Community Experience Distilled

Build amazing high-performance web applications using Backbone.js

Backbone.js Essentials

Jeremy Walker

What you will learn from this book

- Architect a single-page web application using Backbone.js
- Easily transmit data to and from a server-side API (RESTful or not) using Backbone's Models and Classes
- Add and modify DOM elements with Backbone Views, and create browsable client-side pages with Routers
- Document your Backbone.js logic, with or without popular tools such as JSDoc or Doxco, to ensure long-term maintainability
- Prevent bugs and ease refactoring by testing your code with frameworks such as QUnit, Mocha, or Buster and auxiliary tools such as Sinon
- Use Backbone.js's sister library, Underscore, to realize the full power of a Backbone application
- Explore advanced Backbone techniques and save time and effort by using the most powerful third-party tools available

Who this book is written for

If you are a developer with baseline JavaScript proficiency and are familiar with the jQuery library, then this book is ideal for you. Whether you’ve tried building complex web applications before and been frustrated by the challenge of doing so without the proper tools, or whether you’ve only built simple websites and are now looking to create full-featured web applications, this book has everything you need to get ahead of the curve.


$ 24.99 US
£ 16.99 UK

Prices do not include local sales tax or VAT where applicable

Jeremy Walker

Backbone.js Essentials

Build amazing high-performance web applications using Backbone.js


Backbone.js Essentials

Jeremy Walker

What you will learn from this book

- Architect a single-page web application using Backbone.js
- Easily transmit data to and from a server-side API (RESTful or not) using Backbone's Models and Classes
- Add and modify DOM elements with Backbone Views, and create browsable client-side pages with Routers
- Document your Backbone.js logic, with or without popular tools such as JSDoc or Doxco, to ensure long-term maintainability
- Prevent bugs and ease refactoring by testing your code with frameworks such as QUnit, Mocha, or Buster and auxiliary tools such as Sinon
- Use Backbone.js's sister library, Underscore, to realize the full power of a Backbone application
- Explore advanced Backbone techniques and save time and effort by using the most powerful third-party tools available

Who this book is written for

If you are a developer with baseline JavaScript proficiency and are familiar with the jQuery library, then this book is ideal for you. Whether you’ve tried building complex web applications before and been frustrated by the challenge of doing so without the proper tools, or whether you’ve only built simple websites and are now looking to create full-featured web applications, this book has everything you need to get ahead of the curve.


Backbone.js Essentials

Jeremy Walker

What you will learn from this book

- Architect a single-page web application using Backbone.js
- Easily transmit data to and from a server-side API (RESTful or not) using Backbone's Models and Classes
- Add and modify DOM elements with Backbone Views, and create browsable client-side pages with Routers
- Document your Backbone.js logic, with or without popular tools such as JSDoc or Doxco, to ensure long-term maintainability
- Prevent bugs and ease refactoring by testing your code with frameworks such as QUnit, Mocha, or Buster and auxiliary tools such as Sinon
- Use Backbone.js's sister library, Underscore, to realize the full power of a Backbone application
- Explore advanced Backbone techniques and save time and effort by using the most powerful third-party tools available

Who this book is written for

If you are a developer with baseline JavaScript proficiency and are familiar with the jQuery library, then this book is ideal for you. Whether you’ve tried building complex web applications before and been frustrated by the challenge of doing so without the proper tools, or whether you’ve only built simple websites and are now looking to create full-featured web applications, this book has everything you need to get ahead of the curve.

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Building a Single-Page Site Using Backbone'
- A synopsis of the book’s content
- More information on Backbone.js Essentials

About the Author

Jeremy Walker is a writer and programmer who was born, raised, and presently resides in Silicon Valley. Jeremy currently works as a Staff Software Engineer at Syapse, where he develops the company's precision medicine platform and helps doctors treat serious medical conditions using genetic sequencing. Prior to Syapse, Jeremy worked in the multifamily tenant industry, where he drafted multiple XML standards while serving as Technical Vice Chair of the industry's standards committee.

In his free time, Jeremy enjoys reading, playing old computer games, and answering questions on Stack Overflow. If the weather is nice, he can often be found hiking or just lying on the beach in neighboring Santa Cruz. Jeremy is also the author of two programming libraries: Underscore Grease and BackSupport, and he is an active proponent of all things Backbone-related.
Backbone.js Essentials

We JavaScript developers live in exciting times. Every month we're presented with an incredible new browser feature, open source library, or software methodology that promises to vastly improve on whatever came before it. And then, just as soon as one new technology arrives, another almost inevitably follows it, promising still further innovation.

