Writing a web application with Ruby on Rails

Unless you have lived under a rock for the past 5 years or so, you should have already heard of Ruby on Rails. In this article, we'll write a very simple web application to show its power and speed of development.

We'll cover the following topics:

- install Ruby
- install the rails gem
- how to start a new rails project
- directory structure of a rails project
- how to manage data with a database
- use already available components to handle common tasks

- add CRUD resources
- improve the app flawlessy

Quick setup

The following section offers a very quick overview of the tools you need to get started with Ruby on Rails development. If you already have a working rails environment, excluding the default ruby installation that some operating systems have, then feel free to skip the steps.

Install Ruby

If you run Linux or MacOSX on your computer, you may already have Ruby installed by default, however these kind of installations are out of date (eg: ruby 1.8.x) and *gems* are difficult to manage. The best way is to install it manually or just use some tool that helps to manage custom Ruby installations, even multiple versions (eg: 1.9.x and 2.0.0 releases).

Instead of repeating what others have already said, here are some useful guides:

- MacOSX: Ruby on Rails development with Mac OS X Mountain Lion. Follow the steps until you reach *Install SQLite3* section, we'll see here how to install rails.
- Linux: Setup Ruby On Rails on Ubuntu 13.04. Same as above, follow the steps until you reach *Configuring Git*.
- Windows: SQLite3 Ruby 2.0 on Windows and Getting Started with Git and GitHub on Windows. I haven't too much experience on windows, so keep the finger crossed and/or ask on the same forum thread. The Ruby community is very kind 🙂

Now, you should have all the requirements to get started.

Install Bundler gem

Bundler is a Ruby gem that helps you to manage development dependencies in a project. We'll see how it works later, for now, just install it:

```
gem install bundler
```

Install Rails v4.0.0.rc2 gem

We're going to use the upcoming new version of rails. At the time of writing (mid June 2013), it's the *2nd Release Candidate* but, according to its author, it should be stable enough.

```
gem install rails -v4.0.0.rc2
```

A Simple Web Application to Manage and Share Bookmarks

Someone says that *doing it is the best way to learn*, so here's a *less common* tutorial app: a simple platform to manage or share bookmarks. It's nothing fancy, but it's enough to apply some basic concepts. Moreover, aren't you bored of *todo lists* and *blog engines* over and over? 🙂

Create the rails app

When you install rails, it comes with a bunch of other libraries and/or command line tools. The main one is `rails`. For now, we'll use it to generate the initial skeleton:

```
rails new bookmarks -T -d sqlite3 -B
```

I've also passed some arguments to the command:

- `bookmarks`: is the name of our project.
- `-T`: skip `Test::Unit` files. It's not fundamental for our purpose (but they are in the everyday Ruby *and* Rails development), so I skipped them because *testing* is an argument apart and I don't want to overcomplicate this article.
- `-d sqlite3`: we want to use SQLite database. It fits perfectly for our purposes. Please note that, Rails has support for a wide variety of databases, even NoSQL ones. It's also common to have different database engines based on the environment (eg: SQLite for development, MySQL or PostgreSQL for production, etc...) without code changes.
- `-B`: don't run `bundle install`, we'll do it manually later.

Once you run the command, you should see a long list of lines that explain what is happening:

```
andrea@mbair ~/Works/12dos % rails new
bookmarks -T -d sqlite3 -B
    create
    create  README.rdoc
    create  Rakefile
    create  config.ru
    create  .gitignore
    create  Gemfile
    create  app
```

```
    [...]
andrea@mbair ~/Works/12dos % cd bookmarks
andrea@mbair ~/Works/12dos/bookmarks %
```

A new directory called `bookmarks` (or whatever project name you passed) has been created, along with a set of files and directories inside it. Enter this directory, we'll see the important contents during the article.

