Erik Hanchett, Benjamin Listwon

Vue.js w akcji

Helion

Tytuł oryginału: Vue.js in Action

Tłumaczenie: Piotr Rajca

Projekt okładki: Studio Gravite / Olsztyn; Obarek, Pokoński, Pazdrijowski, Zaprucki
Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

ISBN: 978-83-283-5734-1

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/vueakc
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/vueakc.zip

Printed in Poland.

Spis treści

Słowo wstępne 9
Przedmowa 11
Podziękowania 13
O książce 15
O autorze 19

CZĘŚĆ 1. POZNAJEMY VUE.JS 21

Rozdział 1. Wprowadzenie do Vue.js 23

1.1. Na ramionach giganta 24
1.1.1. Wzorzec projektowy Model-Widok-Kontroler 24
1.1.2. Wzorzec Model-Widok-ModelWidoku 26
1.1.3. Czym są aplikacje reaktywne? 27
1.1.4. Kalkulator JavaScript 28
1.1.5. Kalkulator w wersji Vue 30
1.1.6. Porównanie wersji JavaScript i Vue 31
1.1.7. W jaki sposób Vue ułatwia użycie wzorca MVVM oraz zapewnienie reaktywności? 32
1.2. Dlaczego Vue.js? 33
1.3. Dalsze przemyślenia 35
Podsumowanie 36

Rozdział 2. Instancja Vue 37

2.1. Nasza pierwsza aplikacja 38
2.1.1. Główna instancja Vue 38
2.1.2. Jak się upewnić, że aplikacja działa? 40
2.1.3. Wyświetlanie czegoś w widoku 43
2.1.4. Sprawdzanie właściwości w przeglądarce 44
2.2. Cykl życia Vue 45
2.2.1. Dodawanie funkcji obsługi cyklu życia 47
2.2.2. Badanie kodu demonstrującego cykl życia 48
2.2.3. Czy zostawiać kod funkcji zwrotnych cyklu życia, czy nie? 49
2.3. Wyświetlanie produktu 50
2.3.1. Definiowanie danych produktu 50
2.3.2. Przygotowywanie widoku produktu 51

2.4. Stosowanie filtrów wyjścia 54
2.4.1. *Pisanie filtra wyjścia* 55
2.4.2. *Dodawanie filtra do kodu i testowanie różnych wartości* 56
Ćwiczenie 57
Podsumowanie 58

CZĘŚĆ 2. WIDOK I MODEL WIDOKU 59

Rozdział 3. Dodawanie interaktywności 61

3.1. Początkiem danych koszyka jest dodanie tablicy 62
3.2. Powiązania ze zdarzeniami DOM 63
3.2.1. *Podstawy dowiązywania zdarzeń* 63
3.2.2. *Powiązanie zdarzenia z przyciskiem Dodaj do koszyka* 64
3.3. Dodanie przycisku koszyka i liczby produktów 65
3.3.1. *Kiedy stosować właściwości obliczane?* 66
3.3.2. *Sprawdzanie zdarzeń aktualizacji w przypadku stosowania właściwości obliczanych* 68
3.3.3. *Wyświetlanie liczby produktów w koszyku i testowanie* 71
3.4. Dodawanie afordancji do przycisku 74
3.4.1. *Śledzenie stanu magazynu* 75
3.4.2. *Praca z właściwościami obliczanymi i stanem magazynu* 76
3.4.3. *Podstawy dyrektywy v-show* 77
3.4.4. *Stosowanie dyrektyw v-if oraz v-else w celu wyświetlania nieaktywnego przycisku* 78
3.4.5. *Dodanie przycisku koszyka działającego jako przełącznik* 80
3.4.6. *Użycie dyrektywy v-if do wyświetlania formularza zamówienia* 81
3.4.7. *Porównanie dyrektyw v-show oraz v-if i v-else* 83
Ćwiczenie 84
Podsumowanie 84

Rozdział 4. Formularze i pola 85

4.1. Stosowanie powiązań v-model 86
4.2. Rzut oka na powiązania wartości 94
4.2.1. *Powiązanie wartości z polem wyboru* 94
4.2.2. *Stosowanie powiązań danych i przycisków opcji* 96
4.2.3. *Przedstawienie dyrektywy v-for* 97
4.2.4. *Dyrektywa v-for bez opcjonalnych kluczy* 100
4.3. Prezentacja modyfikatorów 101
4.3.1. *Stosowanie modyfikatora .number* 101
4.3.2. *Usuwanie odstępów z wpisanych wartości* 103
4.3.3. *Modyfikator .lazy dyrektywy v-model* 104
Ćwiczenie 105
Podsumowanie 105

Rozdział 5. Dyrektywy warunkowe, pętle i listy 107

5.1. Wyświetlanie komunikatu o stanie magazynu 108
5.1.1. Wyświetlanie dostępnych egzemplarzy przy użyciu v-if 108
5.1.2. Dodawanie innych komunikatów z użyciem dyrektyw v-else oraz v-else-if 110
5.2. Przeglądanie listy produktów 112
5.2.1. Dodawanie oceny przy użyciu zakresu dyrektywy v-for 112
5.2.2. Powiązanie klasy elementu HTML z oceną produktu 114
5.2.3. Dodanie produktów 117
5.2.4. Importowanie produktów z pliku products.json 119
5.2.5. Refaktoryzacja kodu aplikacji i dodanie dyrektywy v-for 120
5.3. Sortowanie rekordów 126
Ćwiczenie 127
Podsumowanie 127

Rozdział 6. Stosowanie komponentów 129

6.1. Czym są komponenty? 130
6.1.1. Tworzenie komponentów 130
6.1.2. Rejestracja globalna 131
6.1.3. Rejestracja lokalna 132
6.2. Zależności w komponentach 133
6.3. Stosowanie właściwości props do przekazywania danych 135
6.3.1. Właściwości literałowe 135
6.3.2. Dynamiczne właściwości props 136
6.3.3. Walidacja właściwości props 139
6.4. Definiowanie szablonu komponentu 142
6.4.1. Stosowanie wpisanych łańcuchów szablonów 142
6.4.2. Element script typu text/x-template 143
6.4.3. Komponenty jednoplikowe 144
6.5. Stosowanie zdarzeń niestandardowych 145
6.5.1. Nasłuchiwanie zdarzeń 146
6.5.2. Modyfikowanie właściwości przy użyciu .sync 148
Ćwiczenie 149
Podsumowanie 149

