

Addison-Wesley Professional Ruby Series



RUBY ON RAILS™ TUTORIAL

SIXTH EDITION

Learn Web Development with Rails



MICHAEL HARTL

Foreword by DEREK SIVERS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Praise for Michael Hartl's Books and Videos on Ruby on Rails

“My former company (CD Baby) was one of the first to loudly switch to Ruby on Rails, and then even more loudly switch back to PHP. (Google me to read about the drama.) This book by Michael Hartl came so highly recommended that I had to try it, and the *Ruby on Rails™ Tutorial* is what I used to switch back to Rails again.”

—From the Foreword by Derek Sivers (sivers.org)

Formerly: founder of CD Baby

Currently: founder of Thoughts Ltd.

“Michael Hartl’s Rails Tutorial book is the #1 (and only, in my opinion) place to start when it comes to books about learning Rails. . . . It’s an amazing piece of work and, unusually, walks you through building a Rails app from start to finish with testing. If you want to read just one book and feel like a Rails master by the end of it, pick the *Ruby on Rails™ Tutorial*.”

—Peter Cooper, editor, Ruby Inside

“Michael Hartl’s *Ruby on Rails™ Tutorial* seamlessly taught me about not only Ruby on Rails, but also the underlying Ruby language, HTML, CSS, a bit of JavaScript, and even some SQL—but most importantly it showed me how to build a web application (Twitter) in a short amount of time.”

—Mattan Griffel, co-founder & CEO of One Month

“Although I’m a Python/Django developer by trade, I can’t stress enough how much this book has helped me. As an undergraduate, completely detached from industry, this book showed me how to use version control, how to write tests, and, most importantly—despite the steep learning curve for setting up and getting stuff running—how the end result of perseverance is extremely gratifying. It made me fall in love with technology all over again. This is the book I direct all my friends to who want to start learning programming/building stuff. Thank you Michael!”

—Prakhar Srivastav, software engineer, Xcite.com, Kuwait

“It has to be the best-written book of its type I’ve ever seen, and I can’t recommend it enough.”

—Daniel Hollands, administrator of Birmingham.IO

“For those wanting to learn Ruby on Rails, Hartl’s *Ruby on Rails™ Tutorial* is (in my opinion) the best way to do it.”

—David Young, software developer and author at deepinthe.com

“This is a great tutorial for a lot of reasons, because aside from just teaching Rails, Hartl is also teaching good development practices.”

—Michael Denomy, full-stack web developer

“Without a doubt, the best way I learned Ruby on Rails was by building an actual working app. I used Michael Hartl’s *Ruby on Rails™ Tutorial*, which showed me how to get a very basic Twitter-like app up and running from scratch. I cannot recommend this tutorial enough; getting something up and going fast was key; it beats memorization by a mile.”

—James Fend, serial entrepreneur, JamesFend.com

“The book gives you the theory and practice, while the videos focus on showing you in person how it’s done. Highly recommended combo.”

—Antonio Cangiano, software engineer, IBM

“The author is clearly an expert at the Ruby language and the Rails framework, but more than that, he is a working software engineer who introduces best practices throughout the text.”

—Gregory Charles, principal software developer at Fairway Technologies

RUBY ON RAILS™ TUTORIAL

Sixth Edition

This page intentionally left blank

RUBY ON RAILS™ TUTORIAL

Learn Web Development with Rails

Sixth Edition

Michael Hartl

◆ Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the web: informit.com/aw

Library of Congress Control Number: 2020932223

Copyright © 2021 Michael Hartl

Cover image: Olga Altunina/Shutterstock

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions.

The source code in *Ruby on Rails™ Tutorial, Sixth Edition*, is released under the MIT License.

ISBN-13: 978-0-13-670265-8

ISBN-10: 0-13-670265-1

ScoutAutomatedPrintCode

Vice President and Publisher

Mark L. Taub

Executive Editor

Debra Williams Cauley

Associate Editor

Manjula Anaskar

Managing Producer

Sandra Schroeder

Sr. Content Producer

Julie B. Nahil

Project Editor

diacriTech

Copy Editor

Jill Hobbs

Indexer

diacriTech

Proofreader

diacriTech

Cover Designer

Chuti Prasertsith

Composer

diacriTech

Contents

Foreword	xvii
Acknowledgments	xix
About the Author	xxi

Chapter 1 From Zero to Deploy 1

1.1	Up and Running	4
1.1.1	Development Environment	6
1.1.2	Installing Rails	9
1.2	The First Application	13
1.2.1	Bundler	17
1.2.2	<code>rails server</code>	22
1.2.3	Model-View-Controller (MVC)	27
1.2.4	Hello, World!	28
1.3	Version Control with Git	33
1.3.1	Installation and Setup	34
1.3.2	What Good Does Git Do for You?	38
1.3.3	GitHub	39
1.3.4	Branch, Edit, Commit, Merge	41

- 1.4 Deploying 49
 - 1.4.1 Heroku Setup and Deployment 49
 - 1.4.2 Heroku Commands 54
- 1.5 Conclusion 55
 - 1.5.1 What We Learned in this Chapter 56
- 1.6 Conventions Used in this Book 56

Chapter 2 A Toy App 59

- 2.1 Planning the Application 60
 - 2.1.1 A Toy Model for Users 64
 - 2.1.2 A Toy Model for Microposts 64
- 2.2 The Users Resource 65
 - 2.2.1 A User Tour 68
 - 2.2.2 MVC in Action 73
 - 2.2.3 Weaknesses of this Users Resource 80
- 2.3 The Microposts Resource 80
 - 2.3.1 A Micropost Microtour 81
 - 2.3.2 Putting the *Micro* in Microposts 86
 - 2.3.3 A User `has_many` Microposts 88
 - 2.3.4 Inheritance Hierarchies 91
 - 2.3.5 Deploying the Toy App 95
- 2.4 Conclusion 98
 - 2.4.1 What We Learned in this Chapter 99

Chapter 3 Mostly Static Pages 101

- 3.1 Sample App Setup 101
- 3.2 Static Pages 108
 - 3.2.1 Generated Static Pages 109
 - 3.2.2 Custom Static Pages 118
- 3.3 Getting Started with Testing 118
 - 3.3.1 Our First Test 121
 - 3.3.2 Red 123
 - 3.3.3 Green 125
 - 3.3.4 Refactor 128

- 3.4 Slightly Dynamic Pages 128
 - 3.4.1 Testing Titles (Red) 129
 - 3.4.2 Adding Page Titles (Green) 131
 - 3.4.3 Layouts and Embedded Ruby (Refactor) 135
 - 3.4.4 Setting the Root Route 142
- 3.5 Conclusion 145
 - 3.5.1 What We Learned in this Chapter 146
- 3.6 Advanced Testing Setup 146
 - 3.6.1 Minitest Reporters 147
 - 3.6.2 Automated Tests with Guard 148

Chapter 4 Rails-Flavored Ruby 153

- 4.1 Motivation 153
 - 4.1.1 Built-in Helpers 154
 - 4.1.2 Custom Helpers 155
- 4.2 Strings and Methods 159
 - 4.2.1 Strings 160
 - 4.2.2 Objects and Message Passing 164
 - 4.2.3 Method Definitions 167
 - 4.2.4 Back to the Title Helper 169
- 4.3 Other Data Structures 170
 - 4.3.1 Arrays and Ranges 170
 - 4.3.2 Blocks 175
 - 4.3.3 Hashes and Symbols 178
 - 4.3.4 CSS Revisited 183
- 4.4 Ruby Classes 185
 - 4.4.1 Constructors 185
 - 4.4.2 Class Inheritance 187
 - 4.4.3 Modifying Built-in Classes 190
 - 4.4.4 A Controller Class 192
 - 4.4.5 A User Class 195
- 4.5 Conclusion 198
 - 4.5.1 What We Learned in this Chapter 198

Chapter 5 Filling in the Layout 201

- 5.1 Adding Some Structure 201
 - 5.1.1 Site Navigation 202
 - 5.1.2 Bootstrap and Custom CSS 210
 - 5.1.3 Partials 220
- 5.2 Sass and the Asset Pipeline 225
 - 5.2.1 The Asset Pipeline 226
 - 5.2.2 Syntactically Awesome Stylesheets 228
- 5.3 Layout Links 235
 - 5.3.1 Contact Page 236
 - 5.3.2 Rails Routes 238
 - 5.3.3 Using Named Routes 242
 - 5.3.4 Layout Link Tests 244
- 5.4 User Signup: A First Step 248
 - 5.4.1 Users Controller 248
 - 5.4.2 Signup URL 250
- 5.5 Conclusion 253
 - 5.5.1 What We Learned in this Chapter 254

Chapter 6 Modeling Users 255

- 6.1 User Model 256
 - 6.1.1 Database Migrations 257
 - 6.1.2 The Model File 263
 - 6.1.3 Creating User Objects 264
 - 6.1.4 Finding User Objects 268
 - 6.1.5 Updating User Objects 270
- 6.2 User Validations 271
 - 6.2.1 A Validity Test 272
 - 6.2.2 Validating Presence 274
 - 6.2.3 Length Validation 278
 - 6.2.4 Format Validation 280
 - 6.2.5 Uniqueness Validation 286
- 6.3 Adding a Secure Password 295
 - 6.3.1 A Hashed Password 296
 - 6.3.2 User Has Secure Password 299

- 6.3.3 Minimum Password Standards 301
- 6.3.4 Creating and Authenticating a User 303
- 6.4 Conclusion 305
 - 6.4.1 What We Learned in this Chapter 306

Chapter 7 Sign Up 309

- 7.1 Showing Users 310
 - 7.1.1 Debug and Rails Environments 311
 - 7.1.2 A Users Resource 316
 - 7.1.3 Debugger 322
 - 7.1.4 A Gravatar Image and a Sidebar 324
- 7.2 Signup Form 331
 - 7.2.1 Using `form_with` 332
 - 7.2.2 Signup Form HTML 335
- 7.3 Unsuccessful Signups 339
 - 7.3.1 A Working Form 339
 - 7.3.2 Strong Parameters 343
 - 7.3.3 Signup Error Messages 346
 - 7.3.4 A Test for Invalid Submission 351
- 7.4 Successful Signups 355
 - 7.4.1 The Finished Signup Form 355
 - 7.4.2 The Flash 358
 - 7.4.3 The First Signup 361
 - 7.4.4 A Test for Valid Submission 364
- 7.5 Professional-Grade Deployment 367
 - 7.5.1 SSL in Production 368
 - 7.5.2 Production Webserver 369
 - 7.5.3 Production Database Configuration 370
 - 7.5.4 Production Deployment 371
- 7.6 Conclusion 373
 - 7.6.1 What We Learned in this Chapter 373

Chapter 8 Basic Login 375

- 8.1 Sessions 376
 - 8.1.1 Sessions Controller 376
 - 8.1.2 Login Form 380

- 8.1.3 Finding and Authenticating a User 383
- 8.1.4 Rendering with a Flash Message 388
- 8.1.5 A Flash Test 390
- 8.2 Logging In 393
 - 8.2.1 The `log_in` Method 394
 - 8.2.2 Current User 396
 - 8.2.3 Changing the Layout Links 401
 - 8.2.4 Testing Layout Changes 416
 - 8.2.5 Login Upon Signup 422
- 8.3 Logging Out 425
- 8.4 Conclusion 429
 - 8.4.1 What We Learned in this Chapter 429

Chapter 9 Advanced Login 431

- 9.1 Remember Me 431
 - 9.1.1 Remember Token and Digest 432
 - 9.1.2 Login with Remembering 439
 - 9.1.3 Forgetting Users 448
 - 9.1.4 Two Subtle Bugs 451
- 9.2 “Remember Me” Checkbox 456
- 9.3 Remember Tests 462
 - 9.3.1 Testing the “Remember Me” Checkbox 462
 - 9.3.2 Testing the Remember Branch 468
- 9.4 Conclusion 472
 - 9.4.1 What We Learned in this Chapter 472

Chapter 10 Updating, Showing, and Deleting Users 475

- 10.1 Updating Users 475
 - 10.1.1 Edit Form 476
 - 10.1.2 Unsuccessful Edits 483
 - 10.1.3 Testing Unsuccessful Edits 484
 - 10.1.4 Successful Edits (with TDD) 486
- 10.2 Authorization 491
 - 10.2.1 Requiring Logged-in Users 491
 - 10.2.2 Requiring the Right User 497
 - 10.2.3 Friendly Forwarding 502

- 10.3 Showing All Users 507
 - 10.3.1 Users Index 507
 - 10.3.2 Sample Users 513
 - 10.3.3 Pagination 516
 - 10.3.4 Users Index Test 520
 - 10.3.5 Partial Refactoring 523
- 10.4 Deleting Users 525
 - 10.4.1 Administrative Users 526
 - 10.4.2 The `destroy` Action 530
 - 10.4.3 User Destroy Tests 533
- 10.5 Conclusion 536
 - 10.5.1 What We Learned in this Chapter 537

Chapter 11 Account Activation 539

- 11.1 Account Activations Resource 541
 - 11.1.1 Account Activations Controller 541
 - 11.1.2 Account Activation Data Model 542
- 11.2 Account Activation Emails 549
 - 11.2.1 Mailer Templates 549
 - 11.2.2 Email Previews 554
 - 11.2.3 Email Tests 558
 - 11.2.4 Updating the Users `create` Action 561
- 11.3 Activating the Account 565
 - 11.3.1 Generalizing the `authenticated?` Method 565
 - 11.3.2 Activation `edit` Action 571
 - 11.3.3 Activation Test and Refactoring 574
- 11.4 Email in Production 581
- 11.5 Conclusion 584
 - 11.5.1 What We Learned in this Chapter 584

Chapter 12 Password Reset 587

- 12.1 Password Resets Resource 590
 - 12.1.1 Password Resets Controller 590
 - 12.1.2 New Password Resets 593
 - 12.1.3 Password Reset `create` Action 596

- 12.2 Password Reset Emails 599
 - 12.2.1 Password Reset Mailer and Templates 600
 - 12.2.2 Email Tests 605
- 12.3 Resetting the Password 607
 - 12.3.1 Reset `edit` Action 607
 - 12.3.2 Updating the Reset 610
 - 12.3.3 Password Reset Test 615
- 12.4 Email in Production (Take Two) 621
- 12.5 Conclusion 624
 - 12.5.1 What We Learned in this Chapter 625
- 12.6 Proof of Expiration Comparison 625

Chapter 13 User Microposts 627

- 13.1 A Micropost Model 627
 - 13.1.1 The Basic Model 628
 - 13.1.2 Micropost Validations 631
 - 13.1.3 User/Micropost Associations 634
 - 13.1.4 Micropost Refinements 638
- 13.2 Showing Microposts 643
 - 13.2.1 Rendering Microposts 644
 - 13.2.2 Sample Microposts 649
 - 13.2.3 Profile Micropost Tests 654
- 13.3 Manipulating Microposts 657
 - 13.3.1 Micropost Access Control 658
 - 13.3.2 Creating Microposts 662
 - 13.3.3 A Proto-Feed 670
 - 13.3.4 Destroying Microposts 680
 - 13.3.5 Micropost Tests 684
- 13.4 Micropost Images 688
 - 13.4.1 Basic Image Upload 689
 - 13.4.2 Image Validation 696
 - 13.4.3 Image Resizing 701
 - 13.4.4 Image Upload in Production 705
- 13.5 Conclusion 714
 - 13.5.1 What We Learned in this Chapter 716

Chapter 14 Following Users 717

- 14.1 The Relationship Model 718
 - 14.1.1 A Problem with the Data Model (and a Solution) 719
 - 14.1.2 User/Relationship Associations 725
 - 14.1.3 Relationship Validations 728
 - 14.1.4 Followed Users 730
 - 14.1.5 Followers 733
 - 14.2 A Web Interface for Following Users 736
 - 14.2.1 Sample Following Data 736
 - 14.2.2 Stats and a Follow Form 738
 - 14.2.3 Following and Followers Pages 748
 - 14.2.4 A Working Follow Button the Standard Way 757
 - 14.2.5 A Working Follow Button with Ajax 759
 - 14.2.6 Following Tests 765
 - 14.3 The Status Feed 768
 - 14.3.1 Motivation and Strategy 768
 - 14.3.2 A First Feed Implementation 771
 - 14.3.3 Subselects 774
 - 14.4 Conclusion 780
 - 14.4.1 Guide to Further Resources 780
 - 14.4.2 What We Learned in this Chapter 781
- Index 783

This page intentionally left blank

Foreword

My former company (CD Baby) was one of the first to loudly switch to Ruby on Rails, and then even more loudly switch back to PHP (Google me to read about the drama). This book by Michael Hartl came so highly recommended that I had to try it, and the *Ruby on Rails Tutorial* is what I used to switch back to Rails again.

Though I've worked my way through many Rails books, this is the one that finally made me “get” it. Everything is done very much “the Rails way”—a way that felt very unnatural to me initially, but now after doing this book finally feels natural. This is also the only Rails book that does test-driven development the entire time, an approach that is highly recommended by the experts but that has never been so clearly demonstrated before. Finally, by including Git, GitHub, and Heroku in the demo examples, the author really gives you a feel for what it's like to do a real-world project. The tutorial's code examples are not in isolation.

The linear narrative is such a great format. Personally, I powered through the Rails Tutorial in three long days doing all the examples and challenges at the end of each chapter. [This is not typical! Most readers take much longer to finish the tutorial. —Michael] Do it from start to finish, without jumping around, and you'll get the ultimate benefit.

Enjoy!

—Derek Sivers (sivers.org)
Founder, CD Baby

This page intentionally left blank

Acknowledgments

The *Ruby on Rails Tutorial* owes a lot to my previous Rails book, *RailsSpace*, and hence to my coauthor Aurelius Prochazka. I'd like to thank Aure both for the work he did on that book and for his support of this one. I'd also like to thank Debra Williams Cauley, my editor on both *RailsSpace* and the *Ruby on Rails Tutorial*; as long as she keeps taking me to baseball games, I'll keep writing books for her.

I'd like to acknowledge a long list of Rubyists who have taught and inspired me over the years: David Heinemeier Hansson, Yehuda Katz, Carl Lerche, Jeremy Kemper, Xavier Noria, Ryan Bates, Geoffrey Grosenbach, Peter Cooper, Matt Aimonetti, Mark Bates, Gregg Pollack, Wayne E. Seguin, Amy Hoy, Dave Chelimsky, Pat Maddox, Tom Preston-Werner, Chris Wanstrath, Chad Fowler, Josh Sussner, Obie Fernandez, Ian McFarland, Steph Bristol, Pratik Naik, Sarah Mei, Sarah Allen, Wolfram Arnold, Alex Chaffee, Giles Bowkett, Evan Dorn, Long Nguyen, James Lindenbaum, Adam Wiggins, Tikhon Bernstam, Ron Evans, Wyatt Greene, Miles Forrest, Sandi Metz, Ryan Davis, Aaron Patterson, Aja Hammerly, Richard "Schneems" Schneeman, the good people at Pivotal Labs, the Heroku gang, the thoughtbot folks, and the GitHub crew.

I'd like to thank technical reviewer Andrew Thai for his careful reading of the original manuscript and for his helpful suggestions. I'd also like to thank my cofounders

at Learn Enough, Nick Merwin and Lee Donahoe, for all their help in preparing this tutorial.

Finally, many, many readers—far too many to list—have contributed a huge number of bug reports and suggestions during the writing of this book, and I gratefully acknowledge their help in making it as good as it can be.

About the Author

Michael Hartl is the creator of the *Ruby on Rails™ Tutorial*, one of the leading introductions to web development, and is cofounder and principal author at LearnEnough.com. Previously, he was a physics instructor at the California Institute of Technology (Caltech), where he received a Lifetime Achievement Award for Excellence in Teaching. He is a graduate of Harvard College, has a PhD in Physics from Caltech, and is an alumnus of the Y Combinator entrepreneur program.

This page intentionally left blank

CHAPTER 2

A Toy App

In this chapter, we develop a toy demo application to show off some of the power of Rails. The purpose is to get a high-level overview of Ruby on Rails programming (and web development in general) by rapidly generating an application using *scaffold generators*, which create a large amount of functionality automatically. As discussed in Box 2.1, the rest of the book will take the opposite approach, developing a full sample application incrementally and explaining each new concept as it arises, but for a quick overview (and some instant gratification) there is no substitute for scaffolding. The resulting toy app will enable us to interact with it through its URLs, giving us insight into the structure of a Rails application, including a first example of the *REST architecture* favored by Rails.