A first look at Bundler and Gemfile

We mentioned Bundler as a tool to manage development dependencies in a Ruby app. It works by reading a file called `Gemfile` that should be present in the root directory of the app. You need this to make sure all the required gems are installed, even rails itself. Here's a brief look at the Gemfile generated by the `rails new` command:

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

# Our rails version
gem 'rails', '4.0.0.rc2'

# Use sqlite3 as the database for Active
Record
gem 'sqlite3'

# Use SCSS for stylesheets
gem 'sass-rails', '~> 4.0.0.rc2'

# Use Uglifier as compressor for JavaScript
assets
gem 'uglifier', '>= 1.3.0'
```

```
# Use CoffeeScript for .js.coffee assets and
views
gem 'coffee-rails', '~> 4.0.0

# Uncomment this if you haven't a nodejs and
coffeescript installed
# gem 'therubyracer', platforms: :ruby

[...]
```

As you can see, this is the default list of gems you need to start working on a Rails project. Of course, you'll change it very often to add, remove or upgrade gems.

Now you can finally run `bundle install` (the command we skipped by specifying `-B`):

```
andrea@mbair ~/Works/12dos/bookmarks %
bundle install
Fetching gem metadata from
https://rubygems.org/.........
Fetching gem metadata from
https://rubygems.org/..
Resolving dependencies...
Using rake (10.0.4)
Installing i18n (0.6.4)
[...]
Your bundle is complete!
Use `bundle show [gemname]` to see where a
bundled gem is installed.
andrea@mbair ~/Works/12dos/bookmarks %
```

If you use *git*, this is a good moment to initialize the repo and proceed with the initial commit.

Run the development server

Even if you haven't yet wrote any line of code, you can already use the Rails development server. Run the following command:

```
andrea@mbair ~/Works/12dos/bookmarks % rails
server
=> Booting WEBrick
=> Rails 4.0.0.rc2 application starting in
development on http://0.0.0.0:3000
=> Run `rails server -h` for more startup
options
=> Ctrl-C to shutdown server
[2013-06-14 16:10:48] INFO  WEBrick 1.3.1
[2013-06-14 16:10:48] INFO  ruby 1.9.3
(2012-04-20) [x86_64-darwin12.2.0]
[2013-06-14 16:10:48] INFO
WEBrick::HTTPServer#start: pid=1174 port=3000
[...]
```

then point your browser to http://localhost:3000 and you'll get a standard welcome page. This time we've used `rails server`, another useful command offered by Rails, moreover, it works only when you run it inside a Rails project folder. We'll see other useful commands later.

A quick introduction to MVC patterns and Rails

Rails is usually defined as an *MVC framework*. Basically, it means that the app behaviour is defined in this way:

- **Model:** manages data between the rest of the application and the database. You can define how a single entity behaves, this includes data validation, before/after save hooks, etc...
- **Controller:** this is the *glue* between the data managed by the Models and the rendered Views. Http requests that come to your app, are *routed* to a controller which usually will interact with one or more models and finally renders the ouput along with an Http response.
- **View:** the final output for a request. It's usually a piece of HTML, but it might also be JSON or XML. We'll see this later.

This is, of course, an over-simplification, but it should be enough to get a *vague idea* of how it works. I'll dive into some details later.

Add a Bookmark resource

The goal of our app is to manage bookmarks, right? So we'll need to create our `Bookmark` resource:

```
andrea@mbair ~/Works/12dos/bookmarks % rails
generate scaffold bookmark title:string
url:string
    invoke  active_record
    create    db/migrate
/20130614142337_create_bookmarks.rb
    create    app/models/bookmark.rb
    invoke  resource_route
     route    resources :bookmarks
    invoke  scaffold_controller
    create    app/controllers
/bookmarks_controller.rb
```

```
    [...]
```

This time, we've used `rails generate`, one of the most useful commands. In this case, we've generated a *scaffold* that creates all the necessary files and code to automatically get a full *MVC* stack that manages a *Bookmark*. In the next few paragraphs, we'll see in detail what the above command did.

Database migration

We've specified that a *Bookmark* record is made of a *title* and a *url* fields, both of type *string* (255 chars by default). So, the rails scaffold generator has automatically created a *migration* script to update your database schema: `db/migrate/db/migrate/20130614142337_create_bookmarks.rb`

