Extracted from:

Agile Web Development with Rails 5.1

This PDF file contains pages extracted from *Agile Web Development with Rails 5.1*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2017 The Pragmatic Programmers, LLC.

All rights reserved.

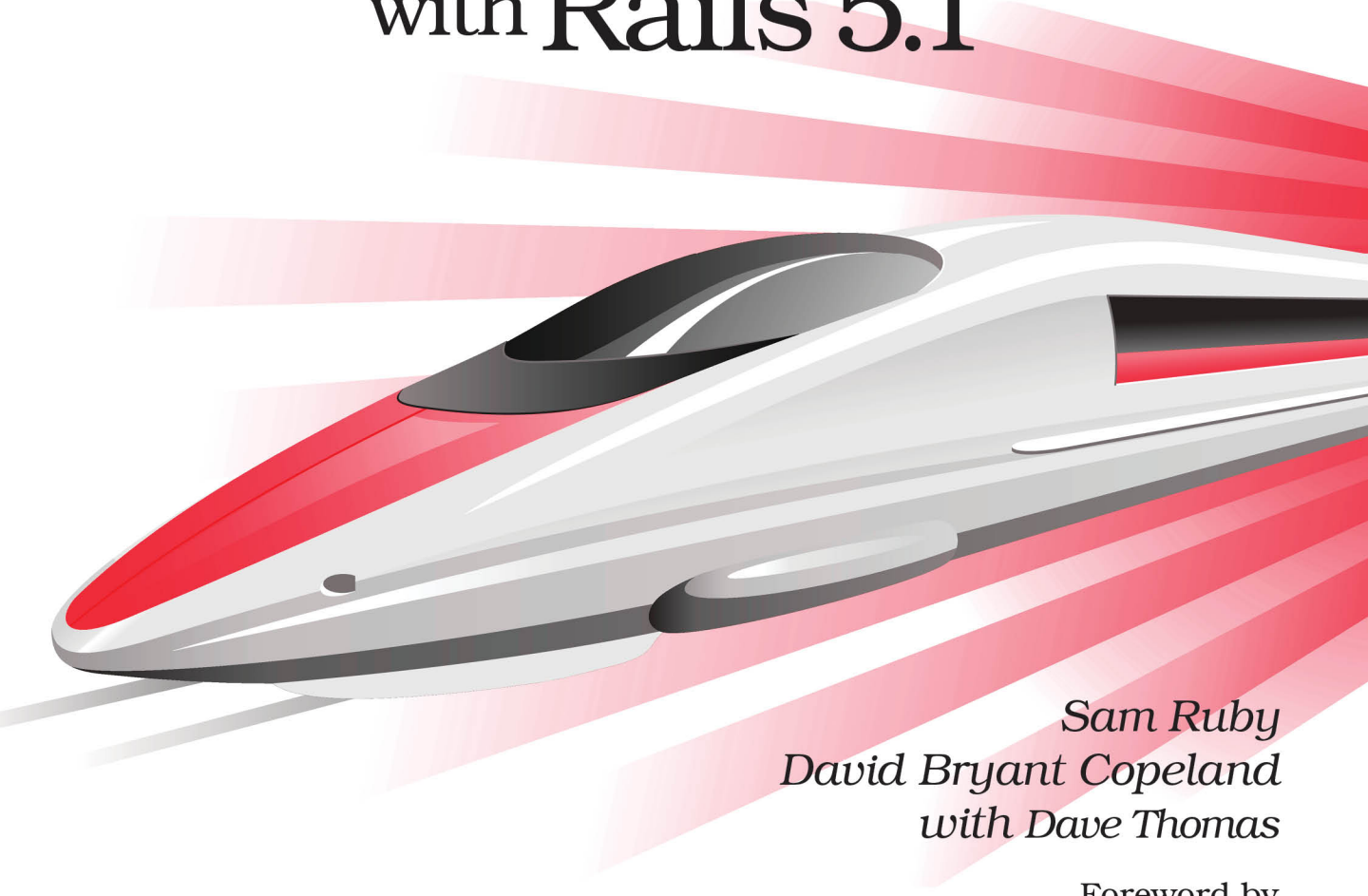
No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Agile Web Development with Rails 5.1



Sam Ruby
David Bryant Copeland
with Dave Thomas

Foreword by
David Heinemeier Hansson

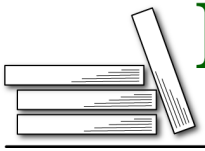
RAILS

Edited by Susannah Davidson Pfalzer

Agile Web Development with Rails 5.1

Sam Ruby
David Bryant Copeland
with Dave Thomas

The Pragmatic Bookshelf
Raleigh, North Carolina



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 Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

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:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Development Editor: Susannah Davidson Pfalzer
Indexing: Potomac Indexing, LLC
Copy Editor: Molly McBeath
Layout: Gilson Graphics

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

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

Copyright © 2017 The Pragmatic Programmers, LLC.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 978-1-68050-251-0
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—November 2017

In this chapter, you'll see:

- Using Webpacker to manage app-like Javascript
- Setting up a development environment that includes Webpack
- Using React to build a dynamic web form
- Using Capybara and ChromeDriver to test JavaScript-powered features

CHAPTER 13

Task H: Entering Additional Payment Details

Our customer is enthusiastic about our progress, but after playing with the new checkout feature for a few minutes, she has a question: how does a user enter payment details? It's a great question, since there isn't a way to do that. Making that possible is somewhat tricky, because each payment method requires different details. If users want to pay with a credit card, they need to enter a card number and expiration date. If they want to pay with a check, we'll need a routing number and an account number. And for purchase orders, we need the purchase order number.