Rozdział 7. Zaawansowane zastosowania komponentów i trasowanie 151

7.1. Stosowanie gniazd 152
7.2. Rzut oka na gniazda nazwane 155
7.3. Gniazda z zasięgiem 157
7.4. Tworzenie aplikacji z komponentami dynamicznymi 159
7.5. Tworzenie komponentów asynchronicznych 162
7.6. Konwersja aplikacji składu dla zwierzaków z użyciem Vue-CLI 163
7.6.1. Tworzenie nowej aplikacji przy użyciu Vue-CLI 165
7.6.2. Konfigurowanie tras 166
7.6.3. Dodanie stylów CSS, Bootstrapa i biblioteki Axios 168

7.6.4. Przygotowanie komponentów 170
7.6.5. Tworzenie komponentu Form 172
7.6.6. Dodanie komponentu Main 173
7.7. Stosowanie tras 176
7.7.1. Dodanie trasy produktu z parametrami 176
7.7.2. Konfiguracja router-link z użyciem znaczników 180
7.7.3. Określanie stylów z użyciem komponentu router-link 182
7.7.4. Dodanie trasy podrzędnej edit 183
7.7.5. Stosowanie przekierowań i znaków wieloznacznych 185
Ćwiczenie 187
Podsumowanie 187

Rozdział 8. Efekty przejść i animacje 189

8.1. Podstawy efektów przejść 189
8.2. Podstawy animacji 194
8.3. Funkcje zwrotne animacji 196
8.4. Efekty przejść dla komponentów 199
8.5. Aktualizacja aplikacji składu dla zwierzaków 202
8.5.1. Dodawanie efektu przejścia do aplikacji sklepu dla zwierzaków 202
8.5.2. Dodawanie animacji do aplikacji składu dla zwierzaków 204
Ćwiczenie 206
Podsumowanie 206

Rozdział 9. Rozszerzanie Vue 207

9.1. Wielokrotne stosowanie możliwości funkcjonalnych dzięki wstawkom 208
9.1.1. Wstawki globalne 212
9.2. Poznawanie dyrektyw niestandardowych na przykładach 213
9.2.1. Globalne dyrektywy niestandardowe z modyfikatorami, wartościami i argumentami 216
9.3. Funkcje renderujące i JSX 219
9.3.1. Przykład funkcji renderującej 220
9.3.2. Przykład JSX 223
Ćwiczenie 227
Podsumowanie 228

CZĘŚĆ 3. MODELOWANIE DANYCH, KORZYSTANIE Z API I TESTOWANIE 229

Rozdział 10. Vuex 231

10.1. Biblioteka Vuex — do czego może się przydać? 232
10.2. Stan i modyfikacje w bibliotece Vuex 233
10.3. Akcesory get i akcje 237
10.4. Dodawanie Vuex do aplikacji Vue-CLI na przykładzie składu dla zwierzaków 240
10.4.1. Instalacja Vuex w aplikacji Vue-CLI 240

10.5. Metody pomocnicze Vuex 244
10.6. Krótka prezentacja modułów 247
Ćwiczenie 249
Podsumowanie 249

Rozdział 11. Komunikacja z serwerem 251

11.1. Renderowanie po stronie serwera 252
11.2. Wprowadzenie do Nuxt.js 253
11.2.1. Tworzenie aplikacji do wyszukiwania muzyki 254
11.2.2. Tworzenie projektu i zainstalowanie zależności 257
11.2.3. Tworzenie elementów aplikacji i komponentów 260
11.2.4. Aktualizacja domyślnego układu 263
11.2.5. Dodanie magazynu Vuex 264
11.2.6. Stosowanie oprogramowania warstwy pośredniej 264
11.2.7. Generowanie tras w Nuxt.js 266
11.3. Komunikacja z serwerem na przykładzie Firebase i VuexFire 270
11.3.1. Konfiguracja Firebase 271
11.3.2. Konfigurowanie aplikacji składu dla zwierzaków do korzystania z Firebase 274
11.3.3. Przechowywanie stanu uwierzytelniania w magazynie Vuex 277
11.3.4. Uzupełnienie komponentu nagłówka o informacje uwierzytelniające 278
11.3.5. Modyfikacja Main.vue i zastosowanie w nim bazy danych Firebase 282
Ćwiczenie 283
Podsumowanie 283

Rozdział 12. Testowanie 285

12.1. Tworzenie przypadków testowych 286
12.2. Ciągła integracja, dostarczanie i wdrażanie 287
12.2.1. Integracja ciągła 287
12.2.2. Dostarczanie ciągłe 288
12.2.3. Wdrażanie ciągłe 289
12.3. Rodzaje testów 289
12.4. Przygotowywanie środowiska 290
12.5. Pisanie pierwszego przypadku testowego z użyciem vue-test-utils 292
12.6. Testowanie komponentów 296
12.6.1. Testowanie właściwości props 296
12.6.2. Testowanie tekstów 297
12.6.3. Testowanie klas CSS 298
12.6.4. Testowanie z użyciem atrapy Vuex 299
12.7. Konfiguracja debuggera Chrome 301
Ćwiczenie 303
Podsumowanie 304

Dodatek A. Przygotowywanie środowiska 305

Dodatek B. Rozwiązania ćwiczeń 313

1 Wprowadzenie do Vue.js

Ten rozdział opisuje następujące zagadnienia:

- Prezentację wzorców projektowych MVC i MVVM.
- Definiowanie aplikacji reaktywnej.
- Opis cyklu życia Vue.
- Ocena projektu Vue.js.

Interaktywne witryny WWW pojawiły się już wiele lat temu. Na początku ery Web 2.0, w połowie pierwszej dekady tego wieku, zaczęto zwracać znacznie większą uwagę na interaktywność i angażowanie użytkowników. Firmy takie, jak Twitter, Facebook i YouTube, powstały właśnie w tamtym czasie. Nadejście mediów społecznościowych i treści generowanych przez użytkowników przyczyniło się do zmiany Sieci na lepszą.

Programiści musieli dotrzymać kroku tym zmianom, by zapewnić użytkownikom większą interaktywność, a na początku tego okresu zaczęły już powstawać biblioteki i frameworki ułatwiające tworzenie interaktywnych witryn internetowych. W 2006 roku John Resig udostępnił swoją bibliotekę jQuery, która ogromnie ułatwiała tworzenie skryptów JavaScript wykonywanych w przeglądarkach WWW. Po pewnym czasie zaczęły powstawać frameworki i inne biblioteki działające po stronie przeglądarki.