```
class CreateBookmarks <
ActiveRecord::Migration
  def change
    create_table :bookmarks do |t|
      t.string :title
      t.string :url

      t.timestamps
    end
  end
end
```

Even if it's almost easy to read, this code basically means:

- create a *bookmarks* table on database (note the pluralized form)

- it has the fields *title* and *url* of type string
- also add timestmps by default (this is automatically translated to *created_at* and *updated_at* datetime fields)

However, your database still doesn't know about these changes to the schema, you need to run the migration task:

```
andrea@mbair ~/Works/12dos/bookmarks % rake
db:migrate
==  CreateBookmarks: migrating
================================================
===
-- create_table(:bookmarks)
   -> 0.0011s
==  CreateBookmarks: migrated (0.0012s)
=======================================
```

Rake is another common and useful tool in the Ruby world. It acts like a *Makefile* in a Ruby fashion. Rails comes with several *rake* tasks ready to use (you can also create custom ones, if needed):

```
andrea@mbair ~/Works/12dos/bookmarks % rake
-T
rake about                    # List versions
of all Rails frameworks and the environment
rake assets:clean             # Remove old
compiled assets
rake assets:clobber           # Remove
compiled assets
rake assets:environment       # Load asset
compile environment
```

```
[...]
```

The Bookmark model

The model representing a *bookmark* was created in `app/models/bookmark.rb`, at the moment it's almost empty in terms of code, but it already knows how to behave with the database. To demonstrate this, we'll use another Rails command to open a *console*:

```
andrea@mbair ~/Works/12dos/bookmarks % rails
console
Loading development environment (Rails
4.0.0.rc2)
>
```

Now, we have a shell to manually issue commands to our rails app. Let's see if it knows something about our *Bookmark*:

```
> Bookmark
=> Bookmark(id: integer, title: string, url:
string, user_id: integer, created_at:
datetime, updated_at: datetime)

> Bookmark.count
   (0.3ms)  SELECT COUNT(*) FROM "bookmarks"
=> 0
```

It knows about the *bookmarks* table, its fields and how to query the database. Ok, create one *Bookmark*:

```
> Bookmark.create(title: "Hello Bookmarks
```

```
app!", url: "http://localhost:3000")
   (0.1ms)  begin transaction
  SQL (7.7ms)  INSERT INTO "bookmarks"
("created_at", "title", "updated_at", "url")
VALUES (?, ?, ?, ?)  [["created_at", Fri, 14
Jun 2013 15:16:17 UTC +00:00], ["title",
"Hello Bookmarks app!"], ["updated_at", Fri,
14 Jun 2013 15:16:17 UTC +00:00], ["url",
"http://localhost:3000"]]
   (0.8ms)  commit transaction
=> #<Bookmark id: 1, title: "Hello Bookmarks
app!", url: "http://localhost:3000",
user_id: nil, created_at: "2013-06-14
15:16:17", updated_at: "2013-06-14 15:16:17

> Bookmark.count
   (0.3ms)  SELECT COUNT(*) FROM "bookmarks"
=> 1
```

The BookmarksController and its views

We said that the *scaffold generator* has created all the parts of the MVC stack, now is the time for the controller in `app/controllers/bookmarks_controller.rb`. You'll see a `BookmarksController` class and a bunch of methods (called *actions*). Each action corresponds to an HTTP PATH (eg: `/bookmarks`) and an HTTP verb (GET, POST, PUT, DELETE).

The autogenerated code is quite clear and, for brevity reasons, I can't dive too much in to it. The interesting parts are:

- nowhere in the code is specified how to render views. There's a reason for that: HTML views are based on the controller's actions names. For example, `BookmarksController#show`

will render a template placed in `app/views/bookmarks/show.html.erb`. Of course, only the actions that respond to *HTTP GET* have a template, the rest are usually redirects. Check the contents of `app/views/bookmarks/` yourself to get an idea.

- it looks like there's some automagical *JSON* support. That's true, rails supports *HTML* and *JSON* request by default, I'll show you this later.

Routes

The final component for the *MVC* stack is about routing the *HTTP* requests to the proper controller and action. To make this, the Rails generator has updated `config/routes.rb` file and has added the line `resources :bookmarks` that is a shorthand to say: *use REST routes for the the resource bookmark*. If you're wondering *what* routes are available, you can use the relative *rake task*:

