SAMPLE CHAPTER

Vue.js IN ACTION

Erik Hanchett
with Benjamin Listwon
Foreword by Chris Fritz

MANNING

Vue.js in Action

by Erik Hanchett with Ben Listwon

Sample Chapter 1

brief contents

Part 1

Getting to know Vue.js

Before we can learn all the cool things Vue has to offer, we need to get to know it first. In these first two chapters, we'll look at the philosophy behind Vue.js, the MVVM pattern, and how it relates to other frameworks.

Once we understand where Vue is coming from, we'll look deeper at the Vue instance. The root Vue instance is the heart of the application, and we'll explore how it's structured. Later, we'll look at how we can bind data in our application to Vue.

These chapters will give you a great start in Vue.js. You'll learn how to create a simple app and how Vue works.

1 Introducing Vue.js

This chapter covers

- Exploring the MVC and MVVM design patterns
- Defining a reactive application
- Describing the Vue lifecycle
- Evaluating the design of Vue.js

Interactive websites have been around for a long time. During the beginning of the Web 2.0 days in the mid-2000s, a much larger focus was put on interactivity and engaging users. Companies such as Twitter, Facebook, and YouTube were all created during this time. The rise of social media and user-generated content was changing the web for the better.

Developers had to keep up with these changes to allow more interactivity for the end user and early on, libraries and frameworks started making interactive websites easier to build. In 2006, jQuery was released by John Resig, greatly simplifying the client-side scripting of HTML. As time progressed, client-side frameworks and libraries were created.

At first these frameworks and libraries were big, monolithic, and opinionated. Now, we've seen a shift to smaller, lighter-weight libraries that can be easily added to any project. This is where Vue.js comes in.

Vue.js is a library that enables us to add that interactive behavior and functionality to any context where JavaScript can run. Vue can be used on individual webpages for simple tasks or it can provide the foundation for an entire enterprise application.

> **TIP** The terms Vue and Vue.js are used somewhat interchangeably around the web. Throughout the book, I use the more colloquial Vue for the most part, reserving Vue.js for when I'm referring specifically to the code or the library.

From the interface that visitors interact with to the database that provides our application with its data, we'll explore how Vue and its supporting libraries enable us to build complete, sophisticated web applications.

Along the way, we'll examine how each chapter's code fits into the bigger picture, what industry best practices are applicable, and how you can incorporate what we're working on into your own projects, both existing and new.

This book is primarily written for web developers who have a moderate degree of JavaScript familiarity and a healthy understanding of HTML and CSS. That said, owing much to the versatility of its application programming interface (API), Vue is a library that grows with you as a developer as it grows with your project. Anyone who wants to build a prototype or an app for a personal side project should find this book a reliable guide on that journey.

1.1 On the shoulders of giants

Before we write any code for our first application, or even dig into Vue at a high level, it's important to understand a little bit of software history. It's difficult to truly appreciate what Vue does for us without knowledge of the problems and challenges that web applications have faced in the past and what advantages Vue brings to the table.

1.1.1 The Model–View–Controller pattern

A testament to its utility, the client-side Model–View–Controller (MVC) pattern provides the architectural blueprint used by many modern web application development frameworks. (If you're familiar with MVC, feel free to skip ahead.)

It's worth mentioning before we continue that the original MVC design pattern has changed throughout the years. Sometimes known as Classic MVC, it involved a separate set of rules on how the view, controller, and model interacted. For the sake of simplicity, we'll discuss a simplified version of the client-side MVC pattern. This pattern is a more modern interpretation for the web.

As you can see in figure 1.1, the pattern is used to separate the application's concerns. The view is responsible for displaying information to the user. This represents the graphical user interface (GUI). The controller is in the middle. It helps transform events from the view to the model and data from the model to the view. Finally, the model holds business logic and could contain a kind of datastore.