Amid this ever-changing landscape of new technologies only a very few libraries manage to stay not just relevant but essential over a long period of time, despite having newer challengers. For many web programmers jQuery was the first such library, but in recent years another library has proven itself to be similarly indispensable. That library is Backbone.

Backbone offers developers a wide range of foundational pieces from which they can build any manner of web application. From its simple yet flexible class system, to its event-based and AJAX-simplifying data containers, to its DOM manipulation and single-page user-facing components, Backbone provides everything needed to form the underlying framework of a site.

However, at first Backbone's incredible power and flexibility can be intimidating. When faced with over a hundred different methods spread across four base classes it can be challenging for a new Backbone developer to determine which ones to use and when. In addition, while Backbone is very "opinionated" about its core functions, it takes a deliberately agnostic approach to just about everything else. This allows developers to choose the perfect approach for their application, but at the same time all those choices can be daunting to someone unfamiliar with Backbone.

In this book we offer two things, both drawn from years of experience using Backbone to create and maintain a real-world web-based application. First, we will provide you with an understanding of all of Backbone's essentials. By the time you complete this book, you will have learned everything you need to create powerful and maintainable web applications using Backbone.

But at the same time, in addition to just explaining Backbone itself, we'll also explain the wider, "meta" level of Backbone programming. From advice on how to implement important details of your classes, to examinations of how Backbone can be made even more powerful when connected to the larger JavaScript ecosystem, we have endeavored to not only show how to use Backbone, but how to use it well.

Welcome to Backbone Essentials.
What This Book Covers

Chapter 1, Building a Single-Page Site Using Backbone, introduces Backbone and explains why it is such a popular choice of framework.

Chapter 2, Object-Oriented JavaScript with Backbone Classes, details Backbone's class system, a significant improvement over JavaScript's native system.

Chapter 3, Accessing Server Data with Models, begins our exploration of Backbone's data management, event listening, and AJAX capabilities with Backbone's Model class.

Chapter 4, Organizing Models with Collections, continues our exploration of Backbone's data management capabilities, only this time using multiple data sets with the Collection class.

Chapter 5, Adding and Modifying Elements with Views, examines Backbone's DOM rendering and event-handling View class.

Chapter 6, Creating Client-side Pages with Routers, introduces the heart of Backbone's single-page architecture, the Router class.

Chapter 7, Fitting Square Pegs in Round Holes — Advanced Backbone Techniques, explores advanced Backbone patterns for solving tricky problems.

Chapter 8, Scaling Up — Ensuring Performance in Complex Applications, looks at the causes of performance issues in Backbone applications, and how to prevent them.

Chapter 9, What Was I Thinking? Documenting Backbone Code, considers different strategies for documenting your application, including JSDoc and Doxco.

Chapter 10, Keeping the Bugs Out — How to Test a Backbone Application, recommends best practices for testing a Backbone application, with examples from the Mocha and Sinon libraries.

Chapter 11, (Not) Re-Inventing the Wheel — Utilizing Third-Party Libraries, previews a variety of 3rd party libraries, some Backbone-specific and some not, which can benefit a Backbone application.

Chapter 12, Summary and Further Reading, looks back on the topics previously discussed and considers their application in the real world.
Building a Single-Page Site Using Backbone

In this chapter, you’ll learn both what Backbone is and why you will want to use it to create web applications. In particular, we’ll examine the following topics:

- Backbone's history and how it fits into the larger history of web development
- The advantages of Backbone's single-page architecture
- How real-world companies are using Backbone to power their sites

What is Backbone?

Created in 2010 by Jeremy Ashkenas, Backbone is a part of an entirely new breed of JavaScript libraries. Depending on who you ask, this type of library can be referred to as a rich application framework, a single page library, a thick client library, or just a JavaScript framework. Whatever you choose to call them, Backbone and its related libraries, such as Angular, Ember, and CanJS, provide tools that can be used to build websites that are so powerful that they go beyond being mere sites and become full-fledged web applications.

Backbone is made up of the following five major tools:

- A class system, which makes it easy to practice object-oriented programming
- A Model class, which allows you to store and manipulate any kind of data as well as exchange this data with and from your remote server using AJAX
- A Collection class, which allows you to perform the same data manipulation and transmission but on groups of Models instead
• A View class, which can be used both to render the DOM elements that make up the page and to manage any user interactions that occur on them

• A Router class, which enables you to create an entire site, with any number of virtual pages, using only a single HTML file

While conceptually very simple, together these components allow you to create websites with a level of sophistication and robustness previously unseen on the World Wide Web (WWW).

