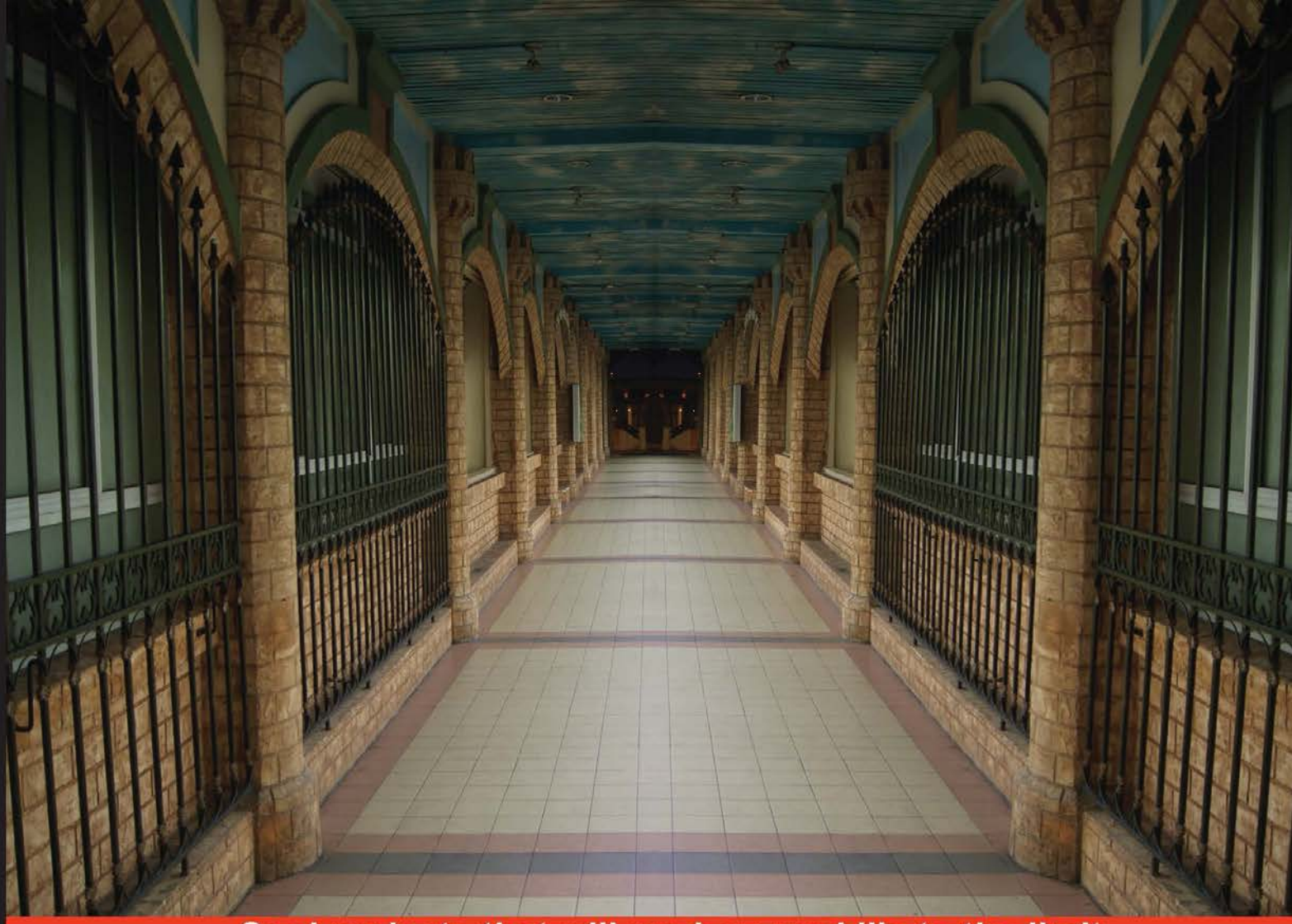


Cool projects that will push your skills to the limit

Rails 4 Application Development

Build simple to advanced applications in Rails 4 through 10 exciting projects

HOTSHOT

Saurabh Bhatia

[PACKT] PUBLISHING open source* community experience distilled

Rails 4 Application Development HOTSHOT

Build simple to advanced applications in Rails 4 through 10 exciting projects

Saurabh Bhatia

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Rails 4 Application Development HOTSHOT

First published: April 2014

Production Reference: 1030414

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78328-629-4

www.packtpub.com

Cover Image by Faiz Fattohi (faizfattohi@gmail.com)

Credits

Author

Saurabh Bhatia

Reviewers

Gabriel Hilal

Andrew Montgomery-Hurrell

Peter F. Philips

Philip De Smedt

Acquisition Editors

Nikhil Chinnari

Rubal Kaur

Content Development Editor

Priya Singh

Technical Editors

Venu Manthena

Mrunmayee Patil

Shruti Rawool

Copy Editors

Alisha Aranha

Mradula Hegde

Gladson Monteiro

Alfida Paiva

Project Coordinator

Leena Purkait

Proofreaders

Simran Bhogal

Maria Gould

Paul Hindle

Indexers

Rekha Nair

Priya Subramani

Production Coordinator

Aparna Bhagat

Cover Work

Aparna Bhagat

About the Author

Saurabh Bhatia has been developing professional software since 2005. However, his programming interests date back to his school days. Starting with Java, he quickly moved to Ruby on Rails in 2006, and it has been his primary choice of development framework since then. He built a Ruby on Rails consulting company and ran it for five years. He has worked with several companies in the tech industry, from getting two-person startups off the ground to developing software for large corporates. He is currently the CTO of Ruling Digital Inc., a software company that develops software for universities.

He has been an open source enthusiast and has helped Ubuntu penetrate the Indian market since 2007. He was a part of the open source promotion society called Twincling Society for Open Source in Hyderabad. He started and moderated Bangalore Ruby Users Group and also moderates the Mumbai Ruby Users Group. He is also a part of the RailsBridge initiative for mentoring new Rails developers.

Over the years, he has written several articles online and in print for different publications, such as *Linux User and Developer*, *Linux For You*, *Rails Magazine*, Developer.com (`http://www.developer.com/`), and SitePoint Ruby (`http://www.sitepoint.com/ruby/`). He currently resides in Taiwan. He wishes to continue writing and share his knowledge as much as possible with budding developers.

