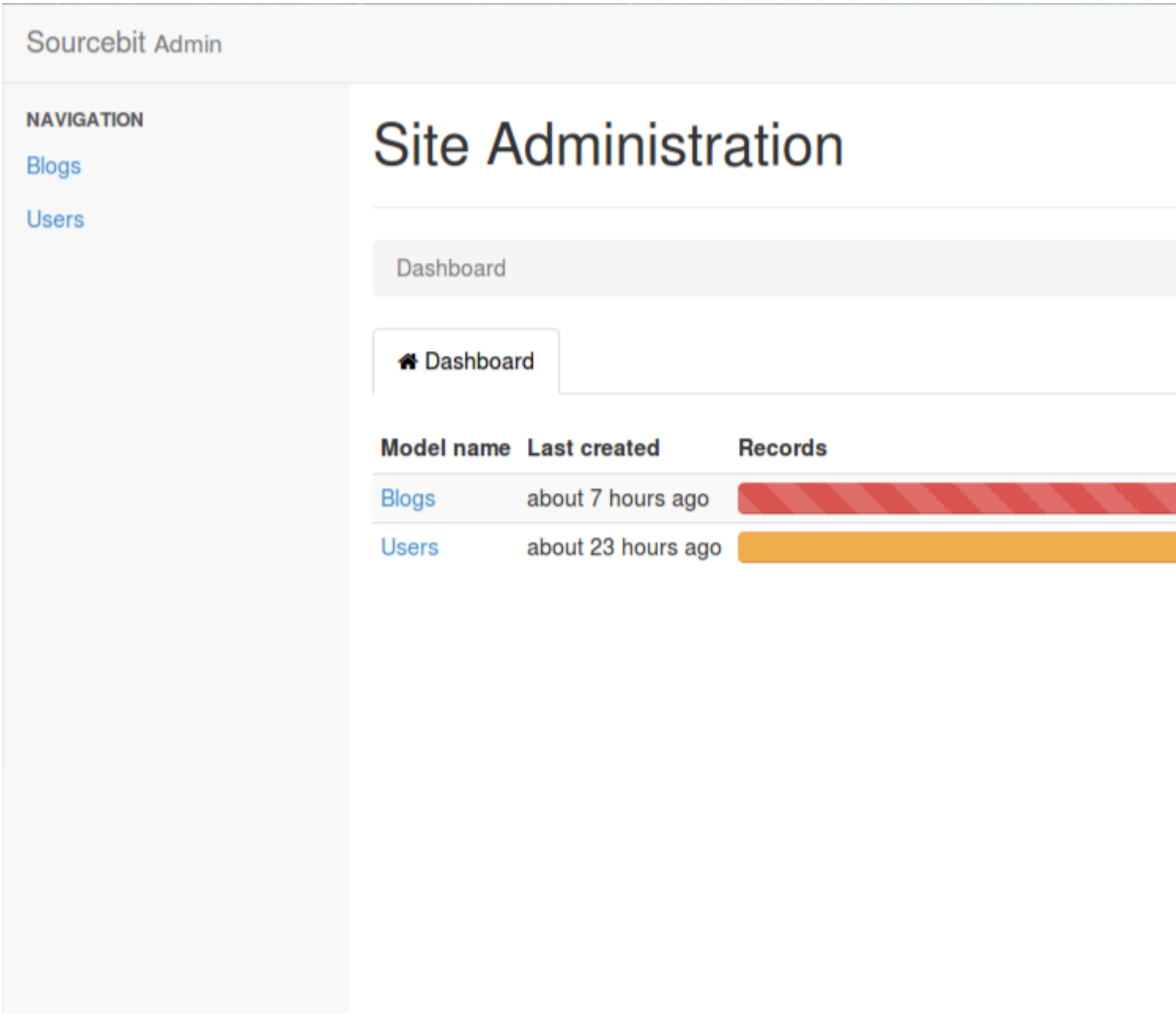
Remarques

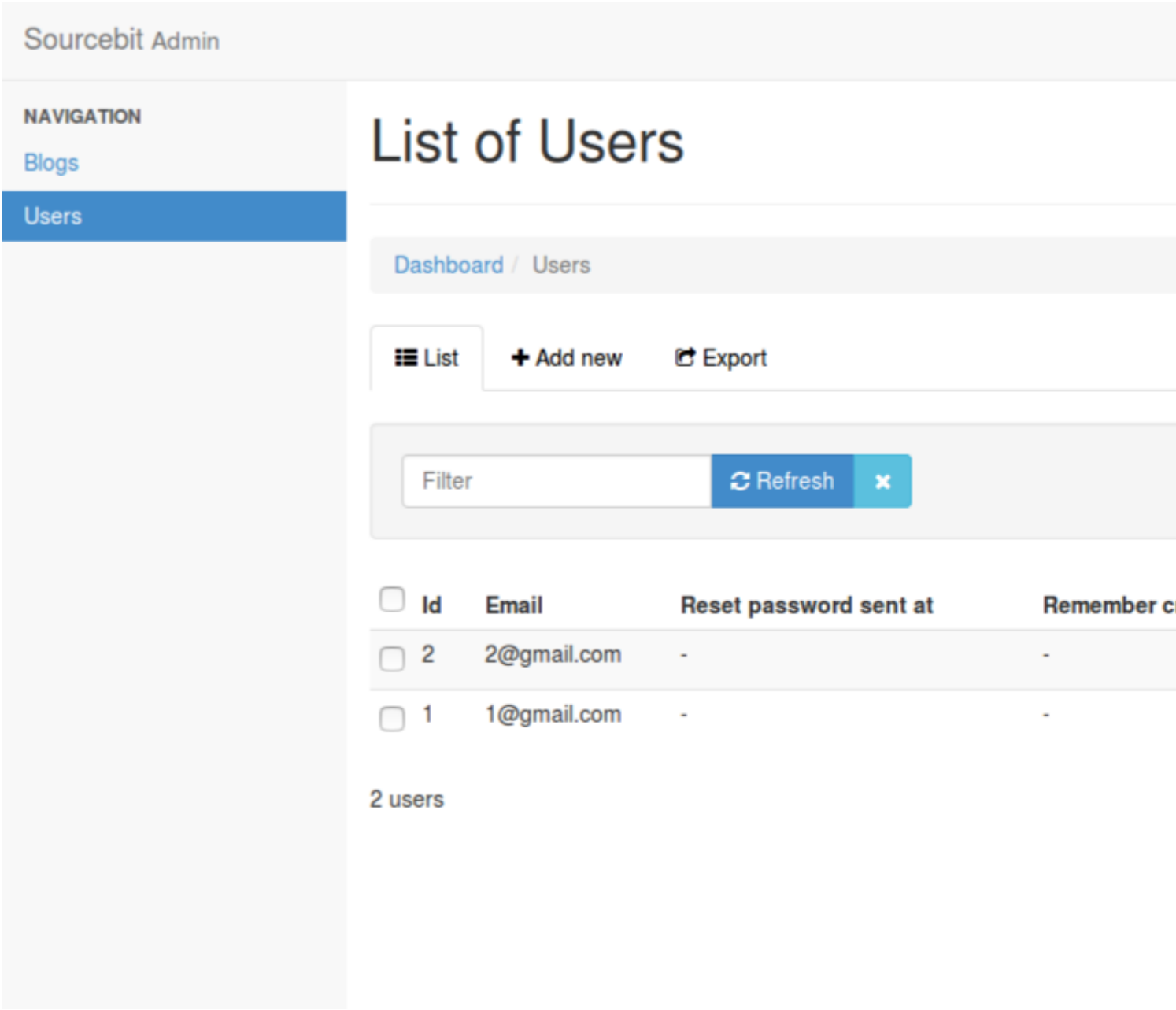
Utilisez-le si vous souhaitez avoir Admin sur votre site Web, sinon cela n'est pas nécessaire. C'est plus simple et puissant qu'active_admin gem. Vous pouvez ajouter ceci à n'importe quel moment après la création des utilisateurs et n'oubliez pas de rendre l'administrateur de chaque utilisateur avant la 4ème étape. Utilisez cancan pour accorder des rôles.

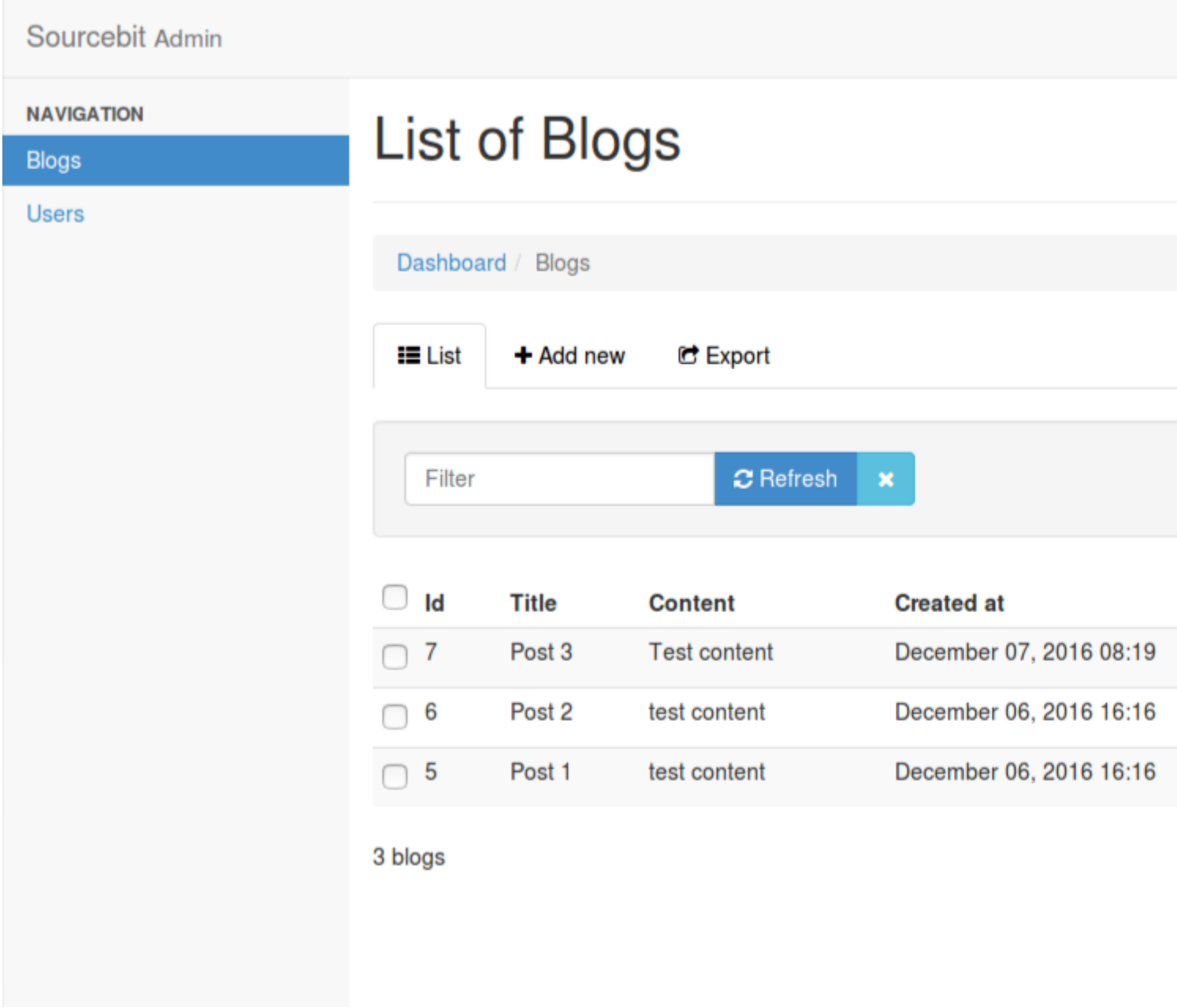
Examples

Donc, voici quelques captures d'écran du panneau d'administration utilisant rails_admin gem.

Comme vous pouvez le voir, la mise en page de ce bijou est très attrayante et conviviale.


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



Lire Ajouter un panneau d'administration en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/8128/ajouter-un-panneau-d-administration

Chapitre 11: API Rails

Examples

Création d'une application API uniquement

Pour créer une application Rails qui sera un serveur API, vous pouvez commencer avec un sous-ensemble de Rails plus limité dans Rails 5.

Pour générer une nouvelle application API Rails:

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

Ce `--api` fait `--api` est de supprimer les fonctionnalités inutiles lors de la création d'une API. Cela inclut les sessions, les cookies, les ressources et tout ce qui fait fonctionner Rails sur un navigateur.

Il configurera également les générateurs de sorte qu'ils ne génèrent pas de vues, d'aides et d'actifs lors de la génération d'une nouvelle ressource.

Lorsque vous comparez `ApplicationController` sur une application Web par rapport à une application API, vous verrez que la version Web s'étend depuis `ActionController::Base` , alors que la version de l'API s'étend depuis `ActionController::API` , qui comprend un sous-ensemble de fonctionnalités beaucoup plus petit.

Lire API Rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/4305/api-rails

Chapitre 12: Associations ActiveRecord

Examples

appartient à

Une association `belongs_to` établit une connexion un-à-un avec un autre modèle, de sorte que chaque instance du modèle déclarant "appartient à" une instance de l'autre modèle.

Par exemple, si votre application inclut des utilisateurs et des publications, et que chaque publication peut être affectée à un seul utilisateur, vous devez déclarer le post-modèle de la manière suivante:

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

Dans votre structure de table, vous pourriez alors avoir

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

en a un

Une association `has_one` établit une connexion un-à-un avec un autre modèle, mais avec une sémantique différente. Cette association indique que chaque instance d'un modèle contient ou possède une instance d'un autre modèle.

Par exemple, si chaque utilisateur de votre application ne possède qu'un seul compte, vous devez déclarer le modèle d'utilisateur comme suit:

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

Dans Active Record, lorsque vous avez une relation `has_one` , l'enregistrement actif garantit que le seul enregistrement existe avec la clé étrangère.

Ici, dans notre exemple: Dans la table des comptes, il ne peut y avoir qu'un seul enregistrement avec un user_id particulier. Si vous essayez d'associer un compte supplémentaire pour le même utilisateur, la clé étrangère de l'entrée précédente devient null (ce qui la rend orpheline) et en crée une nouvelle automatiquement. Cela rend l'entrée précédente nulle même si la sauvegarde échoue pour que la nouvelle entrée conserve sa cohérence.

```
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]
```

a beaucoup

Une association `has_many` indique une connexion un-à-plusieurs avec un autre modèle. Cette association est généralement située de l'autre côté de l'association Appartie à.

Cette association indique que chaque instance du modèle a zéro ou plusieurs instances d'un autre modèle.

Par exemple, dans une application contenant des utilisateurs et des publications, le modèle d'utilisateur peut être déclaré comme suit:

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

La structure de la table de `Post` resterait la même que dans l'application `belongs_to` ; en revanche, l' `User` ne nécessiterait aucune modification de schéma.

Si vous souhaitez obtenir la liste de tous les messages publiés pour l' `User` , vous pouvez ajouter les éléments suivants (vous pouvez ajouter des étendues à vos objets d'association):

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

Association polymorphe

Ce type d'association permet à un modèle ActiveRecord d'appartenir à plus d'un type d'enregistrement de modèle. Exemple commun:

```
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
```

Sans cette association, vous auriez toutes ces clés étrangères dans votre table Address, mais vous ne pourriez jamais avoir une valeur pour l'une d'entre elles car une adresse, dans ce scénario, ne peut appartenir qu'à une seule entité (humaine ou société). Voici à quoi ça ressemblerait:

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

Le has_many: par association

Une association `has_many :through` est souvent utilisée pour établir une connexion `many-to-many` avec un autre modèle. Cette association indique que le modèle déclarant peut correspondre à zéro ou plusieurs instances d'un autre modèle en passant par un troisième modèle.

Par exemple, considérons un cabinet médical où les patients prennent rendez-vous pour consulter un médecin. Les déclarations d'association pertinentes pourraient ressembler à ceci:

```
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
```

Le has_one: par association

Une association `has_one :through` configure `one-to-one` connexion `one-to-one` avec un autre modèle. Cette association indique que le modèle déclarant peut être associé à une instance d'un autre modèle en passant par un troisième modèle.

Par exemple, si chaque `supplier` possède un `account` et que chaque compte est associé à un historique de compte, le modèle de fournisseur peut alors ressembler à ceci:

```
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
```

L'association has_and_belongs_to_many

Une association `has_and_belongs_to_many` crée une connexion directe `many-to-many` avec un autre modèle, sans modèle intermédiaire.

Par exemple, si votre application comprend des `assemblies` et des `parts` , chaque assemblage comportant de nombreuses pièces et chaque pièce apparaissant dans de nombreux assemblages, vous pouvez déclarer les modèles de cette manière:

```
class Assembly < ApplicationRecord
  has_and_belongs_to_many :parts
end

class Part < ApplicationRecord
  has_and_belongs_to_many :assemblies
end
```

Association auto-référentielle

L'association autoréférentielle est utilisée pour associer un modèle à lui-même. L'exemple le plus fréquent serait de gérer l'association entre un ami et son suiveur.

ex.

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

maintenant vous pouvez associer des modèles comme;

```
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
```

et l'autre modèle ressemblera;

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

Lire Associations ActiveRecord en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/1820/associations-activerecord

Chapitre 13: Authentification de Rails 5 API

Examples

Authentification avec Rails authenticate_with_http_token

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

Vous pouvez tester ce noeud final avec `curl` en faisant une requête comme

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

Lire Authentification de Rails 5 API en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/7852/authentification-de-rails-5-api

Chapitre 14: Authentification utilisateur dans Rails

Introduction

Devise est un joyau très puissant, il vous permet de vous inscrire, de vous connecter et de vous déconnecter après l'installation. De plus, l'utilisateur peut ajouter des authentifications et des restrictions à ses applications. Devise est également livré avec ses propres vues, si l'utilisateur veut utiliser. Un utilisateur peut également personnaliser les formulaires d'inscription et de connexion en fonction de ses besoins et de ses exigences. Il convient de noter que Devise vous recommande d'implémenter votre propre identifiant si vous êtes novice dans le domaine des rails.

Remarques

Au moment de générer des configurations à l'aide de `rails generate devise:install` outils `rails generate devise:install` , inventer listera des instructions sur le terminal à suivre.

Si vous avez déjà un modèle `USER` , l'exécution de cette commande de `rails generate devise USER` pour ajouter les colonnes nécessaires à votre modèle `USER` existant.

Utilisez cette méthode d'assistance `before_action :authenticate_user!` en haut de votre contrôleur pour vérifier si l' `user` est connecté ou non. sinon, ils seront redirigés vers la page de connexion.

Examples

Authentification avec Devise

Ajouter une gemme au Gemfile:

```
gem 'devise'
```

Ensuite, exécutez la commande d' `bundle install` l' `bundle install` .

Utilisez la commande `$ rails generate devise:install` pour générer le fichier de configuration requis.

Configurez les options d'URL par défaut du mailer Devise dans chaque environnement Dans l'environnement de développement, ajoutez cette ligne:

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

à votre `config/environments/development.rb`

de même dans la production ceci éditer le fichier `config/environments/production.rb` et ajouter

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

Créez ensuite un modèle en utilisant: `$ rails generate devise USER` Où `USER` est le nom de classe pour lequel vous souhaitez implémenter l'authentification.

Enfin, lancez: `rake db:migrate` et vous êtes tous ensemble.

Affichages personnalisés

Si vous avez besoin de configurer vos vues, vous pouvez utiliser le générateur de générateur de `rails generate devise:views` qui copiera toutes les vues dans votre application. Ensuite, vous pouvez les modifier comme vous le souhaitez.

Si vous avez plusieurs modèles Devise dans votre application (par exemple, User et Admin), vous remarquerez que Devise utilise les mêmes vues pour tous les modèles. Devise offre un moyen simple de personnaliser les vues. Définissez `config.scoped_views = true` dans le `config.scoped_views = true` `config/initializers/devise.rb` .

Vous pouvez également utiliser le générateur pour créer des vues scope: `rails generate devise:views users` de `rails generate devise:views users` les `rails generate devise:views users`

Si vous souhaitez générer quelques ensembles de vues, comme celles du module et confirmable utiliser enregistrable le drapeau -v: `rails generate devise:views -v registrations confirmations`

Devise Controller Filters & Helpers

Pour configurer un contrôleur avec authentification de l'utilisateur à l'aide de la commande, ajoutez ceci before_action: (en supposant que le modèle de votre appareil est "Utilisateur"):

```
before_action :authenticate_user!
```

Pour vérifier si un utilisateur est connecté, utilisez l'aide suivante:

```
user_signed_in?
```

Pour l'utilisateur connecté actuel, utilisez cet assistant:

```
current_user
```

Vous pouvez accéder à la session pour cette portée:

```
user_session
```

- Notez que si votre modèle Devise est appelé `Member` au lieu de `User` , remplacez l' `user` ci-dessus par `member`

Omniauth

Choisissez d'abord votre stratégie d'authentification et ajoutez-la à votre `Gemfile` . Vous pouvez trouver une liste de stratégies ici: https://github.com/intridea/omniauth/wiki/List-of-Strategies

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

Vous pouvez ajouter ceci à votre middleware de rails comme ceci:

```
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
```

Par défaut, OmniAuth ajoutera `/auth/:provider` à vos routes et vous pourrez commencer par utiliser ces chemins.

Par défaut, en cas d'échec, omniauth redirigera vers `/auth/failure`

has_secure_password

Créer un modèle d'utilisateur

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

Ajouter le module has_secure_password au modèle utilisateur

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

Maintenant, vous pouvez créer un nouvel utilisateur avec mot de passe

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

Vérifier le mot de passe avec la méthode d'authentification

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

has_secure_token

Créer un modèle d'utilisateur

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

Maintenant, lorsque vous créez un nouvel utilisateur, un jeton et auth_token sont automatiquement générés

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

Vous pouvez mettre à jour les jetons à l'aide de `regenerate_token` et `regenerate_auth_token`

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

Lire Authentification utilisateur dans Rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/1794/authentification-utilisateur-dans-rails

Chapitre 15: Authentifier Api en utilisant Devise

Introduction

Devise est une solution d'authentification pour Rails. Avant d'aller plus loin, j'aimerais ajouter une note rapide sur l'API. Donc, l'API ne gère pas les sessions (sans état), ce qui signifie une réponse après votre demande, et ne nécessite alors aucune attention supplémentaire, ce qui signifie qu'aucun état antérieur ou futur n'est requis pour que le système fonctionne transmettre les détails d'authentification à toutes les API et indiquer à Devise de ne pas stocker les détails de l'authentification.

Examples

Commencer

Nous allons donc d'abord créer un projet de rails et un dispositif de configuration

créer une application de rails

```
rails new devise_example
```

Maintenant, ajoutez le concept à la liste de gemme

> vous pouvez trouver un fichier nommé 'Gemfile' à la racine du projet rails

Puis lancez l' `bundle install`

Ensuite, vous devez lancer le générateur:

```
rails generate devise:install
```

Maintenant, sur la console, vous pouvez trouver quelques instructions juste suivre.

Générer un modèle de modèle

```
rails generate devise MODEL
```

Puis lancez `rake db:migrate`

Pour plus de détails, allez à: Devise Gem

Jeton d'authentification

Le jeton d'authentification est utilisé pour authentifier un utilisateur avec un jeton unique. Avant de procéder à la logique, nous devons d'abord ajouter le champ `auth_token` à un modèle Devise.

Par conséquent,

```
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
```

Puis lancez `rake db:migrate`

Maintenant, nous sommes tous prêts à faire de l'authentification avec `auth_token`

Dans `app/controllers/application_controllers.rb`

Tout d'abord cette ligne à elle

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

cela aidera l'application rails à répondre avec à la fois HTML et json

alors

```
protect_from_forgery with: :null
```

va changer ceci `:null` car nous ne traitons pas avec les sessions.

maintenant, nous allons ajouter une méthode d'authentification dans application_controller

Ainsi, par défaut, Devise utilise le courrier électronique comme champ unique. Nous pouvons également utiliser des champs personnalisés. Dans ce cas, nous nous authentifierons avec user_email et 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
```

> Note: Le code ci-dessus est purement basé sur votre logique j'essaie juste d'expliquer l'exemple de travail

Sur la ligne 6 du code ci-dessus, vous pouvez voir que j'ai défini `store: false` ce qui empêchera de créer une session sur chaque requête.

Lire Authentifier Api en utilisant Devise en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/9787/authentifier-api-en-utilisant-devise

Chapitre 16: Autorisation avec CanCan

Introduction

CanCan est une stratégie d'autorisation simple pour Rails qui est découplée des rôles d'utilisateur. Toutes les autorisations sont stockées dans un seul emplacement.

Remarques

Avant d'utiliser CanCan, n'oubliez pas de créer des utilisateurs soit par gem, soit manuellement. Pour obtenir un maximum de fonctionnalités de CanCan, créez un utilisateur Admin.

Examples

Commencer avec CanCan

CanCan est une bibliothèque d'autorisation populaire pour Ruby on Rails qui restreint l'accès des utilisateurs à des ressources spécifiques. Le dernier bijou (CanCanCan) est une continuation du projet mort CanCan .

Les autorisations sont définies dans la classe `Ability` et peuvent être utilisées à partir de contrôleurs, de vues, de helpers ou de tout autre endroit du code.

Pour ajouter un support d'autorisation à une application, ajoutez le joyau CanCanCan au `Gemfile` :

```
gem 'cancancan'
```

Ensuite, définissez la classe de capacité:

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

  def initialize(user)
  end
end
```

Ensuite, vérifiez l'autorisation à l'aide de `load_and_authorize_resource` pour charger les modèles autorisés dans le contrôleur:

```
class ArticlesController < ApplicationController
  load_and_authorize_resource

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

`authorize!` vérifier l'autorisation ou lever une exception

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

`can?` vérifier si un objet est autorisé par rapport à une action particulière dans les contrôleurs, les vues ou les assistants

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

Remarque: Cela suppose que l'utilisateur signé est fourni par la méthode `current_user` .

Définir les capacités

Les capacités sont définies dans la classe `Ability` utilisant les méthodes `can` et `cannot` . Considérez l'exemple ci-dessous pour la référence de base:

```
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
```

Gérer un grand nombre de capacités

Une fois que le nombre de définitions de capacités commence à augmenter, il devient de plus en plus difficile de gérer le fichier Ability.

La première stratégie pour gérer ces problèmes consiste à déplacer des capacités dans des méthodes significatives, comme dans cet exemple:

```
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
```

Une fois que cette classe est devenue suffisamment importante, vous pouvez essayer de la diviser en différentes classes pour gérer les différentes responsabilités comme ceci:

```
# 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
```

et définissez ensuite ces classes comme suit:

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

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

et ainsi de suite avec des `Abilities::Authenticated` , `Abilities::Admin` ou tout autre.

Testez rapidement une capacité

Si vous souhaitez tester rapidement si une classe de capacité donne les autorisations correctes, vous pouvez initialiser une capacité dans la console ou dans un autre contexte avec l'environnement des rails chargé, il suffit de passer une instance d'utilisateur à tester:

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

Plus d'informations: https://github.com/ryanb/cancan/wiki/Testing-Abilities

Lire Autorisation avec CanCan en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3021/autorisation-avec-cancan

Chapitre 17: Colonnes multi-usages ActiveRecord

Syntaxe

- `serialize: <field_plural_symbol>`

Examples

Enregistrer un objet

Si vous avez un attribut qui doit être enregistré et récupéré dans la base de données en tant qu'objet, spécifiez le nom de cet attribut à l'aide de la méthode `serialize` et il sera géré automatiquement.

L'attribut doit être déclaré en tant que champ de `text` .

Dans le modèle, vous devez déclarer le type du champ (`Hash` ou `Array`)

Plus d'infos sur: serialize >> apidock.com

Comment

Dans votre 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
```

Dans votre modèle

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

Lire Colonnes multi-usages ActiveRecord en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/7602/colonnes-multi-usages-activerecord

Chapitre 18: Configuration

Examples

Configuration personnalisée

Créez un fichier `YAML` dans le répertoire `config/` , par exemple: `config/neo4j.yml`

Le contenu de `neo4j.yml` peut être quelque chose comme ci-dessous (pour plus de simplicité, la `default` est utilisée pour tous les environnements):

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

development:
  <<: *default

test:
  <<: *default

production:
  <<: *default
```

dans `config/application.rb` :

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

Maintenant, votre configuration personnalisée est accessible comme ci-dessous:

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

Plus d'informations

Le document API officiel Rails décrit la méthode `config_for` comme `config_for` :

> Commodité pour charger config / foo.yml pour les env Rails actuels.

Si vous ne voulez pas utiliser de fichier `yaml`

Vous pouvez configurer votre propre code via l'objet de configuration Rails avec une configuration personnalisée sous la propriété `config.x` .

Exemple

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

Ces points de configuration sont alors disponibles via l'objet de configuration:

```
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
```

Lire Configuration en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2558/configuration

Chapitre 19: Configuration

Examples

Environnements en Rails

Les fichiers de configuration des rails se trouvent dans `config/environments/` . Par défaut, les rails ont 3 environnements, `development` , `production` et `test` . En éditant chaque fichier, vous modifiez uniquement la configuration pour cet environnement.

Rails a également un fichier de `config/application.rb` dans `config/application.rb` . Ceci est un fichier de configuration commun car tous les paramètres définis ici sont écrasés par la configuration spécifiée dans chaque environnement.

Vous ajoutez ou modifiez les options de configuration dans `Rails.application.configure do` options de bloc et de configuration commencent par `config`.

Configuration de la base de données

La configuration de la base de données d'un projet rails se trouve dans un fichier `config/database.yml` . Si vous créez un projet à l'aide de la commande `rails new` et que vous ne spécifiez pas de moteur de base de données à utiliser, alors rails utilise `sqlite` comme base de données par défaut. Un fichier `database.yml` typique avec une configuration par défaut ressemblera à ceci.

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

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

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

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

Si vous souhaitez modifier la base de données par défaut lors de la création d'un nouveau projet, vous pouvez spécifier la base de données: `rails new hello_world --database=mysql`

Configuration générale des rails

Les options de configuration suivantes doivent être appelées sur un objet `Rails::Railtie`

- **config.after_initialize** : Prend un bloc qui sera exécuté après l'initialisation des rails par l'application.
- **config.asset_host** : Ceci définit l'hôte pour les actifs. Ceci est utile lorsque vous utilisez un *réseau de distribution de contenu* . Ceci est un raccourci pour `config.action_controller.asset_host`
- **config.autoload_once_paths** : Cette option accepte un tableau de chemins où Rails charge automatiquement les constantes. La valeur par défaut est un tableau vide
- **config.autoload_paths** : Ceci accepte un tableau de chemins où Rails charge automatiquement les constantes. Il utilise par défaut tous les répertoires sous `app`
- **config.cache_classes** : Détermine si les classes et les modules doivent être rechargés à chaque demande. En mode de développement, la valeur par défaut est `false` et dans les modes de production et de test, elle est définie par défaut sur `true`
- **config.action_view.cache_template_loading** : détermine si les modèles doivent être rechargés à chaque demande. Par défaut, le paramètre `config.cache_classes`
- **config.beginning_of_week** : Ceci définit le début de semaine par défaut. Il nécessite un symbole valide du jour de la semaine (`:monday`)
- **config.cache_store** : choisissez le magasin de cache à utiliser. Les options incluent `:file_store` `:memory_store` , `mem_cache_store` ou `null_store` .
- **config.colorize_logging** : contrôle si les informations de journalisation sont colorisées
- **config.eager_load** : tous les utilisateurs sont chargés
- **config.encoding** : Spécifie le codage de l'application. La valeur par défaut est `UTF-8`
- **config.log_level** : définit la verbosité du logger Rails. La valeur par défaut est `:debug` dans tous les environnements.
- **config.middleware** : Utilisez ceci pour configurer le middleware de l'application
- **config.time_zone** : définit le fuseau horaire par défaut de l'application.

Configuration des actifs

Les options de configuration suivantes peuvent être utilisées pour configurer les actifs

- **config.assets.enabled** : détermine si le pipeline de ressources est activé. Cela par défaut à true
- **config.assets.raise_runtime_errors** : Ceci active la vérification des erreurs à l'exécution. C'est utile pour le `development mode`
- **config.assets.compress** : Permet de compresser les actifs. En mode production, la valeur par défaut est true
- **config.assets.js_compressor** : Spécifie le compresseur JS à utiliser. Les options incluent `:closure` `:uglifier` et `:yui`
- **config.assets.paths** : Spécifie les chemins à rechercher pour les actifs.
- **config.assets.precompile** : permet de choisir des ressources supplémentaires à

précompiler lorsque les `rake assets:precompile` est exécutée
- **config.assets.digest** : cette option permet d'utiliser les empreintes digitales `MD-5` dans les noms d'actifs. La valeur par défaut est true en mode de développement
- **config.assets.compile** : Active / **désactive la** compilation des `Sprockets` live en mode production

Configuration des générateurs

Rails vous permet de configurer les générateurs utilisés lors de l'exécution de commandes avec des `rails generate` . Cette méthode, `config.generators` prend un bloc

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

Voici quelques options

Option	La description	Défaut
les atouts	Crée des ressources lors de la génération d'un échafaudage	vrai
force_plural	Permet des noms de modèle pluralisés	faux
assistant	Détermine s'il faut générer des helpers	vrai
outil d'intégration	Spécifier l'outil d'intégration	`test_unit`
javascript_engine	Configure le moteur JS	`:js`
resource_route	Génère un itinéraire de ressource	vrai
stylesheet_engine	Configure le moteur de feuille de style	`:cs`
scaffold_stylesheet	Crée des CSS sur des échafaudages	vrai
test_framework	Spécifier le cadre de test	`Minitest`
template_engine	Configure le moteur de template	`:erb`

Lire Configuration en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2841/configuration

Chapitre 20: Configurer Angular avec Rails

Examples

Angulaire Avec Rails 101

Étape 1: Créer une nouvelle application Rails

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

Étape 2: Supprimer les Turbolinks

La suppression de turbolinks nécessite de la retirer du fichier Gemfile.