```
andrea@mbair ~/Works/12dos/bookmarks % rake
routes
       Prefix Verb   URI
Pattern                    Controller#Action
    bookmarks GET
/bookmarks(.:format)          bookmarks#index
              POST
/bookmarks(.:format)
bookmarks#create
 new_bookmark GET    /bookmarks
/new(.:format)      bookmarks#new
edit_bookmark GET    /bookmarks
/:id/edit(.:format) bookmarks#edit
     bookmark GET    /bookmarks
```

```
/:id(.:format)      bookmarks#show
               PATCH  /bookmarks
/:id(.:format)      bookmarks#update
               PUT    /bookmarks
/:id(.:format)      bookmarks#update
               DELETE /bookmarks
/:id(.:format)      bookmarks#destroy
```

Try on the server

Now, the fun part. Restart your rails server with `rails server` and go to http://localhost:3000/bookmarks. If you followed the above example in console, you should already see the Bookmark created earlier, otherwise, you can create a new one by clicking on *New bookmark* link.

Try http://localhost:3000/bookmarks.json and see what you'll get. You still haven't wrote a single line of Ruby, and it already does a lot of things. Awesome, isn't it? 🙂

Adding users and authentication

Once we have the *Bookmark* resource, we need *Users* that own bookmarks and a way to authenticate them. The Ruby and Rails community is very active: whatever the task is, you'll probably find at least one gem to do it (if you don't, it might be a good idea to write one). Luckily, user authentication is a very common task, and there are several options. The most used is Devise, let's use it.

Installing Devise gem

Open `Gemfile` and add this line:

```
gem 'devise', '~> 3.0.0.rc'
```

We also told Bundler to use a version of Devise that should be *greater_or_equal* than its *minor version* (eg: 3.0.1 is ok, 3.1.0 isn't).

Run `bundle install` to install the gem:

```
andrea@mbair ~/Works/12dos/bookmarks %
bundle install
Resolving dependencies...
Using rake (10.0.4)
[...]
Installing devise (3.0.0.rc)
[...]
Your bundle is complete!
Use `bundle show [gemname]` to see where a
bundled gem is installed.
andrea@mbair ~/Works/12dos/bookmarks %
```

and run the Devise generator:

```
andrea@mbair ~/Works/12dos/bookmarks % rails
generate devise:install
      create  config/initializers/devise.rb
      create  config/locales/devise.en.yml
===============================================
===================================

Some setup you must do manually if you
haven't yet:

  1. Ensure you have defined default url
options in your environments files. Here
     is an example of default_url_options
```

```
appropriate for a development environment
     in config/environments/development.rb:


config.action_mailer.default_url_options = {
:host => 'localhost:3000' }

     In production, :host should be set to
the actual host of your application.

  2. Ensure you have defined root_url to
*something* in your config/routes.rb.
     For example:

       root :to => "home#index"

  3. Ensure you have flash messages in
app/views/layouts/application.html.erb.
     For example:

       <p class="notice"><%= notice %></p>
       <p class="alert"><%= alert %></p>

  4. If you are deploying Rails 3.1+ on
Heroku, you may want to set:


config.assets.initialize_on_precompile =
false

     On config/application.rb forcing your
application to not access the DB
     or load models when precompiling your
```

```
assets.

  5. You can copy Devise views (for
customization) to your app by running:

       rails g devise:views

==========================================
==================================
andrea@mbair ~/Works/12dos/bookmarks %
```

As you can see, it has created two files under `config/` directory:

- `config/initializers/devise.rb`: here you can change several Devise settings. However, we'll use defaults because they are enough.
- `config/locales/devise.en.yml`: this file contains i18n translations for Devise. Please note that `config/locales/` is the default path to put all the locales for a Rails app.

Devise also told us to check five steps to complete the setup:

- add `config.action_mailer.default_url_options = { :host => 'localhost:3000' }` to `config/environments/development.rb`
- define a *root_url* in `config/routes.rb`. Fow now, we can achieve this by pointing `BookmarksController#index` to the root of the site:
  Bookmarks::Application.routes.draw do
  # [...]
  root 'bookmarks#index'

