Extracted from:

Serverless Single Page Apps

Fast, Scalable, and Available

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Serverless Single Page Apps

Fast, Scalable, and Available

Ben Rady

edited by Jacquelyn Carter

Serverless Single Page Apps

Fast, Scalable, and Available

Ben Rady

The Pragmatic Bookshelf

Raleigh, North Carolina

The team that produced this book includes:

Jacquelyn Carter (editor)
Potomac Indexing, LLC (index)
Nicole Abramowitz, 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: 978-1-68050-149-0
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—June 2016

Introduction

After years of building single page web applications and wishing that I could break free of the constraints imposed by application servers, my wish has come true. Amazon (and, increasingly, other companies) have developed technologies that make it possible to use a *serverless* approach that removes a lot of the risks and costs of building and scaling web applications. This idea is so compelling and so transformative that I had to write a book about it.

I wrote this book for the people I know in my life who want to build something on the web. Some have an app they want to build—something they think will be the next big thing. Some are just starting out in web development and have never built any kind of app—world-changing or otherwise. And some are experienced web developers who've built dozens of Model-View-Controller based web apps with Java, Ruby on Rails, or Node.js. With the emergence of this new technology, I want to share what I've learned about it, in the hopes that someone out there will use it to build something great.

In this introductory chapter, you'll get an idea of what to expect from this book and what you can do to get the most out of it.

Guiding Principles

I wrote this book with a few guiding principles in mind. The purpose of these principles is to focus the scope of the book and make it accessible and relevant to you, the reader. I've listed these principles here to give you some insight into how I've written this book and to help you better understand the context of the material you're about to read.

Some of these principles may be potentially controversial. Some of them even run counter to popular thinking about how to build web applications. However, these principles will help you understand this topic in depth. It's better to be controversial than try to be everything to everybody.

Use Web Standards and Familiar Tools

In this book, you'll use a small, curated set of tools to build a single page web application. At certain points in the book, you'll create functionality that other tools already provide—tools that I've intentionally not included in the app. You might wonder why we're not using those tools if they provide needed functionality.

Reading a book of this sort is fundamentally an act of learning. When learning, it's helpful to work with familiar tools. Otherwise, you wind up spending more time learning about the tools than learning the technique. I didn't want the choice of a framework or library to be a distraction. This book is about serverless web applications, not frameworks or libraries. To remain agnostic, we'll instead lean on tools that are familiar to web developers (like jQuery), web standards, and the web services that make a serverless design possible.

This book is about web services, not web frameworks.

It's possible that after reading this book, you'll reach for a client-side web framework, such as React or Angular, to build your own web apps. These tools have gained a lot of traction in the web development community in the last few years, and I expect we'll see many successful projects that use them. Everything you'll learn in this book is compatible with anything you'll want to do with these frameworks. They are complementary, not contradictory.

Use Functional JavaScript

In this book, you won't be creating any classes in JavaScript. Creating class hierarchies to solve problems can make sense in languages with rich, object-oriented type systems, but JavaScript isn't one of those languages. Instead, we'll use a functional style that's easier to understand.

This means you won't get caught up in scoping issues with the `this` keyword. You'll avoid prototypes and inheritance altogether. You won't use the `new` keyword paired with specially designed functions to create objects; instead, you'll create them using the object literal: `{}`.

You can decide for yourself if this is a style you want to adopt. While some real, tangible benefits to this approach exist, at the end of the day, many software design decisions come down to preference and style. Code, after all, should be written for humans first and computers second. As long as it works, it doesn't matter to the computer what the code looks like.

Avoid Yaks

When working on a project, my goal is always this: deliver incremental improvement steadily over time, in any direction the environment demands, for as long as is needed. Meeting this goal means avoiding anything that causes my rate of delivery to grind to a halt, like a lot of *yak shaving*.

If you're not familiar with the term *yak shaving*, imagine you want to buy a sweater as a gift for a friend. You go to the store only to discover it has no sweaters to sell. Fortunately, another customer there knows of a great tailor down the street who might be able to make you one. So you head to the tailor, who has a wonderful sweater pattern and a machine that can knit it, but the yarn supplier hasn't made a delivery today. So you head to the supplier…

And this continues on and on until you find yourself in a field in western Tibet, shaving a yak to spin yarn. "How did I get here?" you might ask yourself. "All I wanted was a sweater." When a chain of seemingly related and necessary tasks distracts you from your real goal, you're yak shaving. Thankfully, the cure for yak shaving is often simply to realize that you're yak shaving and buy a hat instead.