I would like to thank my parents, my sister, and my wife for being very understanding while I was writing this book. They have been pushing me to do better on this front and have inspired me to write more and more. I would also like to thank my boss for encouraging and supporting me during the process.

About the Reviewers

Gabriel Hilal is a full stack web developer who specializes in Ruby on Rails and related technologies. He has a bachelor's degree in Information Systems (Internet business) and a master's degree in Information Systems with Management Studies, both from Kingston University, London. During his time at the university, he developed a passion for Ruby on Rails and has since then done freelance work using behavior-driven development and agile methodologies to build high-quality Rails applications. Gabriel can be contacted on his website (`www.gabrielhilal.com`) or by e-mail at `gabriel@gabrielhilal.com`.

Andrew Montgomery-Hurrell is a software developer, hacker, and an all-round geek who enjoys everything from Dungeons and Dragons to DevOps. From an early age, he was fascinated with computers, and after cutting his teeth on BASIC with aging Amstrad CPCs and Amigas, he moved on to Linux admin, C/C++, followed by Python and then Ruby. Since the early 2000s, he has worked on a number of web applications in a range of languages and technologies, right from small company catalog sites to large web applications that serve thousands of people across the globe. Trained and interested in computing from the bottom up and coming from a background in electronics and computer interfacing, Andrew has experience in the full stack of computing technology, from ASICs to applications.

When he isn't working on web applications or infrastructure tools for gaming events and hosting company Multiplay, he can be found hacking code, reading or writing fiction, playing computer games, or slaying dragons with his wife, Laura.

Peter F. Philips is a software engineer, data scientist, and problem solver from New York City who now resides in San Francisco, CA. He is the founder of TechForProgress and cofounder of Planet (`http://planet.io/`) and Recognize (`https://recognizeapp.com/`) apps. Peter has been working with Ruby on Rails for seven years since Version 1.6. He is determined to use technology to improve the planet. In his spare time, Peter enjoys photography, hiking, rock climbing, and travelling to remote areas of the globe.

Philip De Smedt is a freelance full-stack developer and cofounder of Compete Hub, the definitive database of all endurance races. His main focus is on API-driven development using Rails and AngularJS. Philip is also the author of *Upgrading to Rails 4*, a step-by-step guide on upgrading your Rails 3 application to Rails 4. He is a Bitcoin and Dogecoin advocate and has spoken at multiple user groups on Rails and cryptocurrencies. When he's not coding or creating products, he likes to cycle, read books, or go for a run.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

In the past few years, Rails has emerged as one of the most popular choices of framework for developing web applications. It is also one of the most popular courses on all the major websites that teach web development, and a lot of developers have built a career out of it. Rails is known for providing productivity to developers and allows them to write clean and functional human-readable code. The latest major version of Rails, Rails 4, is a feature-packed update with a lot of new syntaxes and patterns.

Rails 4 Application Development Hotshot presents a practical approach to upgrade your Rails knowledge to Rails 4. This is done by building the most popular types of applications that people usually build using Rails and highlighting the new ways of doing this as opposed to the old ones in the latest version. The book also closely follows best practices and the commonly used gems and their compatibility with the latest Rails version. While working on these projects, we will also see some new design patterns and get ideas to refactor our current codebase. This book will help you write basic applications that are customizable and scalable and introduce you to a wide spectrum of concepts and ideas.

What this book covers

Project 1, *A Social Recipe-sharing Website*, explains how to create a website where many users can sign up, log in, create food recipes, and categorize them into different types.

Project 2, *Conference and Event RSVP Management*, explains how to create an application where users can create events, organize meetups for different topics and themes, and other users can join them in these events.

Project 3, *Creating an Online Social Pinboard*, covers how to create an online pinboard, where a user can pin whatever he/she likes on to it and organize these objects. These pins can be repinned by other users on to their pinboards and thus create an online collection of the things or objects that people like.

Project 4, *Creating a Restaurant Menu Builder*, covers how to build a fully responsive system to create and manage menus for a restaurant. This project will port restaurant menus to tablets and smartphones and also demonstrate how to make an SaaS application in Rails.

Project 5, *Building a Customizable Content Management System*, explains how to create a customizable content management system to power simple content-driven websites. We will effectively create a system where designers will have the freedom to choose the frontend they want and end users can easily manage the content for that frontend.

Project 6, *Creating an Analytics Dashboard using Rails and Mongoid*, will cover tracking clicks, page views, and the location of the visitors who read the content generated from the website. We will analyze the data and generate different types of graphs that represent different types of data.

Project 7, *Creating an API Mashup – Twitter and Google Maps*, will dive into an API mashup of Twitter and Google Maps that will generate an application to map the locations of your friends who are tweeting. We will also filter these people based on country names.

Project 8, *API Only Application – Backend for a Mobile App*, explains an application where the entire backend is in the form of an API. The entire data will be available on the frontend in the form of JSON through API endpoints. The frontend can be a web or mobile application.

Project 9, *Video Streaming Website using Rails and HTML5*, explains how to create an application to upload and encode videos. This application will allow visitors to stream and watch videos using an HTML5-based player.

Project 10, *A Rails Engines-based E-Commerce Platform*, explains how to create a Rails engine for generating an e-commerce application. This is mountable inside a blank Rails application.

What you need for this book

In order to work with the projects in this book, you will need the following installed on your system:

- Ruby 1.9.3
- Rails 4
- MySQL 5+
- MongoDB

- jQuery
- ImageMagick
- RMagick
- Git
- Morris.js
- Apache Solr
- Apache Tomcat
- Bootstrap
- Sass
- Sublime Text
- A tool for mock-ups
- Haml
- Memcached
- Twitter API keys
- Google Maps API keys
- The Rails API
- FFmpeg
- Redis
- Video.js
- A GitHub account
- Devise
- Doorkeeper

All projects have been upgraded and tested with Ruby 2.0 and Rails 4.1.0 beta.

Who this book is for

This book is aimed at developers who are already familiar with the basics of the Rails framework and have worked with Rails 3.2 or earlier versions. As the book follows a practical approach and uses terminology specific to Rails and web programming, it is assumed you have some prior experience with the development of applications. This book will help you upgrade your knowledge and improve its applicability.