```
# [...]
end
```

- add the two pieces of html in `app/views/layouts/application.html.erb`. This file is the main layout your Rails app.
- skip the fourth step because we're running a Rails v4.0.x app, not a 3.1.x one)
- generate the Devise default views with `rails generate devise:views`. They contain a basic scaffold to get Devise to work (eg: login/signup forms, etc...). Please note that Devise has several features concerning user authentication and registration, but we're going to use only a small set of them, so don't panic if you see a lot of generated views.

Generate a User model

Setup for Devise is done, but we still haven't a *User* model, we need to create one:

```
andrea@mbair ~/Works/12dos/bookmarks % rails
generate devise User
      invoke  active_record
      create    db/migrate
/20130617132545_devise_create_users.rb
      create    app/models/user.rb
      insert    app/models/user.rb
       route  devise_for :users
```

If you remember the scaffold generator we used to generate a *Bookmark*, this one is very similar even if just focused on the model, plus the route that tells Devise to handle `/users/*` paths.

In fact, it created a migration for users and the *User* class. Edit the migration to look like this (remove or comment the rest):

```
class DeviseCreateUsers <
ActiveRecord::Migration
  def change
    create_table(:users) do |t|
      ## Database authenticatable
      t.string :email,              :null =>
false, :default => ""
      t.string :encrypted_password, :null =>
false, :default => ""

      ## Rememberable
      t.datetime :remember_created_at

      t.timestamps
    end

    add_index :users, :email,
:unique => true
  end
end
```

Also the model needs to reflect the changes made to the migration:

```
class User < ActiveRecord::Base
  devise :database_authenticatable,
:registerable,
         :rememberable, :validatable
end
```

The last step, is to run the migration with `rake db:migrate`

In this way, we have a simple authentication system that lets a user to register, login and logout. Try to run `rake routes` and you'll see the new routes for these actions, all referred to *users*. You can also play in the Rails console to create some users.

Users have many Bookmarks

Now that we have *User* and *Bookmark*, we need to *associate* them. In this case, it's a *one to many* relationship. As a shared convention, foreign keys name are made of the model name suffixed by *_id*. Of course, we'll not change the already created migrations, instead, we're going to create a new one:

```
rails generate migration AddUserIdToBookmark
user_id:integer
```

Choosing an appropriate, arbitrary name is not mandatory, but it's a *best practice* that you should follow. Our generated migration will add a *user_id* integer column to the *bookmarks* table. Also, feel free to run the migration now.

At the moment, none of our two rails models knows about each other, we've only added a field on the database, but it's meaningless without some additional instruction:

- add `has_many :bookmarks` to `app/models/user.rb`
- add `belongs_to :user` to `app/models/user.rb`

With these changes, a `User` instance will automatically have a `bookmarks` method (and many others), referred to all *Bookmark* records with that specific *user id*.

Require authentication to manage your bookmarks

Looking at the models, it looks like the *data part* is complete and working. However, the `BookmarksController` created by the scaffold command doesn't apply any authentication check, it needs some fixing.

First of all, check user authentication before executing whatever action:

```
class BookmarksController <
ApplicationController
  # other before_actions
  before_action :authenticate_user!

  # [...] actions
end
```

This means that, from now on, each request that arrives to the `BookmarksController` will first check if a user is authenticated (`authenticate_user!` is a method provided by Devise), otherwise it will be redirected to the login page.

Then, access only bookmarks owned by the authenticated user. To do this, replace `Bookmark` occurrencies with `current_user.bookmarks`. As you may wonder, `current_user` is an object provided by Devise that represents the authenticated user, while `.bookmarks` is the method provided by the model association between *User* and *Bookmark* models.

Try on the server

If you haven't played with the web interface yet, now is a good time to do it. Here are some hints:

- go to http://localhost:3000, it should redirect to http://localhost:3000/users/sign_in, the login page;
- if you haven't registered any user, then go to http://localhost:3000/users/sign_up and register a new one;
- retry to go to http://localhost:3000/ again, it will work as well as going to http://localhost:3000/bookmarks because they're the same action;
- create some bookmarks from http://localhost:3000/bookmarks/new.

Handling wrong inputs

At the moment, we have several areas where a wrong user input might damage you application:

- providing incomplete or malformed data when creating a new Bookmark
- requesting a bookmark id that doesn't exist as a record on the database, for example by visiting http://localhost:3000/bookmarks/111, will cause an error in your app, very ugly to see.

Let's see how to solve these issues.

Model validations

If you tried to use a password shorter than 8 chars or a bad formatted email during user registrations, you'd note that Devise has spotted those errors by not saving the record and rendering the user registration form again. In fact, Devise provided some default validations for users, but our app needs validations on the `Bookmark` model too. Change `app/models/bookmark.rb` to

look like this:

```
class Bookmark < ActiveRecord::Base
  belongs_to :user

  # ensure that a user_id is present
  validates :user_id, presence: true

  # ensure that title is present and at
least 10 chars long
  validates :title, length: { minimum: 10 },
presence: true

  # ensure the url is present, and respects
the URL format for http/https
  validates :url, format: {with:
Regexp.new(URI::regexp(%w(http https)))},
presence: true
end
```

These validations are self-explanatory, so I'll not dive into them. Just keep in mind that there are many others or, in this case, you can create custom ones.

When a resource is not found…

The second problem we addressed is about passing a wrong/non-existent identifier for a given resource. You can handle it in many ways, perhaps the simplest (and laziest) one is to redirect to the index, with a *flash message*. To do this, we only need to change some line of code in `BookmarksController`:

```
class BookmarksController <
```

```
ApplicationController
  # [...] other code here

  private
    # Use callbacks to share common setup or
constraints between actions.
    def set_bookmark
      unless @bookmark =
current_user.bookmarks.where(id:
params[:id]).first
        flash[:alert] = 'Bookmark not found.'
        redirect_to root_url
      end
    end
end
```

The `set_bookmark` is a method called in a *before_action* only for actions that need a resource id (show, edit, update, destroy). The new lines will safely check for existence in the database, otherwise the user will be redirected to the root path of the site, with a short flash message.

Improving the app one step at a time

Our app is almost complete, or at least, it lacks several little details that make a difference. We'll try to address some of these. The goal, is to demonstrate how to improve an existing app with small steps.

Add a better root page

This is a simple task. All we need is a new route, controller and view. This time, we'll use a generator dedicated to the controller:

```
rails generate controller site index
```

I've omitted the output because at the moment you should be able to guess what it does. In short, it has created a `SiteController` with an action called `index`, plus the related view in `app/views/site/index.html.erb` and a new route in `config/routes.rb`. However, the route should be changed to reflect our goal:

```
Bookmarks::Application.routes.draw
  # [...] other routes

  # Comment/remove these lines
  # get "site/index"
  # root 'bookmarks#index'

  # Use this
  root 'site#index'
end
```

If you try to load http://localhost:3000 you'll see the empty template of `app/views/site/index.html.erb`. To render content, we need to change the controller in `app/controllers/site_controller.rb`:

```
class SiteController < ApplicationController
  def index
    # retrieve all Bookmarks ordered by
descending creation timestamp
    @bookmarks = Bookmark.order('created_at
```

```
desc')
  end
end
```

while the view in `app/views/site/index.html.erb`, might contain this basic markup as a starting point:

```
<h2>Latest Bookmarks</h2>
<table style="width: 100%">
  <thead>
    <tr>
      <th>Url</th>
    </tr>
  </thead>

  <tbody>
  <% @bookmarks.each do |bookmark| %>
    <tr>
      <td><%= link_to bookmark.title,
bookmark.url %></td>
    </tr>
  <% end %>
  </tbody>
</table>
```

Again, note how the controller processes the request by retrieving some records from the database through the model `Bookmark`, then renders its view that contains the proper data to show the bookmarks.

Prettier GUI

Being a programmer, I'm not too skilled in design and layouts, but

there are several CSS frameworks that help a lot. Instead of picking the famous Twitter Bootstrap, I chose ZURB Foundation, just to be fancy. There are a lot of ways to integrate existing CSS/Javascript frameworks and libraries inside a Rails app, the simplest one, is to use a gem when possible. Luckily, there's one for Zurb Foundation, just add it to your Gemfile:

```
gem 'zurb-foundation', '~> 4.2.2'
```

run `bundle install`, plus its install generator:

```
rails generate foundation:install
```

this command, will ask you to overwrite your default application layout in `app/views/layouts/application.html.erb`, press *Y* without worries. For brevity reasons, I've put the contents of the file here.

Moreover, we can also remove old scaffold's stylesheets:

```
rm app/assets/stylesheets/scaffolds.css.scss
```

Prettier forms

Forms are still ugly and boring to write, so we'll use another useful gem (from the same authors of *Devise*) to mitigate the problem. Add it to your Gemfile:

```
gem 'simple_form', '~> 3.0.0.rc'
```

run `bundle install` and the install generator (note that it

already supports Foundation's markup structure):

```
rails generate simple_form:install
--foundation
```

We'll also re-generate *Devise* views, because it supports *SimpleForm*:

```
rails generate devise:views
```

The generator will ask you if you want to overwrite existent files, press *Y* because that's what we really want.

The last step is to fix the *Bookmarks* form, it wasn't updated by the other generators, we'll do it manually by overwriting its (partial) template in `app/views/bookmarks/_form.html.erb` with the following content:

```
<%= simple_form_for(@bookmark) do |f| %>
  <%= f.error_notification %>

  <div class="form-inputs">
    <%= f.input :title %>
    <%= f.input :url %>
  </div>