Why choose Backbone?

The question of why you will want to choose Backbone for your project really has two parts. First, there's the question of "Why use a rich application framework at all?" and second, there's the question of "Why choose Backbone over its alternatives?" Let's start with the first question.

To truly appreciate the value of a single-page application (SPA), it's essential to understand what came before. All previous websites can be grouped into three major categories, which I'll call static, server-based, and JavaScript-assisted. Each of these types correlates with a different era in web development history.

A Brief History of Web Development

In many ways, the history of web development can be seen as a progression from server-based logic to client-based logic. The story begins in 1993, with the introduction of the world's first real web browser: Mosaic. At that time, the Web didn't even have JavaScript (or CSS, for that matter), just HTML. In those early days, most sites were simple static sites, and any site with dynamic elements had to be entirely server based. The very first version of JavaScript would only be introduced two years later at the end of 1995, and it would take several more years before the language was useful for anything other than simple form validation.
Yahoo!'s JavaScript-less website in 1997

Luckily, the Web did evolve, and soon JavaScript developers witnessed the birth of a whole new wave of JavaScript libraries, such as Dojo, MochiKit, YUI, and, of course, jQuery. These libraries allowed developers to easily manipulate the DOM, avoid the rampant cross-browser issues of the time, and take advantage of a newly introduced technology known as **AJAX**. In other words, they enabled developers to create a new type of site, the JavaScript-assisted but still largely server-based web application.

Even with these advances, the server still maintained control over two critical pieces of a site's infrastructure: navigation and page rendering. This problem wouldn’t be solved until years later, with the introduction of the modern generation of JavaScript frameworks, the first and popular one being Backbone. Using Backbone, web developers were finally able to control an entire site using only the client-side technologies of JavaScript, HTML, and CSS, which meant that they could create an entirely new type of web application, the thick client or single page site.
Today, even with the advent of Backbone and related libraries, many developers still continue to create the three previous types of site, which is perfectly reasonable as long as their goals are modest. In other words, if you simply want to show off your wedding photos to friends, then you probably don't need the full power of Backbone. However, if your goal is to build a powerful and robust web application, then the advantages of a Backbone-powered site are clear.

Benefits of Backbone and single-page applications

While there are numerous benefits of adopting thick client architecture for a site, they can be grouped into three main categories: asset control, easier data management, and improved performance.

Full User Interface Asset Control

One of the challenges of developing a traditional multipage website is the sharing of HTML assets. On such a site, the HTML is generated using server-side tools, such as Django templates, ERBs, or JavaServer Pages (JSPs) but, of course, the client-side logic also depends heavily on that same HTML. In smaller organizations, this means that programmers frequently have to divide their focus between JavaScript and a server-side language, which can be frustrating due to the frequent context switching.

In large organizations where teams are separate, the HTML assets are usually managed by the server team. This sometimes makes it difficult for the client team to even make the most basic changes to the site's HTML, as they have to work across the aisle. When they fail to do so, often the result is such that they create parallel versions of the server team's work instead, with such duplication inevitably resulting in bugs.

Backbone-powered thick client applications solve these problems by leaving the site's HTML firmly under the control of the client team, either in the form of a template system, raw HTML files, or in DOM-manipulation JavaScript logic. Any interactions between the two teams happen through a carefully negotiated set of APIs, allowing both groups to focus on their core specialties without stepping on each others toes.
Simpler Data Management and Event Triggers
As an application scales, it may become difficult to manage the interactions between its various components. One powerful approach to solve this problem is to use event-based control systems, but before Backbone, such systems were rarely found in JavaScript. True, DOM events have long been a part of web development, but without a framework such as Backbone, developers have been limited to just the user-generated events. To truly realize the power of an event-based system, you also need data-driven events, which are an integral part of Backbone.

Another common scaling challenge comes from JavaScript's lack of support for object-oriented programming (OOP). OOP allows programmers to organize large, complicated logic into smaller, more manageable classes and is very useful when growing an application. While JavaScript has a built-in class system, it is fairly unconventional and often discourages developers from employing OOP techniques. Backbone solves this problem by providing a more friendly system that, while still built within the limits of the JavaScript language, looks much closer to what you'd find in a solid OOP language, such as Java.

