

Olga Filipova

# Vue.js 2

Tworzenie  
reaktywnych  
aplikacji WWW

Helion 

Packt 

Tytuł oryginału: Learning Vue.js 2

Tłumaczenie: Krzysztof Wołowski

ISBN: 978-83-283-3874-6

Copyright © Packt Publishing 2016

First published in the English language under the title 'Learning Vue.js 2 - (9781786469946)'

Polish edition copyright © 2018 by Helion SA  
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto: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/vuejs2>  
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

# Spis treści

<b>Podziękowania</b>	<b>11</b>
<b>O autorze</b>	<b>13</b>
<b>O recenzencie</b>	<b>14</b>
<b>Wstęp</b>	<b>15</b>
<b>Rozdział 1. Zakupy z Vue.js</b>	<b>19</b>
<b>Terminologia</b>	<b>20</b>
<b>Historia Vue.js</b>	<b>22</b>
<b>Rzeczy, które musisz wiedzieć o Vue</b>	<b>22</b>
<b>Idziemy na zakupy!</b>	<b>23</b>
Implementacja listy zakupów przy użyciu jQuery	23
Implementacja listy zakupów za pomocą Vue.js	27
Analiza wiązania danych za pomocą narzędzi programisty	28
Modyfikacja modelu po wprowadzeniu danych przez użytkownika	29
Wyświetlanie listy elementów za pomocą dyrektywy v-for	30
Zaznaczanie elementów listy zakupów	31
Dodawanie nowych elementów do listy zakupów za pomocą dyrektywy v-on	32
<b>Korzystanie z Vue.js w istniejącym projekcie</b>	<b>34</b>
<b>Vue.js 2.0!</b>	<b>37</b>
<b>Projekty, w których wykorzystano Vue.js</b>	<b>38</b>
Grammarly	38
Optimizely	39
FilterBlend	40
PushSilver	41
<b>Organizacja książki</b>	<b>41</b>

<b>Zarządzamy czasem!</b>	<b>43</b>
Zmiana tytułu przy użyciu właściwości obliczanych	45
Dopełnienie wartości za pomocą właściwości obliczanych	47
Kontrola stanu przy użyciu przycisków startu, pauzy i stopu	48
<b>Ćwiczenie</b>	<b>50</b>
<b>Podsumowanie</b>	<b>50</b>
<b>Rozdział 2. Podstawy — instalacja i użytkowanie</b>	<b>51</b>
<b>Wzorzec architektoniczny MVVM</b>	<b>52</b>
<b>Metoda defineProperty, gettery i settery</b>	<b>53</b>
<b>Porównanie z innymi frameworkami</b>	<b>56</b>
React	56
Angular	58
Vue	58
<b>Podstawy Vue.js</b>	<b>59</b>
Komponenty	59
Dyrektywy Vue.js	63
Wtyczki Vue.js	64
Ćwiczenie	67
Stan aplikacji i Vuex	67
vue-cli	68
Wtyczki Vue dostępne w środowiskach IDE	70
<b>Instalacja, użytkowanie i debugowanie aplikacji Vue.js</b>	<b>70</b>
Instalacja Vue.js	70
<b>Debugowanie aplikacji Vue</b>	<b>80</b>
<b>Tworzenie szkieletu aplikacji</b>	<b>82</b>
Szkielet aplikacji Lista zakupów	82
Szkielet aplikacji Pomodoro	85
<b>Ćwiczenie</b>	<b>85</b>
<b>Podsumowanie</b>	<b>85</b>
<b>Rozdział 3. Komponenty — zasada działania i zastosowanie</b>	<b>87</b>
<b>Powrót do komponentów</b>	<b>87</b>
<b>Korzyści z używania komponentów</b>	<b>88</b>
Deklaracja szablonów w formacie HTML	88
Obsługa właściwości data i el w komponencie	89
Zasięg komponentów	90
Komponenty zawierające inne komponenty	92
<b>Przebudowa aplikacji Lista zakupów za pomocą prostych komponentów</b>	<b>96</b>
Definiowanie szablonów dla wszystkich komponentów	97
Definiowanie i rejestrowanie komponentów	99
<b>Ćwiczenie</b>	<b>100</b>
<b>Komponenty jednoplikowe</b>	<b>100</b>
Wtyczki dla środowisk IDE	102
Styl i zasięg	102
Automatyczne odświeżanie	103
Preprocesory	104