  <div class="form-actions">
    <%= f.button :submit %>
  </div>
<% end %>
```

A welcome page

It would be nice and *professional* to show a welcome page for non-registered users. To keep things simple, I'll use the same `SiteController#index` used for the root page. Modify `app/views/site/index.html.erb` to look like this.

Conclusions

Congratulations! If you carefully followed all the steps, you should have a basic, yet working, Ruby On Rails app. Of course, this is only the *tip of the iceberg*: for brevity reasons, I've oversimplified some concepts, and I've skipped several important aspects (eg: using CSS/SASS and Javascript/Coffeescript, testing, deploy, etc...) that you should know if you want to be a proficient Ruby On Rails developer. However, as you can see, this is a huge article already, at least in terms of length 😛

I've uploaded the *complete app* on github: https://github.com/apeacox/12dos-bookmarks. It's a bit different from the one we wrote here, but it also adds some more features:

- pagination with the kaminari gem
- use *profile pages* to show single user's bookmarks

Clone or fork the code and experiment with it.

Webography

- http://guides.rubyonrails.org: official documentation and guides
- http://railscasts.com: videos (and transcripts)
- http://www.ruby-lang.org/en/documentation: Ruby's official

documentation

- https://www.ruby-toolbox.com: a place where to find gems for your tasks

Bibliography

- Programming Ruby 1.9 & 2.0 (4th edition): The Pragmatic Programmers' Guide
- Agile Web Development with Rails 4
- Crafting Rails 4 Applications: Expert Practices for Everyday Rails Development
- The Rails 4 Way

Andrea Pavoni

Andrea Pavoni is a passionate Italian programmer. He's mostly focused on web and mobile development, always looking for the best tools available. Andrea is also an active member of the Italian Ruby community. He helped in the organization for Ruby Days (2011, 2012) and coached at the first Italian Rails Girls event in Rome.

BACK TO THE ARTICLES

23 Comments

Leave a comment

Francis Needham
July 13, 2013

Great Article ! A must for starters and newbies !

Joe Molloy

Great article – one thing I noticed:
ny :bookmarks to app/models/user.rb
_to :user to app/models/user.rb
should read
add has_many :bookmarks to app/models/user.rb
_to :user to app/models/bookmark.rb

Agustín Gugliotta
August 13, 2013

icle!!!! I was searching a good guide for Rails4.

sailakshmi
December 26, 2013

Great Article.I tried this and got the results but when I restart the system I am getting an error called
NoMethodError in Site#index
Showing /Users/sailakshmi/Plackal/Workspaces /plackalwbapp/app/views/site/index.html.erb where line #15 raised:
undefined method `username' for nil:NilClass
please help me

sg
April 26, 2014

Thanks for writing this. Question about adding the line in the bookmarks_controller `before_action :authenticate_user!`. How is this visible? You say it is part of Devise, but how is Devise making this method visible?

Thomas
April 28, 2014

Awesome! Shows the power of rails. Incredible!!!
thx

Sandhya S P
May 10, 2014

Nice article. short n sweet,, great article for starters. thanks

---

Beginner's Tutorials for Developing with Ruby on Rails » CSS 3 & HTML 5 Links und Infos
June 6, 2014

[...] Ruby on Rails Web Application [...]

Anil kum
June 26, 2014

Nice article. short n sweet,, great article for starters. thanks

---

Neelam Maurya
July 2, 2014

Fantastic article.Helped a lot.Thanx

Install Ruby on Rails on Windows guide | egeek
August 28, 2014

[...] get started developing your first Ruby application, I highly recommend this guide by Andrea [...]

Easy Meals
November 18, 2014

You make it sound a lot easier than it actually is I am sure. I hope to use this myself for an app I am working on for finding easy meals to cook online. Thanks for the headsup I will use this myself. Cheers.

---

vivek
January 17, 2015

very useful for students

Paul
January 19, 2015

Under “Adding users and authentication”, …
“Devise also told us to check five steps to complete the setup:…”
Step 2 is unintelligible to me as a Rails newb:
“define a root_url in config/routes.rb. Fow now, we can achieve this by pointing BookmarksController#index to the root of the site:
Bookmarks::Application.routes.draw do # […] root ‘bookmarks#index’
[…] end”
If I put :
Bookmarks::Application.routes.draw do # […] root ‘bookmarks#index’
[…] end”
in config/routes.rb it errors out.
Looking at the checkin, what appears to go in here is simply:
root ‘site#index’
Why not just have that in this section instead of something which errors out and confuses newbs?
It’s gum on the shoe of an otherwise very good How To.
-Thanks Andrea

Paul
January 19, 2015

Due to the age of this post, I ended up using rails 4.2.0 and devise 3.4.0 instead of 4.0.0.rc and 3.0.0.rc as specified above. Might be good to update this since there is a bug in devise 3.0.0.rc with ‘merge’.

Florian
March 10, 2015

Great article, thanks!

---

devZone
April 7, 2015

Hi Andrea,
Thanks for putting this together. i come from (amongst) the MS ASP.NET MVC and was looking for a compact demo so i could explore more afterwards myself. mission accomplished!
cheers

IT Trends » Blog Archive » Scurtul ghid al unicornului din Ruby on Rails
July 20, 2015

[...] Getting started with Rails tutorial.(11) [...]

JCLL
July 23, 2015

Excellent tutorial ! Thanks !
(however I got a final error : ActionView::Template::Error (couldn't find file 'foundation' ...)
Not easy to solve for a beginnner...

---

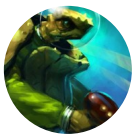
direct bad credit lenders
August 5, 2015

I'm amazed, I must say. Rarely do I come across a blog that's both equally educative and amusing, and without a doubt, you have hit the nail on the head. The issue is something that too few people are speaking intelligently about. Now i'm very happy that I came across this in my hunt for something regarding this.

Me
September 30, 2015

Great article, I was trying to learn Rails for sometime and mind you this is the only article with no errors when followed step by step.
I just got one error with User_id not being reflected…. I can live with that :P.
Anyways, Great article.

Michael
October 18, 2015



I spent a lot of time debugging a “Could not find devise-3.0.4 in any of the sources” error.
Turns out that if I write:
gem ‘devise’
in the Gemfile instead of:
gem ‘devise’, ‘~› 3.0.0.rc’
It worked fine 🙂

---

13 Best Ruby on Rails Tutorials for Beginners | ThePixelBeard
October 25, 2015

[…] 7. Writing a Web Application with Ruby on Rails […]

Your email address will not be published.
Comment

Name

Email

POST COMMENT

Notify me of follow-up comments by email.

Notify me of new posts by email.