



*Daniel Kehoe*

# Rails and Bootstrap

*Ruby on Rails tutorial from the RailsApps Project*

# Contents

1.	Introduction .....	3
2.	Concepts.....	7
3.	Resources .....	9
4.	Accounts You May Need .....	13
5.	Getting Started .....	14
6.	Create the Application.....	16
7.	Gems .....	20
8.	Configuration .....	22
9.	Home Page .....	23
10.	Integrating Bootstrap .....	26
11.	Application Layout.....	32
12.	Flash Messages and Navigation .....	35
13.	Adding Pages .....	39
14.	Exploring Bootstrap.....	42
15.	Carousel .....	47
16.	Survey Form .....	50
17.	Spreadsheet Connection.....	59
18.	Modal Window .....	64
19.	Deploy .....	69
20.	Comments .....	77

## Chapter 1

# Introduction

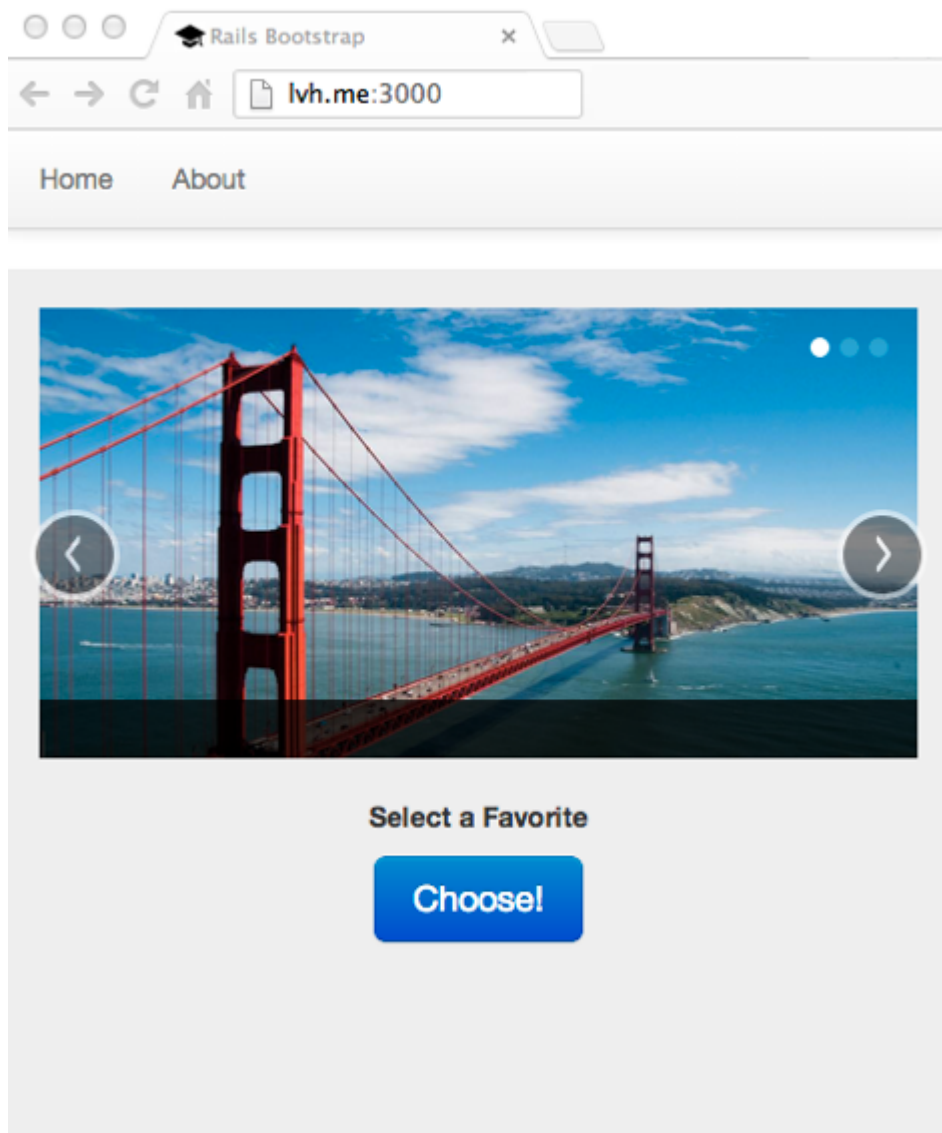
Welcome. This tutorial shows how to integrate Bootstrap with Rails. Versions:

- Bootstrap 3.1
- Rails 4.1

**Bootstrap** is a framework for front-end (browser-based or client-side) development, much like Ruby on Rails is a framework for server-side development. Bootstrap provides CSS stylesheets and JavaScript code.

With this tutorial, you'll build a working web application and gain hands-on experience with Bootstrap and Rails.

# Screenshot



## Is It for You?

If you're new to Rails, start with the [Learn Ruby on Rails](#) book from the RailsApps project. The [Learn Ruby on Rails](#) book will give you a solid introduction to Rails so you are prepared for this tutorial. You need at least beginner-level experience with Rails, which means you should have experience building a simple Rails web application.

If you are an experienced Rails developer, this tutorial will get you started quickly building a complete Rails starter application using Bootstrap.

# The Application

We'll build a basic web application that you can use as a starter application for your own projects. The starter application integrates Bootstrap with Rails. Features include:

- Home page
- "About" page
- Navigation bar

You'll find the [rails-bootstrap](#) example application on GitHub. It is a working application that is maintained by a team of experienced developers so you can always check the "reference implementation" if you have problems. You can use the [Rails Composer](#) tool to quickly generate the complete starter application.

The starter application uses the [high\\_voltage gem](#) for the "About" page. Additional pages can easily be added. The high\_voltage gem makes it easy to add pages with static content incorporating elements of a site-wide application layout such as header, navigation links, and footer.

After building the starter application, this tutorial goes further, introducing features of Bootstrap by implementing a simple survey website. Features include:

- Image carousel
- Survey form
- Google spreadsheet datastore
- Modal window

The tutorial uses Bootstrap to implement a carousel displaying multiple graphic images on the home page. The visitor is encouraged to vote for a favorite image. Voting is implemented with a survey form. When the visitor submits the form, the data is recorded in a Google Drive spreadsheet. Administrative access to the data is managed by Google Drive, eliminating the requirement for authentication and authorization in the application.

The application does not require a database. No pages are generated using information from a database and no user-submitted data is saved to a database. However, the `sqlite3` gem must be present in the Gemfile because Rails ActiveRecord is used for form validation.

By using Google Drive for data storage, you get a fully functional survey website without the complexity of user management, session handling, or database migrations (these topics are addressed in other tutorials).

The tutorial will show you how to deploy the application to Heroku, a popular hosting platform for Rails applications.

# Support the Project

The [RailsApps project](#) provides example applications that developers use as starter apps. Hundreds of developers use the apps, report problems as they arise, and propose solutions. Rails changes frequently; each application is known to work and serves as your personal “reference implementation” so you can stay up to date. Maintenance and development of the RailsApps applications is supported by subscriptions to the [RailsApps tutorials](#).

## Chapter 2

# Concepts

[Bootstrap](#) and other CSS front-end frameworks are toolkits that provide the kind of structure and convention that make Rails popular for back-end development. Bootstrap provides a standard grid for layout plus dozens of reusable components for common page elements such as navigation, forms, and buttons.

## Similar Frameworks

Bootstrap is the most popular framework for front-end development but it is not the only one.

Web developers began putting together “boilerplate” CSS stylesheets as early as 2000, when browsers first began to fully support CSS. Boilerplate CSS made it easy to reuse CSS stylesheet rules from project to project. More importantly, designers often implemented “CSS reset” stylesheets to enforce typographic uniformity across different browsers.

In particular, front-end developers at large companies were faced with a need to establish standards for use of CSS across projects. Engineers at Yahoo! released the [Yahoo! User Interface Library](#) (YUI) as an open source project in [February 2006](#). Inspired by an [article by Jeff Croft](#), and reacting to the huge size of the YUI library, independent developers began releasing other CSS frameworks such as the [960 grid system](#) and the [Blueprint](#) CSS framework.

There are [dozens of CSS frameworks](#). In general, they all seek to implement a common set of requirements:

- An easily customizable grid
- Some default typography
- A typographic baseline
- CSS reset for default browser styles
- A stylesheet for printing

More recently, with the ubiquity of smartphones and tablets, CSS frameworks support [responsive web design](#), accommodating differences in screen sizes across a range of devices.

Bootstrap came from an effort to document and share common design patterns and assets across projects at Twitter. It was released as an open source project in August 2011. Perhaps

because of Twitter's well-known brand identity, as well as an active developer community, Bootstrap is more popular than the other CSS frameworks.

[Zurb Foundation](#) is an excellent alternative to Bootstrap. For a comparison of the two frameworks, see [Framework Fight](#). You'll find a [rails-foundation](#) example application in the RailsApps collection.



## Chapter 3

# Resources

Before you dive into integrating Bootstrap with Rails, let's consider how you can get help when you need it.

We'll also take a look at the ways you can enrich your application with add-ons for Bootstrap.

## Getting Help With the Application

The [rails-bootstrap](#) example application on GitHub serves as a “reference implementation” if you have problems. The example application is updated more frequently than this tutorial. And errors are reported as GitHub issues.

Take a moment now to look at the [open issues](#) on GitHub to see what problems you may encounter as you work your way through the tutorial. You can look at the [closed issues](#) to see some of the solved problems.

The code in this tutorial was tested by many people and worked flawlessly at the time this was written. But the code is dependent on specific gem versions. If problems have developed since this tutorial was last updated, you'll find the details reported in the GitHub issues. If you discover a bug, you can clone the repo and run the code locally to confirm there is an error in the reference implementation. If you find an anomaly, open an issue so we can help.

[Stack Overflow](#) provides a question-and-answer forum for readers of this tutorial. As the author of this tutorial, I can't answer specific questions or help you directly when you contact me by email. If I did, I would not have time to create the tutorials that benefit so many people. However, I watch for questions on Stack Overflow that are tagged `railsapps` and provide answers when I can. Ask your question on Stack Overflow and use the tag `railsapps`.

## Getting Help With Rails or Bootstrap

What to do when you get stuck?

“Google it,” of course. Often you need to eliminate outdated advice that is relevant only for older versions of Rails. Google has options under “Search tools” to show only recent results from the past year.

[Stack Overflow](#) is a popular site for questions and answers about Bootstrap. Answers from Stack Overflow are helpful if you check carefully to make sure the answers are recent. It is wise to compare several answers to find what's relevant to your own circumstances.

Free sources of help:

- [Rails Hotline](#) — free telephone hotline for Rails questions staffed by volunteers
- [RailsMentors](#) — network of volunteer mentors

Commercial services offering help:

- [AirPair](#) — marketplace of mentors available via video chat and screen sharing
- [HackHands](#) — another marketplace of mentors

Directory of Rails consulting companies:

- [Rails Development Directory](#)

## Staying Up-to-Date

Bootstrap is regularly updated, often with significant changes from version to version. For news, you can follow the official account for Bootstrap, [@twbootstrap](#), on Twitter.

Peter Cooper's [HTML5 Weekly](#) and [JavaScript Weekly](#) include occasional items about Bootstrap. I recommend his email newsletters, including [Ruby Weekly](#).

You can follow [@rails\\_apps](#) on Twitter for news about the RailsApps project.