Conventions

In this book, you will find several headings that appear frequently. To give clear instructions of how to complete a procedure or task, we use:

Mission briefing

This section explains what you will build, with a screenshot of the completed project.

Why is it awesome?

This section explains why the project is cool, unique, exciting, and interesting. It describes what advantage the project will give you.

Your Hotshot objectives

This section explains the eight major tasks required to complete your project:

- Task 1
- Task 2
- Task 3
- Task 4
- Task 5
- Task 6
- Task 7
- Task 8

Mission checklist

This section explains any prerequisites for the project, such as resources or libraries that need to be downloaded, and so on.

Task 1

This section explains the task that you will perform.

Prepare for lift off

This section explains any preliminary work that you may need to do before beginning work on the task.

Engage thrusters

This section lists the steps required in order to complete the task.

Objective complete - mini debriefing

This section explains how the steps performed in the previous section allow us to complete the task. This section is mandatory.

Classified intel

This section provides extra information that is relevant to the task.

You will also find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "In case the form validation fails, the `file` field is reset."

In all code blocks, the first line is the name of the file kept there for your reference, followed by the code. An example of a code block is shown as follows:

```
app/models/event.rb
class Event < ActiveRecord::Base
  belongs_to :organizers, class_name: "User"
end
```

Database migrations that appear in the book appear without the filename as the generated filename varies from system to system. Following is how it is defined in the book:

```
class AddPlanIdToUsers < ActiveRecord::Migration
  def change
    add_column :users, :plan_id, :integer
  end
end
```