```
gem 'turbolinks'
```

Supprimez le `require` de `app/assets/javascripts/application.js` :

```
//= require turbolinks
```

Étape 3: Ajouter AngularJS au pipeline d'actifs

Pour que Angular fonctionne avec le pipeline de ressources Rails, nous devons ajouter au fichier Gemfile:

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

Maintenant, lancez la commande

```
bundle install
```

Ajoutez `bower` pour pouvoir installer la dépendance AngularJS:

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

Ajoutez Angular à `bower.json` :

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

  "dependencies": {

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

Maintenant que `bower.json` est configuré avec les bonnes dépendances, installons-les:

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

Étape 4: Organiser l'application Angular

Créez la structure de dossiers suivante dans `app/assets/javascript/angular-app/` :

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

Dans `app/assets/javascripts/application.js` , ajoutez `require` pour Angular, l'assistant de template et la structure de fichier de l'application Angular. Comme ça:

```
//= 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
```

Étape 5: Amorcez l'application Angular

Créez `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'
)
```

Créez un module angulaire à l' `app/assets/javascripts/angular-app/modules/example.js.coffee.erb` :

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

Créez un contrôleur angulaire pour cette application sur `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"  ])
```

Ajoutez maintenant une route aux Rails pour passer le contrôle à Angular. Dans `config/routes.rb` :

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

Générez le contrôleur Rails pour répondre à cette route:

```
rails g controller Example
```

Dans `app/controllers/example_controller.rb` :

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

Dans la vue, nous devons spécifier quelle application angulaire et quel contrôleur angulaire pilotera cette page. Donc, dans `app/views/example/index.html.erb` :

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

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

</div>
```

Pour afficher l'application, démarrez votre serveur Rails et visitez http: // localhost: 3000 / example .

Lire Configurer Angular avec Rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3902/configurer-angular-avec-rails

Chapitre 21: Conventions de nommage

Examples

Contrôleurs

Les noms de classe de contrôleur sont pluralisés. La raison en est que le contrôleur contrôle plusieurs instances d'instance d'objet.

Par exemple : `OrdersController` serait le contrôleur pour une table de `orders` . Rails recherchera alors la définition de la classe dans un fichier appelé `orders_controller.rb` dans le `orders_controller.rb /app/controllers` .

Par exemple : `PostsController` serait le contrôleur pour une table de `posts` .

Si le nom de la classe du contrôleur comporte plusieurs mots en majuscules, le nom de la table est supposé avoir des traits de soulignement entre ces mots.

Par exemple: Si un contrôleur s'appelle `PendingOrdersController` le nom de fichier supposé pour ce contrôleur sera `pending_orders_controller.rb` .

Des modèles

Le modèle est nommé en utilisant la convention de dénomination de classe de MixedCase ininterrompue et est toujours le singulier du nom de la table.

Par exemple : Si une table était nommée `orders` , le modèle associé serait nommé `Order`

Par exemple : si une table était nommée `posts` , le modèle associé serait nommé `Post`

Rails recherchera alors la définition de la classe dans un fichier appelé `order.rb` dans le `order.rb /app/models` .

Si le nom de la classe de modèle comporte plusieurs mots en majuscules, le nom de la table est supposé avoir des traits de soulignement entre ces mots.

Par exemple: si un modèle s'appelle `BlogPost` le nom de table supposé sera `blog_posts` .

Vues et mises en page

Lorsqu'une action de contrôleur est rendue, Rails tente de trouver une disposition et une vue correspondantes en fonction du nom du contrôleur.

Les vues et les dispositions sont placées dans le répertoire `app/views` .

Étant donné une requête à l'action d' `PeopleController#index` , Rails recherchera:

- la mise en page a appelé les `people` dans `app/views/layouts/` (ou `application` si aucune

correspondance n'est trouvée)
- une vue appelée `index.html.erb` dans `app/views/people/` par défaut
- si vous souhaitez rendre un autre fichier appelé `index_new.html.erb` vous devez écrire le code correspondant dans l'action d' `PeopleController#index` comme `render 'index_new'`
- nous pouvons définir différentes `layouts` pour chaque `action` en écrivant `render 'index_new', layout: 'your_layout_name'`

Noms de fichiers et chargement automatique

Les fichiers Rails - et les fichiers Ruby en général - doivent être nommés avec les `lower_snake_case` fichiers `lower_snake_case` . Par exemple

```
app/controllers/application_controller.rb
```

est le fichier qui contient la définition de la classe `ApplicationController` . Notez que même si `PascalCase` est utilisé pour les noms de classe et de module, les fichiers dans lesquels ils résident doivent toujours être `lower_snake_case` .

Le nommage cohérent est important car Rails utilise les fichiers à chargement automatique en fonction des besoins et utilise "flexion" pour transformer différents styles de dénomination, tels que la transformation de `application_controller` en `ApplicationController` et inversement.

Par exemple, si Rails voit que la classe `BlogPost` n'existe pas (elle n'a pas encore été chargée), elle recherchera un fichier nommé `blog_post.rb` et tentera de charger ce fichier.

Il est donc également important de nommer les fichiers pour ce qu'ils contiennent, car l'autochargeur s'attend à ce que les noms de fichiers correspondent au contenu. Si, par exemple, `blog_post.rb` contient une classe nommée juste `Post` , vous verrez un `LoadError` : `Expected [some path]/blog_post.rb to define BlogPost` .

Si vous ajoutez un répertoire sous `app/something/` (par exemple / models / products /), et

- vouloir des modules d'espace de nommage et des classes dans le nouveau répertoire alors vous n'avez rien à faire et il sera chargé lui-même. Par exemple, dans `app/models/products/` `you would need to wrap your class in` module Products`.
- Je ne veux pas que les modules et les classes d'espace de noms se trouvent dans mon nouveau `config.autoload_paths += %W( #{config.root}/app/models/products )` vous devez ensuite ajouter `config.autoload_paths += %W( #{config.root}/app/models/products )` à votre `application.rb` pour le charger automatiquement.

Une autre chose à laquelle vous devez faire attention (en particulier si l'anglais n'est pas votre langue maternelle) est le fait que Rails représente des noms pluriels irréguliers en anglais. Donc, si vous avez un modèle nommé "Foot", le contrôleur correspondant doit être appelé "FeetController" plutôt que "FootsController" si vous voulez que les rails "magiques" routage (et beaucoup d'autres fonctionnalités de ce type) fonctionnent.

Classe de modèles à partir du nom du contrôleur

Vous pouvez obtenir une classe Model à partir d'un nom de contrôleur de cette manière (le contexte est la classe Controller):

```
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
```

Lire Conventions de nommage en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/1493/conventions-de-nommage

Chapitre 22: Crevettes PDF

Examples

Exemple avancé

C'est l'approche avancée avec exemple

```
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
```

J'ai le code ci-dessus, nous avons cette ligne `FundsPdf.new(@fund, view_context)` . Ici, nous initialisons la classe FundsPdf avec l'instance @fund et view_context pour utiliser des méthodes d'assistance dans FundsPdf. FundsPdf ressemblerait à ceci

```
 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
```

C'est l'une des meilleures approches pour générer des fichiers PDF avec des classes utilisant la gemme Prawn.

Exemple de base

Vous devez ajouter Gem et PDF MIME: saisissez mime_types.rb car nous devons notifier les rails à propos du type MIME PDF.

Après cela, nous pouvons générer du Pdf avec Prawn de la manière suivante:

Ceci est l'assignation de base

```
pdf = Prawn::Document.new
pdf.text "Hello World"
pdf.render_file "assignment.pdf"
```

Nous pouvons le faire avec un bloc implicite

```
Prawn::Document.generate("implicit.pdf") do
 text "Hello World"
end
```

Avec bloc explicite

```
Prawn::Document.generate("explicit.pdf") do |pdf|
 pdf.text "Hello World"
end
```

Lire Crevettes PDF en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/4163/crevettes-pdf

Chapitre 23: Déploiement d'une application Rails sur Heroku

Examples

Déploiement de votre application

Assurez-vous d'être dans le répertoire contenant votre application Rails, puis créez une application sur Heroku.

```
$ heroku create example
Creating □ example... done
https://example.herokuapp.com/ | https://git.heroku.com/example.git
```

La première URL de la sortie, http://example.herokuapp.com , correspond à l'emplacement où l'application est disponible. La deuxième URL, git@heroku.com: example.git, est l'URL du référentiel git distant.

Cette commande ne doit être utilisée que sur un dépôt git initialisé. La commande heroku create ajoute automatiquement une télécommande git nommée «heroku» pointant vers cette URL.

L'argument du nom de l'application («exemple») est facultatif. Si aucun nom d'application n'est spécifié, un nom aléatoire sera généré. Étant donné que les noms d'applications Heroku se trouvent dans un espace de noms global, vous pouvez vous attendre à ce que des noms communs, tels que «blog» ou «wiki», soient déjà utilisés. Il est souvent plus facile de commencer avec un nom par défaut et de renommer l'application ultérieurement.

Ensuite, déployez votre code:

```
$ 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
```

Si vous utilisez la base de données dans votre application, vous devez migrer manuellement la base de données en exécutant:

```
$ heroku run rake db:migrate
```

Toutes les commandes après l' `heroku run` seront exécutées sur un dyno Heroku. Vous pouvez obtenir une session shell interactive en exécutant:

```
$ heroku run bash
```

Assurez-vous d'avoir un dyno exécutant le type de processus Web:

```
$ heroku ps:scale web=1
```

La commande heroku ps répertorie les dynos en cours d'exécution de votre application:

```
$ 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)
```

Vous pouvez maintenant visiter l'application dans notre navigateur avec `heroku open` .

```
$ heroku open
```

Heroku vous donne une URL Web par défaut dans le domaine `herokuapp.com` . Lorsque vous êtes prêt à évoluer pour la production, vous pouvez ajouter votre propre domaine personnalisé.

Gestion des environnements de production et de mise en scène pour un Heroku

Chaque application Heroku s'exécute dans au moins deux environnements: sur Heroku (nous appellerons cette production) et sur votre machine locale (développement). Si plusieurs personnes travaillent sur l'application, vous disposez de plusieurs environnements de développement, généralement un par ordinateur. Habituellement, chaque développeur aura également un environnement de test pour exécuter des tests. Malheureusement, cette approche échoue à mesure que les environnements deviennent moins similaires. Windows et Mac, par exemple, offrent des environnements différents de ceux de la pile Linux sur Heroku. Vous ne pouvez donc pas toujours être sûr que le code qui fonctionne dans votre environnement de développement local fonctionnera de la même façon lorsque vous le déploierez en production.

La solution consiste à disposer d'un environnement de stockage aussi proche que possible de la production. Cela peut être réalisé en créant une seconde application Heroku qui héberge votre application de transfert. Avec le staging, vous pouvez vérifier votre code dans un paramètre similaire à celui de la production avant qu'il affecte vos utilisateurs réels.

À partir de zéro

Supposons qu'une application s'exécute sur votre machine locale et que vous êtes prêt à la transmettre à Heroku. Nous aurons besoin de créer à la fois des environnements distants, une mise en scène et une production. Pour prendre l'habitude de pousser d'abord à la mise en scène, nous allons commencer par ceci:

```
$ 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
```

Par défaut, la CLI de heroku crée des projets avec une télécommande heroku git. Ici, nous spécifions un nom différent avec l'indicateur --remote, donc pousser le code vers Heroku et exécuter des commandes sur l'application est un peu différent du maître git push heroku:

```
 $ git push staging master
...
 $ heroku ps --remote staging
=== web: `bundle exec puma -C config/puma.rb``
web.1: up for 21s
```

Une fois que votre application de transfert est opérationnelle, vous pouvez créer votre application de production:

```
$ 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
```

Et avec cela, vous disposez du même code source que deux applications Heroku distinctes: une mise en scène et une production, configurées de manière identique. Rappelez-vous juste que vous devrez spécifier quelle application vous allez utiliser dans votre travail quotidien. Vous pouvez soit utiliser l'option "--remote", soit utiliser votre configuration git pour spécifier une application par défaut.

Lire Déploiement d'une application Rails sur Heroku en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/4485/deploiement-d-une-application-rails-sur-heroku

Chapitre 24: Des vues

Examples

Partiels

Les modèles partiels (partiels) permettent de décomposer le processus de rendu en morceaux plus faciles à gérer. Les Partials vous permettent d'extraire des morceaux de code de vos modèles pour séparer les fichiers et les réutiliser dans tous vos modèles.

Pour *créer* un partiel, créez un nouveau fichier qui commence par un trait de soulignement: `_form.html.erb`

Pour *rendre* un partiel dans une vue, utilisez la méthode de rendu dans la vue: `<%= render "form" %>`

- Notez que le trait de soulignement est omis lors du rendu
- Un partiel doit être rendu en utilisant son chemin s'il est situé dans un autre dossier

Pour *passer* une variable dans le partiel en tant que variable locale, utilisez cette notation:

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

Les partiels sont également utiles lorsque vous devez *réutiliser* exactement le même code (**philosophie DRY**).

Par exemple, pour réutiliser le code `<head>` , créez un partiel nommé `_html_header.html.erb` , entrez votre code `<head>` à réutiliser et restituez le partiel lorsque cela est nécessaire: `<%= render 'html_header' %>` .

Object Partials

Les objets qui répondent à `to_partial_path` peuvent également être rendus, comme dans `<%= render @post %>` . Par défaut, pour les modèles ActiveRecord, ce sera quelque chose comme `posts/post` , donc en `@post` réellement `@post` , le fichier `views/posts/_post.html.erb` sera rendu.

Un `post` local nommé sera automatiquement attribué. Au final, `<%= render @post %>` est un raccourci pour `<%= render 'posts/post', post: @post %>` .

Des collections d'objets répondant à `to_partial_path` peuvent également être fournies, telles que `<%= render @posts %>` . Chaque article sera rendu consécutivement.

Partials globaux

Pour créer un partiel global pouvant être utilisé n'importe où sans faire référence à son chemin

exact, le partiel doit être situé dans le chemin des `views/application` . L'exemple précédent a été modifié ci-dessous pour illustrer cette fonctionnalité.

Par exemple, il s'agit d'un chemin d'accès à une `app/views/application/_html_header.html.erb:` partielle globale `app/views/application/_html_header.html.erb:`

Pour rendre ce partiel global n'importe où, utilisez `<%= render 'html_header' %>`

AssetTagHelper

Laisser les rails relier automatiquement et correctement les ressources (css / js / images) dans la plupart des cas, vous souhaitez utiliser des aides intégrées. (Documentation officielle)

Aides à l'image

chemin_image

Cela renvoie le chemin d'accès à un élément d'image dans `app/assets/images` .

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

URL de l'image

Cela renvoie l'URL complète à un élément d'image dans `app/assets/images` .

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

image_tag

Utilisez cette aide si vous souhaitez inclure une étiquette `<img src="" />` avec l'ensemble source.

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

Assistants JavaScript

javascript_include_tag

Si vous souhaitez inclure un fichier JavaScript dans votre vue.

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

javascript_path

Cela retourne le chemin de votre fichier JavaScript.

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

javascript_url

Cela renvoie l'URL complète de votre fichier JavaScript.

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

Aides à la feuille de style

stylesheet_link_tag

Si vous souhaitez inclure un fichier CSS dans votre vue.

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

stylesheet_path

Cela renvoie le chemin de votre actif de feuille de style.

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

stylesheet_url

Cela renvoie l'URL complète de votre ressource de feuille de style.

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

Exemple d'utilisation

Lors de la création d'une nouvelle application rails, vous aurez automatiquement deux de ces assistants dans `app/views/layouts/application.html.erb`

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

Cela produit:

```
// 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>
```

Structure

Comme Rails suit le pattern M **V** C Les `Views` sont là où vos "templates" sont pour vos actions.

Disons que vous avez un contrôleur `articles_controller.rb` . Pour ce contrôleur, vous auriez un dossier dans les vues appelé `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
    |
    '-- [...]
```

Cette structure vous permet d'avoir un dossier pour chaque contrôleur. Lorsque vous appelez une action dans votre contrôleur, la vue appropriée sera automatiquement affichée.

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

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

Remplacer le code HTML dans les vues

Si vous avez toujours voulu déterminer le contenu HTML à imprimer sur une page au moment de l'exécution, alors rails a une très bonne solution pour cela. Il a quelque chose appelé **content_for** qui nous permet de passer un bloc à une vue de rails. S'il vous plaît vérifier l'exemple ci-dessous,

Déclarer contenu_pour

```
<div>
```

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

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

HAML - une autre façon d'utiliser dans vos vues

HAML (langage de balisage d'abstraction HTML) est une manière élégante et belle de décrire et de concevoir le code HTML de vos vues. Au lieu d'ouvrir et de fermer les balises, HAML utilise l'indentation pour la structure de vos pages. Fondamentalement, si quelque chose doit être placé dans un autre élément, il vous suffit de le mettre en retrait en utilisant une tabulation. Les tabulations et les espaces blancs sont importants dans HAML, assurez-vous donc d'utiliser toujours le même nombre d'onglets.

Exemples:

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

Et dans HAML:

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

Vous voyez, la structure de la mise en page est beaucoup plus claire qu'avec HTML et ERB.

Installation

Il suffit d'installer le bijou en utilisant

```
gem install haml
```

et ajouter la gemme au Gemfile

```
gem "haml"
```

Pour utiliser HAML au lieu de HTML / ERB, remplacez simplement les extensions de fichier de `something.html.erb` par `something.html.haml` chose.html.haml.

Conseils rapides

Les éléments communs comme les divs peuvent être écrits de manière courte

HTML

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

HAML

```
%div.myclass
```

HAML, sténographie

```
.myclass
```

Les attributs

HTML

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

HAML

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

Insérer le code ruby

Vous pouvez insérer un code Ruby avec les signes = et -.

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

Le code commençant par **=** sera exécuté et incorporé dans le document.

Le code commençant par **-** sera exécuté, mais pas inséré dans le document.

Documentation complète

HAML est très facile au début, mais est également très complexe, donc je vous recommande de lire la documentation .

Lire Des vues en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/850/des-vues

Chapitre 25: Elasticsearch

Examples

Installation et test

La première chose que vous voulez faire pour le développement local est d'installer ElasticSearch sur votre ordinateur et de le tester pour voir s'il fonctionne. Il faut que Java soit installé.
L'installation est assez simple:

- Mac OS X: `brew install elasticsearch`
- Ubuntu: `sudo apt-get install elasticsearch`

Alors commencez:

- Mac OS X: les `brew services start elasticsearch`
- Ubuntu: `sudo service elasticsearch start`

Pour le tester, le plus simple est d' `curl` . Cela peut prendre quelques secondes, alors ne paniquez pas si vous n'obtenez aucune réponse au début.

```
curl localhost:9200
```

Exemple de réponse:

```
{
  "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"
}
```

Mise en place d'outils pour le développement

Lorsque vous commencez à utiliser ElasticSearch (ES), il peut être utile de disposer d'un outil graphique qui vous aide à explorer vos données. Un plugin appelé `elasticsearch-head` ne fait que cela. Pour l'installer, procédez comme suit:

- Découvrez dans quel dossier ES est installé: `ls -l $(which elasticsearch)`
- `cd` dans ce dossier et lance le programme d'installation du plugin: `elasticsearch/bin/plugin -install mobz/elasticsearch-head`
- Ouvrez `http://localhost:9200/_plugin/head/` dans votre navigateur

Si tout fonctionnait comme prévu, vous devriez voir une interface graphique agréable sur laquelle

vous pouvez explorer vos données.

introduction

ElasticSearch possède une API JSON bien documentée, mais vous voudrez probablement utiliser certaines bibliothèques qui gèrent cela pour vous:

- `Elasticsearch` - le wrapper de bas niveau officiel pour l'API HTTP
- `Elasticsearch-rails` - l'intégration Rails officielle de haut niveau qui vous permet de connecter vos modèles Rails avec ElasticSearch à l'aide du modèle ActiveRecord ou Repository
- `Chewy` - Une intégration alternative non officielle de haut niveau de Rails, qui est très populaire et possède sans doute une meilleure documentation

Utilisons la première option pour tester la connexion:

```
gem install elasticsearch
```

Ensuite, lancez le terminal Ruby et essayez-le:

```
require 'elasticsearch'

client = Elasticsearch::Client.new log: true
# by default it connects to http://localhost:9200

client.transport.reload_connections!
client.cluster.health

client.search q: 'test'
```

Searchkick

Si vous souhaitez configurer rapidement elasticsearch, vous pouvez utiliser le joyau de searchkick:

```
gem 'searchkick'
```

Ajoutez le moteur de recherche aux modèles que vous souhaitez rechercher.

```
class Product < ActiveRecord::Base
  searchkick
end
```

Ajouter des données à l'index de recherche.

```
Product.reindex
```

Et pour interroger, utilisez:

```
products = Product.search "apples"
products.each do |product|
  puts product.name
end
```

Assez vite, les connaissances élastiques de recherche ne sont pas nécessaires ;-)

Plus d'informations ici: https://github.com/ankane/searchkick

Lire Elasticsearch en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/6500/elasticsearch

Chapitre 26: Emplois actifs

Examples

introduction

Disponible depuis Rails 4.2, Active Job est une structure permettant de déclarer des travaux et de les exécuter sur divers serveurs principaux en attente. Les tâches récurrentes ou ponctuelles qui ne bloquent pas et peuvent être exécutées en parallèle sont de bons cas d'utilisation pour les tâches actives.

Exemple de travail

```
class UserUnsubscribeJob < ApplicationJob
  queue_as :default

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

Créer un job actif via le générateur

```
$ rails g job user_unsubscribe
```

Lire Emplois actifs en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/8033/emplois-actifs

Chapitre 27: Enregistreur de rails

Examples

Rails.logger

Utilisez toujours `Rails.logger.{debug|info|warn|error|fatal}` plutôt que `puts` . Cela permet à vos journaux de tenir dans le format de journal standard, d'avoir un horodatage et d'avoir un niveau afin que vous choisissiez s'ils sont suffisamment importants pour être affichés dans un environnement spécifique. Vous pouvez voir les fichiers journaux séparés pour votre application sous `log/` directory avec le nom de votre environnement d'application rails. comme: `development.log` ou `production.log` ou `staging.log`

Vous pouvez facilement faire pivoter les journaux de production des rails avec LogRotate. Il vous suffit de faire une petite configuration comme ci-dessous

Ouvrez `/etc/logrotate.conf` avec votre éditeur linux préféré `vim` ou `nano` et ajoutez le code ci-dessous dans ce fichier en bas.

```
/YOUR/RAILSAPP/PATH/log/*.log {
  daily
  missingok
  rotate 7
  compress
  delaycompress
  notifempty
  copytruncate
}
```

Alors, **comment ça marche?** C'est incroyablement facile. Chaque bit de la configuration effectue les opérations suivantes:

- **daily** - Faites pivoter les fichiers journaux chaque jour. Vous pouvez également utiliser chaque semaine ou chaque mois ici.
- **missingok** - Si le fichier journal n'existe pas, ignorez-le
- **tourner 7** - Ne conserve que 7 jours de journaux
- **compresser** - GZip le fichier journal en rotation
- **delaycompress** - **Faites** pivoter le fichier un jour, puis compressez-le le lendemain pour être sûr qu'il n'interfère pas avec le serveur Rails
- **notifempty** - Ne faites pas pivoter le fichier si les journaux sont vides
- **copytruncate** - Copiez le fichier journal, puis **videz** -le. Cela permet de s’assurer que le fichier journal Rails est en cours d’écriture afin que vous ne rencontriez pas de problèmes car le fichier ne change pas réellement. Si vous ne l'utilisez pas, vous devrez redémarrer votre application Rails à chaque fois.

Exécution de Logrotate Puisque nous venons d'écrire cette configuration, vous voulez la tester.

Pour lancer logrotate manuellement, faites simplement: `sudo /usr/sbin/logrotate -f`

```
/etc/logrotate.conf
```

C'est tout.

Lire Enregistreur de rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3904/enregistreur-de-rails

Chapitre 28: Fonctionnalité de paiement dans les rails

Introduction

Ce document prétend vous présenter, avec un exemple complet, comment vous pouvez implémenter différentes méthodes de paiement avec Ruby on Rails.

Dans l'exemple, nous couvrirons Stripe et Braintree, deux plateformes de paiement très connues.

Remarques

Documentation.

Bande

Braintree

Examples

Comment intégrer avec Stripe

Ajoutez la gemme Stripe à notre `Gemfile`

```
gem 'stripe'
```

Ajoutez les fichiers `initializers/stripe.rb` . Ce fichier contient les clés nécessaires pour vous connecter à votre compte de bande.

```
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]
```

Comment créer un nouveau client à Stripe

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

Ce code crée un nouveau client sur Stripe avec une adresse e-mail et une source données.

`payment_token` est le jeton donné du côté du client qui contient un mode de paiement tel qu'une carte de crédit ou un compte bancaire. Plus d'infos: Stripe.js côté client

Comment récupérer un plan de Stripe

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

Ce code récupère un plan de Stripe par son identifiant.

Comment créer un abonnement

Lorsque nous avons un client et un plan, nous pouvons créer un nouvel abonnement sur Stripe.

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

Il créera un nouvel abonnement et facturera notre utilisateur. Il est important de savoir ce qui se passe réellement sur Stripe lorsque nous inscrivons un utilisateur à un plan, vous trouverez plus d'informations ici: Cycle de vie de l'abonnement Stripe .

Comment facturer un utilisateur avec un paiement unique

Parfois, nous voulons facturer à nos utilisateurs une seule fois, car nous ferons le prochain.

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

Dans ce cas, nous facturons notre utilisateur une fois pour un montant donné.

Erreurs courantes:

- Le montant doit être envoyé sous forme d'entier, ce qui signifie que 2000 sera 20 unités de devise. Vérifiez cet exemple
- Vous ne pouvez pas facturer un utilisateur dans deux devises. Si l'utilisateur a été facturé en EUR à tout moment dans le passé, vous ne pouvez pas facturer l'utilisateur en USD.
- Vous ne pouvez pas facturer l'utilisateur sans source (méthode de paiement).

Lire Fonctionnalité de paiement dans les rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/10929/fonctionnalite-de-paiement-dans-les-rails

Chapitre 29: Form Helpers

Introduction

Rails fournit des aides à la vue pour générer un balisage de formulaire.

Remarques

- Les types de saisie de `date` , y compris `date` , `datetime` , `datetime-local` , `time` , `month` et `week` ne fonctionnent pas dans FireFox.
- `input<type="telephone">` ne fonctionne qu'avec Safari 8.
- `input<type="email">` ne fonctionne pas sur Safari

Examples

Créer un formulaire

Vous pouvez créer un formulaire en utilisant l'aide `form_tag`

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

Cela crée le code HTML suivant

```
<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>
```

Cette balise de formulaire a créé un champ de saisie `hidden` . Cela est nécessaire, car les formulaires ne peuvent pas être soumis avec succès sans cela.

Le second champ de saisie, nommé `authenticity_token` ajoute une protection contre la `cross-site request forgery` .