<b>Przebudowa aplikacji Lista zakupów za pomocą jednoplikowych komponentów</b>	<b>105</b>
AddItemComponent	108
Konfiguracja komponentów ItemComponent i ItemsComponent	109
<b>Ćwiczenie</b>	<b>111</b>
<b>Przebudowa aplikacji Pomodoro za pomocą komponentów jednoplikowych</b>	<b>111</b>
<b>Reaktywne wiązanie efektów przejścia CSS</b>	<b>115</b>
<b>Podsumowanie</b>	<b>117</b>
<b>Rozdział 4. Reaktywność — wiązanie danych</b>	<b>119</b>
<b>Wiązanie danych raz jeszcze</b>	<b>119</b>
<b>Interpolacja danych</b>	<b>120</b>
Dodajemy nagłówek ze stanem aplikacji	121
Ćwiczenie	122
<b>Wyrażenia i filtry</b>	<b>122</b>
Wyrażenia	122
Filtry	126
Ćwiczenie	127
<b>Dyrektywy raz jeszcze</b>	<b>127</b>
Wiązanie dwukierunkowe przy użyciu dyrektywy v-model	128
Dwukierunkowe wiązanie między komponentami	129
Wiązanie atrybutów za pomocą dyrektywy v-bind	129
Wyświetlanie warunkowe przy użyciu dyrektyw v-if i v-show	131
Przetwarzanie tablicy za pomocą dyrektywy v-for	134
Detektory zdarzeń i dyrektywa v-on	141
Skróty	145
Ćwiczenie	146
<b>Koty</b>	<b>146</b>
<b>Podsumowanie</b>	<b>146</b>
<b>Rozdział 5. Vuex — zarządzanie stanem aplikacji</b>	<b>149</b>
<b>Komunikacja typu rodzic – dziecko między komponentami, zdarzenia oraz łamigłówka</b>	<b>149</b>
<b>Dlaczego potrzebujemy globalnego magazynu stanu?</b>	<b>156</b>
<b>Czym jest Vuex?</b>	<b>156</b>
<b>Jak działa magazyn i co jest w nim takiego szczególnego?</b>	<b>157</b>
<b>Pozdrowienia z magazynu</b>	<b>159</b>
Stan magazynu i gettery	164
Mutacje	168
Akcje	169
<b>Instalacja magazynu Vuex i jego wykorzystanie w naszych aplikacjach</b>	<b>174</b>
<b>Zastosowanie magazynu Vuex w aplikacjach Lista zakupów i Pomodoro</b>	<b>176</b>
<b>Zastosowanie magazynu Vuex w aplikacji Pomodoro</b>	<b>180</b>
Ożywiamy przyciski startu, pauzy i stopu	180
Obsługa minut i sekund w aplikacji Pomodoro	185
Tworzenie zegara Pomodoro	188
Modyfikacja kota	190
<b>Podsumowanie</b>	<b>193</b>

<b>Rozdział 6. Wtyczki — buduj dom ze swoich własnych cegieł</b>	<b>195</b>
<b>Specyfika wtyczek Vue</b>	<b>195</b>
<b>Zastosowanie wtyczki vue-resource w aplikacji Lista zakupów</b>	<b>196</b>
Tworzenie prostego serwera	197
Instalacja vue-resource, tworzenie zasobów oraz metod	198
Pobieranie list zakupów przy uruchamianiu aplikacji	199
Aktualizowanie danych na serwerze po zmianach	201
Tworzenie nowej listy zakupów	206
Usuwanie istniejących list zakupów	210
Ćwiczenie	212
<b>Niestandardowa wtyczka w aplikacji Pomodoro</b>	<b>212</b>
Wtyczka NoiseGenerator	213
Zastosowanie wtyczki w aplikacji Pomodoro	216
Przycisk do przełączania dźwięku	218
Ćwiczenie	222
<b>Podsumowanie</b>	<b>222</b>
<b>Rozdział 7. Testy — sprawdzanie poprawności działania aplikacji</b>	<b>225</b>
<b>Dlaczego testy jednostkowe?</b>	<b>225</b>
<b>Testy jednostkowe dla aplikacji Vue</b>	<b>228</b>
<b>Testowanie jednostkowe aplikacji Lista zakupów</b>	<b>229</b>
Testowanie akcji, getterów i mutacji	230
Kryteria dobrego testu	233
Stopień pokrycia kodu	234
Symulowanie odpowiedzi serwera i tworzenie asynchronicznych testów	237
Testowanie komponentów	243
<b>Tworzenie testów jednostkowych dla aplikacji Pomodoro</b>	<b>245</b>
<b>Co to są testy E2E?</b>	<b>249</b>
<b>Nightwatch do testów E2E</b>	<b>249</b>
<b>Tworzenie testów E2E dla aplikacji Pomodoro</b>	<b>250</b>
<b>Podsumowanie</b>	<b>253</b>
<b>Rozdział 8. Wdrażanie — startujemy w sieci!</b>	<b>255</b>
<b>Wdrażanie oprogramowania</b>	<b>255</b>
GitHub — co to?	258
Travis — co to?	258
Heroku — co to?	258
<b>Przeniesienie aplikacji do repozytorium GitHuba</b>	<b>258</b>
<b>Konfiguracja procesu ciągłej integracji za pomocą Travisa</b>	<b>260</b>
<b>Wdrażanie aplikacji Pomodoro</b>	<b>265</b>
Dziennik zdarzeń	266
Przygotowanie aplikacji do uruchomienia w Heroku	267
<b>Wdrażanie aplikacji Lista zakupów</b>	<b>270</b>
Używanie Heroku lokalnie	272
<b>Podsumowanie</b>	<b>273</b>

<b>Rozdział 9. Co dalej?</b>	<b>275</b>
<b>Co już wiemy</b>	<b>275</b>
<b>Vue 2.0</b>	<b>277</b>
<b>Aplikacje raz jeszcze</b>	<b>278</b>
Aplikacja Lista zakupów	278
Aplikacja Pomodoro	279
<b>Dlaczego to dopiero początek?</b>	<b>281</b>
Dodawanie funkcji do naszych aplikacji	281
Upiększamy nasze aplikacje	284
Rozszerzanie dostępności naszych aplikacji na inne urządzenia	285
<b>Podsumowanie</b>	<b>285</b>
<b>Dodatek A. Rozwiązania ćwiczeń</b>	<b>287</b>
<b>Ćwiczenie do rozdziału 1.</b>	<b>287</b>
<b>Ćwiczenia do rozdziału 2.</b>	<b>289</b>
Poszerzenie możliwości wtyczki MathPlugin	289
Zegar Pomodoro jako aplikacja Chrome	290
<b>Ćwiczenia do rozdziału 3.</b>	<b>290</b>
Ćwiczenie 1.	290
Ćwiczenie 2.	292
<b>Skorowidz</b>	<b>293</b>