Any command-line input or output is written as follows:

```
:~/pinpost$ rails g jquery:install
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "We are going to select **From Scratch** and build our wireframes using the given set of tools."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

Project 1

A Social Recipe-sharing Website

Food-recipe websites have been in existence since the advent of the Internet. `Food.com`, `thefoodnetwork.com`, and `bbcgoodfood.com` are some of most visited sites. Food is also one of the most popular searched categories on the Internet. Most of these websites have experts writing content for them. In this project, we will develop a website where amateur users can upload their recipes and those recipes can be viewed and shared by others and generated by several users. The recipes can be shared over various social networking sites by the readers.

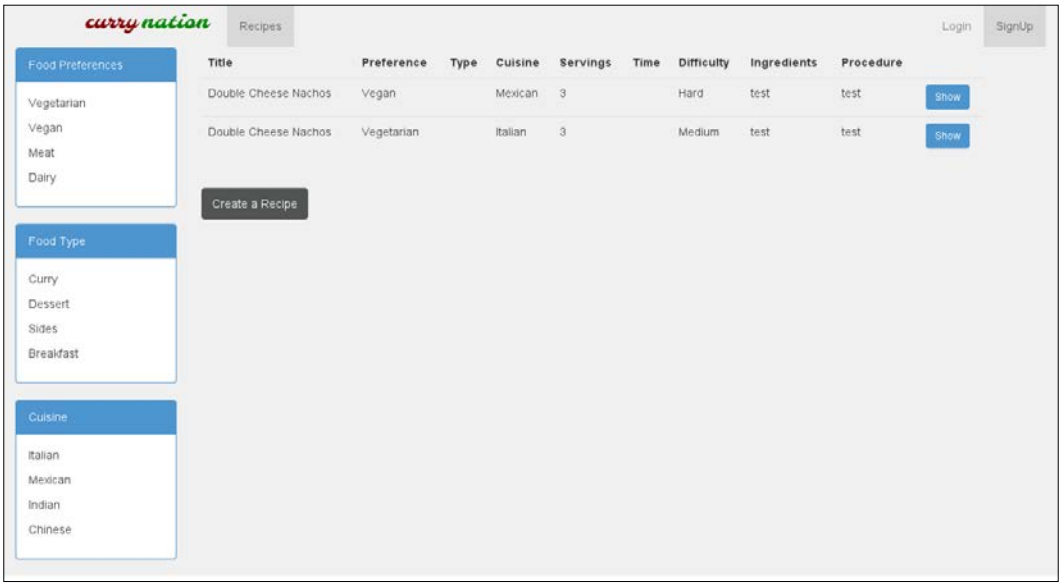
Mission briefing

Our goal is to create a very basic social website where users can sign up and create recipes they know the best. Other users can filter these recipes based on their interests, tastes, and food preferences and share it on Facebook, Twitter, or other social networking sites of their choice. At the end of this project, we should be able to perform the following tasks:

- Create an application
- Know what's the best way for creating an application
- Make use of some of the new features available for creating the application

User stories are a very important part of the entire project. They can make or break project schedules and have a drastic effect on the product in the long run. Once defined, our use cases will have steps on how a user interacts with the application and the validations required for it to pass. It will be much easier for us to keep this as a reference while coding. A good specification, both visual and technical, goes a long way in helping developers save time.

The home page will contain feed of the entire system—users who have newly joined the system, created new recipes, and edited new recipes. The screenshot of the home page of the final system is as follows:



Why is it awesome?

Everyone loves food, and some of us like to cook food too. The simplest and the most interesting way to build momentum for development is with a simple project. We will use this project to lay the foundation of Rails 4 comprehensively and build a base for the upcoming projects. Developers who have been using earlier versions of Rails will get a chance to work with new features in Version 4.0.0. Also, this will set the tone for the rest of the book, in terms of the process we will follow or we should follow while building our applications. We are following a test-driven development approach in the context of Rails 4. So, we will get a fair amount of exposure to the minitest framework, which has been newly introduced, and we will follow it up with some basics of ActiveRecord. While running through this, we will also work with Bootstrap 3.0 to style our views.

Your Hotshot objectives

While building this application, we will complete the following tasks:

- Creating mockups
- Adding test data and creating tests
- Adding categories

- Creating and adding recipes
- Creating associations – recipes and categories
- Adding authentication
- Beautifying your views