Créer un formulaire de recherche

Pour créer un formulaire de recherche, entrez le code suivant

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

- `form_tag` : Ceci est l'aide par défaut pour créer un formulaire. C'est le premier paramètre, `/search` est l'action et le second paramètre spécifie la méthode HTTP. Pour les formulaires de recherche, il est important de toujours utiliser la méthode `get`
- `label_tag` : Cet assistant crée une `<label>` html `<label>` .
- `text_field_tag` : Cela va créer un élément d'entrée avec du `text` type
- `submit_tag` : crée un élément d'entrée avec le type `submit`

Helpers pour les éléments de formulaire

Cases à cocher

```
<%= 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") %>
```

Cela va générer le HTML suivant

```
<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>
```

Boutons Radio

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

Cela génère le code HTML suivant

```
<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>
```

Zone de texte

Pour créer une zone de texte plus grande, il est recommandé d'utiliser `text_area_tag`

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

Cela créera le code HTML suivant

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

Numéro de champ

Cela va créer un élément d' `input<type="number">`

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

Pour spécifier une plage de valeurs, nous pouvons utiliser l'option `in:`

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

Champ de mot de passe

Parfois, vous voulez que les caractères saisis par l'utilisateur soient masqués. Cela va générer un `<input type="password">`

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

Champ Email

Cela va créer un `<input type="email">`

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

Champ téléphonique

Cela créera un `<input type="tel">` .

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

Assistants de date

- `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) %>
```

Menu déroulant

Exemple standard: @models = Model.all select_tag "models", options_from_collection_for_select (@models, "id", "name"), {}

Cela générera le code HTML suivant: David

Le dernier argument est des options, qui acceptent les éléments suivants: {multiple: false, disabled: false, include_blank: false, prompt: false}

Plus d'exemples peuvent être trouvés:
http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select_tag

Lire Form Helpers en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/4509/form-helpers

Chapitre 30: Formulaire imbriqué en Ruby on Rails

Examples

Comment configurer un formulaire imbriqué dans Ruby on Rails

Le premier à avoir: un modèle contenant une relation `has_many` avec un autre modèle.

```
class Project < ApplicationRecord
  has_many :todos
end

class Todo < ApplicationRecord
  belongs_to :project
end
```

Dans `ProjectsController` :

```
class ProjectsController < ApplicationController
  def new
    @project = Project.new
  end
end
```

Dans un formulaire imbriqué, vous pouvez créer des objets enfants avec un objet parent en même temps.

```
<%= 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 %>
```

Comme nous avons initialisé `@project` avec `Project.new` pour avoir quelque chose pour créer un nouvel objet `Project` , de la même manière pour créer un objet `Todo` , nous devons avoir quelque chose comme ça, et il y a plusieurs façons de le faire:

1. Dans `Projectscontroller` , dans la `new` méthode, vous pouvez écrire: `@todo = @project.todos.build` ou `@todo = @project.todos.new` pour instancier un nouvel objet `Todo` .
2. Vous pouvez aussi le faire dans la vue: `<%= f.fields_for :todos, @project.todos.build %>`

Pour les paramètres forts, vous pouvez les inclure de la manière suivante:

```
def project_params
  params.require(:project).permit(:name, todo_attributes: [:name])
end
```

Depuis, les objets `Todo` seront créés lors de la création d'un objet `Project` . Vous devez donc spécifier cette chose dans le modèle de `Project` en ajoutant la ligne suivante:

```
accepts_nested_attributes_for :todos
```

Lire Formulaire imbriqué en Ruby on Rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/8203/formulaire-imbrique-en-ruby-on-rails

Chapitre 31: Gemmes

Remarques

Documentation Gemfile

Pour les projets qui sont censés se développer, il est `Gemfile` ajouter des commentaires à votre `Gemfile` . De cette façon, même dans les grandes configurations, vous saurez toujours ce que fait chaque bijou, même si le nom n'est pas explicite et que vous l'avez ajouté il ya 2 ans.

Cela peut également vous aider à vous rappeler pourquoi vous avez choisi une version donnée et, par conséquent, à réévaluer ultérieurement les exigences de la version.

Exemples:

```
# temporary downgrade for TeamCity
gem 'rake', '~> 10.5.0'
# To upload invoicing information to payment provider
gem 'net-sftp'
```

Examples

Qu'est-ce qu'un bijou?

Un bijou est l'équivalent d'un plugin ou d'une extension pour le langage de programmation ruby.

Être exact, même les rails, n'est rien de plus qu'un joyau. Beaucoup de gemmes sont construites sur des rails ou d'autres gemmes (elles dépendent de la gemme) ou sont autonomes.

Dans votre projet Rails

Gemfile

Pour votre projet Rails, vous avez un fichier appelé `Gemfile` . Ici, vous pouvez ajouter des gemmes que vous souhaitez inclure et utiliser dans votre projet. Une fois ajouté, vous devez installer la gem en utilisant `bundler` (voir la section `bundler`).

Gemfile.lock

Une fois que vous avez fait cela, votre `Gemfile.lock` sera mis à jour avec vos gems nouvellement ajoutés et leurs dépendances. Ce fichier verrouille vos gems utilisés afin qu'ils utilisent cette version spécifique déclarée dans ce fichier.

```
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)
```

Cet exemple est pour le bijou `devise` . Dans le `Gemfile.lock` la version `4.0.3` est déclarée pour indiquer lors de l'installation de votre projet sur une autre machine ou sur votre serveur de production la version spécifiée à utiliser.

Développement

Une seule personne, un groupe ou une communauté entière travaille et entretient une gemme. Le travail effectué est généralement libéré après certains `issues` ont été corrigés ou `features` ont été ajoutées.

Généralement, les versions suivent le principe Semantic Versioning 2.0.0 .

Bundler

Le moyen le plus simple de gérer et de gérer les gemmes consiste à utiliser un `bundler` . Bundler est un gestionnaire de paquets comparable à Bower.

Pour utiliser un bundler, vous devez d'abord l'installer.

```
gem install bundler
```

Une fois que vous avez un bundler, tout ce que vous avez à faire est d'ajouter des gemmes à votre `Gemfile` et de l'exécuter.

```
bundle
```

dans votre terminal. Cela installe vos gems nouvellement ajoutés à votre projet. En cas de problème, vous recevrez une invite dans votre terminal.

Si vous êtes intéressé par plus de détails, je vous suggère de regarder les documents .

Gemfiles

Pour commencer, les gemfiles nécessitent au moins une source, sous la forme de l'URL d'un serveur RubyGems.

Générez un Gemfile avec la source rubygems.org par défaut en exécutant `bundle init` . Utilisez https pour que votre connexion au serveur soit vérifiée avec SSL.

```
source 'https://rubygems.org'
```

Ensuite, déclarez les gemmes dont vous avez besoin, y compris les numéros de version.

```
gem 'rails', '4.2.6'
gem 'rack',  '>=1.1'
gem 'puma',  '~>3.0'
```

La plupart des spécificateurs de version, tels que> = 1.0, sont explicites. Le spécificateur ~> a une signification particulière. ~> 2.0.3 est identique à> = 2.0.3 et <2.1. ~> 2.1 est identique à> = 2.1 et <3.0. ~> 2.2.beta correspondra aux versions préliminaires comme 2.2.beta.12.

Les référentiels Git sont également des sources de gem, à condition que le référentiel en contienne un ou plusieurs. Spécifiez ce que vous voulez extraire avec `:tag` `:branch` ou `:ref` . La valeur par défaut est le `master` branche.

```
gem 'nokogiri', :git => 'https://github.com/sparklemotion/nokogiri', :branch => 'master'
```

Si vous souhaitez utiliser une gem sans décompression directement depuis le système de fichiers, définissez simplement l'option: path sur le chemin contenant les fichiers de la gem.

```
gem 'extracted_library', :path => './vendor/extracted_library'
```

Les dépendances peuvent être regroupées. Les groupes peuvent être ignorés lors de l'installation (en utilisant `--without`) ou requis en même temps (en utilisant `Bundler.require`).

```
gem 'rails_12factor', group: :production

group :development, :test do
  gem 'byebug'
  gem 'web-console', '~> 2.0'
  gem 'spring'
  gem 'dotenv-rails'
end
```

Vous pouvez spécifier la version requise de Ruby dans le Gemfile avec `ruby` . Si le Gemfile est chargé sur une autre version de Ruby, Bundler générera une exception avec une explication.

```
ruby '2.3.1'
```

Gemsets

Si vous utilisez `RVM(Ruby Version Manager)` utiliser un `gemset` pour chaque projet est une bonne idée. Un `gemset` est juste un conteneur que vous pouvez utiliser pour séparer les gemmes les unes des autres. Créer un `gemset` par projet vous permet de modifier des gemmes (et des versions de gemme) pour un projet sans casser tous vos autres projets. Chaque projet n'a besoin que de s'inquiéter de ses propres joyaux.

`RVM` fournit (> = 0.1.8) un `@global gemset` par interprète de ruby. Les gemmes que vous installez sur

le `@global gemset` pour un rubis donné sont disponibles pour tous les autres gemsets que vous créez en association avec ce ruby. C'est un bon moyen de permettre à tous vos projets de partager le même bijou installé pour une installation spécifique d'interpréteur Ruby.

Créer des gemsets

Supposons que vous avez déjà installé `ruby-2.3.1` et que vous l'avez sélectionné en utilisant cette commande:

```
rvm use ruby-2.3.1
```

Maintenant, pour créer gemset pour cette version ruby:

```
rvm gemset create new_gemset
```

où le `new_gemset` est le nom de gemset. Pour voir la liste des gemsets disponibles pour une version ruby:

```
rvm gemset list
```

lister les gemmes de toutes les versions rubis:

```
rvm gemset list_all
```

utiliser un gemset de la liste (supposons que `new_gemset` soit le gemset que je veux utiliser):

```
rvm gemset use new_gemset
```

vous pouvez également spécifier la version ruby avec le gemset si vous souhaitez passer à une autre version de ruby:

```
rvm use ruby-2.1.1@new_gemset
```

spécifier un gemset par défaut pour une version ruby particulière:

```
rvm use 2.1.1@new_gemset --default
```

pour supprimer toutes les gemmes installées d'un gemset, vous pouvez le vider par:

```
rvm gemset empty new_gemset
```

pour copier un gemset d'un rubis à un autre, vous pouvez le faire en:

```
rvm gemset copy 2.1.1@rails4 2.1.2@rails4
```

supprimer un gemset:

```
rvm gemset delete new_gemset
```

pour voir le nom actuel du gemset:

```
rvm gemset name
```

installer un bijou dans le gemset global:

```
rvm @global do gem install ...
```

Initialisation des gemsets pendant les installations Ruby

Lorsque vous installez un nouveau ruby, RVM ne crée pas seulement deux gemsets (le gemset vide et le gemset global par défaut), il utilise également un ensemble de fichiers modifiables par l'utilisateur pour déterminer les gems à installer.

En travaillant dans `~/.rvm/gemsets` , rvm recherche `global.gems` et `default.gems` utilisant une hiérarchie arborescente basée sur la chaîne ruby installée. En utilisant l'exemple de `ree-1.8.7-p2010.02` , rvm va vérifier (et importer depuis) les fichiers suivants:

```
~/.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
```

Par exemple, si vous avez modifié `~/.rvm/gemsets/global.gems` en ajoutant ces deux lignes:

```
bundler
awesome_print
```

Chaque fois que vous installez un nouveau ruby, ces deux gems sont installés dans votre gemset global. `global.gems` fichiers `default.gems` et `global.gems` sont généralement écrasés lors de la mise à jour de rvm.

Lire Gemmes en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3130/gemmes

Chapitre 32: Héritage de table unique

Introduction

L'héritage de table unique (STI) est un modèle de conception basé sur l'idée de sauvegarder les données de plusieurs modèles qui héritent tous du même modèle de base, dans une seule table de la base de données.

Examples

Exemple de base

Nous avons d'abord besoin d'un tableau pour contenir nos données

```
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
```

Puis laisse créer des modèles

```
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
```

Lorsque vous faites un `Guest.create(name: 'Bob')` ActiveRecord le traduira pour créer une entrée dans la table Users avec le `type: 'Guest'` .

Lorsque vous récupérez l'enregistrement `bob = User.where(name: 'Bob').first` l'objet renvoyé sera d'abord une instance de l' `Guest` , qui peut être traité de manière forcée en tant `bob.becomes(User)` avec `bob.becomes(User)`

devient très utile lorsque vous traitez des partials partagés ou des routes / contrôleurs de la superclasse au lieu de la sous-classe.

Colonne d'héritage personnalisé

Par défaut, le nom de la classe de modèle STI est stocké dans un `type` nommé colonne. Mais son nom peut être changé en substituant la valeur `inheritance_column` dans une classe de base. Par exemple:

```
class User < ActiveRecord::Base
  self.inheritance_column = :entity_type # can be string as well
end

class Admin < User; end
```

La migration dans ce cas ressemblera à ceci:

```
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
```

Lorsque vous effectuez `Admin.create` , cet enregistrement sera enregistré dans la table users avec `entity_type = "Admin"`

Modèle Rails avec colonne de type et sans STI

Avoir `type` colonne de `type` dans un modèle Rails sans invoquer STI peut être obtenu en affectant `:_type_disabled` à `inheritance_column` :

```
class User < ActiveRecord::Base
  self.inheritance_column = :_type_disabled
end
```

Lire Héritage de table unique en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/9125/heritage-de-table-unique

Chapitre 33: I18n - Internationalisation

Syntaxe

- I18n.t ("clé")
- I18n.translate ("key") # équivalent à `I18n.t("key")`
- I18n.t ("clé", compte: 4)
- I18n.t ("key", param1: "Quelque chose", param2: "Else")
- I18n.t ("doesnt_exist", par défaut: "key") # spécifier un défaut si la clé est manquante
- I18n.locale # =>: en
- I18n.locale =: en
- I18n.default_locale # =>: en
- I18n.default_locale =: en
- t (". key") # identique à `I18n.t("key")` , mais à la portée de l'action / du modèle appelé

Examples

Utiliser I18n dans les vues

En supposant que vous avez ce fichier de paramètres régionaux YAML:

```
# config/locales/en.yml
en:
  header:
    title: "My header title"
```

et vous voulez afficher votre chaîne de titre, vous pouvez le faire

```
# in ERB files
<%= t('header.title') %>

# in SLIM files
= t('header.title')
```

I18n avec des arguments

Vous pouvez passer des paramètres à la méthode **I18n** `t` :

```
# Example config/locales/en.yml
en:
  page:
    users: "%{users_count} users currently online"

# In models, controller, etc...
I18n.t('page.users', users_count: 12)

# In views
```

```
# ERB
<%= t('page.users', users_count: 12) %>

#SLIM
= t('page.users', users_count: 12)

# Shortcut in views - DRY!
# Use only the dot notation
# Important: Consider you have the following controller and view page#users

# ERB Example app/views/page/users.html.erb
<%= t('.users', users_count: 12) %>
```

Et obtenez la sortie suivante:

```
"12 users currently online"
```

Pluralisation

Vous pouvez laisser **I18n** gérer la pluralisation pour vous, utilisez simplement l'argument `count` .

Vous devez configurer votre fichier de paramètres régionaux comme ceci:

```
# config/locales/en.yml
en:
  online_users:
    one: "1 user is online"
    other: "%{count} users are online"
```

Et puis, utilisez la clé que vous venez de créer en transmettant l'argument `count` à l'assistant `I18n.t` :

```
I18n.t("online_users", count: 1)
#=> "1 user is online"

I18n.t("online_users", count: 4)
#=> "4 users are online"
```

Définir la localisation via les requêtes

Dans la plupart des cas, vous souhaiterez peut-être définir les **`I18n`** régionaux **`I18n`** . Vous pouvez définir les paramètres régionaux de la session en cours, de l'utilisateur actuel ou d'un paramètre d'URL. Cela est facilement réalisable en implémentant une action `before_action` dans l'un de vos contrôleurs ou dans `ApplicationController` dans tous vos contrôleurs.

```
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
```

Basé sur URL

Le `locale` régional pourrait provenir d'une URL comme celle-ci

```
http://yourapplication.com/products?locale=en
```

Ou

```
http://yourapplication.com/en/products
```

Pour atteindre ce dernier, vous devez modifier vos `routes` en ajoutant une `scope` :

```
# config/routes.rb
scope "(:locale)", locale: /en|fr/ do
  resources :products
end
```

Ce faisant, en visitant `http://yourapplication.com/en/products` , vous définissez vos paramètres régionaux sur `:en` . Au lieu de cela, la visite de `http://yourapplication.com/fr/products` le définira comme `http://yourapplication.com/fr/products :fr` . De plus, vous ne recevrez pas d'erreur de routage en cas d'absence du `:locale` , car la visite de `http://yourapplication.com/products` chargera les **paramètres** régionaux **I18n** par défaut.

Basé sur la session ou basé sur la persistance

Cela suppose que l'utilisateur peut cliquer sur un indicateur de bouton / langue pour changer de langue. L'action peut être acheminée vers un contrôleur qui définit la session sur la langue en cours (et éventuellement persister les modifications apportées à une base de données si l'utilisateur est connecté).

```
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
```

> Vous devez définir sanitize_language_param avec votre liste de langues disponibles et éventuellement gérer les erreurs si la langue n'existe pas.

Si vous avez très peu de langues, il peut être utile de les définir comme ceci:

```
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
```

Note: n'oubliez pas d'ajouter des routes à vos actions `change_language`

Paramètres régionaux par défaut

N'oubliez pas que vous devez définir les paramètres régionaux par défaut de votre application. Vous pouvez le faire soit en le `config/application.rb` dans `config/application.rb` :

```
config.i18n.default_locale = :de
```

ou en créant un initialiseur dans le dossier `config/initializers` :

```
# config/initializers/locale.rb
I18n.default_locale = :it
```

Obtenir les paramètres régionaux à partir d'une requête HTTP

Parfois, il peut être utile de définir les paramètres régionaux de votre application en fonction de l'adresse IP de la requête. Vous pouvez facilement y parvenir en utilisant **`Geocoder`** . Parmi les nombreuses fonctions de **`Geocoder`** , il peut également indiquer l' `location` d'une `request` .

Tout d'abord, ajoutez **`Geocoder`** à votre `Gemfile`

```
# Gemfile
gem 'geocoder'
```

`Geocoder` ajoute des méthodes de `location` et `safe_location` à l'objet `Rack::Request` standard afin que vous puissiez facilement rechercher l'emplacement de toute requête HTTP par adresse IP. Vous pouvez utiliser cette méthode dans une action `before_action` dans votre `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
```

Attention, cela ne fonctionnera pas dans `test` environnements de `development` et de `test` , car des éléments tels que `0.0.0.0` et `localhost` sont des adresses IP Internet valides.

Limitations et alternatives

`Geocoder` est très puissant et flexible, mais doit être configuré pour fonctionner avec un *service de géocodage* (voir plus de détails); beaucoup d'entre eux limitent l'utilisation. Il convient également de garder à l'esprit que l'appel à un service externe à chaque demande peut avoir un impact sur les performances.

Pour y remédier, il peut également être intéressant de considérer:

1. Une solution hors ligne

L'utilisation d'un joyau comme **`GeoIP`** (voir ici) permet d'effectuer des recherches sur un fichier de données local. Il peut y avoir un compromis en termes de précision, car ces fichiers de données doivent être tenus à jour.

2. Utilisez CloudFlare

Les pages servies via CloudFlare peuvent être géocodées de manière transparente, le code de pays étant ajouté à l'en-tête (`HTTP_CF_IPCOUNTRY`). Plus de détails peuvent être trouvés ici .

Traduction des attributs du modèle ActiveRecord

`globalize` gem est une excellente solution pour ajouter des traductions à vos modèles `ActiveRecord` . Vous pouvez l'installer en ajoutant ceci à votre `Gemfile` :

```
gem 'globalize', '~> 5.0.0'
```

Si vous utilisez `Rails 5` vous devrez également ajouter `activemodel-serializers-xml`

```
gem 'activemodel-serializers-xml'
```

Les traductions de modèles vous permettent de traduire les valeurs d'attributs de vos modèles, par exemple:

```
class Post < ActiveRecord::Base
  translates :title, :text
end

I18n.locale = :en
post.title # => Globalize rocks!

I18n.locale = :he
post.title # => גלובאלייז 2 שלום!
```

Après avoir défini les attributs de modèle à traduire, vous devez créer une table de traduction via une migration. **`globalize`** fournit `create_translation_table!` et `drop_translation_table!` .

Pour cette migration, vous devez utiliser `up` et `down` , et **non pas** `change` . De plus, pour réussir cette migration, vous devez d'abord définir les attributs traduits dans votre modèle, comme indiqué ci-dessus. Voici une migration correcte du modèle `Post` précédent:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table! title: :string, text: :text
  end

  def down
    Post.drop_translation_table!
  end
end
```

Vous pouvez également passer des options pour des options spécifiques, telles que:

```
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
```

Si vous avez déjà **des données existantes** dans vos colonnes de traduction requises, vous pouvez facilement les migrer vers la table de traductions, en ajustant votre migration:

```
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
```

Assurez-vous de supprimer les colonnes traduites de la table parente après la migration sécurisée de toutes vos données. Pour supprimer automatiquement les colonnes traduites de la table parent après la migration des données, ajoutez l'option `remove_source_columns` à la migration:

```
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
```

Vous pouvez également ajouter de nouveaux champs à une table de traductions créée précédemment:

```
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
```

Utiliser I18n avec des balises et des symboles HTML

```
# config/locales/en.yml
en:
  stackoverflow:
    header:
      title_html: "Use <strong>I18n</strong> with Tags &amp; Symbols"
```

Notez l'ajout de `_html` supplémentaire après le `title` du nom.

Et dans les vues,

```
# ERB
<h2><%= t(:title_html, scope: [:stackoverflow, :header]) %></h2>
```

Lire I18n - Internationalisation en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2772/i18n---internationalisation

Chapitre 34: Identification amicale

Introduction

FriendlyId est le "bulldozer de l'armée suisse" des plugins de slugging et permalink pour Active Record. Il vous permet de créer de jolies URL et de travailler avec des chaînes conviviales, comme si elles étaient des identificateurs numériques. Avec FriendlyId, il est facile de faire en sorte que votre application utilise des URL telles que:

http://example.com/states/washington

Examples

Rails Quickstart

```
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
```

éditer app / models / user.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')
```

Une nouvelle fonctionnalité "candidats" qui facilite la mise en place d'une liste de slugs alternatifs pouvant être utilisés pour distinguer de manière unique les enregistrements, plutôt que d'ajouter une séquence. Par exemple:

```
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
```

Définir la longueur limite du slug en utilisant la gem friendly_id?

```
def normalize_friendly_id(string)
   super[0..40]
end
```

Lire Identification amicale en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/9664/identification-amicale

Chapitre 35: Importer des fichiers CSV entiers à partir d'un dossier spécifique

Introduction

Dans cet exemple, disons que nous avons beaucoup de fichiers CSV dans un dossier. Chaque fichier CSV doit télécharger notre base de données depuis notre console pour écrire une commande. Exécutez la commande suivante dans un projet nouveau ou existant pour créer ce modèle.

Examples

Télécharge le fichier CSV à partir de la commande de la console

Commandes Terminal:

```
rails g model Product name:string quantity:integer price:decimal{12,2}
rake db:migrate
```

Les Lates créent un contrôleur.

Commandes Terminal:

```
rails g controller Products
```

Code du contrôleur:

```
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
```

Modèle:

```
class Product< ApplicationRecord
  def self.import(file)
      CSV.foreach(file.path, headers: true) do |row|
          Product.create! row.to_hash
      end
  end
end
```

routes.rb

```
resources :products
```

app / config / application.rb

```
require 'csv'
```

Maintenant, ouvrez votre `console` développement et `run`

```
=> ProductsController.new.create #=> Uploads your whole CSV files from your folder directory
```

Lire Importer des fichiers CSV entiers à partir d'un dossier spécifique en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/8658/importer-des-fichiers-csv-entiers-a-partir-d-un-dossier-specifique

Chapitre 36: Intégration de React.js avec Rails à l'aide d'Hyperloop

Introduction

Cette rubrique couvre l'intégration de React.js aux Rails à l'aide du joyau Hyperloop

D'autres approches non couvertes ici utilisent les gemmes react-rails ou react_on_rails.

Remarques

Les classes de composants génèrent simplement les classes de composants javascript équivalents.

Vous pouvez également accéder aux composants et aux bibliothèques javascript directement à partir de vos classes de composants Ruby.

Hyperloop va "prérégler" le côté serveur de vue pour que votre vue initiale se charge exactement comme les modèles ERB ou HAML. Une fois chargée sur le client, la réaction prend le relais et mettra à jour progressivement le DOM en tant que changement d'état dû aux entrées de l'utilisateur, aux requêtes HTTP ou aux données de socket Web entrantes.

Outre les composants, Hyperloop a des magasins pour gérer l'état partagé, les opérations pour encapsuler la logique métier isomorphe et les modèles qui donnent un accès direct à vos modèles ActiveRecord sur le client en utilisant la syntaxe AR standard.

Plus d'infos ici: http://ruby-hyperloop.io/

Examples

Ajouter un simple composant de réaction (écrit en ruby) à votre application Rails

1. Ajoutez le joyau hyperloop à vos rails (4.0 - 5.1) Gemfile
2. `bundle install`
3. Ajoutez le manifeste hyperloop au fichier application.js:

```
// app/assets/javascripts/application.js
...
//= hyperloop-loader
```

4. Créez vos composants de réaction et placez-les dans le `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. Les composants agissent comme des vues. Ils sont "montés" en utilisant la méthode `render_component` dans un contrôleur:

```
# somewhere in a controller:
  ...
  def hello_world
    render_component # renders HelloWorld based on method name
  end
```

Déclaration des paramètres du composant (accessoires)

```
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'
```

Balises HTML

```
# 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
```

Gestionnaires d'événements

```
# 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
```

États

```
# 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
```

Notez que les états peuvent être partagés entre les composants en utilisant Hyperloop :: Stores

Rappels

```
# 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
```

Lire Intégration de React.js avec Rails à l'aide d'Hyperloop en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/9809/integration-de-react-js-avec-rails-a-l-aide-d-hyperloop

Chapitre 37: Interface de requête ActiveRecord

Introduction

ActiveRecord est le M dans MVC qui est la couche du système chargée de représenter les données et la logique métier. La technique qui relie les objets riches d'une application à tables dans un système de gestion de base de données relationnelle est **O** bjet **R** elational **M** apper **(ORM).**

ActiveRecord effectuera des requêtes sur la base de données pour vous et est compatible avec la plupart des systèmes de base de données. Quel que soit le système de base de données que vous utilisez, le format de la méthode ActiveRecord sera toujours le même.

Examples

.où

La méthode `where` est disponible sur tout modèle `ActiveRecord` et permet d'interroger la base de données pour un ensemble d'enregistrements correspondant aux critères donnés.

La méthode `where` accepte un hachage où les clés correspondent aux noms de colonne de la table que le modèle représente.

Comme exemple simple, nous utiliserons le modèle suivant:

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

Pour trouver toutes les personnes avec le prénom de `Sven` :

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

Pour trouver toutes les personnes avec le prénom de `Sven` et le nom de famille de `Schrodinger` :

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

Dans l'exemple ci - dessus, la sortie sql montre que les enregistrements ne seront retournés à la fois si le `first_name` et la `last_name` correspondance.

requête avec condition OU

Pour rechercher des enregistrements avec `first_name == 'Bruce'` OR `last_name == 'Wayne'`

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

.part avec un tableau

La méthode `where` de tout modèle ActiveRecord peut être utilisée pour générer du code SQL de la forme `WHERE column_name IN (a, b, c, ...)` . Ceci est réalisé en passant un tableau en argument.

Comme exemple simple, nous utiliserons le modèle suivant:

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

people = Person.where(first_name: ['Mark', 'Mary'])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary')"
```

Si le tableau contient un `nil` , le SQL sera modifié pour vérifier si la colonne est `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"
```

Les portées

Les portées agissent comme des filtres prédéfinis sur les modèles `ActiveRecord` .

Une portée est définie à l'aide de la méthode de la classe de `scope` .

Comme exemple simple, nous utiliserons le modèle suivant:

```
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
```

Les étendues peuvent être appelées directement à partir de la classe de modèle:

```
minors = Person.minors
```

Les portées peuvent être chaînées:

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

La méthode `where` et les autres méthodes de type requête peuvent également être chaînées:

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

En coulisses, les scopes sont simplement du sucre syntaxique pour une méthode de classe standard. Par exemple, ces méthodes sont fonctionnellement identiques:

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

# This ^ is the same as this:

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

Portée par défaut

dans votre modèle pour définir une étendue par défaut pour toutes les opérations sur le modèle.

> Il y a une différence notable entre la méthode de `scope` et la méthode de classe: les étendues définies par la `scope` renvoient *toujours* un `ActiveRecord::Relation` , même si la logique dans un résultat est nulle. Les méthodes de classe, cependant, n'ont pas un tel filet de sécurité et peuvent briser la chaîne si elles renvoient autre chose.

où.pas

`where` clauses peuvent être annulées en utilisant la syntaxe `where.not` :

```
class Person < ApplicationRecord
  #attribute :first_name, :string
end

people = Person.where.not(first_name: ['Mark', 'Mary'])
# => SELECT "people".* FROM "people" WHERE "people"."first_name" NOT IN ('Mark', 'Mary')
```

Pris en charge par ActiveRecord 4.0 et versions ultérieures.

Commande

Vous pouvez commander les résultats de la requête **ActiveRecord** à l'aide de `.order` :

```
User.order(:created_at)
#=> => [#<User id: 2, created_at: "2015-08-12 21:36:23">, #<User id: 11, created_at: "2015-08-
15 10:21:48">]
```

Si non spécifié, la commande sera effectuée dans l'ordre croissant. Vous pouvez le spécifier en faisant:

```
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` accepte également une chaîne, vous pouvez donc aussi le faire

```
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">]
```

Comme la chaîne est du SQL brut, vous pouvez également spécifier une table et pas seulement un attribut. En supposant que vous voulez commander les `users` fonction de leur nom de `role` , vous pouvez le faire:

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

Class Role < ActiveRecord::Base
  has_many :users
end

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

La portée de la `order` peut également accepter un noeud Arel:

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

Méthodes ActiveRecord Bang (!)

Si vous avez besoin d'une méthode **ActiveRecord** pour générer une exception au lieu d'une valeur `false` en cas d'échec, vous pouvez ajouter `!` pour eux. C'est très important. Comme certaines exceptions / échecs sont difficiles à détecter si vous n'utilisez pas! sur eux. J'ai recommandé de le faire dans votre cycle de développement pour écrire tout votre code ActiveRecord de cette manière afin de vous faire gagner du temps et d'éviter des problèmes.