## Bootstrap Resources

If you add nothing to Bootstrap, you'll have a solid front-end framework for creating a graphic design, plus Bootstrap's dozen built-in JavaScript-based interactive elements. But because Bootstrap provides a structure for web design, and has a large and active developer community, there are hundreds of additional components, tools, and themes that can enhance your application.

You can find resources for Bootstrap on the [Big Badass List of Useful Twitter Bootstrap Resources](#) at the Bootstrap Hero site. The Bootstrap Hero list, in its magnitude, demonstrates the popularity of Bootstrap and the vitality of the open source community.

# Bootstrap Components

The Bootstrap documentation gives you a small selection of example code, just enough to illustrate how to use each Bootstrap component. But you can assemble Bootstrap's primitive components into complex design elements such as calendars, pricing tables, login forms, and many others. Instead of building what you need by trial and error, visit the [Bootsnipp.com](https://bootsnipp.com) gallery for dozens of design elements and code examples.

For more components you can add to Bootstrap, see the extensive list of additional [Bootstrap-compatible addons](#) listed on the Bootstrap Hero site.

## Bootstrap Themes

Frameworks such as Bootstrap are intended to provide the building blocks for a custom website design. If you've got strong design skills, or can partner with an experienced web designer, you can develop a custom design that expresses the purpose and motif of your website. If you don't have the skill or resources to customize the design, you can use the generic Bootstrap design.

An alternative is to find or purchase a pre-designed theme for your website. [Themes for Bootstrap](#) aggregates themes from many sites.

You can adapt free themes from sites such as:

- [Themestrap](#)
- [Start Bootstrap](#)
- [Bootswatch](#)

You can adapt themes that you purchase on marketplaces such as:

- [WrapBootstrap](#)

Here's more on [How to integrate a WrapBootstrap theme into a Rails app](#).

A few consultants offer Bootstrap themes that are specifically designed for Rails projects:

- [Dressed Rails Themes](#) from Marc-André Cournoyer
- [Railsview.com](#) from Richardson Dackam

You can design your pages using Bootstrap with drag-and-drop design tools such as:

- [Easel](#)
- [Divshot](#)

- [Jetstrap](#)
- [LayoutIt](#)

If you want the retro look of the 1990s web, use:

- [Geo for Bootstrap](#)

You've got to see it. Believe it or not, websites used to look like that.

## Chapter 4

# Accounts You May Need

The tutorial will show how to use [Git](#) for version control. If you want to store your application on GitHub, you can get a [free personal GitHub account](#). See the article [Rails and Git](#) for more information.

## Google Drive

The application will use a Google Drive spreadsheet for data storage. If you have a [Gmail](#) account, your username and password are the credentials you use to access Google Drive. If you don't have a Gmail account, you can [sign up for a Google account](#) for free.

Some Google accounts require 2-step verification, which sends a unique code to your mobile phone each time you log in from an unfamiliar device. If your Google account requires two-factor authentication, you have three choices:

- [set up an application-specific password](#)
- turn off 2-step verification
- create a new Gmail account for use with this tutorial

## Heroku

The tutorial will show how to deploy the application to [Heroku](#) which provides Rails application hosting. It costs nothing to set up a Heroku account and deploy as many applications as you want.

To deploy an app to Heroku, you must have a Heroku account. Visit [Heroku sign-up page](#) to set up an account.

Be sure to use the same email address you used to register for GitHub. It's very important that you use the same email address for GitHub and Heroku accounts.

## Chapter 5

# Getting Started

## Before You Start

If you follow this tutorial closely, you'll have a working application that closely matches the example app in the [rails-bootstrap](#) GitHub repository. If your application doesn't work after following the tutorial, compare the code to the example app in the GitHub repository, which is known to work.

If you find problems or wish to suggest improvements, please create a [GitHub issue](#). It's best to download and check the example application from the GitHub repository before you report an issue, just to make sure the error isn't a result of your own mistake.

## Your Development Environment

The [Learn Ruby on Rails](#) tutorial explains how to set up a text editor and terminal application.

This tutorial will use [Git](#) for version control. See the article [Rails and Git](#) for more information.

## Ruby 2.1

See article [Installing Rails](#) to install Ruby 2.1, the newest version of Ruby. The article will guide you to update various supporting gems and utilities as needed.

Check that the appropriate version of Ruby is installed in your development environment. Open your terminal application and enter:

```
$ ruby -v
```

You should see Ruby version 2.1.1 or newer.

If you are running older versions of Ruby on your computer, you must install a newer version to avoid unexpected problems.

# Project-Specific Gemset

The instructions in the article [Installing Rails](#) recommend using [RVM](#), the Ruby version manager. If you are an experienced Unix administrator, you can consider alternatives such as [Chruby](#), Sam Stephenson's [rbenv](#), or others [on this list](#). RVM is popular, well-supported, and an excellent utility to help a developer install Ruby and manage gemsets; that's why I recommend it.

For our rails-bootstrap application, we'll create a project-specific gemset using RVM. We'll give the gemset the same name as our application.

After we create a the project-specific gemset, we'll install the Rails gem into the gemset. Enter these commands:

```
$ rvm use ruby-2.1.1@rails-bootstrap --create  
$ gem install rails
```

Make sure Rails is ready to run. Open a terminal and type:

```
$ rails -v
```

You should have Rails version 4.1.1 or newer.

## Chapter 6

# Create the Application

We'll use the [Rails Composer](#) tool to generate a starter app.

We already created a project-specific gemset using RVM. Make sure it's ready to use:

```
$ rvm use ruby-2.1.1@rails-bootstrap
```

To create the Rails default starter application, type:

```
$ rails new rails-bootstrap -m https://raw.github.com/RailsApps/rails-composer/master/composer.rb
```

This will create a new Rails application named "rails-bootstrap."

You may give the app a different name if you are building it for another purpose. For this tutorial, we'll assume the name is "rails-bootstrap." You'll avoid extra steps and errors by using this name.

You'll see a prompt:

```
question  Build a starter application?
          1) Build a RailsApps example application
          2) Build a contributed application
          3) I want to build my own application
```

Enter "1" to select **Build a RailsApps example application**. You'll see a prompt:

```
question  Starter apps for Rails 4.1. More to come.
          1) learn-rails
          2) rails-bootstrap
          3) rails-foundation
          4) rails-omniauth
          5) rails-devise
          6) rails-devise-pundit
```

Choose **rails-bootstrap**. The Rails Composer tool may give you other options (other applications may have been added since these notes were written).

The application generator template will ask you for additional preferences:



```

question Web server for development?
  1) WEBrick (default)
  2) Thin
  3) Unicorn
  4) Puma
  5) Phusion Passenger (Apache/Nginx)
  6) Phusion Passenger (Standalone)
question Web server for production?
  1) Same as development
  2) Thin
  3) Unicorn
  4) Puma
  5) Phusion Passenger (Apache/Nginx)
  6) Phusion Passenger (Standalone)
question Template engine?
  1) ERB
  2) Haml
  3) Slim
question Test framework?
  1) None
  2) RSpec with Capybara
extras Set a robots.txt file to ban spiders? (y/n)
extras Create a GitHub repository? (y/n)
extras Use or create a project-specific rvm gemset? (y/n)

```

## Web Servers

We recommend Thin in development for speed and less noise in the log files.

If you plan to deploy to Heroku, select Thin as your production webserver. Unicorn is recommended by Heroku but configuration is more complex.

## Template Engine

The example application uses the default “ERB” Rails template engine. Optionally, you can use another template engine, such as Haml or Slim. See instructions for [Haml and Rails](#).

## Testing

If you are a beginner, select “None.”

## Other Choices

Set a robots.txt file to ban spiders if you want to keep your new site out of Google search results.

If you choose to create a GitHub repository, the generator will prompt you for a GitHub username and password.

It is a good idea to use [RVM](#), the Ruby Version Manager, and create a project-specific rvm gemset (not available on Windows). See [Installing Rails](#). Making this choice creates two files, **.ruby-version** and **.ruby-gemset**, that set RVM every time we `cd` to the project directory.

## Troubleshooting

If you get an error “OpenSSL certificate verify failed” or “Gem::RemoteFetcher::FetchError: SSL\_connect” see the article [OpenSSL Errors and Rails](#).

## Your Project Directory

After you create the application, switch to its folder to continue work directly in the application:

```
$ cd rails-bootstrap
```

This is your project directory. It is also called the application root directory.

## Replace the READMEs

Please edit the README files to add a description of the app and your contact info. Changing the README is important if your app will be publicly visible on GitHub. Otherwise, people will think I am the author of your app. If you like, add an acknowledgment and a link to the [RailsApps project](#).

## Set Up Source Control (Git)

If you used the [Rails Composer](#) tool to generate the starter app, the application template script has already set up a source control repository. You can confirm with:

```
$ git status
```

If you are new to Git, see detailed instructions for [Rails and Git](#).

If you want to store your application on GitHub, you should now set up your [GitHub repository](#), if you didn't make the choice using the [Rails Composer](#) tool.

## Test the Application

You've created a starter application. It's ready to run.

### Launching the Web Server

You can launch the application by entering the command:

```
$ rails server
```

The `rails server` command launches the default [WEBrick web server](#) that is provided with Ruby.

### Viewing in the Web Browser

To see your application in action, open a web browser window and navigate to <http://localhost:3000/>. You'll see the starter app home page.

### Stopping the Web Server

You can stop the server with Control-c to return to the command prompt.

Most of the time you'll keep the web server running as you add or edit files in your project. Changes will automatically appear when you refresh the browser or request a new page. There is a tricky exception, however. If you make changes to the Gemfile, or changes to configuration files, the web server must be shut down and relaunched for changes to be activated.

## Chapter 7

# Gems

Bootstrap can be installed using either its native [LESS CSS](#) language or the [Sass](#) language. Sass is a default for Rails, and the gem [bootstrap-sass](#) is the [official Sass port of Bootstrap](#). Using a gem means you'll install Bootstrap into the [Rails asset pipeline](#).

If you want to explore the differences between LESS and Sass, see the article [Sass vs. LESS](#). In general, it recommends Sass.

The starter app already contains a Gemfile with the gems you will need.

Here are gems we'll use:

- [bootstrap-sass](#) – Bootstrap for CSS and JavaScript
- [high\\_voltage](#) – for static pages like “about”

We'll also add utilities that make development easier:

- [better\\_errors](#) – helps when things go wrong
- [quiet\\_assets](#) – suppresses distracting messages in the log
- [rails\\_layout](#) – generates files for an application layout

Later, when we begin to explore Bootstrap and customize our starter app, we'll add more gems. For now, these are all basic gems that are useful for any starter application.

Examine the **Gemfile**:

```

source 'https://rubygems.org'
ruby '2.1.1'
gem 'rails', '4.1.1'
gem 'sqlite3'
gem 'sass-rails', '~> 4.0.1'
gem 'uglifier', '>= 1.3.0'
gem 'coffee-rails', '~> 4.0.0'
gem 'jquery-rails'
gem 'turbolinks'
gem 'jbuilder', '~> 2.0'
gem 'sdoc', '~> 0.4.0',      group: :doc
gem 'spring',              group: :development
gem 'bootstrap-sass'
gem 'high_voltage'
group :development do
  gem 'better_errors'
  gem 'binding_of_caller', :platforms=>[:mri_19, :mri_20, :rbx]
  gem 'quiet_assets'
  gem 'rails_layout'
end

```

Check for the [current version of Rails](#) and replace `gem 'rails', '4.1.1'` accordingly. The Rails default gems may have changed as well.

