LEARN RUBY ON RAILS

DANIEL KEHOE

Learn Ruby on Rails: Book One

Version 3.0.0, 14 January 2016

Daniel Kehoe

Contents

Chapter 1

Free Offer

I want you to have Book Two in this series for free.

You are reading Book One, which introduces basic concepts and gives you the background you need to succeed. In Book Two, you'll build a useful web application, for hands-on learning. The two books go together, which is why I want you to have both books.

I've created an online version of both books at the website learn-rails.com. You'll also find PDFs available for download. Look for the link "Free Online Edition" when you visit the site:

- learn-rails.com

Use the invitation code:

- SOFTCOVER300B1

I'll ask you to provide your email address when you sign up to get free access to Book Two. I work hard to keep the books up to date, incorporating improvements and fixing errors as readers report issues. I update the online edition of

the books often and I send email to notify of updates. If you bought the book from Amazon or another retailer, email is the only way to learn about updates.

I'll also let you know about the Capstone Rails Tutorials, which you'll want to read after you finish this book series. I promise there is no better way to learn Rails.

Chapter 2

Introduction

Welcome. This is a first step on your path to learn Ruby on Rails.

This book contains the background that's missing from other tutorials. Here you'll learn key concepts so you'll have a solid foundation for continued study. Whether you choose to continue with another book in this series, a video course, or a code school, everything will make sense when you start here.

You can read this book anywhere, at your leisure, on your phone or tablet. Use this book to gain background understanding when you are not at your computer. With Book Two, the next in the series, you'll need a computer at hand so you can build your first web application.

In Book Two, you'll build a working web application so you'll gain hands-on experience. Along the way, you'll practice techniques used by professional Rails developers. And I'll help you'll understand why Rails is a popular choice for web development.

You can start with Book Two before finishing this book if you're eager to get started building your first application. In fact, I recommend it, because the hands-on learning in Book Two reinforces the concepts you learn in this book.

Is It for You?

If you've built simple websites using HTML, you'll quickly progress to building websites with Rails. Or, if you have experience in a language such as PHP or Java, you'll make the jump to the Rails framework. But I promise you don't need to be a programmer to succeed with this book or the next. You'll be surprised how quickly you become familiar with the Unix command line interface and the Ruby programming language even if you've never tried programming before.

My books are ideal if you are:

- a student
- a startup founder
- making a career change

Some readers are happy to learn to communicate with developers they work with. If you are starting a business, and hiring developers, or working alongside developers as a manager or developer, this book will help you talk with developers. However, the true purpose of my book is to help you become you a Rails developer yourself. I want to help you launch a startup or begin a new career.

What To Expect

There is deep satisfaction in building an application and making it run. With this book and the next, I'll give you everything you need to build a real-world Rails application. More importantly, I'll explain everything you build, so you understand how it works.

When you've completed this tutorial, you will be ready for more advanced self-study, including the Capstone Rails Tutorials, textbook introductions to Rails,

or workshops and code camps that provide intensive training in Ruby on Rails. Other curriculums often skip the basics. With this tutorial you'll have a solid grounding in key concepts. You won't feel overwhelmed or frustrated as you continue your studies. I think you'll also have fun!

This book is good preparation for:

- textbooks such as Michael Hartl's Ruby on Rails Tutorial
- introductory workshops from RailsBridge or Rails Girls
- intensive training with immersive code camps
- Capstone Rails Tutorials from the RailsApps Project

We are blessed with many textbooks, workshops, and classroom programs that teach Ruby on Rails. I believe this book is unique in covering the basics while introducing the tools and techniques of professional Rails development.

What's in Book One

Book One is a self-help book that can change your life, though here you won't find any inspirational quotes or magical thinking.

I explain the culture and practices of the Rails community. I introduce the basic concepts you'll need to understand web application development. You'll learn how to be a successful learner and how to get help when you need it. I also provide a plan for study so you can learn more when you need it. There's so much to learn, it helps to have a map so you know where to go next.

Programming can be frustrating and Rails isn't easy for beginners. The chapter, "Rails Challenges," describes many of the problems learners encounter. It's natural to get discouraged so take a look when you begin to feel overwhelmed.

Two chapters, "Crossing the Chasm", and "Level Up", will help you after you put the book down. Many learners feel stranded if their only experience is

step-by-step tutorials. These chapters are designed to give you a strategy for building an application on your own.

What's in Book Two

You'll start coding in Book Two. It's a hands-on tutorial that will lead you through the code needed to build a real-world web application. Don't skip around in Book Two. The tutorial is designed to unfold in steps, one section leading to another, until you reach the "Testing" chapter.

