Extracted from:

Take My Money

Accepting Payments on the Web

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Take My Money

Accepting Payments on the Web

Noel Rappin

edited by Katharine Dvorak

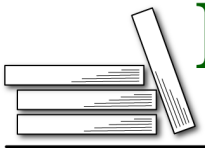
Take My Money

Accepting Payments on the Web

Noel Rappin

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Katharine Dvorak (editor)
Potomac Indexing, LLC (index)
Liz Welch (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-199-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January 2017

Disclaimer: This book is intended only as an informative guide on setting up a financial transaction website. Information in this book is general, and the author, editors, and The Pragmatic Programmers, LLC disclaim all liability for compliance to federal, state, and local laws in connection with the use of this book. This work is sold with the understanding that the author, editors, and The Pragmatic Programmers, LLC do not offer legal, financial, or other professional services. If professional assistance is required, the services of a competent professional person should be sought.

Preface

A few years ago, I started working on my first web application that required serious payment business logic. The application was a legacy rescue, meaning that the actual core of the payment section already existed—the application already had a payment gateway, which is the third-party service that handles the credit card transaction, and knew how to communicate credit card information.

I was asked to significantly expand the payment logic. You'd think that because the API communication was already in place, the hard part would already be done. I certainly thought that but quickly learned otherwise. Suddenly I had to deal with problems managing inventory, data validation, refunds, administrators who needed to be able to override user rules, security, and fraud. I looked around for information about good practice in this area, and I didn't find much. The different gateway APIs all have documentation and tutorials about how to connect to their servers, some of which are quite helpful; however, I couldn't find answers to my questions as they related to the larger context of the application.

So I fell back on general principles of software development. And I made mistakes. This book comes from somewhere between the things I did right, the things I did wrong, and the things I wish I had done. It is my hope that with this book, you will be able to build your payment application with less stress and without making the same mistakes.

About This Book

When people talk about software design, they often refer to "business logic" as an abstract blob of complexity that they need to manage. The topics in this book cover the literal core of business logic: taking payments, providing a service or good in exchange, managing the flow of money, and reporting finances. Not only are many of these topics arcane in their own right, but they also tend to be the locus of the most tangled and complex logic in any system. And as if that wasn't enough, people tend to react more strongly to

bugs or surprises that involve money than they do to bugs that involve, say, the search algorithm.

A payment application can be really rewarding to build—you can actually see the return on investment as genuine money (or at least digital bits that we pretend are money) gets added to your account. But it can also be amazingly stressful to manage, with security concerns, compliance concerns, and the fact that your business may be dependent on the features working smoothly.

Over the course of this book we are going to build a robust web application that takes credit card information in return for goods and services. We will start by building a shopping cart, which is often dependent on a lot of business logic but doesn't have the same dependence on money. We'll use that opportunity to talk about general principles of building complex logic without the added complexity of a third-party payment gateway.

With the shopping cart in place, we'll next look at the basics of taking a credit cart payment, first using Stripe as our gateway, then using Stripe's client-side authentication, and then again using PayPal. Once we have a successful payment in the books, you'll learn about some of the many things that can go wrong in payment processing and how to work around them. And after that, you'll learn how to handle recurring payments in Stripe.

However, taking payments is only part of what you need to do to manage a fully functional payment application. Once we have payments covered, you'll learn about administration and how to bend the rules that sometimes need to be bent. Your business model may also involve making payments, so you'll take a look at that.

Finally, one key difference between financial transactions and many other kinds of business logic is that financial transactions often require interaction with the law. As such, we'll go over three key legal issues that pertain to online payment applications: taxes, reporting, and compliance.

About You

Inevitably, a technical book needs to make decisions about what languages that examples will be presented in and what knowledge we expect the readers will either already have or be willing to learn elsewhere. While I hope that the principles of design and interaction have value outside the specific tools used, the code samples are written in a specific set of tools.

This book uses Ruby and Rails for its server-side code and JavaScript ES6 for its client-side code. I'm assuming that you're already comfortable with

Ruby and Rails and you don't need this book to explain how to build a Rails project, or how JavaScript syntax works. I am not assuming that you have any previous familiarity with payment processing.

About the Project

The web application we are going to build over the course of this book is for a small-town theater company. We'll pretend we've been granted a lucrative contract to create a website for the Snow Globe Theater, a small theater group that brings Shakespeare to the wilds of Alaska. And by "lucrative contract," I mean that we and the person who runs the theater are old buddies, the theater desperately needs to start selling tickets online, and our buddy is calling in some favors.

We'll use a payment gateway and their API to actually process credit card payments and transfer the money to us, but that's only part of the work we need to do:

- Tickets to our shows are a finite resource, so we have inventory management to deal with.
- When purchases are made, we need to notify the user, track the ticket, and update some totals, so we have workflow issues.
- We're going to want to keep an eye on sales, which means reporting.
- We have legal issues to contend with, like taxes and compliance laws.
- Sometimes, bad things will happen in our code, so we need to be able to identify and modify bad data.
- Some of those bad things will be malicious, so we'll look at security.
- Some of those bad things will be deliberate overrides of our normal logic, so we need administration.

Over the course of the book we're going to touch on all of these issues, and by the end, the Snow Globe Theater will have a robust payment-taking machine.

A Note About the Code

Before we add features to the application, I need to say a few words about the application itself. The application uses Ruby on Rails and was built up using Daniel Kehoe's Rails Composer.[1] It uses PostgreSQL as its database.[2]

1. http://www.railscomposer.com
2. http://www.postgresql.org

The source code for this application is available as a zip file on the book's web page,[3] which is where you can also find the book's interactive discussion forum. The readme.md file in the code has any other information you will need to start the application, which may include code errata that was discovered after the book was published. Each directory in the zip file is a complete, working version of the Rails application in progress. Code samples throughout the book specify which directory, and therefore, which version of the app, they are from.

Important code note: The code samples presented in the book focus on the business logic and interaction with payment gateways. Often, in the interest of focus, boilerplate code for items like views, controllers, and even sometimes tests are not displayed in the book. The code samples provided have all the fully tested code needed to run the site. Unless otherwise noted, each individual branch is fully tested and has minimal views needed to run the site.

The code samples used throughout this book will show you:

- How to work with the relevant APIs and third-party tools to get something to work.
- How to mitigate complexity, meaning how to manage the business logic for long-term issues of readability, changeability, debugging, and so on.
- How to test. This isn't a book on test-driven development or testing in general, but you'll see examples of testing techniques that are particularly effective in dealing with business logic or third-party libraries. All the code features in the book were written driven with tests, so even where the tests in the book aren't shown, you can still look at the sample application to see further examples of testing in action.

Handling payment logic is complicated, but it is also concrete and quite literally rewarding. Let's go build a website, shall we?

Noel Rappin
noel@noelrappin.com

3. http://www.pragprog.com/book/nrwebpay