Notice that we've placed several gems inside a "group." Specifying a group for development or testing insures a gem is not loaded in production, reducing the application's memory footprint.

## Install the Gems

Each time you edit the Gemfile, run `bundle install` and restart your web server.

```
$ bundle install
```

You can check which gems are loaded into the development environment:

```
$ gem list
```

Keep in mind that you have installed these gems locally. When you deploy the application to another server, the same gems (and versions) must be available.

We're ready to configure the application.

## Chapter 8

# Configuration

Many applications need a way to set configuration values such as account credentials or API keys. It is a basic feature for our starter application.

To consolidate configuration settings in a single location, store credentials in the **config/secrets.yml** file. To keep your credentials private, use Unix environment variables to set your credentials. See the article [Rails Environment Variables](#) for more information.

This is the default **config/secrets.yml** file:

```
development:
  secret_key_base: (not shown)
```

Later in this tutorial, when we go beyond the basic starter application and begin to explore Bootstrap in more depth, we'll set credentials for your Google account in the **config/secrets.yml** file. All configuration values in the **config/secrets.yml** file are available anywhere in the application as variables, for example as

```
Rails.application.secrets.secret_key_base .
```

If you don't want to use Unix environment variables, you can set each value directly in the **config/secrets.yml** file. The file must be in your git repository when you deploy to Heroku. However, you shouldn't save the file to a public GitHub repository where other people can see your credentials.

We're ready to examine the home page from the starter application.

## Chapter 9

# Home Page

The starter app contains a placeholder home page. It provides a controller, view file, and routing. Later we'll add a graphic carousel and survey form so we can explore features of Bootstrap.

## User Story

Here's a user story for the home page:

```
*Home page*  
As a visitor to the website  
I want to see the name of the website  
And a link to an "About" page  
so I can identify the website and learn about its purpose
```

The user story is simple and obvious. We'll modify it as we add features to the website.

The starter app created a Visitors controller and a home page in the file **app/views/visitors/index.html.erb**. Later we'll replace the contents of the home page with a survey form. To prepare for that step, we'll move the home page from **app/views/visitors/index.html.erb** to **app/views/visitors/new.html.erb**.

## Controller

The starter app created a controller that will render our home page. We could call it the Home controller or the Welcome controller, but based on our user story, a Visitors controller is appropriate. Later, when we add a form to the home page, we'll create a Visitor model to match the Visitors controller.

The controller class name is `VisitorsController` and **visitors\_controller.rb** is the filename.

Examine the file **app/controllers/visitors\_controller.rb**:

```
class VisitorsController < ApplicationController  
end
```

Replace the file with this:

```
class VisitorsController < ApplicationController

  def new
  end

end
```

We define the class and name it `class VisitorsController`, inheriting behavior from the `ApplicationController` class which is defined in the Rails API.

We define the `new` method so we have a stub for features we'll add later. Hidden behavior inherited from the `ApplicationController` does all the work of rendering the view. Invoking the `new` method calls a `render` method supplied by the `ApplicationController` parent class. The `render` method searches in the **app/views/visitors** directory for a view file named **new** (the file extension **.html.erb** is assumed by default).

## View

The starter app created an **app/views/** directory for our view file.

Examine the file **app/views/visitors/index.html.erb**:

```
<h3>Welcome</h3>
```

Rename the file from **app/views/visitors/index.html.erb** to **app/views/visitors/new.html.erb**.

This is a placeholder page we'll modify later.

## Routing

The starter app created a route to the home page.

Examine the file **config/routes.rb**:

```
Rails::Application.routes.draw do
  root :to => 'visitors#index'
end
```

Change the route from `visitors#index` to `visitors#new`:



```
RailsBootstrap::Application.routes.draw do
  root :to => 'visitors#new'
end
```

Any request to the application root (<http://localhost:3000/>) will be directed to the VisitorsController `new` action.

These changes set up our home page to be served by the VisitorsController `new` action. Later we'll replace our placeholder home page with a survey form.

For more information about routing, see the reference documentation, [RailsGuides: Routing from the Outside In](#).

Now we'll look at integrating Bootstrap with our application.

## Chapter 10

# Integrating Bootstrap

Here are the steps to add Bootstrap to a Rails application:

- add a gem to the **Gemfile**
- modify the file **app/assets/javascripts/application.js** to add Bootstrap's Javascript files
- add the file **app/assets/stylesheets/framework\_and\_overrides.css.scss** to add Bootstrap's CSS files

We don't have to do anything because the Rails Composer tool has done it all for us.

## Rails Layout Gem

The Rails Composer tool uses the [rails\\_layout](#) gem to set up Bootstrap and add the necessary files.

The rails\_layout gem renamed the file:

- **app/assets/stylesheets/application.css**

to:

- **app/assets/stylesheets/application.css.scss**

It created the file:

- **app/assets/stylesheets/framework\_and\_overrides.css.scss**

and modified the file:

- **app/assets/javascripts/application.js**

The rails\_layout gem created or replaced four files:

- **app/views/layouts/application.html.erb**
- **app/views/layouts/\_messages.html.erb**
- **app/views/layouts/\_navigation.html.erb**
- **app/views/layouts/\_navigation\_links.html.erb**

Let's examine the files to see how our application is configured to use Bootstrap.

## Renaming the application.css File

The rails\_layout gem renamed the **app/assets/stylesheets/application.css** file as **app/assets/stylesheets/application.css.scss**. Note the **.scss** file extension. This will allow you to use the advantages of the Sass syntax for your application stylesheet.

Ordinary CSS is not a programming language so CSS rules are verbose and often repetitive. [Sass](#) is a preprocessor for CSS so your stylesheets can use variables, mixins, and nesting of CSS rules. You can create a variable such as `$blue: #3bbfce` and specify colors anywhere using the variable, for example, `border-color: $blue`. Mixins are like variables that let you use snippets of reusable CSS. Nesting eliminates repetition by layering CSS selectors.

For more on the advantages of Sass and how to use it, see the [Sass](#) website or the [Sass Basics RailsCast](#) from Ryan Bates.

Sass has two syntaxes. The most commonly used syntax is known as "SCSS" (for "Sassy CSS"), and is a superset of the CSS syntax. This means that every valid CSS stylesheet is valid SCSS as well. You can use Sass in any file by adding the file extension **.scss**. The asset pipeline will preprocess any **.scss** file and expand it as standard CSS.

Before you continue, make sure that the rails\_layout gem renamed the **app/assets/stylesheets/application.css** file as **app/assets/stylesheets/application.css.scss**. Otherwise you won't see the CSS styling we will apply.

## The application.css.scss File

The Rails asset pipeline will concatenate and compact CSS stylesheets for delivery to the browser when you add them to this directory:

- **app/assets/stylesheets/**

The asset pipeline helps web pages display faster in the browser by combining all CSS files into a single file (it does the same for JavaScript).

Let's examine the file **app/assets/stylesheets/application.css.scss**:

```

/*
 * This is a manifest file that'll be compiled into application.css, which will include
all the files
 * listed below.
 *
 * Any CSS and SCSS file within this directory, lib/assets/stylesheets, vendor/assets/
stylesheets,
 * or vendor/assets/stylesheets of plugins, if any, can be referenced here using a
relative path.
 *
 * You're free to add application-wide styles to this file and they'll appear at the
bottom of the
 * compiled file so the styles you add here take precedence over styles defined in any
styles
 * defined in the other CSS/SCSS files in this directory. It is generally better to
create a new
 * file per style scope.
 *
 *= require_tree .
 *= require_self
 */

```

The **app/assets/stylesheets/application.css.scss** file serves two purposes.

First, you can add any CSS rules to the file that you want to use anywhere on your website. Second, the file serves as a *manifest*, providing a list of files that should be concatenated and included in the single CSS file that is delivered to the browser.

## A Global CSS File

Any CSS style rules that you add to the **app/assets/stylesheets/application.css.scss** file will be available to any view in the application. You could use this file for any style rules that are used on every page, particularly simple utility rules such as highlighting or resetting the appearance of links. However, in practice, you are more likely to modify the style rules provided by Bootstrap. These modifications don't belong in the **app/assets/stylesheets/application.css.scss** file; they will go in the **app/assets/stylesheets/framework\_and\_overrides.css.scss** file.

In general, it's bad practice to place a lot of CSS in the **app/assets/stylesheets/application.css.scss** file (unless your CSS is very limited). Instead, structure your CSS in multiple files. CSS that is used on only a single page can go in a file with a name that matches the page. Or, if sections of the website share common elements, such as themes for landing pages or administrative pages, make a file for each theme. How you organize your CSS is up to you; the asset pipeline lets you organize your CSS so it is easier to develop and maintain. Just add the files to the **app/assets/stylesheets/** folder.

## A Manifest File

It's not obvious from the name of the **app/assets/stylesheets/application.css.scss** file that it serves as a *manifest file* as well as a location for miscellaneous CSS rules. For most websites, you can ignore its role as a manifest file. In the comments at the top of the file, the `*= require_self` directive indicates that any CSS in the file should be delivered to the browser. The `*= require_tree .` directive (note the Unix “dot operator”) indicates any files in the same folder, including files in subfolders, should be combined into a single file for delivery to the browser.

If your website is large and complex, you can remove the `*= require_tree .` directive and specify individual files to be included in the file that is generated by the asset pipeline. This gives you the option of reducing the size of the application-wide CSS file that is delivered to the browser. For example, you might segregate a file that includes CSS that is used only in the site's administrative section. In general, only large and complex sites need this optimization. The speed of rendering a single large CSS file is faster than fetching multiple files.

## Bootstrap JavaScript

Bootstrap provides both CSS and JavaScript libraries.

Like the **application.css.scss** file, the **application.js** file is a manifest that allows a developer to designate the JavaScript files that will be combined for delivery to the browser.

The rails\_layout gem modified the file **app/assets/javascripts/application.js** to include the Bootstrap JavaScript libraries:

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

It added the directive `//= require bootstrap` before `//= require_tree .`

## Bootstrap CSS

The rails\_layout gem added a file **app/assets/stylesheets/framework\_and\_overrides.css.scss** containing:

```
// import the CSS framework
@import "bootstrap";
.
.
.
```

The file **app/assets/stylesheets/framework\_and\_overrides.css.scss** is automatically included and compiled into your Rails application.css file by the `*= require_tree .` statement in the **app/assets/stylesheets/application.css.scss** file.

The `@import "bootstrap";` directive will import the Bootstrap CSS rules from the Bootstrap gem.

You could add the Bootstrap `@import` code to the **app/assets/stylesheets/application.css.scss** file. However, it is better to have a separate **app/assets/stylesheets/framework\_and\_overrides.css.scss** file. You may wish to modify the Bootstrap CSS rules; placing changes to Bootstrap CSS rules in the **framework\_and\_overrides.css.scss** file will keep your CSS better organized.

## Overriding Bootstrap Classes

The file **app/assets/stylesheets/framework\_and\_overrides.css.scss** shows how to customize Bootstrap classes.

```
// make all images responsive by default
img {
  @extend .img-responsive;
  margin: 0 auto;
}
// override for the 'Home' navigation link
.navbar-brand {
  font-size: inherit;
}
```

The first style rule make all images responsive by default. With this rule, all images will resize to accommodate browser windows of varying widths. We use the Sass `@extend` directive to add the Bootstrap class `img-responsive` to the HTML `img` element.

