[Chapter 2A demo app](http://ruby.railstutorial.org/chapters/a-demo-app" \l "top)

In this chapter, we’ll develop a simple demonstration application to show off some of the power of Rails. The purpose is to get a high-level overview of Ruby on Rails programming (and web development in general) by rapidly generating an application using *scaffold generators*. As discussed in [Box 1.2](http://ruby.railstutorial.org/chapters/beginning#sidebar-scaffolding), the rest of the book will take the opposite approach, developing a full application incrementally and explaining each new concept as it arises, but for a quick overview (and some instant gratification) there is no substitute for scaffolding. The resulting demo app will allow us to interact with it through its URLs, giving us insight into the structure of a Rails application, including a first example of the *REST architecture* favored by Rails.

As with the forthcoming sample application, the demo app will consist of *users* and their associated*microposts* (thus constituting a minimalist Twitter-style app). The functionality will be utterly under-developed, and many of the steps will seem like magic, but worry not: the full sample app will develop a similar application from the ground up starting in [Chapter 3](http://ruby.railstutorial.org/chapters/static-pages#top), and I will provide plentiful forward-references to later material. In the mean time, have patience and a little faith—the whole point of this tutorial is to take you *beyond* this superficial, scaffold-driven approach to achieve a deeper understanding of Rails.

[2.1 Planning the application](http://ruby.railstutorial.org/chapters/a-demo-app#sec-planning_the_application)

In this section, we’ll outline our plans for the demo application. As in [Section 1.2.3](http://ruby.railstutorial.org/chapters/beginning#sec-the_first_application), we’ll start by generating the application skeleton using the **rails** command:

**$** cd ~/rails\_projects

**$** rails new demo\_app

**$** cd demo\_app

Next, we’ll use a text editor to update the **Gemfile** needed by Bundler with the contents of[Listing 2.1](http://ruby.railstutorial.org/chapters/a-demo-app#code-demo_gemfile_sqlite_version_redux).

**Listing 2.1.** A **Gemfile** for the demo app.

source 'https://rubygems.org'

ruby '2.0.0'

*#ruby-gemset=railstutorial\_rails\_4\_0*

gem 'rails', '4.0.2'

group :development **do**

gem 'sqlite3', '1.3.8'

**end**

gem 'sass-rails', '4.0.1'

gem 'uglifier', '2.1.1'

gem 'coffee-rails', '4.0.1'

gem 'jquery-rails', '3.0.4'

gem 'turbolinks', '1.1.1'

gem 'jbuilder', '1.0.2'

group :doc **do**

gem 'sdoc', '0.3.20', require: false

**end**

group :production **do**

gem 'pg', '0.15.1'

gem 'rails\_12factor', '0.0.2'

**end**

Note that [Listing 2.1](http://ruby.railstutorial.org/chapters/a-demo-app#code-demo_gemfile_sqlite_version_redux) is identical to [Listing 1.9](http://ruby.railstutorial.org/chapters/beginning#code-gemfile_pg_gem).

As in [Section 1.4.1](http://ruby.railstutorial.org/chapters/beginning#sec-heroku_setup), we’ll install the local gems while suppressing the installation of production gems using the --without production option:

**$** bundle install --without production

**$** bundle update

**$** bundle install

(Recall that if Bundler complains about a readline error, try adding **gem ’rb-readline’** to your **Gemfile**.)

Finally, we’ll put the demo app under version control. Recall that the **rails** command generates a default **.gitignore** file, but depending on your system you may find the augmented file from[Listing 1.7](http://ruby.railstutorial.org/chapters/beginning#code-gitignore) to be more convenient. Then initialize a Git repository and make the first commit:

**$** git init

**$** git add .

**$** git commit -m "Initial commit"

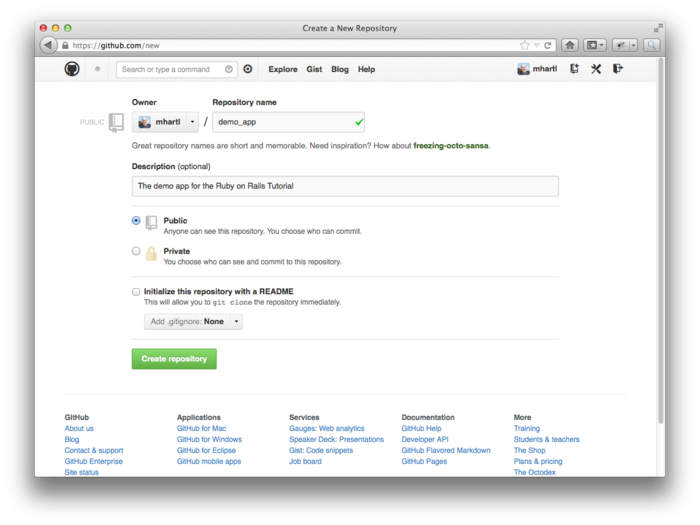


Figure 2.1: Creating a demo app repository at GitHub. [(full size)](http://railstutorial.org/images/figures/create_demo_repo_4_0-full.png)

You can also optionally create a new repository ([Figure 2.1](http://ruby.railstutorial.org/chapters/a-demo-app#fig-create_demo_repo)) and push it up to GitHub:

**$** git remote add origin https://github.com/<username>/demo\_app.git

**$** git push -u origin master

(As with the first app, take care *not* to initialize the GitHub repository with a **README** file.)