Enhanced performance
On the Web, speed is paramount, and one significant factor in a site's speed is the weight of its HTML files. In a multipage application, every time the user visits a new page, their browser has to send a request and wait for a response from the server. When the response comes back, it doesn't just contain a unique HTML for that page. Instead, the response contains the HTML for everything, including any common site components such as menus or footers. When the user visits the next page, they once again have to download that same common component HTML, even if it hasn't changed.

Moreover, that's not the only redundant HTML downloaded: multiple rows in a table, multiple search results in a list, or any other repeated content also has to have its HTML downloaded multiple times. For instance, consider the following HTML:

```html
<tr>
  <td>Fake Book</td>
  <td>This is a description of a fake book</td>
  <td><a href="/buy/book1">Buy Fake Book</a></td>
</tr>
<tr>
  <td>Another Fake Book</td>
  <td>I hope you like fake book titles because plenty more are coming in future chapters...</td>
  <td><a href="/buy/book2">Buy Another Fake Book</a></td>
</tr>
```
Only the names, descriptions, and URLs of the two books are unique in the preceding code, but even so all of the nonunique parts of the code have to be downloaded with it. If the site shows 50 books, the user downloads 50 copies of the book row HTML. Even when a site has no common components or repeated elements, there's still a performance cost when the user visits a new page because the browser has to go through an entire request-response cycle and then reload and redraw the page, all of which takes time.

In a single-page application, none of this is an issue. The site's foundation HTML is downloaded only once, and after that, all page transitions happen entirely through JavaScript. Since the client knows how to render both common and repeated components, there's no need to download any HTML for them at all. On a Backbone site, the server sends only the unique data via AJAX, and if there is no unique data to download, the user can progress without making a single new request to the server.

**Backbone and Its Competitors**

Many of the advantages we've just discussed apply to any single-page application, not just a Backbone one. This means that you can achieve many of those benefits even if you use one of Backbone's competing libraries, such as Ember or Angular. Whether you've considered using these frameworks or not, you're probably at least wondering, "Will Backbone provide me with everything I need to build my site, both now and in the future?"

The first thing to consider when answering this question is whether or not Backbone has an active community and will continue to be actively developed. Backbone users can feel safe in this regard: at the time of writing this book, Backbone's GitHub page had more than 1,500 watchers and more than 21,000 stars, beating its next closest competitor (Ember) by more than 400 watchers and 7,000 stars. Other frameworks such as CanJS and Google's Angular have even less interest on GitHub. While this certainly doesn't make Backbone better than those libraries, it shows the strength of its community and should provide you with the assurance that Backbone will be around for many years to come.

Another reason to feel confident when selecting Backbone is that it only tries to do a specific set of tasks, leaving everything else to external libraries. This means that if you find a better template system, dependency management tool, or any other library in the future, you can easily switch to using it. Other frameworks tightly couple things such as their template systems to their framework, leaving you with less options in the future.
However, perhaps the biggest indicator of Backbone's vitality is the companies that are already using it to accomplish amazing things. Companies as varied as USA Today, Pandora, Hulu, Gawker Media, AirBnB, Khan Academy, Groupon, and even Walmart use Backbone to create powerful web applications. If Backbone is powerful enough to support these major companies, it's almost certain to be powerful enough for your project.

There's one other company that uses Backbone, which is the company that I work for—Syapse. At Syapse, we've built a precision medicine data platform that helps hospitals receive genetic data in a structured format, pull patients' clinical data from a variety of internal health IT systems, and present this data together in an interactive web application. Through this interface, physicians see their patients' genetic and clinical data in context, enabling them to choose the most effective drugs possible tailored to a patient's own genetic profile.

Creating an application like Syapse did isn't easy, and with serious diseases such as cancer on the line, there's little room for error. However, using Backbone, Syapse has managed to grow from just one developer to a six-person client-side team with over 21,000 lines of code (not counting libraries) in just 3 years. Were it not for Backbone's ability to scale, there's simply no way we could have grown that quickly, at least without making major changes to our architecture along the way.

In short, while Backbone itself may be just under half a decade old, the real-world usage of the library has proven both its value and scalability. If your goal is to create a powerful and robust web application that a single developer can easily get off the ground but which can also grow and be supported by a full-sized team, you cannot go wrong with Backbone.

**Summary**

In this chapter, we explored how Backbone represents a new chapter in web development and why it's the best framework for your project if your goal is to make powerful and scalable web applications.

In the next chapter, we'll begin to take a look at the components that make up Backbone, in particular its easy-to-use class system. We'll also look at Backbone's sister library, Underscore, which was also created by Jeremy Ashkenas and is a requirement for Backbone itself.
Where to buy this book

You can buy Backbone.js Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.