Początkowo te frameworki i biblioteki były bardzo duże, monolityczne, a ich twórcy — zadufani w sobie. Później jednak nastąpiła zmiana podejścia i zaczęto tworzyć mniejsze biblioteki, które z łatwością można dodawać do dowolnego projektu. I właśnie tak powstała biblioteka Vue.js.

Vue.js to biblioteka pozwalająca na dodawanie interaktywnych zachowań i możliwości funkcjonalnych w dowolnym kontekście, w jakim jest wykonywany kod JavaScript. Można jej używać na pojedynczych stronach WWW do prostych zadań, może jednak także stanowić podstawę całych aplikacji korporacyjnych.

WSKAZÓWKA W internecie terminy Vue i Vue.js są używane w zasadzie zamiennie. W tej książce w większości tekstu używam bardziej kolokwialnego określenia, Vue, rezerwując użycie Vue.js dla sytuacji, w których odwołuję się konkretnie do biblioteki lub kodu.

Zaczynając od interfejsu, z którym użytkownicy będą prowadzić interakcję, poprzez bazy danych dostarczające informacji dla aplikacji w tej książce przyjrzymy się, jak Vue i inne, towarzyszące jej biblioteki pozwalają na tworzenie kompletnych, złożonych aplikacji internetowych.

Po drodze dowiesz się, jak kody przedstawione w poszczególnych rozdziałach pasują do większej układanki, jakie najlepsze praktyki można stosować oraz w jaki sposób rozwiązania i techniki prezentowane w tej książce możesz stosować we własnych projektach, i to zarówno tych już istniejących, jak i nowych.

Ta książka jest przeznaczona głównie dla programistów aplikacji internetowych, którzy mają przeciętną znajomość języka JavaScript oraz dobrze rozumieją języki HTML i CSS. Warto zwrócić uwagę, że Vue dzięki wszechstronności swojego interfejsu programowania aplikacji (API) jest biblioteką, która rośnie wraz z nami — programistami — i naszymi projektami. Niniejsza książka będzie dobrym, solidnym przewodnikiem dla każdego, kto planuje stworzenie prototypu lub swojej prywatnej witryny.

1.1. Na ramionach giganta

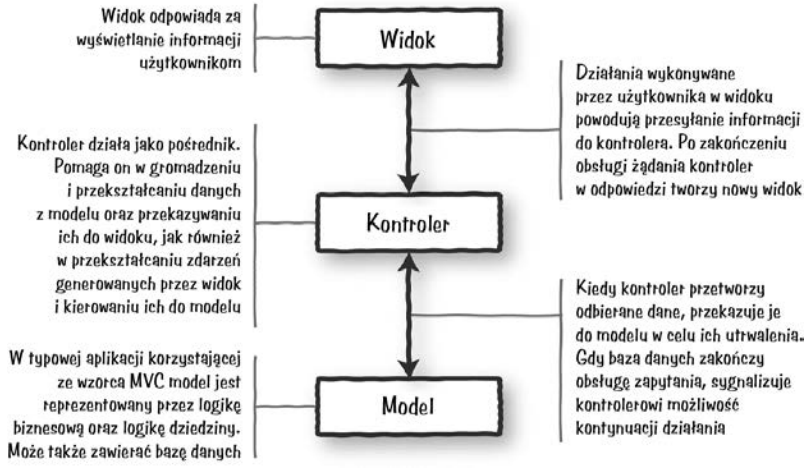
Zanim zabierzemy się za pisanie jakiegokolwiek kodu pierwszej aplikacji, a nawet zanim przyjrzymy się Vue na bardzo ogólnym poziomie, musisz poznać i zrozumieć nieco historii programowania. Trudno bowiem będzie docenić to, co daje nam Vue, bez znajomości problemów i wyzwań, jakie w przeszłości stały przed twórcami aplikacji internetowych, oraz korzyści, jakie Vue zapewnia.

1.1.1. Wzorzec projektowy Model-Widok-Kontroler

Wzorzec projektowy Model-Widok-Kontroler (ang. *Model-View-Controller*, w skrócie: *MVC*) stanowi, czym dowodzi swojej przydatności, architektoniczny szablon stosowany przez wiele istniejących frameworków do tworzenia aplikacji internetowych. (Jeśli znasz ten wzorzec, to spokojnie możesz pominąć dalszą część tego punktu).

Warto zwrócić uwagę, że pierwotny wzorzec MVC w ciągu lat się zmienił. Wzorzec nazywany czasami *klasycznym wzorcem MVC* (ang. *Classic MVC*) korzystał z odrębnego zestawu reguł określających wzajemne interakcje jego poszczególnych elementów — modelu, widoku oraz kontrolera. Dla uproszczenia rozważań przedstawię tu uproszczoną, kliencką, wersję wzorca MVC. Stanowi ona bardziej nowoczesną interpretację pierwotnego wzorca, powszechnie stosowaną w tworzeniu aplikacji internetowych.

Jak widać na rysunku 1.1, wzorzec MVC służy do separowania obowiązków w aplikacji. Widok jest odpowiedzialny za wyświetlanie informacji użytkownikom. Reprezentuje on graficzny interfejs użytkownika (GUI — ang. *Graphical User Interface*). Pośrodku znajduje się kontroler. Pomaga on w przekształcaniu zdarzeń pochodzących z widoku i przesyłaniu ich do modelu oraz w przekazywaniu danych z modelu do widoku. I w końcu model zawiera logikę biznesową i może zawierać wszelkiego rodzaju magazyny danych.



Rysunek 1.1. Role modelu, widoku i kontrolera opisywane przez wzorzec MVC

UWAGA Jeśli chcesz dowiedzieć się czegoś więcej o wzorcu MVC, zacznij od strony Martina Fowlera poświęconej ewolucji tego wzorca — *https://martinfowler.com/eaaDev/uiArchs.html*.

Ze względu na niezawodną i świetnie przetestowaną architekturę wzorca MVC wielu twórców frameworków internetowych zdecydowało się go zastosować. Gdybyś chciał się dowiedzieć, jak są projektowane nowoczesne frameworki do tworzenia aplikacji internetowych oraz jaka jest ich architektura, to polecam książkę Emmitta A. Scotta Jr. pt. „SPA Design and Architecture", wydaną przez wydawnictwo Manning w 2015 roku.

