Book Reviews

MARK LAMOURINE

SPA Design and Architecture

Emmit A. Scott, Jr.
Manning Publications, 2016, 288 pages
ISBN 978-1-61729-243-1

It's been a long time since I worked on the front end of a Web service. I'm passing familiar with JavaScript and completely comfortable with HTML and the DOM. The ideas of AJAX and REST services are clear. I thought I had a reasonable handle on how the client side of Web applications were built.

The "SPA" in the title stands for "Single Page Application," and you've almost certainly used one. If you've used almost any of Google's applications or searched for local movies you've seen an SPA.

SPAs have become prevalent in the last five years with the adoption of the HTML5 standard and a set of JavaScript frameworks which take care of much of the boilerplate and common behaviors. They are designed to move much of the application logic to the client side (your browser) and to minimize the delays involved in repeated page loads that characterized early Web applications.

Scott's goal in this book is to show you the internals of an SPA and then how to assemble them into an application.

After the mandatory chapter introducing SPAs in general, I was a little surprised that the next two didn't seem to speak directly to SPAs at all. In Chapter 2 Scott gives a survey of the MVC (model view controller) pattern and its derivatives. All of the JavaScript frameworks for SPAs are based on one of these models. Scott cites a number of the most popular ones as examples and notes some of the benefits, costs, and quirks of each one without picking a favorite.

Chapter 3 is a tutorial on modules in JavaScript. I would have thought the module construct was a well-known idiom, but I admit I learned a lot from Scott's description.

My confusion resolved as I continued to read.

It turns out that this kind of book is hard to write and not easy to read and understand at the first pass. Developing modern Web applications requires fluency in at least three "languages" as well as the behaviors and quirks of all of the major Web browser rendering systems (even when using a framework to abstract them). Designing and implementing the client side of an application requires the developer to have an intricate understanding of how the data flows from a server through the client application and how that is, in turn, presented by the browser. Scott guides the reader through these interactions from the browser back to the server as well as touching on testing and debugging. That it took me a couple of times through to digest it is a reflection of my own meager background in this area.

Scott starts with application "routing" and the idea that in an SPA there is only a single "Web page" but there are multiple views of the service. The view is selected through the "router," which takes advantage of the browser URL history and the ability of the browser to decompose a URL and respond as instructed by some loaded JavaScript.

In the next chapter, Scott reveals how to control the layout and presentation of the views using HTML, CSS, and various templating frameworks. This is where the application is given both a structure and a style that (it is hoped) presents the user with the information and behaviors they need to complete their tasks efficiently.

The JavaScript module pattern comes back to the fore now. Scott shows how this pattern can be used to map logic to each view in a clean, coherent manner. He also discusses how the data will be represented in the client-side model of the application. This leads nicely into the final active part of the SPA: communicating with the server.

In this chapter Scott examines how to generate and respond to asynchronous requests to the server both with several of the major SPA framework mechanisms and using the XMLHttpRequest method directly. He details asynchronous data exchange in both directions and shows how to build the service interactions into the modules that make up the client-side data model.

The final sections cover unit testing and client-side tasks. The latter are actions that the client may take which are not directly associated with any particular model or view. He presents them as a means to run repetitive tasks during development such as code CI and testing.

Scott doesn't try to create a single application in his narrative. Because he is reflecting on a number of different frameworks, using their contrasts to highlight behaviors and features, no single sample application would fit. He does include a short application example using Angular.js and Backbone.js in the appendices, but his presentation in the main body of the book is fairly agnostic to any framework selection.

Each chapter concludes with a set of questions and exercises that are meant to help the reader set the main concepts in memory and to give some active practice. These "challenges" are indeed a challenge, not something you can merely cut-and-paste from the text. A reader who follows through will get much more than a casual reader.

In the end I liked Scott's focus on concepts and options rather than advocating for one framework or another. I think I was not as prepared as I should have been to take this book on. My knowledge of JavaScript and DOM is rusty, and I have not kept up on current practice and idiom. This required me to go outside and brush up to be sure I'd understood and absorbed what was presented. This is a good book for someone who has gotten their hands dirty with browser programming and is ready to start learning how to design a fast modern Web service.

Single Page Applications: JavaScript End-to-End

Michael S. Mikowski and Josh C. Powell
Manning Publications, 2014, 408 pages
ISBN 978-1-617290-75-0

"You can do it all in JavaScript. Here's how" is the message that the authors offer in *Single Page Applications: JavaScript End-to-End.* Their aim is to build an SPA demonstrator, client, and server side completely in JavaScript. They go so far as to avoid even the popular SPA JavaScript frameworks, choosing instead to build the core functionality, the routing and view selection logic, even the HTML template resolution directly in JavaScript.

