JavaScript Frameworks for Modern Web Development

The Essential Frameworks, Libraries, and Tools to Learn Right Now

—

Second Edition

—

Sufyan bin Uzayr
Nicholas Cloud
Tim Ambler

Apress®

JavaScript Frameworks for Modern Web Development

The Essential Frameworks, Libraries, and Tools to Learn Right Now

Second Edition

Sufyan bin Uzayr
Nicholas Cloud
Tim Ambler

Apress®

JavaScript Frameworks for Modern Web Development

Sufyan bin Uzayr
Al Manama, United Arab Emirates

Nicholas Cloud
Florissant, MO, USA

Tim Ambler
Nashville, TN, USA

ISBN-13 (pbk): 978-1-4842-4994-9 ISBN-13 (electronic): 978-1-4842-4995-6
https://doi.org/10.1007/978-1-4842-4995-6

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Louise Corrigan
Development Editor: James Markham
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484249949. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

For Anza
—Sufyan bin Uzayr

Table of Contents

About the Authors

Sufyan bin Uzayr is a web developer with over 10 years of experience in the industry. He specializes in a wide variety of technologies, including JavaScript, WordPress, Drupal, PHP, and UNIX/Linux shell and server management, and is the author of five previous books. Sufyan is the Director of Parakozm, a multinational design and development consultancy firm that offers customized solutions to a global clientele. He is also the CTO at Samurai Servers, a server management and security company catering mainly to enterprise-scale audience. He takes a keen interest in technology, politics, literature, history, and sports, and in his spare time he enjoys teaching coding and English to students. Read more about his works at `www.sufyanism.com`.

Nicholas Cloud is a software developer who lives in the very humid city of St. Louis. For over a decade, he has forged his skills into a successful career. He has developed web applications, web services, and desktop software on diverse platforms with JavaScript, C#, and PHP. A strong proponent of open source software, Nicholas contributes to userland projects and has written several of his own open source libraries. He speaks at a variety of user groups and conferences and writes books, technical articles, and blog posts in his spare time. He opines on Twitter at `@nicholascloud`.

Tim Ambler is a software engineer from Nashville, Tennessee. His passion for programming follows in the footsteps of his father, who introduced him to computers at a young age with a Commodore 64. Tim is the author of several popular open source projects, one of which (whenLive) has been featured by GitHub's staff. An occasional conference speaker and frequent writer, Tim has been referenced multiple times in online publications such as *JavaScript Weekly* and *Node Weekly*. He currently lives in the 12 South area with his wife, Laura, and two cats. You can follow him on Twitter at `@tkambler`.

About the Technical Reviewer

Aleemullah Samiullah is a seasoned developer with over 10+ years of experience in front-end technologies. He is a senior engineer at Software AG and has previously worked at Publicis Sapient and Infosys, gaining considerable expertise, working with major brands such as Tesco, Target, and Holt Renfrew. He enjoys crafting digital experiences based on human-centered design, with a keen focus on usability and accessibility. In his career, Aleem has used various JavaScript libraries, including React, Angular, Backbone, ExtJS, jQuery, and so on. He is passionate about the latest technologies and actively participates in JS meetups and conferences. When he's not coding, he likes to travel, write blog posts, and spend time with family.

Aleem can be found on LinkedIn at `www.linkedin.com/in/aleemullah/`.

Acknowledgments

There are several people who deserve to be on this page because this book would not have come into existence without their support. That said, some names deserve a special mention, and I am genuinely grateful to

- My mom and dad, for everything they have done for me
- Faisal Fareed and Sadaf Fareed, my siblings, for helping with things back home
- Nancy Chen, Content Development Editor for this book, for keeping track of everything and for being very patient as I kept missing one deadline or the other
- The Apress team, especially Louise Corrigan, Jade Scard, and James Markham, for ensuring that the book's content, layout, formatting, and everything else remains perfect throughout
- The coauthors of this book's first edition and the tech reviewer, for going through the manuscript and providing his insight and feedback
- Typesetters, cover designers, printers, and everyone else, for their part in the development of this book
- All the folks associated with Parakozm, either directly or indirectly, for their help and support
- The JavaScript community at large, for all their hard work and efforts