W nowoczesnych rozwiązaniach wzorzec MVC jest często stosowany jako element pojedynczych aplikacji, ponieważ stanowi doskonały mechanizm separacji ról w kodzie aplikacji. W przypadku witryn WWW korzystających z tego wzorca każde żądanie inicjuje przepływ informacji z klienta na serwer i do bazy danych, a następnie w przeciwnym kierunku. Ten proces jest czasochłonny, wymaga wielu zasobów, a pod względem płynności działania interfejsu nie zapewnia odpowiednich doznań użytkownika.

Wraz z upływem lat programiści poprawili interaktywność aplikacji internetowych przez zastosowanie żądań asynchronicznych oraz klienckiego wzorca MVC, dzięki czemu żądania przesyłane na serwer są realizowane w sposób, który nie blokuje aplikacji i pozwala na jej dalsze działanie bez konieczności odebrania odpowiedzi z serwera. Jednak zmiany, które sprawiają, że aplikacje internetowe w coraz to większym stopniu zaczynają działać jak klasyczne programy komputerowe, powodują także, że oczeki-

wanie na interakcję klient-serwer może prowadzić do spowolnienia funkcjonowania aplikacji lub wrażenia, że w ogóle przestała działać. I właśnie w tym miejscu na scenę wkracza następny wzorzec projektowy.

Kilka słów o logice biznesowej

Kliencki wzorzec MVC zapewnia sporą swobodę, jeśli chodzi o miejsce implementacji logiki biznesowej. W przypadku wzorca przedstawionego na rysunku 1.1 logikę biznesową ze względu na prostotę rozwiązania umieściliśmy w modelu, jednak równie dobrze można ją zaimplementować w innych warstwach, w tym w kontrolerze. Model MVC, odkąd został wprowadzony przez Trygve Reenskauga w 1979 roku dla języka Smalltalk-76, trochę się zmienił.

Rozważmy problem weryfikacji poprawności kodu pocztowego podawanego przez użytkownika:

- Widok może zawierać kod JavaScript sprawdzający poprawność wpisanego kodu pocztowego jeszcze przed jego przesłaniem.
- Model może sprawdzać poprawność kodu w czasie, gdy będzie tworzony obiekt adresu zawierający przesłane dane.
- Ograniczenia bazy danych zdefiniowane dla pola kodu pocztowego oznaczają, że także model może wymuszać elementy logiki biznesowej, choć można to uznać za złą praktykę.

A zatem określenie, co stanowi faktyczną logikę biznesową, może być stosunkowo trudne, a w wielu przypadkach w trakcie obsługi żądania mogą być uwzględniane wszystkie wymienione wcześniej ograniczenia.

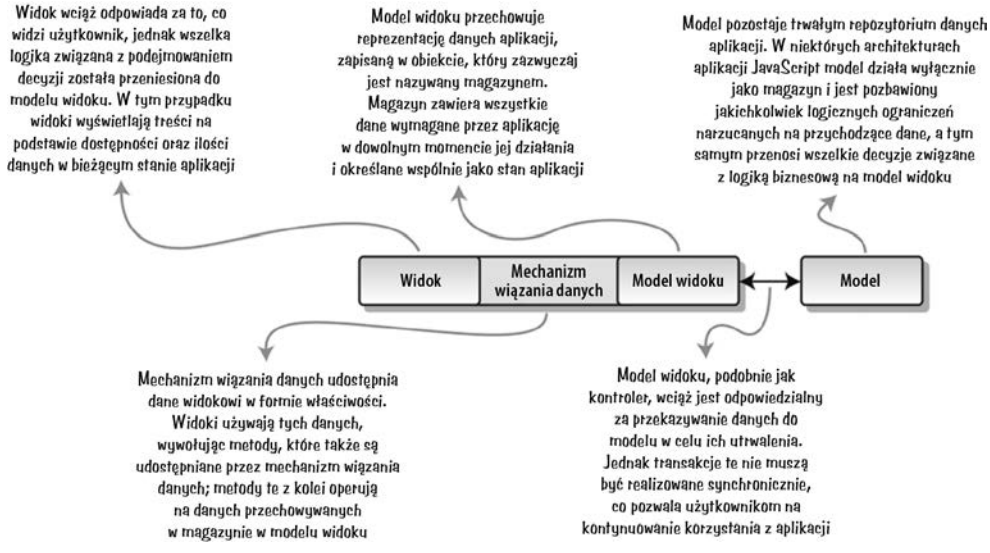
Podczas tworzenia aplikacji prezentowanej w tej książce dowiesz się, jak i gdzie należy organizować logikę biznesową oraz w jaki sposób Vue i inne współpracujące z nią biblioteki ułatwiają utrzymanie separacji możliwości funkcjonalnych i uchronienie się przed ich przenikaniem pomiędzy poszczególnymi warstwami aplikacji.

1.1.2. Wzorzec Model-Widok-ModelWidoku

Kiedy frameworki JavaScript zaczęły obsługiwać techniki programowania asynchronicznego, zniknęła konieczność, by aplikacje internetowe pobierały z serwerów kompletne strony WWW. Witryny i aplikacje mogły zacząć odpowiadać szybciej, gdyż musiały pobierać tylko częściowe aktualizacje widoków, jednak takie rozwiązania wymagały pewnych powtarzających się działań. Logika prezentacji często musiała odwzorowywać logikę biznesową.

W ramach usprawnienia wzorca MVC opracowany został nowy wzorzec projektowy — Model-Widok-ModelWidoku (ang. *Model-View-ViewModel*, w skrócie *MVVM*). Wprowadzał on *model widoku* oraz powiązania danych (określane wspólnie jako *mechanizm wiązania danych*, ang. *binder*). Model MVVM to szablon pozwalający na tworzenie aplikacji klienckich cechujących się sprawniejszą obsługą interakcji z użytkownikiem i szybszymi reakcjami oraz unikanie kosztownego powielania kodu i działań w obrębie całej architektury aplikacji. Aplikacje tworzone na podstawie modelu MVVM można także łatwiej testować przy użyciu testów jednostkowych. Pomimo tego wszystkiego stosowanie wzorca MVVM do obsługi prostego interfejsu użytkownika może być pewną przesadą, zatem należy je starannie rozważyć.

W przypadku aplikacji internetowych projekt wzorca MVVM umożliwia pisanie oprogramowania reagującego błyskawicznie na interakcje z użytkownikiem i pozwalającego użytkownikom na nieskrępowane przechodzenie od jednego zadania do drugiego. Jak pokazałem na rysunku 1.2, także model widoku ma różne przeznaczenia. Ta konsolidacja odpowiedzialności ma jedną kluczową konsekwencję — każdy widok powiązany z modelem widoku będzie automatycznie aktualizowany. Mechanizm wiązania danych udostępnia dane i pomaga zagwarantować, że kiedy dane te się zmienią, zmiany zostaną odzwierciedlone w widoku.