Mission checklist

We need the following software installed on the system before we start with our mission:

- Ruby 1.9.3 / Ruby 2.0.0
- Rails 4.0.0
- MySQL 6
- Bootstrap 3.0
- Sass
- Devise
- Git
- A tool for mockups; I personally use MockFlow

Creating mockups

Before we actually start developing the application, we will build two types of specifications: visual specifications called mockups and technical specifications called user stories. Visual imagination needs a fair bit of creativity and is best left to the designers; however, for our reference here, we will see how to create mockups in case you are working on an end-to-end process.

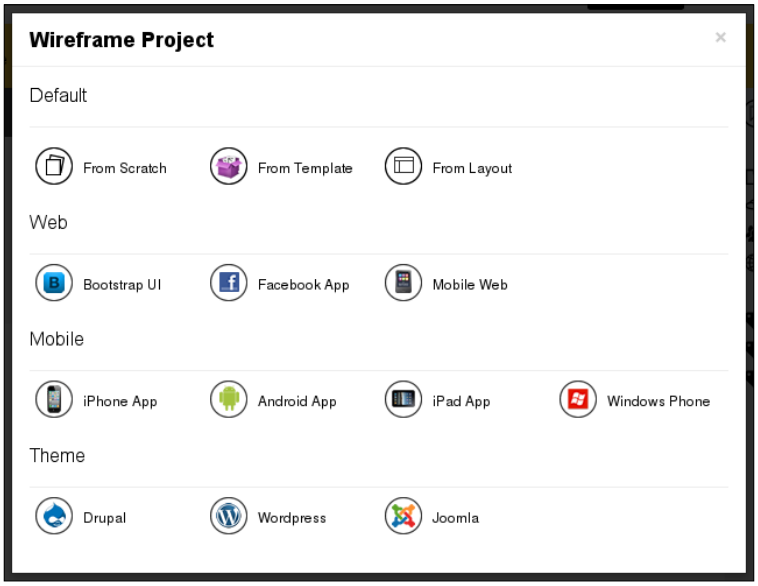
Prepare for lift off

There are several mockup tools available online and are free to download and install. **Balsamiq** (`https://www.mybalsamiq.com`), **MockFlow** (`http://mockflow.com`), and **mockingbird** (`https://gomockingbird.com/`) are some of the tools that I have explored and are fairly useful. We will use MockFlow for our projects. Sign up and create a free account with MockFlow.

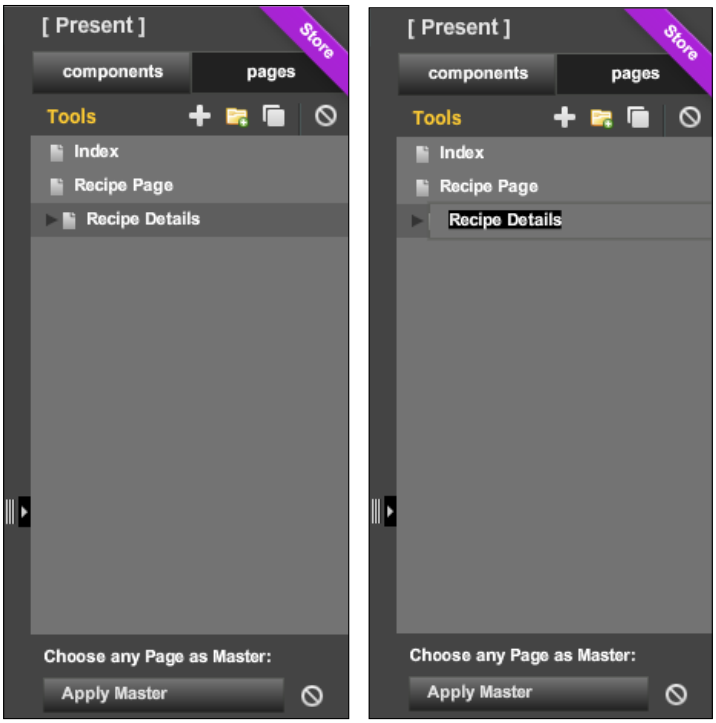
Engage thrusters

For creating mockups, we will perform the following steps:

1. Setting up a project in MockFlow is pretty straightforward. As soon as we log in to the account, we will be able to see an **Add Project** button. Once we click on it, the following screen shows up with various options for setting up different kinds of projects. We are going to select **From Scratch** and build our Wireframes using the given set of tools.

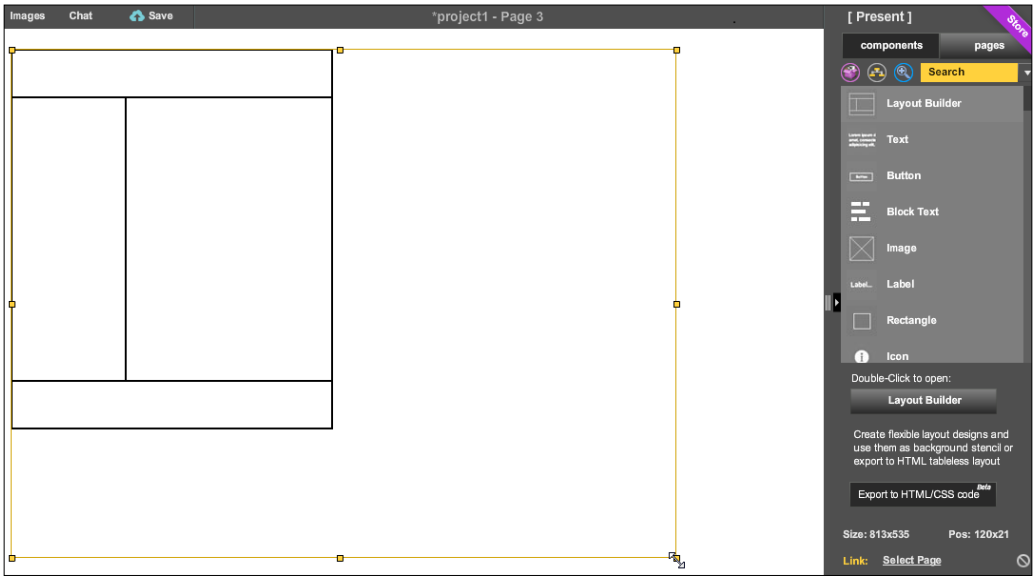


2. We will select the **From Scratch** option present under the **Wireframe Project** screen, name it, and proceed with the setup of the pages we want in our application.
3. The tool to the right contains two tabs:
    - **pages**: With this option, you can **Create**, **Sort**, **Duplicate**, and **Delete** pages in your application
    - **components**: With this option, the textboxes, text areas, scrollbars, logos, images, and different elements of the page can be simply dragged-and-dropped from the **component** panel to the canvas on the center of the page to create a Wireframe

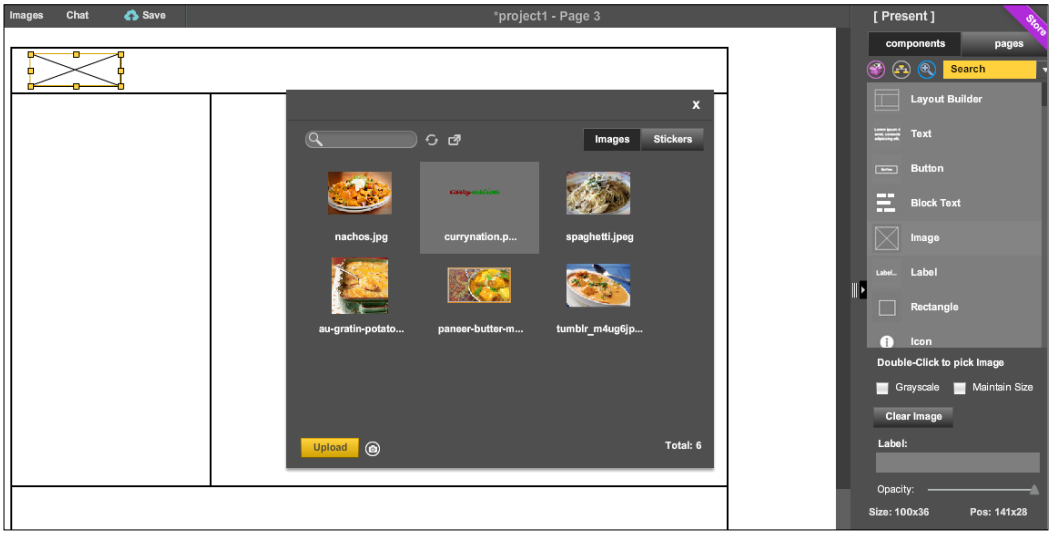


4. Let's start building our first mockup. Drag-and-drop the **Layout Builder** icon located in the **components** panel, and using your mouse, create and resize it so it fits on the page.
5. This layout suits our application needs because our aim is to build an application with a filter bar to the left that would allow users to filter categories with ease. The central portion will display the content and will contain the list of various recipes. The portion to the left will contain the list of various categories.

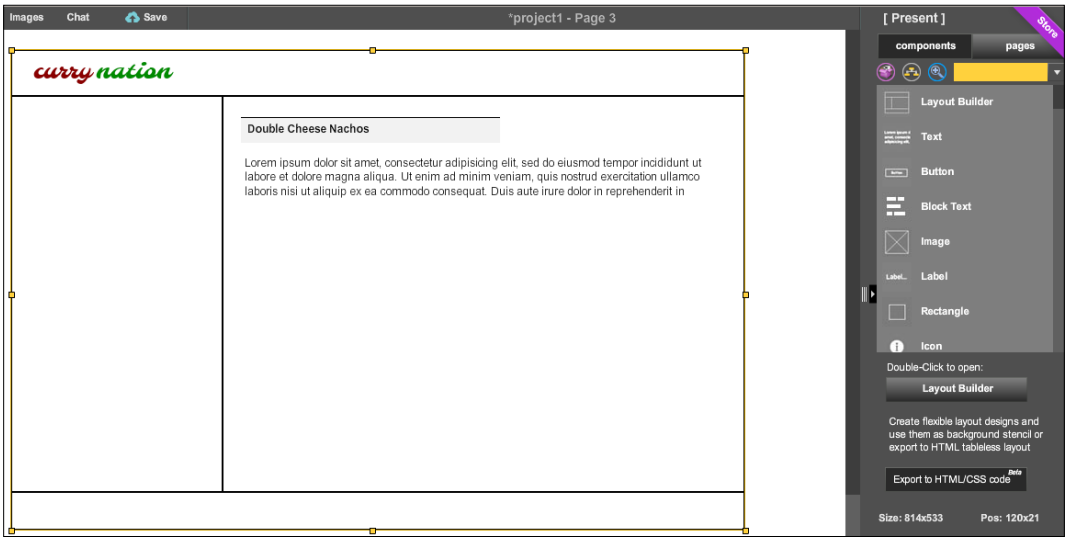
6. The header will contain the logo, login details, and dashboard links, whereas the footer will contain copyright information and company information links.



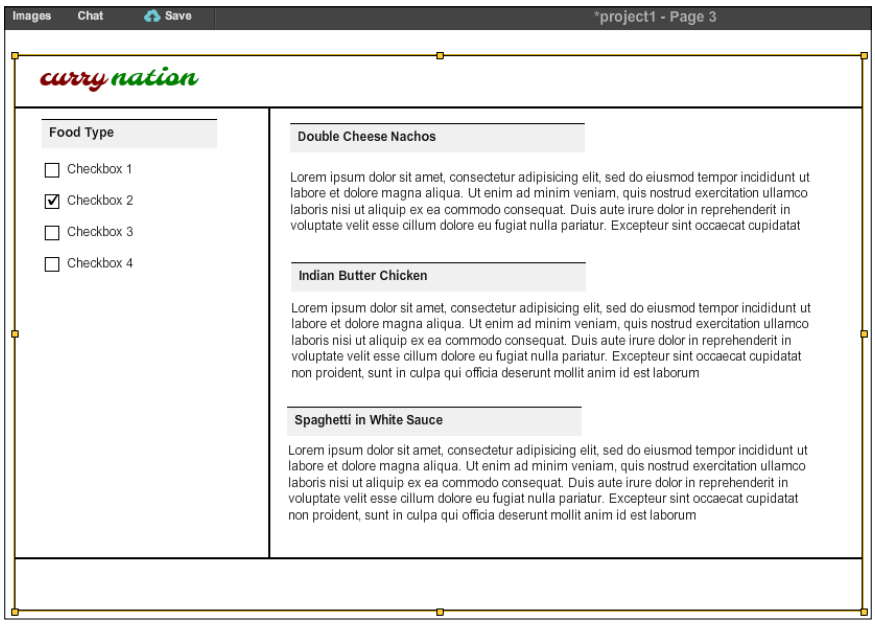
7. After resizing the layout builder, we will add the logo and images to the header. In order to do so, we will first drag-and-drop the **Image** component from the **components** panel and double-click on it. We will be presented with a modal box to manage and upload images. Browse and upload images using this tool. Once an image is selected, just drag and move it to the position where you want to see the logo placed.



