JavaScript Application Design

A Build First Approach

Nicolas Bevacqua

Foreword by Addy Osmani

MANNING

JavaScript Application Design

A Build First Approach

NICOLAS BEVACQUA

MANNING
SHELTER ISLAND

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact

Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964
Email: orders@manning.com

Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964

Development editor:	Susan Conant
Technical development editor:	Douglas Duncan
Copyeditor:	Katie Petito
Proofreader:	Alyson Brener
Technical proofreaders:	Deepak Vohra
	Valentin Crettaz
Typesetter:	Marija Tudor
Cover designer:	Marija Tudor

ISBN: 9781617291951
Printed in the United States of America
1 2 3 4 5 6 7 8 9 10 – EBM – 20 19 18 17 16 15

To Marian, for withstanding the birth of this book,
your unconditional love, and your endless patience.

I love you!

Will you marry me?

brief contents

contents

foreword

The process of designing a robust JavaScript web app has gone through a roaring renaissance in recent years. With the language being used to develop increasingly ambitious apps and interfaces, this is the perfect time for *JavaScript Application Design*. Through concise examples, lessons learned from the field, and key concepts for scalable development, Nico Bevacqua will give you a whirlwind tour of taking the process and design of your apps to the next level.

This book will also help you craft build processes that will save you time. Time is a key factor in staying productive. As web app developers, we want to make the most of ours, and a Build First philosophy can help us hit the ground running with clean, testable apps that are well structured from the get-go. Learning process workflow and how to manage complexity are fundamental cornerstones of modern JavaScript app development. Getting them right can make a massive difference in the long run.

JavaScript Application Design will walk you through automation for the front end. It covers everything from avoiding repetitive tasks and monitoring production builds to mitigating the cost of human error through a clean tooling setup. Automation is a big factor here. If you aren't using automation in your workflow today, you're working too hard. If a series of daily tasks can be accomplished with a single command, follow Nico's advice and spend the time you save improving the code quality of your apps.

Modularity is the final crucial concept that can assist with building scalable, maintainable apps. Not only does this help ensure that the pieces composing our application can be more easily tested and documented, it encourages reuse and focus on quality. In *JavaScript Application Design*, Nico expertly walks you through writing modu-

lar JavaScript components, getting asyncronous flow right, and enough client-side MVC for you to build an app of your own.

Strap on your seatbelts, adjust your command line, and enjoy a ride through the process of improving your development workflow.

ADDY OSMANI
SENIOR ENGINEER
WITH A PASSION FOR DEVELOPER TOOLING
GOOGLE

preface

Like most people in our field, I've always been fascinated with problem solving. The painful thrill of hunting for a solution, the exhilarating relief of having found a fix—there's nothing quite like it. When I was young I really enjoyed strategy games, such as chess, which I've played ever since I was a kid; StarCraft, a real-time strategy game I played for 10 years straight; and Magic: The Gathering, a trading card game that can be described as the intersection between poker and chess. They presented plenty of problem-solving opportunities.

At primary school I learned Pascal and rudimentary Flash programming. I was psyched. I would go on and learn Visual Basic, PHP, C, and start developing websites, reaping the benefits of a masterful handle on `<marquee>` and `<blink>` tags, paired with a modest understanding of MySQL; I was unstoppable, but my thirst for problem solving didn't end there, and I went back to gaming.

Ultima Online (UO), a massively multiplayer online role-playing game (no wonder they abbreviate that as MMORPG), wasn't any different than other games that got me hooked for years. Eventually I found out that there was an open source[1] implementation of the UO server, which was named RunUO and written entirely in C#. I played on a RunUO server where the administrators had no programming experience. They slowly started trusting me to handle minor bug fixes by literally emailing source code files back and forth. I was hooked. C# was a wonderful, expressive language, and the open source software for the UO server was amicable and inviting—you didn't even

[1] You can check out the RunUO website at runuo.com, although the project isn't maintained anymore.

need an IDE (or even need to know what that was) because the server would compile script files dynamically for you. You'd be essentially writing a file with 10 to 15 lines in it, inheriting from the `Dragon` class, and adding an intimidating text bubble over their head, or overriding a method so they'd spit more fireballs. You'd learn the language and its syntax without even trying, simply by having fun!

Eventually, a friend revealed that I could make a living out of writing C# code: "You know, people actually pay you to do that," he said. That's when I started developing websites again, except I wasn't using only Front Page and piles of `<marquee>` tags or Java applets for fun anymore. It still feels like a game to me, though.

A few years ago I read *The Pragmatic Programmer*[2], and something clicked. The book has a lot of solid advice, and I can't recommend it highly enough. One thing that particularly affected me: the authors advocate you get out of your comfort zone and try something you've been meaning to do but haven't gotten around to. My comfort zone was C# and ASP.NET at that point, so I decided to try Node.js, an unmistakably UNIX-y platform for JavaScript development on the server side, certainly a break from my Microsoft-ridden development experience so far.