You can complete Book Two in one long weekend, though it will take concentration and stamina. If you work through the book over a longer timespan, try to set aside uninterrupted blocks of two hours or more for reading and coding, as it takes time to focus and concentrate.

Feel free to start Book Two before you finish this book. Begin coding with Book Two while you get background knowledge from this book at your leisure.

A Warning About Links

My books are densely packed with links to background reading. If you click every link, you'll be a well-informed student, but you may never finish the book! It's up to you to master your curiosity. Follow the links only when you want to dive deeper.

What Comes Next

The best way to learn is by doing; when it comes to code, that means building applications. Hands-on learning with actual Rails applications is the key to absorbing and retaining knowledge.

After you read this book, you'll be able to work with the example applications from the RailsApps Project. The project provides open source example ap-

plications for Rails developers, for free. Each application is accompanied by a tutorial in the Capstone Rails Tutorials series, so there's no mystery code. Each application can be generated in a few minutes with the Rails Composer tool, which professional developers use to create starter applications.

The RailsApps Project is solely supported by sales of the books and the Capstone Rails Tutorials. If you purchase the Capstone Rails Tutorials, you'll keep the project going. And you'll have my sincere appreciation for your support.

Versions

Book One is relevant and useful for any version of Rails. Book Two requires a specific version of Rails (the newest at the time it was revised) and shows how to install the latest version of Rails.

Staying In Touch

If you obtained this book from Amazon or another retailer, take a moment to get on the mailing list for the book. I'll let you know when I release updates to the book.

- Get on the mailing list for the book

A Note to Reviewers and Teachers

This book approaches the subject differently than most introductions to Rails. It introduces concepts of product planning, project management, and website analytics to place development within a larger context of product development and marketing. In Book Two, rather than show the student how to use scaffolding, I introduce the model-view-controller design pattern by creating the components manually. Lastly, though every other Rails tutorial shows how to

use a database, Book Two doesn't, because I want the book to be a short introduction and I believe the basic principles of a web application stand out more clearly without adding a database to the application. Though this tutorial is not a typical Rails introduction, I hope you'll agree that it does a good job in preparing Rails beginners for continued study, whether it is a course or more advanced books.

Using the Book in the Classroom

If you've organized a workshop, course, or code camp, and would like to assign the book as recommended reading, contact me at daniel@danielkehoe.com to arrange access to the book for your students. The book is available at no charge to students enrolled in qualified workshops or classes.

Let's Get Started

In the next chapter, we'll start with basic concepts.