The second style rule will force the font size of a `navbar-brand` navigation link to be the same size as the fonts specified in an enclosing container.

## Using Sass Mixins with Bootstrap

In addition to the simple `@import "bootstrap";` directive, the **app/assets/stylesheets/framework\_and\_overrides.css.scss** contains a collection of Sass mixins. These are examples that you can remove.

You can use Sass mixins to map Bootstrap class names to your own semantic class names. The rails\_layout gem provides examples of Sass mixins that apply CSS style rules to the default application layout. In doing so, the default application layout is free of framework-specific code and can be used with Bootstrap, Zurb Foundation, or other front-end frameworks. The book [Learn Ruby on Rails](#) explains how to use Sass mixins.

## Chapter 11

# Application Layout

In this section we'll look closely at the application layout, so we can organize the design of our web pages. The default application layout is where you put HTML that you want to include on every page of your website.

## Application Layout

We already have a view file for our home page. Rails will combine the `Visitors#New` view with the default application layout file.

Let's look at the application layout file provided by the starter app.

Examine the contents of the file **`app/views/layouts/application.html.erb`**:

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><%= content_for?(:title) ? yield(:title) : "Rails Bootstrap" %></title>
    <meta name="description" content="<%= content_for?(:description) ?
yield(:description) : "Rails Bootstrap" %>">
    <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track' => true
%>
    <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
    <%= csrf_meta_tags %>
  </head>
  <body>
    <header>
      <%= render 'layouts/navigation' %>
    </header>
    <main role="main">
      <%= render 'layouts/messages' %>
      <%= yield %>
    </main>
  </body>
</html>
```

The default application layout contains the Ruby keyword `yield`. The `yield` keyword is replaced with a view file that is specific to the controller and action, in this case, the **`app/views/visitors/new.html.erb`** view file. The content from the view is inserted where you place the `yield` keyword.



The default application layout uses partials for navigation and flash messages. In the next chapter, we'll look at the contents of the partials.

The book [Learn Ruby on Rails](#) covers the application layout file in detail, explaining the purpose of each element.

## Bootstrap Grid

You can add Bootstrap classes directly to the application layout.

You can organize your layout in horizontal sections using `row` classes. Rows should be nested within a `container` class.

A row can contain a single column or you can split it into multiple columns. A row contains a maximum of 12 columns.

The width of columns automatically adjusts to the width of the browser window. Here's a table that shows the width of the page on different devices:

	Phones	Tablets	Desktops	Desktops
<b>Container width</b>	none (auto)	750px	970px	1170px
<b>Class prefix</b>	.col-xs-	.col-sm-	.col-md-	.col-lg-
<b>Column width</b>	Auto	60px	78px	95px

Here's how to add Bootstrap classes so all your pages display within a single full-width column:

```
<main role="main" class="container">
  <div class="row">
    <div class="col-lg-12">
      <%= render 'layouts/messages' %>
      <%= yield %>
    </div>
  </main>
```

Here's a footer presented as a row with two sections:

```
<footer class="container">
  <div class="row">
    <section class="col-sm-4">
      Copyright 2014
    </section>
    <section class="col-sm-8">
      All rights reserved.
    </section>
  </div>
</footer>
```

The Bootstrap `row` class will create a horizontal break. The footer will contain two side-by-side sections. The first will be four columns wide; the second will be eight columns wide. By specifying `col-sm-`, the footer will contain side-by-side columns for all desktops and tablets. On phones, the columns will collapse and display as stacked rows.

To see the grid in action, change `col-sm-` to `col-lg-` and watch what happens when you resize the browser on a desktop computer. For any width narrower than 1170px, the columns will collapse and display as stacked rows.

See the [documentation for the Bootstrap Grid](#) to learn about the Bootstrap grid classes.

## Chapter 12

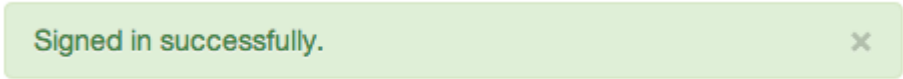
# Flash Messages and Navigation

In this chapter, we'll look at flash messages and navigation used by our application layout.

## Flash Messages

Rails provides a standard convention to display alerts (including error messages) and other notices (including success messages), called a `flash message`. The name comes from the term “flash memory” and should not be confused with the “Adobe Flash” web development platform that was once popular for animated websites. The flash message is documented in the [RailsGuides: Action Controller Overview](#).

Here's a flash message you might see after logging in to an application:



Signed in successfully.

It is called a “flash message” because it appears on a page temporarily. When the page is reloaded or another page is visited, the message disappears.

Flash messages are created in a controller. For example, we could add messages to the home page by modifying the `VisitorsController` like this:

```
class VisitorsController < ApplicationController

  def new
    flash[:notice] = 'Welcome!'
    flash[:alert] = 'Access not allowed.'
  end

end
```

Rails provides the `flash` object so that messages can be created in the controller and displayed on the rendered web page.

Use `flash.now` in the controller when you immediately render a page, for example with a `render :new` directive. Using `flash.now` will make sure the message only appears on the rendered page and will not persist after a user follows a link to a new page. Use the simple

variant, `flash`, in the controller when you redirect to another page, for example with a `redirect_to root_path` directive.

## Flash Messages with Bootstrap

Rails uses `:notice` and `:alert` as flash message keys. Bootstrap provides a base class `alert` with additional classes `alert-success` and `alert-danger`. A bit of parsing is required to get a Rails “notice” message to be styled with the Bootstrap `alert-success` style. Any other message, including a Rails “alert” message, will be styled with the Bootstrap `alert-danger` style.

By default, Bootstrap applies a green background to `alert-success` and a red background to `alert-danger`. Bootstrap provides additional classes `alert-info` (blue) and `alert-warning` (yellow). With a little hacking, it’s possible to create a Rails flash message with a custom name, such as `:info`, that will display with the Bootstrap `info` class. However, it’s wise to stick with the Rails convention of using only “alert” and “notice.”

You can include code to display flash messages directly in your application layout file or you can create a [partial template](#) – a “partial” – to better organize the default application layout.

The application layout file includes a messages partial:

```
<%= render 'layouts/messages' %>
```

Examine the file **`app/views/layouts/_messages.html.erb`**:

```
<%=# Rails flash messages styled for Bootstrap 3.0 %>
<% flash.each do |name, msg| %>
  <% if msg.is_a?(String) %>
    <div class="alert alert-<%= name.to_s == 'notice' ? 'success' : 'danger' %>">
      <button type="button" class="close" data-dismiss="alert"
        aria-hidden="true">&times;</button>
      <%= content_tag :div, msg, :id => "flash_#{name}" %>
    </div>
  <% end %>
<% end %>
```

We use `each` to iterate through the flash hash, retrieving a `name` and `msg` that are passed to a block to be output as a string. The expression `if msg.is_a?(String)` serves as a test to make sure we only display messages that are strings. We construct a `div` that applies Bootstrap CSS styling around the message. Bootstrap recognizes a class `alert` to construct an alert box. A class of either `alert-success` or `alert-danger` styles the message. Rails `notice` messages will get styled with the Bootstrap `alert-success` class. Any other Rails messages, including `alert` messages, will get styled with the Bootstrap `alert-danger` class.

We use the Rails `content_tag` view helper to create a div containing the message.

Finally, we create a “close” icon by applying the class `close` to a link. We use the HTML entity `&times;` (a big “X” character) for the link; it could be the word “close” or anything else we like. Bootstrap’s integrated JavaScript library will hide the alert box when the “close” link is clicked.

Bootstrap provides [detailed documentation](#) if you want to change the styling of the alert boxes.

## Navigation Partial with Bootstrap

Every website needs navigation links. For our application, we want links for Home and About.

The application layout file includes a navigation partial:

```
<%= render 'layouts/navigation' %>
```

The layout and styling required for the Bootstrap navigation bar are in the navigation partial file.

Examine the file **app/views/layouts/\_navigation.html.erb**:

```
<%=# navigation styled for Bootstrap 3.0 %>
<nav class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <%= link_to 'Home', root_path, class: 'navbar-brand' %>
    </div>
    <div class="collapse navbar-collapse">
      <ul class="nav navbar-nav">
        <%= render 'layouts/navigation_links' %>
      </ul>
    </div>
  </div>
</nav>
```

The navigation partial includes layout and Bootstrap classes needed to produce a responsive navigation bar.

The responsive navigation bar adjusts to different browser widths. At small sizes, the navigation links will disappear and be replaced by a menu icon. Clicking the icon will reveal a vertical menu of navigation links. The navigation menu is a great demonstration of the ability of Bootstrap to adjust to the small screen size of a tablet or smartphone.

If you'd like to add a site name or logo to the tutorial application, you can replace the link helper `<%= link_to 'Home', root_path, class: 'navbar-brand' %>`.

We wrap the nested partial `render 'layouts/navigation_links'` with Bootstrap layout and classes to complete the navigation bar.

## Navigation Links Partial

The file **app/views/layouts/\_navigation\_links.html.erb** is very simple:

```
<%= add navigation links to this file %>
<li><%= link_to 'About', page_path('about') %></li>
```

You can add links to this file, for example:

```
<%= add navigation links to this file %>
<li><%= link_to 'About', page_path('about') %></li>
<li><%= link_to 'Contact', new_contact_path %></li>
```

The navigation links partial is simply a list of navigation links. It doesn't require additional CSS styling. By separating the links from the styling that creates the navigation bar, we segregate the code that is unique to Bootstrap. In the future, if the Bootstrap layout or CSS classes change, we can make changes without touching the navigation links.

Our application layout is set up for Bootstrap with flash messages and navigation links. Next we'll look at adding additional pages using the [high\\_voltage gem](#).

## Chapter 13

# Adding Pages

Let's add a page to our web application.

Most websites contain pages such as:

- “About” page
- Contact page
- Legal page
- FAQ page

We call these “static pages” because the content does not vary (they would be “dynamic pages” if the content was obtained from a database or varied depending on a visitor's interaction).

It's possible to place these pages in the **public/** folder and copy the HTML and CSS from the default application layout but this leads to duplicated code and maintenance headaches. And dynamic elements such as navigation links can't be included. For these reasons, developers seldom create static pages in the **public/** folder.

We can use Rails to create a page that uses no model, a nearly-empty controller, and a view that contains no instance variables. This solution is implemented so frequently that many developers create a gem to encapsulate the functionality. We'll use the best-known of these gems to create an “About” Page, the [high\\_voltage gem](#) created by the [Thoughtbot](#) consulting firm.

## HighVoltage Gem

We can add a page using the HighVoltage gem almost effortlessly. The gem implements Rails “convention over configuration” so well that there is nothing to configure. There are alternatives to its defaults which can be useful but we won't need them; visit the [GitHub](#) page for the [high\\_voltage gem](#) if you want to explore all the options.

In your **Gemfile**, you already have:

```
gem 'high_voltage'
```

# Views Folder

The starter app contains a folder **app/views/pages**:

```
$ mkdir app/views/pages
```

Any view files we add to this directory will automatically use the default application layout and appear when we use a URL that contains the filename.

The HighVoltage gem contains all the controller and routing magic required for this to happen.

## “About” Page

The starter app contains a file **app/views/pages/about.html.erb**:

```
<% content_for :title do %>About<% end %>
<h3>About the Website</h3>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua.</p>
```

Our simple “About” view will be combined with the default application layout by the HighVoltage gem.

We include a `content_for` Rails view helper that passes a page title to the application layout.

## Routing for the HighVoltage Gem

The HighVoltage gem provides a PagesController. You’ll never see it; it is packaged inside the gem.

In addition to providing a controller, the HighVoltage gem provides default routing so any URL with the form <http://localhost:3000/pages/about> will obtain a view from the **app/views/pages** directory.

Like the PagesController, the code that sets up the route is packaged inside the gem. If we wanted to add the route explicitly to the file **config/routes.rb**, the file would look like this:



```
LearnRails::Application.routes.draw do
  get "/pages/*id" => 'pages#show'
  root :to => 'visitors#new'
end
```

Again, you don't need to add the code above because the HighVoltage gem already provides the route.

You can use a Rails route helper to create a link to any view in the **app/views/pages** directory like this:

```
link_to 'About', page_path('about')
```

We already have a link to the “About” page in the navigation partial so we can test the application immediately.

## Test the Application

The web server may already be running. If not, enter the command:

```
$ rails server
```

Open a web browser window and navigate to <http://localhost:3000/>.

The link to the “About” page should work.

At this point, you've examined a basic starter application that integrates Bootstrap and Rails. You can use this starter application as a foundation for your own projects that require Bootstrap and Rails. You can use the [Rails Composer](#) tool anytime you need a starter application.

## Chapter 14

# Exploring Bootstrap

Bootstrap is a rich toolkit for front-end design. Explore the [Bootstrap documentation](#) to see all the design elements that are available.

So far, our tutorial has shown how to integrate Bootstrap with Rails but we've only created a home page, an "About" page, and used Bootstrap to style a navigation bar. It's a good starter app you can use for any number of projects. But we haven't explored many of the design elements offered by Bootstrap.

In the chapters to come, we'll go beyond the starter application and implement a simple survey website. Features include:

- Graphic carousel
- Survey form
- Google spreadsheet datastore
- Modal window

With the survey website example, we'll only scratch the surface of all the possibilities offered by Bootstrap, but you'll see how a few interesting design elements are added to your starter application, making it easier to add others on your own.

## Additional Gems

We'll need additional gems to implement our example. These gems are not needed in the starter application, so we didn't add them earlier. But we'll add them now so we can learn more about Bootstrap.

Here are gems we'll add to the Gemfile:

- [activerecord-tableless](#) – makes Rails easier without a database
- [google\\_drive](#) – use Google Drive spreadsheets for data storage

Open your **Gemfile** and replace the contents with the following:

```

source 'https://rubygems.org'
ruby '2.1.1'
gem 'rails', '4.1.1'
gem 'sqlite3'
gem 'sass-rails', '~> 4.0.1'
gem 'uglifier', '>= 1.3.0'
gem 'coffee-rails', '~> 4.0.0'
gem 'jquery-rails'
gem 'turbolinks'
gem 'jbuilder', '~> 2.0'
gem 'sdoc', '~> 0.4.0',      group: :doc
gem 'spring',              group: :development
gem 'activerecord-tableless'
gem 'bootstrap-sass'
gem 'google_drive'
gem 'high_voltage'
gem 'simple_form'
group :development do
  gem 'better_errors'
  gem 'binding_of_caller', :platforms=>[:mri_19, :mri_20, :rbx]
  gem 'quiet_assets'
  gem 'rails_layout'
end

```

Check for the [current version of Rails](#) and replace `gem 'rails', '4.1.1'` accordingly. The Rails default gems may have changed as well.

## Install the Gems

Now that you've edited the Gemfile, run `bundle install`:

```
$ bundle install
```

You can check which gems are loaded into the development environment:

```
$ gem list
```

You'll need to restart your web server before you can use the new gems.

## Form Helpers

Rails provides a set of view helpers for forms. They are described in the [RailsGuides: Rails Form Helpers](#) document. Many developers use an alternative set of form helpers named

SimpleForm, provided by the [SimpleForm gem](#). The SimpleForm helpers are more powerful, easier to use, and offer an option for styling with Bootstrap.

The [SimpleForm and Twitter Bootstrap Demo](#) shows how the form helpers work with Bootstrap.

In your Gemfile, you've already got:

```
gem 'simple_form'
```

You already ran `$ bundle install`.

You need to run a generator to install SimpleForm with a Bootstrap option:

```
$ rails generate simple_form:install --bootstrap
```

which installs several configuration files:

```
config/initializers/simple_form.rb
config/initializers/simple_form_bootstrap3.rb
config/locales/simple_form.en.yml
lib/templates/erb/scaffold/_form.html.erb
```

Here the SimpleForm gem uses the generator command to create files for initialization and localization (language translation). SimpleForm can be customized with settings in the initialization file.

We're ready to configure the application.

## Configuration File

To implement our survey website and save data submitted by the visitors, we'll need to configure an account name and password for Google Drive access.

We'll set credentials for your Google account in the **config/secrets.yml** file.

```
development:
  gmail_username: <%= ENV["GMAIL_USERNAME"] %>
  gmail_password: <%= ENV["GMAIL_PASSWORD"] %>
  secret_key_base: (not shown)

test:
  secret_key_base: (not shown)

# Do not keep production secrets in the repository,
# instead read values from the environment.
production:
  gmail_username: <%= ENV["GMAIL_USERNAME"] %>
  gmail_password: <%= ENV["GMAIL_PASSWORD"] %>
  secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>
```

Set Unix environment variables for `GMAIL_USERNAME` and `GMAIL_PASSWORD`. Refer to the [Learn Ruby on Rails](#) book to learn how to set Unix environment variables. If your password contains punctuation characters, use single quotes around the password when you set the environment variable.

If you have a [Gmail](#) account, your username and password are the credentials you use to access Google Drive. If you don't have a Gmail account, you can [sign up for a Google account](#) for free.

Some Google accounts require 2-step verification, which sends a unique code to your mobile phone each time you log in from an unfamiliar device. If your Google account requires two-factor authentication, you have three choices:

- [set up an application-specific password](#)
- turn off 2-step verification
- create a new Gmail account for use with this tutorial

Configuration values in the **config/secrets.yml** file are available anywhere in the application as variables, for example as `Rails.application.secrets.gmail_username`.

If you don't want to use Unix environment variables, you can set each value directly in the **config/secrets.yml** file. The file must be in your git repository when you deploy to Heroku. However, you shouldn't save the file to a public GitHub repository where other people can see your credentials.

## Git

Let's commit our changes to the Git repository.

```
$ git add -A  
$ git commit -m "add gems"  
$ git push origin master
```

We're ready to improve the application home page.

## Chapter 15

# Carousel

A *carousel* is a design element that is ideal for displaying a series of images.

The carousel is an impressive effect that can be implemented in a few lines of code using the Bootstrap JavaScript library.

We've already created a home page with a Visitors controller and a view file, as well as routing. In this chapter, we'll modify the Visitors#New view file to add an image carousel.

## User Story

We'll plan our work with a user story:

```
*Home page*
As a visitor to the website
I want to see a slideshow of images
so I can pick my favorite
```

In the next chapter we'll add a survey form for the visitor to vote on favorite images.

## Add an Image Carousel

The Bootstrap JavaScript library provides more than a dozen design elements that can be implemented in a few lines of HTML. You can see the documentation for [JavaScript in Bootstrap](#). One of the elements is the [Bootstrap Carousel](#).

Adding the carousel is as easy as copying the Bootstrap example and adding images.

Earlier, in the "Home Page" chapter, you moved the the home page from **app/views/visitors/index.html.erb** to **app/views/visitors/new.html.erb**. Now we'll add an image carousel to the home page.

Replace the contents of the file **app/views/visitors/new.html.erb**:

```

<% content_for :title do %>Rails Bootstrap<% end %>
<% content_for :description do %>Rails Bootstrap Example<% end %>
<div id="myCarousel" class="carousel slide" data-ride="carousel">
  <ol class="carousel-indicators">
    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
    <li data-target="#myCarousel" data-slide-to="1"></li>
    <li data-target="#myCarousel" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="item active">
      
      <div class="carousel-caption text-center">
        <h4>San Francisco</h4>
      </div>
    </div>
    <div class="item">
      
      <div class="carousel-caption text-center">
        <h4>Sydney</h4>
      </div>
    </div>
    <div class="item">
      
      <div class="carousel-caption text-center">
        <h4>Paris</h4>
      </div>
    </div>
  </div>
  <a class="left carousel-control" href="#myCarousel" data-slide="prev"><span
class="glyphicon glyphicon-chevron-left"></span></a>
  <a class="right carousel-control" href="#myCarousel" data-slide="next"><span
class="glyphicon glyphicon-chevron-right"></a>
</div>

```

We include `content_for` view helpers that pass a title and description to the application layout.

We add photos using the `<img>` tag. We're taking a shortcut and using placeholder photos from the [lorempixel.com](http://lorempixel.com) service.

Each photo has a caption set by the `<h4>` tag and the `<div class="carousel-caption">` style. We've added the `text-center` class to the example code we copied from the Bootstrap example code.

The best way to understand the code is to remove elements and refresh the page to see the changes. Removing the `<ol class="carousel-indicators">` section will remove the little dots in the lower center of the image that provide image selection.

You can remove the navigation arrows by removing the section:



```
<a class="left carousel-control" href="#myCarousel" data-slide="prev"><span
class="glyphicon glyphicon-chevron-left"></span></a>
<a class="right carousel-control" href="#myCarousel" data-slide="next"><span
class="glyphicon glyphicon-chevron-right"></a>
```

You can add another image by adding a `<div class="item">` section. Additional placeholder images are available by changing the last number in the image URL.

## Photo Options

You might wish to modify the placeholder photo. If you don't like the photos I've selected, try <http://lorempixel.com/1170/600/cats/1> or any other categories from the [lorempixel.com](http://lorempixel.com) service. You can change the size by modifying the dimensions from 1170 (pixel width) by 800 (pixel height).

You can replace the placeholder photo with your own by creating an **app/assets/images** folder and adding images. Instead of the HTML `<img>` tag, use the Rails `image_tag` view helper, like this:

```
<%= image_tag "myphoto.jpg" %>
```

## Test the Application

The web server may already be running. If not, enter the command:

```
$ rails server
```

Open a web browser window and navigate to <http://localhost:3000/>.

You should see the image carousel.

## Git

Let's commit our changes to the Git repository.