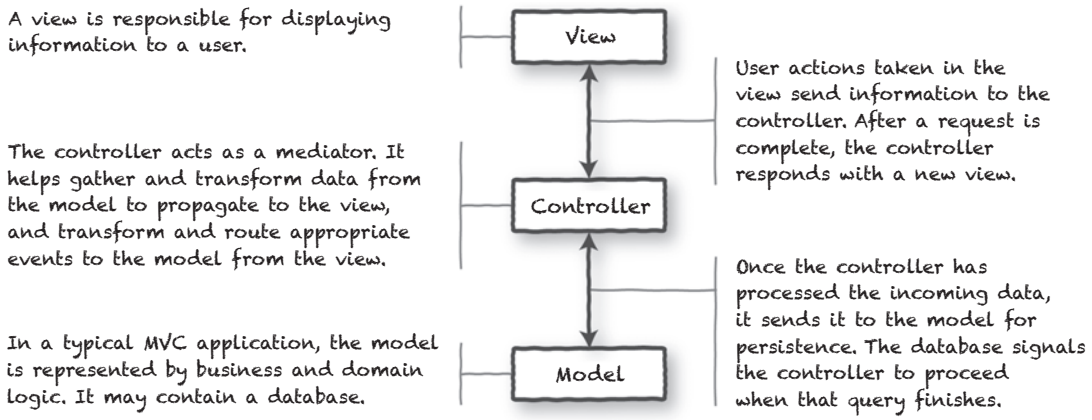

Figure 1.1 The roles of the model, view, and controller as described by the MVC pattern.

> **INFO** If you're interested in learning more about the MVC pattern, start with Martin Fowler's page on the evolution of MVC at https://martinfowler.com/eaaDev/uiArchs.html.

Many web framework authors have used a variation of this MVC pattern because of its solid, time-tested architecture. If you want to know more about how modern web frameworks are designed and architected, check out *SPA Design and Architecture* by Emmitt A. Scott Jr. (Manning, 2015).

In modern software development, the MVC pattern is often used as a part of a single application and provides a great mechanism for separating the roles of application code. For websites using the MVC pattern, every request initiates a flow of information from the client to the server, then the database, and all the way back again. That process is time-consuming, resource-intensive, and doesn't provide a responsive user experience.

Over the years, developers have increased the interactivity of web-based applications by using asynchronous web requests and client-side MVC so that requests sent to the server are non-blocking and execution continues without a reply. But as web applications begin to function more like their desktop counterparts, waiting for any client/server interaction can make an application feel sluggish or broken. That's where our next pattern comes to the rescue.

A word about business logic

You'll find a good degree of flexibility in the client-side MVC pattern when considering where business logic should be implemented. In figure 1.1 we consolidated the business logic in the model for simplicity's sake, but it may also exist in other tiers of the

(continued)

application, including the controller. The MVC pattern has changed since it was introduced by Trygve Reenskaug in 1979 for Smalltalk-76.

Consider the validation of a ZIP Code provided by a user:

- The view might contain JavaScript that validates a ZIP Code as it's entered or prior to submission.
- The model might validate the ZIP Code when it creates an address object to hold the incoming data.
- Database constraints on the ZIP Code field may mean that the model is also enforcing business logic, although this could be considered bad practice.

It can be difficult to define what constitutes actual business logic, and in many cases, all the previous constraints may come into play within a single request.

As we build our application in this book, we'll examine how and where we're organizing our business logic, as well as how Vue and its supporting libraries can help keep functionality from bleeding across boundaries.

1.1.2 The Model–View–ViewModel pattern

When JavaScript frameworks began to support asynchronous programming techniques, web applications were no longer required to make requests for complete web pages. Websites and applications could respond faster with partial updates to the view, but doing so required a degree of duplicated effort. Presentation logic often mirrored business logic.

A refinement of MVC, the primary difference in the Model–View–ViewModel (MVVM) pattern is the introduction of the *view-model*, and its data bindings (collectively, the *binder*). MVVM provides a blueprint for us to build client-side applications with more responsive user interaction and feedback, while avoiding costly duplication of code and effort across the overall architecture. It's also easier to unit test. With that said, MVVM may be overkill for simple UIs, so take that into consideration.