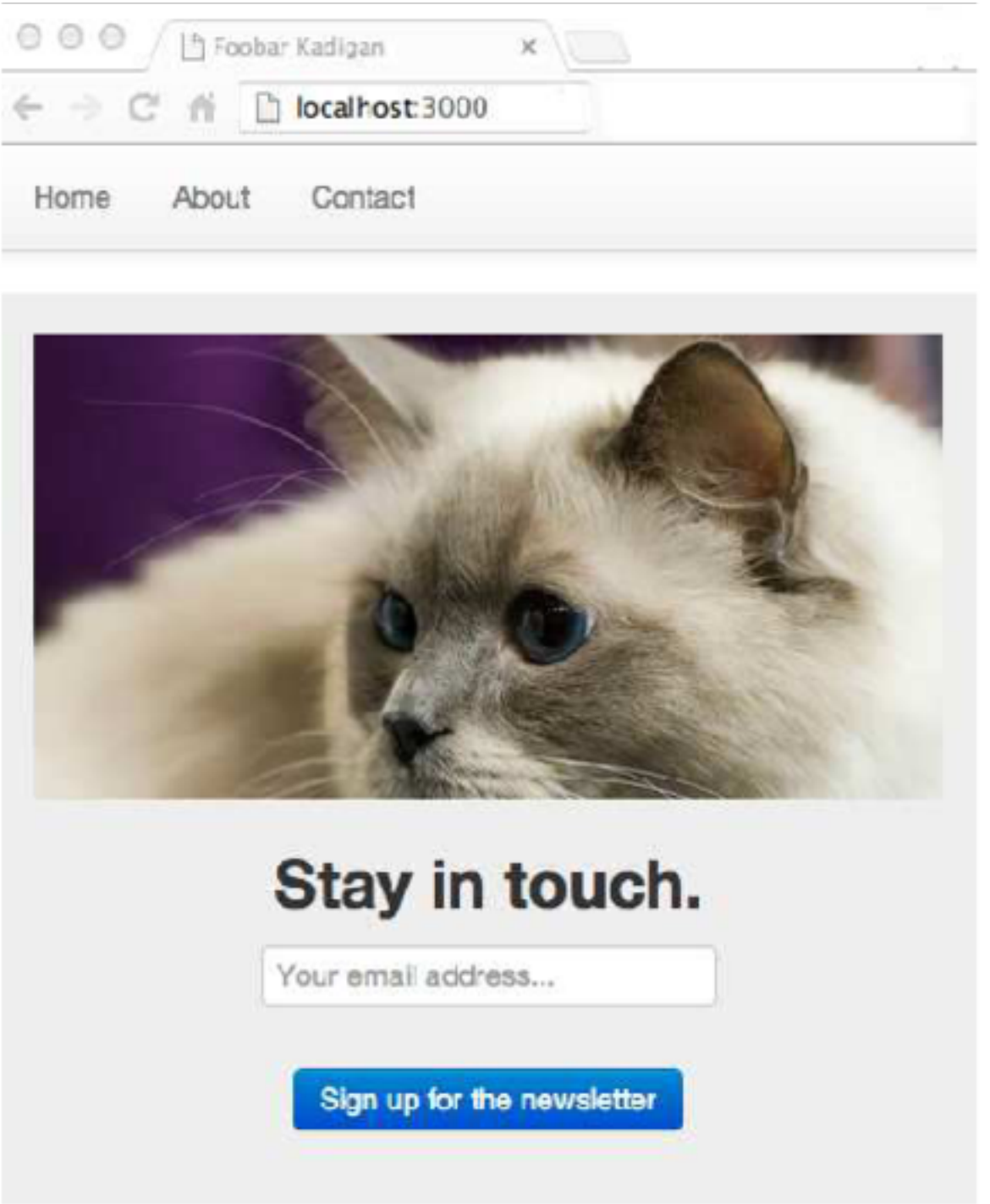

Figure 2.1: The application you will build in Book Two.

Chapter 3

Concepts

This chapter provides the background, or big picture, you will need to understand Rails.

These are the key concepts you'll need to know before you try to use Rails.

In the following two chapters, you'll gain a deeper understanding of Rails, including its history, the guiding principles of Rails, and reasons for its popularity. First, let's consider how the web works.

How the Web Works

We start with absolute basics, as promised.

When you "visit a website on the Internet" you use a *web browser* such as Safari, Chrome, Firefox, or Internet Explorer.

Web browsers are *applications* (software programs) that work by reading *files*.

Compare a *word processing program* with a *web browser*. Both word processing programs and web browsers read files. Microsoft Word reads files that are stored on your computer to display documents. A web browser retrieves files from remote computers called *servers* to display web pages. Simply put, the

World Wide Web is nothing more than files delivered to web browsers by web servers.

Web browsers make *requests* to web servers. Every web address, or *URL*, is a request to a web server. A web server *responds* by sending one or more files. We call this the *request-response cycle*.

Everything displayed by a web browser comes from four kinds of files:

- HTML - *structure* (layout) and *content* (text)
- CSS - *stylesheets* to set visual appearance
- JavaScript - *programming* to alter the page
- Multimedia - images, video, or other media files

At a minimum, a web page requires an HTML file. HTML files contain the words you see on a web page, along with *markup tags* that indicate headlines, paragraphs, and other types of text such as lists. If a web browser receives only an HTML file, it will display text, with default styles for headlines and paragraphs supplied by the browser.

Because it is the World Wide Web, HTML files also contain *hypertext links* to other web pages. Sometimes links appear in the form of a button or an image. Sometimes a web page contains a form with a button that sends information to the web server. Links are web addresses, or URLs, and (you guessed it), they return files.

If the page is always the same, every time it is displayed by the web browser, we say it is *static*. Webmasters don't need software such as Rails to deliver static documents; they just create files for delivery by an ordinary *web server* program. When you learn HTML and create simple web pages, you learn to upload files to a hosting service that provides web servers that deliver your HTML files to web browsers. In principle, you can run a web server delivering web pages from your computer at home but, in practice, most people want a

web server that runs 24 hours a day and is located in a *data center* that has fast and reliable connections to the Internet.

Static websites are ideal for particle-physics papers (which was the original use of the World Wide Web). But most sites on the web, especially those that allow a user to sign in, post comments, or order products and services, generate web pages *dynamically*. When you see a form with a button, you probably are looking at a page that makes a request to a web application.

Dynamic websites often combine web pages with information from a database. A database stores information such as a user's name, comments, advertisements, or any other repetitive, structured data. A database *query* can provide a selection of data that customizes a webpage for a particular user or changes the web page so it varies with each visit.

Dynamic websites use a programming language such as Ruby to assemble HTML, CSS, and JavaScript files on the fly from component files or a database. A software program written in Ruby and organized using the Rails *development framework* is a Rails *web application*. A web server program that runs Rails applications to generate dynamic web pages is an *application server* (but usually we just call it a web server).