Rysunek 1.2. Komponenty wzorca Model-Widok-ModelWidoku.

UWAGA Więcej informacji na temat wzorca MVVM można znaleźć na stronie Martina Fowlera poświęconego modelowi prezentacji — *https://martinfowler.com/eaaDev/PresentationModel.html*.

1.1.3. Czym są aplikacje reaktywne?

Paradygmat programowania reaktywnego nie jest wcale nowym pomysłem. Jednak dopiero stosunkowo niedawno został on wykorzystany w aplikacjach internetowych, co w dużej mierze jest związane z dostępnością frameworków JavaScript, takich jak Vue, React i Angular.

W internecie można znaleźć wiele wspaniałych informacji dotyczących teorii reaktywności, jednak nasze potrzeby dotyczą raczej bardziej konkretnej problematyki. Aby aplikację internetową można było uznać za reaktywną, powinna ona mieć następujące cechy:

- obserwować zmiany zachodzące w stanie aplikacji;
- rozgłaszać powiadomienia o zmianach stanu w obrębie całej aplikacji;
- automatycznie wyświetlać widoki w odpowiedzi na zmiany stanu;
- zapewniać szybką reakcję na interakcje prowadzone przez użytkownika.

Reaktywne aplikacje internetowe mają te cechy dzięki użyciu wzorca projektowego MVVM, technik programowania asynchronicznego oraz (tam, gdzie to możliwe) idiomów programowania funkcyjnego, co pozwala im unikać blokowania interakcji.

Choć zastosowanie wzorca MVVM wcale nie oznacza, że aplikacja będzie reaktywna, jak również reaktywność aplikacji nie oznacza, że korzysta ona ze wzorca MVVM, to jednak użycie obu tych rozwiązań ma ten sam cel: zapewnienie użytkownikom aplikacji jak najlepszych i solidnych doświadczeń. Superman i Clark Kent mogą wyglądać inaczej, jednak ze względu na swoje cechy obaj chcą tego samego. (Nie, nie powiem ci, które z pary: wzorzec MVVM i reaktywność nosi pelerynę, a które okulary).

INFORMACJA Jeśli chciałbyś dowiedzieć się czegoś więcej na temat paradygmatu programowania reaktywnego zastosowanego we frameworku Vue, zajrzyj do poradnika *Reactivity in Depth*, dostępnego na stronie *https://vuejs.org/v2/guide/reactivity.html*.

1.1.4. Kalkulator JavaScript

Abyś lepiej zrozumiał idee wiązania danych oraz reaktywności, zaczniemy od zaimplementowania kalkulatora z wykorzystaniem jedynie zwyczajnego kodu JavaScript. Kod strony z kalkulatorem przedstawiłem na listingu 1.1.

Listing 1.1. Kalkulator JavaScript: rodzial-01/calculator.html

```
<!DOCTYPE>
<html>
  <head>
    <title>Kalkulator JavaScript</title>
    <style>
     p, input { font-family: monospace; }
     p { white-space: pre; }
    </style>
  </head>
  <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);
```

Pola formularza służące do pobierania wartości x i y oraz powiązane z funkcją runCalc.

Element wyświetlający wynik obliczeń na podstawie wartości x i y.

Konstruktor służący do tworzenia instancji kalkulatora.

```
  }
};

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;

    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
            );
```

Konstruktor do tworzenia wartości używanych przez instancję kalkulatora.

Funkcja inicjująca komponent kalkulatora.

Funkcja obsługująca zdarzenia.

```
                break;
              case calc.yInput:
                newValues =
                  lastValues.copyWith(
                    Calc.yName,
                    elem.value
                  );
                break;
            }

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

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

        return lastValues;
      }

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

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

Ustawienie procedury obsługi zdarzeń keyup.

Ten kalkulator korzysta z języka JavaScript w wersji ES5 (w dalszej części książki będziemy używać nowszej wersji JavaScript — ES5/2015). Do uruchomienia skryptu używamy wyrażenia funkcyjnego, które zostaje natychmiast wykonane. Do przechowywania wartości jest używany konstruktor, a każde zdarzenie `keyup` jest obsługiwane przez funkcję `handleCalcEvent`.

1.1.5. Kalkulator w wersji Vue

Nie zwracaj zbytniej uwagi na składnię kolejnego przykładu, w którym wykorzystaliśmy bibliotekę Vue, gdyż jego celem nie jest wyjaśnienie ci wszystkiego, co dzieje się w kodzie, lecz jedynie porównanie obu implementacji. Dlatego, jeśli tylko w miarę dobrze rozumiesz działanie przykładu z listingu 1.2, to kod kalkulatora korzystającego z Vue powinien być zrozumiały, przynajmniej na poziomie teorii.

Listing 1.2. Kalkulator Vue: rozdzial-01/calculatorvue.html