For web applications, the design of MVVM allows us to write software that responds immediately to user interaction and allows users to move freely from one task to the next. As you can see from figure 1.2, the view-model also wears different hats. This consolidation of responsibility has a single, profound implication for our application's views: when data changes in the view-model, any view bound to it is automatically updated. The data binder exposes data and helps guarantee that when data changes, it's reflected in the view.

INFO You can find more information on the MVVM pattern on Martin Fowler's page on the Presentation model at https://martinfowler.com/eaaDev/PresentationModel.html.

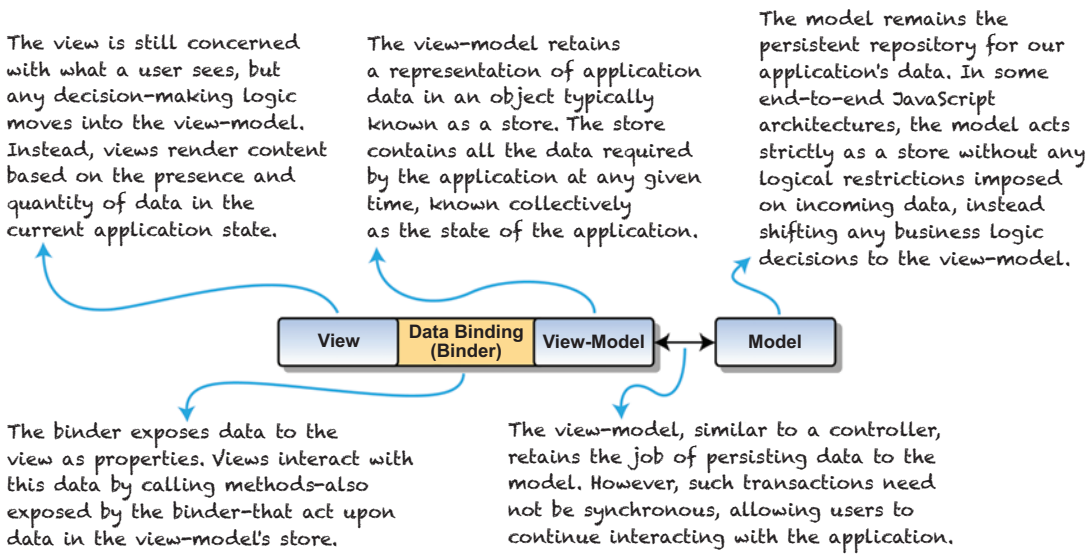

Figure 1.2 The components of the Model–View–ViewModel pattern.

1.1.3 What's a reactive application?

The reactive programming paradigm isn't necessarily a new idea. Its adoption by web applications is relatively new and owes much to the availability of JavaScript frameworks such as Vue, React, and Angular.

Many great resources on reactive theory are available on the web, but our needs are perhaps a bit more focused. For a web application to be thought of as reactive, it should do the following:

- Observe changes in application state
- Propagate change notification throughout the application
- Render views automatically in response to changes in state
- Provide timely feedback for user interactions

Reactive web applications accomplish these goals by employing MVVM design principles using asynchronous techniques to avoid blocking continued interaction and using functional programming idioms where possible.

While the MVVM pattern doesn't imply a reactive application and vice versa, they share a common intention: to provide a more responsive, reliable experience to the users of an application. Superman and Clark Kent may present themselves differently, but they both want to do right by humanity. (No, I won't say which of MVVM and Reactive I think wears the cape and which the glasses.)

> **INFO** If you'd like to learn more about Vue's reactive programming paradigm, check out the *Reactivity in Depth* guide at https://vuejs.org/v2/guide/reactivity.html.

1.1.4 A JavaScript calculator

To better understand the notions of data binding and reactivity, we'll start by implementing a calculator in plain, vanilla JavaScript, as shown in this listing.