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

Box 2.1: Scaffolding: Quicker, Easier, More Seductive

From the beginning, Rails has benefited from a palpable sense of excitement, starting with the famous 15-minute weblog video (youtu.be/Gzj723LkRJY) by Rails creator David Heinemeier Hansson. That video and its successors are a great way to get a taste of Rails' power, and I recommend watching them. But be warned: They accomplish their amazing 15-minute feat using a feature called *scaffolding*, which relies heavily on *generated code*, magically created by the Rails `generate scaffold` command.

When writing a Ruby on Rails tutorial, it is tempting to rely on the scaffolding approach—it's quicker, easier, more seductive. But the complexity and sheer amount of code in the scaffolding can be utterly overwhelming to a beginning Rails developer; you may be able to use it, but you probably won't understand it. Following the scaffolding approach risks turning you into a virtuoso script generator with little (and brittle) actual knowledge of Rails.

In the *Ruby on Rails Tutorial*, we'll take the (nearly) polar opposite approach: Although this chapter will develop a small toy app using scaffolding, the core of the Rails Tutorial is the sample app, which we'll start writing in Chapter 3. At each stage of developing the sample application, we will write *small, bite-sized* pieces of code—simple enough to understand, yet novel enough to be challenging. The cumulative effect will be a deeper, more flexible knowledge of Rails, giving you a good background for writing nearly any type of web application.

2.1 Planning the Application

In this section, we'll outline our plans for the toy application. As in Section 1.2, we'll start by generating the application skeleton using the `rails new` command with a specific Rails version number:

```
$ cd ~/environment
$ rails _6.0.2.1_ new toy_app
$ cd toy_app/
```

If you're using the cloud IDE as recommended in Section 1.1.1, note that this second app can be created in the same environment as the first. It is not necessary to create a new environment. To get the files to appear, you may need to click the gear icon in the file navigator area and select "Refresh File Tree."

Next, we'll use a text editor to update the **Gemfile** needed by Bundler with the contents of Listing 2.1.

Important note: For all the Gemfiles in this book, you should use the version numbers listed at gemfiles-6th-ed.railstutorial.org instead of the ones listed below (although they should be identical if you are reading this online).

Listing 2.1: A **Gemfile** for the toy app.

```
source 'https://rubygems.org'
git_source(:github) { |repo| "https://github.com/#{repo}.git" }

gem 'rails',      '6.0.2.1'
gem 'puma',       '3.12.2'
gem 'sass-rails', '5.1.0'
gem 'webpacker',  '4.0.7'
gem 'turbolinks', '5.2.0'
gem 'jbuilder',   '2.9.1'
gem 'bootsnap',   '1.4.5', require: false

group :development, :test do
  gem 'sqlite3', '1.4.1'
  gem 'byebug',  '11.0.1', platforms: [:mri, :mingw, :x64_mingw]
end

group :development do
  gem 'web-console', '4.0.1'
  gem 'listen',      '3.1.5'
  gem 'spring',      '2.1.0'
  gem 'spring-watcher-listen', '2.0.1'
end

group :test do
  gem 'capybara', '3.28.0'
  gem 'selenium-webdriver', '3.142.4'
  gem 'webdrivers', '4.1.2'
end

group :production do
  gem 'pg', '1.1.4'
end

# Windows does not include zoneinfo files, so bundle the tzinfo-data gem
gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]
```

Note that Listing 2.1 is identical to Listing 1.18.

As in Section 1.4.1, we'll install the local gems while preventing the installation of production gems using the **--without production** option:

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

As noted in Section 1.2.1, you may need to run **bundle update** as well (Box 1.2).

Finally, we'll put the toy app under version control with Git:

```
$ git init
$ git add -A
$ git commit -m "Initialize repository"
```

You should also create a new repository at GitHub by following the same steps as in Section 1.3.3 (taking care to make it private as in Figure 2.1), and then push up to the remote repository:

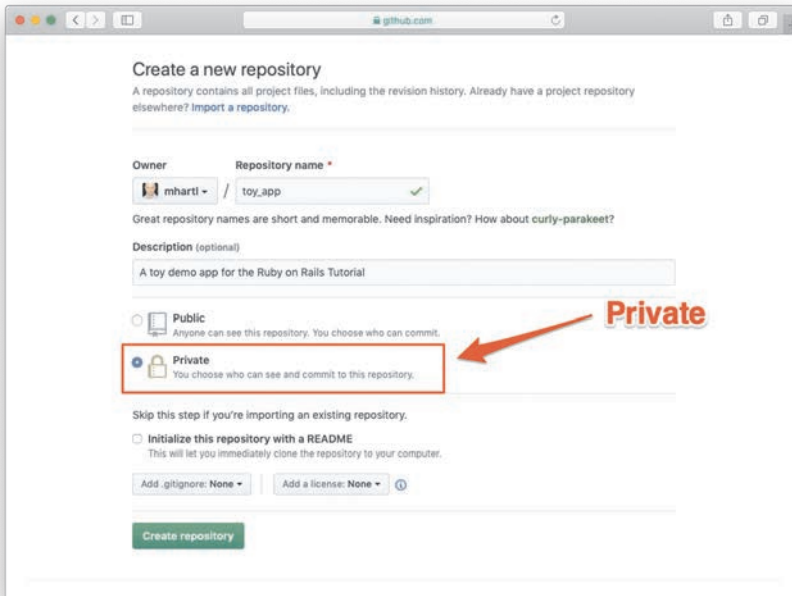


Figure 2.1: Creating the toy app repository at GitHub.

```
$ git remote add origin https://github.com/<username>/toy_app.git
$ git push -u origin master
```

Finally, it's never too early to deploy, which I suggest doing by following the same “hello, world!” steps from Section 1.2.4, as shown in Listing 2.2 and Listing 2.3.

Listing 2.2: Adding a **hello** action to the Application controller.

app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base

  def hello
    render html: "hello, world!"
  end
end
```

Listing 2.3: Setting the root route.

config/routes.rb

```
Rails.application.routes.draw do
  root 'application#hello'
end
```

Then commit the changes and push up to Heroku, and, at the same time, GitHub—it's a good idea to keep the two copies in sync:

```
$ git commit -am "Add hello"
$ heroku create
$ git push && git push heroku master
```

Here we've used the double ampersand operator **&&** (read “and”) to combine the pushes to GitHub and Heroku; the second command will execute only if the first one succeeds.¹

1. The **&&** operator is described in Chapter 4 of *Learn Enough Command Line to Be Dangerous* (www.learnenough.com/command-line).

As in Section 1.4, you may see some warning messages, which you should ignore for now. We'll deal with them in Section 7.5. Apart from the URL of the Heroku app, the result should be the same as in Figure 1.31.

2.1.1 A Toy Model for Users

Now we're ready to start making the app itself. The typical first step when making a web application is to create a *data model*, which is a representation of the structures needed by our application, including the relationships between them. In our case, the toy app will be a Twitter-style microblog, with only users and short (micro)posts. Thus, we'll begin with a model for *users* of the app in this section, and then we'll add a model for *microposts* (Section 2.1.2).

There are as many choices for a user data model as there are different registration forms on the web; for simplicity, we'll go with a distinctly minimalist approach. Users of our toy app will have a unique identifier called **id** (of type **integer**), a publicly viewable **name** (of type **string**), and an **email** address (also of type **string**) that will double as a unique username. (Note that there is no **password** attribute at this point, which is part of what makes this app a “toy.” We'll cover passwords starting in Chapter 6.) A summary of the data model for users appears in Figure 2.2.

As we'll see starting in Section 6.1.1, the label **users** in Figure 2.2 corresponds to a *table* in a database, and the **id**, **name**, and **email** attributes are *columns* in that table.

2.1.2 A Toy Model for Microposts

Recall from the introduction that a *micropost* is simply a short post, essentially a generic term for the brand-specific “tweet” (with the prefix “micro” motivated by Twitter's original description as a “micro-blog”). The core of the micropost data model is even

users	
id	integer
name	string
email	string

Figure 2.2: The data model for users.

simpler than the one for users: A micropost has only an **id** and a **content** field for the micropost's text (of type **text**).² There's an additional complication, though: We want to *associate* each micropost with a particular user. We'll accomplish this by recording the **user_id** of the owner of the post. The results are shown in Figure 2.3.

We'll see in Section 2.3.3 (and more fully in Chapter 13) how this **user_id** attribute allows us to succinctly express the notion that a user potentially has many associated microposts.

2.2 The Users Resource

In this section, we'll implement the users data model in Section 2.1.1, along with a web interface to that model. The combination will constitute a *Users resource*, which will allow us to think of users as objects that can be created, read, updated, and deleted through the web via the HTTP protocol. As promised in the introduction, our Users resource will be created by a scaffold generator program, which comes standard with each Rails project. I urge you not to look too closely at the generated code; at this stage, it will only serve to confuse you.

Rails scaffolding is generated by passing the **scaffold** command to the **rails generate** script. The argument of the **scaffold** command is the singular version of

microposts	
id	integer
content	text
user_id	integer

Figure 2.3: The data model for microposts.

2. Because microposts are short by design, the **string** type might actually be big enough to contain them, but using **text** better expresses our intent, while also giving us greater flexibility should we ever wish to relax the length constraint. Indeed, Twitter's change from allowing 140 to 280 characters in English-language tweets is a perfect example of why such flexibility is important: A **string** typically allows 255 ($2^8 - 1$) characters, which is big enough for 140-character tweets but not for 280-character ones. Using **text** allows a unified treatment of both cases.

the resource name (in this case, **User**), together with optional parameters for the data model's attributes:³

```
$ rails generate scaffold User name:string email:string

  invoke  active_record
  create  db/migrate/<timestamp>_create_users.rb
  create  app/models/user.rb
  invoke  test_unit
  create  test/models/user_test.rb
  create  test/fixtures/users.yml
  invoke  resource_route
  route   resources :users
  invoke  scaffold_controller
  create  app/controllers/users_controller.rb
  invoke  erb
  create  app/views/users
  create  app/views/users/index.html.erb
  create  app/views/users/edit.html.erb
  create  app/views/users/show.html.erb
  create  app/views/users/new.html.erb
  create  app/views/users/_form.html.erb
  invoke  test_unit
  create  test/controllers/users_controller_test.rb
  create  test/system/users_test.rb
  invoke  helper
  create  app/helpers/users_helper.rb
  invoke  test_unit
  invoke  jbuilder
  create  app/views/users/index.json.jbuilder
  create  app/views/users/show.json.jbuilder
  create  app/views/users/_user.json.jbuilder
  invoke  assets
  invoke  scss
  create  app/assets/stylesheets/users.scss
  invoke  scss
  create  app/assets/stylesheets/scaffolds.scss
```

By including **name:string** and **email:string**, we have arranged for the User model to have the form shown in Figure 2.2. (Note that there is no need to include a parameter for **id**; Rails creates it automatically for use as the *primary key* in the database.)

3. The name of the scaffold follows the convention of *models*, which are singular, rather than resources and controllers, which are plural. Thus, we have **User** instead of **Users**.

To proceed with the toy application, we first need to *migrate* the database using **rails db:migrate**, as shown in Listing 2.4.

Listing 2.4: Migrating the database.

```
$ rails db:migrate
== CreateUsers: migrating =====
-- create_table(:users)
   -> 0.0027s
== CreateUsers: migrated (0.0036s) =====
```

The effect of Listing 2.4 is to update the database with our new **users** data model. (We'll learn more about database migrations starting in Section 6.1.1.)

Having run the migration in Listing 2.4, we can run the local webserver in a separate tab (Figure 1.15). Users of the cloud IDE should first add the same configuration as in Section 1.2.2 to allow the toy app to be served (Listing 2.5).

Listing 2.5: Allowing connections to the local web server.

config/environments/development.rb

```
Rails.application.configure do
  .
  .
  .
  # Allow Cloud9 connections.
  config.hosts.clear
end
```

Then run the Rails server as in Section 1.2.2:

```
$ rails server
```

Now the toy application should be available on the local server as described in Section 1.2.2. In particular, if we visit the root URL at / (read “slash”, as noted in Section 1.2.4), we get the same “hello, world!” page shown in Figure 1.20.

2.2.1 A User Tour

In generating the Users resource scaffolding in Section 2.2, Rails created a large number of pages for manipulating users. For example, the page for listing all users is at `/users`, and the page for making a new user is at `/users/new`. The rest of this section is dedicated to taking a whirlwind tour through these user pages. As we proceed, it may help to refer to Table 2.1, which shows the correspondence between pages and URLs.

We start with the page that shows all the users in our application, called **index** and located at `/users`. As you might expect, initially there are no users at all (Figure 2.4).

Table 2.1: The correspondence between pages and URLs for the Users resource.

URL	Action	Purpose
<code>/users</code>	index	page to list all users
<code>/users/1</code>	show	page to show user with id 1
<code>/users/new</code>	new	page to make a new user
<code>/users/1/edit</code>	edit	page to edit user with id 1

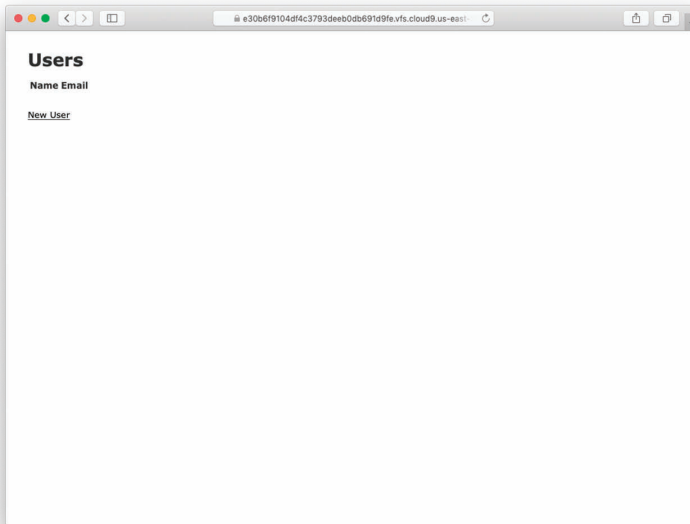


Figure 2.4: The initial index page for the Users resource (`/users`).

To make a new user, we can click on the New User link in Figure 2.4 to visit the **new** page at `/users/new`, as shown in Figure 2.5. In Chapter 7, this will become the user signup page.

We can create a user by entering name and email values in the text fields and then clicking the Create User button. The result is the user **show** page at `/users/1`, as seen in Figure 2.6. (The green welcome message is accomplished using the *flash*, which we'll learn about in Section 7.4.2.) Note that the URL is `/users/1`; as you might suspect, the number **1** is simply the user's **id** attribute from Figure 2.2. In Section 7.1, this page will become the user's profile page.

To change a user's information, we click the Edit link to visit the **edit** page at `/users/1/edit` (Figure 2.7). By modifying the user information and clicking the Update User button, we arrange to change the information for the user in the toy application (Figure 2.8). (As we'll see in detail starting in Chapter 6, this user data is stored in a database back end.) We'll add user edit/update functionality to the sample application in Section 10.1.

Now we'll create a second user by revisiting the **new** page at `/users/new` and submitting a second set of user information. The resulting user **index** is shown in

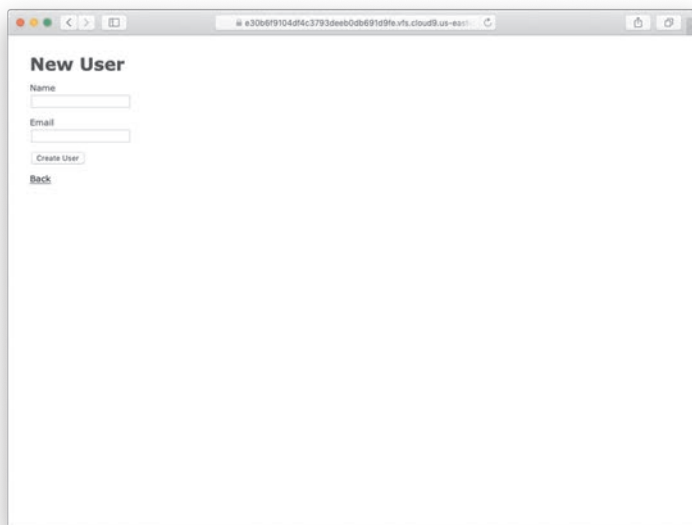


Figure 2.5: The new user page (`/users/new`).

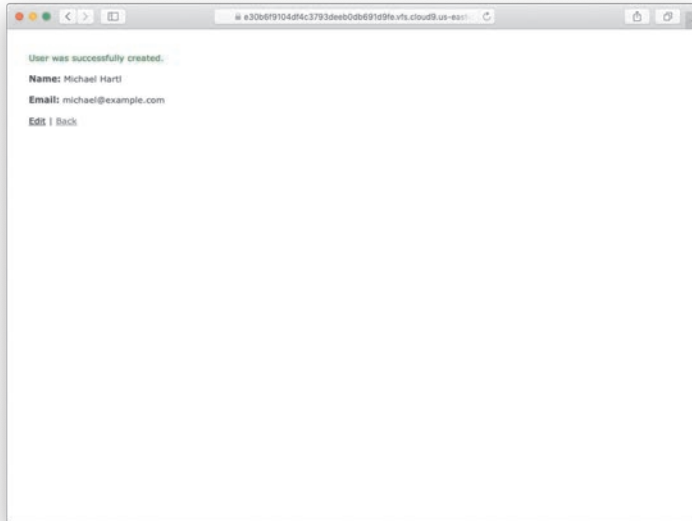


Figure 2.6: The page to show a user (/users/1).

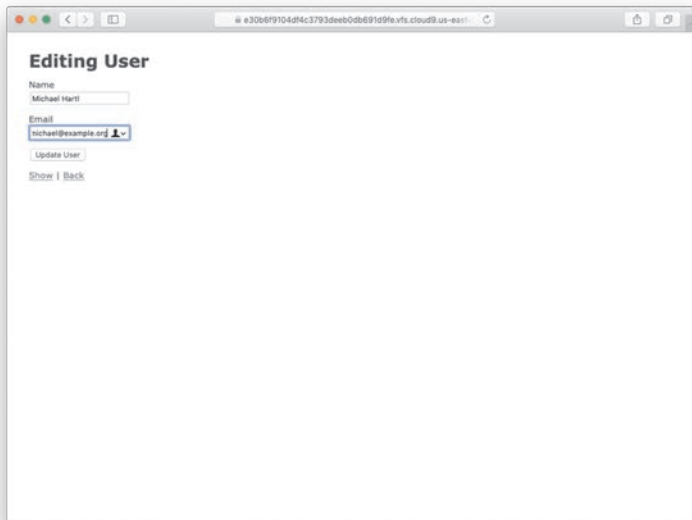


Figure 2.7: The user edit page (/users/1/edit).

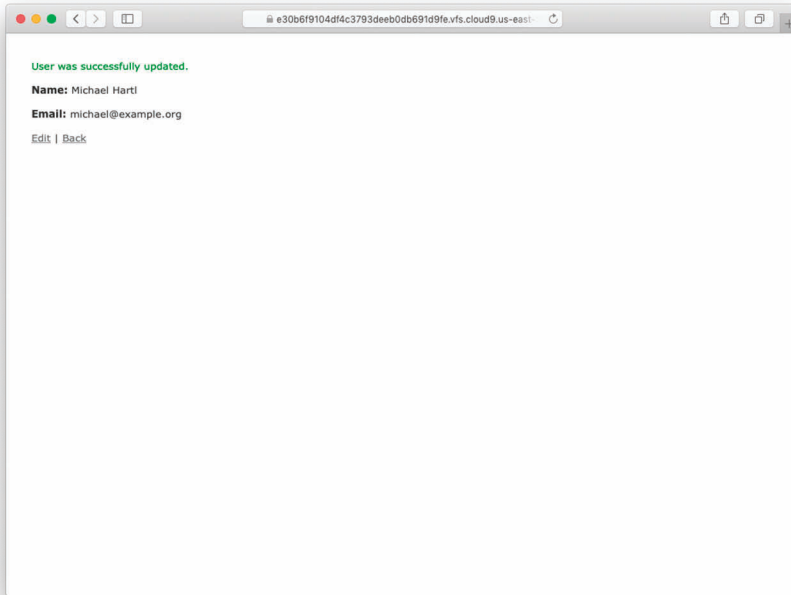


Figure 2.8: A user with updated information.

Figure 2.9. In Section 7.1, we will develop the user index into a more polished page for showing all users.

Having shown how to create, show, and edit users, we come finally to destroying them (Figure 2.10). You should verify that clicking on the link in Figure 2.10 destroys the second user, yielding an index page with only one user. (If it doesn't work, be sure that JavaScript is enabled in your browser; Rails uses JavaScript to issue the request needed to destroy a user.) Section 10.4 adds user deletion to the sample app, taking care to restrict its use to a special class of administrative users.

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers at <https://www.railstutorial.org/aw-solutions>.

To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

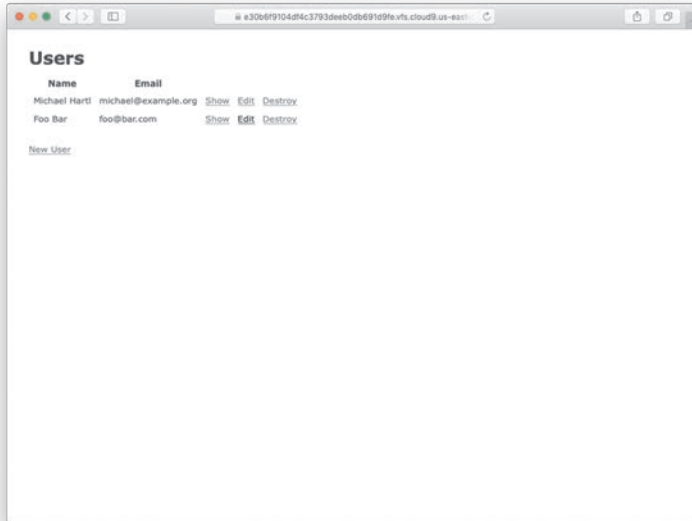


Figure 2.9: The user index page (/users) with a second user.

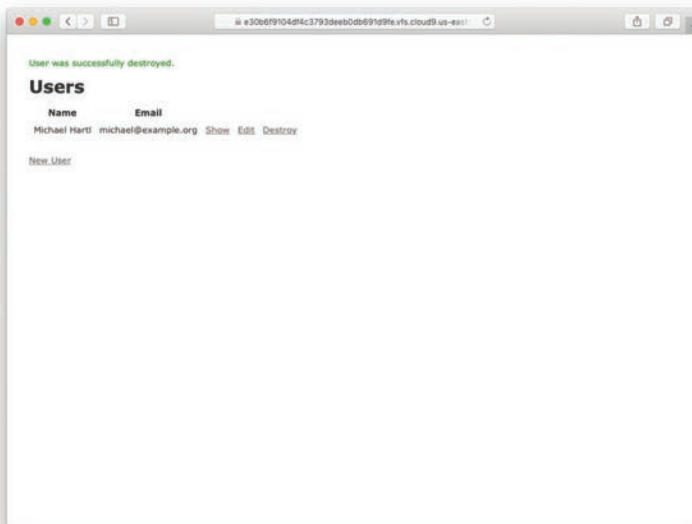


Figure 2.10: Destroying a user.

1. (For readers who know CSS) Create a new user, then use your browser’s HTML inspector to determine the CSS id for the text “User was successfully created.” What happens when you refresh your browser?
2. What happens if you try to create a user with a name but no email address?
3. What happens if you try create a user with an invalid email address, like “@example.com”?
4. Destroy each of the users created in the previous exercises. Does Rails display a message by default when a user is destroyed?

2.2.2 MVC in Action

Now that we’ve completed a quick overview of the Users resource, let’s examine one particular part of it in the context of the model–view–controller (MVC) pattern introduced in Section 1.2.3. Our strategy will be to describe the results of a typical browser hit—a visit to the user index page at /users—in terms of MVC (Figure 2.11).

Here is a summary of the steps shown in Figure 2.11:

1. The browser issues a request for the /users URL.
2. Rails routes /users to the **index** action in the Users controller.
3. The **index** action asks the User model to retrieve all users (**User.all**).
4. The User model pulls all the users from the database.
5. The User model returns the list of users to the controller.
6. The controller captures the users in the **@users** variable, which is passed to the **index** view.
7. The view uses embedded Ruby to render the page as HTML.
8. The controller passes the HTML back to the browser.⁴

Now let’s take a look at these steps in more detail. We start with a request issued from the browser—that is, the result of typing a URL in the address bar or clicking on a link (Step 1 in Figure 2.11). This request hits the *Rails router* (Step 2), which

4. Some references indicate that the view returns the HTML directly to the browser (via a webserver such as Apache or Nginx). Regardless of the implementation details, I find it helpful to think of the controller as a central hub through which all the application’s information flows.

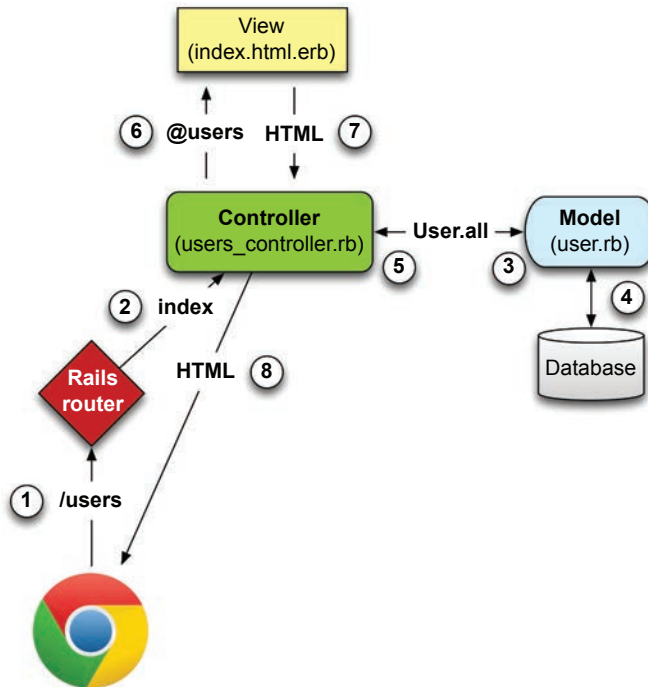


Figure 2.11: A detailed diagram of MVC in Rails.

dispatches the request to the proper *controller action* based on the URL (and, as we'll see in Box 3.2, the type of request). The code to create the mapping of user URLs to controller actions for the Users resource appears in Listing 2.6. This code effectively sets up the table of URL/action pairs seen in Table 2.1. (The strange notation `:users` is a *symbol*, which we'll learn about in Section 4.3.3.)

Listing 2.6: The Rails routes, with a rule for the Users resource.

`config/routes.rb`