```
<!DOCTYPE html>
<html>
<head>
```

```
  <title>Kalkulator Vue.js</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">
    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>
```

To uchwyt DOM dla aplikacji.

To są pola formularza używane przez aplikację.

W tym elemencie span będzie wyświetlany wynik obliczeń.

Znacznik script dodający do strony bibliotekę Vue.js.

Inicjalizacja aplikacji.

Określenie elementu łączącego aplikację z DOM.

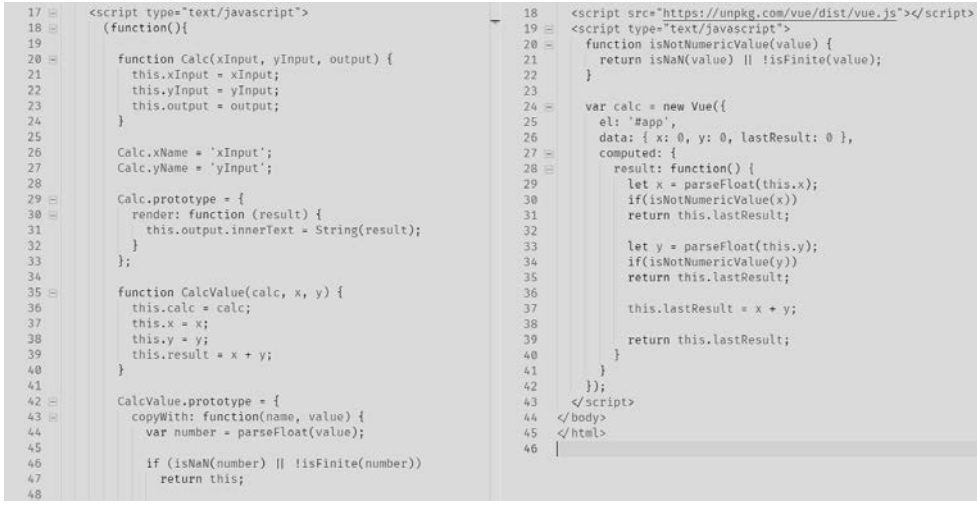
Ten wiersz określa zmienne dodawane do aplikacji.

Obliczenia są wykonywane przy użyciu właściwości computed.

1.1.6. Porównanie wersji JavaScript i Vue

Kod obu przedstawionych implementacji kalkulatora jest, w przeważającej części, inny. Każdy z fragmentów pokazanych na rysunku 1.3 jest dostępny w przykładach dołączonych do książki, w katalogu odpowiadającym numerowi rozdziału; dzięki temu możesz każdy z nich uruchomić i porównać ich działanie.

Kluczową różnicą pomiędzy obiema aplikacjami jest sposób wyzwalania aktualizacji ostatecznych wyników oraz sposób, w jaki wyniki te trafiają z powrotem na stronę WWW. W przypadku aplikacji korzystającej z Vue za wszystkie aktualizacje i obliczenia na stronie odpowiada jedno powiązanie — `v-model`. Kiedy tworzymy instancję