Now we’re ready to start making the app itself. The typical first step when making a web application is to create a *data model*, which is a representation of the structures needed by our application. In our case, the demo app will be a microblog, with only users and short (micro)posts. Thus, we’ll begin with a model for *users* of the app ([Section 2.1.1](http://ruby.railstutorial.org/chapters/a-demo-app#sec-modeling_demo_users)), and then we’ll add a model for*microposts* ([Section 2.1.2](http://ruby.railstutorial.org/chapters/a-demo-app#sec-modeling_demo_microposts)).

[2.1.1 Modeling demo users](http://ruby.railstutorial.org/chapters/a-demo-app#sec-modeling_demo_users)

There are as many choices for a user data model as there are different registration forms on the web; we’ll go with a distinctly minimalist approach. Users of our demo app will have a unique**integer** identifier called **id**, a publicly viewable **name** (of type **string**), and an **email** address (also a **string**) that will double as a username. A summary of the data model for users appears in[Figure 2.2](http://ruby.railstutorial.org/chapters/a-demo-app#fig-demo_user_model).

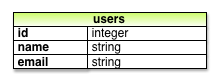


Figure 2.2: The data model for users.

As we’ll see starting in [Section 6.1.1](http://ruby.railstutorial.org/chapters/modeling-users#sec-database_migrations), the label **users** in [Figure 2.2](http://ruby.railstutorial.org/chapters/a-demo-app#fig-demo_user_model) corresponds to a *table* in a database, and the **id**, **name**, and **email** attributes are *columns* in that table.

[2.1.2 Modeling demo microposts](http://ruby.railstutorial.org/chapters/a-demo-app#sec-modeling_demo_microposts)

The core of the micropost data model is even simpler than the one for users: a micropost has only an **id** and a **content** field for the micropost’s text (of type **string**).[1](http://ruby.railstutorial.org/chapters/a-demo-app#fn-2_1) There’s an additional complication, though: we want to *associate* each micropost with a particular user; we’ll accomplish this by recording the **user\_id** of the owner of the post. The results are shown in [Figure 2.3](http://ruby.railstutorial.org/chapters/a-demo-app#fig-demo_micropost_model).

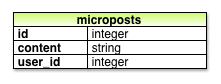


Figure 2.3: The data model for microposts.

We’ll see in [Section 2.3.3](http://ruby.railstutorial.org/chapters/a-demo-app#sec-demo_user_has_many_microposts) (and more fully in [Chapter 10](http://ruby.railstutorial.org/chapters/user-microposts#top)) how this **user\_id** attribute allows us to succinctly express the notion that a user potentially has many associated microposts.

[2.2 The Users resource](http://ruby.railstutorial.org/chapters/a-demo-app#sec-demo_users_resource)

In this section, we’ll implement the users data model in [Section 2.1.1](http://ruby.railstutorial.org/chapters/a-demo-app#sec-modeling_demo_users), along with a web interface to that model. The combination will constitute a *Users resource*, which will allow us to think of users as objects that can be created, read, updated, and deleted through the web via the [HTTP protocol](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol). As promised in the introduction, our Users resource will be created by a scaffold generator program, which comes standard with each Rails project. I urge you not to look too closely at the generated code; at this stage, it will only serve to confuse you.

Rails scaffolding is generated by passing the **scaffold** command to the **rails generate** script. The argument of the **scaffold** command is the singular version of the resource name (in this case, **User**), together with optional parameters for the data model’s attributes:[2](http://ruby.railstutorial.org/chapters/a-demo-app" \l "fn-2_2)

$ rails generate scaffold User name:string email:string

invoke active\_record

create db/migrate/20130305221714\_create\_users.rb

create app/models/user.rb

invoke test\_unit

create test/models/user\_test.rb

create test/fixtures/users.yml

invoke resource\_route

route resources :users

invoke jbuilder\_scaffold\_controller

create app/controllers/users\_controller.rb

invoke erb

create app/views/users

create app/views/users/index.html.erb

create app/views/users/edit.html.erb

create app/views/users/show.html.erb

create app/views/users/new.html.erb

create app/views/users/\_form.html.erb

invoke test\_unit

create test/controllers/users\_controller\_test.rb

invoke helper

create app/helpers/users\_helper.rb

invoke test\_unit

create test/helpers/users\_helper\_test.rb

invoke jbuilder

exist app/views/users

create app/views/users/index.json.jbuilder

create app/views/users/show.json.jbuilder

invoke assets

invoke coffee

create app/assets/javascripts/users.js.coffee

invoke scss

create app/assets/stylesheets/users.css.scss

invoke scss

create app/assets/stylesheets/scaffolds.css.scss

By including **name:string** and **email:string**, we have arranged for the User model to have the form shown in [Figure 2.2](http://ruby.railstutorial.org/chapters/a-demo-app#fig-demo_user_model). (Note that there is no need to include a parameter for **id**; it is created automatically by Rails for use as the *primary key* in the database.)