Listing 1.1 The JavaScript calculator: chapter-01/calculator.html

```
<!DOCTYPE>
<html>
  <head>
    <title>A JavaScript Calculator</title>
    <style>
     p, input { font-family: monospace; }
     p, { white-space: pre; }
    </style>
  </head>
  <!-- Bind to the init function -->
  <body>
    <div id="myCalc">
      <p>x <input class="calc-x-input" value="0"></p>
      <p>y <input class="calc-y-input" value="0"></p>
      <p>--------------------</p>
         <p>= <span class="calc-result"></span></p>
    </div>
    <script type="text/javascript">
     (function(){

       function Calc(xInput, yInput, output) {
         this.xInput = xInput;
         this.yInput = yInput;
         this.output = output;
       }

       Calc.xName = 'xInput';
       Calc.yName = 'yInput';

       Calc.prototype = {
         render: function (result) {
           this.output.innerText = String(result);
         }
       };

       function CalcValue(calc, x, y) {
         this.calc = calc;
         this.x = x;
         this.y = y;
         this.result = x + y;
       }

       CalcValue.prototype = {
         copyWith: function(name, value) {
           var number = parseFloat(value);

           if (isNaN(number) || !isFinite(number))
             return this;
```

Forms input to collect x and y that bind to the runCalc function

Shows results of x and y

Shows constructor to create calc instance

Shows constructor to create values for a calc instance

```
    if (name === Calc.xName)
      return new CalcValue(this.calc, number, this.y);

    if (name === Calc.yName)
      return new CalcValue(this.calc, this.x, number);

    return this;
  },
  render: function() {
    this.calc.render(this.result);
  }
};

function initCalc(elem) {
  var calc =
    new Calc(
      elem.querySelector('input.calc-x-input'),
      elem.querySelector('input.calc-y-input'),
      elem.querySelector('span.calc-result')
    );
  var lastValues =
    new CalcValue(
      calc,
      parseFloat(calc.xInput.value),
      parseFloat(calc.yInput.value)
    );

  var handleCalcEvent =
    function handleCalcEvent(e) {
      var newValues = lastValues,
          elem = e.target;

      switch(elem) {
        case calc.xInput:
          newValues =
            lastValues.copyWith(
              Calc.xName,
              elem.value
            );
          break;
        case calc.yInput:
          newValues =
            lastValues.copyWith(
              Calc.yName,
              elem.value
            );
          break;
      }

      if(newValues !== lastValues){
        lastValues = newValues;
        lastValues.render();
      }
    };

  elem.addEventListener('keyup', handleCalcEvent, false);
```

Initializes calc component

Shows the event handler

Sets the event listener on keyup

```
        return lastValues;
      }

      window.addEventListener(
        'load',
        function() {
          var cv = initCalc(document.getElementById('myCalc'));
          cv.render();
        },
        false
      );

    }());
  </script>
 </body>
</html>
```

This is a calculator using ES5 JavaScript (we'll use the more modern version of JavaScript ES6/2015 later in the book). We're using an immediately invoked function expression that kicks off our JavaScript. A constructor is used to hold values and the `handleCalcEvent` event handler fires on any `keyup`.

1.1.5 A Vue calculator

Don't worry too much about the syntax of the Vue example because our goal here isn't to understand everything going on in the code, but to compare the two implementations. That said, if you have a good sense of how the JavaScript example works (as shown in the following listing), much of the Vue code should make sense at least on a theoretical level.

Listing 1.2 The Vue calculator: chapter-01/calculatorvue.html

```
<!DOCTYPE html>
<html>
<head>
  <title>A Vue.js Calculator</title>
  <style>
    p, input { font-family: monospace; }
    p { white-space: pre; }
  </style>
</head>
<body>
  <div id="app">
    <p>x <input v-model="x"></p>
    <p>y <input v-model="y"></p>
    <p>----------------------</p>
    <p>= <span v-text="result"></span></p>
  </div>

  <script src="https://unpkg.com/vue/dist/vue.js"></script>
  <script type="text/javascript">
```

Shows the DOM anchor for our app

Shows the form inputs for the application

Results will show up in this span.