I want to keep you from shaving yaks while reading this book. That's why I use a prepared workspace and a minimal set of tools. You should spend your time learning, not installing, configuring, and troubleshooting.

Move Faster with Tests

Have you ever been scared to change a bit of code? Maybe you were unsure about what it was supposed to do or why it was supposed to do it. "First, do no harm" applies to programmers as well as doctors. Situations like this can make you feel like you're stuck.

Imagine if you had a trusted colleague—an expert in that system—sitting next to you as you made those changes. If you introduced any bug into the system, you would be interrupted with a clear and concise explanation of why this change would be a Bad Idea. If you had this trusted colleague, would you have been stuck then?

Uncertainty slows us down and limits the scope of our solutions. To build software quickly, you must work with *confidence*. To get that confidence, you can create your own automated expert—something that knows every detail of the system, how it works, and why it works like it does. This expert can live alongside the system, changing as it changes, one a reflection of the other. You can create this expert with a suite of tests that is fast enough to run continuously—hundreds of tests per second—after every change you make to the code.

Once you have a suite of tests in place, new options open up to you. When you no longer fear change, you can design the application as you go, rather than trying to get it "right" from the start. This means you can quickly adapt to the world as it changes, instead of trying to predict what you'll need and what you won't. You can base decisions on current information instead of speculation, and the application will be as simple as it can be for now.

In this book, you'll use a technique called *test-driven development (TDD)* to write both the application and its tests. Learning to build software with tests is a skill, and you have to practice it if you want to reap the benefits. My goal in including TDD in this book is not only to show you how to test specific kinds of behavior in web apps, but also to demonstrate how easy it is to test a typical web application, if you know how to do it.

When practicing TDD, you work in a three-step cycle, usually expressed as Red-Green-Refactor. First, you write a test that checks for behavior that the program doesn't yet have. If the test fails the way you expect, then you can be confident that it is testing the behavior you want to add to the app. Now the tests are *Red*. Once you have a failing test, add the simplest code you can to make the test pass…usually a few lines. Now the tests are *Green*.

After you've added some behavior to the app using a test, it's time to take a step back and look at the bigger picture. Have you introduced some duplicate code with the implementation? Do all the variables and functions have descriptive and accurate names? Is there a simpler approach that you missed? Now is the time to think about these things and *Refactor*. Refactoring is changing code without changing its behavior. The tests you've written will tell you whether or not you've changed the behavior, so it's important to refactor only when they're passing. Using this as your safety net, you can clean up any issues with the code before moving on to the next test.

The more you practice TDD, the faster you'll go, and the more value you'll get from it. By repeating this Red-Green-Refactor process over and over, you'll learn how to incrementally build an application that is well designed and well tested. This will not only give you confidence that your software works, but will also make it easier for you to change it over time.

Learn by Doing

This book is a tutorial, so you'll learn by doing. Throughout this book, you'll build a serverless single page web application as a running example. The purpose of this tutorial is to explain the concepts of serverless architecture in a concrete way. Because the result of this tutorial is a working application,[1] you can have confidence that the techniques in this book work as advertised.

Taking this approach means I can't go into as much depth as I might like without turning the book into a 400-page tome. For that reason, I've added a "Next Steps" section to every chapter to give you some additional topics to dig into if you want a little more.

Start with a Prepared Workspace

To get you up and running quickly, I've provided a *prepared workspace*. It includes everything you need to get started and shouldn't take long to set up. Imagine you were painting a work of art; I've gathered the paint, the easel, and the canvas for you. All that's left for you to do is create.

To use this prepared workspace, you'll need a computer with a Bash-compatible shell to use the scripts and utilities included. This can be OS X, any *nix flavor, or FreeBSD. You can probably get by with Windows if you have Cygwin installed. You'll also need a web browser with a developer console. I used Google Chrome for most of the examples in this book, but Firefox provides similar functionality in most cases.

How to Read This Book

You can read a book in many ways, not just the obvious one. Which approach you should take with this book depends on what you want to achieve. Here, you'll find some common reasons for reading a book like this, and my recommendation for how to use this book to best achieve those goals.