Software such as Rails can access a database, combining the results of a database query with static content to be delivered to a web browser as HTML, CSS, and JavaScript files. Keep in mind that the web browser only receives ordinary HTML, CSS, and JavaScript files; the files themselves are assembled dynamically by the Rails application running on the server.

Even if you are not going to use a database, there are good reasons to generate a website using a programming language. For example, if you are creating several web pages, it often makes sense to assemble an HTML file from smaller components. For example, you might make a small file that will be included on every page to make a footer (Rails calls these "partials"). Just as importantly, if you are using Rails, you can add features to your website with code that has been developed and tested by other people so you don't have to build everything yourself.

The widespread practice of sharing code with other developers for free, and collaborating with strangers to build applications or tools, is known as *open source* software development. Rails is at the heart of a vibrant open source development community, which means you leverage the work of tens of thousands of skilled developers when you build a Rails application. When Ruby code is packaged up for others to share, the package is called a *gem*. The name is apt because shared code is valuable, like a gem.

Ruby is a programming language. *Rails* is a development framework. Rails is software code written in the Ruby language. It is a *library* or collection of gems that we add to the core Ruby language. More importantly, Rails is a set of *structures and conventions* for building a web application using the Ruby language. By using Rails, you get well-tested code that implements many of the most-needed features of a dynamic website. When you need additional features, you can add additional gems.

With Rails, you will be using shared standard practices that make it easier to collaborate with others and maintain your application. As an example, consider the code that is used to access a database. Using Ruby without the Rails framework, or using another language such as PHP, you could mix the complex programming code that accesses the database with the code that generates HTML. With the insight of years of developers' collective experience in maintaining and debugging such code, Rails provides a library of code that segregates database access from the code that displays pages, enforcing *separation of concerns*, and making more modular, maintainable programs.

In a nutshell, that's how the web works, and why Rails is useful.

For more on the history of Rails, and an explanation of why it is popular, see the next chapters. But before we dive into Rails, let's look at the increasingly complex world of web development, particularly the difference between front-end and back-end applications, and the programming languages we use, Ruby and JavaScript.

Programming Languages

JavaScript and Ruby are both general-purpose programming languages.

Developers use other popular programming languages such as C, Python, and Java. And developers like to talk about newer languages such as Elixir and Go, often comparing the popularity of programming languages. Most developers use only one or two popular languages on the job such as Ruby and JavaScript but hardcode programmers love to try new languages.

Just a note: Java and JavaScript are unrelated, except by name. Java is a general-purpose language used in large enterprises, such as banking, where large teams of developers build applications. JavaScript is a language that was developed for use in web browsers. It was named "JavaScript" to take advantage of the popularity of Java but has little in common with Java except for the name.

And a further note: HTML, the Hypertext Markup Language, is not a programming language. It is a *markup language* that uses tags to add structure and links to text. It doesn't allow *conditional execution* such as `if... then... else` which is key to programming. If you know HTML you can be a "coder," writing HTML code, but you are not really a programmer.

Ruby and JavaScript

Ruby is the programming language you'll use when creating web applications that run on your local computer or a remote server using the Rails web application development framework.

JavaScript is the programming language that controls every web browser. The companies that build web browsers (Google, Apple, Microsoft, Mozilla, and others) agreed to use JavaScript as the standard browser programming language. You might imagine an alternative universe in which Ruby was the browser programming language. That's not the real world; plus it would be

boring, as learning more than one language makes us smarter and better programmers.

JavaScript and JQuery

Though most of the code in Rails applications is written in Ruby, developers add JavaScript to Rails applications to implement features such as browser-based visual effects and user interaction. For simple Rails applications, you only need to learn Ruby. For more sophisticated web applications, you'll need to know both Ruby and JavaScript.

JavaScript was first used on websites to add little features to the browser. For example, JavaScript can be used to display the current date and time on a web page. Or JavaScript can be used to pop up an annoying window when you try to leave a web page. There was little consistent structure to early JavaScript programs. And because JavaScript is an older language without a built-on package manager, there were no package libraries like Ruby gems to add functionality. Instead, web developers shared scripts or snippets of code to add commonly-implemented features.

JQuery

In 2006, a group of developers released jQuery, a robust collection of scripts that are a foundation for most of the simple interactive user features found on websites today. Rails includes jQuery as part of any Rails application. You'll find jQuery on 65% of websites.

To understand jQuery, you need to know that every web browser takes an intermediate step between receiving an HTML file and displaying a web page. After a web browser receives a file from a web server, it creates code in the computer's memory that describes the web page, complete with text and formatting, which we call the *Document Object Model*, or DOM. JQuery scripts