Lists the script tag that adds the Vue.js library

```
function isNotNumericValue(value) {
  return isNaN(value) || !isFinite(value);
}
var calc = new Vue({
  el: '#app',
  data: { x: 0, y: 0, lastResult: 0 },
  computed: {
    result: function() {
        let x = parseFloat(this.x);
        if(isNotNumericValue(x))
          return this.lastResult;

        let y = parseFloat(this.y);
        if(isNotNumericValue(y))
          return this.lastResult;

        this.lastResult = x + y;

        return this.lastResult;
    }
  }
});
</script>
</body>
</html>
```

Initializes the application

Shows the variables added to the app

Connects to the DOM

Calculation is done here using a computed property.

1.1.6 Comparison of JavaScript and Vue

The code for both calculator implementations is, for the most part, different. Each sample shown in figure 1.3 is available in the repository that accompanies this chapter, so you can run each one and compare how they operate.

```
<script type="text/javascript">
 (function(){

   function Calc(xInput, yInput, output) {
     this.xInput = xInput;
     this.yInput = yInput;
     this.output = output;
   }

   Calc.xName = 'xInput';
   Calc.yName = 'yInput';

   Calc.prototype = {
     render: function (result) {
       this.output.innerText = String(result);
     }
   };

   function CalcValue(calc, x, y) {
     this.calc = calc;
     this.x = x;
     this.y = y;
     this.result = x + y;
   }
```

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
function isNotNumericValue(value) {
  return isNaN(value) || !isFinite(value);
}
  var calc = new Vue({
    el: '#app',
    data: { x: 0, y: 0, lastResult: 0 },
    computed: {
      result: function() {
        let x = parseFloat(this.x);
        if(isNotNumericValue(x))
        return this.lastResult;

        let y = parseFloat(this.y);
        if(isNotNumericValue(y))
        return this.lastResult;

        this.lastResult = x + y;