```
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
```

Les méthodes **ActiveRecord** qui acceptent un *bang* (`!`) Sont:

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

- `.last!`
- `.find_by!`
- `.find_or_create_by!`
- `#save!`
- `#update!`
- tous les détecteurs dynamiques AR

.find_by

Vous pouvez trouver des enregistrements par n'importe quel champ de votre table en utilisant `find_by` .

Donc, si vous avez un modèle `User` avec un attribut `first_name` vous pouvez le faire:

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

`find_by` que `find_by` ne lance aucune exception par défaut. Si le résultat est un ensemble vide, il renvoie `nil` au lieu de `find` .

Si l'exception est nécessaire, utilisez `find_by!` qui déclenche une erreur `ActiveRecord::RecordNotFound` comme `find` .

.delete_all

Si vous devez supprimer beaucoup d'enregistrements rapidement, **ActiveRecord** donne la méthode **`.delete_all`** . être appelé directement sur un modèle, pour supprimer tous les enregistrements de cette table ou une collection. Attention cependant, comme **`.delete_all`** aucun objet et ne fournit donc aucun rappel (`before_*` et `after_destroy` ne sont pas déclenchés).

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

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

ActiveRecord recherche insensible à la casse

Si vous devez rechercher un modèle ActiveRecord pour des valeurs similaires, vous pouvez être tenté d'utiliser `LIKE` ou `ILIKE` mais ce n'est pas portable entre les moteurs de base de données. De même, le recours à la réduction des coûts ou à la mise à jour permanente peut créer des problèmes de performance.

Vous pouvez utiliser la méthode de `matches` Arel sous-jacente d'ActiveRecord pour le faire en toute sécurité:

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

Arel appliquera la construction LIKE ou ILIKE appropriée pour le moteur de base de données

configuré.

Obtenez le premier et le dernier enregistrement

Les rails ont un moyen très facile d'obtenir le `first` et le `last` enregistrement de la base de données.

Pour obtenir le `first` enregistrement de la table des `users` , vous devez taper la commande suivante:

```
User.first
```

Il va générer une requête `sql` suivante:

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

Et reviendra l'enregistrement suivant:

```
#<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 >
```

Pour obtenir le `last` enregistrement de la table des `users` , vous devez taper la commande suivante:

```
User.last
```

Il va générer une requête `sql` suivante:

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

Et reviendra l'enregistrement suivant:

```
#<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 >
```

Passer un entier à la **première** et à la **dernière** méthode crée une requête **LIMIT** et renvoie un tableau d'objets.

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

Il générera la requête `sql` suivante.

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

Et

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

Il générera la requête `sql` suivante.

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

.group et .count

Nous avons un modèle de `Product` et nous voulons les regrouper par `category` .

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

Cela interrogera la base de données comme suit:

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

Assurez-vous que le champ groupé est également sélectionné. Le regroupement est particulièrement utile pour compter l'occurrence - dans ce cas - des `categories` .

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

Comme l'indique la requête, la base de données sera utilisée pour le comptage, ce qui est beaucoup plus efficace que la récupération de tous les enregistrements et le comptage dans le code:

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

.distinct (ou .uniq)

Si vous souhaitez supprimer les doublons d'un résultat, vous pouvez utiliser `.distinct()` :

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

Cela interroge la base de données comme suit:

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

`.uniq()` a le même effet. Avec Rails 5.0, il est devenu obsolète et il sera supprimé de Rails avec la version 5.1. La raison en est que le mot `unique` n'a pas le même sens que distinct et peut être trompeur. De plus, `distinct` est plus proche de la syntaxe SQL.

Joint

`joins()` vous permet de joindre des tables à votre modèle actuel. Pour ex.

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

produira la requête SQL suivante:

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

Ayant joint la table, vous y aurez accès:

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

Faites attention au pluriel. Si votre relation est `:has_many` , alors l'argument `joins()` doit être pluralisé. Sinon, utilisez singulier.

Emboîtée `joins` :

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

qui produira:

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

Comprend

ActiveRecord avec `includes` garantit que toutes les associations spécifiées sont chargées en utilisant le nombre minimum possible de requêtes. Ainsi, lorsque vous interrogez une table pour des données avec une table associée, les deux tables sont chargées en mémoire.

```
@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 } )` ne chargera que les **auteurs** avec des conditions en mémoire **sans charger les livres** . Utilisez des `joins` lorsque des informations supplémentaires sur les associations imbriquées ne sont pas requises.

```
@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
```

Limite et décalage

Vous pouvez utiliser `limit` pour indiquer le nombre d'enregistrements à récupérer et utiliser `offset` pour indiquer le nombre d'enregistrements à ignorer avant de commencer à renvoyer les enregistrements.

Par exemple

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

Il générera la requête SQL suivante.

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

Comme offset n'est pas mentionné dans la requête ci-dessus, il retournera les trois premiers enregistrements.

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

Il générera la requête SQL suivante.

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

Lire Interface de requête ActiveRecord en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2154/interface-de-requete-activerecord

Chapitre 38: Le débogage

Examples

Application de rails de débogage

Pour pouvoir déboguer une application, il est très important de comprendre le flux de la logique et des données d'une application. Il aide à résoudre les bogues logiques et ajoute de la valeur à l'expérience de programmation et à la qualité du code. Le débogueur (pour Ruby 1.9.2 et 1.9.3) et le byebug (pour ruby> = 2.x) sont deux joyaux populaires pour le débogage.

Pour déboguer les fichiers `.rb` , procédez comme suit:

1. Ajouter le `debugger` ou le `byebug` au groupe de `development` de `Gemfile`
2. Exécuter une `bundle install`
3. Ajouter un `debugger` ou un `byebug` comme point d'arrêt
4. Exécuter le code ou faire une demande
5. Voir le journal du serveur de rails arrêté au point d'arrêt spécifié
6. A ce stade, vous pouvez utiliser votre terminal serveur comme la `rails console` et vérifier les valeurs de la variable et des paramètres.
7. Pour passer à l'instruction suivante, tapez `next` et appuyez sur `enter`
8. Pour sortir de type `c` et appuyez sur `enter`

Si vous souhaitez déboguer les fichiers `.html.erb` , un point d'arrêt sera ajouté en tant que `<% debugger %>`

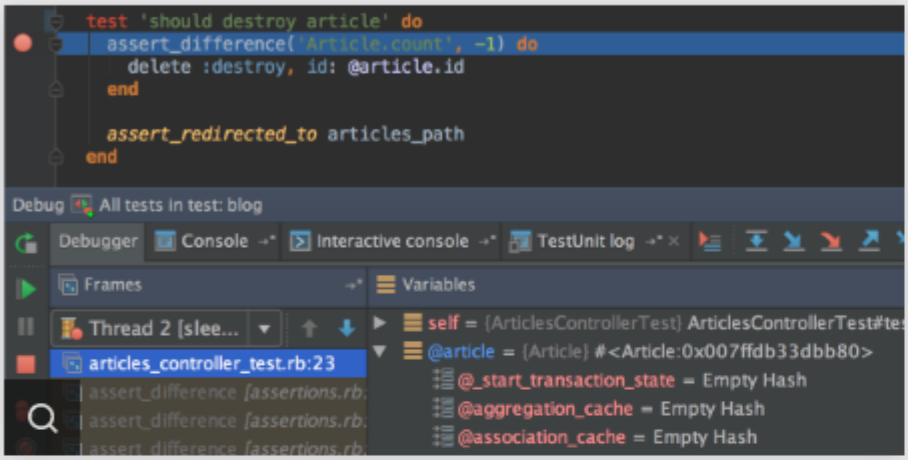
Déboguer dans votre IDE

Chaque bon IDE fournit une interface graphique pour le débogage interactif des applications Ruby (et donc Rails) où vous pouvez ajouter des points d'arrêt, des montres, une pause automatique à l'exception et suivre l'exécution du code, étape par étape, ligne par ligne.

Par exemple, regardez l'un des meilleurs IDE Ruby, les fonctionnalités de débogage de RubyMine sur la photo.

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 ti
  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

Déboguer en soulevant des exceptions est *bien plus simple* que de plisser des instructions dans `print` journaux d' `print` et, pour la plupart des bogues, il est généralement *beaucoup plus rapide* que d'ouvrir un débogueur comme `pry` ou `byebug` . Ces outils ne devraient pas être votre première étape.

Débogage rapide de Ruby / Rails:

1. Méthode rapide: élevez une `Exception` puis `.inspect` son résultat

Le moyen le *plus rapide* de déboguer le code Ruby (en particulier Rails) consiste à `raise` une exception le long du chemin d'exécution de votre code tout en appelant `.inspect` sur la méthode ou l'objet (par exemple, `foo`):

```
raise foo.inspect
```

Dans le code ci-dessus, *`raise`* déclenche une *`Exception`* qui *interrompt l'exécution de votre code* et renvoie un message d'erreur contenant des informations `.inspect` sur l'objet / la méthode (`foo`) sur la ligne que vous essayez de déboguer.

Cette technique est utile pour examiner *rapidement* un objet ou une méthode (*par exemple, est-ce que c'est `nil` ?*) Et pour confirmer immédiatement si une ligne de code est même exécutée dans un contexte donné.

2. Repli: utilisez un débogueur *IRB* ruby tel que `byebug` ou `pry`

Seulement après avoir des informations sur l'état de votre code flux d'exécution si vous envisagez de passer à un bijou rubis débogueur comme irb `pry` ou `byebug` où vous pouvez plonger plus profondément dans l'état des objets dans votre chemin d'exécution.

Pour utiliser le joyau de `byebug` pour le débogage dans Rails:

1. Ajouter `gem 'byebug'` dans le groupe de *développement* de votre *Gemfile*
2. Exécuter une `bundle install`
3. Ensuite, pour utiliser, insérez la phrase `byebug` dans le chemin d'exécution du code que vous voulez examiner.

`byebug` fois exécutée, `byebug` variable de `byebug` ouvrira une session IRB ruby de votre code, vous donnant un accès direct à l'état des objets tels qu'ils sont à ce stade de l'exécution du code.

Les débogueurs IRB tels que Byebug sont utiles pour analyser en profondeur l'état de votre code lors de son exécution. Cependant, leur procédure est plus longue que la génération d'erreurs. Dans la plupart des cas, elles ne devraient donc pas être votre première étape.

Conseil général pour débutant

Lorsque vous essayez de déboguer un problème, il est recommandé de toujours: **Lire le message d'erreur! @ # $ Ing (RTFM)**

Cela signifie lire les messages d'erreur *soigneusement* et *complètement* avant d'agir afin que vous *compreniez ce que l'on essaie de vous dire.* Lorsque vous déboguez, posez les questions mentales suivantes, *dans cet ordre* , lors de la lecture d'un message d'erreur:

1. Quelle est la ***classe de*** référence de l'erreur? (c.-à *-d. ai-je la classe d'objet correcte ou mon objet est-il* `nil` *?*)
2. Quelle est la ***méthode de*** référence de l'erreur? (c.-à - d. *est-ce un type dans la méthode? Puis-je appeler cette méthode sur ce type / cette classe d'objet?*)
3. Enfin, en utilisant ce que je peux déduire de mes deux dernières questions, quelles ***lignes de code*** dois-je rechercher? (rappelez-vous: la dernière ligne de code de la trace de la pile n'est pas nécessairement à l'endroit du problème.)

Dans la trace de pile, faites particulièrement attention aux lignes de code qui proviennent de votre projet (par exemple, les lignes commençant par `app/...` si vous utilisez Rails). 99% du temps, le problème concerne votre propre code.

Illustrer pourquoi interpréter *dans cet ordre* est important ...

Par exemple, un message d'erreur Ruby qui confond beaucoup de débutants:

Vous exécutez du code qui s'exécute à un moment donné en tant que tel:

```
@foo = Foo.new

...

@foo.bar
```

et vous obtenez une erreur qui indique:

```
undefined method "bar" for Nil:nilClass
```

Les débutants voient cette erreur et pensent que le problème est que la `bar` méthode n'est *pas définie* . **Ce n'est pas.** Dans cette erreur, la vraie partie qui compte est:

```
for Nil:nilClass
```

`for Nil:nilClass` signifie que `@foo` est `@foo` ! `@foo` n'est pas une variable d'instance `Foo` ! Vous avez un objet qui est `Nil` . Lorsque vous voyez cette erreur, c'est simplement ruby en essayant de vous dire que la `bar` méthode n'existe pas pour les objets de la classe `Nil` . (eh bien, puisque nous essayons d'utiliser une méthode pour un objet de la classe `Foo` not `Nil`).

Malheureusement, en raison de la façon dont cette erreur est écrite (`undefined method "bar" for`

`Nil:nilClass`), il est facile de penser que cette erreur est `undefined method "bar" for Nil:nilClass` au fait que la `bar` n'est `undefined` . Lorsqu'elle n'est pas lue attentivement, cette erreur amène par erreur les débutants à fouiller dans les détails de la méthode de `bar` sur `Foo` , manquant complètement la partie de l'erreur qui indique que l'objet est de la mauvaise classe (ici: nil). C'est une erreur qui peut être facilement évitée en lisant l'intégralité des messages d'erreur.

Résumé:

Lisez toujours attentivement **le message d'erreur *complet*** avant de commencer tout débogage. Cela signifie: vérifiez toujours le type de ***classe*** d'un objet dans un message d'erreur en *premier* , puis ses ***méthodes*** , *avant de* commencer à rechercher des empilements ou des lignes de code dans lesquels vous pensez que l'erreur se produit. Ces 5 secondes peuvent vous faire économiser 5 heures de frustration.

tl; dr: Ne plissez pas les journaux d'impression **:** placez plutôt des exceptions. Évitez les trous de lapin en lisant attentivement les erreurs avant le débogage.

Débogage de l'application ruby-on-rails avec pry

Pry est un outil puissant qui peut être utilisé pour déboguer n'importe quelle application Ruby. Mettre en place une application Ruby-on-Rails avec cette gemme est très simple et facile.

Installer

Pour commencer à déboguer votre application avec pry

- Ajoutez `gem 'pry'` au `Gemfile` l'application et regroupez-le

```
group :development, :test do
  gem 'pry'
end
```

- Accédez au répertoire racine de l'application sur la console du terminal et exécutez l' `bundle install` . Vous êtes prêt à l'utiliser partout dans votre application.

Utilisation

L'utilisation de Pry dans votre application consiste simplement à inclure `binding.pry` sur les points d'arrêt que vous souhaitez inspecter lors du débogage. Vous pouvez ajouter des points d'arrêt `binding.pry` n'importe où dans votre application interprétée par l'interpréteur de ruby (tous les fichiers app / controllers, app / models, app / views)

i) Déboguer un contrôleur

app / controllers / 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
```

Dans cet exemple, le serveur rails s'arrête avec une console au point d'arrêt lorsque vous essayez de visiter un routage de page pour `show` action sur `UsersController` . Vous pouvez inspecter l'objet `params` et faire une requête ActiveRecord sur `User` modèle `User` partir de ce point d'arrêt

ii) Déboguer une vue

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"
```

Dans cet exemple, le point d'arrêt s'arrête avec la console pry lorsque la page `users/show` est précompilée sur le serveur rails avant de la renvoyer au navigateur du client. Ce point de rupture permet de déboguer la correction de `@user.logged_in?` quand il se comporte mal.

ii) Déboguer un modèle

```
app/models/user.rb

class User < ActiveRecord::Base
  def full_name
    binding.pry
    "#{self.first_name} #{self.last_name}"
  end
end
```

Dans cet exemple, le point de rupture peut être utilisé pour déboguer `User` méthode par exemple de modèle `full_name` lorsque cette méthode est appelée à partir de n'importe où dans l'application.

En conclusion, pry est un puissant outil de débogage pour les applications de rails avec une configuration simple et des directives simples de débogage. Essayez ceci.

Lire Le débogage en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3877/le-debogage

Chapitre 39: Le modèle indique: AASM

Examples

Etat de base avec AASM

Habituellement, vous finirez par créer des modèles qui contiendront un état et cet état changera pendant la durée de vie de l'objet.

AASM est une bibliothèque d'activateur de machine à états finis qui peut vous aider à gérer facilement la conception de processus de vos objets.

Avoir quelque chose comme ça dans votre modèle va assez bien avec l'idée de Fat Model, Skinny Controller , l'une des meilleures pratiques de Rails. Le modèle est le seul responsable de gérer son état, ses modifications et de générer les événements déclenchés par ces changements.

Pour installer, dans Gemfile

```
gem 'aasm'
```

Considérons une application où l'utilisateur cite un produit pour un prix.

```
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
```

Les états de la classe de devis peuvent aller, mais c'est le mieux pour votre processus.

Vous pouvez penser aux états comme étant passés, comme dans l'exemple précédent ou à d'autres moments, par exemple: tarification, paiement, livraison, etc. La désignation des États dépend de vous. D'un point de vue personnel, les états passés fonctionnent mieux car votre état final sera sûrement une action passée et sera mieux lié aux noms des événements, ce qui sera expliqué plus tard.

REMARQUE: Faites attention aux noms que vous utilisez, vous devez vous soucier de ne pas utiliser les mots-clés réservés Ruby ou Ruby on Rails, tels que `valid` , `end` , `being` , etc.

Après avoir défini les états et les transitions, nous pouvons maintenant accéder à certaines méthodes créées par AASM.

Par exemple:

```
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.
```

Comme vous pouvez voir que l'événement a des transitions, ces transitions déterminent la manière dont l'état changera lors de l'appel de l'événement. Si l'événement est invalide en raison de l'état actuel, une erreur sera générée.

Les événements et les transitions ont également d'autres callbacks, par exemple

```
guard: product_delivered?
```

`product_delivered?` le `product_delivered?` méthode qui retournera un booléen. S'il s'avère faux, la transition ne sera pas appliquée et si aucune autre transition n'est disponible, l'état ne changera pas.

```
success: :reverse_charges
```

Si cette traduction réussit, la méthode `:reverse_charges` sera appelée.

Il existe plusieurs autres méthodes dans AASM avec plus de rappels dans le processus, mais cela vous aidera à créer vos premiers modèles avec des états finis.

Lire Le modèle indique: AASM en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/7826/le-modele-indique--aasm

Chapitre 40: Le routage

Introduction

Le routeur Rails reconnaît les URL et les distribue à l'action d'un contrôleur. Il peut également générer des chemins et des URL, évitant d'avoir à coder en dur des chaînes dans vos vues.

Remarques

"Routage" est en général la manière dont les URL sont "gérées" par votre application. Dans le cas de Rails, c'est généralement quel contrôleur et quelle action de ce contrôleur traitera une URL entrante particulière. Dans les applications Rails, les routes sont généralement placées dans le fichier `config/routes.rb` .

Examples

Routage des ressources (de base)

Les routes sont définies dans `config/routes.rb` . Ils sont souvent définis comme un groupe d'itinéraires connexes, en utilisant les `resources` ou `resource` méthodes de `resource` .

`resources :users` créent les sept itinéraires suivants, tous `UsersController` aux actions de `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'
```

Les noms des actions sont affichés après le `#` dans le paramètre `to` ci-dessus. Les méthodes avec ces mêmes noms doivent être définies dans `app/controllers/users_controller.rb` comme suit:

```
class UsersController < ApplicationController
  def index
  end

  def create
  end

  # continue with all the other methods…
end
```

Vous pouvez limiter les actions générées avec `only` ou `except` :

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

Vous pouvez afficher tous les itinéraires de votre application à tout moment en exécutant:

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
```

Pour afficher uniquement les itinéraires mappés sur un contrôleur particulier:

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
```

Vous pouvez rechercher dans les itinéraires en utilisant l'option `-g` . Cela montre toute route qui correspond partiellement au nom de la méthode d'assistance, au chemin de l'URL ou au verbe HTTP:

5.0

```
$ rake routes -g new_user     # Matches helper method
$ rake routes -g POST         # Matches HTTP Verb POST
```

5.0

```
$ rake routes -g new_user     # Matches helper method
$ rake routes -g POST         # Matches HTTP Verb POST
# OR
$ rails routes -g new_user    # Matches helper method
$ rails routes -g POST        # Matches HTTP Verb POST
```

De plus, lorsque vous exécutez le serveur `rails` en mode développement, vous pouvez accéder à une page Web qui affiche tous vos itinéraires avec un filtre de recherche, priorisé de haut en bas, à `<hostname>/rails/info/routes` . Il ressemblera à ceci:

Assistant	Verbe HTTP	Chemin	Action du contrôleur
Chemin / URL		[Path Match]	
chemin_utilisateur	OBTENIR	/users(.:format)	utilisateurs # index
	POSTER	/users(.:format)	les utilisateurs # créent
new_user_path	OBTENIR	/users/new(.:format)	utilisateurs # new
edit_user_path	OBTENIR	/users/:id/edit(.:format)	utilisateurs # modifier
chemin_utilisateur	OBTENIR	/users/:id(.:format)	utilisateurs # montrent
	PIÈCE	/users/:id(.:format)	utilisateurs # mise à jour
	METTRE	/users/:id(.:format)	utilisateurs # mise à jour
	EFFACER	/users/:id(.:format)	les utilisateurs # détruisent

Les routes peuvent être déclarées disponibles uniquement pour les membres (pas les collections) à l'aide de la `resource` méthode au lieu de `resources` dans `routes.rb` . Avec la `resource` , une route d' `index` n'est pas créée par défaut, mais uniquement lorsque vous en demandez explicitement une comme ceci:

```
resource :orders, only: [:index, :create, :show]
```

Contraintes

Vous pouvez filtrer les routes disponibles à l'aide de contraintes.

Il existe plusieurs manières d’utiliser des contraintes, notamment:

- contraintes de segment ,
- contraintes basées sur les demandes
- contraintes avancées

Par exemple, une contrainte basée sur la demande pour autoriser uniquement une adresse IP spécifique à accéder à une route:

```
constraints(ip: /127\.0\.0\.1$/) do
  get 'route', to: "controller#action"
end
```

Voir d'autres exemples similaires ActionDispatch :: Routing :: Mapper :: Scoping .

Si vous voulez faire quelque chose de plus complexe, vous pouvez utiliser des contraintes plus avancées et créer une classe pour envelopper la logique:

```
# 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
```

Un formulaire, plusieurs boutons de soumission

Vous pouvez également utiliser la valeur des balises submit d'un formulaire comme contrainte pour acheminer une action différente. Si vous avez un formulaire avec plusieurs boutons d'envoi (par exemple "preview" et "submit"), vous pouvez capturer cette contrainte directement dans vos `routes.rb` , au lieu d'écrire du javascript pour changer l'URL de destination du formulaire. Par exemple, avec le gem commit_param_routing, vous pouvez tirer parti de rails `submit_tag`

Rails `submit_tag` premier paramètre `submit_tag` vous permet de modifier la valeur de votre paramètre de validation de formulaire

```
# 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
```

Itinéraires de portée

Rails propose plusieurs méthodes pour organiser vos itinéraires.

Portée par URL :

```
scope 'admin' do
  get 'dashboard', to: 'administration#dashboard'
  resources 'employees'
end
```

Cela génère les routes suivantes

```
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'
```

Du côté du serveur, il peut être plus judicieux de conserver certaines vues dans un sous-dossier différent, afin de séparer les vues d'administration des vues des utilisateurs.

Portée par module

```
scope module: :admin do
  get 'dashboard', to: 'administration#dashboard'
end
```

`module` recherche les fichiers du contrôleur sous le sous-dossier du nom donné

```
get       '/dashboard',          to: 'admin/administration#dashboard'
```

Vous pouvez renommer le préfixe des aides de chemin en ajoutant un paramètre `as`

```
scope 'admin', as: :administration do
  get 'dashboard'
end

# => administration_dashboard_path
```

Rails fournit un moyen pratique de faire tout ce qui précède, en utilisant la méthode d' `namespace` . Les déclarations suivantes sont équivalentes

```
namespace :admin do
end

scope 'admin', module: :admin, as: :admin
```

Portée par contrôleur

```
scope controller: :management do
  get 'dashboard'
  get 'performance'
end
```

Cela génère ces routes

```
get       '/dashboard',          to: 'management#dashboard'
get       '/performance',        to: 'management#performance'
```

Nidification peu profonde

Les itinéraires de ressources acceptent une option `:shallow` qui permet de raccourcir les URL lorsque cela est possible. Les ressources ne doivent pas être imbriquées à plus d'un niveau. Une façon d'éviter cela est de créer des routes peu profondes. L'objectif est de laisser de côté les segments d'URL de la collection parent où ils ne sont pas nécessaires. Le résultat final est que les seules routes imbriquées générées sont pour `:index` `:create` et `:new` actions. Les autres sont conservés dans leur propre contexte d'URL peu profond. Il existe deux options pour la portée des itinéraires peu profonds personnalisés:

- **: shallow_path** : préfixes des chemins de membre avec un paramètre spécifié

  ```
  scope shallow_path: "sekret" do
    resources :articles do
      resources :comments, shallow: true
    end
  end
  ```

- **: shallow_prefix** : ajoute les paramètres spécifiés aux helpers nommés

  ```
  scope shallow_prefix: "sekret" do
    resources :articles do
      resources :comments, shallow: true
    end
  end
  ```

Nous pouvons également illustrer des routes `shallow` en:

```
resources :auctions, shallow: true do
  resources :bids do
   resources :comments
  end
end
```

alternativement codé comme suit (si vous êtes bloqué):

```
resources :auctions do
 shallow do
   resources :bids do
     resources :comments
   end
 end
end
```

Les itinéraires résultants sont:

Préfixe	Verbe	Modèle d'URI
bid_comments	OBTENIR	/bids/:bid_id/comments(.:format)
	POSTER	/bids/:bid_id/comments(.:format)
new_bid_comment	OBTENIR	/bids/:bid_id/comments/new(.:format)
edit_comment	OBTENIR	/comments/:id/edit(.:format)
commentaire	OBTENIR	/comments/:id(.:format)
	PIÈCE	/comments/:id(.:format)
	METTRE	/comments/:id(.:format)
	EFFACER	/comments/:id(.:format)
auction_bids	OBTENIR	/auctions/:auction_id/bids(.:format)
	POSTER	/auctions/:auction_id/bids(.:format)
new_auction_bid	OBTENIR	/auctions/:auction_id/bids/new(.:format)
edit_bid	OBTENIR	/bids/:id/edit(.:format)
offre	OBTENIR	/bids/:id(.:format)
	PIÈCE	/bids/:id(.:format)
	METTRE	/bids/:id(.:format)

Préfixe	Verbe	Modèle d'URI
	EFFACER	/bids/:id(.:format)
les enchères	OBTENIR	/auctions(.:format)
	POSTER	/auctions(.:format)
new_auction	OBTENIR	/auctions/new(.:format)
edit_auction	OBTENIR	/auctions/:id/edit(.:format)
enchères	OBTENIR	/auctions/:id(.:format)
	PIÈCE	/auctions/:id(.:format)
	METTRE	/auctions/:id(.:format)
	EFFACER	/auctions/:id(.:format)

Si vous analysez les routes générées avec soin, vous remarquerez que les parties imbriquées de l'URL ne sont incluses que lorsqu'elles sont nécessaires pour déterminer les données à afficher.

Les soucis

Pour éviter la répétition dans les routes imbriquées, les préoccupations offrent un excellent moyen de partager des ressources communes réutilisables. Pour créer un problème, utilisez la méthode `concern` dans le fichier `routes.rb` . La méthode attend un symbole et un bloc:

```
concern :commentable do
  resources :comments
end
```

Tout en ne créant aucune route elle-même, ce code permet d'utiliser l'attribut `:concerns` sur une ressource. L'exemple le plus simple serait:

```
resource :page, concerns: :commentable
```

La ressource imbriquée équivalente ressemblerait à ceci:

```
resource :page do
  resource :comments
end
```

Cela permettrait, par exemple, de créer les itinéraires suivants:

```
/pages/#{page_id}/comments
/pages/#{page_id}/comments/#{comment_id}
```

Pour que les préoccupations soient significatives, il doit y avoir plusieurs ressources qui utilisent le

problème. Des ressources supplémentaires peuvent utiliser l'une des syntaxes suivantes pour appeler le problème:

```
resource :post, concerns: %i(commentable)
resource :blog do
  concerns :commentable
end
```

Redirection

Vous pouvez effectuer la redirection dans les itinéraires Rails comme suit:

4.0

```
get '/stories', to: redirect('/posts')
```

4.0

```
match "/abc" => redirect("http://example.com/abc")
```

Vous pouvez également rediriger tous les itinéraires inconnus vers un chemin donné:

4.0

```
match '*path' => redirect('/'), via: :get
# or
get '*path' => redirect('/')
```

4.0

```
match '*path' => redirect('/')
```

Itinéraires de membre et de collection

La définition d'un bloc membre dans une ressource crée un itinéraire pouvant agir sur un membre individuel de cet itinéraire basé sur une ressource:

```
resources :posts do
  member do
    get 'preview'
  end
end
```

Cela génère l'itinéraire de membre suivant:

```
get '/posts/:id/preview', to: 'posts#preview'
# preview_post_path
```

Les itinéraires de collecte permettent de créer des itinéraires pouvant agir sur une collection d'objets ressource:

```
resources :posts do
  collection do
    get 'search'
  end
end
```

Cela génère l'itinéraire de collecte suivant:

```
get '/posts/search', to: 'posts#search'
# search_posts_path
```

Une syntaxe alternative:

```
resources :posts do
  get 'preview', on: :member
  get 'search',  on: :collection
end
```

Paramètres d'URL avec un point

Si vous souhaitez prendre en charge un paramètre d'URL plus complexe qu'un numéro d'identification, vous pouvez rencontrer des problèmes avec l'analyseur si la valeur contient un point. Tout ce qui suit une période sera considéré comme un format (c.-à-d. Json, xml).

Vous pouvez contourner cette limitation en utilisant une contrainte pour *élargir* l'entrée acceptée.

Par exemple, si vous souhaitez référencer un enregistrement d'utilisateur par adresse électronique dans l'URL:

```
resources :users, constraints: { id: /.*/ }
```

Route racine

Vous pouvez ajouter un itinéraire de page d'accueil à votre application avec la méthode `root` .