Goal #1: Understand Serverless vs. Traditional Single Page Apps

What to Do

1. Read the first three sections of Chapter 1, *Starting Simple*, on page ? to understand the benefits and disadvantages.
2. Skim through the remainder of Chapter 1, as well as Chapters 2 and 3.
3. Read through Chapters 4–8, working along with the tutorial where possible.

1. http://learnjs.benrady.com

If you're an experienced web developer who has built single page web apps before, and you want to learn more about serverless web applications, you probably don't need all the material in the first three chapters. These chapters show a (more or less) vanilla.js approach to building single page web applications. The intent here is to demonstrate the fundamental components of a single page web app. I'm trying to define what parts are essential, and offer some basic implementations to act as a reference. If you don't write a lot of tests for your web apps, you can work through some of the testing examples in Chapter 2 to pick up some new skills.

After reading the first three sections in the first chapter, you'll want to concentrate your attention on Chapters 4–8. I do suggest that you build a simple web application, or at least a skeleton of one, so that you can try out the techniques in the later chapters for yourself. You can learn a lot through experimentation.

Goal #2: Build Your First Single Page App

What to Do

1. Read any of the following necessary supporting material before getting started.
2. Work through all the chapters, and, if necessary, the appendices.

If you're building a web application for the first time, you'll want to read the entire book, possibly cover to cover. In addition, you'll want to make sure you're up to speed on basic web technologies, including HTML, Cascading Style Sheets (CSS), and JavaScript. Check out the *Free Resources* here to get going. Once you understand these topics, dive into the book.

Free Resources

Learn to Code HTML & CSS [How14] by Shay Howe[a]

Eloquent JavaScript [Hav14] by Marijn Haverbeke[b]

a. http://learn.shayhowe.com/html-css/
b. http://eloquentjavascript.net/

Goal #3: Build a Serverless App Using Your Favorite Web Framework

What to Do

1. Work through Chapter 1.
2. Replicate the tests and functionality in Chapters 2 and 3 using your favorite web framework.
3. Work through Chapters 4–8.

As I've already mentioned, I didn't want this book to focus on web frameworks, but you can certainly use them to build serverless single page apps. If you're familiar with the basic elements of a single page app, and you rely on a client-side web framework to provide those for you, you can easily reproduce the functionality we create in Chapters 2 and 3 using that framework. Once you have that functionality in place, you should be able to follow along with the rest of the tutorial, adapting where necessary to meet the expectations of your framework of choice.

Goal #4: Create a Minimum Viable Product (MVP)

What to Do

1. Read the first three sections of Chapter 1, *Starting Simple*, on page ? to understand the benefits and disadvantages.
2. Read *Costs of the Cloud*, on page ? to understand the costs of building an MVP this way.
3. If the approach seems reasonable, read all the remaining chapters (and the appendices, if necessary).
4. Use the prepared workspace as a starting point to build your MVP.

When starting a new product or business, your first and most important challenge is to figure out what the market wants to buy and what it's willing to pay. Many people refer to this as *product/market fit*, and finding it is the key to building a successful product or service.

One effective way to find a product/market fit is to simply build a product and try to sell it. Validating both the demand in the market and your ability to connect with those customers via a sales or marketing channel is a critical hurdle that you should overcome as soon as possible. Of course, building a complete application for this purpose can take much more time and money than what you have available, so an alternative is to build a minimum viable product that demonstrates the product's core value.

Serverless single page apps are a great way to try a new idea, explore a potential new market, or create a minimum viable product. Building these kinds of applications in lieu of traditional web apps or native apps means you can get to your customers faster. You can build an initial version in only hours and deploy it in seconds. These apps can be updated instantly, are easily split tested, and can provide detailed usage metrics that help you understand what your customers want.

In addition to being inexpensive to run, quick to build, and almost universally accessible by users, serverless single page apps can scale "without limits" (as Amazon likes to say), so that if your product taps into a strong demand in the marketplace, you can meet that demand and retain the customers you acquire.

Online Resources

You can find the apps and examples shown in this book at The Pragmatic Bookshelf website for this book.[2] You'll also find the community forum and the errata-submission form, where you can report problems with the text or make suggestions for future versions.

You can find the prepared workspace under my account (benrady) at Github.com.[3] If you don't have one already, create an account and fork the repository to get started. For more detailed instructions on how to do this, see *Using Your Workspace*, on page ?.

Ben Rady

benrady@gmail.com

June 2016

2. https://pragprog.com/book/brapps/serverless-page-apps
3. https://github.com/benrady/learnjs