I learned a ton from that experiment and ended up with a blog[3] where I'd write about everything I learned in the process. About six months later I'd decided that I'd put my years of experience in C# design into a book about JavaScript. I contacted Manning, and they jumped at the opportunity, helping me brainstorm and turn raw ideas into something more deliberate and concise.

This book is the result of many hours of hard work, dedication, and love for the web. In it, you'll find practical advice about application design, process automation, and best practices that will improve the quality of your web projects.

---

[2] *The Pragmatic Programmer: From Journeyman to Master* by Andrew Hunt and David Thomas (Addison Wesley, 1999) is a timeless classic you should seriously consider reading.

[3] You can read my blog, "Pony Foo," at ponyfoo.com. I write articles about the web, performance, progressive enhancement, and JavaScript.

acknowledgments

You wouldn't be holding this book in your hands if it weren't for everyone who supported and endured me throughout the writing process. I can only hope that those who deserve acknowledgment the most, my friends and family, already know that I can't put into words how thankful I am for their love, understanding, and frequent reassurance.

Many more people contributed—directly or indirectly—a great deal of wisdom and inspiration to this book.

The open source JavaScript community is an endless spring of insight, encouragement, and selfless contributions. They have opened my eyes to a better approach to software development, where collaboration isn't only possible, but actively encouraged. Most of these people have contributed indirectly by evangelizing for the web, maintaining blogs, sharing their experience and resources, and otherwise educating me. Others have contributed directly by developing tools discussed in the book. Among these individuals are Addy Osmani, Chris Coyier, Guillermo Rauch, Harry Roberts, Ilya Grigorik, James Halliday, John-David Dalton, Mathias Bynens, Max Ogden, Mikeal Rogers, Paul Irish, Sindre Sorhus, and T.J. Holowaychuk.

There are also many book authors and content distributors who have influenced and motivated me to become a better educator. Through their writing and sharing, these people have significantly helped shape my career. They include Adam Wiggins, Alan Cooper, Andrew Hunt, Axel Rauschmayer, Brad Frost, Christian Heilmann, David Thomas, Donald Norman, Frederic Cambus, Frederick Brooks, Jeff Atwood, Jeremy Keith, Jon Bentley, Nicholas C. Zakas, Peter Cooper, Richard Feynmann, Steve Krug, Steve McConnell, and Vitaly Friedman.

Susan Conant, my developmental editor at Manning, deserves to be singled out. She held this book to the greatest standard of quality I could possibly create, and it's in much better shape than it would've been if not for her. On top of that, she had to hand-hold me through the delicate and intimate process of writing my first book. Through her relentless, yet gentle, guidance she helped shape my lumps of ideas into a book that I'm not afraid to publish. I've become a better writer because of her, and I'm grateful for that.

She wasn't alone in that endeavor. All of the staff at Manning wanted this book to be the best that it could be. The publisher, Marjan Bace—along with his editorial collective—are to be thanked for that. Valentin Crettaz and Deepak Vohra, the technical proofreaders, were not only instrumental in ensuring the code samples were consistent and useful, but provided me with great feedback as well.

There are also the hordes of anonymous souls that were willing to read through the manuscript, leaving their impressions and helping improve the book. Thanks to the MEAP readers who posted corrections and comments in the Author Online forum, and to the reviewers who read the chapters at various stages of development: Alberto Chiesa, Carl Mosca, Dominic Pettifer, Gavin Whyte, Hans Donner, Ilias Ioannou, Jonas Bandi, Joseph White, Keith Webster, Matthew Merkes, Richard Harriman, Sandeep Kumar Patel, Stephen Wakely, Torsten Dinkheller, and Trevor Saunders.

Special thanks to Addy Osmani for contributing the foreword, and to everyone else who played a part. Even if they didn't make the keystrokes themselves, they played an instrumental role in getting this book published, and one step closer to you.

about this book

Web development has grown out of proportion, and today it's hard to imagine a world without the web. The web is famously fault tolerant. While traditional programming teaches us that missing a semicolon, forgetting to add a closing tag, or declaring invalid properties will have crippling consequences, the same cannot be said about the web. The web is a place where it's okay to make mistakes, yet there's increasingly less room for error. This dichotomy stems from the fact that modern web applications are an order of magnitude more complex than they used to be. During the humble beginnings of the web, we would maybe modestly make a minor change in web pages using JavaScript; whereas on the modern web, entire sites are rendered in a single page, powered by JavaScript.

JavaScript Application Design is your guide to a better modern web development experience, one where you can develop maintainable JavaScript applications as you would if you were using any other language. You'll learn how to leverage automation as a replacement for tedious and repetitive error-prone processes, how to design modular applications that are easy to test, and how to test them.