```
# config/routes.rb
Rails.application.routes.draw do
  root "application#index"
  # equivalent to:
  # get "/", "application#index"
end

# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  def index
    render "homepage"
  end
end
```

Et dans les terminaux, les `rake routes` (`rake routes rails routes` dans Rails 5) produiront:

```
root     GET    /         application#index
```

Étant donné que la page d'accueil est généralement l'itinéraire le plus important et que les itinéraires sont classés par ordre de priorité dans l'ordre dans lequel ils apparaissent, l'itinéraire `root` doit généralement être le premier de votre fichier de routes.

Actions RESTful supplémentaires

```
resources :photos do
  member do
    get 'preview'
  end
  collection do
    get 'dashboard'
  end
end
```

Cela crée les routes suivantes **en plus des 7 routes RESTful par défaut** :

```
get       '/photos/:id/preview',          to: 'photos#preview'
get       '/photos/dashboards',           to: 'photos#dashboard'
```

Si vous voulez faire cela pour des lignes simples, vous pouvez utiliser:

```
resources :photos do
  get 'preview',   on: :member
  get 'dashboard', on: :collection
end
```

Vous pouvez également ajouter une action au chemin `/new` :

```
resources :photos do
  get 'preview', on: :new
end
```

Qui va créer:

```
get       '/photos/new/preview',          to: 'photos#preview'
```

Soyez attentif lorsque vous ajoutez des actions à vos routes RESTful, vous manquez probablement une autre ressource!

Portée disponible locale

Si votre application est disponible dans différentes langues, vous affichez généralement les paramètres régionaux actuels dans l'URL.

```
scope '/(:locale)', locale: /#{I18n.available_locales.join('|')}/ do
    root 'example#root'
    # other routes
```

```
end
```

Votre racine sera accessible via les paramètres régionaux définis dans `I18n.available_locales` .

Monter une autre application

le montage est utilisé pour monter une autre application (essentiellement une application en rack) ou des moteurs de rails à utiliser dans l'application en cours

syntaxe:

```
mount SomeRackApp, at: "some_route"
```

Vous pouvez maintenant accéder aux applications montées ci-dessus en utilisant l'assistant de route `some_rack_app_path` ou `some_rack_app_url` .

Mais si vous voulez renommer ce nom d'assistant, vous pouvez le faire en tant que:

```
mount SomeRackApp, at: "some_route", as: :myapp
```

Cela générera les `myapp_path` et `myapp_url` qui peuvent être utilisés pour naviguer dans cette application montée.

Itinéraires de redirections et de caractères génériques

Si vous souhaitez fournir une URL hors de la convenance pour votre utilisateur, mais la mapper directement à une autre que vous utilisez déjà. Utilisez une redirection:

```
# config/routes.rb
TestApp::Application.routes.draw do
  get 'courses/:course_name' => redirect('/courses/%{course_name}/lessons'), :as => "course"
end
```

Eh bien, ça a été intéressant rapidement. Le principe de base consiste à utiliser la méthode `#redirect` pour envoyer une route à une autre. Si votre itinéraire est assez simple, c'est une méthode très simple. Mais si vous voulez aussi envoyer les paramètres d'origine, vous devez faire un peu de gymnastique en capturant le paramètre dans `%{here}` . Notez les guillemets simples autour de tout.

Dans l'exemple ci-dessus, nous avons également renommé la route pour plus de commodité en utilisant un alias avec le paramètre: as. Cela nous permet d'utiliser ce nom dans des méthodes telles que les aides #_path. Encore une fois, testez vos `$ rake routes` avec des questions.

Diviser les itinéraires en plusieurs fichiers

Si votre fichier de routes est extrêmement volumineux, vous pouvez placer vos itinéraires dans plusieurs fichiers et inclure chacun des fichiers avec la méthode `require_relative` de Ruby:

`config/routes.rb` :

```
YourAppName::Application.routes.draw do
  require_relative 'routes/admin_routes'
  require_relative 'routes/sidekiq_routes'
  require_relative 'routes/api_routes'
  require_relative 'routes/your_app_routes'
end
```

`config/routes/api_routes.rb` :

```
YourAppName::Application.routes.draw do
  namespace :api do
    # ...
  end
end
```

Routes imbriquées

Si vous souhaitez ajouter des routes imbriquées, vous pouvez écrire le code suivant dans le fichier `routes.rb` .

```
resources :admins do
  resources :employees
end
```

Cela générera les itinéraires suivants:

```
     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
```

Lire Le routage en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/307/le-routage

Chapitre 41: Les frameworks Rails au fil des ans

Introduction

Lorsque vous êtes novice dans Rails et que vous travaillez sur des applications Rails héritées, il peut être difficile de comprendre quel framework a été introduit lorsque. Ce sujet est conçu pour être la liste *définitive* de tous les frameworks sur les versions Rails.

Examples

Comment trouver les frameworks disponibles dans la version actuelle de Rails?

Utilisez le

```
config.frameworks
```

option pour obtenir un tableau de `Symbol` s représentant chaque framework.

Versions Rails dans Rails 1.x

- ActionMailer
- ActionPack
- ActionWebService
- ActiveRecord
- ActiveSupport
- Chemins de fer

Framework Rails dans Rails 2.x

- ActionMailer
- ActionPack
- ActiveRecord
- ActiveResource (*ActiveWebService a été remplacé par ActiveResource, avec cela, Rails a été déplacé de SOAP vers REST par défaut*)
- ActiveSupport
- Chemins de fer

Framework Rails dans Rails 3.x

- ActionMailer
- ActionPack
- ActiveModel

- ActiveRecord
- ActiveResource
- ActiveSupport
- Chemins de fer

Lire Les frameworks Rails au fil des ans en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/8107/les-frameworks-rails-au-fil-des-ans

Chapitre 42: Migrations ActiveRecord

Paramètres

Type de colonne	La description
`:primary_key`	Clé primaire
`:string`	Type de données de chaîne plus court. Permet l'option `limit` pour le nombre maximum de caractères.
`:text`	Plus grande quantité de texte Permet l'option `limit` pour le nombre maximum d'octets.
`:integer`	Entier. Permet l'option `limit` pour le nombre maximum d'octets.
`:bigint`	Entier plus grand
`:float`	Flotte
`:decimal`	Nombre décimal avec précision variable. Permet `precision` options de `precision` et d' `scale` .
`:numeric`	Permet `precision` options de `precision` et d' `scale` .
`:datetime`	Objet DateTime pour les dates / heures.
`:time`	Objet de temps pour les temps.
`:date`	Objet date pour les dates.
`:binary`	Données binaires. Permet l'option `limit` pour le nombre maximum d'octets.
`:boolean`	Booléen

Remarques

- La plupart des fichiers de migration sont `db/migrate/` dans `db/migrate/` répertoire `db/migrate/` . Ils sont identifiés par un horodatage UTC au début de leur nom de fichier: `YYYYMMDDHHMMSS_create_products.rb` .
- La commande de `rails generate` peut être raccourcie en `rails g` .
- Si un `:type` n'est pas transmis à un champ, sa valeur par défaut est une chaîne.

Examples

Exécuter une migration spécifique

Pour exécuter une migration spécifique vers le haut ou le bas, utilisez `db:migrate:up` ou `db:migrate:down` .

Mettre en place une migration spécifique:

5.0

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

5.0

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

En bas d'une migration spécifique:

5.0

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

5.0

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

Le numéro de version dans les commandes ci-dessus est le préfixe numérique dans le nom de fichier de la migration. Par exemple, pour migrer vers la migration `20160515085959_add_name_to_users.rb` , vous utiliseriez `20160515085959` comme numéro de version.

Créer une table de jointure

Pour créer une table de jointure entre les `students` et les `courses` , exécutez la commande suivante:

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

Cela générera la migration suivante:

```
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
```

Exécution de migrations dans différents environnements

Pour exécuter des migrations dans l'environnement de `test` , exécutez cette commande shell:

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

5.0

À partir de Rails 5.0, vous pouvez utiliser des `rails` au lieu de `rake` :

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

Ajouter une nouvelle colonne à une table

Pour ajouter un nouveau `name` colonne à la table `users` , exécutez la commande:

```
rails generate migration AddNameToUsers name
```

Cela génère la migration suivante:

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

Lorsque le nom de la migration est de la forme `AddXXXToTABLE_NAME` suivi d'une liste de colonnes avec des types de données, la migration générée contiendra les instructions `add_column` appropriées.

Ajouter une nouvelle colonne avec un index

Pour ajouter un nouvel `email` colonne *indexée* à la table `users` , exécutez la commande:

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

Cela générera la migration suivante:

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

Supprimer une colonne existante d'une table

Pour supprimer un `name` de colonne existant de la table `users` , exécutez la commande:

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

Cela générera la migration suivante:

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

Lorsque le nom de la migration est de la forme `RemoveXXXFromYYY` suivi d'une liste de colonnes avec des types de données, la migration générée contiendra les instructions `remove_column` appropriées.

Bien qu'il ne soit pas nécessaire de spécifier le type de données (par exemple `:string`) en tant que paramètre de `remove_column` , il est fortement recommandé. Si le type de données n'est *pas* spécifié, la migration ne sera pas réversible.

Ajouter une colonne de référence à une table

Pour ajouter une référence à une `team` à la table d' `users` , exécutez cette commande:

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

Cela génère la migration suivante:

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

Cette migration créera une colonne `team_id` dans la table `users` .

Si vous souhaitez ajouter un `index` approprié et `foreign_key` sur la colonne ajoutée, modifiez la commande pour que les `rails generate migration AddTeamRefToUsers team:references:index` . Cela générera la migration suivante:

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

Si vous souhaitez nommer votre colonne de référence autre que celle générée automatiquement par Rails, ajoutez ce qui suit à votre migration: (Par exemple, vous pouvez appeler l' `User` qui a créé la `Post` tant `Author` dans la table des `Post`)

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

Créer une nouvelle table

Pour créer une nouvelle table d' `users` avec le `name` et le `salary` colonnes, exécutez la commande:

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

Cela générera la migration suivante:

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

Lorsque le nom de la migration est de la forme `CreateXXX` suivi d'une liste de colonnes avec des types de données, une migration sera générée pour créer la table `XXX` avec les colonnes répertoriées.

Ajout de plusieurs colonnes à une table

Pour ajouter plusieurs colonnes à une table, séparez les `field:type` paires avec des espaces lorsque vous utilisez des `rails generate migration` commande de `rails generate migration` .

La syntaxe générale est la suivante:

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

Par exemple, les éléments suivants ajouteront les champs `name` , `salary` et `email` au tableau des `users` :

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

Qui génère la migration suivante:

```
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
```

Exécution de migrations

Exécuter la commande:

5.0

```
rake db:migrate
```

5.0

```
rails db:migrate
```

La spécification de la version cible exécutera les migrations requises (haut, bas, modification) jusqu'à ce qu'elle atteigne la version spécifiée. Ici, le `version number` est le préfixe numérique du nom de fichier de la migration.

5.0

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

5.0

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

Migrations de restauration

Pour `rollback` la dernière migration, soit en annulant la méthode de `change` , soit en exécutant la méthode `down` . Exécuter la commande:

5.0

```
rake db:rollback
```

5.0

```
rails db:rollback
```

Rétrograder les 3 dernières migrations

5.0

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

5.0

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

`STEP` fournit le nombre de migrations à restaurer.

Annuler toutes les migrations

5.0

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

5.0

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

Tables à langer

Si vous avez créé une table avec un schéma incorrect, le moyen le plus simple de modifier les colonnes et leurs propriétés est `change_table` . Consultez l'exemple suivant:

```
change_table :orders do |t|
  t.remove :ordered_at # removes column ordered_at
  t.string :skew_number # adds a new column
  t.index  :skew_number #creates an index
  t.rename :location, :state #renames location column to state
end
```

La migration ci-dessus modifie les `orders` une table. Voici une description ligne par ligne des modifications:

1. `t.remove :ordered_at` supprime la `ordered_at` des `orders` de la table.
2. `t.string :skew_number` ajoute une nouvelle colonne de type chaîne nommée `skew_number` dans la table `orders` .
3. `t.index :skew_number` ajoute un index sur la colonne `skew_number` de la table `orders` .
4. `t.rename :location, :state` renomme la colonne d' `location` dans la table des `orders` pour `state` .

Ajouter une colonne unique à une table

Pour ajouter un nouvel `email` colonne *unique* aux `users` , exécutez la commande suivante:

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

Cela créera la migration suivante:

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

Changer le type d'une colonne existante

Pour modifier une colonne existante dans Rails avec une migration, exécutez la commande suivante:

```
rails g migration change_column_in_table
```

Cela créera un nouveau fichier de migration dans le répertoire `db/migration` (s'il n'existe pas déjà), qui contiendra le fichier avec le préfixe timestamp et le nom du fichier de migration qui contient le contenu ci-dessous:

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

Guide Rails - Modification des colonnes

Une méthode plus longue mais plus sûre

Le code ci-dessus empêche l'utilisateur de revenir en arrière sur la migration. Vous pouvez éviter ce problème en divisant le `change` méthode dans différents `up` le `down up` et `down` le `down` des méthodes:

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

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

Refaire les migrations

Vous pouvez restaurer, puis migrer à nouveau à l'aide de la commande `redo` . Il s'agit essentiellement d'un raccourci combinant des tâches de `rollback` et de `migrate` .

Exécuter la commande:

5.0

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

5.0

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

Vous pouvez utiliser le paramètre `STEP` pour revenir sur plusieurs versions.

Par exemple, pour revenir en arrière de 3 migrations:

5.0

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

5.0

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

Ajouter une colonne avec une valeur par défaut

L'exemple suivant ajoute une colonne `admin` à la table `users` et attribue à cette colonne la valeur

par défaut `false` .

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

Les migrations avec des valeurs par défaut peuvent prendre beaucoup de temps dans les grandes tables avec, par exemple, PostgreSQL. En effet, chaque ligne devra être mise à jour avec la valeur par défaut de la colonne nouvellement ajoutée. Pour contourner ce problème (et réduire les temps d'arrêt pendant les déploiements), vous pouvez diviser votre migration en trois étapes:

1. Ajouter un `add_column` -migration similaire à celui ci-dessus, mais ne pas définir par défaut
2. Déployez et mettez à jour la colonne dans une tâche rake ou sur la console pendant que votre application est en cours d'exécution. Assurez-vous que votre application écrit déjà des données dans cette colonne pour les lignes nouvelles / mises à jour.
3. Ajouter une autre migration `change_column` , qui modifie ensuite la valeur par défaut de cette colonne à la valeur par défaut souhaitée

Interdire les valeurs nulles

Pour interdire `null` valeurs `null` dans vos colonnes de table, ajoutez le paramètre `:null` à votre migration, comme ceci:

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

Vérification du statut de migration

Nous pouvons vérifier l'état des migrations en exécutant

3.0 5.0

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

5.0

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

La sortie ressemblera à ceci:

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

Dans le champ d'état, `up` signifie la migration a été exécuté et `down` le `down` signifie que nous devons exécuter la migration.

Créer une colonne hstore

`Hstore` colonnes `Hstore` peuvent être utiles pour stocker des paramètres. Ils sont disponibles dans les bases de données PostgreSQL après avoir activé l'extension.

```
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
```

Ajouter une référence personnelle

Une référence automatique peut être utile pour construire un arbre hiérarchique. Cela peut être réalisé avec `add_reference` dans une migration.

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

La colonne de clé étrangère sera `pages_id` . Si vous souhaitez décider du nom de la colonne de clé étrangère, vous devez d'abord créer la colonne et ajouter la référence après.

```
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
```

Créer une colonne de tableau

Une colonne de `array` est prise en charge par PostgreSQL. Rails convertira automatiquement un tableau PostgreSQL en un tableau Ruby, et inversement.

Créez une table avec une colonne de `array` :

```
create_table :products do |t|
  t.string :name
  t.text :colors, array: true, default: []
end
```

Ajoutez une colonne de `array` à une table existante:

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

Ajoutez un index pour une colonne de `array` :

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

Ajout d'une contrainte NOT NULL aux données existantes

Supposons que vous souhaitiez ajouter une clé étrangère `company_id` à la table `users` et que vous souhaitiez avoir une contrainte `NOT NULL` . Si vous avez déjà des données dans les `users` , vous devrez le faire en plusieurs étapes.

```
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
```

Lire Migrations ActiveRecord en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/1087/migrations-activerecord

Chapitre 43: Mise à niveau des rails

Examples

Mise à niveau de Rails 4.2 vers Rails 5.0

Remarque: Avant de mettre à niveau votre application Rails, veillez à toujours enregistrer votre code sur un système de contrôle de version, tel que Git.

Pour mettre à niveau Rails 4.2 vers Rails 5.0, vous devez utiliser Ruby 2.2.2 ou une version plus récente. Après la mise à niveau de votre version de Ruby si nécessaire, accédez à votre Gemfile et changez la ligne:

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

à:

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

et sur la ligne de commande exécutée:

```
$ bundle update
```

Maintenant, lancez la tâche de mise à jour en utilisant la commande:

```
$ rake rails:update
```

Cela vous aidera à mettre à jour les fichiers de configuration. Vous serez invité à écraser les fichiers et vous avez plusieurs options à saisir:

- Y - oui, écraser
- n - non, ne pas écraser
- a - tout, écraser ceci et tous les autres
- q - quitte, abandonne
- d - diff, montre les différences entre l'ancien et le nouveau
- h - aide

En règle générale, vous devez vérifier les différences entre les anciens et les nouveaux fichiers pour vous assurer de ne pas recevoir de modifications indésirables.

Les modèles Rails 5.0 `ActiveRecord` héritent d' `ApplicationRecord` plutôt que d' `ActiveRecord::Base` . `ApplicationRecord` est la super-classe pour tous les modèles, similaire à la façon dont `ApplicationController` est la super-classe pour les contrôleurs. Pour tenir compte de cette nouvelle manière de gérer les modèles, vous devez créer un fichier dans votre dossier `app/models/` nommé `application_record.rb` , puis modifier le contenu de ce fichier pour qu'il soit:

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

Rails 5.0 gère également les rappels légèrement différents. Les rappels renvoyés par `false` n'interrompront pas la chaîne de rappel, ce qui signifie que les rappels suivants seront toujours exécutés, contrairement à Rails 4.2. Lorsque vous effectuez une mise à niveau, le comportement de Rails 4.2 restera, mais vous pouvez passer au comportement Rails 5.0 en ajoutant:

```
ActiveSupport.halt_callback_chains_on_return_false = false
```

dans le fichier `config/application.rb` . Vous pouvez explicitement arrêter la chaîne de rappel en appelant `throw(:abort)` .

Dans Rails 5.0, `ActiveJob` héritera d' `ApplicationJob` , plutôt `ActiveJob::Base` comme dans Rails 4.2. Pour mettre à niveau vers Rails 5.0, créez un fichier appelé `application_job.rb` dans le dossier `app/jobs/` . Modifier le contenu de ce fichier pour qu'il soit:

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

Ensuite, vous devez modifier tous vos travaux pour qu'ils héritent d' `ApplicationJob` plutôt que d' `ActiveJob::Base` .

L'un des autres changements les plus importants de Rails 5.0 ne nécessite aucun changement de code, mais changera la façon dont vous utilisez la ligne de commande avec vos applications Rails. Vous pourrez utiliser `bin/rails` , ou simplement des `rails` , pour exécuter des tâches et des tests. Par exemple, au lieu d'utiliser `$ rake db:migrate` , vous pouvez maintenant faire `$ rails db:migrate` . Si vous exécutez `$ bin/rails` , vous pouvez afficher toutes les commandes disponibles. Notez que la plupart des tâches qui peuvent maintenant être exécutées avec `bin/rails` fonctionnent toujours avec `rake` .

Lire Mise à niveau des rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3496/mise-a-niveau-des-rails

Chapitre 44: Mise en cache

Examples

Poupée Russe Caching

Vous souhaiterez peut-être imbriquer des fragments en cache dans d'autres fragments mis en cache. Cela s'appelle `Russian doll caching` .

L'avantage de `Russian doll caching` est que si un seul produit est mis à jour, tous les autres fragments internes peuvent être réutilisés lors de la régénération du fragment externe.

Comme expliqué dans la section précédente, un fichier mis en cache expirera si la valeur de `updated_at` change pour un enregistrement dont dépend directement le fichier en cache. Cependant, cela n'expire aucun cache dans lequel le fragment est imbriqué.

Par exemple, prenez la vue suivante:

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

Qui à son tour rend cette vue:

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

Si un attribut de jeu est modifié, la valeur `updated_at` sera définie sur l'heure actuelle, expirant ainsi le cache.

Cependant, parce que `updated_at` ne sera pas modifié pour l'objet produit, ce cache ne sera pas expiré et votre application servira les données obsolètes. Pour résoudre ce problème, nous lions les modèles avec la méthode tactile:

```
class Product < ApplicationRecord
  has_many :games
end

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

Mise en cache SQL

La mise en cache des requêtes est une fonctionnalité `Rails` qui met en cache l'ensemble de résultats renvoyé par chaque requête. Si `Rails` rencontre à nouveau la même requête pour cette requête, il utilisera le jeu de résultats mis en cache au lieu d'exécuter à nouveau la requête sur la

base de données.

Par exemple:

```
class ProductsController < ApplicationController

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

    ...

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

end
```

La deuxième fois que la même requête est exécutée sur la base de données, elle ne touchera pas réellement la base de données. La première fois que le résultat est renvoyé par la requête, il est stocké dans le cache de la requête (en mémoire) et la deuxième fois dans la mémoire.

Cependant, il est important de noter que les caches de requêtes sont créés au début d'une action et détruits à la fin de cette action et ne persistent donc que pendant la durée de l'action. Si vous souhaitez stocker les résultats de la requête de manière plus persistante, vous pouvez utiliser la mise en cache de bas niveau.

Mise en cache des fragments

`Rails.cache` , fourni par ActiveSupport, peut être utilisé pour mettre en cache tout objet Ruby sérialisable dans les requêtes.

Pour extraire une valeur du cache pour une clé donnée, utilisez `cache.read` :

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

Utilisez `cache.write` pour écrire une valeur dans le cache:

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

Vous pouvez également utiliser `cache.fetch` pour lire une valeur du cache et éventuellement écrire une valeur par défaut s'il n'y a pas de valeur:

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

La valeur de retour du bloc passé sera attribuée au cache sous la clé donnée, puis renvoyée.

Vous pouvez également spécifier une expiration du cache:

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

Mise en cache de page

Vous pouvez utiliser le joyau ActionPack page_caching pour mettre en cache des pages individuelles. Cela stocke le résultat d'une requête dynamique en tant que fichier HTML statique, qui est utilisé à la place de la demande dynamique lors des requêtes suivantes. Le fichier README contient des instructions de configuration complètes. Une fois configuré, utilisez la méthode de classe `caches_page` dans un contrôleur pour mettre en cache le résultat d'une action:

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

Utilisez `expire_page` pour forcer l'expiration du cache en supprimant le fichier HTML stocké:

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

  def index
    @users = User.all
  end

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

La syntaxe de `expire_page` imite celle de `url_for` et des amis.

Mise en cache HTTP

Rails> = 3 est livré avec des capacités de mise en cache HTTP prêtes à l'emploi. Cela utilise les en `Cache-Control` têtes `Cache-Control` et `ETag` pour contrôler la durée pendant laquelle un client ou un intermédiaire (tel qu'un CDN) peut mettre en cache une page.

Dans une action de contrôleur, utilisez `expires_in` pour définir la longueur de la mise en cache pour cette action:

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

Utilisez `expires_now` pour forcer l'expiration immédiate d'une ressource en cache sur tout client ou intermédiaire en visite:

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

Mise en cache des actions

Comme pour la mise en cache des pages, la mise en cache des actions met en cache toute la page. La différence est que la requête frappe la pile Rails avant que les filtres ne soient exécutés avant que le cache ne soit servi. Il est extrait de Rails à la gem actionpack-action_caching .

Un exemple courant est la mise en cache d'une action nécessitant une authentification:

```
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
```

Les options incluent `:expires_in` , un `:expires_in` personnalisé `:cache_path` (pour les actions avec plusieurs routes devant être mises en cache différemment) et `:if` / `:unless` pour contrôler quand l'action doit être mise en cache.

```
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
```

Lorsque la mise en page a un contenu dynamique, ne mettez en cache que le contenu de l’action en passant la `layout: false` en `layout: false` .

Lire Mise en cache en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2833/mise-en-cache

Chapitre 45: Modifier le fuseau horaire par défaut

Remarques

> **config.active_record.default_timezone** détermine s'il faut utiliser Time.local (si défini sur: local) ou Time.utc (si défini sur: utc) lors de l'extraction des dates et des heures de la base de données. La valeur par défaut est: utc.
> http://guides.rubyonrails.org/configuring.html

Si vous souhaitez modifier le fuseau horaire **Rails** , mais continuer à enregistrer **Active Record** dans la base de données en **UTC** , utilisez

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

Si vous souhaitez modifier le fuseau horaire **Rails ET** avoir les durées de stockage **Active Record** dans ce fuseau horaire, utilisez

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

Attention : vous devriez vraiment réfléchir à deux fois, voire à trois fois, avant de sauvegarder les temps dans la base de données au format non-UTC.

> **Remarque**
> N'oubliez pas de redémarrer votre serveur Rails après avoir modifié `application.rb` .

Rappelez-vous que `config.active_record.default_timezone` ne peut prendre que deux valeurs

- **: local** (convertit dans le fuseau horaire défini dans `config.time_zone`)
- **: utc** (convertit en UTC)

Voici comment trouver tous les fuseaux horaires disponibles

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

Examples

Changer le fuseau horaire Rails, mais continuer à enregistrer Active Record

dans la base de données en UTC

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

Modifier le fuseau horaire Rails ET avoir les durées de stockage Active Record dans ce fuseau horaire

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

Lire Modifier le fuseau horaire par défaut en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3367/modifier-le-fuseau-horaire-par-defaut

Chapitre 46: Modifier un environnement d'application Rails par défaut

Introduction

Cela discutera de la manière de changer l'environnement, de sorte que lorsque quelqu'un tape des `rails s` ils démarrent non pas en développement, mais dans l'environnement souhaité.

Examples

Fonctionnement sur une machine locale

Normalement, lorsque l'environnement des rails est exécuté en tapant. Cela exécute juste l'environnement par défaut qui est généralement le `development`

```
rails s
```

L'environnement spécifique peut être sélectionné en utilisant le drapeau `-e` par exemple:

```
rails s -e test
```

Qui exécutera l'environnement de test.

L'environnement par défaut peut être modifié dans le terminal en éditant le fichier `~/.bashrc` et en ajoutant la ligne suivante:

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

En cours d'exécution sur un serveur

Si vous utilisez un serveur distant qui utilise Passenger, changez apache.conf avec l'environnement que vous souhaitez utiliser. Par exemple, vous voyez la `RailsEnv production` .

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

Lire Modifier un environnement d'application Rails par défaut en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/9915/modifier-un-environnement-d-application-rails-par-defaut

Chapitre 47: Mongoïde

Examples

Installation

Tout d'abord, ajoutez `Mongoid` à votre `Gemfile` :

```
gem "mongoid", "~> 4.0.0"
```

puis exécutez l' `bundle install` . Ou lancez simplement:

```
$ gem install mongoid
```

Après l'installation, exécutez le générateur pour créer le fichier de configuration:

```
$ rails g mongoid:config
```

qui va créer le fichier `(myapp)/config/mongoid.yml` .

Créer un modèle

Créez un modèle (appelons-le `User`) en exécutant:

```
$ rails g model User
```

qui va générer le fichier `app/models/user.rb` :

```
class User
  include Mongoid::Document

end
```

C'est tout ce dont vous avez besoin pour avoir un modèle (mais rien d'autre qu'un champ d' `id`). Contrairement à `ActiveRecord` , il n'y a pas de fichiers de migration. Toutes les informations de base de données pour le modèle sont contenues dans le fichier modèle.

Les horodatages ne sont pas automatiquement inclus dans votre modèle lorsque vous le générez. Pour ajouter `created_at` et `updated_at` à votre modèle, ajoutez

```
include Mongoid::Timestamps
```

à votre modèle ci-dessous `include Mongoid::Document` comme ça:

```
class User
  include Mongoid::Document
```

```
  include Mongoid::Timestamps

end
```

Des champs

Selon la documentation Mongoid , il existe 16 types de champs valides:

- Tableau
- BigDecimal
- Booléen
- Rendez-vous amoureux
- DateTime
- Flotte
- Hacher
- Entier
- BSON :: ObjectId
- BSON :: Binaire
- Gamme
- Regexp
- Chaîne
- symbole
- Temps
- TimeWithZone

Pour ajouter un champ (appelons-le `name` et faites-en une `String`), ajoutez-le à votre fichier modèle:

```
field :name, type: String
```

Pour définir une valeur par défaut, il suffit de transmettre l'option `default` :

```
field :name, type: String, default: ""
```

Associations Classiques

Mongoid permet les associations classiques `ActiveRecord` :

- One-to-one: `has_one` / `belongs_to`
- Un-à-plusieurs: `has_many` / `belongs_to`
- Plusieurs à plusieurs: `has_and_belongs_to_many`

Pour ajouter une association (disons l'utilisateur `has_many` posts), vous pouvez l'ajouter à votre fichier de modèle d' `User` :

```
has_many :posts
```

et ceci à votre fichier modèle `Post` :

```
belongs_to :user
```

Cela va ajouter un champ `user_id` dans votre modèle `Post` , ajouter une méthode `user` à votre classe `Post` et ajouter une méthode de `posts` à votre classe `User` .

Associations embarquées

Mongoid autorise les associations embarquées:

- One-to-one: `embeds_one` / `embedded_in`
- Un à plusieurs: `embeds_many` / `embedded_in`

Pour ajouter une association (disons l'utilisateur `embeds_many` adresses), ajoutez ceci à votre fichier `User` :

```
embeds_many :addresses
```

et ceci à votre fichier de modèle d' `Address` :

```
embedded_in :user
```

Cela incorporera l' `Address` dans votre modèle d' `User` , en ajoutant une méthode d' `addresses` à votre classe d' `User` .

Appels de base de données

Mongoid essaie d'avoir une syntaxe similaire à `ActiveRecord` quand c'est possible. Il prend en charge ces appels (et beaucoup plus)

```
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
```

Lire Mongoïde en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3071/mongoide

Chapitre 48: Moteur Rails - Rails Modulaires

Introduction

Aperçu rapide des moteurs Rails

Les moteurs sont de petites applications Rails pouvant être utilisées pour ajouter des fonctionnalités à l'application qui les héberge. La classe définissant une application Ruby on Rails est `Rails::Application` qui hérite en fait beaucoup de son comportement de `Rails::Engine` , la classe définissant un moteur. Nous pouvons dire qu'une application Rails régulière est simplement un moteur avec plus de fonctionnalités.