# Zakupy z Vue.js

*Vue.js to framework JavaScript ułatwiający tworzenie niezwykłych aplikacji WWW.*

*Vue.js to biblioteka JavaScript do tworzenia interfejsów WWW.*

*Vue.js to narzędzie, które wykorzystuje architekturę MVVM.*

Według strony Simplified JavaScript Jargon Vue.js to biblioteka JavaScript służąca do tworzenia interfejsów użytkownika (widoków) na podstawie modeli danych (<http://jargon.js.org/glossary/VUEJS.md>).

Oficjalna strona Vue.js (<https://vuejs.org/>) kilka miesięcy temu charakteryzowała Vue.js jako zbiór reaktywnych komponentów dla nowoczesnych interfejsów WWW (zob. rysunek 1.1).



**Rysunek 1.1.** Strona domowa Vue.js kilka miesięcy temu

Teraz natomiast określa Vue.js jako progresywny framework JavaScript (zob. rysunek 1.2).



Rysunek 1.2. Strona domowa Vue.js dziś

Czym zatem jest Vue.js? Frameworkiem? Narzędziem? Biblioteką? Czy można za jego pomocą tworzyć całe aplikacje WWW, czy tylko rozszerzać ich funkcjonalność? Czy warto porzucić dla niego swój ulubiony framework? A jeżeli tak, to dlaczego? Czy można go używać w swoim projekcie obok innych narzędzi? Co tak naprawdę daje?

W tym rozdziale spróbuję udzielić odpowiedzi na wszystkie te pytania. Na razie jednak nie będę szczegółowo omawiać Vue.js — ograniczę się do kilku prostych przykładów.

W tym rozdziale:

- poznamy architekturę frameworka Vue.js, a także jego historię;
- dowiemy się, gdzie jest wykorzystywany;
- zbudujemy za jego pomocą prostą listę zakupów i porównamy tę implementację z implementacją takiej samej aplikacji utworzonej przy użyciu jQuery;
- utworzymy prostą aplikację — zegar Pomodoro;
- wykonamy krótkie, proste ćwiczenie.

## Terminologia

Ta książka jest pełna modnej nowomowy, dziwnych skrótów i innych hipsterskich kombinacji liter. Ale nie bój się! Aby tworzyć aplikacje w Vue.js lub w dowolnym innym frameworku, nie musisz znać ich wszystkich na pamięć! Jeżeli jednak w jakimś punkcie książki pogubisz się w terminologii, w każdym momencie możesz wrócić do poniższego słowniczka:

- **Stan aplikacji:** Jest to globalny stan aplikacji. Dane stanu aplikacji są inicjalizowane podczas jej uruchamiania. Każdy komponent aplikacji może je odczytać, ale nie może ich łatwo zmienić. Każdy element stanu posiada mutację — funkcję wywoływaną po wystąpieniu określonych zdarzeń w komponentach aplikacji.
- **Bootstrap:** Zestaw stylów i narzędzi JavaScript wspomagających tworzenie eleganckich responsywnych aplikacji bez konieczności bezpośredniego stosowania reguł CSS.
- **Sieci dystrybucji treści (CDN, ang. Content Distribution Network):** Specjalny serwer, którego zadaniem jest dostarczenie użytkownikom danych, charakteryzujący się wysoką dostępnością i wydajnością. Twórcy bibliotek i frameworków udostępniają

je często poprzez CDN, ponieważ w instrukcji instalacji wystarczy wtedy podać adres URL danego zasobu. Vue.js znajduje się pod adresem <https://npmcdn.com/>. Ten niezawodny serwer o globalnym zasięgu udostępnia wszystkie pakiety przeznaczone dla npm.