        return this.lastResult;
      }
    }
  });
</script>
```

Figure 1.3 Side-by-side comparison of a reactive calculator written using vanilla JavaScript (on the left) and Vue (on the right).

The key difference between the two applications is how an update to the final calculation is triggered and how the result finds its way back to the page. In our Vue example, a single binding `v-model` takes care of all the updates and calculations on the page. When we instantiate our application with `new Vue({ ... })`, Vue examines our JavaScript code and HTML markup, then creates all the data and event bindings needed for our application to run.

1.1.7 How does Vue facilitate MVVM and reactivity?

Vue is sometimes referred to as a *progressive framework*, which broadly means that it can be incorporated into an existing web page for simple tasks or that it can be used entirely as the basis for a large-scale web application.

Regardless of how you choose to incorporate Vue into your project, every Vue application has at least one *Vue instance*. The most basic application will have a single instance that provides bindings between designated markup and data stored in a view-model (see figure 1.4).

Being built entirely out of web technologies, a single Vue instance exists entirely in the web browser. Crucially, this means that we don't depend on server-based page reloads for updated views, executing business logic, or any other task that falls under the domain of the view or view-model. Let's revisit our form submission example with that in mind.

Perhaps the most striking change relative to the client-side MVC architecture is that the browser page needs to rarely, if ever, reload during the user's entire session. Because the view, view-model, and data bindings are all implemented in HTML and JavaScript,

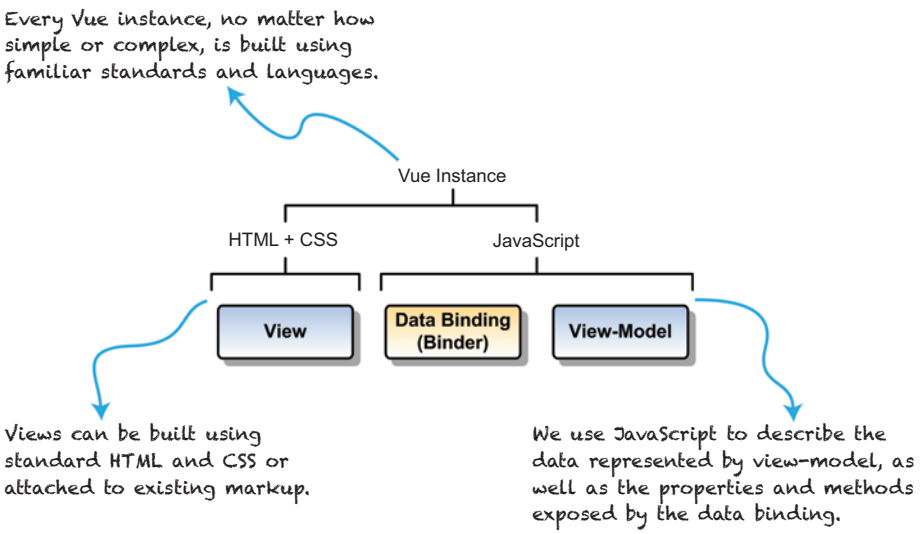

Figure 1.4 A typical Vue instance binds HTML markup to data in a view-model by creating a data binding between them.

our application can delegate tasks to the model asynchronously, leaving users free to continue with other tasks. When new data is returned from the model, the bindings established by Vue will trigger whatever updates need to happen in the view.

Arguably, it's Vue's primary role to facilitate user interaction by creating and maintaining the binding between the views we create and the data in our view-model. In this capacity, as we'll see in our first application, Vue provides a solid bedrock for any reactive web application.

1.2 Why Vue.js?

When starting a new project, there are many decisions to make. One of the most important is the framework or library that should be used. If you're an agency or even a solo developer, picking the correct tool for the job is extremely important. Luckily, Vue.js is versatile and can handle many different situations.

What follows are several of the most commonly voiced concerns that you might have when starting a new project as a solo developer or agency, plus a description of how Vue helps to address them, either directly or as part of a larger movement toward reactive web applications.

- *Our team isn't strong at working with web frameworks.* One of the greatest advantages of using Vue for a project is that it doesn't require any specialist knowledge. Every Vue application is built with HTML, CSS, and JavaScript—familiar tools that allow you to be productive right from the get-go. Even teams that have little experience developing any sort of frontend find a comfortable foothold in the MVVM pattern because of their familiarity with MVC in other contexts.
- *We've got existing work we'd like to continue using.* Don't worry, there's no need to scrap your carefully crafted CSS or that cool carousel you built. Whether you're dropping Vue into an existing project with many dependencies, or you're starting a new project and want to leverage other libraries you are already familiar with, Vue won't get in the way. You can continue using tools such as Bootstrap or Bulma as a CSS framework, keep jQuery or Backbone components around, or incorporate your preferred library for making HTTP requests, handling Promises or other extended functionality.
- *We need to prototype quickly and gauge users' reactions.* As we saw in our first Vue application, all we need to do to start building with Vue is include Vue.js in any standalone webpage. No complicated build tools required! Getting a prototype in front of users can happen within a week or two of starting development, allowing you to gather feedback early and iterate often.
- *Our product is used almost exclusively on mobile devices.* The minified and gzipped Vue.js file weighs in at around 24 KB, which is quite compact for a frontend framework. The library is easily delivered over cellular connections. New to Vue 2 is server-side rendering (SSR). Such a strategy means that an application's initial load can be minimal, allowing you to pull in new views and resources only as

required. Combining SSR with efficient caching of components reduces bandwidth consumption even further.

- *Our product has unique and custom functionality.* Architected from the ground up with modularity and extensibility in mind, Vue applications use reusable components. Vue also supports extending components through inheritance, incorporating functionality with mix-ins, and extending Vue's functionality with plugins and custom directives.