I'm not sure most people would be willing to take on the extra work that the SPA frameworks offload for you, but there's certainly a lot to be learned by looking at how one would do it.

Mikowski and Powell follow the tried-and-true narrative of building a simple demo app and enhancing it chapter by chapter. In their case it actually shows a good Agile style progression, although I'm not sure if that was their intent. Because they are building both the client and the server in JavaScript, and you don't get to the server part until more than half way through, you also learn a lot about mocking data and services in JavaScript.

They begin by building a skeleton for the application, which they call the "Shell." This is a kind of root module for the application. Features will be hung off this module and will add functionality as the development process progresses. From here they show how to add logic and presentation to each new feature in a clean, incremental way. They develop the data models and views in conjunction so the reader can see how the back and front are related and how data flows in and out.

On the server side, Mikowski and Powell use Node.js and MongoDB. They promote the idea that using a single language for both the client and server makes development easier. While in general I agree, I wouldn't normally have picked JavaScript as my one language, but since the browsers have chosen for us it will have to do. Certainly the use of JSON for data transfer and storage does remove lots of the hassle of encoding and decoding data both for communications and database storage.

The authors are very conscious of the development environment and developer tasks. While developing the components, they also lay out best practices for directory structure and file naming for consistency and maintainability. These seem to mirror other recommendations I've seen. They have an appendix devoted to a set of JavaScript coding styles. For someone overwhelmed with the actual design and implementation of a service, these nicely structured guidelines are actually a time-saver when learned and applied. There's no need to spend time rediscovering what others have already done (and likely as not having to refactor the mess to conform when you find out what they already knew).

The final area Mikowski and Powell talk about in the main section is actually something I hadn't considered part of the normal development process: design and adaptation to search engine and analytics services that crawl your site, and third-party caching services. This section was an eye opener for someone who's never worked with these except as a user or out of intellectual curiosity. In this section you learn some about how these services work and how to make your application friendly to them.

I don't think I'll be adopting this approach to application design, but what I learned here will certainly inform how I look at the systems I work on and how I build new ones.

Go in Practice

Matt Butcher and Matt Farina
Manning Publications, 2016, 288 pages
ISBN 978-1-63343-007-6

I'm familiar with Manning's "in Action" series and have actually reviewed *Go in Action* here. I was curious what would be different about *Go in Practice*. The cover notes that the book "includes 70 techniques." It turns out that "in practice" means this is a Go cookbook.

Most cookbooks I've read have spent most of the time on the shelf gathering dust. Either the recipes are for things that are either obvious or rare and obscure. I was pleasantly surprised by *Go in Practice*. Butcher and Farina have managed to create a reference for Go idiom and good practice. Given how quirky Go can be, especially for someone coming from a scripting language (I had some nostalgic flashbacks to my days coding C), a manual of good practice is a welcome find.

I think Butcher and Farina may have the same impression of the cookbook metaphor as I do. They avoid the term throughout the book, substituting "task" and "technique." The word "cookbook" is only used once, in what I suspect is an editorial description on the back cover. I'll use their terminology because I think what they present is better than a set of recipes.

Go in Practice is not a language reference. The authors do highlight some of the significant language features in the first section: multiple return values, dummy return values, goroutines, and channels. They also discuss package management, revision control, and Go's relationship to other popular languages.

The next section is one I particularly liked and will use often: a complete section on managing inputs and configuration for CLI programs. This has always seemed to me to be an overlooked part of most language teaching.

The full set of techniques covers things you'd expect—e.g., testing and debugging, Web service communications—but it also includes some things that I haven't seen in other places, such as aspects of coding for the cloud. This includes writing API interactions with cloud services which avoid lock in and how cloud-hosted programs can get VM information from the providers.

They close out the set of techniques with a section on code reflection and automatic code generation in Go. These are advanced techniques and probably shouldn't be used lightly. Most people will end up using annotations and tags for tasks like JSON or XML processing. However unlikely it is, they also show how to create new ones and then process them.

Each technique opens with a paragraph or two on the problem to be solved, then a brief description of the solution. The meat is in the discussion and code fragments that follow. The layout of the code is clean and contains clear annotations. I often try to read both the paper and ebook forms of the books I review. As much as I love to have bound paper on a shelf, the ebook has an edge in this case. The diagrams and code samples in the ebook have color graphics and highlighting which add an appeal that the black and white on paper can't match.

The ebook format is also well suited to handy access on tablets or browsers. For as long as I'm coding Go, I expect I'll keep *Go in Practice* close.