—Sufyan bin Uzayr

Introduction

> *They tell me we're living in an information age, but none of it seems to be the information I need or brings me closer to what I want to know. In fact (I'm becoming more and more convinced) all this electronic wizardry only adds to our confusion, delivering inside scoops and verdicts about events that have hardly begun: a torrent of chatter moving at the speed of light, making it nearly impossible for any of the important things to be heard.*
>
> —Matthew Flaming, *The Kingdom of Ohio*

The notion that "technology moves quickly" is a well-worn aphorism, and with good reason: technology does move quickly. But at this moment, JavaScript in particular is moving very quickly indeed—much like that "torrent of chatter moving at the speed of light" that Matthew Flaming refers to in *The Kingdom of Ohio*. The language is in the midst of what many have called a renaissance, brought about by the rapidly increasing sophistication of browser-based applications and the rising popularity of JavaScript on the server, thanks to Node.js.

An almost feverish pace of innovation is occurring within the JavaScript community that, while endlessly fascinating to follow, also presents some unique challenges of its own. JavaScript's ecosystem of libraries, frameworks, and utilities has grown dramatically. Where once a small number of solutions for any given problem existed, many can now be found, and the options continue to grow by the day. As a result, developers find themselves faced with the increasingly difficult task of choosing the appropriate tools from among many seemingly good options.

If you've ever found yourself wondering why JavaScript seems to be attracting so much attention, as we have, it's worth stopping for a moment to consider the fact that JavaScript, a language that was created by one person in 10 days, now serves as the foundation upon which much of the Web as we know it sits. A language that was originally created to solve relatively simple problems is now being applied in new and innovative ways that were not originally foreseen. What's more, JavaScript is a beautifully expressive language, but it's not without its share of rough edges and potential pitfalls.

While flexible, efficient, and ubiquitous, JavaScript concepts such as the event loop and prototypal inheritance can prove particularly challenging for those coming to the language for the first time.

For these and many other reasons, the development community at large is still coming to terms with how best to apply the unique features that JavaScript brings to the table. We've no doubt only scratched the surface of what the language and the community behind it are capable of. For those with an insatiable appetite for knowledge and a desire to create, now is the perfect time to be a JavaScript developer.

We have written *JavaScript Frameworks for Modern Web Development* to serve as your guide to a wide range of popular JavaScript tools that solve difficult problems at both ends of the development stack: in the browser and on the server. The tutorials and downloadable code examples contained within this book illustrate the usage of tools that manage dependencies, structure code in a modular fashion, automate repetitive build tasks, create specialized servers, structure client-side applications, facilitate horizontal scaling, perform event logging, and interact with disparate data stores.

The libraries and frameworks covered include Grunt, Yeoman, PM2, RequireJS, Browserify, Knockout, Angular, Kraken, Mongoose, Knex, Bookshelf, Async.js, Underscore, Lodash, React, and Vue.js.

In writing *JavaScript Frameworks for Modern Web Development*, our goal was to create a filter for the "torrent of chatter" that often seems to surround JavaScript and, in so doing, to allow what we believe are some important things to be heard. We hope the information contained within these pages proves as useful to you as it has to us.

Who This Book Is For

This book is intended for web developers who are already confident with JavaScript, but also frustrated with the sheer number of solutions that exist for seemingly every problem. This book helps lift the fog, providing the reader with an in-depth guide to specific libraries and frameworks that well-known organizations are using right now with great success. Topics pertaining to both client-side and server-side development are covered. As a result, readers will gain the most benefit from this book if they already have at least an intermediate familiarity with both the web browser Document Object Model (DOM), common client-side libraries like jQuery, and Node.js.

How This Book Is Structured

This book covers a wide selection of JavaScript tools that are applicable throughout the entire development process, from a project's first commit to its first release and beyond. To that end, the chapters have been grouped into the following parts.