- **Komponenty:** Wyodrębnione fragmenty aplikacji posiadające własne dane oraz widoki, które mogą być użyte wielokrotnie. Ich rolę w procesie budowy aplikacji można porównać do roli cegieł przy budowie domu.
- **Kaskadowe arkusze stylów (CSS):** Zestaw stylów stosowanych w dokumencie HTML, aby nadać mu odpowiedni wygląd.
- **Deklaratywne widoki:** Widoki umożliwiające bezpośrednie wiązanie danych modeli JavaScript z ich reprezentacją.
- **Dyrektywy:** Specjalne atrybuty elementów HTML w Vue.js, które umożliwiają różne sposoby wiązania danych.
- **Model DOM** (ang. *Document Object Model*): Konwencja reprezentacji węzłów w takich językach znaczników, jak HTML, XML czy XHTML. Węzły dokumentu są zorganizowane w strukturę drzewa DOM. Gdy mówimy o interakcji z modelem DOM, mamy na myśli interakcję z elementami HTML.
- **npm:** Menedżer pakietów JavaScript umożliwiający ich wyszukiwanie, instalowanie i zarządzanie nimi.
- **Markdown:** Prosty i przyjazny dla użytkownika język pozwalający zapisać tekst bez obaw o style i znaczniki HTML. Pliki zawierające tekst zapisany w ten sposób mają rozszerzenie *.md*.
- **Wzorzec MVVM:** Wzorzec architektoniczny **Model-Widok-WidokModel** (skrótowa nazwa pochodzi od ang. *Model View ViewModel*), którego najważniejszym elementem jest **WidokModel** (ang. *ViewModel*), odgrywający rolę pośrednika między widokiem a modelem i umożliwiający przepływ danych między nimi.
- **Wzorzec MVC:** Wzorzec architektoniczny **Model-Widok-Kontroler** (skrótowa nazwa pochodzi od ang. *Model View Controller*). Umożliwia oddzielenie widoku od modelu i sposobu, w jaki przepływa między nimi informacja.
- **Jednokierunkowe wiązanie danych:** Rodzaj wiązania danych, w którym zmiany w modelu danych są automatycznie propagowane do warstwy widoku, ale nie na odwrót.
- **Szybkie prototypowanie:** Technika łatwej i szybkiej budowy makiet interfejsu użytkownika, umożliwiających niektóre podstawowe interakcje z użytkownikiem.
- **Reaktywność:** Natychmiastowa propagacja zmian danych do warstwy widoku.
- **Dwukierunkowe wiązanie danych:** Rodzaj wiązania danych, przy którym zmiany w modelu danych są automatycznie propagowane do warstwy widoku, a zmiany zachodzące w warstwie widoku są natychmiast odzwierciedlane w modelu danych.
- **Interfejs użytkownika (UI):** Zestaw komponentów graficznych umożliwiających użytkownikowi komunikację z aplikacją.
- **Vuex:** Architektura aplikacji Vue. Umożliwia proste zarządzanie stanem aplikacji.

## Historia Vue.js

Evan You, twórca Vue.js (<http://evanyou.me/>), pracował kiedyś w Google Creative Labs nad jednym z projektów. W trakcie projektowania aplikacji okazało się, że potrzebny jest szybki prototyp i dosyć rozbudowany interfejs użytkownika. Ponieważ powielanie tych samych fragmentów kodu HTML jest czasochłonne, Evan zaczął szukać narzędzi, które mogłyby go w tym wyręczyć. Ku swojemu zdziwieniu odkrył, że nie ma narzędzia, biblioteki ani frameworka nadającego się do szybkiego prototypowania! W tamtym czasie Angular był już powszechnie stosowany, React.js dopiero startował, a takie frameworki jak Backbone.js były używane jedynie w przypadku dużych aplikacji na bazie wzorca MVC. Brakowało niewielkiego i elastycznego frameworka tylko do celów szybkiego prototypowania interfejsu użytkownika.

Kiedy zdajesz sobie sprawę, że coś fajnego nie istnieje, a Ty jesteś w stanie to stworzyć — zrób to!

Framework Vue.js powstał jako narzędzie do szybkiego prototypowania. Dziś można go również wykorzystać do tworzenia złożonych, skalowalnych i reaktywnych aplikacji WWW.

Evan nie zastanawiał się ani przez chwilę. I tak powstała biblioteka przeznaczona do szybkiego prototypowania, zapewniająca prostotę i elastyczność przy reaktywnym wiązaniu danych oraz autonomiczne komponenty.

Jak każda dobra biblioteka, tak i Vue.js rozwija się i ewoluuje, dzięki czemu liczba jej funkcji stale rośnie. Obecna wersja umożliwia dołączanie i tworzenie wtyczek, tworzenie i stosowanie domieszek (ang. *mixins*) oraz dodawanie niestandardowego zachowania. Vue jest na tyle elastyczny i neutralny względem struktury aplikacji, że z całą pewnością można go uznać za framework wspierający budowę kompleksowych aplikacji WWW.

## Rzeczy, które musisz wiedzieć o Vue

Vue.js pozwala powiązać modele danych z warstwą reprezentacji. Umożliwia również wielokrotne wykorzystywanie komponentów w aplikacji.

Nie wymaga tworzenia specjalnych modeli czy kolekcji ani rejestrowania obiektów zdarzeń. Nie wymaga używania skomplikowanej składni. Nie wymaga także instalowania niekończących się kaskad zależności.

Modele to zwykłe obiekty JavaScript. Możesz je powiązać z dowolnym elementem widoku (tekstem, wprowadzanymi danymi, klasami, atrybutami i tak dalej).

Aby całość zaczęła działać, wystarczy dołączyć w projekcie plik *vue.js*. Alternatywnie możesz skorzystać z narzędzia *vue-cli*, które pozwala tworzyć i konfigurować szablony projektów na

bazie bibliotek Webpack i Browserify. Oprócz tego `vue-cli` zapewnia obsługę automatycznego odświeżania (ang. *hot reloading*), a także zawiera narzędzia dla programistów.

Możesz oddzielić warstwę widoku od stylów i logiki JavaScript albo umieścić je w jednym pliku Vue. Dla wszystkich popularnych środowisk programistycznych istnieją wtyczki do obsługi Vue.

Możesz używać dowolnych preprocesorów i tworzyć kod w standardzie ES2015. Możesz używać Vue razem z ulubionym frameworkiem lub osobno. Możesz z niego korzystać tylko po to, aby wprowadzić w aplikacji drobne zmiany, ale równie dobrze możesz tworzyć od podstaw złożone aplikacje.