Although we could put all five fields on the screen at once, the customer immediately balks at the poor user experience that would result. Can we show the appropriate fields, depending on what payment type is chosen? Changing elements of a user interface dynamically is certainly possible with some JavaScript, but it's quite a bit more complex than the JavaScript we've used thus far. Rails calls JavaScript like this *app-like JavaScript*, and it includes a tool named Webpacker that will help us manage it. Webpacker will handle a lot of complex setup for us so that we can focus most of our efforts on giving our customer—and our users—a great experience checking out. (Refer back to Chapter 1, *Installing Rails*, on page ?, for installation instructions for the tools used in this chapter.)

Iteration H1: Adding Fields Dynamically to a Form

We need a dynamic form that changes what fields are shown based on what pay type the user has selected. While we could cobble something together with jQuery, it would be a bit cleaner if we could use a more modern

• Click HERE to purchase this book now. discuss

JavaScript library like React.[1] This will also form a solid base from which we can easily add additional features later.

Using JavaScript libraries or frameworks can often be difficult, as the configuration burden they bear is far greater than what we've seen with Rails. To help us manage this complexity, Rails includes *Webpacker*, which provides configuration for *Webpack*.[2] Webpack is a tool to manage the JavaScript files that we write. Note the similar names. Webpack*er* is a gem that's part of Rails and sets up Webpack inside our Rails app.

Managing JavaScript is surprisingly complex. By using Webpack we can easily put our JavaScript into several different files, bring in third-party libraries (like React), and use more advanced features of JavaScript not supported by a browser (such as the ability to define classes). Webpack then compiles all of our JavaScript, along with the third-party libraries we are using, into a *pack*. Because this isn't merely sprinkling small bits of JavaScript in our view, Rails refers to this as *app-like* JavaScript.

While we could use Webpack directly with Rails, configuring Webpack is extremely difficult. It's highly customizable and not very opinionated, meaning developers must make many decisions just to get something working. Webpacker essentially *is* the decisions made by the Rails team and bundled up into a gem. Almost everything Webpacker does is to provide a working configuration for Webpack and React so that we can focus on writing JavaScript instead of configuring tools. But Webpack is the tool that manages our JavaScript day-to-day.

React is a JavaScript view library designed to quickly create dynamic user interfaces. We'll use it to create a dynamic payment method details form, and Webpacker will ensure that the configuration and setup for all this is as simple as possible. That said, there's a bit of setup we need to do.

First, we'll configure Webpacker and install React. After that, we'll replace our existing payment-type drop-down with a React-rendered version, which will demonstrate how all the moving parts fit together. With that in place, we'll enhance our React-powered payment type selector to show the dynamic form elements we want.

1. https://facebook.github.io/react/
2. https://webpack.js.org

• Click HERE to purchase this book now. discuss

Configuring Webpacker and Installing React

Webpacker is a separate gem that you must install in addition to Rails. Add it to your Gemfile like so:

```
gem 'webpacker', '~> 3.0'
```

Install this with bundle install.

Next, set up Webpack by running bin/rails webpacker:install.

```
$ bin/rails webpacker:install
Creating javascript app source directory
      create  app/javascript
      create  app/javascript/packs/application.js
Copying binstubs
       exist  bin
      create  bin/webpack-dev-server
      create  bin/webpack
   identical  bin/yarn
Copying webpack core config and loaders
      create  config/webpack
      create  config/webpack/configuration.js
      create  config/webpack/development.js
      create  config/webpack/development.server.js
      create  config/webpack/development.server.yml
      create  config/webpack/paths.yml
      create  config/webpack/production.js
      create  config/webpack/shared.js
      create  config/webpack/test.js
      create  config/webpack/loaders
      create  config/webpack/loaders/assets.js
      create  config/webpack/loaders/babel.js
      create  config/webpack/loaders/coffee.js
      create  config/webpack/loaders/erb.js
      create  config/webpack/loaders/sass.js
      create  .postcssrc.yml
      append  .gitignore
Installing all JavaScript dependencies
         run  ./bin/yarn add webpack webpack-merge js-yaml…
yarn add v0.20.3
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

«lots of output»

Done in 24.95s.
Installing dev server for live reloading
         run  ./bin/yarn add --dev webpack-dev-server from "."
yarn add v0.20.3
```

```
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
success Saved lockfile.
success Saved 82 new dependencies.

«lots more output»

Done in 5.11s.
Webpacker successfully installed
```

As you can see from the output, this created several configuration files in config/webpack and installed various JavaScript libraries. The libraries that were installed are listed in package.json. package.json is the JavaScript equivalent to our Gemfile——it lists all the necessary JavaScript libraries for our app to run. The equivalent of Bundler is *Yarn*.