Syntaxe

- rails plugin nouveau my_module --montable

Examples

Créer une application modulaire

Commencer

D'abord, générons une nouvelle application Ruby on Rails:

```
rails new ModularTodo
```

L'étape suivante consiste à générer un moteur!

```
cd ModularTodo && rails plugin new todo --mountable
```

Nous allons également créer un dossier "moteurs" pour stocker les moteurs (même si nous en avons juste un!).

```
mkdir engines && mv todo ./engines
```

Les moteurs, tout comme les gemmes, sont livrés avec un fichier gemspec. Mettons des valeurs réelles pour éviter les avertissements.

```
 #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
```

Maintenant, nous devons ajouter le moteur Todo à l'application parent Gemfile.

```
#ModularTodo/Gemfile
#Other gems
gem 'todo', path: 'engines/todo'
```

Exécutons l' `bundle install` . Vous devriez voir ce qui suit dans la liste des gemmes:

```
Using todo 0.0.1 from source at engines/todo
```

Super, notre moteur Todo est chargé correctement! Avant de commencer à coder, nous avons une dernière chose à faire: monter le moteur Todo. Nous pouvons le faire dans le fichier routes.rb de l'application parent.

```
Rails.application.routes.draw do
  mount Todo::Engine => "/", as: 'todo'
end
```

Nous le montons à `/` mais nous pourrions aussi le rendre accessible à `/todo` . Comme nous n'avons qu'un seul module, `/` c'est bien.

Vous pouvez maintenant lancer votre serveur et le vérifier dans votre navigateur. Vous devriez voir la vue Rails par défaut car nous n'avons pas encore défini de contrôleurs / vues. Faisons ça maintenant!

Construire la liste Todo

Nous allons échafauder un modèle nommé `Task` dans le module Todo, mais pour migrer correctement la base de données de l'application parente, nous devons ajouter un petit initialiseur au fichier `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
```

Ça y est, maintenant que nous effectuons des migrations à partir de l'application parente, les migrations dans le moteur Todo seront également chargées.

Créons le modèle de `Task` . La commande d' `scaffold` doit être exécutée à partir du dossier du moteur.

```
cd engines/todo && rails g scaffold Task title:string content:text
```

Exécutez les migrations à partir du dossier parent:

```
rake db:migrate
```

Maintenant, il suffit de définir la route racine dans le moteur Todo:

```
#ModularTodo/engines/todo/config/routes.rb
Todo::Engine.routes.draw do
  resources :tasks
  root 'tasks#index'
end
```

Vous pouvez jouer avec, créer des tâches, les supprimer… Oh, attendez, la suppression ne fonctionne pas! Pourquoi?! Eh bien, il semblerait que JQuery ne soit pas chargé, ajoutons-le donc au fichier `application.js` à l'intérieur du moteur!

```
// ModularTodo/engines/todo/app/assets/javascripts/todo/application.js
//= require jquery
//= require jquery_ujs
//= require_tree .
```

Oui, maintenant nous pouvons détruire des tâches!

Lire Moteur Rails - Rails Modulaires en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/9080/moteur-rails---rails-modulaires

Chapitre 49: Motif de décorateur

Remarques

Le **motif Décorateur** vous permet d'ajouter ou de modifier le comportement des objets de manière situationnelle sans affecter l'objet de base.

Cela peut être réalisé en utilisant Ruby en utilisant le stdlib, ou via des gemmes populaires telles que Draper .

Examples

Décorer un modèle en utilisant SimpleDelegator

La plupart des développeurs Rails commencent par modifier leurs informations de modèle dans le modèle même:

```
<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>
```

Pour les modèles avec beaucoup de données, cela peut rapidement devenir encombrant et conduire à une logique de copier-coller d'un modèle à un autre.

Cet exemple utilise `SimpleDelegator` partir du stdlib.

Toutes les demandes `SimpleDelegator` à un objet `SimpleDelegator` sont transmises à l'objet parent par défaut. Vous pouvez remplacer n'importe quelle méthode par la logique de présentation ou ajouter de nouvelles méthodes spécifiques à cette vue.

`SimpleDelegator` propose deux méthodes: `__setobj__` pour définir quel objet est délégué et `__getobj__` pour obtenir cet objet.

```
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
```

Certains décorateurs comptent sur la magie pour configurer ce comportement, mais vous pouvez rendre plus évidente l'origine de la logique de présentation en initialisant l'objet sur la page.

```
<% user = UserDecorator.new(@user, self) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

En transmettant une référence à l'objet view dans le décorateur, nous pouvons toujours accéder à tous les assistants de la vue tout en construisant la logique de présentation sans avoir à l'inclure.

Maintenant, le modèle de vue ne concerne que l'insertion de données dans la page, et il est beaucoup plus clair.

Décorer un modèle en utilisant Draper

Draper associe automatiquement les modèles avec leurs décorateurs par convention.

```
# 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
```

Étant donné une variable `@user` contenant un objet ActiveRecord, vous pouvez accéder à votre décorateur en appelant `#decorate` sur `@user` ou en spécifiant la classe Draper si vous souhaitez être spécifique.

```
<% user = @user.decorate %><!-- OR -->
<% user = UserDecorator.decorate(@user) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

Lire Motif de décorateur en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/5694/motif-de-decorateur

Chapitre 50: Mots réservés

Introduction

Vous devriez être prudent en utilisant ces mots pour la variable, le nom du modèle, le nom de la méthode ou etc.

Examples

Liste de mots réservés

- ADDITIONAL_LOAD_PATHS
- ARGF
- ARGV
- ActionController
- ActionView
- ActiveRecord
- ArgumentError
- Tableau
- BasicSocket
- Référence
- Bignum
- Contraignant
- CGI
- Méthodes CGIM
- CROSS_COMPILING
- Classe
- ClassInheritableAttributes
- Comparable
- ConditionVariable
- Config
- Continuation
- DRb
- DRbIdConv
- DRbObject
- DRbUndumped
- Les données
- Rendez-vous amoureux
- DateTime
- Délégué
- Délégué
- Digérer
- Dir
- ENV

- EOFError
- ERB
- Enumerable
- Errno
- Exception
- FAUX
- FalseClass
- Fcntl
- Fichier
- FileList
- FileTask
- Test de fichier
- FileUtils
- Fixnum
- Flotte
- FloatDomainError
- GC
- Gemme
- GetoptLong
- Hacher
- IO
- IOError
- IPSocket
- IPsocket
- IndexError
- Inflecteur
- Entier
- Interrompre
- Noyau
- LN_SUPPORTED
- LoadError
- LocalJumpError
- Enregistreur
- Maréchal
- MatchData
- MatchingData
- Math
- Méthode
- Module
- Mutex
- Mysql
- MysqlError
- MysqlField
- MysqlRes
- NÉANT
- NameError

- NilClass
- NoMemoryError
- NoMethodError
- NoWrite
- Erreur non implémentée
- Numérique
- OPT_TABLE
- Objet
- ObjectSpace
- Observable
- Observateur
- PGError
- PGconn
- PGlarge
- PGresult
- PLATE-FORME
- PStore
- ParseDate
- Précision
- Proc
- Processus
- Queue
- RAKEVERSION
- DATE DE SORTIE
- RUBIS
- RUBY_PLATFORM
- RUBY_RELEASE_DATE
- RUBY_VERSION
- Grille
- Râteau
- RakeApp
- RakeFileUtils
- Gamme
- RangeError
- Rationnel
- Regexp
- RegexpError
- Demande
- Erreur d'exécution
- STDERR
- STDIN
- STDOUT
- ScanError
- Erreur de script
- Erreur de sécurité
- Signal

- SignalException
- SimpleDelegater
- SimpleDelegator
- Singleton
- SizedQueue
- Prise
- SocketError
- Erreur standard
- Chaîne
- StringScanner
- Struct
- symbole
- Erreur de syntaxe
- SystemCallError
- SystemExit
- SystemStackError
- Serveur TCPS
- TCPSocket
- TCPserver
- TCPsocket
- TOPLEVEL_BINDING
- VRAI
- Tâche
- Texte
- Fil
- ThreadError
- ThreadGroup
- Temps
- Transaction
- TrueClass
- Erreur-type
- UDPSocket
- UDPsocket
- Serveur UNIX
- UNIXSocket
- UNIXserver
- UNIXsocket
- UnboundMethod
- URL
- VERSION
- Verbeux
- YAML
- ZeroDivisionError
- @base_path
- Acceptez
- Accès

- Axi
- action
- les attributs
- application2
- rappeler
- Catégorie
- connexion
- base de données
- répartiteur
- afficher1
- conduire
- les erreurs
- format
- hôte
- clé
- disposition
- charge
- lien
- Nouveau
- notifier
- ouvrir
- Publique
- citation
- rendre
- demande
- des enregistrements
- les réponses
- enregistrer
- portée
- envoyer
- session
- système
- modèle
- tester
- temps libre
- to_s
- type
- URI
- visites
- Observateur

Noms de champs de base de données

- créé à
- créé sur
- updated_at
- mis à jour le

- deleted_at
- (paranoïa
- gemme)
- Version serrure
- type
- id
- # {nom_table} _count
- position
- parent_id
- lft
- rgt
- quote_value

Mots réservés Ruby

- alias
- et
- COMMENCER
- commencer
- Pause
- Cas
- classe
- déf
- défini?
- faire
- autre
- elsif
- FIN
- fin
- assurer
- faux
- pour
- si
- module
- prochain
- néant
- ne pas
- ou
- refaire
- porter secours
- recommencez
- revenir
- soi
- super
- puis
- vrai
- undef

- sauf si
- jusqu'à
- quand
- tandis que
- rendement
- _ *FICHIER* _
- _ *LIGNE* _

Lire Mots réservés en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/10818/mots-reserves

Chapitre 51: Organisation de classe

Remarques

Cela semble être une chose simple à faire, mais lorsque vous commencez à grossir, vous serez reconnaissant d'avoir pris le temps de les organiser.

Examples

Classe de modèle

```
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
```

Les modèles sont généralement responsables de:

- établir des relations
- validation des données
- fournir un accès aux données via des portées et des méthodes
- Effectuer des actions sur la persistance des données.

Au plus haut niveau, les modèles décrivent les concepts de domaine et gèrent leur persistance.

Classe de service

Le contrôleur est un point d'entrée dans notre application. Cependant, ce n'est pas le seul point d'entrée possible. Je voudrais avoir ma logique accessible depuis:

- Tâches de ratissage
- emplois de fond
- console
- des tests

Si je lance ma logique dans un contrôleur, il ne sera pas accessible de tous ces endroits. Essayons donc l'approche «maigre contrôleur, gros modèle» et déplaçons la logique vers un modèle. Mais lequel? Si une logique donnée implique des modèles d' `User` , de `Cart` et de `Product` , où devrait-elle vivre?

Une classe qui hérite d' `ActiveRecord::Base` déjà beaucoup de responsabilités. Il gère l'interface de requête, les associations et les validations. Si vous ajoutez encore plus de code à votre modèle, il deviendra rapidement un gâchis intraitable avec des centaines de méthodes publiques.

Un service est simplement un objet Ruby normal. Sa classe n'a pas à hériter d'une classe spécifique. Son nom est une expression verbale, par exemple `CreateUserAccount` plutôt que `UserCreation` ou `UserCreationService` . Il vit dans le répertoire app / services. Vous devez créer ce répertoire par vous-même, mais Rails va automatiquement charger les classes à l'intérieur.

Un objet de service fait une chose

Un objet de service (objet de méthode aka) effectue une action. Il détient la logique métier pour effectuer cette action. Voici un exemple:

```
# app/services/accept_invite.rb
class AcceptInvite
  def self.call(invite, user)
    invite.accept!(user)
    UserMailer.invite_accepted(invite).deliver
  end
end
```

Les trois conventions que je suis sont:

Les services vont dans le `app/services directory` . Je vous encourage à utiliser des sous-répertoires pour les domaines à forte logique applicative. Par exemple:

- Le fichier `app/services/invite/accept.rb` définira `Invite::Accept` lorsque `app/services/invite/create.rb` définira `Invite::Create`
- Les services commencent par un verbe (et ne se terminent pas par Service): `ApproveTransaction` , `SendTestNewsletter` , `ImportUsersFromCsv`
- Les services répondent à la méthode d' `call` . J'ai trouvé que l'utilisation d'un autre verbe le rend un peu redondant: `ApproveTransaction.approve()` ne lit pas bien. De plus, la méthode `call` est la méthode de facto pour `lambda` objets `lambda` , `procs` et method.

Avantages

Les objets de service montrent ce que fait mon application

Je peux simplement parcourir le répertoire des services pour voir ce que mon application fait: `ApproveTransaction` , `CancelTransaction` , `BlockAccount` , `SendTransactionApprovalReminder` ...

Un regard rapide sur un objet de service et je sais quelle logique métier est impliquée. Je n'ai pas besoin de passer par les contrôleurs, les rappels de modèles `ActiveRecord` et les observateurs pour comprendre ce que signifie «approuver une transaction».

Modèles et contrôleurs de nettoyage

Les contrôleurs transforment la requête (params, session, cookies) en arguments, les transmettent au service et les redirigent ou les rendent en fonction de la réponse du service.

```
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
```

Les modèles ne traitent que des associations, des étendues, des validations et de la persistance.

```
class Invite < ActiveRecord::Base
  def accept!(user, time=Time.now)
    update_attributes!(
      accepted_by_user_id: user.id,
      accepted_at: time
    )
  end
end
```

Cela rend les modèles et les contrôleurs beaucoup plus faciles à tester et à entretenir!

Quand utiliser la classe de service

Atteindre des objets de service lorsqu'une action répond à un ou plusieurs de ces critères:

- L'action est complexe (par exemple, la fermeture des livres à la fin d'une période comptable)
- L'action s'étend sur plusieurs modèles (par exemple, un achat en ligne via des objets Order, CreditCard et Customer).
- L'action interagit avec un service externe (par exemple, publication sur les réseaux sociaux)
- L'action n'est pas une préoccupation centrale du modèle sous-jacent (par exemple, balayer des données obsolètes après une certaine période).
- Il y a plusieurs façons d'effectuer l'action (par exemple, authentifier avec un jeton d'accès ou un mot de passe).

Sources

Adam Niedzielski Blog

Blog de la brasserie

Code Climate Blog

Lire Organisation de classe en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/7623/organisation-de-classe

Chapitre 52: Outils pour l'optimisation du code Ruby on Rails et le nettoyage

Introduction

Garder votre code propre et organisé tout en développant une grande application Rails peut constituer un véritable défi, même pour un développeur expérimenté. Heureusement, il existe toute une catégorie de gemmes qui rendent ce travail beaucoup plus facile.

Examples

Si vous souhaitez que votre code soit maintenable, sécurisé et optimisé, examinez quelques joyaux pour l'optimisation et le nettoyage du code:

Balle

Celui-ci m'a particulièrement frappé. La puce gem vous aide à tuer toutes les requêtes N + 1, ainsi que les relations inutilement chargées. Une fois que vous l'avez installé et que vous commencez à visiter les différents itinéraires en cours de développement, des boîtes d'alerte contenant des avertissements indiquant les requêtes de base de données devant être optimisées s'affichent. Cela fonctionne parfaitement et est extrêmement utile pour optimiser votre application.

Rails Best Practices

Analyseur de code statique pour trouver des odeurs de code spécifiques à Rails. Il offre une variété de suggestions; Utilisez l'accès à la portée, restreignez les itinéraires générés automatiquement, ajoutez des index de base de données, etc. Néanmoins, il contient de nombreuses suggestions intéressantes qui vous donneront une meilleure idée de la manière de recadrer votre code et d'apprendre les meilleures pratiques.

Rubocop

Un analyseur de code statique Ruby que vous pouvez utiliser pour vérifier si votre code est conforme aux directives du code de communauté Ruby. La gem signale des violations de style via la ligne de commande, avec de nombreuses fonctionnalités utiles de refactorisation de code, telles que l'affectation de variables inutiles, l'utilisation redondante de Object # to_s dans l'interpolation ou même l'argument de méthode non utilisé.

Une bonne chose est qu'il est très configurable, car l'analyseur peut être très irritant si vous ne suivez pas le guide de style Ruby à 100% (c.-à-d. Vous avez beaucoup d'espaces blancs ou vous doublez vos chaînes même sans interpolation). .

Il est divisé en 4 sous-analyseurs (appelés flics): Style, Lint, Metrics et Rails.

Lire Outils pour l'optimisation du code Ruby on Rails et le nettoyage en ligne:

https://riptutorial.com/fr/ruby-on-rails/topic/8713/outils-pour-l-optimisation-du-code-ruby-on-rails-et-le-nettoyage

Chapitre 53: Ouvrière

Examples

Définir les usines

Si vous avez une classe ActiveRecord User avec des attributs de nom et de courrier électronique, vous pouvez créer une fabrique en faisant en sorte que FactoryGirl le devienne:

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

Ou vous pouvez le rendre explicite et même changer son nom:

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

Ensuite, dans vos spécifications, vous pouvez utiliser les méthodes de FactoryGirl avec celles-ci, comme ceci:

```
# 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)
```

Les méthodes les plus courantes sont:

```
# 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)
```

Lire Ouvrière en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/8330/ouvriere

Chapitre 54: Pipeline d'actifs

Introduction

Le pipeline d'actifs fournit une structure pour concaténer et réduire ou compresser les actifs JavaScript et CSS. Il ajoute également la possibilité d'écrire ces ressources dans d'autres langues et dans des pré-processeurs tels que CoffeeScript, Sass et ERB. Il permet de combiner automatiquement les actifs de votre application avec les actifs d'autres gemmes. Par exemple, jquery-rails inclut une copie de jquery.js et active les fonctionnalités AJAX dans Rails.

Examples

Tâches de ratissage

Par défaut, les `sprockets-rails` sont livrés avec les tâches suivantes:

- `assets:clean[keep]` : supprimer les anciens éléments compilés
- `assets:clobber` : supprimer les actifs compilés
- `assets:environment` : environnement de compilation des actifs de chargement
- `assets:precompile` : compile tous les actifs nommés dans `config.assets.precompile`

Fichiers manifestes et directives

Dans l'initalizer des `assets` (`config/initializers/assets.rb`), il y a quelques fichiers explicitement définis pour être précompilés.

```
# 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 )
```

Dans cet exemple, les fichiers `application.coffee` et `application.scss` sont appelés "fichiers manifestes". Ces fichiers doivent être utilisés pour inclure d'autres ressources JavaScript ou CSS. Les commandes suivantes sont disponibles:

- `require <path>` : L' `require` des fonctions directive similaires à son propre Ruby `require` . Il fournit un moyen de déclarer une dépendance sur un fichier dans votre chemin et garantit qu'il est chargé une seule fois avant le fichier source.
- `require_directory <path>` : requiert tous les fichiers d'un même répertoire. Il est similaire à `path/*` car il ne suit pas les répertoires imbriqués.
- `require_tree <path>` : nécessite tous les fichiers imbriqués dans un répertoire. Son équivalent en glob est `path/**/*` .
- `require_self` : provoque le corps du fichier en cours à insérer avant toute modification ultérieure `require` les directives. Utile dans les fichiers CSS, où il est courant que le fichier d'index contienne des styles globaux devant être définis avant le chargement des autres

dépendances.

- `stub <path>` : supprime un fichier de l'inclusion
- `depend_on <path>` : vous permet d'affirmer une dépendance sur un fichier sans l'inclure. Ceci est utilisé à des fins de mise en cache. Toute modification apportée au fichier de dépendance invalidera le cache du fichier source.

Un fichier `application.scss` pourrait ressembler à:

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

Un autre exemple est le fichier `application.coffee` . Ici avec y compris `jquery` et `Turbolinks` :

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

Si vous n'utilisez pas CoffeeScript, mais que JavaScript est clair, la syntaxe serait la suivante:

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

Utilisation de base

Le pipeline d’actifs est utilisé de deux manières:

1. Lors de l'exécution d'un serveur en mode de développement, il pré-traite automatiquement et prépare vos actifs à la volée.

2. En mode production, vous l'utiliserez probablement pour pré-traiter, dimensionner, compresser et compiler vos actifs. Vous pouvez le faire en exécutant la commande suivante:

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

Lire Pipeline d'actifs en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3386/pipeline-d-actifs

Chapitre 55: Rails 5

Examples

Créer une API Ruby on Rails 5

Pour créer une nouvelle API Rails 5, ouvrez un terminal et exécutez la commande suivante:

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

La structure de fichier suivante sera créée:

```
  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
```

Cette structure de fichier sera créée dans un nouveau dossier appelé `app_name` . Il contient tous les actifs et le code nécessaires pour démarrer votre projet.

Entrez le dossier et installez les dépendances:

```
cd app_name
bundle install
```

Vous devriez également démarrer votre base de données. Rails utilise SQLite comme base de données par défaut. Pour le créer, lancez:

```
rake db:setup
```

Maintenant, lancez votre application:

```
$ rails server
```

Lorsque vous ouvrez votre navigateur à l' `http://localhost:3000` , votre nouvelle API brillante (vide) devrait être lancée!

Comment installer Ruby on Rails 5 sur RVM

RVM est un excellent outil pour gérer vos versions de ruby et configurer votre environnement de travail.

En supposant que RVM est déjà installé, pour obtenir la dernière version de ruby, nécessaire pour ces exemples, ouvrez un terminal et lancez:

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

Vérifiez votre version de Ruby en lançant:

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

Pour installer Rails 5, créez d'abord un nouveau gemset à l'aide de la dernière version de Ruby, puis installez les rails:

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

Pour vérifier la version de vos rails, exécutez:

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

Lire Rails 5 en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3019/rails-5

Chapitre 56: Rails Best Practices

Examples

Ne te répète pas (DRY)

Pour aider à maintenir un code propre, Rails suit le principe de DRY.

Cela implique, autant que possible, de réutiliser autant de code que possible plutôt que de dupliquer un code similaire à plusieurs endroits (par exemple, en utilisant des partiels). Cela réduit les *erreurs* , maintient votre code *propre* et applique le principe d' *écriture du code une fois* , puis de le réutiliser. Il est également plus facile et plus efficace de mettre à jour le code au même endroit que de mettre à jour plusieurs parties du même code. Ainsi, votre code est plus modulaire et robuste.

Aussi *Fat Model, Skinny Controller* est DRY, car vous écrivez le code dans votre modèle et dans le contrôleur ne faites que l'appel, comme:

```
# 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
```

Cela permet également de conduire à une structure pilotée par l'API où les méthodes internes sont masquées et les modifications obtenues grâce à la transmission de paramètres en mode API.

Convention sur la configuration

Dans Rails, vous découvrez les *contrôleurs, les vues et les modèles* de votre base de données.

Pour réduire le besoin d'une configuration lourde, Rails implémente des règles pour faciliter l'utilisation de l'application. Vous pouvez définir vos propres règles, mais pour le début (et pour plus tard), il est conseillé de respecter les conventions proposées par Rails.

Ces conventions vont accélérer le développement, garder votre code concis et lisible et vous permettre une navigation facile dans votre application.

Les conventions abaissent également les barrières à l'entrée pour les débutants. Il y a tellement de conventions dans Rails qu'un débutant n'a même pas besoin de connaître, mais peut simplement profiter de l'ignorance. Il est possible de créer de superbes applications sans savoir pourquoi tout est comme ça.

Par exemple

Si vous avez une table de base de données appelée `orders` avec l' `id` clé primaire, le modèle correspondant est appelé `order` et le contrôleur qui gère toute la logique s'appelle `orders_controller` . La vue est divisée en différentes actions: si le contrôleur a une `new` action et une action d' `edit` , il y a également une `new` vue d' `edit` .

Par exemple

Pour créer une application, il vous suffit d'exécuter `rails new app_name` . Cela générera environ 70 fichiers et dossiers comprenant l'infrastructure et les bases de votre application Rails.

Il comprend:

- Dossiers contenant vos modèles (couche de base de données), contrôleurs et vues
- Dossiers contenant des tests unitaires pour votre application
- Dossiers pour contenir vos ressources Web comme les fichiers Javascript et CSS
- Fichiers par défaut pour les réponses HTTP 400 (fichier introuvable)
- Beaucoup d'autres

Fat Model, Skinny Controller

«Fat Model, Skinny Controller» fait référence à la manière dont les parties M et C de MVC fonctionnent parfaitement ensemble. À savoir, toute logique liée à la non-réponse devrait aller dans le modèle, idéalement dans une belle méthode testable. Pendant ce temps, le contrôleur «skinny» est simplement une interface agréable entre la vue et le modèle.

En pratique, cela peut nécessiter différents types de refactorisation, mais tout se résume à une idée: en déplaçant une logique qui ne concerne pas la réponse au modèle (au lieu du contrôleur), non seulement vous avez favorisé la réutilisation. dans la mesure du possible, mais vous avez également permis de tester votre code en dehors du contexte d'une demande.

Regardons un exemple simple. Disons que vous avez un code comme celui-ci:

```
def index
  @published_posts = Post.where('published_at <= ?', Time.now)
  @unpublished_posts = Post.where('published_at IS NULL OR published_at > ?', Time.now)
end
```

Vous pouvez le changer en ceci:

```
def index
  @published_posts = Post.published
  @unpublished_posts = Post.unpublished
end
```

Ensuite, vous pouvez déplacer la logique vers votre post-modèle, où il pourrait ressembler à ceci:

```
scope :published, ->(timestamp = Time.now) { where('published_at <= ?', timestamp) }
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at >
?', timestamp) }
```

Attention à default_scope

ActiveRecord inclut `default_scope` pour `default_scope` automatiquement un modèle par défaut.

```
class Post
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

Le code ci-dessus servira les publications déjà publiées lorsque vous effectuez une requête sur le modèle.

```
Post.all # will only list published posts
```

Cette portée, bien qu'inoffensive, a de multiples effets secondaires cachés que vous ne voudrez peut-être pas.

`default_scope` et `order`

Puisque vous avez déclaré une `order` dans `default_scope` , l' `order` appel sur `Post` sera ajouté en tant que commandes supplémentaires au lieu de remplacer la valeur par défaut.

```
Post.order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."created_at"
DESC, "posts"."updated_at" DESC
```

Ce n'est probablement pas le comportement que vous vouliez; vous pouvez remplacer cela en excluant la `order` du périmètre en premier

```
Post.except(:order).order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."updated_at"
DESC
```

`default_scope` et initialisation du modèle

Comme pour toute autre `ActiveRecord::Relation` , `default_scope` modifiera l'état par défaut des modèles initialisés à partir de celui-ci.

Dans l'exemple ci-dessus, `Post` a défini `where(published: true)` la valeur par défaut, de sorte que

les nouveaux modèles de `Post` définiront également.

```
Post.new # => <Post published: true>
```

unscoped

`default_scope` peut être nominalement effacé en appelant non- `unscoped` premier, mais cela a aussi des effets secondaires. Prenez, par exemple, un modèle STI:

```
class Post < Document
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

Par défaut, les requêtes sur `Post` seront étendues pour `type` colonnes contenant `'Post'` . Mais non `unscoped` ceci avec votre propre `default_scope` , donc si vous utilisez `unscoped` vous devez vous rappeler de le prendre en compte.

```
Post.unscoped.where(type: 'Post').order(updated_at: :desc)
```

unscoped et modèles

Envisager une relation entre `Post` et `User`

```
class Post < ApplicationRecord
  belongs_to :user
  default_scope ->{ where(published: true).order(created_at: :desc) }
end

class User < ApplicationRecord
  has_many :posts
end
```

En obtenant un `User` individuel, vous pouvez voir les messages associés:

```
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]]
```

Mais vous voulez effacer le `default_scope` de la relation des `posts` , de sorte que vous `unscoped`

```
user.posts.unscoped
```

```
SELECT "posts".* FROM "posts"
```

Cela efface la condition `user_id` ainsi que le `default_scope` .

Un exemple d'utilisation pour `default_scope`

Malgré tout cela, il existe des situations où l'utilisation de `default_scope` est justifiable.

Considérons un système multi-locataire où plusieurs sous-domaines sont servis à partir de la même application mais avec des données isolées. L'un des moyens de parvenir à cette isolation consiste à `default_scope` . Les inconvénients dans d'autres cas deviennent des avantages ici.

```
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
```

Tout ce que vous avez à faire est de définir `Tenant.current_id` sur quelque chose au début de la requête, et toute table qui contient `tenant_id` deviendra automatiquement portée sans code supplémentaire. L'instanciation d'enregistrements héritera automatiquement l'identifiant du locataire sous lequel ils ont été créés.

L'important à propos de ce cas d'utilisation est que la portée est définie une fois par requête et qu'elle ne change pas. Les seuls cas dont vous aurez besoin de les `unscoped` sont des cas particuliers tels que les travailleurs en arrière-plan qui s'exécutent en dehors d'une étendue de requête.

Vous n'en aurez pas besoin (YAGNI)

Si vous pouvez dire "YAGNI" (vous n'en aurez pas besoin) à propos d'une fonctionnalité, vous feriez mieux de ne pas l'implémenter. Il est possible de gagner beaucoup de temps de développement en se concentrant sur la simplicité. L'implémentation de telles fonctionnalités peut de toute façon entraîner des problèmes:

Problèmes

Ingénierie

Si un produit est plus compliqué que cela doit être, il est sur-conçu. Habituellement, ces fonctionnalités «non encore utilisées» ne seront jamais utilisées de la manière prévue, mais elles devront être remises à neuf si elles sont utilisées. Les optimisations prématurées, en particulier les optimisations de performances, conduisent souvent à des décisions de conception qui se révéleront erronées dans le futur.

Ballonnement de code

Code Bloat signifie un code compliqué inutile. Cela peut se produire, par exemple, par abstraction, redondance ou application incorrecte des modèles de conception. La base de code devient difficile à comprendre, déroutante et coûteuse à maintenir.

Fonction de fluage

La fonction de fluage des fonctionnalités fait référence à l'ajout de nouvelles fonctionnalités qui vont au-delà des fonctionnalités de base du produit et entraînent une complexité inutilement élevée du produit.

Long temps de développement

Le temps nécessaire pour développer les fonctionnalités nécessaires est utilisé pour développer des fonctionnalités inutiles. Le produit prend plus de temps à livrer.

Solutions

KISS - Reste simple, stupide

Selon KISS, la plupart des systèmes fonctionnent le mieux s'ils sont conçus simplement. La simplicité devrait être l’un des principaux objectifs de conception pour réduire la complexité. Cela peut être réalisé en suivant le «principe de la responsabilité unique», par exemple.

YAGNI - Vous n'en aurez pas besoin

Moins est plus. Pensez à toutes les fonctionnalités, est-ce vraiment nécessaire? Si vous pensez à YAGNI, laissez-le de côté. Il est préférable de le développer quand il le faut.

Refactoring en continu

Le produit est amélioré régulièrement. Avec le refactoring, nous pouvons nous assurer que le produit est conforme aux meilleures pratiques et ne dégénère pas en patch.

Objets de domaine (No More Fat Models)

"Fat Model, Skinny Controller" est une très bonne première étape, mais elle ne s’améliore pas une fois que votre base de code commence à se développer.

Pensons à la responsabilité unique des modèles. Quelle est la responsabilité unique des modèles? Est-ce que c'est la logique métier? S'agit-il d'une logique liée à la non-réponse?

Non, sa responsabilité est de gérer la couche de persistance et son abstraction.

La logique métier, de même que toute logique associée à la non-réponse et toute logique liée à la non-persistance, doit figurer dans les objets de domaine.

Les objets de domaine sont des classes conçues pour n'avoir qu'une seule responsabilité dans le domaine du problème. Laissez vos cours " Crier leur architecture " pour les problèmes qu'ils résolvent.

En pratique, vous devez vous efforcer d'utiliser des modèles réduits, des vues maigres et des manettes minces. L'architecture de votre solution ne doit pas être influencée par le cadre que vous choisissez.

Par exemple

Disons que vous êtes un marché qui facture une commission fixe de 15% à vos clients via Stripe. Si vous facturez une commission fixe de 15%, cela signifie que votre commission change en fonction du montant de la commande, car Stripe facture 2,9% + 30 ¢.

Le montant que vous facturez en tant que commission doit être: `amount*0.15 - (amount*0.029 + 0.30)` .

N'écrivez pas cette logique dans le modèle:

```
# 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
```

Dès que vous intégrez un nouveau mode de paiement, vous ne pourrez plus adapter cette fonctionnalité à ce modèle.

De plus, dès que vous commencez à intégrer davantage de logique métier, votre objet `Order` commence à perdre sa cohésion .

Préférer les objets du domaine, avec le calcul de la commission complètement abstrait de la responsabilité des ordres persistants:

```
# 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
```

L'utilisation d'objets de domaine présente les avantages architecturaux suivants:

- Il est extrêmement facile de tester les unités, car aucune installation ou usine n'est requise pour instancier les objets avec la logique.
- fonctionne avec tout ce qui accepte le `amount` du message.
- maintient chaque objet de domaine petit, avec des responsabilités clairement définies et une plus grande cohésion.
- évolue facilement avec les nouveaux modes de paiement par ajout, et non par modification .
- arrête la tendance à avoir un objet `User` toujours croissant dans chaque application Ruby on Rails.

Personnellement, j'aime mettre des objets de domaine dans `lib` . Si vous le faites, n'oubliez pas de l'ajouter à `autoload_paths` :

```
# config/application.rb
```

```
config.autoload_paths << Rails.root.join('lib')
```

Vous pouvez également préférer créer des objets de domaine plus orientés vers l'action, en suivant le modèle Command / Query. Dans un tel cas, placer ces objets dans les `app/commands` pourrait être un meilleur endroit car tous les sous-répertoires d' `app` sont automatiquement ajoutés au chemin de chargement automatique.

Lire Rails Best Practices en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/1207/rails-best-practices

Chapitre 57: Rails Cookbook - Advanced rails recettes / apprentissages et techniques de codage

Examples

Jouer avec des tables en utilisant la console de rails

Voir les tableaux

```
ActiveRecord::Base.connection.tables
```

Supprimer n'importe quelle table .