Jeśli chcesz się dowiedzieć, jak Vue wypada na tle innych frameworków, takich jak Angular czy React, odwiedź stronę <http://vuejs.org/guide/comparison.html>.

Aby poznać niesamowite możliwości Vue.js, zajrzyj na stronę <https://github.com/vuejs/awesome-vue>.

## Idziemy na zakupy!

Podejrzewam, że u Ciebie też zbliża się weekend i powoli zaczynasz planować zakupy na przyszły tydzień. O ile nie jesteś człowiekiem z doskonałą pamięcią ani nie prowadzisz ascetycznego trybu życia, prawdopodobnie przed pójściem do sklepu sporządzasz listę rzeczy do kupienia. Może nawet używasz do tego jakiejś aplikacji. Dlaczego jednak nie wykorzystać w tym celu własnej? Co powiesz na to, żebyśmy ją wspólnie zaprojektowali i utworzyli? Zróbmy to! Utwórzmy własną aplikację zakupową. Zacznijmy od przygotowania szybkiego prototypu. Budowa interaktywnego prototypu dla listy zakupów nie powinna sprawić nam większych problemów.

Powinien on wyświetlać listę i pozwalać na dodawanie i usuwanie elementów, podobnie jak na liście rzeczy do zrobienia. Najpierw spróbujemy podejścia klasycznego — skorzystamy z HTML-a, CSS, JavaScriptu i jQuery. Użyjemy również frameworka Bootstrap (<http://getbootstrap.com/>), dzięki któremu nadamy interfejsowi odpowiedni wygląd bez konieczności dodawania zbyt wielu reguł CSS. (Ta książka nie jest o CSS, a praca z frameworkiem Bootstrap jest niezwykle prosta i przyjemna!).

## Implementacja listy zakupów przy użyciu jQuery

Poniżej znajduje się jedno z możliwych rozwiązań.

Kod HTML:

```
<div class="container">
  <h2>Moja lista zakupów</h2>
  <div class="input-group">
```

```
<input placeholder="dodaj produkt" type="text" class="js-new-item
↳form-control">
<span class="input-group-btn">
  <button @click="addItem" class="js-add btn btn-default"
  ↳type="button">Dodaj!</button>
</span>
</div>
<ul>
  <li>
    <div class="checkbox">
      <label>
        <input class="js-item" name="list" type="checkbox"> Marchewka
      </label>
    </div>
  </li>
  <li>
    <div class="checkbox">
      <label>
        <input class="js-item" name="list" type="checkbox"> Książka
      </label>
    </div>
  </li>
  <li class="removed">
    <div class="checkbox">
      <label>
        <input class="js-item" name="list" type="checkbox" checked>
        ↳Prezent na urodziny cioci
      </label>
    </div>
  </li>
</ul>
</div>
```

Kod CSS:

```
.container {
  width: 40%;
  margin: 20px auto 0px auto;
}

.removed {
  color: gray;
}

.removed label {
  text-decoration: line-through;
}

ul li {
  list-style-type: none;
}
```

Kod JavaScript i jQuery:

```
$(document).ready(function() {
  /**
   * Dodaj procedurę obsługi kliknięcia.
   */
  function onAdd() {
    var $ul, li, $li, $label, $div, value;

    value = $('.js-new-item').val();
    // Sprawdź, czy wprowadzono wartość.
    if (value === '') {
      return;
    }
    $ul = $('ul');
    $li = $('<li>').appendTo($ul);
    $div = $('<div>')
      .addClass('checkbox')
      .appendTo($li);
    $label = $('<label>').appendTo($div);
    $('<input>')
      .attr('type', 'checkbox')
      .addClass('item')
      .attr('name', 'list')
      .click(toggleRemoved)
      .appendTo($label);
    $label
      .append(value);
    $('.js-new-item').val('');
  }

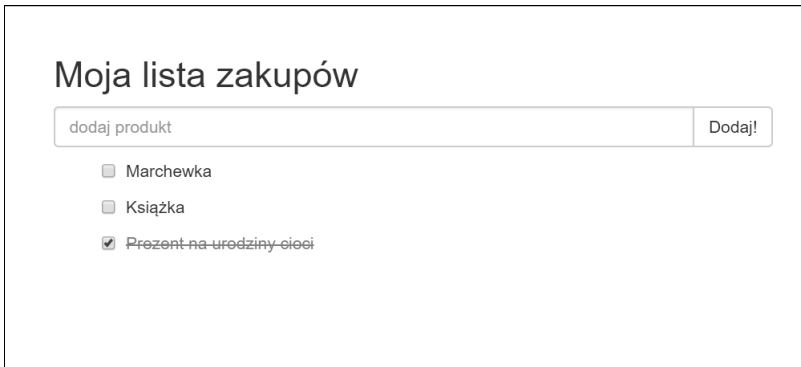
  /**
   * Procedura obsługi kliknięcia pola wyboru –
   * przełącza klasę elementu li zawierającego element.
   * @param ev
   */
  function toggleRemoved(ev) {
    var $el;

    $el = $(ev.currentTarget);
    $el.closest('li').toggleClass('removed');
  }

  $('.js-add').click(onAdd);
  $('.js-item').click(toggleRemoved);
});
```

Kod do wszystkich przykładów z tej książki dostępny jest na GitHubie pod adresem <https://github.com/PacktPublishing/Learning-Vue.js-2>.