Process automation is a critical time-saver across the board. Automation in the development environment helps us focus our efforts on thinking, writing code, and debugging. Automation helps ensure our code works after every change that we publish to version control. It saves time when preparing the application for production by bundling, minifying assets, creating spritesheets, and adding other performance optimization techniques. It also helps with deployments by reducing risk and automating away a complicated and error-prone process. Many books discuss processes and

automation when it comes to back-end languages, but it's much harder to find material on the subject when it comes to JavaScript-driven applications.

The core value of *JavaScript Application Design* is quality. Automation gives you a better environment in which to build your application, but that alone isn't enough: the application itself needs to be quality conscious as well. To that end, the book covers application design guidelines, starting with a quick rundown of language-specific caveats, teaching you about the power of modularity, helping you untangle asynchronous code, develop client-side MVC applications, and write unit tests for your JavaScript code.

This book relies on specific tools and framework versions, as books about web technologies usually do, but it separates library-specific concerns from the theory at hand. This is a concession to the fact that tooling changes frequently in the fast-paced web development arena, but design and the processes behind tooling tend to have a much slower rhythm. Thanks to this separation of concerns, I hope this book stays relevant for years to come.

Road map

JavaScript Application Design is broken into two parts and four appendixes. The first part is dedicated to the Build First approach, what it is, and how it can aid your everyday job. This part covers process automation in detail, from everyday development to automated deployments, as well as continuous integration and continuous deployments; it spans 4 chapters.

- Chapter 1 describes the core principles that drive Build First, and the different processes and flows you can set up. It then introduces the application design guidelines that we'll discuss throughout the book and lays the foundation for the rest of the book.
- In chapter 2 you learn about Grunt, and how you can use it to compose build flows. Then we look at a few different build tasks that you can easily perform using Grunt.
- Chapter 3 is all about environments and the development workflow. You'll learn that not all environments are born the same, and how you can prioritize debugging and productivity in the development environment.
- Chapter 4 walks you through the release flow and discusses deployments. You'll learn about a few more build tasks that are geared toward performance optimization, and discover how to perform automated deployments. You'll also learn how to hook up continuous integration and how to monitor your application once in production.

While part 1 is focused on building applications using Grunt, appendix C teaches you to choose the best build tool for the job. Once you've read past part 1, you'll go into the second part of the book, which is dedicated to managing complexity in your application designs. Modules, MVC, asynchronous code flows, testing, and a well-designed API all play significant roles in modern applications and are discussed in the next chapters.

- Chapter 5 focuses on developing modular JavaScript. It starts by expressing what constitutes a module and how you can design applications modularly and lists the benefits of doing so. Afterward, you'll get a crash course on lexical scoping and related quirks in the JavaScript language. Later you get a rundown of the major ways to attain modularity: RequireJS, CommonJS, and the upcoming ES6 module system. The chapter concludes by going over different package management solutions such as Bower and npm.
- In chapter 6 you learn about asynchronous code flows. If you ever descend into callback hell, this may be your way out. This chapter discusses different approaches to deal with complexity in asynchronous code flows, namely callbacks, Promises, events, and ES6 generators. You'll also learn how to do proper error handling under each of those paradigms.
- Chapter 7 starts by describing MVC architectures, and then ties them specifically to the web. You'll learn how you can use Backbone to develop rich client-side applications that separate concerns using MVC. Later, you'll learn about Rendr, which can be used to render Backbone views on the server side, optimizing the performance and accessibility of your applications.
- In chapter 8, now that your applications are modular, clean-cut, and maintainable, you'll take the next logical step and look into testing your applications in different ways. To this end we'll go over an assortment of JavaScript testing tools and get hands-on experience using them to test small components. Then we'll go back to the MVC application built in chapter 7 and add tests to it. You won't be doing unit testing only, you'll also learn more about continuous integration, visual testing, and measuring performance.
- Chapter 9 is the last chapter of the book, and it's dedicated to REST API design. This is the layer where the client side interacts with the server, and it sets the scene for everything that we do in the application. If the API is convoluted and complicated, chances are the application as a whole will be as well. REST introduces clear guidelines when designing an API, making sure the API is concise. Last, we'll look at consuming these services in the client side in a conventional manner.

The appendixes can be read after you're done with the book, but you'll probably get the most value from them by reading them if you get stuck with the areas they cover, as they contain answers to questions you might have. Throughout the book, you'll be pointed to the appendixes where it makes sense to expand a little on one of these subjects.

- Appendix A is a soft introduction to Node.js and its module system, CommonJS. It'll help you troubleshoot your Node.js installation and answer a few questions on how CommonJS works.
- Appendix B is a detailed introduction to Grunt. Whereas the chapters in part I only explain what's absolutely necessary about Grunt, the appendix covers its inner workings in more detail, and will be handy if you're serious about developing a full-blown build process using Grunt.