```
$ git add -A
$ git commit -m "home page carousel"
$ git push origin master
```

In the next chapter, we'll add a survey form.

## Chapter 16

# Survey Form

Aside from links, forms are the most common way for users to interact with web applications. The combination of Rails form handling and Bootstrap form styling greatly improves the website user experience.

In this chapter we'll add a survey form to the home page. Our primary purpose is to explore how Bootstrap applies styling to form elements.

You'll recall that we set up the [SimpleForm gem](#) when we added Bootstrap to our application. The SimpleForm gem gives us view helpers to generate the HTML required by forms. Forms contain some of the most complex HTML a developer will encounter, so the SimpleForm gem is truly worthwhile. You can see a demonstration of [SimpleForm and Twitter Bootstrap](#) for all the possible form elements.

## User Story

Let's plan our work with a user story:

```
*Survey Form*
As a visitor to the website
I want to select a favorite image
And submit a comment
so my opinion is known
```

There are many ways to implement a selection system, such as clicking a “like” icon or selecting from a “star” matrix. We'll simply implement a form with a select box and a comment field.

## Visitor Model

We'll need a Visitor model before we create the form. Strictly speaking, we could create a form without a model, but the model gives us the benefits of Rails form validation provided by the ActiveRecord API.

Create a file **app/models/visitor.rb**:

```
class Visitor < ActiveRecord::Base
  has_no_table
  column :favorite, :string
  column :comment, :string
  validates_presence_of :favorite

  IMAGE_LABELS = ['San Francisco', 'Sydney', 'Paris']

end
```

We inherit behavior from the ActiveRecord parent class. We are not using a database for our tutorial application, so we use the `has_no_table` keyword from the [activerecord-tableless](#) gem to disable the database features of ActiveRecord.

*Note:* There's another way to create a model without a database using only the [ActiveModel](#) class, described in the [RailsCasts: ActiveModel](#) screencast. Either approach is fine; we're using the `activerecord-tableless` gem because a tableless implementation using ActiveModel requires an understanding of Ruby modules, get and set methods, and object initialization. It's easier to use the `activerecord-tableless` gem.

We create attributes for `favorite` and `comment`.

Validation requirements for our survey form are simple. We want to make sure a "favorite" is selected and we don't care if a comment is missing.

Finally, we create a constant named `IMAGE_LABELS` that returns an array containing a name for each image the visitor will see on the home page. We'll use this array in several places, to provide a caption for each image, and to populate a collection of radio buttons, so it makes sense to keep our code DRY (*Don't Repeat Yourself*) by setting a constant in the model. There are other ways to approach this, including hard coding the variables in the view, or setting an instance variable in the controller, but the preferred Rails approach is to define data values in a model.

By convention, Ruby constants are styled in uppercase with underscores to separate words. To obtain the constant value elsewhere in our code, we'll use the syntax

```
Visitor::IMAGE_LABELS.
```

## Modify the Visitors Controller

We already have a Visitors controller that contains a simple `new` method. We'll change the `new` method, add a `create` method, and provide a `secure_params` private method to secure the controller from mass assignment exploits.

Replace the contents of the file **`app/controllers/visitors_controller.rb`**:

```

class VisitorsController < ApplicationController

  def new
    @visitor = Visitor.new
  end

  def create
    @visitor = Visitor.new(secure_params)
    if @visitor.valid?
      flash.now[:notice] = "Chose #{@visitor.favorite}."
      render :new
    else
      render :new
    end
  end

  private

  def secure_params
    params.require(:visitor).permit(:favorite, :comment)
  end

end

```

The controller `new` action will instantiate an empty `Visitor` model, assign it to the `@visitor` instance variable, and render the **`app/views/visitors/new.html.erb`** view.

`SimpleForm` will set a destination URL that corresponds to the `VisitorsController#create` action. The `create` method will instantiate a new `Visitor` model using the data from the form (we take steps to avoid security vulnerabilities first—more on that below).

The ActiveRecord parent class provides a method `valid?` which we can call on the `Visitor` model. Our conditional statement `if @visitor.valid?` checks the validation requirements we've set in the model.

If the validation check succeeds, we set a flash notice and redisplay the home page.

If the the validation check fails, we redisplay the home page and the form will highlight errors.

## Mass-Assignment Vulnerabilities

Rails protects us from a class of security exploits called “mass-assignment vulnerabilities.” Rails won't let us initialize a model with just any parameters submitted on a form. Suppose we were creating a new user and one of the user attributes was a flag allowing administrator access. A malicious hacker could create a fake form that provides a user name and sets the administrator status to “true.” Rails forces us to “white list” each of the parameters used to initialize the model.

We create a private method named `secure_params` to screen the parameters sent from the browser. The `params` hash contains two useful methods we use for our screening:

- `require(:visitor)` – makes sure that `params[:visitor]` is present
- `permit(:favorite, :comment)` – our “white list”

With this code, we make sure that `params[:visitor]` only contains `::favorite, :comment`. If other parameters are present, they are stripped out. Rails will raise an error if a controller attempts to pass params to a model method without explicitly permitting attributes via `permit`.

For our simple survey form, we don’t need to be concerned with mass-assignment vulnerabilities, as there is nothing to compromise. But our example follows Rails recommended practice because you may decide to adapt the form if you use the tutorial application as a starter app for your own project.

## Add a Form to the Home Page

Earlier we built a home page that contained an image carousel. We’ll add a form to the home page so a visitor can vote for a favorite image.

Replace the contents of the file **`app/views/visitors/new.html.erb`**:

```

<% content_for :title do %>Rails Bootstrap<% end %>
<% content_for :description do %>Rails Bootstrap Example<% end %>
<div id="myCarousel" class="carousel slide" data-ride="carousel">
  <ol class="carousel-indicators">
    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
    <li data-target="#myCarousel" data-slide-to="1"></li>
    <li data-target="#myCarousel" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="item active">
      
      <div class="carousel-caption text-center">
        <h4><%= Visitor::IMAGE_LABELS[0] %></h4>
      </div>
    </div>
    <div class="item">
      
      <div class="carousel-caption text-center">
        <h4><%= Visitor::IMAGE_LABELS[1] %></h4>
      </div>
    </div>
    <div class="item">
      
      <div class="carousel-caption text-center">
        <h4><%= Visitor::IMAGE_LABELS[2] %></h4>
      </div>
    </div>
  </div>
  <a class="left carousel-control" href="#myCarousel" data-slide="prev"><span
class="glyphicon glyphicon-chevron-left"></span></a>
  <a class="right carousel-control" href="#myCarousel" data-slide="next"><span
class="glyphicon glyphicon-chevron-right"></span></a>
</div>
<div class="text-center">
  <br />
  <%= simple_form_for @visitor do |f| %>
    <%= f.error_notification %>
    <div class="container">
      <div class="row">
        <div class="col-md-4 col-md-offset-4">
          <%= f.input :favorite, :collection => Visitor::IMAGE_LABELS,
            label: false, as: :select, prompt: 'Select a favorite...' %>
        </div>
      </div>
      <div class="row">
        <div class="col-md-4 col-md-offset-4">
          <%= f.input :comment, label: false, :placeholder => 'Add a comment...',
            :class => "col-md-12" %>
        </div>
      </div>
    </div>
  </div>

```

```

</div>
<div class="row">
  <div class="col-md-4 col-md-offset-4">
    <%= f.submit "Choose!", :class => "btn btn-primary"%>
  </div>
</div>
</div>
<% end %>
</div>

```

## Carousel

We've added a Visitors model so we obtain the caption for each image from the `IMAGE_LABELS` constant we've created in the Visitors model. The syntax `Visitor::IMAGE_LABELS[0]` gives us the value for the first item in the `IMAGE_LABELS` array (Ruby counts array items starting from zero).

## Survey Form

We add a div and apply the Bootstrap `text-center` class for our form.

The code for the form is compact but complex.

We've mixed in a variety of divs for the form layout, using the Bootstrap grid classes. It's a lot of clutter but it stacks three form elements at the center of the page.

We see several form elements:

- `simple_form_for` is the view helper for the form

The `simple_form_for` view helper instantiates a form object which we assign to a variable named `f`. SimpleForm offers many standard form elements, such as text fields and submit buttons. Each element is available as a method call on the form object.

The view helper `simple_form_for` requires *parameters* and a *block*.

We initialize the form with the `@visitor` instance variable. SimpleForm uses this parameter to name the form, set a destination for the form data (the `VisitorsController#create` action), and initialize each field in the form using attributes from the Visitor model. Setting the values for the form fields from the attributes in the model is called "binding the form to the object" and you can read about it in the [RailsGuides: Form Helpers](#) article.

The `simple_form_for` view helper accommodates a Ruby block. The block begins with `do` and closes with `end`.

Inside the block, the `form` object methods generate HTML for:

- error notifications
- a select box
- a field for “comment”
- a submit button

Each of the form methods takes various parameters.

The first input field creates a select box from parameters:

- `:favorite` – the field name matching an attribute in the Visitor model
- `:collection => Visitor::IMAGE_LABELS` – choices from an array in the Visitor model
- `label: false` – don’t display a label for the field
- `as: :select` – display the choices as a drop-down select menu
- `prompt: 'Select a favorite...'` – a prompt that appears above the selections

If you study the [SimpleForm documentation](#), you’ll see that `as: :select` can be replaced with `as: :radio_buttons` or `as: :check_boxes`. You can use radio buttons if you prefer.

The second input field creates a comment field from parameters:

- `:comment` – the field name matching an attribute in the Visitor model
- `label: false` – don’t display a label for the field
- `:placeholder` – a user prompt

The final method creates a submit button, setting the text to display in the button and applying the Bootstrap classes `btn btn-primary`.

The structure of the form is clearly visible in the code. The form begins with a `simple_form_for` helper and closes with the `end` keyword. Each line of code produces an element in the form such as a field or a button.

## Routing

We’ll modify the routing to accommodate the new controller actions.

In addition to rendering the `Visitors#new` view as the application root (the home page), we need to handle the `create` action. We can use Rails [resourceful routing](#) to define two new routes in a single line of code.

We’ll use two of the seven “resourceful” controller actions:

- *new* – display the home page with the empty visitor form



- `create` – validate and process the submitted form

Open the file **config/routes.rb**. Replace the contents with this:

```
RailsBootstrap::Application.routes.draw do
  resources :visitors, only: [:new, :create]
  root :to => 'visitors#new'
end
```

The root path remains `visitors#new`. Order is significant in the **config/routes.rb** file. As the final designated route, the root path will only be active if nothing above it matches the route.

We've added `visitors :contacts, only: [:new, :create]`.

We only want two routes so we've added the restriction `only: [:new, :create]`.

The `new` route has these properties:

- `new_visitor_path` – route helper
- `visitors` – name of the controller (VisitorsController)
- `new` – controller action
- <http://localhost:3000/visitors/new> – URL generated by the route helper
- `GET` – HTTP method to display a page

The `create` route has these properties:

- `visitors_path` – route helper
- `visitors` – name of the controller (VisitorsController)
- `create` – controller action
- <http://localhost:3000/visitors> – URL generated by the route helper
- `POST` – HTTP method to submit form data

You can run the `rake routes` command to see these in the console:

```
$ rake routes
  Prefix Verb URI Pattern               Controller#Action
  visitors POST /visitors(.:format)    visitors#create
  new_visitor GET /visitors/new(.:format) visitors#new
  root GET /                          visitors#new
  page GET /pages/*id                high_voltage/pages#show
```

The output of the `rake routes` command shows we've created the routes we need.

# Test the Application

You've modified the routing, so stop the web server with Control-c and restart:

```
$ rails server
```

Open a web browser window and navigate to <http://localhost:3000/>.

You'll see our new home page with the survey form.

## Git

Let's commit our changes to the Git repository.