W przeglądarce strona powinna wyglądać tak jak na rysunku 1.3.



**Rysunek 1.3.** Implementacja listy zakupów zrealizowana za pomocą HTML-a, CSS i jQuery

Popatrz na kod i na efekt w JSFiddle (<https://jsfiddle.net/chudaol/u5pcnLw9/2/>).

Jest to prosty fragment kodu HTML z nieuporządkowaną listą elementów, z których każdy zawiera pole wyboru, pole tekstowe do wprowadzania danych oraz przycisk *Dodaj!* Po każdym kliknięciu przycisku *Dodaj!* zawartość pola tekstowego jest przekształcana w element listy i dołączana na jej końcu. Kliknięcie pola wyboru przy danej pozycji powoduje przełączenie stanu między *kup* (bez zaznaczenia) a *kupione* (zaznaczone).

Dodajmy jeszcze możliwość zmiany nazwy listy zakupów (co może być przydatne, gdy w aplikacji będziemy mieli wiele list).

Potrzebujemy więc dodatkowego kodu HTML oraz dodatkowego kodu jQuery z detektorami i procedurami obsługi zdarzeń:

```
<div class="container">
  <h2>Moja lista zakupów</h2>
  <!-- ... -->
  <div class="footer">
    <hr/>
    <em>Zmień nazwę listy zakupów</em>
    <input class="js-change-title" type="text" value="Moja lista zakupów"/>
  </div>
</div>
```

```
// I kod JavaScript:
function onChangeTitle() {
  $('h2').text($('.js-change-title').val());
}
$('.js-change-title').keyup(onChangeTitle);
```

Całość znajdziesz pod adresem <https://jsfiddle.net/chudaol/47u38fvh/3/>.



## Implementacja listy zakupów za pomocą Vue.js

To był bardzo prosty przykład. Spróbujmy powtórzyć wykonane kroki, tym razem przy użyciu Vue.js. Istnieje wiele sposobów korzystania z *vue.js* w projekcie. Tu zastosujemy najprostszy, umieszczając w kodzie plik JavaScript z serwera CDN:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.3/vue.js">
</script>
```

Zacznijmy od wyświetlenia listy elementów.

Utwórz plik HTML i umieść w nim następujący kod:

```
<div id="app" class="container">
  <h2>{{ title }}</h2>
  <ul>
    <li>{{ items[0] }}</li>
    <li>{{ items[1] }}</li>
  </ul>
</div>
```

Teraz dodaj kod JavaScript:

```
var data = {
  items: ['Banany', 'Jabłka'],
  title: 'Moja lista zakupów'
};