```
Rails.application.routes.draw do
  resources :users
  root 'application#hello'
end
```

While we're looking at the routes file, let's take a moment to associate the root route with the users index, so that “slash” goes to /users. Recall from Listing 2.3 that we added the root route

```
root 'application#hello'
```

so that the root route went to the **hello** action in the Application controller. In the present case, we want to use the **index** action in the Users controller, which we can arrange using the code shown in Listing 2.7.

Listing 2.7: Adding a root route for users.

config/routes.rb

```
Rails.application.routes.draw do
  resources :users
  root 'users#index'
end
```

A *controller* contains a collection of related *actions*, and the pages from the tour in Section 2.2.1 correspond to actions in the Users controller. The controller generated by the scaffolding is shown schematically in Listing 2.8. Note the code **class UsersController < ApplicationController**, which is an example of a Ruby *class* with *inheritance*. (We'll discuss inheritance briefly in Section 2.3.4 and cover both subjects in more detail in Section 4.4.)

Listing 2.8: The Users controller in schematic form.

app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  .
  .
  def index
    .
    .
  end
end
```



```
def show
  .
  .
end

def new
  .
  .
end

def edit
  .
  .
end

def create
  .
  .
end

def update
  .
  .
end

def destroy
  .
  .
end
end
```

You might notice that there are more actions than there are pages. The **index**, **show**, **new**, and **edit** actions all correspond to pages from Section 2.2.1, but there are additional **create**, **update**, and **destroy** actions as well. These actions don't typically render pages (although they can); instead, their main purpose is to modify information about users in the database.

This full suite of controller actions, summarized in Table 2.2, represents the implementation of the REST architecture in Rails (Box 2.2), which is based on the idea of *representational state transfer*, a concept identified and named by computer scientist

Table 2.2: RESTful routes provided by the Users resource in Listing 2.6.

HTTP request	URL	Action	Purpose
GET	/users	index	page to list all users
GET	/users/1	show	page to show user with id 1
GET	/users/new	new	page to make a new user
POST	/users	create	create a new user
GET	/users/1/edit	edit	page to edit user with id 1
PATCH	/users/1	update	update user with id 1
DELETE	/users/1	destroy	delete user with id 1

Roy Fielding.⁵ Note from Table 2.2 that there is some overlap in the URLs; for example, both the user **show** action and the **update** action correspond to the URL /users/1. The difference between them is the HTTP request method they respond to. We'll learn more about HTTP request methods starting in Section 3.3.

Box 2.2: REpresentational State Transfer (REST)

If you read much about Ruby on Rails web development, you'll see a lot of references to "REST," which is an acronym for REpresentational State Transfer. REST is an architectural style for developing distributed, networked systems and software applications such as the World Wide Web and web applications. Although REST theory is rather abstract, in the context of Rails applications REST means that most application components (such as users and microposts) are modeled as *resources* that can be created, read, updated, and deleted—operations that correspond both to the CRUD operations of relational databases and to the four fundamental HTTP request methods: POST, GET, PATCH, and DELETE. (We'll learn more about HTTP requests in Section 3.3 and especially Box 3.2.)

As a Rails application developer, the RESTful style of development helps you make choices about which controllers and actions to write: You simply structure the application using resources that get created, read, updated, and deleted. In the case of users and microposts, this process is straightforward, since they are naturally resources in their own right. In Chapter 14, we'll see an example where

5. Fielding, Roy Thomas. *Architectural Styles and the Design of Network-Based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

REST principles allow us to model a subtler problem, “following users,” in a natural and convenient way.

To examine the relationship between the Users controller and the User model, let’s focus on the **index** action, shown in Listing 2.9. (Learning how to read code even when you don’t fully understand it is an important aspect of technical sophistication (Box 1.2).)

Listing 2.9: The simplified user **index** action for the toy application.

app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  .
  .
  def index
    @users = User.all
  end
  .
  .
  .
end
```

This **index** action includes the line **@users = User.all** (Step 3 in Figure 2.11), which asks the User model to retrieve a list of all the users from the database (Step 4), and then places them in the variable **@users** (pronounced “at-users”) (Step 5).

The User model itself appears in Listing 2.10. Although it is rather plain, it comes equipped with a large amount of functionality because of inheritance (Section 2.3.4 and Section 4.4). In particular, by using the Rails library called *Active Record*, the code in Listing 2.10 arranges for **User.all** to return all the users in the database.

Listing 2.10: The User model for the toy application.

app/models/user.rb

```
class User < ApplicationRecord
end
```

Once the `@users` variable is defined, the controller calls the *view* (Step 6), shown in Listing 2.11. Variables that start with the `@` sign, called *instance variables*, are automatically available in the views; in this case, the `index.html.erb` view in Listing 2.11 iterates through the `@users` list and outputs a line of HTML for each one. (Remember, you aren't supposed to understand this code right now. It is shown only for purposes of illustration.)

Listing 2.11: The view for the users index.

app/views/users/index.html.erb

```
<p id="notice"><%= notice %></p>

<h1>Users</h1>

<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Email</th>
      <th colspan="3"></th>
    </tr>
  </thead>

  <tbody>
    <% @users.each do |user| %>
      <tr>
        <td><%= user.name %></td>
        <td><%= user.email %></td>
        <td><%= link_to 'Show', user %></td>
        <td><%= link_to 'Edit', edit_user_path(user) %></td>
        <td><%= link_to 'Destroy', user, method: :delete,
          data: { confirm: 'Are you sure?' } %></td>
      </tr>
    <% end %>
  </tbody>
</table>

<br>

<%= link_to 'New User', new_user_path %>
```

The view converts its contents to HTML (Step 7), which is then returned by the controller to the browser for display (Step 8).

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers at <https://www.railstutorial.org/aw-solutions>.

To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. By referring to Figure 2.11, write out the analogous steps for visiting the URL `/users/1/edit`.
2. Find the line in the scaffolding code that retrieves the user from the database in the previous exercise. *Hint:* It's in a special location called `set_user`.
3. What is the name of the view file for the user edit page?

2.2.3 Weaknesses of this Users Resource

Though good for getting a general overview of Rails, the scaffold Users resource suffers from a number of severe weaknesses.

- **No data validations.** Our User model accepts data such as blank names and invalid email addresses without complaint.
- **No authentication.** We have no notion of logging in or out, and no way to prevent any user from performing any operation.
- **No tests.** This isn't technically true—the scaffolding includes rudimentary tests—but the generated tests don't test for data validation, authentication, or any other custom requirements.
- **No style or layout.** There is no consistent site styling or navigation.
- **No real understanding.** If you understand the scaffold code, you probably shouldn't be reading this book.

2.3 The Microposts Resource

Having generated and explored the Users resource, we turn now to the associated Microposts resource. Throughout this section, I recommend comparing the elements of the Microposts resource with the analogous user elements from Section 2.2; you should see that the two resources parallel each other in many ways. The RESTful structure of Rails applications is best absorbed by this sort of repetition of form. Indeed, seeing the parallel structure of Users and Microposts even at this early stage is one of the prime motivations for this chapter.

2.3.1 A Micropost Microtour

As with the Users resource, we'll generate scaffold code for the Microposts resource using **rails generate scaffold**, in this case implementing the data model from Figure 2.3:⁶

```
$ rails generate scaffold Micropost content:text user_id:integer

  invoke  active_record
  create  db/migrate/<timestamp>_create_microposts.rb
  create  app/models/micropost.rb
  invoke  test_unit
  create  test/models/micropost_test.rb
  create  test/fixtures/microposts.yml
  invoke  resource_route
   route  resources :microposts
  invoke  scaffold_controller
  create  app/controllers/microposts_controller.rb
  invoke  erb
  create  app/views/microposts
  create  app/views/microposts/index.html.erb
  create  app/views/microposts/edit.html.erb
  create  app/views/microposts/show.html.erb
  create  app/views/microposts/new.html.erb
  create  app/views/microposts/_form.html.erb
  invoke  test_unit
  create  test/controllers/microposts_controller_test.rb
  create  test/system/microposts_test.rb
  invoke  helper
  create  app/helpers/microposts_helper.rb
  invoke  test_unit
  invoke  jbuilder
  create  app/views/microposts/index.json.jbuilder
  create  app/views/microposts/show.json.jbuilder
  create  app/views/microposts/_micropost.json.jbuilder
  invoke  assets
  invoke  scss
  create  app/assets/stylesheets/microposts.scss
  invoke  scss
  identical  app/assets/stylesheets/scaffolds.scss
```

To update our database with the new data model, we need to run a migration as in Section 2.2:

6. As with the User scaffold, the scaffold generator for microposts follows the singular convention of Rails models; thus, we have **generate Micropost**.

```
$ rails db:migrate
== CreateMicroposts: migrating =====
-- create_table(:microposts)
   -> 0.0023s
== CreateMicroposts: migrated (0.0026s) =====
```

Now we are in a position to create microposts in the same way we created users in Section 2.2.1. As you might guess, the scaffold generator has updated the Rails routes file with a rule for Microposts resource, as seen in Listing 2.12.⁷ As with users, the **resources :microposts** routing rule maps micropost URLs to actions in the Microposts controller, as seen in Table 2.3.

Listing 2.12: The Rails routes, with a new rule for Microposts resources.

config/routes.rb

```
Rails.application.routes.draw do
  resources :microposts
  resources :users
  root 'users#index'
end
```

Table 2.3: RESTful routes provided by the Microposts resource in Listing 2.12.

HTTP request	URL	Action	Purpose
GET	/microposts	index	page to list all microposts
GET	/microposts/1	show	page to show micropost with id 1
GET	/microposts/new	new	page to make a new micropost
POST	/microposts	create	create a new micropost
GET	/microposts/1/edit	edit	page to edit micropost with id 1
PATCH	/microposts/1	update	update micropost with id 1
DELETE	/microposts/1	destroy	delete micropost with id 1

7. The scaffold code may have extra blank lines compared to Listing 2.12. This is not a cause for concern, as Ruby ignores such extra space.

The Microposts controller itself appears in schematic form in Listing 2.13. Note that, apart from having **MicropostsController** in place of **UsersController**, Listing 2.13 is *identical* to the code in Listing 2.8. This is a reflection of the REST architecture that is common to both resources.

Listing 2.13: The Microposts controller in schematic form.

app/controllers/microposts_controller.rb

```
class MicropostsController < ApplicationController
  .
  .
  .
  def index
    .
    .
    .
  end