```
$ git add -A  
$ git commit -m "add form to home page"  
$ git push origin master
```

Next, let's implement a feature to save data to a spreadsheet.

## Chapter 17

# Spreadsheet Connection

In the last chapter, we added a survey form to the application home page. When a visitor submits the form, we display an acknowledgment message. To complete the implementation, we need a way to capture the data for later analysis or review.

We've purposely chosen not to implement a database application so we can keep this tutorial to a manageable length and focus on Bootstrap. Though many Rails applications are backed by databases, a database adds complexity to a Rails application. One of the requirements that adds complexity is authentication and authorization. If data is stored in an application database, we have to implement access control so only an administrator can view it.

Fortunately, Google Drive (formerly known as Google Docs) gives us an easy way to store and access our visitor data without a database. It's an elegant solution. We can use the Google Drive API (application programming interface) to save form data to a spreadsheet that is stored in Google Drive. We don't have to implement authentication and authorization in our application because Google Drive already manages user access. Our application will send the data to a spreadsheet and you (or colleagues or clients, if you wish) can access the data on Google Drive.

Most computer-literate people have some experience with spreadsheets. Making data available in a spreadsheet makes it easy for an administrator or a website owner to analyze or review the data.

## Google Drive Gem

We'll use the `google_drive` gem to connect to the spreadsheet and save data.

The [google\\_drive](#) gem is a Ruby library to read and write data to spreadsheets in Google Drive. It provides convenient Ruby methods to wrap the [Google Spreadsheets API](#). You can see all the features of the `google_drive` gem by reviewing the [google\\_drive gem API](#).

In your **Gemfile**, you've already added:

```
gem 'google_drive'
```

and previously run `$ bundle install`.

# Implementation

The code that saves data to the spreadsheet could be added to the controller. After all, the form data is received by the controller. It would be easy to add a private method to the controller that sends the data to the spreadsheet. In practice, Rails developers prefer to add code that manipulates data to the model.

Controllers should contain only enough code to instantiate a model and handle rendering. In principle, all data manipulation should be handled by a model. Saving to a spreadsheet is a data operation.

## Modify the Visitor Model

In keeping with the Rails mantra, “skinny controller, fat model,” we’ll add our spreadsheet code to the Visitor model.

Replace the contents of the file **app/models/visitor.rb**:

```
class Visitor < ActiveRecord::Base
  has_no_table
  column :favorite, :string
  column :comment, :string
  validates_presence_of :favorite

  IMAGE_LABELS = ['San Francisco', 'Sydney', 'Paris']

  def update_spreadsheet
    connection = GoogleDrive.login(Rails.application.secrets.gmail_username,
    Rails.application.secrets.gmail_password)
    ss = connection.spreadsheet_by_title('Rails-Bootstrap-Example')
    if ss.nil?
      ss = connection.create_spreadsheet('Rails-Bootstrap-Example')
    end
    ws = ss.worksheets[0]
    last_row = 1 + ws.num_rows
    ws[last_row, 1] = Time.now
    ws[last_row, 2] = self.favorite
    ws[last_row, 3] = self.comment
    ws.save
  end
end
```

We’ll call the new `update_spreadsheet` method from the controller.

The `google_drive` gem gives us a `GoogleDrive` object.

Create a connection to Google Drive by passing your credentials to the `login` method. Here's where we use the configuration variables we set in the **`config/secrets.yml`** file.

We look for a spreadsheet named "Rails-Bootstrap-Example." The first time we attempt to save data, the spreadsheet will not exist, so we use the `create_spreadsheet` method to create it. If it already exists, the `spreadsheet_by_title` method will find it.

A single spreadsheet file can contain multiple *worksheets*. We'll use only one worksheet to store our data, designated as "worksheet 0" (we count from zero).

Here the code gets a little tricky. You might expect the API to provide an "append row" method. In fact, we have to retrieve a count of rows, and then add one, to calculate the row number of the last empty row.

We add data on a cell-by-cell basis, by designating the row number and column number of a cell. We add the current date and time using the Ruby API method `Time.now` to the first cell in the last row. Then we add the `favorite` and `comment` attributes to the second and third columns (we refer to the current instance of the class by using the keyword "self").

Setting the cell value doesn't save the data. We explicitly call the worksheet `save` method to update the worksheet.

## Modify the Visitors Controller

Our Visitor model now has a method to save data to a spreadsheet.

We'll update the Visitors controller to save the data.

Replace the contents of the file **`app/controllers/visitors_controller.rb`**:

```

class VisitorsController < ApplicationController

  def new
    @visitor = Visitor.new
  end

  def create
    @visitor = Visitor.new(secure_params)
    if @visitor.valid?
      @visitor.update_spreadsheet
      flash.now[:notice] = "Chose #{@visitor.favorite}."
      render :new
    else
      render :new
    end
  end

  private

  def secure_params
    params.require(:visitor).permit(:favorite, :comment)
  end

end

```

We've added only one line, the `@visitor.update_spreadsheet` statement.

When the visitor submits the form, the `VisitorsController#create` action is called. The `create` method will instantiate a new `Visitor` model using the form data after securing the parameters. If the validation check succeeds, we save data to the spreadsheet, set a flash notice, and redisplay the home page.

In only a few lines of code, we've added data storage using Google Drive.

## Test the Application

Make sure the web server is running:

```
$ rails server
```

Open a web browser window and navigate to <http://localhost:3000/>.

You'll see the home page with the survey form.

Fill in the form by selecting a menu item, adding a comment, and clicking the submit button. You'll see an acknowledgment message.

You may notice a delay of several seconds before you see the acknowledgment message. As currently implemented, your application has to wait for Google's servers to respond before displaying the acknowledgment. In a more advanced Rails application, we'd reduce the delay for the user by implementing background processing or "queueing." Background processing adds complexity and is not part of the Rails API ([learn why](#)), so we'll simply tolerate the delay in this simple application.

Visit your Google Drive account ("Drive" is in the navigation bar when you visit the Google Search or Gmail home pages). You'll see a list of Google Drive files. The newest one will be a **Rails-Bootstrap-Example** spreadsheet. Open the file and you will see the data from the survey form. Whenever a visitor submits the survey form, the spreadsheet will update within seconds.

## Git

Let's commit our changes to the Git repository.

```
$ git add -A
$ git commit -m "save data to a spreadsheet"
$ git push origin master
```

We've got a fully functional survey website that stores data in a Google Drive spreadsheet.

We could deploy the application now. But let's explore another Bootstrap design element, the modal window.

## Chapter 18

# Modal Window

The functionality of the survey form is complete. Let's add another Bootstrap design element.

We can use Bootstrap to hide the survey form and reveal it when a button is clicked. We'll use a [lightbox](#) effect: The form will appear in a bright overlay with the page darkened behind it. Because no further action can be taken until the form is submitted or cancelled, we say the form appears in a "modal window." A [modal window](#) is a user interface element in desktop applications and on the web, commonly used to block action until the user responds to the message in the window.

## Add a Modal Window

Our "call to action" will be a big button that says, "Choose!" Clicking the button will open a modal window and invite the visitor to select a favorite and click a submit button.

Bootstrap gives us everything we need to implement this in a few lines of code.

Replace the contents of the file **app/views/visitors/new.html.erb**:



```

<% content_for :title do %>Rails Bootstrap<% end %>
<% content_for :description do %>Rails Bootstrap Example<% end %>
<div id="myCarousel" class="carousel slide">
  <ol class="carousel-indicators">
    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
    <li data-target="#myCarousel" data-slide-to="1"></li>
    <li data-target="#myCarousel" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="item active">
      
      <div class="carousel-caption text-center">
        <h4><%= Visitor::IMAGE_LABELS[0] %></h4>
      </div>
    </div>
    <div class="item">
      
      <div class="carousel-caption text-center">
        <h4><%= Visitor::IMAGE_LABELS[1] %></h4>
      </div>
    </div>
    <div class="item">
      
      <div class="carousel-caption text-center">
        <h4><%= Visitor::IMAGE_LABELS[2] %></h4>
      </div>
    </div>
  </div>
  <a class="left carousel-control" href="#myCarousel" data-slide="prev"><span
class="glyphicon glyphicon-chevron-left"></span></a>
  <a class="right carousel-control" href="#myCarousel" data-slide="next"><span
class="glyphicon glyphicon-chevron-right"></span></a>
</div>
<div class="text-center">
  <br />
  <div id="modal-form" class="modal" tabindex="-1" role="dialog"
aria-labelledby="myModalLabel" aria-hidden="true" style="display: <%=
@visitor.errors.any? ? 'block' : 'none';%>" >
    <div class="modal-dialog">
      <div class="modal-content">
        <%= simple_form_for @visitor do |f| %>
          <div class="modal-header">
            <button type="button" class="close" data-dismiss="modal"
aria-hidden="true">&times;</button>
            <h4 class="modal-title" id="myModalLabel">Select a City</h4>
          </div>
          <div class="modal-body">
            <div class="container">
              <div class="row">

```

```

        <div class="col-md-4 col-md-offset-1">
          <%= f.error_notification %>
          <%= f.input :favorite, :collection => Visitor::IMAGE_LABELS,
            label: false, as: :select, prompt: 'Select a favorite...' %>
        </div>
      </div>
      <div class="row">
        <div class="col-md-4 col-md-offset-1">
          <%= f.input :comment, label: false, :placeholder => 'Add a comment...',
            :class => "col-md-12" %>
        </div>
      </div>
    </div>
    <div class="modal-footer">
      <%= f.submit "Choose!", :class => "btn btn-primary" %>
      <button type="button" class="btn btn-default"
data-dismiss="modal">Close</button>
    </div>
  <% end %>
</div>
</div>
<div id="call-to-action">
  <button type="button" class="btn btn-primary btn-lg" data-toggle="modal"
href="#modal-form">Choose!</button>
</div>
</div>

```

The additional code includes HTML and CSS classes from Bootstrap that implement the modal window and sets up a button to reveal the hidden modal window.

Let's look at the button that reveals the window:

```

<div id="call-to-action">
  <button type="button" class="btn btn-primary btn-lg" data-toggle="modal"
href="#modal-form">Choose!</button>
</div>

```

Clicking the button reveals a form with the ID `#modal-form` because the attribute `data-toggle="modal"` is applied to the link.

It doesn't matter where the code for the form is placed in the file as the form will be hidden when the page is initially displayed and will appear as an overlay when revealed by a click on the appropriate link. For convenience, I've appended the code at the end of the file.

Let's look closely at the div that encloses the form:

```
<div id="modal-form" class="modal" tabindex="-1" role="dialog"
  aria-labelledby="myModalLabel" aria-hidden="true" style="display: <%=
@visitor.errors.any? ? 'block' : 'none';%>" >
```

We wrap the form in a div named `modal-form`. The name can be anything but it must match the `href="#modal-form"` found in the button that reveals the modal window.

The div named `modal-form` contains some complex code to handle validation errors. Normally, the form will be hidden when the page is rendered in the browser. That's fine for a first visit. But our Rails controller may detect a validation error after the visitor submits the form, in which case it will redisplay the page. This is a special case where we want the modal window to be forced open as soon as the page is rendered. We use the Ruby [ternary operator](#) as a fancy conditional statement that says, "if the `@visitor` object contains errors, display the div as a 'block', otherwise, set the display property to 'none'."

Next we set up the divs to wrap the modal content and create our form:

```
<div class="modal-dialog">
  <div class="modal-content">
    <%= simple_form_for @visitor do |f| %>
      .
      .
      .
    <% end %>
  </div>
</div>
```

The form contains a modal header element:

```
<div class="modal-header">
  <button type="button" class="close" data-dismiss="modal"
  aria-hidden="true">&times;</button>
  <h4 class="modal-title" id="myModalLabel">Select a City</h4>
</div>
```

The form has a section named "modal-header" that contains a link to close the modal window. We follow Bootstrap's example and use HTML entity `&times;` (an "x" character) for the link.

The next section is named "modal-body" and contains the form input fields, preceded by the `<%= f.error_notification %>` form helper. We adjust the grid to center the input fields with `class="col-md-4 col-md-offset-1"`.

Following the input fields is a section named "modal-footer" that contains a submit button and another link to close the modal window:

```
<div class="modal-footer">
  <%= f.submit "Choose!", :class => "btn btn-primary" %>
  <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
</div>
```

You can use similar code for a modal window for other projects by duplicating this structure.

You can read the Bootstrap documentation for more information about the [Bootstrap modal](#).

## Test the Application

Make sure the web server is running:

```
$ rails server
```

Open a web browser window and navigate to <http://localhost:3000/>.

You'll see the home page but the survey form will be hidden.

Click the big blue button and the modal window will be revealed.

You can fill out and submit the form with the same result as before.

## Git

Let's commit our changes to the Git repository.