To proceed with the demo application, we first need to *migrate* the database using *Rake* ([Box 2.1](http://ruby.railstutorial.org/chapters/a-demo-app#sidebar-rake)):

**$** bundle exec rake db:migrate

== CreateUsers: migrating ====================================================

-- create\_table(:users)

-> 0.0017s

== CreateUsers: migrated (0.0018s) ===========================================

This simply updates the database with our new **users** data model. (We’ll learn more about database migrations starting in [Section 6.1.1](http://ruby.railstutorial.org/chapters/modeling-users#sec-database_migrations).) Note that, in order to ensure that the command uses the version of Rake corresponding to our **Gemfile**, we need to run **rake** using **bundle exec**. (If, as suggested in [Section 1.2.2.3](http://ruby.railstutorial.org/chapters/beginning#sec-install_ruby), you are using RVM, you can omit **bundle exec**, but I’ll include it for completeness. For alternate ways to eliminate the need for **bundle exec**, see [Section 3.6.1](http://ruby.railstutorial.org/chapters/static-pages#sec-eliminating_bundle_exec).)

With that, we can run the local web server using **rails s**, which is a shortcut for **rails server**:

**$** rails s

Now the demo application should be ready to go at <http://localhost:3000/>.

**Box 2.1.Rake**

In the Unix tradition, the [*make*](http://en.wikipedia.org/wiki/Make_(software)) utility has played an important role in building executable programs from source code; many a computer hacker has committed to muscle memory the line

$ ./configure && make && sudo make install

commonly used to compile code on Unix systems (including Linux and Mac OS X).

[2.4 Conclusion](http://ruby.railstutorial.org/chapters/a-demo-app#sec-2_4)

We’ve come now to the end of the 30,000-foot view of a Rails application. The demo app developed in this chapter has several strengths and a host of weaknesses.

**Strengths**

* High-level overview of Rails
* Introduction to MVC
* First taste of the REST architecture
* Beginning data modeling
* A live, database-backed web application in production

**Weaknesses**

* No custom layout or styling
* No static pages (like “Home” or “About”)
* No user passwords
* No user images
* No signing in
* No security
* No automatic user/micropost association
* No notion of “following” or “followed”
* No micropost feed
* No test-driven development
* **No real understanding**

The rest of this tutorial is dedicated to building on the strengths and eliminating the weaknesses.

[« Chapter 1 From zero to deploy](http://ruby.railstutorial.org/chapters/beginning#top)[Chapter 3 Mostly static pages »](http://ruby.railstutorial.org/chapters/static-pages#top)

1. When modeling longer posts, such as those for a normal (non-micro) blog, you should use the**text** type in place of **string**. [**↑**](http://ruby.railstutorial.org/chapters/a-demo-app#fnref-2_1)
2. The name of the scaffold follows the convention of *models*, which are singular, rather than resources and controllers, which are plural. Thus, we have **User** instead **Users**. [**↑**](http://ruby.railstutorial.org/chapters/a-demo-app#fnref-2_2)
3. Some references indicate that the view returns the HTML directly to the browser (via a web server such as Apache or Nginx). Regardless of the implementation details, I prefer to think of the controller as a central hub through which all the application’s information flows. [**↑**](http://ruby.railstutorial.org/chapters/a-demo-app#fnref-2_3)
4. Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. [**↑**](http://ruby.railstutorial.org/chapters/a-demo-app#fnref-2_4)
5. As with the User scaffold, the scaffold generator for microposts follows the singular convention of Rails models; thus, we have **generate Micropost**. [**↑**](http://ruby.railstutorial.org/chapters/a-demo-app#fnref-2_5)
6. The scaffold code may have extra newlines compared to [Listing 2.7](http://ruby.railstutorial.org/chapters/a-demo-app#code-demo_microposts_resource). This is not a cause for concern, as Ruby ignores extra newlines. [**↑**](http://ruby.railstutorial.org/chapters/a-demo-app#fnref-2_6)
7. Your console prompt might be something like **ruby-2.0.0-head >**, but the examples use >>since Ruby versions will vary. [**↑**](http://ruby.railstutorial.org/chapters/a-demo-app#fnref-2_7)