Part 1: Development Tools

Grunt

Larry Wall, the creator of Perl, describes the three virtues of a great programmer as laziness, impatience, and hubris. In this chapter, we'll focus on a tool that will help you strengthen the virtue of laziness—Grunt. This popular task runner provides developers with a framework for creating command-line utilities that automate repetitive build tasks such as running tests, concatenating files, compiling SASS/LESS stylesheets, checking for JavaScript errors, and more. After reading this chapter, you'll know how to use several popular Grunt plugins as well as how to go about creating and sharing your own plugins with the community.

Yeoman

Yeoman provides JavaScript developers with a mechanism for creating reusable templates ("generators") that describe the overall structure of a project (initially required dependencies, Grunt tasks, etc.) in a way that can be easily reused over and over. Broad community support also allows you to take advantage of a wide variety of preexisting templates. In this chapter, we'll walk through the process of installing Yeoman and using several popular preexisting generators. Finally, we'll take a look at how we can create and share our own templates with the community.

PM2

In this chapter, we will close out our discussion of development tools by taking a look at PM2, a command-line utility that simplifies many of the tasks associated with running Node applications, monitoring their status, and efficiently scaling them to meet increasing demand.

Part 2: Module Loaders

RequireJS and Browserify

JavaScript lacks a native method for loading external dependencies in the browser—a frustrating oversight for developers. Fortunately, the community has stepped in to fill this gap with two very different and competing standards: the Asynchronous Module Definition (AMD) API and CommonJS. We'll dive into the details of both and take a look at widely used implementations of each: RequireJS and Browserify. Each has its merits, which we'll discuss in detail, but both can have a profoundly positive impact on the way in which you go about structuring your applications.

Part 3: Client-Side Frameworks

Knockout and Angular

In recent years, web developers have witnessed a sharp rise in popularity of so-called "single-page apps." Such applications exhibit behavior once available only on the desktop, but at the expense of increased code complexity within the browser. In this section, we'll dive into two widely used front-end frameworks that help minimize that complexity by providing proven patterns for solving frequently encountered problems: Knockout and Angular. Knockout focuses on the relationship between view and data, but otherwise leaves the application architecture and plumbing to the developer's discretion. Angular takes a more prescriptive approach, covering the view, data transfer, Dependency Injection, and so on.

Part 4: Server-Side Frameworks

Kraken

Client-side applications aren't very useful without a server with which to interact. In this chapter, we'll take a look at one popular framework that supports developers in the creation of back-end applications: Kraken.

Part 5: Managing Database Interaction

Mongoose, Knex, and Bookshelf

At the core of every application lies the most important component of any development stack—the data that our users seek. In this section, we'll become familiar with two libraries that help simplify some of the complexity that's often experienced when interacting with popular storage platforms such as MongoDB, MySQL, PostgreSQL, and SQLite. After reading this section, you'll be comfortable defining schemas, associations, lifecycle "hooks," and more.

Part 6: Managing Control Flow

Async.js

The asynchronous nature of JavaScript provides developers with a significant degree of flexibility—as opposed to forcing developers to execute their code in a linear fashion, JavaScript allows developers to orchestrate multiple actions simultaneously. Unfortunately, along with this flexibility comes a significant degree of additional complexity—what many developers refer to as "callback hell" or the "pyramid of doom."

Part 7: Further Useful Libraries

A number of wonderfully useful libraries exist that this book would be remiss not to cover, but for which additional parts are not necessarily warranted. This part will cover such libraries.

Underscore and Lodash

Underscore (and its successor, Lodash) is an incredibly useful collection of functions that simplifies many frequently used patterns that can be tedious to implement otherwise. This brief chapter will bring these libraries to your attention, along with some of the more popular extensions that can also be included to enhance their usefulness even further. Examples are included that highlight some of the most frequently used portions of these libraries.

Part 8: Front-End Development

React and Vue.js

In this section, we will cover JavaScript frameworks that are geared for front-end development, such as React and Vue.js.

React, having the backing of Facebook, has risen in popularity in a very short span of time and continues to be a preferred choice for many developers.

On the other hand, Vue.js is a slightly less popular name in the field, but it has been gaining a steady and very loyal following, primarily due to its ease of use and simplicity.