```
<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;
```

```
<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>
</body>
</html>
```

Rysunek 1.3. Porównanie reaktywnego kalkulatora samodzielnie napisanego w języku JavaScript (z lewej) oraz korzystającego z biblioteki Vue.js (z prawej)

aplikacji z użyciem wyrażenia `new Vue({ ... })`, Vue zbada kod JavaScript oraz kod HTML strony i utworzy wszelkie powiązania danych i zdarzeń niezbędne do działania aplikacji.

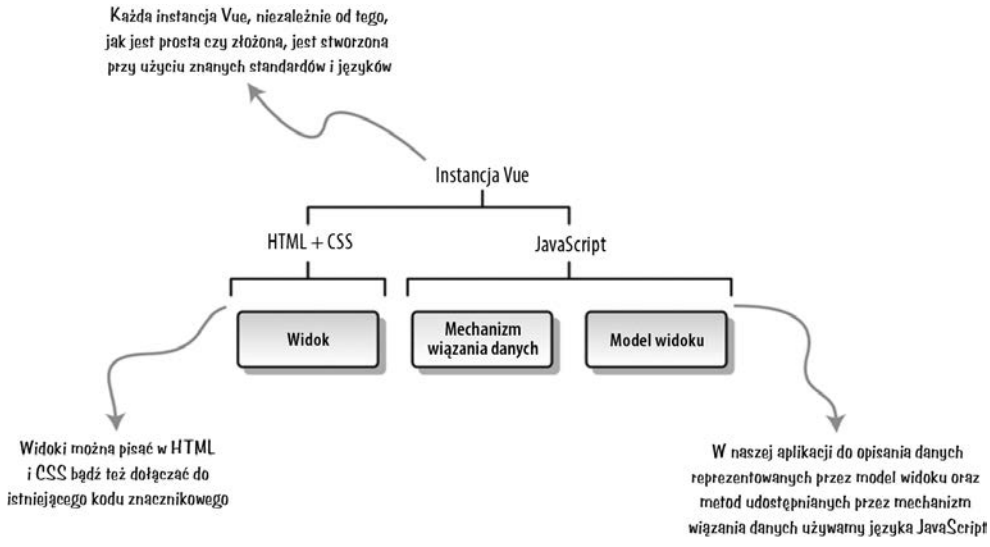
1.1.7. W jaki sposób Vue ułatwia użycie wzorca MVVM oraz zapewnienie reaktywności?

Biblioteka Vue jest czasami określana jako *framework progresywny*, co, ogólnie rzecz biorąc, oznacza, że można ją dodać do istniejącej strony WWW do wykonywania prostych zadań bądź też można jej w całości używać jako podstawy wielkich aplikacji internetowych.

Niezależnie od tego, w jaki sposób będziesz chciał używać Vue w swoim projekcie, każda z aplikacji korzystających z tej biblioteki musi mieć przynajmniej jedną *instancję Vue*. W najprostszych aplikacjach będzie używana jedna instancja Vue, obsługująca wszelkie powiązania pomiędzy wskazanym kodem HTML a magazynem danych opisane w modelu widoku (patrz rysunek 1.4).

Pojedyncza instancja Vue, stworzona wyłącznie przy wykorzystaniu technologii internetowych, istnieje w całości po stronie klienta — w przeglądarce użytkownika. Najważniejszą konsekwencją jest to, że nie jesteśmy uzależnieni od komunikacji z serwerem podczas aktualizowania widoków, wykonywania logiki biznesowej ani wykonywania jakichkolwiek innych zadań związanych z widokiem czy modelem widoku. Pamiętając o tym, wróćmy jeszcze raz do naszego przykładu przesyłania formularza.

Prawdopodobnie najbardziej uderzającą zmianą związaną z kliencką architekturą MVC jest to, że podczas całej sesji z użytkownikiem rzadko kiedy (jeśli w ogóle) konieczne będzie odświeżanie strony w przeglądarce. Dzięki temu, że zarówno widok, jak i model widoku oraz powiązania danych są zaimplementowane w całości w HTML i JavaScripcie, nasza aplikacja może asynchronicznie przekazywać zadania do modelu, co



Rysunek 1.4. W typowej instancji Vue kod HTML, dzięki utworzeniu odpowiedniego powiązania danych, zostaje powiązany z danymi przechowywanymi w modelu widoku

zapewnia użytkownikom możliwość nieskrępowanego działania. Kiedy model zwróci nowe dane, powiązania zdefiniowane przez Vue wyzwolą wszelkie aktualizacje, które muszą zostać wprowadzone w widoku.

Prawdopodobnie najważniejszą rolą Vue jest ułatwianie interakcji z użytkownikiem przez tworzenie i obsługę powiązań pomiędzy przygotowanymi widokami a danymi przechowywanymi w modelu widoku. Pod tym względem, jak zobaczysz w naszej pierwszej aplikacji, Vue może stanowić solidny fundament dla każdej reaktywnej aplikacji internetowej.

1.2. Dlaczego Vue.js?

Podczas uruchamiania nowego projektu trzeba podjąć wiele decyzji. Jedną z najważniejszych jest wybór używanego frameworka lub biblioteki. Dla agencji, a nawet dla samodzielnie pracującego programisty wybór narzędzia odpowiedniego do wykonania zadania ma niezwykle istotne znaczenie. Na szczęście biblioteka Vue.js jest wszechstronna i doskonale sprawdzi się w wielu różnych sytuacjach.

Poniżej przedstawiłem kilka najczęściej pojawiających się obaw, z jakimi mogą spotkać się niezależni programiści oraz firmy programistyczne podczas rozpoczynania prac nad nowym projektem, wraz z informacjami o tym, w jaki sposób Vue ułatwia rozwiązanie tych problemów, bądź to bezpośrednio, bądź też jako element większego trendu do tworzenia reaktywnych aplikacji internetowych.

- *Nasz zespół nie radzi sobie zbyt dobrze w stosowaniu frameworków internetowych.* Jedną z największych zalet stosowania Vue w projektach jest to, że nie wymaga ona żadnej specjalistycznej wiedzy. Każda aplikacja Vue jest tworzona z użyciem języków HTML, CSS i JavaScript — doskonale znanych narzędzi pozwalających

od samego początku uzyskać wysoką efektywność. Nawet zespoły o bardzo małym doświadczeniu w tworzeniu rozwiązań klienckich dzięki znajomości wzorca MVC mogą szybko i sprawnie zacząć korzystać ze wzorca MVVM.

- *Dysponujemy istniejącymi rozwiązaniami, z których wciąż chcielibyśmy korzystać.* Bez obaw — absolutnie nie trzeba rezygnować z pieczołowicie przygotowanych stylów CSS czy też zaimplementowanego wcześniej komponentu karuzeli. Niezależnie od tego, czy chcemy użyć Vue w istniejącym projekcie z wieloma zależnościami, czy też chcemy stworzyć nowy projekt i wykorzystać w nim inne biblioteki, które znamy, Vue w niczym nam nie będzie przeszkadzać. Wciąż możemy używać frameworków CSS, takich jak Bootstrap czy Bulma, możemy zostawić w aplikacji bibliotekę jQuery lub komponenty Backbone i dołączyć do niej ulubioną bibliotekę do obsługi żądań HTTP lub obsługi obietnic czy też biblioteki w inny sposób rozszerzające możliwości funkcjonalne Vue.js.
- *Musimy błyskawicznie stworzyć prototyp aplikacji i ocenić reakcje użytkowników.* Jak się przekonałeś na przykładzie pierwszej przedstawionej aplikacji Vue, wszystkim, czego potrzeba do rozpoczęcia korzystania z tej biblioteki, jest dołączenie Vue.js do zwyczajnej strony WWW. Korzystanie z niej nie wymaga żadnych złożonych narzędzi do budowy aplikacji! Przedstawienie użytkownikom prototypu aplikacji może zająć tydzień lub dwa od uruchomienia projektu, co pozwoli na szybkie zebranie opinii i przejście do następnych cykli pracy.
- *Nasz produkt jest używany niemal wyłącznie na urządzeniach mobilnych.* Zminimalizowana i spakowana wersja Vue.js zajmuje 24 kB, a to naprawdę niewiele, jak na bibliotekę do tworzenia aplikacji klienckich. Bez problemu będzie można ją przekazywać przez połączenia komórkowe. Nowością wprowadzoną w Vue 2 jest generowanie po stronie serwera (ang. *server-side rendering*, w skrócie *SSR*). Strategia ta oznacza, że początkowa ilość danych wczytywanych przez aplikację może zostać ograniczona do minimum, dzięki czemu później nowe widoki i zasoby będą musiały być pobierane wyłącznie w razie konieczności. Połączenie SSR z wydajnym przechowywaniem komponentów w pamięci podręcznej w jeszcze większym stopniu ogranicza zużycie przepustowości.
- *Nasz produkt ma unikatowe i niestandardowe możliwości funkcjonalne.* Aplikacje Vue, tworzone od podstaw pod kątem modularności i możliwości rozszerzania, są komponentami nadającymi się do wielokrotnego użycia. Vue zapewnia także możliwość rozszerzania komponentów przy wykorzystaniu dziedziczenia, pozwala na dodawanie możliwości funkcjonalnych przy użyciu wstawek (ang. *mix-ins*) oraz umożliwia rozszerzanie możliwości samej biblioteki za pomocą wtyczek i niestandardowych dyrektyw.
- *Mamy bardzo dużą bazę klientów i wydajność ma kluczowe znaczenie.* Obecnie, po ostatnich modyfikacjach, wprowadzonych w celu poprawy niezawodności, wydajności i szybkości działania, Vue korzysta z wirtualnego DOM-u. Oznacza to, że Vue najpierw wykonuje operacje na reprezentacji DOM, która nie jest dołączona do przeglądarki, a następnie „kopiuje" zmiany do widzialnego widoku. W efekcie Vue na ogół osiąga lepszą wydajność działania niż inne biblioteki do tworzenia interfejsu użytkownika. Ponieważ ogólne testy często są zbyt abs-