  def show
    .
    .
    .
  end

  def new
    .
    .
    .
  end

  def edit
    .
    .
    .
  end

  def create
    .
    .
    .
  end

  def update
    .
    .
    .
  end
end
```



```
def destroy
  .
  .
end
end
```

To make some actual microposts, we click on New Micropost on the micropost index page (Figure 2.12). We then enter information at the new microposts page, `/microposts/new`, as seen in Figure 2.13.

At this point, go ahead and create a micropost or two, taking care to make sure that at least one has a `user_id` of `1` to match the id of the first user created in Section 2.2.1. The result should look something like Figure 2.14.

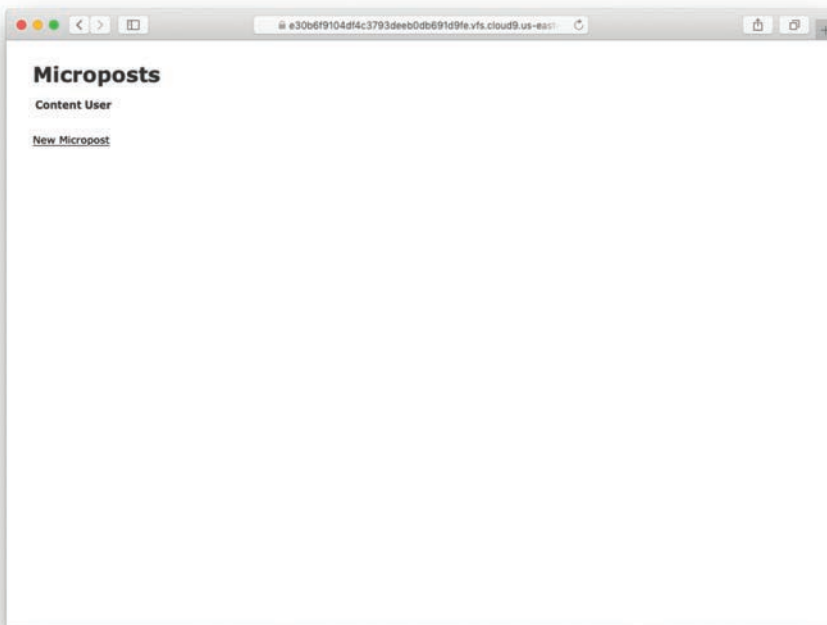


Figure 2.12: The micropost index page (`/microposts`).

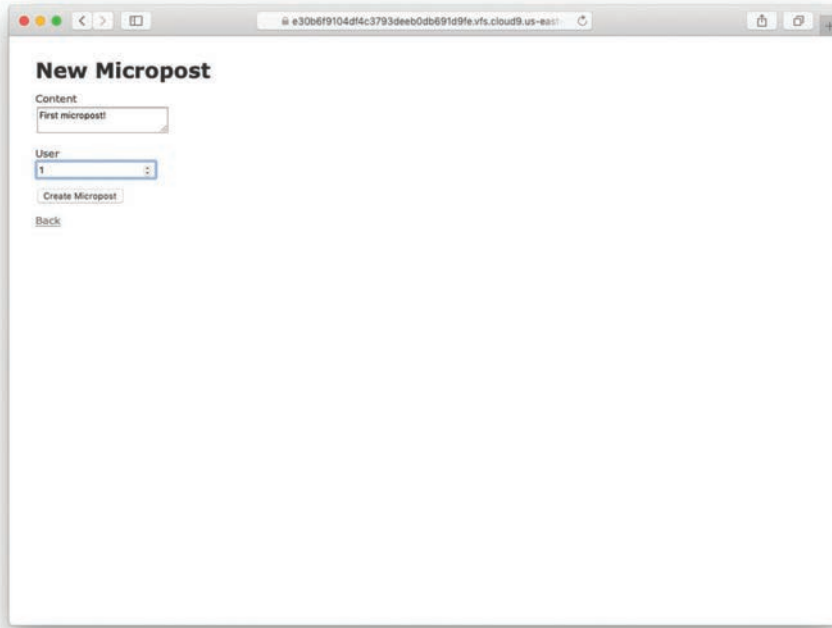


Figure 2.13: The new micropost page (`/microposts/new`).

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers at <https://www.railstutorial.org/aw-solutions>.

To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. (For readers who know CSS) Create a new micropost, then use your browser's HTML inspector to determine the CSS id for the text "Micropost was successfully created." What happens when you refresh your browser?
2. Try to create a micropost with empty content and no user id.
3. Try to create a micropost with more than 140 characters of content (say, the first paragraph from the Wikipedia article on Ruby).
4. Destroy the microposts from the previous exercises.

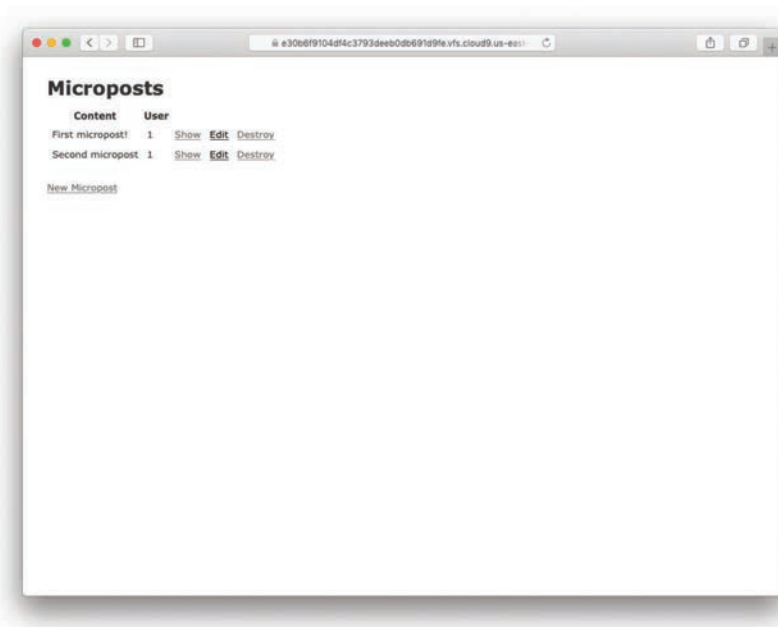


Figure 2.14: The micropost index page with a couple of posts.

2.3.2 Putting the *Micro* in Microposts

Any *micropost* worthy of the name should have some means of enforcing the rules governing the length of the post. Implementing this constraint in Rails is easy with *validations*; to accept microposts with at most 140 characters (à la the original design of Twitter), we use a *length* validation. At this point, you should open the file `app/models/micropost.rb` in your text editor or IDE and fill it with the contents of Listing 2.14.

Listing 2.14: Constraining microposts to be at most 140 characters.

`app/models/micropost.rb`

```
class Micropost < ApplicationRecord
  validates :content, length: { maximum: 140 }
end
```

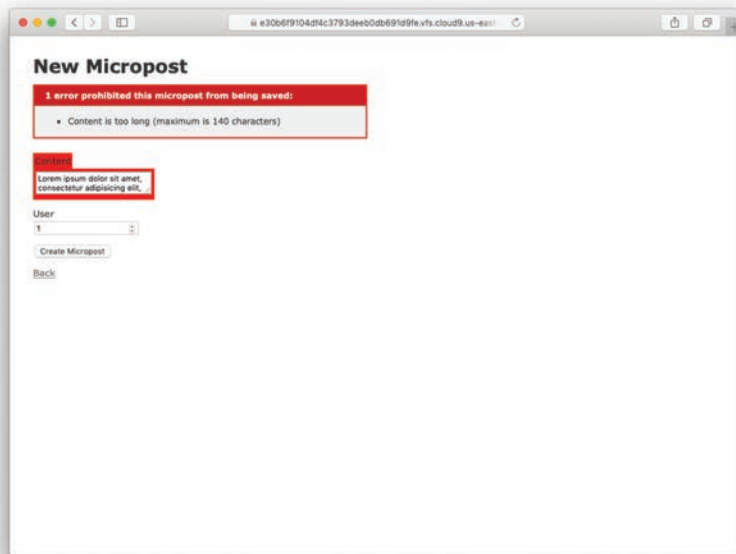


Figure 2.15: Error messages for a failed micropost creation.

The code in Listing 2.14 may look rather mysterious—we’ll cover validations more thoroughly starting in Section 6.2—but its effects are readily apparent if we go to the new micropost page and enter more than 140 characters for the content of the post. As seen in Figure 2.15, Rails renders *error messages* indicating that the micropost’s content is too long. (We’ll learn more about error messages in Section 7.3.3.)

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers at <https://www.railstutorial.org/aw-solutions>.

To see other people’s answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. Try to create a micropost with the same long content used in a previous exercise (Section 2.3.1). How has the behavior changed?
2. (For readers who know CSS) Use your browser’s HTML inspector to determine the CSS id of the error message produced by the previous exercise.

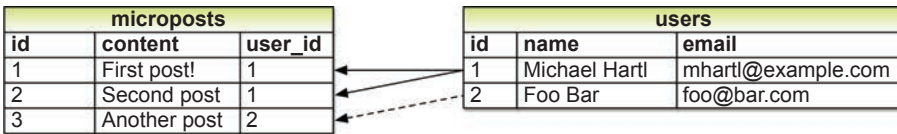


Figure 2.16: The association between microposts and users.

2.3.3 A User has_many Microposts

One of the most powerful features of Rails is the ability to form *associations* between different data models. In the case of our User model, each user potentially has many microposts. We can express this relationship in code by updating the User and Micropost models as in Listing 2.15 and Listing 2.16, respectively.

Listing 2.15: A user has many microposts.

app/models/user.rb

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

Listing 2.16: A micropost belongs to a user.

app/models/micropost.rb

```
class Micropost < ApplicationRecord
  belongs_to :user
  validates :content, length: { maximum: 140 }
end
```

We can visualize the result of this association in Figure 2.16. Because of the **user_id** column in the **microposts** table, Rails (using Active Record) can infer the microposts associated with each user.

In Chapter 13 and Chapter 14, we will use the association of users and microposts both to display all of a user’s microposts and to construct a Twitter-like micropost feed. For now, we can examine the implications of the user–micropost association by using

the *console*, which is a useful tool for interacting with Rails applications. We first invoke the console with **rails console** at the command line, and then retrieve the first user from the database using **User.first** (putting the results in the variable **first_user**), as shown in Listing 2.17.⁸ (I include **exit** in the last line just to demonstrate how to exit the console. On most systems, you can also use Ctrl-D for the same purpose.)⁹

Listing 2.17: Investigating the state of the application using the Rails console.

```
$ rails console
>> first_user = User.first
(0.5ms) SELECT sqlite_version(*)
User Load (0.2ms) SELECT "users".* FROM "users" ORDER BY "users"."id" ASC
LIMIT ? [["LIMIT", 1]]
=> #<User id: 1, name: "Michael Hartl", email: "michael@example.org",
created_at: "2019-08-20 00:39:14", updated_at: "2019-08-20 00:41:24">
>> first_user.microposts
Micropost Load (3.2ms) SELECT "microposts".* FROM "microposts" WHERE
"microposts"."user_id" = ? LIMIT ? [["user_id", 1], ["LIMIT", 11]]
=> #<ActiveRecord::Associations::CollectionProxy [#<Micropost id: 1, content:
"First micropost!", user_id: 1, created_at: "2019-08-20 02:04:13", updated_at:
"2019-08-20 02:04:13">, #<Micropost id: 2, content: "Second micropost",
user_id: 1, created_at: "2019-08-20 02:04:30", updated_at: "2019-08-20
02:04:30">]>
>> micropost = first_user.microposts.first
Micropost Load (0.2ms) SELECT "microposts".* FROM "microposts" WHERE
"microposts"."user_id" = ? ORDER BY "microposts"."id" ASC LIMIT ?
[["user_id", 1], ["LIMIT", 1]]
=> #<Micropost id: 1, content: "First micropost!", user_id: 1, created_at:
"2019-08-20 02:04:13", updated_at: "2019-08-20 02:04:13">
>> micropost.user
=> #<User id: 1, name: "Michael Hartl", email: "michael@example.org",
created_at: "2019-08-20 00:39:14", updated_at: "2019-08-20 00:41:24"
>> exit
```

There's a lot going on in Listing 2.17, and teasing out the relevant parts is a good exercise in technical sophistication (Box 1.2). The output includes the actual return

8. Your console prompt might be something like **2.6.3 :001 >**, but the examples use **>>** since Ruby versions will vary.

9. As in the case of Ctrl-C, the capital "D" refers to the key on the keyboard, not the capital letter, so you don't have to hold down the Shift key along with the Ctrl key.

values, which are raw Ruby objects, as well as the structured query language (SQL) code that produced them.

In addition to retrieving the first user with `User.first`, Listing 2.17 shows two other things: (1) how to access the first user’s microposts using the code `first_user.microposts`, which automatically returns all the microposts with `user_id` equal to the id of `first_user` (in this case, `1`); and (2) how to return the user corresponding to a particular post using `micropost.user`. We’ll learn much more about the Ruby involved in Listing 2.17 in Chapter 4, and more about the association facilities in Active Record in Chapter 13 and Chapter 14.

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers at <https://www.railstutorial.org/aw-solutions>.

To see other people’s answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. Edit the user show page to display the content of the user’s first micropost. (Use your technical sophistication (Box 1.2) to guess the syntax based on the other content in the file.) Visit `/users/1` to confirm that it worked.
2. The code in Listing 2.18 shows how to add a validation for the presence of micropost content to ensure that microposts can’t be blank. Verify that you get the behavior shown in Figure 2.17.
3. Update Listing 2.19 by replacing `FILL_IN` with the appropriate code to validate the presence of name and email attributes in the User model (Figure 2.18).

Listing 2.18: Code to validate the presence of micropost content.

`app/models/micropost.rb`

```
class Micropost < ApplicationRecord
  belongs_to :user
  validates :content, length: { maximum: 140 },
  presence: true
end
```

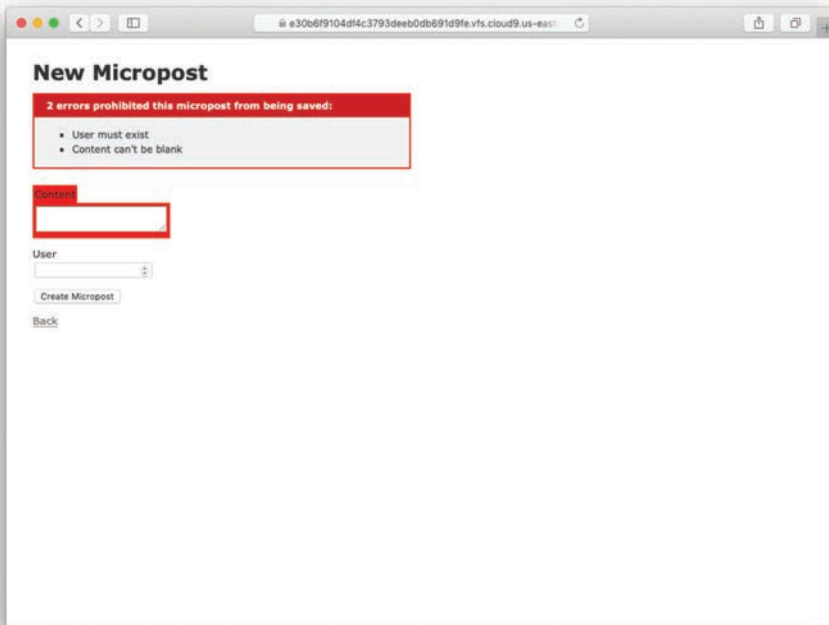


Figure 2.17: The effect of a micropost presence validation.

Listing 2.19: Adding presence validations to the User model.

app/models/user.rb

```
class User < ApplicationRecord
  has_many :microposts
  validates FILL_IN, presence: true # Replace FILL_IN with the right code.
  validates FILL_IN, presence: true # Replace FILL_IN with the right code.
end
```

2.3.4 Inheritance Hierarchies

We end our discussion of the toy application with a brief description of the controller and model class hierarchies in Rails. This discussion will make more sense if you

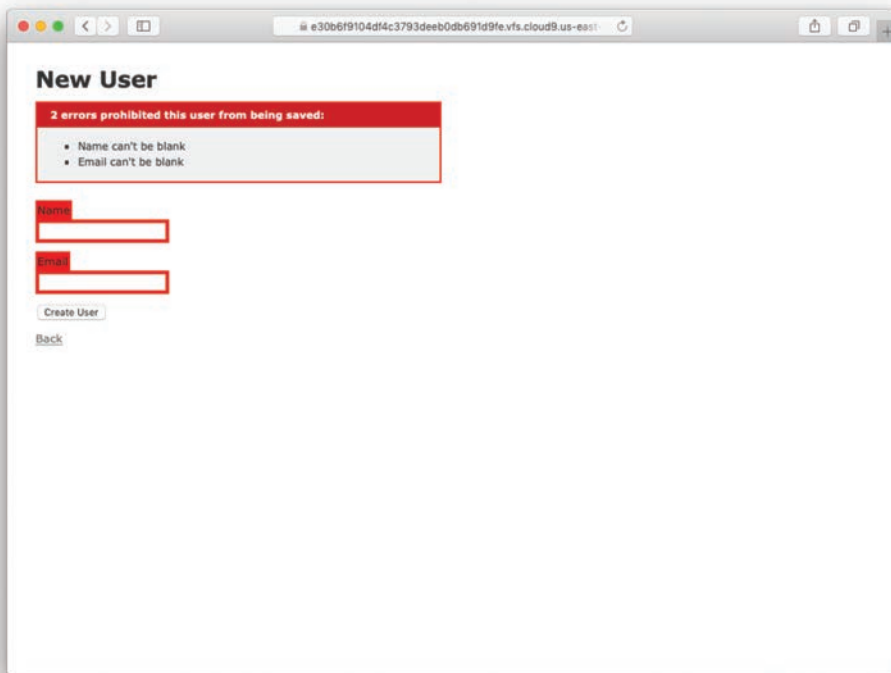


Figure 2.18: The effect of presence validations on the User model.

have some experience with object-oriented programming (OOP), particularly *classes*. Don't worry if it's confusing for now; we'll discuss these ideas more thoroughly in Section 4.4.

We start with the inheritance structure for models. Comparing Listing 2.20 and Listing 2.21, we see that both the User model and the Micropost model inherit (via the left angle bracket `<`) from **ApplicationRecord**, which in turn inherits from **ActiveRecord::Base**, which is the base class for models provided by Active Record; a diagram summarizing this relationship appears in Figure 2.19. By inheriting from **ActiveRecord::Base**, our model objects gain the ability to communicate with the database, treat the database columns as Ruby attributes, and so on.

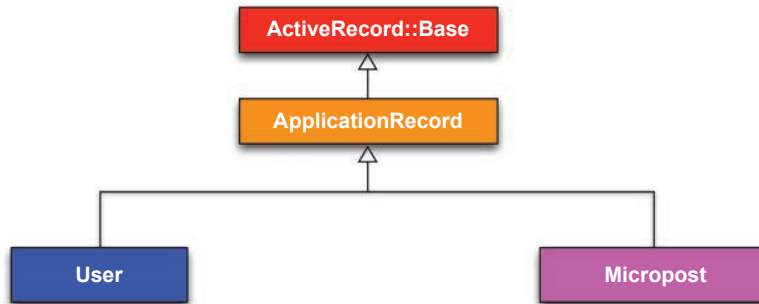


Figure 2.19: The inheritance hierarchy for the User and Micropost models.

Listing 2.20: The `User` class, highlighting inheritance.

`app/models/user.rb`

```

class User < ApplicationRecord
  .
  .
  .
end
  
```

Listing 2.21: The `Micropost` class, highlighting inheritance.

`app/models/micropost.rb`

```

class Micropost < ApplicationRecord
  .
  .
  .
end
  
```

The inheritance structure for controllers is essentially the same as that for models. Comparing Listing 2.22 and Listing 2.23, we see that both the Users controller and the Microposts controller inherit from the Application controller. Examining Listing 2.24, we see that `ApplicationController` itself inherits from `ActionController::Base`, which is the base class for controllers provided by the Rails library Action Pack. The relationships between these classes are illustrated in Figure 2.20.

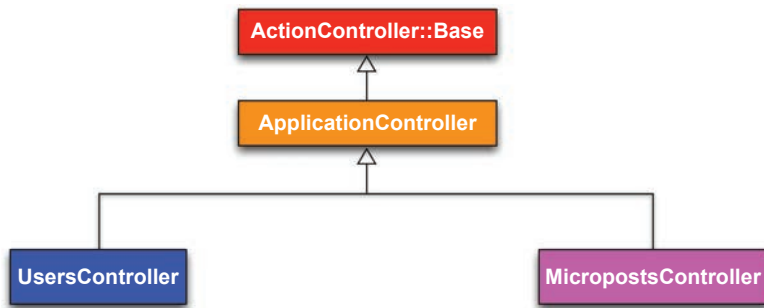


Figure 2.20: The inheritance hierarchy for the Users and Microposts controllers.

Listing 2.22: The **UsersController** class, highlighting inheritance.

app/controllers/users_controller.rb

```

class UsersController < ApplicationController
  .
  .
  .
end
  
```

Listing 2.23: The **MicropostsController** class, highlighting inheritance.

app/controllers/microposts_controller.rb

```

class MicropostsController < ApplicationController
  .
  .
  .
end
  
```

Listing 2.24: The **ApplicationController** class, highlighting inheritance.

app/controllers/application_controller.rb

```

class ApplicationController < ActionController::Base
  .
  .
  .
end
  
```

As with model inheritance, both the Users and Microposts controllers gain a large amount of functionality by inheriting from a base class (in this case, **ActionController::Base**), including the capability to manipulate model objects, filter inbound HTTP requests, and render views as HTML. Since all Rails controllers inherit from **ApplicationController**, all rules defined in the Application controller automatically apply to every action in the application. For example, in Section 9.1 we'll see how to include helpers for logging in and logging out of all of the sample application's controllers.

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers at <https://www.railstutorial.org/aw-solutions>.

To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. By examining the contents of the Application controller file, find the line that causes **ApplicationController** to inherit from **ActionController::Base**.
2. Is there an analogous file containing a line where **ApplicationRecord** inherits from **ActiveRecord::Base**? *Hint*: It would probably be a file called something like **application_record.rb** in the **app/models** directory.

2.3.5 Deploying the Toy App

With the completion of the Microposts resource, now is a good time to push the repository up to GitHub:

```
$ git status      # It's a good habit to check the status before adding
$ git add -A
$ git commit -m "Finish toy app"
$ git push
```

Ordinarily, you should make smaller, more frequent commits, but for the purposes of this chapter a single big commit at the end is fine.

At this point, you can also deploy the toy app to Heroku as is described in Section 1.4:

```
$ git push heroku
```

(This assumes you created the Heroku app in Section 2.1. Otherwise, you should run **heroku create** and then **git push heroku master**.)

At this point, visiting the page at Heroku yields an error message, as shown in Figure 2.21. We can track down the problem by inspecting the Heroku logs:

```
$ heroku logs
```

Scrolling up in the logs, you should see a line that includes something like this:

```
ActionView::Template::Error (PG::UndefinedTable: ERROR: relation "users" does not exist)
```

This line is a big hint that there is a missing **users** table. Luckily, we learned how to handle that problem way back in Listing 2.4: All we need to do is run the database migrations (which will create the **microposts** table as well).

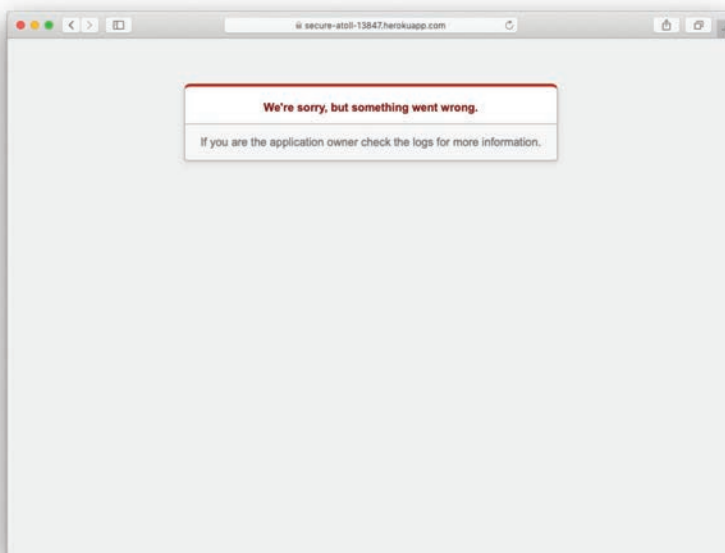


Figure 2.21: An error page at Heroku.

The way to execute this sort of command at Heroku is to prefix the usual Rails command with **heroku run**, like this:

```
$ heroku run rails db:migrate
```

This updates the database at Heroku with the user and micropost data models as required. After running the migration, you should be able to use the toy app in production, with a real PostgreSQL database back end (Figure 2.22).¹⁰

Finally, if you completed the exercises in Section 2.3.3, you will have to remove the code to display the first user's micropost to get the app to load properly. In this case, simply delete the offending code, make another commit, and push again to Heroku.

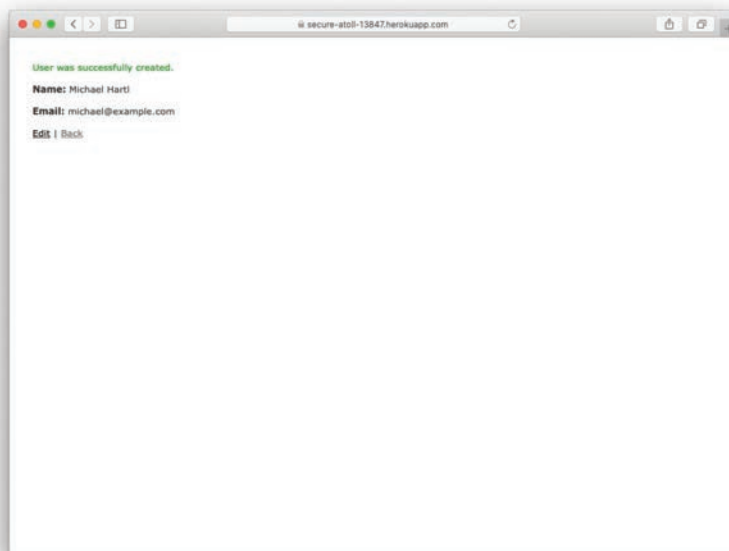


Figure 2.22: Running the toy app in production.

10. The production database should work without any additional configuration, but in fact some configuration is recommended by the official Heroku documentation. We'll take care of this detail in Section 7.5.3.

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers at <https://www.railstutorial.org/aw-solutions>.

To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. Create a few users on the production app.
2. Create a few production microposts for the first user.
3. By trying to create a micropost with content exceeding 140 characters, confirm that the validation from Listing 2.14 works on the production app.

2.4 Conclusion

We've come now to the end of the high-level overview of a Rails application. The toy app developed in this chapter has several strengths and a host of weaknesses.

Strengths

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

Weaknesses

- No custom layout or styling
- No static pages (such as "Home" or "About")
- No user passwords
- No user images
- No logging in
- No security
- No automatic user/micropost association

- No notion of “following” or “followed”
- No micropost feed
- No meaningful tests
- **No real understanding**

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

2.4.1 What We Learned in this Chapter

- Scaffolding automatically creates code to model data and interact with it through the web.
- Scaffolding is good for getting started quickly but is bad for understanding.
- Rails uses the model-view-controller (MVC) pattern for structuring web applications.
- As interpreted by Rails, the REST architecture includes a standard set of URLs and controller actions for interacting with data models.
- Rails supports data validations to place constraints on the values of data model attributes.
- Rails comes with built-in functions for defining associations between different data models.
- We can interact with Rails applications at the command line using the Rails console.

This page intentionally left blank

Index

Symbols

..(double dots), 285–286
\\ (literal backslash), 163
\n (newline), 160, 162–163
||(or) operator, 165
||= (or equals) operator, 398
|i|, block variable syntax, 175–176
+ (plus) operator, 161
== (equality comparison operator), 172, 442
=> (hashrocket), hashes, 179, 180
!(not) operator, 165
#{ } (interpolation) of strings, 161–162
%w [] technique, arrays of strings, 280–281
&& (“and”) operator, 63, 165
[] (square brackets), arrays, 171
{ } (curly braces), 175, 179
<< (shovel operator), arrays, 173
>> (append) command, 9

A

about action

- generating controllers, 110
- testing static pages, 125–126

About page

- adding titles for static pages, 132–134
- testing static pages, 124–125, 127–128
- viewing with embedded Ruby title, 137–139

- viewing with HTML structure removed, 140–141

Access control

- admin, 534
- micropost, 658–661
- tests for relationships, 757–758

Account activation

- controller, 541–542
- data model, 542–548
- edit action, 571–574
- email generally, 549
- email in production, 581–584
- exercises, 542
- generalizing `authenticated?` 565–570
- mailer templates, 549–554
- modeling as resource, 541–549
- overview of, 539
- previewing email, 554–558
- review, 584–585
- sequence for, 539–540
- testing and refactoring, 574–581
- testing email, 558–561
- token callback, 545–547
- updating `create` action, 561–565

`account_activation` method, 549–554

Accounts

- creating/configuring Heroku, 51
- signing up for GitHub, 39

- Actions. *See also* by individual types
 - adding controller action, 28–31, 739
 - adding for Contact page, 238
 - creating set of, 108
 - generating controllers, 110
 - Micropost controller, 83–84
 - organizing functions, 116–117
 - routes and, 113–115
 - Sessions controller, 376–377, 384–385
 - testing static pages, 125–127
 - Users controller, 71–72, 76–77, 249
- `activate` method, 576, 579
- `activated` attribute
 - `activated_at` timestamp, 571
 - User model, 543
- Activation digest
 - account activation data model, 543–544
 - account activation sequence, 539–540
 - activating account, 566–567
 - activation token callback, 545–547
 - testing account activation, 580
- Active Record
 - `ApplicationRecord` class inherits from `ActiveRecord::Base`, 92–94
 - associations, 631
 - creating, saving, and finding data objects, 257
 - creating user objects, 265
 - finding users, 268–269
 - uniqueness validation and, 289
 - User model based on `ActiveRecord::Base`, 264
- Active Storage
 - adding gem for validation, 696
 - `has_many_attached` method, 690–691
- `add_index`, 291
- `add_password_digest_to_users`, 297
- `admin` attribute
 - deleting users, 526–530
 - restricting `destroy` action to administrative users, 533–536
 - signup failure page, 344, 346
- Administrative users
 - deleting users, 526–530
 - granting users administrative access, 709
 - making fixture user an admin, 533–534
 - restricting `destroy` action to administrative users, 533–536
- Advanced login. *See* Login, advanced
- Ajax
 - buttons for follow/unfollow, 759–761
 - form for follow/unfollow, 761–763
 - responding to Ajax requests in Relationships controller, 763
- `alert-danger`
 - form submission error message, 348
 - iterating through `flash` hash in console, 359
- Alignment, applying universal styling on pages, 216
- `all` method, finding users, 269
- Amazon Web Services. *See* AWS (Amazon Web Services)
- “And” (&&) operator, 63, 165
- Anonymous function, “stabby lambda” syntax, 641
- API, Ruby, 160
- Append (>>) command, 9
- `ApplicationController` class
 - deploying toy app, 63
 - inheritance hierarchy, 93–95
- `application.css`, 154–155
- `ApplicationHelper` class, 156
- `ApplicationMailer` class, 550–551
- `ApplicationRecord` class
 - model file for User model, 264
 - User class inherits from, 92–93
- Arrays
 - arrays of strings (%w []) technique, 280–281
 - blocks and, 175–178
 - constructors and, 186
 - hashes resemble, 178–183
 - overview of, 170–173
 - ranges and, 173–174
- `aside` tag, adding sidebar to profile page, 327–331
- `assert` method, validity test, 273
- `assert_difference` method, 364–365, 535
- `assert_equal` method, 470
- `assert_match` method, 559, 657

- `assert_no_difference` method, 352–354
- `assert_not` method, 274–275
- `assert_select`
 - compared with `asset_match`, 657
 - testing invalid signup submission, 352–355
 - testing user login with valid information, 420
 - uses of, 246–248
- `assert_template`, 353
- Asset directories, 226
- Asset pipeline
 - defining variables using Sass, 231–233
 - directories, 226
 - manifest files, 226–228
 - nested elements in stylesheets, 229–231
 - nesting and variables used in converting SCSS file, 233–235
 - overview of, 225–226
 - preprocessor engines, 228
 - production efficiency of, 228
 - Sass and, 228–229
- Assignment operators, 398
- `assigns` method, testing account activation, 576
- Associations
 - Active Record, 631
 - adding `followers` association, 733–735
 - adding `following` association, 730–732
 - `belongs_to` and `has_many` examples, 636–637
 - micropost, 634–638
 - between microposts and users, 88–91
 - summary of micropost association methods, 635
 - summary of relationship association methods, 728
 - testing `followers` association, 735
 - testing `following` utility methods, 732–733
 - User/Micropost, 634
 - between users and relationships, 725–728
- `attr_accessible` method, 344
- `attr_accessor` method
 - activation account, 546–547
 - database migrations, 258
 - User class, 195–197
- `attribute` variable
 - activating accounts, 567
 - User class, 196
- `authenticate` method, 303–305
- `authenticated?` method
 - account activation and, 540
 - data model for account activation, 543
 - generalizing for activating accounts, 565–570
 - storing user id in cookies, 442–443
 - testing logging out, 452–453
 - testing nonexistent digest, 454–455
 - testing remember branch, 470
 - updating to handle nonexistent digest, 455
- Authentication. *See also* Account activation
 - commenting out authentication code, 420–421
 - comparing with authorization, 491
 - data model for account activation, 543
 - in login system, 256–257
 - Rails using HTTP, 54
 - of users, 303–305, 383–385, 387–388
- authenticity token, in signup form, 339
- Authorization
 - adding `store_location` to logged-in user, 505–506
 - `create` action used with friendly forwarding, 506–507
 - `current_user?` method, 500–502
 - friendly forwarding, 502–503
 - implementing friendly forwarding, 504
 - logging in test user, 494–495
 - overview of, 491
 - protecting edit/update pages, 499
 - requiring login, 491–494
 - requiring right user, 497–498
 - testing editing by wrong user, 498
 - testing friendly forwarding, 503–504
 - testing Micropost controller, 659
 - testing protection of `edit` and `update`, 495–497
- Automated tests
 - overview of, 118
 - using *Guard*, 148–151

- AWS (Amazon Web Services)
 - configuring for image upload, 706–711
 - downloading, 208–210
 - hiding all images, 220
 - IAM (Identity and Access Management), 707–711
 - production AWS, 711–714
 - signing up for, 707
- B**
- Bang methods, array, 172
- `base_title`, 155–158
- Basecamp collaboration tool, 2
- Bash, Unix command line and, 14
- `BasicObject` superclass, 187–188
- `bcrypt`
 - creating password digest, 416–417
 - creating password digest with `bcrypt` gem, 298
 - storing user id in cookies with `bcrypt` gem, 441–442
- Before filters
 - commenting out, 495
 - for correct user, 500–502
 - implementing with `before_action` command, 491
 - protecting edit/update pages, 499–500
 - requiring logged-in users, 491–494
 - restricting `destroy` action to administrative users, 533–536
 - uncommenting, 496
- `before_action` command, 491
- `before_create` callback, activation tokens, 545–546
- `before_save` callback, converting email to lowercase, 292, 294–295, 545
- `belongs_to` method
 - association model, 627
 - user/relationship associations, 634–637, 725–728
- `blank?` method, modifying built-in classes, 191
- Block variable syntax (`|i|`), 175–176
- Blocks, 175–178
- Booleans
 - `admin` attribute, 526–527
 - checking login status, 424
 - control flow with, 165
 - `current_user?` Boolean method, 500–501
 - `following?` Boolean method, 731–733
- Bootstrap
 - adding Bootstrap CSS, 213–214
 - adding `bootstrap-sass` gem to Gemfile, 212–213
 - applying CSS to site logo, 218–219
 - commenting out embedded Ruby, 219
 - creating dropdown menus, 404–406
 - creating menu, 401–402
 - hiding all images, 220
 - overview of, 210–212
 - pagination styles, 516
 - partials used for HTML shim, 221
 - partials used for site footer, 222–225
 - partials used for site header, 221–222
 - partials used for stylesheets and header, 220–221
 - replacing default `head` with a call to `render`, 225
 - for typographic effect, 216–218
 - for universal styling on all pages, 214–216
- `bootstrap-sass` gem, 212–213
- `box_sizing` mixin, 314–315, 333
- Branches, GitHub, 43–44
- Built-in classes, modifying, 190–192
- Built-in helpers, 154–155
- `bundle install`
 - Heroku setup for deployment, 51
 - installing gems, 22
 - for sample app, 103
 - without production gems, 51, 62
- `bundle update`
 - installing gems, 22
 - planning toy app, 62
 - for sample app, 104
- Bundler
 - installing gems for new Rails app, 17–22
 - planning toy app, 61–62
- Buttons
 - Ajax buttons for follow/unfollow, 759–761

- standard buttons for follow/unfollow, 757–759
 - testing follow/unfollow buttons, 766–767
- byebug gem, debug information, 322–323
- C**
- Callback
 - before_save callback, 294–295
 - alternative implementation, 295
 - standardizing all lower-case addresses via, 292
 - token callback, 545–547
- Cascading Style Sheets. *See* CSS (Cascading Style Sheets)
- Case
 - CamelCase for class names, 111
 - testing email uniqueness for case-insensitivity, 287–290
- Certificate, SSL, 369
- CGI.escape, escaping email, 554
- change method, 260, 263
- Character sets, unicode, 139
- Characters
 - length returns number of string, 164
 - literal strings contain typed, 163
 - ranges work with, 174
 - strings contain, 160
 - symbols start with letter/using normal word, 180
- check_expiration method, password reset, 611
- Checkbox
 - making, 456–462
 - testing, 462–468
- checkout command, 43–44
- Classes
 - assigning to HTML elements, 205
 - CamelCase for names, 111
 - class methods, 186, 417–418
 - constructors, 185–187
 - Controller class, 192–195
 - hierarchy, 193
 - hierarchy and inheritance, 187–190, 193–194
 - mixing modules into, 170
 - modifying built-in, 190–192
 - static pages, 116
 - User class, 195–198
- Clients, testing application code, 120
- Cloud IDE
 - account activation email previews, 556–558
 - development environment on, 7
 - Git system setup, 34–35
- Cloud9, AWS
 - creating new work environment with, 7–9
 - getting started with, 4–5
- Code
 - commenting out, 219, 420–421
 - simplifying, 422
 - undoing, 112–113
- Color, HTML, 218–219
- Command-line
 - cloud IDE terminal, 7
 - conventions used in this book, 56–58
 - Heroku setup for deployment, 51–52
- Comments
 - commenting out authentication code, 420–421
 - commenting out before filters, 495
 - commenting out embedded Ruby, 219
 - Ruby ignores, 160
 - signup failure page, 341
- commit command, Git repository, 37, 45–47, 95
- config.force_ssl, forcing browsers to use SSL, 368–369
- Configuration settings, installing Rails, 9–10
- Console, AWS, 710–711
- Console, Rails
 - console command, 89
 - creating user objects, 270–271
 - creating/authenticating user, 303–305
 - exploring strings, 160–162
 - iterating through flash hash in, 359
 - as learning tool, 159–160
 - making user profile page, 316–322
 - running in different environment, 313
 - simulating user session in, 400
 - simulating user sessions, 400
 - starting in sandbox for rollback, 264

- Constructors, 185–187
- Contact page
 - code for, 142
 - layout links, 236–238
- `content` field, in data model for toy app, 65
- Content Security Policy (CSP), 140
- Control flow, Booleans for, 165–166
- Controllers
 - Account Activation, 541–542
 - actions in, 108
 - authorization tests for Micropost controller, 659
 - Controller class, 192–195
 - correcting invalid micropost on Home page, 677–678
 - creating Users controller, 248–250
 - `destroy` action in Micropost controller, 682
 - before filters use with, 493
 - generating for static page app, 109–112
 - inheritance hierarchy, 93–95
 - Micropost controller, 82–86
 - for password reset, 590–593
 - responding to Ajax requests in Relationships controller, 763
 - test options, 121
 - user views browser hits using MVC, 74–75
 - Users controller, 76–78
- Conventions, used in this book, 56–58
- Cookies
 - creating permanent, 432–439
 - implementing sessions, 376
 - login with permanent, 439–448
 - session hijacking attacks and, 394
 - testing remember me checkbox, 465
- `cookies` method, 432–439
- `cookies.permanent` method, 439–441
- `correct_user`, 498, 500–502
- `count`
 - displaying number of microposts, 647
 - displaying signup error messages, 348–349
 - testing invalid signup submission, 352–355
- `create`
 - account activation emails, 549
 - creating/authenticating users, 303–305
 - finished signup form, 355–358
 - friendly forwarding and, 506–507
 - handling signup failure, 341
 - microposts, 662–663
 - Microposts controller, 83
 - new session, 384–385
 - parameters in, 345
 - password reset, 596–599
 - profile page, 316–322
 - signup failure page, 339
 - testing invalid signup submission, 353
 - updating users, 561–565
 - user objects, 264–268
 - Users controller, 76–77
- Create User button, 69–70
- `create_activation_digest` method, 545–546, 549
- `create_table` method, 260
- `created_at` column
 - creating user objects, 265–268
 - Micropost migration with, 630
 - updating user objects, 271
- Cross-site request forgery (CSRF) attack, 339
- Cross-site scripting attacks (XSS), 140
- Cryptography, persistent cookie sessions and, 432
- CSP (Content Security Policy), 140
- CSRF (cross-site request forgery) attack, 339
- CSS (Cascading Style Sheets)
 - adding styling to users index, 511
 - applying to site logo, 218–219
 - commenting out embedded Ruby, 219
 - custom CSS. *See* Bootstrap
 - debug output using rules, 314–315
 - hiding all images, 220
 - HTML code produced by, 183–185
 - partials for stylesheets, 220–221
 - for remember me checkbox, 458
 - styling error messages, 349–350
 - styling microposts, 651–653
 - styling user show page with SCSS, 328–329
 - for typographic effect, 216–218
 - for universal styling on all pages, 214–216

- updating footer CSS, 414–416
 - user signup form, 334–335
 - Curly braces (`{ }`), 175, 179
 - `current_user?` 500–502
 - `current_user`
 - defining, 396–401
 - generalized `authenticated?` method, 569–570
 - subtle bugs, 451–453
 - testing remember branch, 468–471
 - updating for persistent sessions, 444–447
 - Custom CSS. *See* Bootstrap
 - Custom helpers, 155–156
- D**
- Data, as resources in REST, 317
 - Data models
 - account activation, 542–548
 - associations between microposts and users, 88–91
 - creating User data model, 257–263
 - for following users, 719–725
 - inheritance structures, 92–93
 - Micropost model, 64–65, 627–628
 - for user relationships, 723–724
 - for web application, 64
 - Databases
 - creating micropost table in, 629–630
 - creating SQLite, 261–262
 - indices, 290–291
 - migrations, 257–263
 - reseeding user database, 528
 - sample of following data, 736–738
 - saving user object to, 265–266
 - seeding user database (`db:seed`), 515
 - working with long-term data storage in Rails, 257
 - DB Browser, viewing SQLite database, 261–262
 - `db/` directory, 263
 - `db:migrate`
 - creating Users resource, 66
 - migrating database, 105
 - migrating up, 261
 - rerunning migration, 263
 - undoing migration, 112–113
 - `db:rollback:`, 263
 - `db:seed`
 - reseeding user database, 528
 - sample of following data, 736–738
 - seeding user database, 515
 - Debug
 - adding CSS rules to output, 314–315
 - adding information to site layout, 312, 314–316
 - remembering user in login, 451–456
 - signup error messages, 345–346
 - signup failure page, 342–343
 - `debugger`, Users controller with, 322–324
 - Default scope, micropost, 638–641
 - Delete
 - administrative user privilege required, 526–530
 - `destroy` action, 530–533
 - microposts, 680–681
 - testing, 533–536
 - user objects, 267
 - users, 525–526
 - `delete logout_path`, 453
 - DELETE operation, HTTP, 114–115
 - DELETE requests, adding Users resource to routes file, 319
 - `deliveries` array, 576
 - Dependability, advantages of Rails, 2
 - `dependent: :destroy`, micropost, 641–643
 - Deployment
 - with Heroku, 49–55
 - signup page, 367–373
 - testing suite before, 146
 - too early, 107–108
 - toy app, 63, 95–98
 - `destroy`
 - deleting microposts, 680–684
 - deleting user objects, 267
 - deleting users, 530–533
 - Microposts controller, 84
 - testing destroying users, 533–536
 - undoing code generation, 112
 - Users controller, 71–72, 76

Development environment
 account activation email previews, 555
 as default console, 159, 313
 restricting debug information to, 312

Devise gem, authentication with, 255

digest method
 activation token callback, 546
 use in fixtures, 417–418

Directory
 asset directories, 226
 conventions in this book for separators, 57
 for Rails projects, 14
 structure for newly created Rails app, 16–17

div, Home page element, 205–206

do.end, longer/multi-line blocks, 175–176

Don't Repeat Yourself (DRY), code guideline, 135

Double dots (**.**), 285–286

Double-quoted strings, 161–164

downcase method
 converting email to lowercase, 545
 enforcing email uniqueness, 292–295

drop_table command, rollback migration, 263

Dropdown menus, 404–406

DRY (Don't Repeat Yourself), code guideline, 135

Duck typing, object manipulation, 269

Duplicate email addresses, test for, 286–287

E

each method
 email format validation, 281
 hashes, 181–182

echo command
 installing Rails, 9–10
 printing strings, 162

Edit, GitHub, 44–45

edit action
 account activation, 541, 571–574
 editing users, 476–477
 Microposts controller, 83
 password reset, 607–609
 testing protection of, 495–498
 Users controller, 76–77

Edit form
 HTML for, 479
 partials use with, 482–483

Edit pages
 friendly forwarding, 502–503
 protecting, 499

edit view, 478, 482

Editing users
 adding URL to "Settings" link in site layout, 480–481
 overview of, 476–480
 partials use for, 481–483
 successful edits, 486–488
 testing editing by wrong user, 498
 testing successful edits, 488
 testing unsuccessful edits, 484–486
 unsuccessful edits, 483–484
 updating user editing errors, 668–670

elsif value, Booleans, 165

Email. *See also* Uniqueness validation, email
 account activation, 549
 account activation in production, 581–584
 adding secure password, 295–305
 development of password reset, 599
 disallowing double dots in domain names, 285–286
 first signup, 362
 "forgot password" form, 599
 Git system setup, 34
 mailer and templates for password reset, 600–605
 previewing, 554–558
 production of password reset, 621–624
 testing, 558–561
 testing password reset, 605–606
 as unique username, 257–261
 user profile page with Gravatar, 324

email attribute
 creating user objects, 265
 data model for, 64
 database migrations, 257–261
 imposing constraints (validations) on, 271
 signup failure page, 343
 testing format validation, 280–286

- testing length validation, 278–280
 - testing presence validation, 276–277
- @email variable, `User` class, 195–197
- Embedded Ruby. *See* ERb (Embedded Ruby)
- `empty?` method
 - signup error messages, 348–349
 - strings, 164–166
- `encrypted` method, storing user id in cookies, 441
- ENV variables, Heroku, 711
- `environment` directory, for Rails project, 14
- Environments, Rails
 - overview of, 313–314
 - types of, 159
- Equality comparison operator (`==`), 172, 442
- ERb (Embedded Ruby)
 - About page with embedded Ruby title, 137–139
 - commenting out, 219
 - filename extension for, 228
 - Help page with embedded Ruby title, 137
 - Home page with embedded Ruby title, 136–137
 - static page layouts and, 135
- Error messages
 - object errors, 666–667
 - password reset errors, 668
 - presence validation, 276
 - routing error, 679
 - signup failure page, 346–351
 - styling, 349–350
 - Update form, 485
 - user editing errors, 668–670
- Error page, at Heroku, 96–97
- `error_explanation`, 349–350
- Exceptions
 - finding users, 268
 - testing remember branch, 468–471
- Expiration, of passwords, 625–626
- `expires` date, cookies, 439–440
- F**
- Failed login, flash message for, 388–390
- `faker` gem, adding to `Gemfile`, 514–517
- `false` value
 - Booleans, 165
 - `nil` object as only object with, 166
- `feed` method, in `User` model, 671
- Feeds. *See* Status feed
- `field_with_errors`, 349–350
- Filename extensions, for Sass and ERb, 228
- Files
 - allowing only valid image formats, 699–701
 - checking file size with jQuery, 698–699
- Filesystem navigator, cloud IDE, 7
- Find
 - current user in a session, 399–400
 - current user with find method, 396–397
 - user objects, 268–269
 - users by name (`find_by_name`), 269
 - users with find method, 268–269
- Finished signup form, successful signups, 355–358
- First application
 - creating, 13–17
 - Hello, World! 28–33
 - model-view-controller (MVC), 27–28
 - `rails server` command, 22–27
 - using Bundler to install gems, 17–22
- `first` method, 269
- First signup, 361–364
- `first user` variable, 89–90
- Fixtures
 - activating by default, 548
 - adding 30 users to fixture file, 521
 - adding digest method for use with, 417–418
 - adding second user to fixture file, 497–498
 - downloading fixture image for use in tests, 695–696
 - for following/follower tests, 755
 - with generated microposts, 655–656
 - loading data into test database, 416–417
 - making fixture user an admin, 533–534
 - micropost, 639–640
 - removing contents of relationship fixture, 729
 - testing default, 291
 - testing empty file, 292
 - testing user login, 418–419

- Flash messages
 - adding to user signup form, 358–361
 - for failed login, 388–390
 - first signup, 362
 - testing flash behavior, 390–393
- flash method, 358–361
- Flash persistence, 390–392
- follow method, 731–733, 744
- follow_redirect! method, 365
- followers
 - action, 750–751
 - adding followers action to User Controller, 739
 - adding followers association, 733–735
 - pages for followed and followers, 748–750
 - partial for follower stats, 740–744
 - relationships, 723
 - rendering, 751–755
 - testing, 735, 755–757
 - testing followers association, 735
- following
 - action, 750–751
 - adding following action to User Controller, 739
 - adding following association, 730–732
 - pages for followed and followers, 748–750
 - relationships, 723
 - rendering, 751–755
 - testing, 732–733, 755–757
- following? Boolean method, 731–733
- Following users
 - adding followers association, 733–735
 - adding following and followers actions to User Controller, 739
 - adding following association, 730–732
 - adding forms and stats to user profile, 746–748
 - Ajax buttons for follow/unfollow, 759–761
 - Ajax form for follow/unfollow, 761–763
 - associations between users and relationships, 725–728
 - conclusion, 780–782
 - creating/destroying following relationship with JavaScript, 765
 - data model for, 719–725
 - degradation of form submission, 763–765
 - following and followers actions, 750–751
 - forms for follow/unfollow, 745–746
 - implementing status feed, 771–774
 - join method for making status feed, 780
 - overview of, 717–718
 - pages for followed and followers, 748–750
 - partial for follow/unfollow, 744