```
    ActiveRecord::Base.connection.drop_table("users")
    ------------OR----------------------
    ActiveRecord::Migration.drop_table(:users)
    ------------OR----------------------
    ActiveRecord::Base.connection.execute("drop table users")
```

Supprimer l'index de la colonne existante

```
 ActiveRecord::Migration.remove_index(:users, :name => 'index_users_on_country')
```

où `country` est un nom de colonne dans le fichier de migration avec l'index **déjà** ajouté dans la table `users` , comme indiqué ci-dessous: -

```
 t.string :country,add_index: true
```

Supprimer la contrainte de clé étrangère

```
ActiveRecord::Base.connection.remove_foreign_key('food_items', 'menus')
```

où les `menus has_many food_items` et leurs migrations respectives.

Ajouter une colonne

```
ActiveRecord::Migration.remove_column :table_name, :column_name
```

par exemple:-

```
 ActiveRecord::Migration.add_column :profiles, :profile_likes, :integer, :default => 0
```

Méthodes Rails - retour des valeurs booléennes

Toute méthode dans le modèle Rails peut renvoyer une valeur booléenne.

méthode simple-

```
##this method return ActiveRecord::Relation
def check_if_user_profile_is_complete
  User.includes( :profile_pictures,:address,:contact_detail).where("user.id = ?",self)
end
```

Encore une fois, la méthode simple retourne la valeur booléenne-

```
##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
```

Donc, la même méthode va maintenant retourner booléen au lieu de tout autre chose :).

Manipulation de l'erreur - méthode indéfinie `where 'pour

Parfois, nous voulons utiliser une requête `where` sur une collection d'enregistrements renvoyée, qui n'est pas `ActiveRecord::Relation` conséquent, nous obtenons l'erreur ci-dessus en tant que clause `Where` est connue de `ActiveRecord` et non de `Array` .

Il existe une solution précise en utilisant les `Joins` .

EXEMPLE : -

> Supposons que je doive trouver tous les profils d'utilisateur (UserProfile) qui ne sont pas un utilisateur (User) avec un identifiant = 10.

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).map(&:user).where.not(:id=>10)
```

Donc, la requête ci-dessus échouera après la `map` car la `map` retournera un `array` qui **ne** fonctionnera **pas** avec la clause `where` .

Mais en utilisant des jointures, cela fonctionnera,

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).joins(:user).where.not(:id=>10)
```

Comme les `joins` produiront des enregistrements similaires à la `map` elles seront `ActiveRecord` et **non** un `Array` .

Lire Rails Cookbook - Advanced rails recettes / apprentissages et techniques de codage en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/7259/rails-cookbook---advanced-rails-recettes---apprentissages-et-techniques-de-codage

Chapitre 58: Rails -Engines

Introduction

Les moteurs peuvent être considérés comme des applications miniatures qui fournissent des fonctionnalités à leurs applications hôtes. Une application Rails est en fait juste un moteur "suralimenté", la classe Rails :: Application héritant beaucoup de son comportement de Rails :: Engine.

Les moteurs sont les applications / plugins de rails réutilisables. Cela fonctionne comme un joyau. Les moteurs célèbres sont les gemmes Device, Spree qui peuvent être intégrées facilement aux applications de rails.

Syntaxe

- `rails plugin new [engine name] --mountable`

Paramètres

Paramètres	Objectif
--montable	option indique au générateur que vous souhaitez créer un moteur "montable" et isolé par espace de noms
--plein	option indique au générateur que vous souhaitez créer un moteur, y compris une structure de squelette

Remarques

Les moteurs sont de très bonnes options pour créer des plug-ins réutilisables pour les applications de rails

Examples

Des exemples célèbres sont

Générer un moteur de blog simple

```
rails plugin new [engine name] --mountable
```

Des exemples de moteurs célèbres sont

Dispositif (gem d'authentification pour rails)

Spree (commerce électronique)

Lire Rails -Engines en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/10881/rails--engines

Chapitre 59: Rails génèrent des commandes

Introduction

Utilisation: les `rails generate GENERATOR_NAME [args] [options]` .

Utilisez les `rails generate` pour `rails generate` liste des générateurs disponibles. Alias: `rails g` .

Paramètres

Paramètre	Détails
`-h` / `--help`	Obtenir de l'aide sur n'importe quelle commande de générateur
`-p` / `--pretend`	Mode de prétention: Exécuter le générateur mais ne créera ni ne modifiera aucun fichier
`field:type`	«nom-de-champ» est le nom de la colonne à créer et «type» est le type de données de la colonne. Les valeurs possibles pour 'type' dans le `field:type` sont indiquées dans la section Remarques.

Remarques

Les valeurs possibles pour 'type' dans le `field:type` sont:

Type de données	La description
`:string`	Pour les petits morceaux de texte (a généralement une limite de caractères de 255)
`:text`	Pour des morceaux de texte plus longs, comme un paragraphe
`:binary`	Stockage de données comprenant des images, des audios et des vidéos
`:boolean`	Stocker des valeurs vraies ou fausses
`:date`	Seulement la date
`:time`	Seul le temps
`:datetime`	Date et l'heure
`:float`	Stockage des flotteurs sans précision
`:decimal`	Stockage des flotteurs avec précision

Type de données	La description
`:integer`	Stocker des nombres entiers

Examples

Rails Generate Model

Pour générer un modèle `ActiveRecord` qui crée automatiquement les migrations de base de données et les fichiers de test corrects pour votre modèle, entrez cette commande.

```
rails generate model NAME column_name:column_type
```

'NAME' est le nom du modèle. "champ" est le nom de la colonne dans la table de base de données et "type" est le type de colonne (par exemple, `name:string` ou `body:text`). Consultez la section Remarques pour obtenir une liste des types de colonne pris en charge.

Pour configurer des clés étrangères, ajoutez `belongs_to:model_name` .

Alors dites que vous vouliez configurer un modèle d' `User` qui a un `username` , un `email` et qui appartient à une `School` .

```
rails generate model User username:string email:string school:belongs_to
```

`rails g` est un raccourci pour `rails generate` . Cela produirait le même résultat

```
rails g model User username:string email:string school:belongs_to
```

Rails Générer Migration

Vous pouvez générer un fichier de migration des rails à partir du terminal à l'aide de la commande suivante:

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

Pour obtenir une liste de toutes les options prises en charge par la commande, vous pouvez exécuter la commande sans aucun argument, car les `rails generate migration` .

Par exemple, si vous voulez ajouter `first_name` et `last_name` champs à `users` table, vous pouvez le faire:

```
rails generate migration AddNamesToUsers last_name:string first_name:string
```

Rails créera le fichier de migration suivant:

```
class AddNamesToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :last_name, :string
    add_column :users, :first_name, :string
  end
end
```

Maintenant, appliquez les migrations en attente à la base de données en exécutant les opérations suivantes dans le terminal:

5.0

```
rake db:migrate
```

5.0

```
rails db:migrate
```

> **Note:** Pour encore moins de saisie, vous pouvez remplacer `generate` par `g` .

Rails Générer un échafaud

AVERTISSEMENT : L'échafaudage n'est pas recommandé, sauf s'il s'agit d'applications / tests CRUD très classiques. Cela peut générer beaucoup de fichiers (vues / modèles / contrôleurs) qui ne sont pas nécessaires dans votre application Web, provoquant ainsi des maux de tête (mauvais :().

Pour générer un échafaudage complet pour un nouvel objet, y compris le modèle, le contrôleur, les vues, les actifs et les tests, utilisez la commande `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
```

Ensuite, vous pouvez exécuter `rake db:migrate` pour configurer la table de base de données.

Ensuite, vous pouvez visiter http: // localhost: 3000 / widgets et vous verrez un échafaudage CRUD entièrement fonctionnel.

Rails Generate Controller

nous pouvons créer un nouveau contrôleur avec des commandes de contrôleur de `rails g controller` .

```
$ bin/rails generate controller controller_name
```

Le générateur de contrôleur attend des paramètres sous la forme de la `generate controller ControllerName action1 action2` .

Ce qui suit crée un contrôleur de message d'accueil avec une action de salut.

```
$ bin/rails generate controller Greetings hello
```

Vous verrez la sortie suivante

```
     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
```

Cela génère le suivant

Fichier	Exemple
Fichier de contrôleur	`greetings_controller.rb`
Voir la fiche	`hello.html.erb`
Fichier de test fonctionnel	`greetings_controller_test.rb`

Fichier	Exemple
Afficher l'aide	`greetings_helper.rb`
Fichier JavaScript	`greetings.coffee`

Il ajoutera également des routes pour chaque action dans `routes.rb`

Lire Rails génèrent des commandes en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2540/rails-generent-des-commandes

Chapitre 60: Rails sur docker

Introduction

Ce tutoriel commencera avec Docker installé et avec une application Rails

Examples

Docker et Docker-Composer

Tout d'abord, nous devrons créer notre `Dockerfile` . Un bon exemple peut être trouvé sur ce blog par Nick Janetakis.

Ce code contient le script qui sera exécuté sur notre machine docker au moment du démarrage. Pour cette raison, nous installons toutes les bibliothèques requises et nous terminons avec le lancement de Puma (serveur de développement RoR).

```
# Use the barebones version of Ruby 2.3.
FROM ruby:2.3.0-slim

# Optionally set a maintainer name to let people know who made this image.
MAINTAINER Nick Janetakis <nick.janetakis@gmail.com>

# Install dependencies:
# - build-essential: To ensure certain gems can be compiled
# - nodejs: Compile assets
# - libpq-dev: Communicate with postgres through the postgres gem
RUN apt-get update && apt-get install -qq -y --no-install-recommends \
      build-essential nodejs libpq-dev git


# Set an environment variable to store where the app is installed to inside
# of the Docker image. The name matches the project name out of convention only.
ENV INSTALL_PATH /mh-backend
RUN mkdir -p $INSTALL_PATH

# This sets the context of where commands will be running in and is documented
# on Docker's website extensively.
WORKDIR $INSTALL_PATH

# We want binstubs to be available so we can directly call sidekiq and
# potentially other binaries as command overrides without depending on
# bundle exec.
COPY Gemfile* $INSTALL_PATH/

ENV BUNDLE_GEMFILE $INSTALL_PATH/Gemfile
ENV BUNDLE_JOBS 2
ENV BUNDLE_PATH /gembox

RUN bundle install

# Copy in the application code from your work station at the current directory
# over to the working directory.
```

```
COPY . .

# Ensure the static assets are exposed to a volume so that nginx can read
# in these values later.
VOLUME ["$INSTALL_PATH/public"]

ENV RAILS_LOG_TO_STDOUT true


# The default command that gets run will be to start the Puma server.
CMD bundle exec puma -C config/puma.rb
```

De plus, nous utiliserons docker-compose, pour cela nous allons créer `docker-compose.yml` . L'explication de ce fichier sera plus un tutoriel de composition de docker qu'une intégration avec Rails et je ne couvrirai pas ici.

```
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
```

Juste avec ces deux fichiers, vous aurez assez pour exécuter `docker-compose up` et réveiller votre Docker

Lire Rails sur docker en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/10933/rails-sur-docker

Chapitre 61: Réagir avec les rails en utilisant la pierre de réaction

Examples

Réagir à l'installation pour Rails à l'aide de rails_react gem

Ajouter des rails de réaction à votre Gemfile:

```
gem 'react-rails'
```

Et installer:

```
bundle install
```

Ensuite, exécutez le script d'installation:

```
rails g react:install
```

Cette volonté:

Créez un fichier manifeste components.js et un répertoire app / assets / javascripts / components /, où vous placerez vos composants dans les champs suivants de votre application.js:

```
//= require react
//= require react_ujs
//= require components
```

Utiliser react_rails dans votre application

React.js construit

Vous pouvez choisir quel React.js compile (développement, production, avec ou sans add-ons) pour servir dans chaque environnement en ajoutant une configuration. Voici les valeurs par défaut:

```
# 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
```

Pour inclure des modules complémentaires, utilisez cette configuration:

```
MyApp::Application.configure do
  config.react.addons = true # defaults to false
end
```

Après avoir redémarré votre serveur Rails, // = require react fournira la version de React.js spécifiée par les configurations.

react-rails propose quelques autres options pour les versions et les builds de React.js. Voir VERSIONS.md pour plus d'informations sur l'utilisation du joyau de react-source ou sur vos propres copies de React.js.

JSX

Après avoir installé react-rails, redémarrez votre serveur. Maintenant, les fichiers .js.jsx seront transformés dans le pipeline des ressources.

Options BabelTransformer

Vous pouvez utiliser les transformateurs et les plug-ins personnalisés de babel, et passer les options au babel transpiler en ajoutant les configurations suivantes:

```
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]
}
```

Sous le capot, react-rails utilise ruby-babel-transpiler pour la transformation.

Rendu et montage

`react-rails` inclut une aide de vue `(react_component)` et un pilote JavaScript discret (react_ujs) qui fonctionnent ensemble pour placer les composants de React sur la page. Vous devriez exiger le pilote UJS dans votre manifeste après la réaction (et après les turbolinks si vous utilisez Turbolinks).

L'aide de vue place un div sur la page avec la classe de composant et les accessoires demandés. Par exemple:

```
<%= react_component('HelloMessage', name: 'John') %>
<!-- becomes: -->
<div data-react-class="HelloMessage" data-react-
props="{&amp;quot;name&amp;quot;:&amp;quot;John&amp;quot;}"></div>
```

Lors du chargement de la page, le pilote react_ujs analysera la page et montera les composants à l'aide de data-react-class et de data-react-props.

Si Turbolinks est présent, les composants sont montés sur la page: changez l'événement et

démonté sur la page: avant-décharger. Turbolinks> = 2.4.0 est recommandé car il expose de meilleurs événements.

En cas d'appels Ajax, le montage UJS peut être déclenché manuellement en appelant à partir de javascript:

ReactRailsUJS.mountComponents () La signature de l'assistant de vue est:

```
react_component(component_class_name, props={}, html_options={})
```

`component_class_name` est une chaîne qui nomme une classe de composants accessible globalement. Il peut avoir des points (par exemple, "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` peut inclure: `tag:` utiliser un élément autre qu'un div pour incorporer data-react-class et data-react-props `prerender: true` pour rendre le composant sur le serveur. `**other` Tout autre argument (par exemple, class :, id :) est transmis à content_tag.

Lire Réagir avec les rails en utilisant la pierre de réaction en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/7032/reagir-avec-les-rails-en-utilisant-la-pierre-de-reaction

Chapitre 62: Routage peu profond

Examples

1. Utilisation de faible profondeur

Un moyen d'éviter une imbrication profonde (comme recommandé ci-dessus) consiste à générer les actions de collecte délimitées sous le parent, de manière à avoir une idée de la hiérarchie, mais à ne pas imbriquer les actions membres. En d'autres termes, créer uniquement des routes avec le minimum d'informations pour identifier de manière unique la ressource, comme ceci:

```
resources :articles, shallow: true do
  resources :comments
  resources :quotes
  resources :drafts
end
```

La méthode peu profonde du DSL crée une portée à l'intérieur de laquelle chaque imbrication est superficielle. Cela génère les mêmes itinéraires que l'exemple précédent:

```
shallow do
  resources :articles do
    resources :comments
    resources :quotes
    resources :drafts
  end
end
```

Il existe deux options de portée pour personnaliser les routes peu profondes. : chemin des membres préfixes chemin_powow avec le paramètre spécifié:

```
scope shallow_path: "sekret" do
  resources :articles do
    resources :comments, shallow: true
  end
end
```

Utilisez la commande Rake pour obtenir les itinéraires générés, comme indiqué ci-dessous:

```
rake routes
```

Lire Routage peu profond en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/7775/routage-peu-profond

Chapitre 63: RSpec et Ruby on Rails

Remarques

RSpec est un framework de test pour Ruby ou, tel que défini par la documentation officielle, *RSpec est un outil de développement basé sur les comportements pour les programmeurs Ruby* .

Cette rubrique couvre l'utilisation de base de RSpec avec Ruby on Rails. Pour des informations spécifiques sur RSpec, visitez le sujet RSpec .

Examples

Installation de RSpec

Si vous souhaitez utiliser RSpec pour un projet Rails, vous devez utiliser la `rspec-rails` , qui peut générer automatiquement des aides et des fichiers de spécifications (par exemple, lorsque vous créez des modèles, des ressources ou des échafaudages à l'aide de `rails generate`).

Ajoutez `rspec-rails` à la fois aux groupes `:development` et `:test` dans le `Gemfile` :

```
group :development, :test do
  gem 'rspec-rails', '~> 3.5'
end
```

Exécutez `bundle` pour installer les dépendances.

Initialisez-le avec:

```
rails generate rspec:install
```

Cela créera un dossier `spec/` pour vos tests, ainsi que les fichiers de configuration suivants:

- `.rspec` contient les options par défaut pour l'outil `rspec` ligne de commande
- `spec/spec_helper.rb` inclut les options de configuration RSpec de base
- `spec/rails_helper.rb` ajoute d'autres options de configuration plus spécifiques pour utiliser RSpec et Rails ensemble.

Tous ces fichiers sont écrits avec des valeurs par défaut adaptées pour vous permettre de démarrer, mais vous pouvez ajouter des fonctionnalités et modifier les configurations en fonction de vos besoins au fur et à mesure que votre suite de tests se développe.

Lire RSpec et Ruby on Rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/5335/rspec-et-ruby-on-rails

Chapitre 64: Safe Constantize

Examples

Safe_constantize réussie

`User` est une classe `ActiveRecord` ou `Mongoid` . Remplacez l' `User` par n'importe quelle classe `Rails` de votre projet (même quelque chose comme `Integer` ou `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]
```

Echec de safe_constantize

Cet exemple ne fonctionnera pas car la chaîne transmise n'est pas reconnue comme une constante dans le projet. Même si vous passez dans `"array"` , cela ne fonctionnera pas car il n’est pas en majuscule.

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

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

Lire Safe Constantize en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3015/safe-constantize

Chapitre 65: Serializers Modèle Actif

Introduction

ActiveModelSerializers, ou AMS en abrégé, apportez la «convention sur la configuration» à votre génération JSON. ActiveModelSerializers fonctionne à travers deux composants: les sérialiseurs et les adaptateurs. Les sérialiseurs décrivent quels attributs et relations doivent être sérialisés. Les adaptateurs décrivent comment les attributs et les relations doivent être sérialisés.

Examples

Utiliser un sérialiseur

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

Lire Serializers Modèle Actif en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/9000/serializers-modele-actif

Chapitre 66: Stockage sécurisé des clés d'authentification

Introduction

De nombreuses API tierces requièrent une clé leur permettant de prévenir les abus. S'ils vous délivrent une clé, il est très important que vous ne commettiez pas la clé dans un référentiel public, car cela permettra aux autres de voler votre clé.

Examples

Stockage des clés d'authentification avec Figaro

Ajoutez `gem 'figaro'` à votre Gemfile et exécutez l' `bundle install` . Puis exécutez `bundle exec figaro install` ; Cela va créer config / application.yml et l'ajouter à votre fichier .gitignore, l'empêchant d'être ajouté au contrôle de version.

Vous pouvez stocker vos clés dans application.yml dans ce format:

```
SECRET_NAME: secret_value
```

où SECRET_NAME et secret_value sont le nom et la valeur de votre clé API.

Vous devez également nommer ces secrets dans config / secrets.yml. Vous pouvez avoir différents secrets dans chaque environnement. Le fichier devrait ressembler à ceci:

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

La manière dont vous utilisez ces clés varie, mais par exemple, `some_component` dans l'environnement de développement nécessite l'accès à `secret_name` . Dans config / environnements / development.rb, vous mettriez:

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

Enfin, supposons que vous souhaitiez créer un environnement de production sur Heroku. Cette commande téléchargera les valeurs dans config / environnements / production.rb vers Heroku:

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

Lire Stockage sécurisé des clés d'authentification en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/9711/stockage-securise-des-cles-d-authentification

Chapitre 67: Téléchargement de fichier

Examples

Téléchargement de fichier unique avec Carrierwave

Commencer à utiliser les téléchargements de fichiers dans Rails est assez simple, la première chose à faire est de choisir le plug-in pour gérer les téléchargements. Les plus courants sont **Carrierwave** et **Paperclip** . Les deux sont des fonctionnalités similaires et riches en documentation sur

Prenons un exemple avec une image téléchargée par avatar simple avec Carrierwave

Après avoir `bundle install` Carrierwave, tapez console

```
$ rails generate uploader ProfileUploader
```

Cela va créer un fichier de configuration situé dans */app/uploaders/profile_uploader.rb*

Ici, vous pouvez configurer le stockage (local ou cloud), appliquer des extensions pour les manipulations d'images (c.-à-d. Générer des vignettes via MiniMagick) et définir la liste blanche des extensions côté serveur.

Ensuite, créez une nouvelle migration avec string tipe pour user_pic et montez-la dans le modèle *user.rb.*

```
mount_uploader :user_pic, ProfileUploader
```

Ensuite, affichez un formulaire pour télécharger l'avatar (peut être une vue d'édition pour l'utilisateur)

```
<% 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 %>
```

Assurez-vous d'inclure {multipart: true} dans le formulaire de commande peut traiter les téléchargements. Accept est une option pour définir la liste blanche des extensions côté client.

Pour afficher un avatar, faites simplement

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

Modèle imbriqué - téléchargements multiples

Si vous souhaitez créer plusieurs téléchargements, la première chose à faire est de créer un

nouveau modèle et de définir des relations.

Supposons que vous souhaitiez plusieurs images pour le modèle de produit. Créer un nouveau modèle et le faire `belongs_to` modèle parent

```
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 est indispensable, car il nous permet de créer un formulaire imbriqué, afin que nous puissions télécharger un nouveau fichier, modifier le nom du produit et définir le prix à partir d'un seul formulaire

Ensuite, créez un formulaire dans une vue (edit / create)

```
    <%= 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 %>
```

Le contrôleur n'a rien de spécial, si vous ne voulez pas en créer un nouveau, créez-en un nouveau dans votre contrôleur de produit.

```
  # 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
```

Afficher toutes les images dans une vue

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

Lire Téléchargement de fichier en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2831/telechargement-de-fichier

Chapitre 68: Test d'applications Rails

Examples

Test de l'unité

L'unité teste des parties de l'application séparément. En général, une unité testée est une classe ou un module.

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

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

la source

Ce genre si le test est aussi direct et spécifique que possible.

Demander un test

Les tests de requête sont des tests de bout en bout qui imitent le comportement d'un utilisateur.

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

la source

Ce type de test se concentre sur les flux d'utilisateurs et passe par toutes les couches du système, parfois même en rendant javascript.

Lire Test d'applications Rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/7853/test-d-applications-rails

Chapitre 69: Transactions ActiveRecord

Remarques

Les transactions sont des blocs protecteurs où les instructions SQL ne sont permanentes que si elles peuvent toutes réussir en une seule action atomique. L'exemple classique est un transfert entre deux comptes où vous ne pouvez avoir de dépôt que si le retrait a réussi et vice versa. Les transactions renforcent l'intégrité de la base de données et protègent les données contre les erreurs de programme ou les pannes de base de données. Donc, fondamentalement, vous devez utiliser des blocs de transaction chaque fois que vous avez un certain nombre d'instructions qui doivent être exécutées ensemble ou pas du tout.

Examples

Exemple de base

Par exemple:

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

Cet exemple ne prendra que de l'argent de David et le donnera à Mary si ni le retrait ni le dépôt ne font exception. Les exceptions forceront un ROLLBACK qui renvoie la base de données à l'état avant le début de la transaction. Sachez cependant que les objets ne verront pas leurs données d'instance retournées à leur état pré-transactionnel.

Différentes classes ActiveRecord en une seule transaction

Bien que la méthode de la classe transaction soit appelée sur certaines classes ActiveRecord, les objets du bloc de transaction ne doivent pas nécessairement tous être des instances de cette classe. Cela est dû au fait que les transactions sont par connexion à la base de données, et non par modèle.

Dans cet exemple, un enregistrement de solde est enregistré en transaction même si la transaction est appelée dans la classe Account:

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

La méthode de transaction est également disponible en tant que méthode d'instance de modèle. Par exemple, vous pouvez également faire ceci:

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

Connexions de bases de données multiples

Une transaction agit sur une seule connexion à la base de données. Si vous avez plusieurs bases de données spécifiques à une classe, la transaction ne protégera pas l'interaction entre elles. Une solution consiste à commencer une transaction sur chaque classe dont vous modifiez les modèles:

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

C'est une solution médiocre, mais les transactions entièrement distribuées dépassent le cadre d'ActiveRecord.

sauvegarder et détruire sont automatiquement encapsulés dans une transaction

#Save et #destroy sont tous deux inclus dans une transaction qui garantit que tout ce que vous faites lors des validations ou des rappels se produira sous sa couverture protégée. Vous pouvez donc utiliser les validations pour vérifier les valeurs dont dépend la transaction ou vous pouvez élever les exceptions dans les rappels à la restauration, y `after_*` rappels `after_*` .

En conséquence, les modifications apportées à la base de données ne sont pas visibles en dehors de votre connexion tant que l'opération n'est pas terminée. Par exemple, si vous essayez de mettre à jour l'index d'un moteur de recherche dans `after_save` l'indexeur ne verra pas l'enregistrement mis à jour. Le rappel `after_commit` est le seul à être déclenché une fois la mise à jour `after_commit` .

Rappels

Il existe deux types de rappels associés aux transactions de `after_commit` et d' `after_rollback` : `after_commit` et `after_rollback` .

`after_commit` rappels `after_commit` sont appelés sur chaque enregistrement enregistré ou détruit dans une transaction immédiatement après la `after_commit` la transaction. `after_rollback` rappels `after_rollback` sont appelés sur chaque enregistrement enregistré ou détruit dans une transaction immédiatement après l'annulation de la transaction ou du point de sauvegarde.

Ces rappels sont utiles pour interagir avec d'autres systèmes, car vous serez assuré que le rappel n'est exécuté que lorsque la base de données est dans un état permanent. Par exemple, `after_commit` est un bon endroit pour effacer un cache, car sa suppression dans une transaction

peut déclencher la régénération du cache avant la mise à jour de la base de données.

Faire reculer une transaction

`ActiveRecord::Base.transaction` utilise l'exception `ActiveRecord::Rollback` pour distinguer une annulation délibérée d'autres situations exceptionnelles. En règle générale, déclencher une exception entraîne la méthode `.transaction` pour `.transaction` la transaction de base de données et transmettre l'exception. Mais si vous `ActiveRecord::Rollback` une exception `ActiveRecord::Rollback` , la transaction de base de données sera annulée, sans transmettre l'exception.

Par exemple, vous pouvez le faire dans votre contrôleur pour annuler une transaction:

```
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
```

Lire Transactions ActiveRecord en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/4688/transactions-activerecord

Chapitre 70: Transactions ActiveRecord

Introduction

Les transactions ActiveRecord sont des blocs protecteurs où la séquence des requêtes d'enregistrement actives n'est permanente que si elles peuvent toutes réussir en une seule action atomique.

Examples

Premiers pas avec les transactions d'enregistrement actives

Les transactions d'enregistrement actives peuvent être appliquées aux classes de modèle ainsi qu'aux instances de modèle. Les objets du bloc de transaction ne doivent pas nécessairement tous être des instances de la même classe. Cela est dû au fait que les transactions sont par connexion à la base de données, et non par modèle. Par exemple:

```
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
```

L'utilisation de save with a bang garantit que la transaction sera automatiquement annulée lorsque l'exception est lancée et après la restauration, le contrôle passe au bloc de secours pour l'exception. **Assurez-vous de sauver les exceptions lancées lors de la sauvegarde! dans le bloc de transaction.**

Si vous ne voulez pas utiliser save !, vous pouvez manuellement relancer `raise ActiveRecord::Rollback` lorsque la sauvegarde échoue. Vous n'avez pas besoin de gérer cette exception. Il annulera alors la transaction et amènera le contrôle à l'instruction suivante après le bloc de transaction.

```
   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"
```

Lire Transactions ActiveRecord en ligne: https://riptutorial.com/fr/ruby-on-

rails/topic/9326/transactions-activerecord

Chapitre 71: Turbolinks

Introduction

Turbolinks est une bibliothèque JavaScript qui accélère la navigation dans votre application Web. Lorsque vous suivez un lien, Turbolinks extrait automatiquement la page, échange son <body> et fusionne son <head>, sans avoir à payer le coût d'un chargement complet de la page.