trakcyjne, zachęcam klientów do wybrania kilku typowych przypadków użycia i kilku skrajnych, opracowania scenariusza testowego oraz samodzielnego zmierzenia wydajności działania Vue. Więcej informacji na temat wirtualnego DOM-u i jego porównanie z konkurencyjnymi rozwiązaniami można znaleźć na stronie *https://vuejs.org/v2/guide/comparison.html*.

- *Korzystamy z opracowanego procesu budowy, testowania i wdrażania aplikacji.* Te zagadnienia przedstawię wyczerpująco w ostatnim rozdziale książki, jednak wnioski są takie, że Vue można łatwo zintegrować z wieloma najpopularniejszymi narzędziami do budowy aplikacji (Webpack, Browserify i innymi) oraz frameworkami testowymi (takimi, jak Karma, Jasmine). W wielu przypadkach testy jednostkowe napisane na istniejące frameworki można przenosić bez żadnych modyfikacji. Tym, którzy zaczynają projekt od początku i chcą skorzystać z takich narzędzi, Vue zapewnia szablony projektów umożliwiające łatwą integrację z nimi. Najprościej rzecz ujmując, dodanie czy też adaptacja Vue do istniejących projektów są bardzo łatwe.
- *Co zrobić, jeśli podczas prac nad projektem lub po ich zakończeniu będziemy potrzebowali pomocy?* Dwie ogromnie ważne korzyści ze stosowania biblioteki Vue to jej społeczność oraz „ekosystem" wsparcia. Biblioteka ma świetną dokumentację, i to zarówno w formie dokumentów dostępnych w internecie, jak i komentarzy bezpośrednio w kodzie, a zespół twórców jest aktywny i chętnie służy pomocą. Zapewne jeszcze ważniejsze jest to, że równie mocna i aktywna jest społeczność programistów korzystających z Vue. W serwisach takich, jak Gitter czy internetowe forum Vue, można znaleźć bardzo wiele osób chętnych do pomocy, a lista dostępnych wtyczek, mechanizmów integracyjnych i rozszerzeń wzbogacających bibliotekę o często stosowane rozwiązania wydłuża się każdego dnia.

Sam zadawałem wiele takich pytań podczas prac nad własnymi projektami, a teraz mogę rekomendować zastosowanie biblioteki Vue niemal we wszystkich projektach, nad którymi pracuję. Mam nadzieję, że kiedy już w pełni opanujesz Vue po lekturze tej książki, będziesz mógł z pełnym przekonaniem polecić wykorzystanie jej w swoim następnym projekcie.

1.3. Dalsze przemyślenia

W tym wprowadzającym rozdziale poruszyliśmy całkiem dużo zagadnień. Jeśli dopiero zacząłeś zajmować się tworzeniem aplikacji internetowych, to mogło to być twoje pierwsze spotkanie z architekturą MVVM lub programowaniem reaktywnym, jednak pokazałem ci, że tworzenie aplikacji reaktywnych wcale nie musi być tak przerażające, jak można by sądzić po używanym żargonie.

Prawdopodobnie jednak najważniejszy wniosek, jaki należy wyciągnąć z lektury tego rozdziału, nie jest związany z samą biblioteką Vue. Jest nim to, w jaki sposób reaktywność ułatwia pisanie aplikacji. Bardzo korzystne jest także to, że aplikacje tego typu ograniczają ilość powtarzającego się kodu interfejsu użytkownika, który trzeba napisać.

To, że nie trzeba samodzielnie implementować obsługi wszystkich interakcji z użytkownikiem, pozwala skoncentrować się na modelowaniu danych oraz projektowaniu interfejsu użytkownika. Powiązanie ich staje się dzięki Vue banalnie proste.

Jeśli jesteś podobny do mnie, to najprawdopodobniej wymyślasz teraz tysiące sposobów ulepszenia naszej skromnej aplikacji. To bardzo dobrze — koniecznie powinieneś poeksperymentować i pobawić się z jej kodem. Oto kilka rzeczy, które mi przychodzą do głowy, kiedy analizuję jej kod:

- W jaki sposób można by uniknąć konieczności powtarzania łańcuchów znaków w tak wielu miejscach?
- Czy można czyścić domyślną zawartość pola tekstowego, kiedy użytkownik do niego przejdzie? A co z jej ponownym wyświetleniem w polu, jeśli użytkownik je opuści bez wpisywania żadnej wartości?
- Czy istnieje możliwość uniknięcia ręcznej obsługi każdego z pól do wprowadzania danych?

Odpowiedzi na wszystkie te pytania znajdziesz w drugiej części książki. Biblioteka Vue została zaprojektowana tak, by mogła rosnąć wraz z nami — programistami — lecz także wraz z naszym kodem, dlatego zawsze będziemy analizować inne strategie, porównywać ich mocne i słabe strony oraz uczyć się, jak określać, co będzie najlepszą praktyką w danej sytuacji.

No dobrze, zobaczmy zatem, jak możemy usprawnić to, co udało się nam napisać w tym rozdziale!

Podsumowanie

- Krótka historia działania modeli, widoków i kontrolerów oraz sposobów ich współdziałania w Vue.js.
- Wyjaśnienie, jak użycie Vue.js pozwala zaoszczędzić czas podczas prac nad tworzeniem aplikacji.
- Informacje, dlaczego warto rozważyć użycie Vue.js w następnym projekcie.

Vue.js jest lekkim, nowoczesnym frameworkiem rozwijanym przez informatyków, projektantów, inżynierów i osoby, które zawodowo interesują się wrażeniami użytkowników. Służy do tworzenia reaktywnych i atrakcyjnych wizualnie klienckich aplikacji internetowych. Pozwala na budowanie zarówno dość prostych, jak i bardzo złożonych systemów. Udostępnia powiązania dwukierunkowe, reaktywny interfejs użytkownika oraz logiczną strukturę projektów. Umożliwia pisanie czytelnego i zwięzłego kodu. Tworzenie aplikacji z użyciem Vue w języku JavaScript i z biblioteką do zarządzania stanem Vuex jest czystą przyjemnością.

Ta książka to praktyczny przewodnik po frameworku Vue.js przeznaczonym dla programistów zaznajomionych z JavaScriptem, HTML-em i CSS-em. Pozwala na szybkie poznanie tego narzędzia. Przedstawiono w niej zasady łatwego zarządzania stanem z użyciem biblioteki Vuex oraz techniki budowania niestandardowych dyrektyw. Poszczególne zagadnienia zilustrowano praktycznymi fragmentami kodu źródłowego. Aby umożliwić czytelnikom płynne przejście do tworzenia kompletnych systemów, kolejne tajniki Vue.js pokazano na przykładzie stopniowo rozwijanej aplikacji sklepu internetowego z koszykiem zakupowym, interfejsem magazynowym i modułem do zarządzania magazynem.

W tej książce między innymi:

- podstawy Vue.js
- instancje Vue.js i budowa aplikacji
- model reaktywny, komponenty i trasowanie
- animacje, efekty przejść i inne możliwości frameworka
- modelowanie danych i testowanie aplikacji

Vue.js. Napisz piękną aplikację!

Erik Hanchett jest programistą aplikacji, blogerem i wiecznym studentem. Okazjonalnie próbuje swoich sił w biznesie. Napisał kilka cenionych książek o programowaniu. Ma ponad 10-letnie doświadczenie w pisaniu kodu.

Benjamin Listwon od ponad 20 lat jest architektem oprogramowania. Pracuje też jako programista i projektant oraz konsultant, a także pisze książki. Uwielbia się uczyć nowych technologii i dzielić się wiedzą z innymi.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-5734-1

9 788328 357341

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 59,00 zł