- *We have a large user base and performance is a concern.* Recently rewritten for reliability, performance, and speed, Vue now uses a virtual DOM. What that means is that Vue first performs operations on a DOM representation that isn't attached to the browser then "copies" those changes to the view we see. As a result, Vue routinely outperforms other frontend libraries. Because generalized tests are often too abstract, I always encourage clients to select several of their typical use cases and a few extreme ones, develop a testing scenario, and measure the results for themselves. You can learn more about the virtual DOM and how it compares to its competitors at https://vuejs.org/v2/guide/comparison.html.
- *We have an existing build, test, and/or deployment process.* In the latter chapters of the book we'll explore these topics in depth, but the takeaway is that Vue is easily integrated into many of the most popular build (Webpack, Browserify, and others) and test (Karma, Jasmine, and so on) frameworks. In many instances, unit tests are directly portable if you've already written them for an existing framework. And if you're starting out but want to use these tools, Vue provides project templates that integrate these tools for you. In the simplest of terms, it's easy to add and adapt Vue to existing projects.
- *What do we do if we need help during or after our engagement?* Two of the immeasurable benefits of Vue are its community and support ecosystem. Vue is well-documented, both in online docs and within the code itself, and the core team is active and responsive. Perhaps even more crucial, the community of developers working with Vue is equally as strong. Resources such as Gitter and the Vue forums are full of helpful folks, and there's a growing list of plugins, integrations, and library extensions that bring popular code to the platform nearly every day.

After asking many of these questions on my own projects, I now recommend Vue on almost all my projects. As you become confident in your mastery of Vue throughout this book, my hope is that you'll feel comfortable advocating for Vue in your next project.

1.3 Future thoughts

We've covered much ground in this introductory chapter alone. If you're new to web application development, this may be your first contact with the MVVM architecture or reactive programming, but we've seen that building a reactive application isn't as intimidating as the jargon can make it feel.

Perhaps the biggest takeaway from this chapter isn't about Vue itself, but how reactive applications are easier to work with and easier to write with. It's also nice that we

have less boilerplate interface code to write. Not having to script all our user's interactions frees us up to focus on how to model our data and design our interface. Wiring them up is something Vue makes effortless.

If you're like me, then you're already thinking of the gazillion ways you can make our modest application better. This is a good thing, and you should absolutely experiment and play with the code. Here are a few things I think about when I look at the app:

- How would we eliminate the need to repeat text strings in so many places?
- Can we clear the default input when a user focuses on an input? What about restoring it if they leave the field blank?
- Is there a way to avoid hand-coding each input?

In part 2, we'll find answers to all these questions and many more. Vue was designed to grow with us as developers, as much as with our code, so we'll always make sure to look at different strategies, compare their strengths and weaknesses, and learn how to decide which is the best practice for a given situation.

All right, let's see how we can improve on some of what we wrote!

Summary

- A brief history of how models, views, and controllers work, and how they're tied into Vue.js.
- How Vue.js can save you time when creating an application.
- Why you should consider Vue.js for your next project.

JAVASCRIPT/WEB DEVELOPMENT

Vue.js IN ACTION

Hanchett • Listwon

Vue.js is a lightweight frontend framework, offering easy two-way data binding, a reactive UI, and a common-sense project structure. It uses UI patterns and modern HTML to deliver impossibly fast page loads and silky smooth transitions—all from a tiny code footprint. It's a delight to develop in Vue using ordinary JavaScript and its integrated Vuex state management tool.

Vue.js in Action is your guide to building modern web apps. You'll start by exploring the reactive UI model while you get comfortable with Vue's unique features. Then, you'll go deeper as you build a shopping cart with an admin interface and the ability to manage stock! Finally, you'll extend your app, adding transitions, tests, and other key features until it's production ready.

What's Inside

- Clearly annotated code and illustrations
- Modeling data and consuming APIs
- Easy state management with Vuex
- Creating custom directives

Written for web developers with some experience in JavaScript, HTML, and CSS.

Erik Hanchett and **Benjamin Listwon** are experienced web engineers and fearless explorers of new ideas.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/books/vue-js-in-action

"Carefully explains the foundational concepts for understanding what Vue is doing and why."

—From the Foreword by Chris Fritz, Vue Core Team Member

"An excellent hands-on introduction to Vue.js and its ecosystem."

—Alex Miller, Slalom

"Practical examples make learning easy and offer a solid foundation for your own projects."

—Doug Warren, Java Web Services

"Provides a strong understanding of the intrinsic mechanisms of Vues.js. Priceless."

—Philippe Charrière
Clever Cloud

ISBN-13: 978-1-61729-462-4
ISBN-10: 1-61729-462-4
54499
9 781617 294624

MANNING $44.99 / Can $59.99 [INCLUDING eBOOK]