 - relationship model, 718–719
 - rendering following and followers, 751–755
 - responding to Ajax requests in Relationships controller, 763
 - routes for user relationships, 744–745
 - sample data, 736–738
 - standard buttons for follow/unfollow, 757–759
 - stats, 738, 740–744
 - status feed, 768–771
 - subselects in implementing status feed, 774–779
 - testing followers association, 735
 - testing following utility methods, 732–733
 - testing following/follower, 755–757
 - testing status feed HTML, 779–780
 - testing Web interface, 765–768
 - validating relationships, 728–730
 - Web interface for, 736
- footer
 - partial with links, 243
 - partials for site footer, 222–225
 - updating footer CSS, 414–416
- forget method, adding to User model, 449–451
- Forgetting users, 448–451
- ”Forgot password” form, 596, 599
- form tag, HTML, 338–339
- form_with
 - HTML for signup, 335–339
 - user signup form, 332–335

Formats

- allowing only valid image formats, 699–701
- validating email address, 280–286
- validating email format, 282–283

`formatted_email` method, `User` class,
196–197

Forms

- adding to user profile, 746–748
- Ajax form for follow/unfollow, 761–763
- for basic login, 380–383
- degradation of form submission, 763–765
- for follow/unfollow, 745–746
- “forgot password” form, 596
- new password resets form, 593–595
- password reset, 610

Forms, signup page

- HTML, 335–339
- overview of, 331–332
- using `form_with`, 332–335

Friendly forwarding

- adding `store_location` to logged-in user,
505–506
- code for implementing, 504
- `create` action used with, 506–507
- overview of, 502–503
- testing, 503–504

from address, mailer with new default, 550–551

`full_title` helper, 155–157, 169–170,
247–248

full-table scan, 290

Functions

- as methods in Ruby, 164
- packaging, 393
- static pages, 116

G

`gem` command, 10–11, 20

Gemfile

- adding `bootstrap-sass` gem, 212–213
- adding `faker` gem, 514–517
- Heroku setup for deployment, 50–51
- installing gems for new Rails app, 17–22
- making password digest, 298
- overview of, 101

planning toy app, 61

setting up static page app, 102–104

version control when deploying to Heroku,
372–373

`.gemrc` file, 10

Gems

- for Active Storage validation, 696
- adding `bootstrap-sass` gem, 212–213
- for image processing, 702–703
- installing, 10–11
- installing with Bundler, 20
- for new Rails app, 17–22
- planning toy app, 62

generate

- script, 109–112
- Sessions controller, 378
- Static Pages controller, 110, 146
- testing, 122
- undoing, 112
- user mailer, 549
- Users controller, 248–249
- Users resource, 65–66

`generate model`, 259, 628

generate scaffold

- Microposts resource, 81–82
- as scaffold generator, 60
- Users resource, 66

GET operation, HTTP, 114–115

GET requests

- adding Users resource to routes file, 317–321
- testing invalid signup submission, 352–354

Getter methods, `User` class, 195

Git. *See also* GitHub

- advantages of, 38–39
- Heroku deployment using, 52–53
- installation and setup, 34–38
- putting toy app under version control, 62–63
- version control with, 33–34

`git add -A` command, 36, 46

`git branch` command, 43

`git checkout (git co)`
command, 35

`git commit`, 46, 105

`git init` command, 36

- `git log` command, 38
 - `git push heroku master` command, 63, 96
 - GitHub
 - branch, 43–44
 - commit, 45–47
 - commit and push sequence, 111
 - creating new repository, 62–63, 106–107
 - as dependable developer platform, 2
 - deploying toy app to, 95–97
 - edit, 44–45
 - merge, 47–48
 - push, 48
 - signing up for, 39–42
 - `.gitignore` file, 36
 - goodbye action, Heroku deployment, 54
 - Gravatar
 - adding sidebar to user profile page, 327–329
 - image-editing interface, 486–487
 - profile page with, 322–326, 330–331
 - specifying Users file size, 511
 - `gravatar_for` helper method, 325–326
 - `@greeting` variable, 550
 - Guard*, automating tests, 148–151
 - Guardfile, 148–151
- ## H
- Hamburger menu
 - markup for, 411
 - for mobile device, 412
 - mobile styling, 409–410
 - view, 413
 - Hansson, David Heinemeier, 2, 60
 - `has_error`, 349–350
 - `has_many`
 - associating models, 627
 - associations, 634–637
 - `dependent: :destroy`, 642
 - microposts, 88–91
 - relationship model, 718, 720
 - user/relationship associations, 725–728
 - `has_many :through`, 730
 - `has_many_attached` method, 690–691
 - `has_secure_password`
 - adding to User model, 299–300
 - creating/authenticating user, 303–305
 - enforcing minimum password standards, 301–302
 - hashed passwords, 296–299
 - password validation, 489–490
 - testing login, 417–418
 - Hashes
 - constructors and, 186
 - hashed passwords, 296–299
 - symbols and, 178–183
 - Hashrocket (`=>`), 179–180
 - header
 - Home page elements, 205–206
 - partial with links, 242
 - partials for, 220–222
 - replacing default headgith a call to render, 225
 - testing application header, 247
 - Hello, world!
 - adding controller action, 28–31
 - creating first application, 13–17
 - exercises, 32–33
 - installing gems with Bundler, 17–22
 - `rails server` command, 22–27
 - `hello` action, 63, 107
 - Hello app, 3
 - `help` action
 - generated view for Help page, 117
 - generating controllers, 110
 - organizing functions, 116
 - in routes file, 113
 - setting routes for static page app, 114–116
 - `help` method, `StaticPagesController`, 192–195
 - Help page
 - adding titles for static pages, 132–134
 - customizing HTML, 118
 - generated view, 117
 - viewing with embedded Ruby title, 137
 - viewing with HTML structure removed, 140–141
 - Helper modules, Session helper, 393
 - Helpers
 - built-in, 154–155

- custom, 155–158
 - defined, 155
 - Gravatar helper specifying Users file size, 511
 - Heroku
 - CLI, 51–52
 - commands, 54–55
 - configuring email sent in production, 581–584
 - database recommendation, 103
 - deploying sign up page, 372
 - deploying to, 623
 - deploying too early, 107–108
 - deploying toy app, 63, 95–97
 - ENV variables, 711
 - error page, 96–97
 - exercises, 54
 - setup and deployment, 49–53
 - SSL use by default, 368
 - viewing app in production environment, 314
 - heroku, 52
 - heroku create, 52–53, 96
 - heroku help, 55
 - heroku run
 - creating/authenticating user, 305
 - deploying toy app, 97
 - hexdigest, profile page with Gravatar, 325
 - Hidden fields, using with password reset, 607–608
 - Highlighted lines, conventions in this book using, 58
 - home action
 - adding feed instance to, 673–674
 - adding micropost instance variable to, 665–666
 - generated view for Home page, 117
 - generating controllers, 110
 - organizing functions, 116–117
 - in routes file, 113
 - setting `root` route, 142–143
 - setting routes for static page app, 114–116
 - testing static pages, 126–127
 - home method, Static Pages Controller, 192–195
 - Home page
 - adding follower stats to, 741–742
 - adding micropost creation to, 663–666
 - adding status feed to, 674–675
 - adding titles for static pages, 132–135
 - customizing HTML, 118
 - with delete links, 683
 - elements, 203–207
 - error messages on, 666–670
 - with footer added, 224
 - generated view, 117
 - invalid micropost on, 677
 - with link to signup page, 207–208
 - micropost form, 676
 - proto-feed option on, 670–675
 - setting root route to, 142–143
 - setting to Home page, 143–144
 - with status feed and following count, 722
 - viewing with embedded Ruby title, 136–137
 - viewing with HTML structure removed, 140–141
 - HTML
 - account activation, 550, 554
 - color, 218–219
 - customizing for Home page, 118
 - for edit form, 479
 - for login form, 383
 - for password reset email, 602
 - for signup form, 335–339
 - structure of typical web page, 129–131
 - testing status feed HTML, 779–780
 - titles for static pages, 131–135
 - viewing Home, Help, and About pages with HTML structure removed, 140–141
 - HTML shim (shiv)
 - making old browser accessible, 204
 - partials for, 221
 - HTTP (Hypertext Transfer Protocol)
 - basic authentication using, 54
 - GET, POST, PATCH, and DELETE operations, 114–115
 - as stateless protocol, 376
- I**
- IAM (Identity and Access Management), 707–711

- id attribute
 - assigning to HTML elements, 205
 - creating user objects, 266
 - data model for microposts, 64–65
 - data model for users, 64
 - finding users, 268–269
 - following table, 720
- Identity and Access Management (IAM), 707–711
- if keyword, 165–166
- if-else, 340
- if-then, 460
- ImageMagick, for image processing, 702, 713
- Images, micropost
 - configuring AWS for image upload, 706–711
 - downloading fixture image for use in tests, 695–696
 - overview of, 688–689
 - production AWS, 711–714
 - resizing, 701–705
 - upload, 689–693
 - upload in production, 705–706
 - validation, 696–701
- Implicit return, of functions, 168
- include method, mixing modules into classes, 170
- Indentation, Cloud9, 8–9, 13
- Index, Micropost migration with, 630
- Index, of users
 - adding styling, 511
 - adding URL to users link, 512–513
 - helper for specifying Users file size, 511
 - index action, 510
 - index view, 510–511
 - Microposts controller, 83
 - overview of, 507–508
 - paginating users, 516–520
 - refactoring for compact views, 523–525
 - requiring logged-in user for, 509–510
 - sample users, 513–516
 - showing all users, 507
 - testing, 520–523
 - testing redirect, 508–509
 - Users controller, 76–78
 - viewing browser hits using MVC, 73–74
- index -1, ranges, 174
- index action
 - listing, 510
 - paginating users, 518
 - requiring logged-in user, 509–510
 - for showing all users, 507
 - verifying proper redirection, 508–509
- index page
 - with one user, 513
 - with pagination, 519–520
 - verifying proper redirection, 508
- index view, 510–511, 523
- Inheritance
 - class, 187–190
 - for controllers, 93–94
 - for models, 92–93
 - overview of, 91–92
 - for static pages, 116, 193–194
 - for String class, 187–188
- Initialization hash, creating user objects, 265
- initialize method, User class, 196–197
- Innovation, advantages of Rails, 2
- input tags, 337, 343
- inspect method, 182, 282
- Installing Git, 34–38
- Installing Rails, 9–13
- Instance methods, 186
- Instance of a class, 186
- Instance variables, 646–647, 665–666
- Instantiation, literal constructor for strings, 185
- integer data type, 64–65
- Integration, advantages of Rails, 1–2
- Integration tests
 - adding account activation to test user signup, 574–581
 - for micropost interface, 686
 - options, 121
 - password reset, 616–618, 620–621
 - testing links, 244–248
 - verifying user destroyed, 535–536
- Interpolation
 - exercise, 164

- single-quoted strings and, 163
- of strings (`#{ }`), 161–162
- Introduction to Rails
 - conventions used in this book, 56–58
 - creating first application, 13–17
 - deploying with Heroku, 49–55
 - deployment options, 49
 - developing application using `rails server` command, 22–27
 - development environment, 6–9
 - getting started, 2–3
 - Hello, World! 28–33
 - installing gems using Bundler, 17–22
 - installing Rails, 9–13
 - many advantages of Rails, 1–2
 - model-view-controller (MVC), 27–28
 - principal teaching method of this tutorial, 3–5
 - review, 55–56
 - technical sophistication and, 6
 - up and running with AWS Cloud9, 4–5
 - up and running with cloud IDE, 5
 - up and running with native OS setup, 5–6
 - version control with Git. *See* Git
- Invalid submission, testing signup page, 351–355
- `irb` (interactive Ruby) configuration, Rails console, 159
- `is password?` 442
- J**
- JavaScript embedded Ruby (`js.erb`), 764–765
- JavaScript libraries, 405–406
- `join` method
 - converting array to string, 173
 - making status feed, 780
- jQuery
 - checking file size with, 698–699
 - making available, 405
- `js.erb` (JavaScript embedded Ruby), 764–765
- K**
- Keys, hash, 178–183
- Key-value pairs, using in status feed where method, 775–776
- L**
- Lambda, “stabby lambda” syntax, 641
- Layout
 - adding Bootstrap CSS, 213–214
 - adding structure, 201–202
 - asset pipeline, 226–228
 - Bootstrap with CSS classes, 210–212
 - `bootstrap-sass` gem added to Gemfile, 212–213
 - changing layout links, 401–402
 - changing layout links for logged-in users, 404–405
 - commenting out embedded Ruby, 219
 - conclusion, 253–254
 - Contact page, 236–238
 - creating signup URL, 250–253
 - creating Users controller, 248–250
 - defining variables using Sass, 231–233
 - downloading images, 208–210
 - hiding all images, 220
 - Home page elements, 203–207
 - Home page with link to signup page, 207–208
 - links, 235–236
 - named routes, 242–244
 - nested elements in stylesheets, 229–231
 - nesting and variables used in converting SCSS file, 233–235
 - overview of, 201
 - partials used for HTML shim, 221
 - partials used for site footer, 222–225
 - partials used for site header, 221–222
 - partials used for stylesheets and header, 220–221
 - replacing default `head` with a call to `render`, 225
 - routes added to static pages, 238–241
 - Sass, 228–229
 - site navigation, 202–203
 - static pages, 135
 - styling site logo, 218–219
 - testing changes, 416–417
 - testing links, 244–248
 - typographic effect, 216–218

- universal styling on all pages, 214–216
 - user signup, 248
 - viewing application site layout, 139–140, 154–155
 - what we learned, 254
- Learn Enough All Access Bundle, 54
- Learn Enough Command Line to Be Dangerous*, 2, 14, 38, 162
- Learn Enough Dev Environment to be Dangerous*, 6
- Learn Enough Git to be Dangerous*, 34, 38
- Learn Enough Ruby to be Dangerous*, 2–3
- Learn Enough tutorials, 2–3, 6
- Length, validating email address, 278–280
- length** method
 - activating accounts, 567
 - arrays, 172
 - enforcing minimum password, 301–302
 - finding length of `User.all`, 269
 - strings, 164
- Libraries, importing JavaScript libraries, 405–406
- Links
 - adding URL to users link, 512–513
 - changing layout links, 401–402
 - changing layout links for `logged-in` users, 404–405
 - Contact page, 236–238
 - delete link added to micropost partial, 680–681
 - footer partial with, 243
 - header partial with, 242
 - layout links, 235–236
 - layout links for `logged_in` users, 404
 - micropost pagination, 654
 - named routes, 242–244
 - ”Next” link on Home page, 678
 - to password reset resource, 591
 - routes added to static pages, 238–241
 - to signup page, 207–208, 251–252
 - testing, 244–248
- Literal backslash (`\\`), 163
- Literal constructors
 - named constructors more explicit than, 185
 - for strings, 185
- Local webserver, allowing connections to, 23–27, 66
- `log` command, 37
- `log_in`
 - calling `log_out` only if logged in, 453–454
 - logging a user in, 394–396
 - logging in on signup, 422–425
 - remembering logged in user, 443–444
- `log_in_as` helper, 462–465
- `log_out`, 426, 453–454
- `logged_in`
 - authorization tests for Micropost controller, 659–661
 - helper method, 402–403
 - layout links for, 404
- Logged-in users
 - adding `store_location` to, 505–506
 - requiring, 491–494, 509–510
- Logging out
 - overview of, 425–429
 - from persistent sessions, 448–451
 - testing logging out, 452–453
- Login
 - analogy, 540
 - authorization, 491–494
 - changing layout links, 401–402
 - changing layout links for logged-in users, 404–405
 - commenting out authentication code, 420–421
 - conclusion, 429–430
 - `current_user` method, 396–401
 - `digest` method for use in fixtures, 417–418
 - entering Safari’s Responsive Design Mode, 407–409
 - finding and authenticating users, 383–385, 387–388
 - finding current user in a session, 399–400
 - form for, 380–383
 - hamburger menu, 409–414
 - importing JavaScript libraries, 405–406
 - `logged_in` helper method, 402–403
 - logging a user in, 393–396
 - logging in on signup, 422–425

- logging in test user, 494–495
- logging out, 425–429
- login page with “forgot password” link, 592
- making JQuery available, 405
- mobile styling, 406–407
- new session, 385
- overview of, 375
- rendering using flash message, 388–390
- session routes, 379–380
- sessions, 376
- Sessions controller, 376–378
- simplifying login code, 422
- simulating user session, 400
- test case with valid email and invalid password, 421
- testing layout changes, 416–417
- testing login behavior, 390–393
- testing user login with valid information, 419–420
- updating footer CSS, 414–416
- using fixture for testing user login, 418–419
- Login, advanced
 - forgetting users, 448–451
 - only logging out if logged in, 453–454
 - overview of, 431
 - remember me checkbox, 456–462
 - remember tests, 462–468
 - remember token and digest, 432–439
 - with remembering, 439–448
 - remembering me, overview, 431–432
 - review, 472–473
 - testing remember branch, 468–471
 - two subtle bugs, 451–456
- Logo, applying CSS to site logo, 218–219
- `log.out` method, 451–453
- Lowercase, downcasing email attribute, 292–293
- M**
- Mailer. *See* User mailer, email for account activation
- Mailer templates
 - account activation email, 549–554
 - password reset, 600–605
- Manifest files, 226–228
- `map` method, block, 176
- Mass assignment, 343
- `master` branch, signup page deployment, 367–368
- `maximum` parameter, length validation, 279–280
- MD5 hashing algorithm, 325
- Media query, 411
- Menus
 - creating dropdown menus, 404–406
 - creating with Bootstrap, 401–402
- Merge, GitHub, 47–48
- Message passing, 164–167
- `meta` tag, viewport, 407–408
- Metaprogramming, 566–567
- Methods. *See also* Functions
 - defining, 167–169
 - as functions in Ruby, 164
 - message passed to objects as, 164
 - method reference code, 545
- `Micropost` class, 93
- Micropost model
 - adding image to, 690–691
 - associations, 634–638
 - creating, 627–631
 - default scope, 638–641
 - `dependent: :destroy`, 641–643
 - validations, 631–634
 - `where` method, 671
- Microposts
 - access control, 658–661
 - adding `@microposts` instance variable, 646–647
 - adding creation to Home page, 663–666
 - adding image display to, 693–694
 - adding to sample data, 649–651
 - adding to `show` page, 647–648
 - associations, 634–638
 - conclusion, 714–716
 - configuring AWS for image upload, 706–711
 - constraining size of, 86–87
 - creating, 662–663
 - creating Micropost model, 627–631
 - data model for toy app, 64–65

- default scope, 638–641
 - dependent: :destroy**, 641–643
 - destroying, 680–684
 - error messages on Home page, 666–670
 - fixtures with generated microposts, 655–656
 - Home page form, 676
 - image resizing, 701–705
 - image upload, 689–693, 705–706
 - image validation, 696–701
 - images, 688–689
 - inheritance hierarchies, 91–95
 - invalid micropost on Home page, 677
 - manipulating, 657–658
 - microtour of, 81–85
 - ”Next” link on, 678
 - overview of, 627
 - pagination links, 654
 - parallel structure of Users and, 80
 - partial for showing single micropost, 645–646
 - production AWS, 711–714
 - proto-feed option on Home page, 670–675
 - rendering, 644–645
 - routing error, 679
 - sample, 649
 - showing, 643–644
 - styling, 651–653
 - testing, 684–688
 - testing profile page, 654–656
 - User has_many, 88–91
 - validating, 631–634
 - @microposts** instance variable, 646–647
 - MicropostsController** class, 94
 - micropost.user**, 89–90
 - Migration
 - for account activation, 544
 - creating micropost table in database, 629–630
 - creating Users resource, 66
 - creating/authenticating user, 305
 - database migrations, 257–263
 - enforcing email uniqueness, 290–291
 - generate model** creating migration file, 259
 - generating for remember digest, 433–434
 - for **password_digest** column, 297–298
 - in Rail, 257
 - undoing, 112–113
 - of User model to create **users** table, 259–261
 - mini_magick**, for image processing, 702–703
 - Minitest reporters, 147–148
 - Mixin facility, 314–315, 333
 - Mobile devices, styling tweaks, 406–412
 - Mockups, 202
 - Model file, User model, 263–264
 - Models
 - data models. *See* Data models
 - Micropost model. *See* Micropost model
 - Relationship model. *See* Relationship model
 - test options, 121
 - User model. *See* User model
 - model-view-controller. *See* MVC (model-view-controller) pattern
 - Modules
 - packaging functions, 393
 - packaging related materials, 170
 - SessionsHelper**, 402–403
 - Mutation, array, 172
 - MVC (model-view-controller) pattern
 - default data structure for data model, 257
 - Rails apps and, 27–28
 - Users resource, 73–80
- ## N
- name** attribute
 - creating user objects, 265
 - data model for users, 64
 - database migrations, 257–261
 - imposing constraints (validations) on, 271–272
 - signup failure page, 343
 - signup form HTML, 337
 - testing length validation, 278–280
 - validating presence of, 274–275
 - Name field, Git system setup, 34
 - @name** variable, **User** class, 195–197
 - Named constructors, 185–186
 - Named routes

- added to static pages, 240–241
 - using, 242–244
 - Naming conventions
 - function arguments, 168
 - literal strings, 161
 - migration files, 260
 - new work environment at AWS
 - Cloud9, 8–9
 - nano editor
 - Git system setup, 34–35
 - irb configuration, 159
 - nav, for navigation links, 205–206
 - Navigation. *See* Site navigation
 - Nesting
 - converting SCSS file, 233–235
 - nested elements in stylesheets, 229–231
 - nested hashes, 181, 386
 - new
 - constructors, 185–186
 - creating layout file, 129
 - first application, 13–17
 - initializing Git repository, 35
 - Microposts controller, 83
 - mixing string types and, 147
 - planning toy app, 60
 - running with specific version number, 15–17
 - Users controller, 76–77
 - New hotness problem, Rails not prone to, 2
 - New password resets form, 593–595
 - New user, 361–364
 - new_token method, 435
 - Newline (`\n`), 160, 162–163
 - ”Next” link, on Home page, 678
 - `nil` object
 - a subtle bug, 451–453
 - understanding, 165–166
 - `nil` values, 265–266
 - Not (!) operator, 165
- O**
- Objects
 - class inheritance and, 187–190
 - creating user, 264–268
 - error messages, 666–667
 - finding user, 268–269
 - instantiating classes to create, 185
 - message passing and, 164–167
 - updating user, 270–271
 - validity test for, 272–274
 - Operators, combining Booleans with, 165
 - Or (`||`) operator, 165
 - Or equals (`||=`) operator, 398
 - Output, conventions in this book for, 57
- P**
- Padding, universal styling on all pages, 215–216
 - Pagination
 - link to second page, 680
 - microposts, 646, 654
 - users index, 516–520
 - palindrome? method, 188, 190–192
 - Palindrome tester, 167, 169
 - Parameters, strong, 343–346
 - params hash
 - login form with “remember me” checkbox, 458–459
 - nested hashes, 386
 - signup failure page, 342–344
 - testing invalid signup submission, 353
 - Partials
 - adding delete link to micropost partial, 680–681
 - for editing users, 481–483
 - for follow/unfollow, 744
 - for footer with links, 243
 - for header with links, 242
 - for HTML shim, 221
 - for microposts, 665
 - showing single micropost, 645–646
 - for site footer, 222–225
 - for site header, 221–222
 - for stats, 740–744
 - for stylesheets and header, 220–221
 - for user info sidebar, 664–665
 - Password digest, 296
 - Password reset
 - conclusion, 624–625
 - create action, 596–599

- edit action, 607–609
 - email function in development, 599
 - email function in production, 621–624
 - email test, 605–606
 - expiration, 625–626
 - generating controller for, 590–593
 - mailer and templates, 600–605
 - new password, 593–596
 - overview of, 587–589
 - as resource, 590
 - testing, 615–621
 - update action, 610–615
 - updating errors for, 668
- password_digest attribute, 303–304
- password_digest column, 296–299
- password_digest:string, 298
- password_reset method
 - creating reset email, 600
 - generating user mailer, 549, 551–552
 - user mailer previews, 556–557
- password_reset_expired method, 613–614
- Passwords
 - account activation, 543
 - configuring email, 582–583
 - creating password digest with bcrypt, 416–417
 - expiration of, 625–626
 - resetting, 607–610
 - setting length of time for, 35–36
 - signup form with fields for, 337–338
 - storing user id in cookies, 441
 - test case with valid email and invalid password, 421
 - two users can have same, 434
 - validating, 489–491
- Passwords, secure
 - creating and authenticating user, 303–305
 - enforcing minimum standards, 301–302
 - has_secure_password added to User model, 299–300
 - hashed passwords, 296–299
 - overview of, 295–296
 - reset, 540
- PATCH operation, HTTP, 114–115
- PATCH requests
 - account activation, 541
 - adding Users resource to routes file, 319
 - issuing, 485
 - routing to update action, 496
- Permanent cookies
 - creating, 432, 444
 - implementing "remember me," 431
 - storing remember token as, 439–440
 - vulnerability to session hijacking, 394
- permanent method, 441
- Persistence
 - creating User data model, 257–263
 - with database for long-term data storage, 257
- Persistent sessions
 - logging out from, 448–451
 - remember token and digest, 432–439
 - storing id and remember token as cookies via, 439–448
 - testing remember branch, 469–470
- pg gem, Heroku deployment setup, 49–51
- pluralize text helper, signup error messages, 349
- Plus (+) operator, 161
- post method, signup form, 338
- POST operation, HTTP, 114–115
- POST requests
 - adding Users resource to routes file, 319
 - signup failure page, 339–340
 - testing invalid signup submission, 352–353
- PostgreSQL database, 49–50
- Preprocessor engines, 228
- Presence validation
 - email addresses, 274–278
 - ensuring non-blank passwords, 302
- Previews, account activation email, 554–558
- Printing strings, to screen, 162
- private keyword
 - activation token callback, 545
 - strong parameters, 344–345
- Proc (procedure), "stabby lambda" syntax, 641

- Production environment
 - sending email in, 581–584
 - signup page deployment, 367–373
 - types of Rails environments, 159
 - viewing app in, 313–314
 - Professional-grade deployment, signup page, 367–373
 - Profile pages
 - adding forms and stats, 746–748
 - current user**, 718
 - with microposts, 644, 647, 652–653
 - testing, 654–656
 - with unstyled microposts, 650
 - Proto-feed
 - feed** options, 670–673
 - on Home page, 670–675
 - with micropost delete links, 681
 - Push
 - deploying toy app, 63, 95–96
 - GitHub, 48
 - push method, 172
 - puts method, 162
- Q**
- Query parameter, 553
- R**
- Rails console. *See* Console, Rails rails test:models, validity test, 274