Downloading the Code

Each chapter in this book contains many examples, the source code for which may be downloaded from `www.apress.com/9781484249949` in zipped form.

Most examples are run with the Node.js runtime, which may be obtained from `https://nodejs.org`. Chapters with additional prerequisites will explain the necessary procedures for downloading and installing the examples. (For example, MongoDB is necessary to run examples in Chapter 9, which covers Mongoose.)

Any additional steps necessary for running code examples (e.g., executing curl requests) or interacting with a running example (e.g., opening a web browser and navigating to a specific URL) are explained alongside each listing.

PART I

Development Tools

CHAPTER 1

Grunt

> *I'm lazy. But it's the lazy people who invented the wheel and the bicycle because they didn't like walking or carrying things.*
>
> —Lech Walesa, former president of Poland

In his book *Programming Perl*, Larry Wall (the well-known creator of the language) puts forth the idea that all successful programmers share three important characteristics: laziness, impatience, and hubris. At first glance, these traits all sound quite negative, but dig a little deeper, and you'll find the hidden meaning in his statement:

> Laziness: Lazy programmers hate to repeat themselves. As a result, they tend to put a lot of effort into creating useful tools that perform repetitive tasks for them. They also tend to document those tools well, to spare themselves the trouble of answering questions about them later.
>
> Impatience: Impatient programmers have learned to expect much from their tools. This expectation teaches them to create software that doesn't just react to the needs of its users, but that actually attempts to anticipate those needs.
>
> Hubris: Good programmers take great pride in their work. It is this pride that compels them to write software that others won't want to criticize—the type of work that we should all be striving for.

In this chapter, we'll focus on the first of these three characteristics, laziness, along with Grunt, a popular JavaScript "task runner" that supports developers in nurturing this trait by providing them with a toolkit for automating the repetitive build tasks that often accompany software development, such as

S. bin Uzayr et al., *JavaScript Frameworks for Modern Web Development*,
https://doi.org/10.1007/978-1-4842-4995-6_1

- Script and stylesheet compilation and minification
- Testing
- Linting
- Database migrations
- Deployments

In other words, Grunt helps developers who strive to work smarter, not harder. If that idea appeals to you, read on. After you have finished this chapter, you will be well on your way toward mastering Grunt. You'll learn how to do the following in this chapter:

- Create configurable tasks that automate the repetitive aspects of software development that accompany nearly every project
- Interact with the file system using simple yet powerful abstractions provided by Grunt
- Publish Grunt plugins from which other developers can benefit and to which they can contribute
- Take advantage of Grunt's preexisting library of community-supported plugins

Installing Grunt

Before continuing, you should ensure that you have installed Grunt's command-line utility. Available as an npm package, the installation process is shown in Listing 1-1.

Listing 1-1. Installing the `grunt` Command-Line Utility via `npm`

```
$ npm install -g grunt-cli
$ grunt –version
grunt-cli v1.3.2
```

How Grunt Works

Grunt provides developers with a toolkit for creating command-line utilities that perform repetitive project tasks. Examples of such tasks include the minification of JavaScript code and the compilation of Sass stylesheets, but there's no limit to how Grunt

can be put to work. Grunt can be used to create simple tasks that address the specific needs of a single project—tasks that you don't intend to share or reuse—but Grunt's true power derives from its ability to package tasks as reusable plugins that can then be published, shared, used, and improved upon by others.

Four core components make Grunt tick, which we will now cover.

Gruntfile.js

At Grunt's core lies the *Gruntfile*, a Node module saved as `Gruntfile.js` (see Listing 1-2) at the root of your project. It's within this file that we can load Grunt plugins, create our own custom tasks, and configure them according to the needs of our project. Each time Grunt is run, its first order of business is to retrieve its marching orders from this module.