Remarques

En tant que développeur de rails, vous interagirez probablement de manière minimale avec les turbolinks pendant votre développement. Il s'agit cependant d'une bibliothèque importante à connaître car elle peut être à l'origine de bogues difficiles à trouver.

Points à retenir:

- Lier à l' `turbolinks:load` au lieu de l'événement `document.ready`
- Utilisez l'attribut `data-turbolinks-false` pour désactiver la fonctionnalité turbolink par liaison.
- Utilisez l'attribut `data-turbolinks-permanent` pour conserver les éléments sur les chargements de page et éviter les bogues liés au cache.

Pour un traitement plus approfondi des turbolinks, visitez le dépôt officiel de github .

Le crédit de la majeure partie de cette documentation va aux personnes qui ont rédigé la documentation de turbolinks sur le dépôt github.

Examples

Liaison au concept de turbolink d'un chargement de page

Avec les turbolinks, l'approche traditionnelle de l'utilisation de:

```
$(document).ready(function() {
  // awesome code
});
```

ne marchera pas Lors de l'utilisation de turbolinks, l'événement `$(document).ready()` ne se déclenche qu'une seule fois: lors du chargement initial de la page. À partir de ce moment, chaque fois qu'un utilisateur clique sur un lien de votre site Web, turbolinks intercepte l'événement link click et effectue une requête ajax pour remplacer la balise <body> et fusionner les balises <head>. L'ensemble du processus déclenche la notion de "visite" dans les terrains turbolins. Par conséquent, au lieu d'utiliser la syntaxe traditionnelle `document.ready()` ci-dessus, vous devrez vous connecter à l'événement de visite de turbolink comme suit:

```
// pure js
document.addEventListener("turbolinks:load", function() {
  // awesome code
});

// jQuery
$(document).on('turbolinks:load', function() {
  // your code
});
```

Désactiver les turbolinks sur des liens spécifiques

Il est très facile de désactiver les turbolinks sur des liens spécifiques. Selon la documentation officielle de turbolinks :

> Turbolinks peut être désactivé pour chaque lien en annotant un lien ou l'un de ses ancêtres avec data-turbolinks = "false".

Exemples:

```
// 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>
```

Comprendre les visites d'application

Les visites d'application sont lancées en cliquant sur un lien Turbolinks-enabled ou par programme en appelant

```
Turbolinks.visit(location)
```

Par défaut, la fonction de visite utilise l'action "avance". De manière plus compréhensible, le comportement par défaut de la fonction de visite consiste à avancer vers la page indiquée par le paramètre "location". Chaque fois qu'une page est visitée, turbolinks envoie une nouvelle entrée dans l'historique du navigateur en utilisant `history.pushState` . L'historique est important car les turbolinks essaieront d'utiliser l'historique pour charger les pages du cache chaque fois que cela est possible. Cela permet un rendu de page extrêmement rapide pour les pages fréquemment visitées.

Toutefois, si vous souhaitez visiter un emplacement sans insérer d'historique dans la pile, vous pouvez utiliser l'action «remplacer» de la fonction de visite comme suit:

```
// using links
<a href="/edit" data-turbolinks-action="replace">Edit</a>

// programatically
Turbolinks.visit("/edit", { action: "replace" })
```

Cela remplacera la partie supérieure de la pile d'historique par la nouvelle page afin que le nombre total d'éléments sur la pile reste inchangé.

Il y a également une action de "restauration" qui aide à la restauration , les visites résultant du fait que l'utilisateur clique sur le bouton Suivant ou sur le bouton Précédent de son navigateur. Turbolinks gère ces types d'événements en interne et recommande aux utilisateurs de ne pas modifier manuellement le comportement par défaut.

Annulation des visites avant qu'elles ne commencent

Turbolinks fournit un écouteur d'événement qui peut être utilisé pour empêcher les visites de se produire. Écoutez les `turbolinks:before-visit` événement `turbolinks:before-visit` pour être averti lorsqu'une visite est sur le point de commencer.

Dans le gestionnaire d'événements, vous pouvez utiliser:

```
// pure javascript
event.data.url
```

ou

```
// jQuery
$event.originalEvent.data.url
```

récupérer le lieu de la visite. La visite peut alors être annulée en appelant:

```
event.preventDefault()
```

REMARQUE:

Selon les documents officiels de turbolinks :

> Les visites de restauration ne peuvent être annulées et ne tirent pas sur les turbolinks: avant la visite.

Des éléments persistants sur les chargements de pages

Considérez la situation suivante: Imaginez que vous développiez un site Web de médias sociaux

permettant aux utilisateurs d'être amis avec d'autres utilisateurs et utilisant des turbolinks pour accélérer le chargement des pages. En haut à droite de chaque page du site, il y a un numéro indiquant le nombre total d'amis qu'un utilisateur possède actuellement. Imaginez que vous utilisez votre site et que vous avez 3 amis. Chaque fois qu'un nouvel ami est ajouté, vous avez un javascript qui s'exécute qui met à jour le compteur d'amis. Imaginez que vous veniez d'ajouter un nouvel ami et que votre javascript soit exécuté correctement et que le nombre d'amis dans la partie supérieure droite de la page soit maintenant à jour. Maintenant, imaginez que vous cliquez sur le bouton Précédent du navigateur. Lorsque la page se charge, vous remarquez que le compteur d'amis indique 3, même si vous avez quatre amis.

Il s'agit d'un problème relativement courant pour lequel turbolinks a fourni une solution. Le problème est dû au fait que turbolinks charge automatiquement les pages du cache lorsqu'un utilisateur clique sur le bouton Précédent. La page mise en cache ne sera pas toujours mise à jour avec la base de données.

Pour résoudre ce problème, imaginez que vous rendiez le nombre d'amis dans une balise avec un identifiant "ami-nombre":

```
<div id="friend-count" data-turbolinks-permanent>3 friends</div>
```

En ajoutant l'attribut `data-turbolinks-permanent` , vous indiquez à turbolinks de conserver certains éléments sur les chargements de page. Les documents officiels disent :

> Désignez les éléments permanents en leur attribuant un identifiant HTML et en les annotant avec data-turbolinks-permanent. Avant chaque rendu, Turbolinks associe tous les éléments permanents par ID et les transfère de la page d'origine à la nouvelle page, en conservant leurs écouteurs de données et d'événement.

Lire Turbolinks en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/9331/turbolinks

Chapitre 72: Utilisation de GoogleMaps avec Rails

Examples

Ajouter la balise javascript google maps à l'en-tête de la mise en page

Pour que Google Maps fonctionne correctement avec turbolinks , ajoutez le tag javascript directement dans l'en-tête de la présentation plutôt que de l'inclure dans une vue.

```
# app/views/layouts/my_layout.html.haml
!!!
%html{:lang => 'en'}
  %head
    - # ...
    = google_maps_api_script_tag
```

`google_maps_api_script_tag` est mieux défini dans un assistant.

```
# 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
```

Vous pouvez enregistrer votre application avec google et obtenir votre clé API dans la console google api . Google a un petit guide sur la façon de demander une clé d'API pour l'API JavaScript de Google Maps .

La clé api est stockée dans le fichier `secrets.yml` :

```
# config/secrets.yml
development:
  google_maps_api_key: '...'
  # ...
production:
  google_maps_api_key: '...'
  # ...
```

N'oubliez pas d'ajouter `config/secrets.yml` à votre fichier `.gitignore` et assurez-vous de ne pas valider la clé api dans le dépôt.

Géocoder le modèle

Supposons que vos utilisateurs et / ou groupes ont des profils et que vous souhaitez afficher les champs de profil d'adresse sur une carte google.

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # Attributes:
  # label, e.g. "Work address"
  # value, e.g. "Willy-Brandt-Straße 1\n10557 Berlin"
end
```

Le géocodeur est un excellent moyen de géocoder les adresses, c'est-à-dire de fournir la `longitude` et la `latitude` .

Ajoutez le géocodeur à votre `Gemfile` et exécutez le `bundle` pour l'installer.

```
# Gemfile
gem 'geocoder', '~> 1.3'
```

Ajouter des colonnes de base de données pour la `latitude` et la `longitude` afin de sauvegarder l'emplacement dans la base de données. C'est plus efficace que d'interroger le service de géocodage chaque fois que vous avez besoin de l'emplacement. C'est plus rapide et la limite de requête n'est pas atteinte si rapidement.

```
➜ 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
```

Ajoutez le mécanisme de géocodage à votre modèle. Dans cet exemple, la chaîne d'adresse est stockée dans l'attribut `value` . Configurez le géocodage pour qu'il fonctionne lorsque l'enregistrement a changé et que seule une valeur est présente:

```
# 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
```

Par défaut, geocoder utilise google comme service de recherche. Il a beaucoup de fonctionnalités intéressantes comme les calculs de distance ou la recherche de proximité. Pour plus d'informations, consultez le fichier README du géocodeur .

Afficher les adresses sur une carte google dans la vue de profil

Dans la vue de profil, affichez les champs de profil d'un utilisateur ou d'un groupe dans une liste,

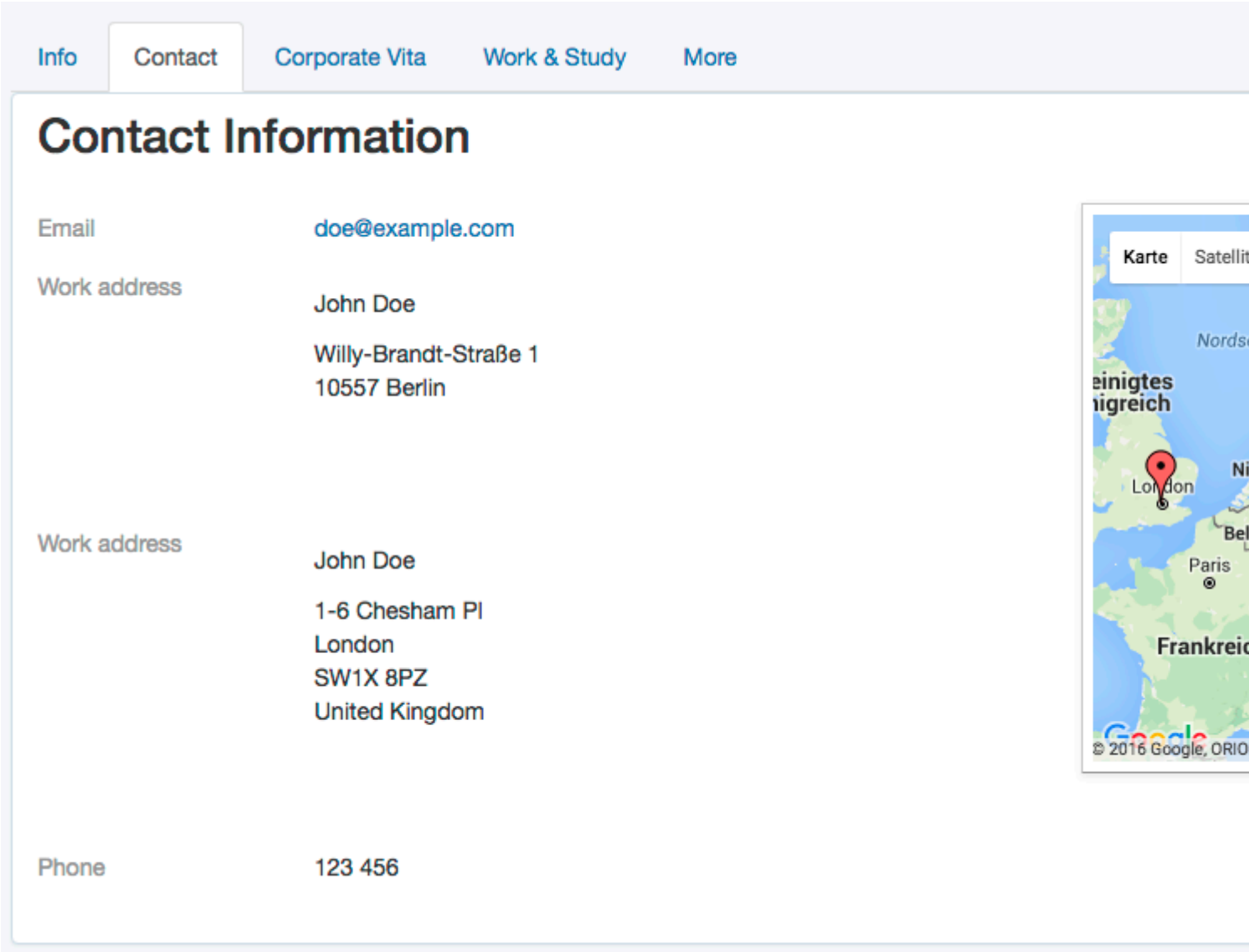
ainsi que les champs d'adresse sur une carte Google.

```
- # app/views/profiles/show.html.haml
%h1 Contact Information
.profile_fields
  = render @profile_fields
.google_map{data: address_fields: @address_fields.to_json }
```

Les `@profile_fields` et `@address_fields` appropriés sont définis dans le contrôleur:

```
# 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
```

Initialisez la carte, placez les marqueurs, définissez le zoom et les autres paramètres de la carte avec JavaScript.



Définir les marqueurs sur la carte avec javascript

Supposons qu'il y ait un div `.google_map` , qui deviendra la carte, et qui comporte les champs d'adresse à afficher en tant que marqueurs en tant qu'attribut de `data` .

Par exemple:

```
<!-- 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>
```

Pour utiliser l'événement `$(document).ready` avec turbolinks sans gérer les événements turbolinks à la main, utilisez la gem jquery.turbolinks .

Si vous souhaitez effectuer d'autres opérations avec la carte, ultérieurement, par exemple, des fenêtres de filtrage ou des fenêtres d'informations, il est pratique de faire gérer la carte par une classe de script café .

```
# app/assets/javascripts/google_maps.js.coffee
window.App = {} unless App?
class App.GoogleMap
  constructor: (map_div)->
    # TODO: initialize the map
    # TODO: set the markers
```

Lorsque vous utilisez plusieurs fichiers de script de café, qui sont placés par nom par défaut, il est pratique de définir un espace de noms `App` global, partagé par tous les fichiers de script de café.

Ensuite, parcourez (éventuellement plusieurs) div `.google_map` et créez une instance de la classe `App.GoogleMap` pour chacun d'eux.

```
# 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
```

Initialisez la carte en utilisant une classe de script café.

Si vous `App.GoogleMap` une classe de script `App.GoogleMap` , la carte Google peut être initialisée comme `App.GoogleMap` :

```
# 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
  }
```

Pour en savoir plus sur les options possibles de `map_configuration` , consultez la documentation de Google MapOptions et leur guide sur l'ajout d'éléments de contrôle .

Pour référence, la classe `google.maps.Map` est largement documentée ici .

Initialiser les marqueurs de carte à l'aide d'une classe de script café

Si une classe de script `App.GoogleMap` Coffee et les informations de marqueur sont stockées dans l'attribut `data-address-fields` du `.google_map` div, les marqueurs de carte peuvent être initialisés sur la carte comme `.google_map` :

```
# 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
```

Pour en savoir plus sur les options de marqueur, consultez la documentation MarkerOptions de Google et son guide des marqueurs .

Zoom automatique sur une carte en utilisant une classe de script café

Avec une classe de script `App.GoogleMap` Coffee avec `google.maps.Map` stockée en tant que `@map` et `google.maps.Marker` stockée en tant que `@markers` , la carte peut être agrandie automatiquement, c'est-à-dire ajustée pour que tous les marqueurs soient visibles, comme ceci: : sur la carte comme ceci:

```
# 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
```

Pour en savoir plus sur les limites, consultez la documentation de Google LatLngBounds .

Exposition des propriétés du modèle en json

Pour afficher les champs de profil d'adresse sous forme de marqueurs sur une carte google, les objets du champ d'adresse doivent être transmis en tant qu'objets json à JavaScript.

Attributs de base de données réguliers

Lors de l'appel à `to_json` sur un objet `ApplicationRecord` , les attributs de la base de données sont automatiquement exposés.

Étant donné un modèle `ProfileFields::Address` avec des attributs `label` , `value` , `longitude` et `latitude` , `address_field.as_json` produit un `Hash` , par exemple une représentation,

```
address_field.as_json  # =>
  {label: "Work address", value: "Willy-Brandt-Straße 1\n10557 Berlin",
    longitude: ..., latitude: ...}
```

qui est converti en chaîne json par `to_json` :

```
address_field.to_json  # =>
  "{\"label\":\"Work address\",\"value\":\"Willy-Brandt-Straße 1\\n
    10557 Berlin\",\"longitude\":...,\"latitude\":...}"
```

Ceci est utile car il permet d'utiliser plus tard l' `label` et la `value` en javascript, par exemple pour afficher des info-bulles pour les marqueurs de carte.

Autres attributs

D'autres attributs virtuels peuvent être exposés en `as_json` méthode `as_json` .

Par exemple, pour exposer un attribut `title` , incluez-le dans le hachage `as_json` fusionné:

```
# 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
```

L'exemple ci-dessus utilise `super` pour appeler la méthode `as_json` origine, qui renvoie l'attribut d'origine de l'objet et le fusionne avec la position requise.

Pour comprendre la différence entre `as_json` et `to_json` , consultez cet article de jjulian .

Position

Pour rendre les marqueurs, Google Maps api nécessite, par défaut, une `position` hachée qui

contient respectivement la longitude et la latitude `lng` et `lat` .

Ce hachage de position peut être créé dans JavaScript, plus tard, ou ici lors de la définition de la représentation json du champ d'adresse:

Pour fournir cette `position` tant qu'attribut json du champ d'adresse, remplacez simplement la méthode `as_json` sur le modèle.

```
# 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
```

Lire Utilisation de GoogleMaps avec Rails en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2828/utilisation-de-googlemaps-avec-rails

Chapitre 73: Validations ActiveRecord

Examples

Valider la numéricité d'un attribut

Cette validation limite l'insertion de valeurs numériques uniquement.

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

Outre `:only_integer` , cet assistant accepte également les options suivantes pour ajouter des contraintes aux valeurs acceptables:

- `:greater_than` - Spécifie que la valeur doit être supérieure à la valeur fournie. Le message d'erreur par défaut pour cette option est "doit être supérieur à% {count}".
- `:greater_than_or_equal_to` - Spécifie que la valeur doit être supérieure ou égale à la valeur fournie. Le message d'erreur par défaut pour cette option est "doit être supérieur ou égal à% {count}".
- `:equal_to` - Spécifie que la valeur doit être égale à la valeur fournie. Le message d'erreur par défaut pour cette option est "doit être égal à% {count}".
- `:less_than` - Spécifie que la valeur doit être inférieure à la valeur fournie. Le message d'erreur par défaut pour cette option est "doit être inférieur à% {count}".
- `:less_than_or_equal_to` - Spécifie que la valeur doit être inférieure ou égale à la valeur fournie. Le message d'erreur par défaut pour cette option est "doit être inférieur ou égal à% {count}".
- `:other_than` - Spécifie que la valeur doit être différente de la valeur fournie. Le message d'erreur par défaut pour cette option est "doit être différent de% {count}".
- `:odd` - Spécifie que la valeur doit être un nombre impair si elle est définie sur true. Le message d'erreur par défaut pour cette option est "doit être impair".
- `:even` - Spécifie que la valeur doit être un nombre pair si elle est définie sur true. Le message d'erreur par défaut pour cette option est "doit être pair".

  Par défaut, la numéricité n'autorise aucune valeur nulle. Vous pouvez utiliser l'option allow_nil: true pour l'autoriser.

Valider l'unicité d'un attribut

Cet assistant vérifie que la valeur de l'attribut est unique juste avant que l'objet soit enregistré.

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

Il existe une option `:scope` que vous pouvez utiliser pour spécifier un ou plusieurs attributs utilisés pour limiter la vérification de l'unicité:

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

Il existe également une option `:case_sensitive` que vous pouvez utiliser pour définir si la contrainte d'unicité doit être sensible à la casse ou non. Cette option est définie par défaut sur `true` .

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

Valider la présence d'un attribut

Cet assistant vérifie que les attributs spécifiés ne sont pas vides.

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

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

Vous pouvez utiliser l'assistant d' `absence` pour valider l' `absence` des attributs spécifiés. Il utilise le `present?` méthode pour vérifier les valeurs nulles ou vides.

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

Remarque: Si l'attribut est `boolean` , vous ne pouvez pas utiliser la validation de présence habituelle (l'attribut ne serait pas valide pour une valeur `false`). Vous pouvez le faire en utilisant une validation d'inclusion:

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

Saut des validations

Utilisez les méthodes suivantes si vous souhaitez ignorer les validations. Ces méthodes enregistreront l'objet dans la base de données même s'il n'est pas valide.

- décrémenter!
- decrement_counter
- incrément!
- increment_counter
- basculer!

- toucher
- Tout mettre à jour
- update_attribute
- update_column
- update_columns
- update_counters

Vous pouvez également ignorer la validation lors de l'enregistrement en transmettant `validate` comme argument pour `save`

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

Valider la longueur d'un attribut

```
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
```

Les options possibles de contrainte de longueur sont les suivantes:

- `:minimum` - L'attribut ne peut pas avoir moins que la longueur spécifiée.
- `:maximum` - L'attribut ne peut pas avoir plus que la longueur spécifiée.
- `:in` (or `:within`) - La longueur de l'attribut doit être incluse dans un intervalle donné. La valeur de cette option doit être une plage.
- `:is` - La longueur de l'attribut doit être égale à la valeur donnée.

Validation groupée

Parfois, il est utile d'avoir plusieurs validations utilisant une seule condition. Cela peut être facilement réalisé en utilisant with_options.

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

Toutes les validations à l'intérieur du bloc with_options auront automatiquement passé la condition si:: is_admin?

Validations personnalisées

Vous pouvez ajouter vos propres validations en ajoutant de nouvelles classes héritant d' `ActiveModel::Validator` ou d' `ActiveModel::EachValidator` . Les deux méthodes sont similaires mais fonctionnent différemment:

`ActiveModel::Validator` et `validates_with`

Implémentez la méthode `validate` qui prend un enregistrement en argument et effectue la validation sur celui-ci. Ensuite, utilisez `validates_with` avec la classe sur le modèle.

```
# 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` et `validate`

Si vous préférez utiliser votre nouveau validateur en utilisant la méthode `validate` commune sur un seul paramètre, créez une classe héritant d' `ActiveModel::EachValidator` et implémentez la méthode `validate_each` qui prend trois arguments: `record` , `attribute` et `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
```

Plus d'informations sur les guides Rails .

Valide le format d'un attribut

Validez que la valeur d'un attribut correspond à une expression régulière utilisant le `format` et l'option `with` .

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

Vous pouvez également définir une constante et définir sa valeur sur une expression régulière et la transmettre à l'option `with:` :. Cela pourrait être plus pratique pour les expressions régulières très complexes

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

Le message d'erreur par défaut `is invalid` . Cela peut être changé avec l'option `:message` .

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

L'inverse répond également, et vous pouvez spécifier qu'une valeur *ne* doit *pas* correspondre à une expression régulière avec l'option `without:`

Valide l'inclusion d'un attribut

Vous pouvez vérifier si une valeur est incluse dans un tableau en utilisant l' `inclusion:` helper. L'option `:in` et son alias `:within` indiquent l'ensemble des valeurs acceptables.

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

Pour vérifier si une valeur n'est pas incluse dans un tableau, utilisez l' `exclusion:` helper

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

Validation conditionnelle

Parfois, vous devrez peut-être valider un enregistrement uniquement sous certaines conditions.

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

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

Si vous êtes conditionnel, vous pouvez utiliser un Proc:

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

Pour conditionnel négatif, vous pouvez utiliser à `unless` :

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

Vous pouvez également passer une chaîne, qui sera exécutée via `instance_eval` :

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

Confirmation de l'attribut

Vous devez l'utiliser lorsque vous avez deux champs de texte qui doivent recevoir exactement le même contenu. Par exemple, vous souhaiterez peut-être confirmer une adresse e-mail ou un mot de passe. Cette validation crée un attribut ***virtuel*** dont le nom est le nom du champ à confirmer avec `_confirmation` ajouté.

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

Remarque Cette vérification est effectuée uniquement si `email_confirmation` n'est pas nul.

Pour demander une confirmation, veillez à ajouter un contrôle de présence pour l'attribut de confirmation.

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

La source

Utiliser: en option

L'option `:on` vous permet de spécifier quand la validation doit avoir lieu. Le comportement par défaut de tous les assistants de validation intégrés doit être exécuté lors de l'enregistrement (à la fois lorsque vous créez un nouvel enregistrement et lorsque vous le mettez à jour).

```
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
```

Lire Validations ActiveRecord en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/2105/validations-activerecord

Chapitre 74: Verrouillage ActiveRecord

Examples

Verrouillage optimiste

```
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
```

Verrouillage pessimiste

```
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
```

Lire Verrouillage ActiveRecord en ligne: https://riptutorial.com/fr/ruby-on-rails/topic/3866/verrouillage-activerecord

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec 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	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
8	ActiveSupport	Adam Lassek
9	Ajout d'une application Amazon RDS à votre application rails	Sathishkumar Jayaraj
10	Ajouter un panneau d'administration	Ahsan Mahmood, MSathieu
11	API Rails	Adam Lassek, hschin
12	Associations ActiveRecord	giniouxe, Hardik Upadhyay, Khanh Pham, Luka Kerr, Manish Agarwal, Niyanta, RareFever, Raynor Kuang, Sapna Jindal
13	Authentification de	HParker

	Rails 5 API	
14	Authentification utilisateur dans Rails	Abhinay, Ahsan Mahmood, Antarr Byrd, ArtOfCode, dgilperez, Kieran Andrews, Luka Kerr, Qchmqs, uzaif,
15	Authentifier Api en utilisant Devise	Vishal Taj PM
16	Autorisation avec CanCan	4444, Ahsan Mahmood, dgilperez, mlabarca, toobulkeh
17	Colonnes multi-usages ActiveRecord	Fabio Ros
18	Configuration	Ali MasudianPour, Undo
19	Configurer Angular avec Rails	B8vrede, Rory O'Kane, Umar Khan
20	Conventions de nommage	Andrey Deineko, Atul Khanduri, br3nt, Flambino, giniouxe, hgsongra, Luka Kerr, Marko Kacanski, Muhammad Abdullah, Sven Reuter, Xinyang Li
21	Crevettes PDF	Awais Shafqat
22	Déploiement d'une application Rails sur Heroku	B Liu, hschin
23	Des vues	danirod, dgilperez, elasticman, Luka Kerr, MikeC, MMachinegun , Pragash, RareFever
24	Elasticsearch	Don Giovanni, Luc Boissaye
25	Emplois actifs	tirdadc
26	Enregistreur de rails	Alejandro Montilla, hgsongra
27	Fonctionnalité de paiement dans les rails	ppascualv, Sathishkumar Jayaraj
28	Form Helpers	aisflat439, owahab, Richard Hamilton, Simon Tsang, Slava.K
29	Formulaire imbriqué en Ruby on Rails	Arslan Ali
30	Gemmes	Deep, hschin, ma_il, MMachinegun, RamenChef
31	Héritage de table unique	Niyanta, Ruslan, Slava.K, toobulkeh, Vishal Taj PM

32	I18n - Internationalisation	Cyril Duchon-Doris, Francesco Lupo Renzi, Frederik Spang, gwcodes, Jorge Najera T, Lahiru, RamenChef
33	Identification amicale	Thang Le Sy
34	Importer des fichiers CSV entiers à partir d'un dossier spécifique	fool
35	Intégration de React.js avec Rails à l'aide d'Hyperloop	Mitch VanDuyn
36	Interface de requête ActiveRecord	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
37	Le débogage	Adam Lassek, Dénes Papp, Dharam, Kelseydh, sa77, titan
38	Le modèle indique: AASM	Lomefin
39	Le routage	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,
40	Les frameworks Rails au fil des ans	Shivasubramanian A
41	Migrations ActiveRecord	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
42	Mise à niveau des rails	hschin, michaelpri, Rodrigo Argumedo

43	Mise en cache	ArtOfCode, Cuisine Hacker, Khanh Pham, RamenChef, tirdadc
44	Modifier le fuseau horaire par défaut	Mihai-Andrei Dinculescu
45	Modifier un environnement d'application Rails par défaut	Whitecat
46	Mongoïde	Ryan K, tes
47	Moteur Rails - Rails Modulaires	Mayur Shah
48	Motif de décorateur	Adam Lassek
49	Mots réservés	Emre Kurt
50	Organisation de classe	Deep, hadees, HParker
51	Outils pour l'optimisation du code Ruby on Rails et le nettoyage	Akshay Borade
52	Ouvrière	Rafael Costa
53	Pipeline d'actifs	fybw id, Robin
54	Rails 5	thiago araujo
55	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
56	Rails Cookbook - Advanced rails recettes / apprentissages et techniques de codage	Milind
57	Rails -Engines	Deepak Kabbur
58	Rails génèrent des commandes	Adam Lassek, ann, Deepak Mahakale, Dharam, Hardik Upadhyay, jackerman09, Jeremy Green, marcamillion, Milind, Muhammad Abdullah, nomatteus, powerup7, Reub, Richard

		Hamilton
59	Rails sur docker	ppascualv, Sathishkumar Jayaraj
60	Réagir avec les rails en utilisant la pierre de réaction	Kimmo Hintikka, tirdadc
61	Routage peu profond	Darpan Chhatravala
62	RSpec et Ruby on Rails	Ashish Bista, Scott Matthewman, Simone Carletti
63	Safe Constantize	Eric Bouchut, Ryan K
64	Serializers Modèle Actif	Flip, owahab
65	Stockage sécurisé des clés d'authentification	DawnPaladin
66	Téléchargement de fichier	Sergey Khmelevskoy
67	Test d'applications Rails	HParker
68	Transactions ActiveRecord	abhas, Adam Lassek
69	Turbolinks	Mark
70	Utilisation de GoogleMaps avec Rails	fiedl
71	Validations ActiveRecord	Adam Lassek, Colin Herzog, Deepak Mahakale, dgilperez, dodo121, giniouxe, Hai Pandu, Hardik Upadhyay, mmichael, Muhammad Abdullah, pablofullana, Richard Hamilton
72	Verrouillage ActiveRecord	Adam Lassek, fatfrog, Muaaz Rafi