Just like bundle install downloads all the gems our app needs, yarn install downloads all the JavaScript libraries we need. As a convenience, the webpacker:install task ran yarn install for us.

Webpacker can also install and configure some common JavaScript frameworks such as Angular, Vue, or React. We chose React because it's the simplest overall and is the best fit for solving our problem. To have Webpacker set it all up for us, run the task webpacker:install:react:

```
$ bin/rails webpacker:install:react
Copying react loader to …config/webpack/loaders
      create  config/webpack/loaders/react.js
Copying .babelrc to app root directory
      create  .babelrc
Copying react example entry file to …app/javascript/packs
      create  app/javascript/packs/hello_react.jsx
Installing all react dependencies
     run  ./bin/yarn add react react-dom babel-preset-react from "."
yarn add v0.20.3
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
warning "react-dom@15.4.2" has unmet peer dependency "react@^15.4.2".
[4/4] Building fresh packages...
success Saved lockfile.
success Saved 26 new dependencies.

«lots of output»
Done in 7.17s.
Webpacker now supports react.js
```

If you've ever tried to set up Webpack and a JavaScript framework like React before, you'll appreciate how much work Webpacker has just done for us. If you've never had the privilege, trust me, this saves a ton of time and aggravation.

Webpacker also created a rudimentary React component in app/javascript/packs/hello_react.jsx. Don't worry about what that means for now. We're going to use this generated code to validate the installation and set up our development environment. This generated code will append the string "Hello React!" to the end of our page, but it's not activated by default. Let's find out why, configure it to be included in our views, and set up our development environment to work smoothly with Webpacker.

Updating Our Development Environment for Webpack

Webpacker includes a helper method called javascript_pack_tag() that takes as an argument the name of the file in app/javascript/packs whose JavaScript should be included on the page.

The reason Rails doesn't simply include all JavaScript all the time is that you might not want that to happen for performance reasons. Although our payment details code won't be terribly complex, it'll still be a chunk of code our users will have to download. Since it won't be needed anywhere else in our app, we can make the user experience faster and better by only downloading the code when it's needed.

Webpacker allows us to have any number of these separately managed *packs*. We can include any that we like, wherever we like. To see how this works, let's add a call to javascript_pack_tag() to our app/views/orders/new.html.erb page to bring in the sample React component that Webpacker created for us.

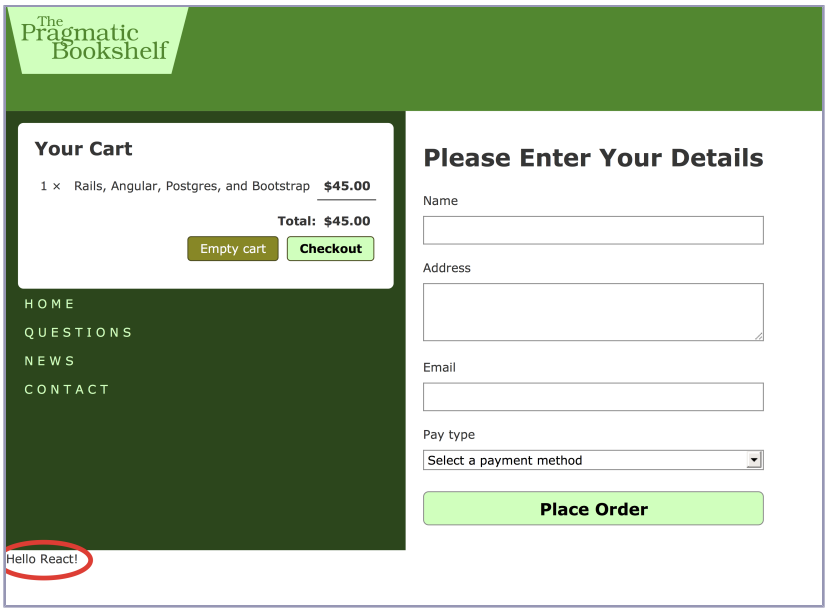
rails51/depot_pa/app/views/orders/new.html.erb

```
<section class="depot_form">
  <h1>Please Enter Your Details</h1>
  <%= render 'form', order: @order %>
</section>

<%= javascript_pack_tag("hello_react") %>
```

If you add some items to your cart and navigate to the checkout page, you should see the string "Hello React!" at the bottom of the page, as shown in the screenshot on page 10.

This validates that all the internals of Webpack are working with the app (which is always a good practice before writing code so we can be sure what might be the cause if something's wrong). Now we can start building our feature. We need to replace the existing drop-down with one powered by React



and our Webpacker-managed JavaScript. Doing *that* requires a slight diversion to learn about React.