8. The next step would logically be to build the inner page. This page will have some text on it. We will drag the title and text from the **components** bar and drop it to the central part of the layout.



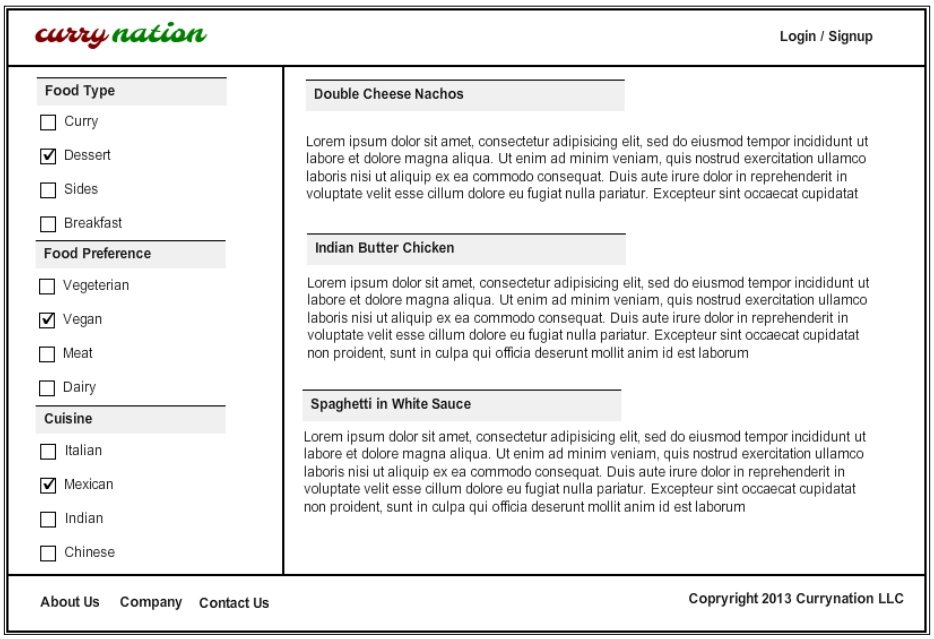
9. Add checkboxes and the remaining elements to the mockup.



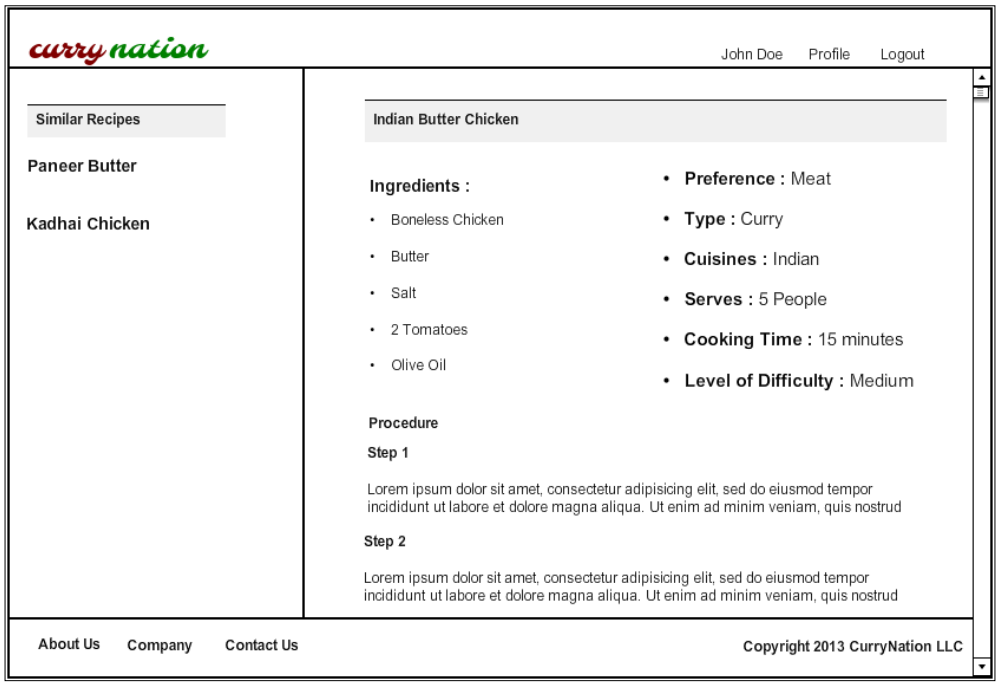
10. We will finally add some checkboxes to the left bar for filters. This includes food type, food preferences, and cuisines in order to properly categorize our recipes.
11. We can now figure out other elements of the page, for example, in order to create links such as **Login/Signup**, and **About Us**, we can use the **Label** component from the **components** panel.

Objective complete – mini debriefing

As seen in the previous steps, we added various page elements, including text areas, a title, and checkboxes to our page. We can use these page elements to create mockups for all the pages. Mockups for the home page and recipe page are shown in the following two screenshots:



The home page now looks complete with different links and information in the footer shown as follows:



Classified intel

The options offered in MockFlow include building mockups for the following:

- Web applications
- Mobile applications
- Themes specific to a particular CMS, or using a particular CSS framework such as Bootstrap
- Simple Wireframing from scratch or from templates

Adding test data and creating tests

Rails does a lot of work for us by providing us with generators, right from a blank application to different parts of the application. The trick lies in using it only when required. Our first application will consider a very simple use case of generators, but we will scarcely use them in subsequent projects. In this task, we will generate our application and write tests before we write the code.

Prepare for lift off

As MySQL and PostgreSQL are the most common RDBMS around, we're going to use either of them for building most of our applications. The default database in the development mode with Rails is SQLite. Make sure you have one of these databases working on your system and also make sure that the connection with Rails is working. We will use MySQL for most of our projects including this one.

Engage thrusters

The steps for creating a new application and setting up the **database** (**db**) are as follows:

1. Let us first create a blank application with a MySQL database as the default database using the following command:

   ```
   :~/$ rails new curry-nation -d mysql
   ```

2. Now we can go ahead and set up the application's `database.yml` file under `config` to connect to the system's database. You would need to make this file suit the database that you are using. We are using MySQL; likewise, you can edit the file for the database of your choice.

   ```
   config/database.yml
   development
     adapter: mysql2
     encoding: utf8
     database: curry-nation_development
     pool: 5
     username: root
     password:
     socket: /var/run/mysqld/mysqld.sock
   test:
     adapter: mysql2
     encoding: utf8
     database: curry-nation_test
     pool: 5
     username: root
     password:
     socket: /var/run/mysqld/mysqld.sock