Listing 1-2. Sample Gruntfile

```
// example-starter/Gruntfile.js

module.exports = function(grunt) {

    /**
     * Configure the various tasks and plugins that we'll be using
     */
    grunt.initConfig({
        /* Grunt's 'file' API provides developers with helpful abstractions for
        interacting  with the file system. We'll take a look at these in
        greater detail later in the chapter. */
        'pkg': grunt.file.readJSON('package.json'),
        'uglify': {
            'development': {
                'files': {
                    'build/app.min.js': ['src/app.js', 'src/lib.js']
                }
            }
        }
    });
```

```
    /**
     * Grunt plugins exist as Node packages, published via npm. Here, we
       load the
     * 'grunt-contrib-uglify' plugin, which provides a task for merging and
        minifying
     * a project's source code in preparation for deployment.
     */
    grunt.loadNpmTasks('grunt-contrib-uglify');

    /**
     * Here we create a Grunt task named 'default' that does nothing more
       than call
     * the 'uglify' task. In other words, this task will serve as an alias to
     * 'uglify'. Creating a task named 'default' tells Grunt what to do
       when it is
     * run from the command line without any arguments. In this example,
       our 'default'
     * task calls a single, separate task, but we could just as easily have
       called
     * multiple tasks (to be run in sequence) by adding multiple entries to
       the array
     * that is passed.
     */
    grunt.registerTask('default', ['uglify']);

    /**
     * Here we create a custom task that prints a message to the console
       (followed by
     * a line break) using one of Grunt's built-in methods for providing
       user feedback.
     * We'll look at these in greater detail later in the chapter.
     */
    grunt.registerTask('hello-world', function() {
        grunt.log.writeln('Hello, world.');
    });

};
```

Tasks

Tasks are the basic building blocks of Grunt and are nothing more than functions that are registered with assigned names via Grunt's `registerTask()` method. In Listing 1-2, a simple `hello-world` task is shown that prints a message to the console. This task can be called from the command line as shown in Listing 1-3.

Listing 1-3. Running the `hello-world` Task Shown in Listing 1-2

```
$ grunt hello-world
Running "hello-world" task
Hello, world.

Done, without errors.
```

Multiple Grunt tasks can also be run in sequence with a single command, as shown in Listing 1-4. Each task will be run in the order in which it was passed.

Listing 1-4. Running Multiple Grunt Tasks in Sequence

```
$ grunt hello-world uglify
Running "hello-world" task
Hello, world.

Running "uglify:development" (uglify) task
>> 1 file created.

Done, without errors.
```

The `hello-world` task that we've just seen serves as an example of a basic, stand-alone Grunt task. Such tasks can be used to implement simple actions specific to the needs of a single project that you don't intend to reuse or share. Most of the time, however, you will find yourself interacting not with stand-alone tasks, but instead with tasks that have been packaged as Grunt plugins and published to npm so that others can reuse them and contribute to them.

Plugins

A Grunt plugin is a collection of configurable tasks (published as an npm package) that can be reused across multiple projects. Thousands of such plugins exist. In Listing 1-2, Grunt's `loadNpmTasks()` method is used to load the `grunt-contrib-uglify` Node module, a Grunt plugin that merges a project's JavaScript code into a single, minified file that is suitable for deployment.

Note A list of all available Grunt plugins can be found at `http://gruntjs.com/plugins`. Plugins whose names are prefixed with `contrib-` are officially maintained by the developers behind Grunt. In the repository, officially maintained plugins are now also marked by a "star" icon, making them easier to differentiate from third-party developers' plugins.

Configuration

Grunt is known for emphasizing "configuration over code": the creation of tasks and plugins whose functionality is tailored by configuration that is specified within each project. It is this separation of code from configuration that allows developers to create plugins that are easily reusable by others. Later in the chapter, we'll take a look at the various ways in which Grunt plugins and tasks can be configured.

Adding Grunt to Your Project

Earlier in the chapter, we installed Grunt's command-line utility by installing the `grunt-cli` npm package as a global module. We should now have access to the `grunt` utility from the command line, but we still need to add a local `grunt` dependency to each project we intend to use it with. The command to be called from within the root folder of your project is shown next. This example assumes that npm has already been initialized within the project and that a `package.json` file already exists.

```
$ npm install grunt --save-dev
```

Our project's `package.json` file should now contain a `grunt` entry similar to that shown in Listing 1-5.