 - Ranges, 173–174, 175–178
 - README file
 - branches, 43–44
 - commit** message, 45–47
 - editing, 44–45
 - Git rendering, 41–43
 - merge, 47–48
 - pushing changes up to GitHub, 48
 - setting up static page app, 104–105
 - :RecordNotFound** exception, finding users, 268
 - redirect_back_or** method, 504–506
 - redirect_to** method, 355–358
 - Refactoring
 - account activation testing and, 574–581
 - layouts and embedded Ruby, 135
 - users index for compact views, 523–525
 - when to test, 120
 - references**, Micropost model, 629
 - Regression, testing to prevent, 120
 - Regular expressions (regex), email formats, 283–285
 - Relationship model
 - adding **followers** association, 733–735
 - adding **following** association, 730–732
 - associations between users and relationships, 725–728
 - data model for, 719–725
 - overview of, 718–719
 - testing **followers** association, 735
 - testing **following** utility methods, 732–733
 - validating relationships, 728–730
 - Relationships controller
 - responding to Ajax requests in Relationships controller, 763
 - tests for relationships, 757–759
 - relationships** table, 724–725
 - reload**, user objects, 270
 - remember**
 - activation token callback, 546
 - associating remember token with user, 436–437
 - remembering logged in user, 436–437
 - testing remember branch, 469
 - remember** helper, 443–444
 - Remember me
 - checkbox, 456–462
 - forgetting users, 448–451
 - login with remembering, 439–448
 - remember token and digest, 432–439
 - review, 472–473
 - testing checkbox, 462–468
 - testing remember branch, 468–471
 - two subtle bugs, 451–456
 - Remember tokens
 - account activation data model, 543
 - associating with user, 436–437
 - creating persistent sessions, 432–433
 - generating, 434–435
 - testing “remember me” checkbox, 465–467

- remember_digest**
 - generalizing `authenticated?` method, 566
 - generating migration for remember digest, 433–434
 - storing user id in cookies, 441–443
- remember_token**
 - analogy, 540
 - associating remember token with user, 436–437
 - exercises, 437–439
 - storing user id in cookies, 441
 - testing remember me checkbox, 465
- render**
 - flash message for failed login, 389
 - replacing default `head` with a call to `render`, 225
 - signup error messages, 347
 - signup failure page, 340
- Rendering**
 - following and followers**, 751–755
 - microposts, 644–645
- Repositories**
 - creating for static page app, 106
 - optimizing/sharing, 39–42
 - setting up, 35–38
- REpresentational State Transfer**. *See* **REST** (REpresentational State Transfer)
- request.referrer** method, 682–683
- Resizing images**, 701–705
- Resources**
 - create** action, 596–599
 - creating new password reset, 593–596
 - generating controller for password reset, 590–593
 - password reset, 590
 - password reset as, 590
 - REST represents data as, 317
 - user microposts. *See* **Microposts**
- resources :microposts**, 81–82
- resources :users**
 - adding to `routes.rb` file, 318
 - routing rule, 66, 74–75
 - signup failure page, 339
- response.body**, count assertion using, 656
- Responsive Design Mode (Safari)**, 407–409
- REST (REpresentational State Transfer)**
 - account activation using REST URL, 541
 - collection of static pages, 116
 - controllers implementing, 76–78
 - destroy** action, 525–526
 - Micropost controller actions, 83–84
 - representing data as resources, 317
 - Sessions controller actions, 376–378
 - treating password reset as a resource, 590
- RESTful routes**
 - adding **Users** resource to routes file, 317–319
 - for **following** and **followers**, 739
 - provided by **Microposts** resource, 82–83, 658
 - provided by **Password Resets** resource, 591
- return** keyword
 - defining methods, 168
 - updating `authenticated?` to handle nonexistent digest, 455
- rf** flag, 38
- RJS (Ruby JavaScript)**, 765
- Rollback**
 - migrations, 263
 - starting console in sandbox for, 264
- root** route
 - adding for users, 75
 - setting for app deployment, 63
 - setting for static page app, 107–108, 142–143
 - setting to Home page, 143–144
 - testing, 144–145
- Routes**
 - added to static pages, 238–241
 - adding for **Contact** page, 237–238
 - for **home** and **help** actions, 114–116
 - login sessions, 379–380
 - mapping for site links, 236
 - micropost routing error, 679
 - named routes, 242–244
 - RESTful routes, 658
 - setting **root** route for Home page, 143–144
 - setting **root** route for static page app, 107–108, 142–143

- for signup page, 250–251
 - testing `root` route, 144–145
 - testing static pages, 125–128
 - for user relationships, 744–745
- Routes file, 74–75
- `routes.rb` file
 - adding `resources :users` to, 317–318
 - signup failure page, 339
- Rubular regular expression editor, 284
- Ruby, Rails subset
 - arrays and ranges, 170–174
 - blocks, 175–178
 - built-in helpers, 154–155
 - classes. *See* Classes
 - conclusion, 198
 - custom helpers, 155–156
 - CVS revisited, 183–185
 - hashes and symbols, 178–183
 - method definitions, 167–169
 - motivation, 153–158
 - objects and message passing, 164–167
 - review, 198–199
 - strings and methods, 159–164
 - `title_helper`, 169–170
- Ruby JavaScript (RJS), 765
- S**
- S3
 - configuring production environment for, 712
 - creating S3 bucket, 710–711
- Safari’s Responsive Design Mode, 407–409
- Sandbox, creating user objects, 270–271
- Sass (Syntactically Awesome Stylesheets)
 - defining variables, 231–233
 - filename extension, 228
 - nested elements in stylesheets, 229–231
 - nesting and variables used in converting SCSS file, 233–235
 - overview of, 228–229
- `save` method, 265–266
- `scaffold` command, 65–66
- Scaffolding
 - code for Microposts resource, 81–82
 - code for Users resource, 65–66
 - overview of, 60
 - pages for User resource, 68–71
- Scalability, advantages of Rails, 2
- `schema.rb` file, 263
- Scope, micropost default, 638–641
- SCSS. *See also* CSS (Cascading Style Sheets)
 - adding styling to users index, 511
 - for Home page with follow stats, 742–744
 - nesting and variables used in converting, 233–235
 - Sass support for, 228–229
 - styling microposts, 651–653
 - styling user show page, 328–329
- `section` element, Home page, 205–206
- Secure Sockets Layer (SSL), 367–368
- SecureRandom module, 434
- Seed users, activating by default, 547
- `self` keyword
 - class inheritance, 189
 - email uniqueness validation, 292–293
- `self.email`, 436–437
- `send`, activating accounts, 566–567
- `send_activation_email`, 577
- SendGrid, sending email in production, 581–584, 621–623
- `server` command
 - allowing connections to local webserver, 23–24
 - default Rails page served by, 28
 - routes and, 114
 - running in separate tab, 24–27
- Servers
 - `byebug` prompt in, 323
 - password reset email in server log, 602
- Session helper module, 393, 504
- Session hijacking attacks, 394
- Sessions, for basic login
 - overview of, 376
 - routes, 379–380
 - Sessions controller, 376–378
- Sessions controller, 376–378, 384–385
- `SessionsHelper` module, 402–403
- Setter methods, `User` class, 195

- Settings link, adding URL to "Settings" link in site layout, 480–481
- setup method
 - adding page titles, 134
 - validity test, 272–273
- shared/ directory, 347–348
- Shim (shiv)
 - making old browser accessible, 204
 - partials for, 221
- Shopify, 2
- Shortcuts, Rails, 110–111
- Shovel operator (<<), arrays, 173
- show action
 - adding @microposts instance variable to, 646–647
 - adding sidebar to user, 328
 - and GET requests, 317–318, 320–321
 - Microposts controller, 83
 - test for valid signup, 365–366
 - Users controller, 76–77
 - Users controller with debugger removed, 323–324
- show pages
 - adding microposts to, 647–648
 - showing microposts, 643
 - showing users, 507
- show_follow view, 751–755
- Sidebar
 - adding initial version to user profile page, 327–329
 - for micropost count test, 688–689
 - partial for user info sidebar, 664–665
 - SCSS for Home page with follow stats, 742–744
- Signup. *See also* Profile pages
 - creating signup URL, 250–253
 - creating Users controller, 248–250
 - debug, 311
 - debug output using CSS rules, 314–315
 - debugger, 322–324
 - form for, 331–339
 - Gravatar image and sidebar, 324–331
 - Home page with link to signup page, 207–208
 - logging in on signup, 422–425
 - overview of, 248, 309
 - professional-grade deployment, 367–373
 - Rails environments, 313–314
 - rendering of user signup errors, 667
 - review, 373–374
 - showing users, 310–311
 - testing account activation, 575–578
 - users resource for, 316–322
- Signup failure
 - error messages, 346–351
 - strong parameters, 343–346
 - test for invalid submission, 351–355
 - a working form, 340–343
- Signup pages
 - link to, 251–252
 - routes for, 250–251
- Signup success
 - finished signup form, 355–358
 - first signup, 361–364
 - the flash, 358–361
 - overview of, 355
 - test for valid submission, 364–367
- Signup URL, 250–253
- signup view, 482–483
- Simulating user session, in console, 400
- Single-quoted strings ('), 163–164
- Site footer, partials for, 222–225
- Site header, partials for, 221–222
- Site layout, adding URL to "Settings" link in, 480–481
- Site logo, applying CSS to, 218–219
- Site navigation
 - downloading images, 208–210
 - Home page elements, 203–207
 - Home page with link to signup page, 207–208
 - overview of, 202–203
- Source code, version control. *See* Git
- Spacing, universal styling on all pages, 215–216
- split method, 171
- SQL injection attacks, 672
- SQL statements

- escaping variables, 672
 - `user.save` and, 266
- Square brackets (`[]`), arrays, 171
- SSL (Secure Sockets Layer), 367–368
- ”stabby lambda” syntax, 641
- Staging area, setting up Git repository, 37
- Standards, password, 301–302
- Stateless protocol, HTTP as, 376
- Static pages
 - adding `hello` action, 107
 - adding minitest reporters, 147–148
 - adding titles, 131–135
 - advanced testing, 146–147
 - allowing connection to local webserver, 106–107
 - automating tests, 148–151
 - classes, functions, inheritance, 116
 - code for Contact page, 142
 - conclusion, 145–146
 - controller for, 109–112
 - creating repository for, 106
 - customizing, 118
 - developing sample app, 108–109
 - green (passed tests), 125–128
 - inheritance hierarchy for, 193–194
 - layouts and embedded Ruby, 135
 - making slightly dynamic, 128–129
 - overview of, 101
 - red (failed tests), 123–125
 - routes added to, 238–241
 - routes for `home` and `help` actions, 114–116
 - running first test, 122–123
 - setting `root` route, 107–108, 142–144
 - setting up gems, 101–104
 - setting up README file, 104–105
 - testing options, 118–121
 - testing `root` route, 144–145
 - testing titles, 129–131
 - undoing code generation, 112–113
 - verifying test suite passes, 141–142
 - viewing About page, 137–139, 140–141
 - viewing application site layout, 139–140
 - viewing Help page, 137, 140–141
 - viewing Home page, 136–137, 140–141
 - views, 117
- `StaticPagesController`, 192–195
- Stats
 - adding forms and stats to user profile, 746–748
 - partial for follower stats, 740–744
 - sample of following data, 738
- Status feed
 - adding feed instance to home action, 673–674
 - adding to Home page, 674–675
 - for following users, 768–770
 - Home page with status feed and following count, 722
 - implementing, 771–774
 - `join` method for making, 780
 - overview of, 657
 - proto-feed option, 670–673
 - subselects in implementing, 774–779
 - testing, 771
 - testing HTML, 779–780
- `store_location` method, 504–506
- String literals (literal strings), 161
- `string` type, 64
- Strings
 - constructors and, 185–186
 - converting any object to, 165
 - dividing into arrays with `split`, 171
 - exercises, 163–164
 - exploring, 160–162
 - as hash keys, 178–179
 - inheritance hierarchy for String class, 187–188
 - and methods, 159–160
 - objects, message passing and, 164–167
 - printing, 162
 - single-quoted, 162–163
 - `string` type, 64
- Stub view, 252, 319–320
- `stylesheet_link_tag`, 154–155
- Stylesheets. *See* CSS (Cascading Style Sheets); Sass (Syntactically Awesome Stylesheets)
- Subselects, implementing status feed, 774–779

- `:success` key, 358–360
 - `superclass` method, 187–188
 - Symbols, hashes using, 179–183
 - Symbol-to-proc, 176
 - Synonyms, commonly accessed arrays, 171–172
 - Syntactically Awesome Stylesheets. *See* Sass (Syntactically Awesome Stylesheets)
- T**
- Tables. *See also* Databases
 - creating micropost table in database, 629–630
 - following table, 720
 - full-table scan, 290
 - relationships table, 724–725
 - `user_id` column in microposts table, 88–89
 - `users` table, 258–261
 - `users` table, 303
 - TDD (test-driven development)
 - successful user edits, 486–488
 - validity test, 272–274
 - when to use, 119–120
 - writing failing tests (Red) first, 123–125
 - Technical sophistication, developing, 6
 - Templates
 - mailer for account activation, 549–554
 - password reset emails, 600–601
 - previews of account activation email, 554–558
 - for sidebar micropost count test, 688–689
 - testing flash message, 366–367
 - for `update_columns` in account activation email, 619–620
 - Ternary operator
 - login form with “remember me” checkbox, 460
 - understanding, 461
 - `test` command
 - automating tests using *Guard*, 148–149
 - beginning test cycle, 123
 - Green (passed tests), 125–127
 - Red (failed tests), 123–125
 - testing dynamic ERb titles, 137, 141
 - testing suite before deploying, 146
 - testing titles, 131
 - Test environment, 159, 313
 - Test suites, 119
 - `test_helper`, 157–158, 465
 - Test-driven development. *See* TDD (test-driven development)
 - Tests/testing
 - account activation, 574–581
 - account activation email, 558–561
 - advanced setup, 146–147
 - automating tests, 148–151
 - Contact page, 236–238
 - conventions used in this book for, 58
 - dependability of Rails and, 2
 - destroying users, 533–536
 - editing by wrong user, 498
 - flash behavior, 390–393
 - following users, 765–768
 - following/follower, 755–757
 - friendly forwarding, 503–504
 - green (passed tests), 125–128
 - index redirection, 508–509
 - layout changes, 416–417
 - layout links, 244–248
 - login, 418–421
 - micropost profile page, 654–656
 - micropost validation, 632–633
 - microposts, 684–688
 - minitest reporters, 147–148
 - options, 118–121
 - password reset, 615–621
 - protection of edit and update, 495–497
 - red (failed tests), 123–125
 - remember branch, 468–471
 - remember me checkbox, 462–468
 - `root` route, 144–145
 - running first test, 122–123
 - signup form, 364–367
 - signup page, 351–355
 - status feed, 779–780
 - temporarily commenting out failing, 562–563
 - titles for static pages, 129–135
 - user edits, 484–486, 488
 - user page, 249–250

- users index, 520–523
- validity, 272–274
- verifying test suite passes, 141–142
- Text, signup form with fields for, 337–338
- Text editor
 - cloud IDE, 7
 - Git system setup, 34–35
 - installing gems for new Rails app, 17–22
 - planning toy app, 61
- text type, 65
- Text view, 550, 553
- time_ago_in_words, 653
- Timestamp, 260
- title_helper, 169–170
- Titles
 - adding for static pages, 131–135
 - testing for static pages, 129–131
 - viewing About page with embedded Ruby title, 137–139
 - viewing Help page with embedded Ruby title, 137
 - viewing Home page with embedded Ruby title, 136–137
- Token callback, activation, 545–547
- token method, 546
- touch command, Unix, 127
- Toy app
 - beginning of, 3–4
 - deploying, 95–98
 - overview of, 59
 - planning, 60–64
 - review, 99
 - strengths and weaknesses of, 98–99
 - toy model for microposts, 64–65
 - toy model for users, 64
- true value
 - Booleans, 165
 - objects and, 167
- t.timestamps command, 260
- U**
- Ubuntu server, 8, 10
- Undo, code generation, 112–113
- unfollow method, 731–733, 744
- Unicode, character sets, 139
- :uniqueness, validates method, 286
- uniqueness: true, 287
- Uniqueness validation, email
 - alternative callback implementation, 295
 - database indices and, 290
 - of email addresses, 286–295
 - ensuring by downcasing email attribute, 292–293
 - migration for enforcing, 291
 - overview of, 286
 - restoring original test for, 293–294
 - testing default user fixtures, 291
 - testing empty fixtures files, 292
 - testing for case-insensitivity, 287–290
 - testing for email downcasing, 294–295
 - testing for rejection of duplicate addresses, 286–287
- Unix
 - common commands, 15
 - touch command, 127
 - using command line, 14
- unless keyword, 166
- uppercase method, 288
- Update
 - password reset errors, 668
 - rendering of user signup errors, 667
 - user editing errors, 668–670
 - user objects, 270–271
 - users for account activation email, 561–565
- update action
 - account activation, 541
 - Microposts controller, 83
 - password reset, 610–615
 - testing protection of, 495–497
 - in unsuccessful user edits, 483–484
 - user update, 488–489
 - Users controller, 76–77
- update method, 270–271
- Update pages, 499
- Update users
 - adding URL to “Settings” link in site layout, 480–481
 - editing, 476–480

- overview of, 475
 - partials use for editing, 481–483
 - password validation, 489–491
 - successful edits, 486–488
 - testing successful edits, 488
 - testing unsuccessful edits, 484–486
 - unsuccessful edits, 483–484
 - update action, 488–489
- update_attribute
 - associating remember token with user, 436–437
 - migration for account activation, 546
 - updating user objects, 271
 - user profile page with custom Gravatar, 326–327
- update_columns, 579–580, 619–620
- updated_at, 265–268
- Uploading images
 - configuring AWS for, 706
 - microposts, 689–693
 - in production, 705–706
- URLs
 - adding to “Settings” link in site layout, 480–481
 - adding to users link, 512–513
 - email previews as, 554–558
 - mailing account activation link, 552–553
 - mapping for site links, 236
 - mapping to actions in Microposts controller, 82
 - signup URL, 250–253
 - updating users create action, 561–565
 - user profile page with Gravatar, 324–325
 - Users resource pages and, 68–69
 - users viewing browser hits using MVC, 73–74
- urlsafe_base64 method, SecureRandom module, 434
- User class, 93, 195–198
- User fixtures. *See* Fixtures
- User info sidebar, 664–665
- User mailer, email for account activation
 - adding account activation to user signup, 561–552
 - previews, 555–556
 - sample from server log, 564–565
 - template, 549–554
 - tests, 559–561
- User microposts. *See* Microposts
- User mismatch, 685–686
- User model
 - adding account activation attributes, 544
 - adding activation account code, 546–547
 - adding password reset methods to User model, 597–598
 - creating user objects, 264–268
 - data structure to capture sign up data, 256–257
 - database migrations, 257–263
 - feed method in, 671
 - finding user objects, 268–269
 - generalized authenticated? method, 568–569
 - model file, 263–264
 - overview of, 255
 - own authentication system, 255–256
 - review, 305–307
 - secure passwords, 295–305
 - for toy app, 64
 - updating user objects, 270–271
- User profile. *See* Profile pages
- User signup. *See* Signup
- @user variable
 - index action for toy application, 78
 - showing user information in Stub view, 319–320
 - signup form HTML, 338
 - testing authenticated? with nonexistent digest, 454–455
 - testing rejection of duplicate email addresses, 287
 - testing remember me checkbox, 465–466
 - user signup form, 332–333
 - users viewing browser hits using MVC, 73–74

- validating presence, 274
- validity test, 272–273
- `user_id`
 - associating each micropost with particular, 65
 - encrypting cookies, 440–441
 - Micropost migration with, 630
 - microposts table, 88–89
 - remembering user in login, 452
 - storing in cookies, 440
 - validating microposts, 632
- `user_name`, 582–583
- `user_params`
 - call to update users, 484
 - password reset, 613
 - strong parameters, 344–345
- `user_url`, 357–358
- `!user.activated?` 571–572
- `User.all`
 - finding users, 269
 - index action for toy application, 78
 - users viewing browser hits using MVC, 73–74
- `user.authenticated?`
 - method, 540
- `User.count` method, 365
- `User.digest` method
 - account activation, 540
 - associating remember token with user, 436–438
- `User.first`, 89–90
- `user.forget` method
 - logging out from persistent session, 448–451
 - a subtle bug, 451
- `UserMailer.account_activation`, 556
- `User.new` (user object)
 - creating user objects, 265
 - signup failure page, 339, 341, 343–344
 - signup form page, 333
 - testing invalid signup submission, 353
- `User.new_token` method
 - account activation, 540
 - associating remember token with user, 436–438
- `user.remember` method
 - associates remember token with user, 436
 - logging in and remembering user, 443–444
 - undoing via `user.forget`, 448–451
- Users
 - activating seed and fixture, 547–548
 - adding password reset methods, 597–598
 - adding `store_location` to logged-in user, 505–506
 - adding styling to users index, 511
 - adding URL to “Settings” link in site layout, 480–481
 - adding URL to users link, 512–513
 - administrative, 526–530
 - associating micropost with, 65
 - associations, 725–728
 - authorization, 491
 - conclusion, 536–538
 - `create` action used with friendly forwarding, 506–507
 - `current_user?` method, 500–502
 - deleting, 525–526
 - `destroy` action, 530–533
 - editing, 476–480
 - finding and authenticating, 383–385, 387–388
 - following. *See* Following users
 - friendly forwarding, 502–504
 - granting administrative access, 709
 - `index` action, 510
 - `index` view, 510–511
 - logging in, 393–396, 494–495
 - paginating, 516–520
 - partials for editing, 481–483
 - password validation, 489–491
 - profile page, 718
 - protecting edit/update pages, 499
 - refactoring users index for compact views, 523–525
 - rendering signup errors, 667
 - requiring logged-in user for `index` action, 509–510
 - requiring login, 491–494

- requiring right user, 497–498
 - sample users, 513–516
 - showing all, 507
 - specifying Users file size, 511
 - successful edits, 486–488
 - testing destroying, 533–536
 - testing editing by wrong user, 498
 - testing edits, 483–486, 488
 - testing friendly forwarding, 503–504
 - testing `index` action redirect, 508–509
 - testing login, 418–420
 - testing protection of `edit` and `update`, 495–497
 - testing users index, 520–523
 - `update` action, 488–489
 - updating, 475
 - updating `create` action for account
 - activation email, 561–565
 - user `index`, 507–508
 - validation. *See* Validation
 - Users controller
 - creating, 248–250
 - with debugger, 322–323
 - with debugger removed, 323–324
 - updating, 251
 - `UserController` class, 94
 - Users index. *See* Index, of users
 - Users resource
 - adding presence validations to User model, 92–93
 - associations between microposts and users, 88–91
 - destroying user, 71–72
 - edit page, 69–70
 - exercises, 71–73
 - index page, 68–72
 - inheritance structures for, 92–98
 - MVC in action, 73–79
 - new page, 69–71
 - overview of, 65–67
 - parallel structure of Microposts and, 80
 - show page, 69–70
 - tour of, 68–73
 - weaknesses of, 80
 - `:users` symbol, 74
 - `users` table, 258–261
 - `users` table, 303
 - `users_signup` file, 352–355
 - `@user.save`, 340
 - `UserController` class, 94
 - Utf-8, character sets, 139
- ## V
- `-v` command, for version number, 11, 26
 - `valid?` method
 - creating user objects, 265
 - validating `user` variable, 276
 - validity test, 273
 - Valid submission
 - frozen page on signup form in, 355–358
 - of signup page, 364–367
 - `VALID_EMAIL_REGEX`, 283
 - `validates` method, 275
 - Validation
 - adding gem for, 696
 - creating user objects, 265
 - formats, 280–286
 - images, 696–701
 - length, 278–280
 - micropost, 631–634
 - micropost presence, 90–91
 - micropost rules for length, 76–77
 - overview of, 271–272
 - presence, 91–92, 274–278
 - relationships, 728–730
 - uniqueness, 286–295
 - validity test, 272–274
 - Validity test, 272–274
 - `value`, cookies contain, 439
 - Variables
 - adding instance variable to `home` action, 665–666
 - adding instance variable to `show` action, 646–647
 - defining using Sass, 231–233
 - escaping in SQL statements, 672
 - Heroku ENV variables, 711
 - use in converting SCSS file, 233–235

- Version control
 - for all Gemfiles in this book, 21
 - deploying to Heroku and, 372–373
 - exercises, 26
 - with Git. *See* Git
 - installing gems with Bundler, 20–22
 - installing Rails with, 11
 - planning toy app, 60
- Viewport meta tag, 407–408
- Views
 - adding for Contact page, 238
 - creating set of, 108
 - creating Users controller, 249
 - edit view, 478, 482
 - index view, 510–511, 523
 - model-view-controller (MVC), 27–28, 73–80
 - show_follow view, 751–755
 - signup view, 482–483
 - static pages, 117
 - stub view, 252, 319–320
 - text view, 550, 553
- W**
- Warning message, for not-yet activated user, 573
- Web interface, for following users
 - adding following and followers actions to User Controller, 739
 - adding forms and stats to user profile, 746–748
 - Ajax buttons for follow/unfollow, 759–761
 - Ajax form for follow/unfollow, 761–763
 - creating/destroying following relationship with JavaScript, 765
 - degradation of form submission, 763–765
 - following and followers actions, 750–751
 - forms for follow/unfollow, 745–746
 - overview of, 736
 - pages for followed and followers, 748–750
 - partial for follow/unfollow, 744
 - rendering following and followers, 751–755
 - responding to Ajax requests in Relationships controller, 763
 - routes for user relationships, 744–745
 - sample data, 736–738
 - standard buttons for follow/unfollow, 757–759
 - stats, 738, 740–744
 - testing following/follower, 755–757
 - tests, 765–768
- Webpack
 - adding jQuery to, 405
 - asset pipeline, 225–226
- Webserver, allowing static page app to connect to, 106–107
- where method
 - key-value pairs in status feed where method, 775–776
 - Micropost model, 671
- will_paginate
 - correcting invalid micropost on Home page, 677
 - paginating microposts, 646
 - paginating users, 518–519
- Wireframes. *See* Mockups
- without production option, 62
- X**
- XSS (cross-site scripting attacks), 140
- Y**
- YAML (YAML Ain't Markup Language), 315
- Yarn, 225–226
- Z**
- Zero-offset, Ruby arrays as, 171