```
$ git add -A
$ git commit -m "add modal to home page"
$ git push origin master
```

We've explored design elements provided by Bootstrap and seen how we can integrate the Bootstrap code with Rails. We've used two elements, the carousel and the modal, that are suitable for our simple survey website. Bootstrap has many other design elements that you can read about in the [Bootstrap documentation](#).

Our survey website is complete.

## Chapter 19

# Deploy

[Heroku](#) provides the best known and most popular hosting for Rails applications. Using Heroku or another *platform-as-a-service* provider means you'll have experts maintaining the production environment, tuning system performance, and keeping the servers running.

Heroku is ideal for hosting our application:

- no system administration expertise is required
- hosting is free
- performance is excellent

Our [Rails and Heroku](#) article goes into more detail, describing costs and deployment options.

Let's deploy!

## Preparing for Heroku

You'll need to prepare your Rails application for deployment to Heroku.

### Gemfile

We need to modify the Gemfile for Heroku.

We add a `group :production` block for gems that Heroku needs:

- [pg](#) – PostgreSQL gem
- [thin](#) – web server
- [rails\\_12factor](#) – logging and static assets

Heroku doesn't support the SQLite database; the company provides a PostgreSQL database. Though we won't need it for our tutorial application, we must include the PostgreSQL gem for Heroku. We'll mark the [sqlite3](#) gem to be used in development only.

The [Thin](#) web server is easy to use and requires no configuration. Note that [Heroku recommends Unicorn](#) for handling higher levels of traffic efficiently. Unicorn can be difficult to setup and configure, so we're using Thin for our tutorial application.

On Heroku, Rails needs an extra gem to handle logging and serve CSS and JavaScript assets. The [rails\\_12factor](#) gem provides these services.

Open your **Gemfile** and replace the contents with the following:

```
source 'https://rubygems.org'
ruby '2.1.1'
gem 'rails', '4.1.1'
gem 'sass-rails', '~> 4.0.1'
gem 'uglifier', '>= 1.3.0'
gem 'coffee-rails', '~> 4.0.0'
gem 'jquery-rails'
gem 'turbolinks'
gem 'jbuilder', '~> 2.0'
gem 'sdoc', '~> 0.4.0',      group: :doc
gem 'spring',              group: :development
gem 'activerecord-tableless'
gem 'bootstrap-sass'
gem 'google_drive'
gem 'high_voltage'
gem 'simple_form'
group :development do
  gem 'sqlite3'
  gem 'better_errors'
  gem 'binding_of_caller', :platforms=>[:mri_19, :mri_20, :rbx]
  gem 'quiet_assets'
  gem 'rails_layout'
end
group :production do
  gem 'pg'
  gem 'thin'
  gem 'rails_12factor'
end
```

We have to run `bundle install` because we've changed the Gemfile. The gems we've added are only needed in production so we don't install them on our local machine. When we deploy, Heroku will read the Gemfile and install the gems in the production environment. We'll run `bundle install` with the `--without production` argument so we don't install the new gems locally:

```
$ bundle install --without production
```

Commit your changes to the Git repository:

```
$ git add -A
$ git commit -m "gems for Heroku"
$ git push origin master
```

## Precompile Assets

In development mode, the Rails asset pipeline “live compiles” all CSS and JavaScript files and makes them available for use. Compiling assets adds processing overhead. In production, a web application would be slowed unnecessarily if assets were compiled for every web request. Consequently, we must precompile assets before we deploy our application to production.

When you precompile assets for production, the Rails asset pipeline will automatically produce concatenated and minified **application.js** and **application.css** files from files listed in the manifest files **app/assets/javascripts/application.js** and **app/assets/stylesheets/application.css.scss**. You must commit the compiled files to your git repository before deploying.

Here’s how to precompile assets and commit to the Git repo:

```
$ RAILS_ENV=production rake assets:precompile
$ git add -A
$ git commit -m "assets compiled for Heroku"
$ git push origin master
```

The result will be several files added to the **public/assets/** folder. The filenames will contain a long unique identifier that prevents caching when you change the application CSS or JavaScript.

If you don’t precompile assets for production, all web pages will look strange. They won’t use the Bootstrap CSS styling.

## Option to Ban Spiders

Do you want your website to show up in Google search results? If there’s a link anywhere on the web to your site, within a few days (sometimes hours) the Googlebot spider will visit your site and add it to the database for the Google search engine. Most webmasters want their sites to be found in Google search results. If that’s not what you want, you may want to add the file **public/robots.txt** to prevent indexing by search engines.

Only add this file if you want to prevent your website from appearing in search engine listings:

```
# public/robots.txt
# To allow spiders to visit the entire site comment out the next two lines:
User-Agent: *
Disallow: /
```

You can learn more about the format of the [robots exclusion standard](#).

## Humans.txt

Many websites include a **robots.txt** file for nosy bots so it's only fair that you offer a **humans.txt** file for nosy people. Few people will look for it but you can add a file **public/humans.txt** to credit and identify the creators and software behind the website. The HTML5 Boilerplate project offers an [example file](#) or you can [borrow from RailsApps](#).

## Sign Up for a Heroku Account

In the chapter, "Accounts You May Need," I suggested you sign up for a Heroku account.

To deploy an app to Heroku, you must have a Heroku account. Visit <https://id.heroku.com/signup/devcenter> to set up an account.

Be sure to use the same email address you used to configure Git locally. You can check the email address you used for Git with:

```
$ git config --get user.email
```

## Heroku Toolbelt

Heroku provides a command line utility for creating and managing Heroku apps.

Visit <https://toolbelt.heroku.com/> to install the Heroku Toolbelt. A one-click installer is available for Mac OS X, Windows, and Linux.

The installation process will install the Heroku command line utility. It also installs the [Foreman](#) gem which is useful for duplicating the Heroku production environment on a local machine. The installation process will also make sure Git is installed.

To make sure the Heroku command line utility is installed, try:

```
$ heroku version
heroku-toolbelt/...
```



You'll see the heroku-toolbelt version number.

You should be able to login using the email address and password you used when creating your Heroku account:

```
$ heroku login
Enter your Heroku credentials.
Email: adam@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/adam/.ssh/id_rsa.pub
```

The Heroku command line utility will create SSH keys if necessary to guarantee a secure connection to Heroku.

## Heroku Create

Be sure you are in your application root directory and you've committed the tutorial application to your Git repository.

Use the Heroku create command to create and name your application.

```
$ heroku create myapp
```

Replace `myapp` with something unique. Heroku demands a unique name for every hosted application. If it is not unique, you'll see an error, "name is already taken." Chances are, "rails-bootstrap" is already taken.

If you don't specify your app name ( `myapp` in the example above), Heroku will supply a placeholder name. You can easily change Heroku's placeholder name to a name of your choice with the `heroku apps:rename` command (see [Renaming Apps from the CLI](#)).

Don't worry too much about getting the "perfect name" for your Heroku app. The name of your Heroku app won't matter if you plan to set up your Heroku app to use your own domain name. You'll just use the name for access to the instance of your app running on the Heroku servers; if you have a custom domain name, you'll set up DNS (*domain name service*) to point your domain name to the app running on Heroku.

The `heroku create` command sets your Heroku application as a Git remote repository. That means you'll use the `git push` command to deploy your application to Heroku.

Next we'll set Heroku environment variables.

# Set Heroku Environment Variables

You'll need to set the configuration values from the **config/secrets.yml** file as Heroku environment variables.

Here's how to set environment variables directly on Heroku with `heroku config:add`.

```
$ heroku config:add GMAIL_USERNAME='myname@gmail.com' GMAIL_PASSWORD='secret'
```

You can check that the environment variables are set with:

```
$ heroku config
```

See the Heroku documentation on [Configuration and Config Vars](#) and the article [Rails Environment Variables](#) for more information.

## Push to Heroku

After all this preparation, you can finally push your application to Heroku.

Be sure you've run `RAILS_ENV=production rake assets:precompile`. Run it each time you change your CSS or JavaScript files.

Be sure to commit your code to the Git local repository before you push to Heroku.

You commit your code to Heroku just like you push your code to GitHub.

Here's how to push to Heroku:

```
$ git push heroku master
```

The push to Heroku takes several minutes. You'll see a sequence of diagnostic messages in the console, beginning with:

```
-----> Ruby/Rails app detected
```

and finishing with:

```
-----> Launching... done
```

# Visit Your Site

Open your Heroku site in your default web browser:

```
$ heroku open
```

Your application will be running at <http://my-app-name.herokuapp.com/>.

## Customizing

For a real application, you'll likely want to use your own domain name for your app.

See [Heroku's article about custom domains](#) for instructions.

You may also want to improve website responsiveness by adding page caching with a content delivery network such as [CloudFlare](#). CloudFlare can also provide an SSL connection for secure connections between the browser and server.

Heroku offers many [add-on services](#). These are particularly noteworthy:

- [Adept Scale](#) – automated scaling of Heroku dynos
- [New Relic](#) – performance monitoring

For an in-depth look at your options, see the [Rails Heroku Tutorial](#).

## Troubleshooting

When you get errors, troubleshoot by reviewing the log files:

```
$ heroku logs
```

If necessary, use the Unix `tail` flag to monitor your log files. Open a new terminal window and enter:

```
$ heroku logs -t
```

to watch the server logs in real time.

# Where to Get Help

Your best source for help with Heroku is [Stack Overflow](#). Your issue may have been encountered and addressed by others.

You can also check the [Heroku Dev Center](#) or the [Heroku Google Group](#).

## Chapter 20

# Comments

## Credits

Daniel Kehoe implemented the application and wrote the tutorial.

## Photos

Images provided by the [lorempixel.com](http://lorempixel.com) service are used under the [Creative Commons license](https://creativecommons.org/licenses/by-sa/4.0/) (CC BY-SA). Visit the Flickr accounts of the photographers to learn more about their work:

- photo of San Francisco by [Eneas De Troya](#)
- photo of Sydney by [Jimmy Harris](#)
- photo of Paris by [Archie Ballantine](#)

The photo of San Francisco by [Eneas De Troya](#) appears in the screenshot in the Introduction chapter and on the tutorial cover page.

## Did You Like the Tutorial?

Was the article useful to you? Follow [@rails\\_apps](#) on Twitter and tweet some praise. I'd love to know you were helped out by the article.

You can also find me on [Facebook](#) or [Google+](#).

Any issues? Please create an [issue](#) on GitHub. Reporting (and patching!) issues helps everyone.