   production:
     adapter: mysql2
     encoding: utf8
     database: curry-nation_production
     pool: 5
     username: root
     password:
     socket: /var/run/mysqld/mysqld.sock
   ```

3. Once the database is set up, we need to create the database using the following commands:

```
:~/curry-nation$ rake db:create
:~/curry-nation$ rake db:migrate
```

4. We will first prepare our fixtures. Fixtures contain test data that loads into the test database. These are placed in the `fixtures` folder under `test` with the filename `recipes.yml`:

```
test/fixtures/recipes.yml
curry:
  title: Curry
  food_preference_id: 1
  food_type: 1
  cuisine_id: 1
  servings: 1
  cooking_time: 1
  level_of_difficulty: Easy
  ingredients: Onions Tomatoes Salt Oil
  procedure: Heat Oil Chop Onions, tomatoes and
    Salt to it.
```

5. Once the fixtures are ready, we can populate the db with fixtures. However, we have not yet created the models and tables. Hence, we will load the fixtures' data once we create our models.

6. We can now go ahead and write integration tests. We will now add an integration test and create it line by line:

```
~/test/integration$ recipe_test.rb
```

7. We will load the test helper that will load the test database and other dependencies for the test:

```
require 'test_helper'
```

8. Load the test record and navigate to the new recipe page:

```
test/integration/recipe_test.rb
    curry = recipes(:curry)
    get "/recipes/new"
```

9. Post the data to the `new` method and assert for a success response. At this point, it even checks for validations if they are defined. Depending on this, it would be redirected to the index page:

```
test/integration/recipe_test.rb
    assert_response :success
    post_via_redirect "/recipes/new", title:
      recipes(:curry).title
```

10. We can now prepare the database and run the test:

```
:~/curry-nation/test/integration$ rake db:create RAILS_ENV="test"
(in /curry-nation)
r:~/curry-nation/test/integration$ rake test recipe_test.rb
  10 tests, 10 assertions, 10 failures, 0 success, 0 skips
```

11. The final integration test looks like this:

```
test/integration/recipe_test.rb
class RecipeFlowsTest < ActionDispatch::IntegrationTest
  fixtures :recipes
  test "create recipes" do
    https!
    curry = recipes(:curry)
    get "/recipes/new"
    assert_response :success
    post_via_redirect "/recipes/new", title:
      recipes(:curry).title
    assert_equal '/recipes', path
    assert_equal 'Create Recipe', flash[:notice]
    https!(false)
    get "/recipes"
    assert_response :success
    assert assigns(:recipes)
  end
end
```

12. Our integration tests look at the way the pages and routes work with each other. Controller tests look at how data is passed between these calls, and the methods call themselves.

13. Set up a `recipe` variable and get the `index` method:

```
test/integration/recipe_test.rb
class RecipesControllerTest < ActionController::TestCase
  setup do
    @recipe = recipes(:one)
  end
  test "should get index" do
    get :index
    assert_response :success
    assert_not_nil assigns(:recipes)
  end
```

14. We will use `assert` to get a new page in our controller test:

```
test/controllers/recipes_controller_test.rb
  test "should get new" do
    get :new
    assert_response :success
  end
```

15. We will also perform a test for creating a recipe:

```
test/controllers/recipes_controller_test.rb
  test "should create recipe" do
    assert_difference('Recipe.count') do
      post :create, recipe: { cooking_time:
        @recipe.cooking_time, cuisine_id:
        @recipe.cuisine_id, food_preference_id:
        @recipe.food_preference_id, food_type:
        @recipe.food_type, ingredients:
        @recipe.ingredients, level_of_difficulty:
        @recipe.level_of_difficulty, procedure:
        @recipe.procedure, servings: @recipe.servings,
        title: @recipe.title }
    end

    assert_redirected_to recipe_path(assigns(:recipe))
  end
```

16. Add a test for showing a recipe using the following code:

```
test/controllers/recipes_controller_test.rb
  test "should show recipe" do
    get :show, id: @recipe
    assert_response :success
  end
```

17. We will test the `edit` and `update` methods:

```
test/controllers/recipes_controller_test.rb
  test "should get edit" do
    get :edit, id: @recipe
    assert_response :success
  end

  test "should update recipe" do
    patch :update, id: @recipe, recipe: { cooking_time:
      @recipe.cooking_time, cuisine_id:
      @recipe.cuisine_id, food_preference_id:
      @recipe.food_preference_id, food_type:
      @recipe.food_type, ingredients:
      @recipe.ingredients, level_of_difficulty:
      @recipe.level_of_difficulty, procedure:
      @recipe.procedure, servings: @recipe.servings,
      title: @recipe.title }
    assert_redirected_to recipe_path(assigns(:recipe))
  end
```

18. Lastly, we will check for deletions:

```
test/controllers/recipes_controller_test.rb
  test "should destroy recipe" do
    assert_difference('Recipe.count', -1) do
      delete :destroy, id: @recipe
    end

    assert_redirected_to recipes_path
  end
end
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Objective complete – mini debriefing

As we saw in the previous task, the structure of the default testing framework in Rails 4 includes the respective style folder structure, which is much cleaner and nicely abstracted compared to the earlier versions. This is how it looks:

```
:~/curry-nation/test$ ls
controllers  fixtures  helpers  integration  mailers  models
  test_helper.rb
```

The test folder is self-descriptive in terms of the folder structure and clearly denotes which test belongs to which part of the system.

Here, we have prepared the test data and written tests that match the specifications. This will help us emulate our functionality. We are now ready to write some code in order to run our tests. The tests in this case failed because there is no code for the tests to run.

Classified intel

Testing is the backbone of your application. If you don't write tests, you are opening a Pandora's box for yourself.

Adding categories

To make the content of the website easily browsable, it makes sense to categorize it in different ways according to the diversity of choice a user has regarding food recipes. In this task, we will build navigation bars that would be visible on the left-hand side. Actually, it goes much deeper than just being the navigation bar. This is because it has to be built in a way that allows us to effectively search for data in future. So, for us, categories are a way to arrange data and make it more accessible, and in this task, we will see how to create categories.

Categories in our application are divided into three parts:

- **Food preferences**: Food preferences include the value system of users. They might like dairy free, vegan, vegetarian, meat, and so on. Recipes are categorized on the basis of this.
- **Food types**: Food types denote whether the food is a main course, a curry, a side dish, or a dessert.
- **Cuisines**: The final categorization is on the basis of cuisine.

Engage thrusters

The steps for adding categories are as follows:

1. We first need to create models that can be associated with the recipes:

```
:~/curry-nation$ rails g model food_type name:string
      invoke  active_record
      create db/migrate/20130803103254
      _create_food_types.rb
      create    app/models/food_type.rb
      invoke    test_unit
      create      test/models/food_type_test.rb
      create      test/fixtures/food_types.yml
```

2. We can't leave the categories blank, and they need some default data. We do not have an interface to load categories so we will use the seeds' data by adding default data using seed scripts.
3. This generates a food type model, fixtures, blank tests, and table migrations. These values have to be available in the database in order to be used with the recipes. We will load them using `seeds.rb`.

```
db/seeds.rb
food_types = ["Curry", "Dessert", "Sides","Breakfast"]
food_types.each{|d| FoodType.where(:name => d).create}
```

Once done, we'll run the following code:

```
rake db:migrate
rake db:seed
```

The following steps will help us to modify seeds:

1. The default seeds, if simply defined, can create duplicate records in the database and might fail validations. This is because every time we run `rake db:seeds`, it runs all the queries again. In order to avoid this, we can add `first_or_create` after the data, which checks for the record in the database before adding it to the database:

   ```
   db/seeds.rb
   food_types.each{|d| FoodType.where(:name => d).first_or_create}
   ```

2. Likewise, we can create other models related to categories in the same way:

   ```
   :~/curry-nation$ rails g model food_preference
     name:string
       invoke  active_record
       create
         db/migrate/20130803110704_create
         _food_preferences.rb
       create    app/models/food_preference.rb
       invoke    test_unit
       create      test/models/food_preference_test.rb
       create      test/fixtures/food_preferences.yml
   :~/curry-nation$ rake db:migrate
   ==  CreateFoodPreferences: migrating ============================
   =============
   -- create_table(:food_preferences)
      -> 0.1313s
   ==  CreateFoodPreferences: migrated (0.1315s) ===================
   =============

   :~/curry-nation$ rails g model cuisine name:string
       invoke  active_record
       create
         db/migrate/20130803111845_create_cuisines.rb
       create    app/models/cuisine.rb
       invoke    test_unit
       create      test/models/cuisine_test.rb
       create      test/fixtures/cuisines.yml
   :~/curry-nation$ rake db:migrate
   ==  CreateCuisines: migrating ===================================
   ======
   -- create_table(:cuisines)
      -> 0.1107s
   ==  CreateCuisines: migrated (0.1109s) ==========================
   ========
   ```

3. Load them into the database as follows:

```
db/seeds.rb
food_preferences = ["Vegetarian", "Vegan",
  "Meat","Dairy"]
food_preferences.each{|d| FoodPreference.where(:name =>
  d).first_or_create}

cuisines = ["Italian", "Mexican", "Indian","Chinese"]
cuisines.each{|d| Cuisine.where(:name =>
  d).first_or_create}
:~/curry-nation$ rake db:seed
```

4. For accessing the console and checking the entered data, we can load the Rails console and check whether all the values are present in the database or not:

```
:~/curry-nation$ rails c
Loading development environment (Rails 4.0.0)
1.9.3-p327 :002 > FoodType.all
  FoodType Load (0.9ms)  SELECT `food_types`.* FROM
    `food_types`
 => #<ActiveRecord::Relation [#<FoodType id: 1, name:
    "Curry", created_at: "2013-08-03 10:57:37",
    updated_at: "2013-08-03 10:57:37">, #<FoodType id: 2,
    name: "Dessert", created_at: "2013-08-03 10:57:37",
    updated_at: "2013-08-03 10:57:37">, #<FoodType id: 3,
    name: "Sides", created_at: "2013-08-03 10:57:37",
    updated_at: "2013-08-03 10:57:37">, #<FoodType id: 4,
    name: "Breakfast", created_at: "2013-08-03 10:57:37",
    updated_at: "2013-08-03 10:57:37">]>
1.9.3-p327 :003 > FoodPreference.all
  FoodPreference Load (0.7ms)  SELECT
    `food_preferences`.* FROM `food_preferences`
 => #<ActiveRecord::Relation [#<FoodPreference id: 1,
    name: "Vegetarian", created_at: "2013-08-03
    11:15:56", updated_at: "2013-08-03 11:15:56">,
    #<FoodPreference id: 2, name: "Vegan", created_at:
    "2013-08-03 11:15:56", updated_at: "2013-08-03
    11:15:56">, #<FoodPreference id: 3, name: "Meat",
    created_at: "2013-08-03 11:15:56", updated_at: "2013-
    08-03 11:15:56">, #<FoodPreference id: 4, name:
    "Dairy", created_at: "2013-08-03 11:15:56",
    updated_at: "2013-08-03 11:15:56">]>
1.9.3-p327 :004 > Cuisine.all
  Cuisine Load (0.6ms)  SELECT `cuisines`.* FROM
    `cuisines`
```