new Vue({
  el: '#app',
  data: data
});
```

Otwórz stronę w przeglądarce. Lista powinna wyglądać tak jak na rysunku 1.4.



Rysunek 1.4. Lista zakupów zaimplementowana przy użyciu Vue.js

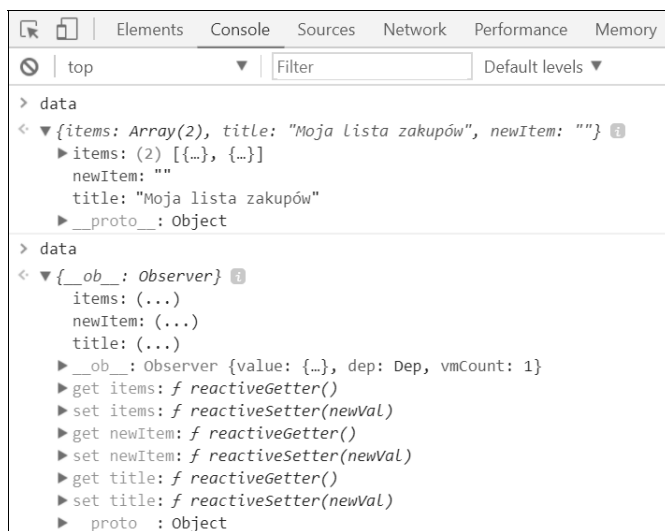
Przeanalizujmy ten przykład. Kod aplikacji Vue rozpoczyna się od słowa kluczowego `new Vue`. W jaki sposób powiązać fragment kodu HTML z danymi aplikacji? Przekazujemy instancji Vue element DOM, z którym będzie powiązana. Żaden inny element strony nie pozna magii Vue.

Jak widać, całość naszego kodu mieści się w elemencie `#app`, który jest z kolei przekazywany jako pierwszy argument w obiekcie z opcjami Vue. Argument `data` zawiera obiekty używane w kodzie w podwójnych nawiasach klamrowych (`{{}}`). Nikt, kto miał do czynienia z preprocesorami szablonów, takimi jak na przykład `handlebars` (więcej informacji na jego temat znajdziesz pod adresem <http://handlebarsjs.com/>), nie będzie miał problemu ze zrozumieniem tego zapisu.

I co z tego? — zapytasz. Czego chcesz mnie nauczyć? Jak używać preprocesorów szablonów? Dziękuję bardzo, wolę iść na piwo albo obejrzeć mecz. Zaczekaj, otwórz sobie piwo i czytaj dalej. Nie pożałujesz!

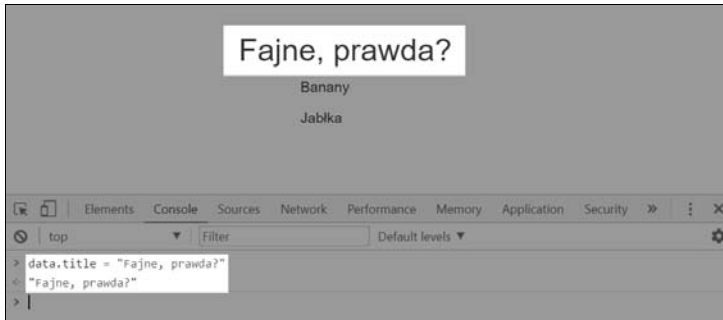
## Analiza wiązania danych za pomocą narzędzi programisty

Zobaczymy, jak działa wiązanie danych w praktyce. Otwórz w swojej przeglądarce narzędzia programisty, przejdź do kodu JavaScript i dodaj na początku skryptu punkt wstrzymania. Zauważ, że zawartość obiektu `data` przed inicjalizacją aplikacji Vue i po niej jest inna. W tym drugim przypadku obiekt `data` jest już przygotowany do reaktywnego wiązania danych (zob. rysunek 1.5).



Rysunek 1.5. Obiekt `data` przed inicjalizacją obiektu Vue i po niej

Jeżeli zmienimy teraz w konsoli narzędzi programisty właściwość `title` obiektu `data` (możemy to zrobić, ponieważ `data` to obiekt globalny), nazwa strony zostanie automatycznie zaktualizowana (zob. rysunek 1.6).



Rysunek 1.6. Wiązanie danych: zmiana właściwości obiektu powoduje natychmiastową aktualizację widoku

## Modyfikacja modelu po wprowadzeniu danych przez użytkownika

W naszym przykładzie udało się przetransportować dane z modelu w kodzie JavaScript na stronę. To był lot na trasie kod aplikacji – strona. Czy nie byłoby jednak lepiej, gdyby dane mogły podróżować w obu kierunkach?

Zobaczmy teraz, jak powiązać dane dwukierunkowo, tak aby można było zmieniać właściwości obiektu data poprzez interakcję ze stroną.

Skopiuj kod HTML odpowiedzialny za wyświetlenie nazwy listy i zmodyfikuj pole tekstowe z przykładu jQuery, dodając w nim atrybut `v-model="title"`.

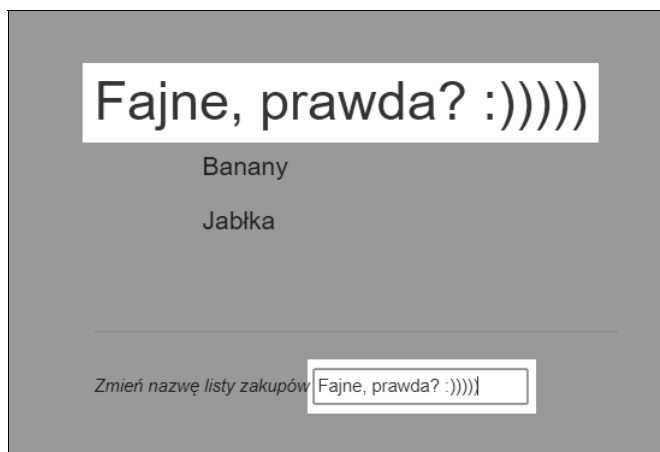
Czy słyszałeś już o dyrektywach Vue.js? Gratulacje, właśnie wykorzystałeś jedną z nich! Atrybut `v-model` to dyrektywa Vue.js, która umożliwia dwukierunkowe wiązanie danych. Więcej informacji na jej temat znajdziesz na oficjalnej stronie Vue pod adresem <http://vuejs.org/api/#v-model>.

Teraz kod HTML naszej listy zakupów wygląda następująco:

```
<div id="app" class="container">
  <h2>{{ title }}</h2>
  <ul>
    <li>{{ items[0] }}</li>
    <li>{{ items[1] }}</li>
  </ul>
  <div class="footer">
    <hr/>
    <em>Zmień nazwę listy zakupów</em>
    <input v-model="title"/>
  </div>
</div>
```

I tylko tyle!

Odśwież teraz stronę i wprowadź dane w polu tekstowym. Nazwa listy będzie automatycznie aktualizowana w trakcie pisania (zob. rysunek 1.7).



**Rysunek 1.7.** Wiązanie danych: zmiana wartości w polu tekstowym powiązanej z właściwością modelu powoduje zmianę zawartości innych elementów powiązanych z tą właściwością

Aplikacja zakupowa działa. Jednak na razie pobiera po prostu dwa elementy tablicy i tworzy z nich elementy listy. My natomiast chcielibyśmy, żeby lista była wyświetlana w całości niezależnie od jej rozmiaru.

## Wyświetlanie listy elementów za pomocą dyrektywy v-for

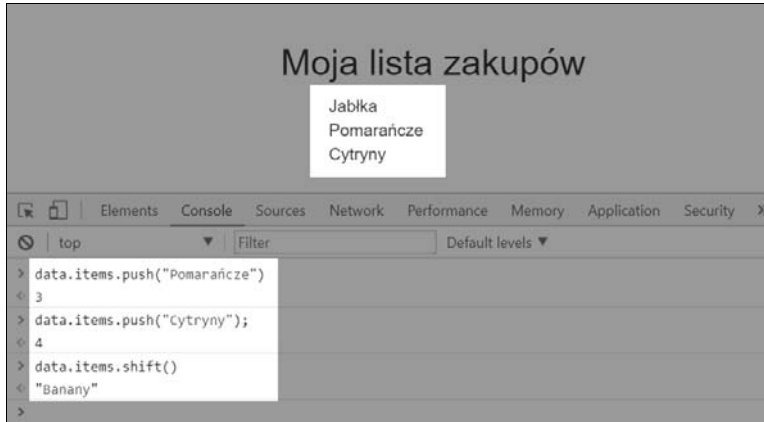
Potrzebujemy jakiegoś mechanizmu do iteracyjnego przetworzenia elementów tablicy i wyświetlenia ich wewnątrz znaczników `<ul></ul>`.

Na szczęście Vue.js oferuje wygodną dyrektywę umożliwiającą przetwarzanie struktur danych JavaScript w pętli. To dyrektywa `v-for`. Użyjemy jej w elemencie listy `<li>`. Zmodyfikuj kod listy, aby wyglądał tak jak poniżej:

```
<ul>
  <li v-for="item in items">{{ item }}</li>
</ul>
```

W dalszej części książki poznasz wiele innych przydatnych dyrektyw, takich jak `v-if`, `v-else`, `v-show`, `v-on`, `v-bind`, czytaj więc dalej.

Odśwież stronę i popatrz na listę. Strona powinna wyglądać tak samo. Teraz dodaj element do tablicy `items` w konsoli, w narzędziach programisty. Spróbuj też jakiś element usunąć. Zgodnie z oczekiwaniami wszystkie operacje na tablicy są natychmiast odzwierciedlane na stronie (zob. rysunek 1.8).



Rysunek 1.8. Wiązanie danych: zmiana tablicy powoduje zmiany na liście, z którą jest powiązana

Mamy teraz listę elementów, do której wyświetlenia na stronie potrzebowaliśmy tylko jednego wiersza kodu. Wciąż jednak brakuje pól wyboru, które pozwolą zaznaczyć zakupione rzeczy lub, w razie potrzeby, usunąć zaznaczenie.

## Zaznaczanie elementów listy zakupów

Aby to osiągnąć, musimy nieco zmodyfikować tablicę `items`. Zamiast ciągów znaków będzie ona teraz zawierać obiekty z dwiema właściwościami: `text` (określającą tekst elementu) i `checked` (określającą jego stan). Zmodyfikujemy też kod HTML, dodając w każdym elemencie listy pole wyboru.

Kod JavaScript z deklaracją danych będzie wyglądał następująco:

```
var data = {
  items: [{ text: 'Banany', checked: true },
          { text: 'Jabłka', checked: false }],
  title: 'Moja lista zakupów',
  newItem: ''
};
```

A kod HTML listy będzie miał taką postać:

```
<ul>
  <li v-for="item in items" v-bind:class="{ 'removed':item.checked }">
    <div class="checkbox">
```

```

    <label>
      <input type="checkbox" v-model="item.checked">{{ item.text }}
    </label>
  </div>
</li>
</ul>

```

Odśwież stronę i zauważ, że właściwość `checked` pola wyboru i klasa `removed` każdego elementu listy `<li>` są zależne od wartości `checked` elementów tablicy `items`. Poeksperymentuj z polami wyboru, aby zobaczyć, co się dzieje. Czy to nie jest fantastyczne, że za pomocą jedynie dwóch dyrektyw jesteśmy w stanie propagować stan elementów tablicy i zmienić klasę odpowiedniego elementu `<li>`?

## Dodawanie nowych elementów do listy zakupów za pomocą dyrektywy `v-on`

Nasz kod wymaga już tylko drobnej modyfikacji — umożliwienia dodawania elementów do listy. Aby osiągnąć ten cel, w zmiennej `data` umieścimy jeszcze jeden obiekt, który nazwiemy `newItem`. Dodamy też krótką metodę, odpowiedzialną za dodawanie nowego elementu tablicy. Wywołamy ją z poziomu kodu HTML za pomocą dyrektywy `v-on`. Dyrektywę tę zastosujemy w elemencie `input` (polu tekstowym zawierającym nazwę nowego elementu) i w przycisku, po którego kliknięciu element zostanie dodany do listy.

Kod JavaScript będzie miał taką postać:

```

var data = {
  items: [{ text: 'Banany', checked: true },
          { text: 'Jabłka', checked: false }],
  title: 'Moja lista zakupów',
  newItem: ''
};
new Vue({
  el: '#app',
  data: data,
  methods: {
    addItem: function() {
      var text;

      text = this.newItem.trim();
      if (text) {
        this.items.push({
          text: text,
          checked: false
        });
        this.newItem = '';
      }
    }
  }
});

```

```

    }
  }
});

```

Dodaliśmy w obiekcie `data` nową właściwość o nazwie `newItem`. Następnie, przy inicjalizacji `Vue`, dodaliśmy opcję `methods` i umieściliśmy w niej metodę `addItem`. Wewnątrz elementu `methods` dostęp do wszystkich właściwości obiektu `data` można uzyskać za pomocą słowa kluczowego `this`. Dlatego w tej metodzie pobieramy po prostu wartość `this.newItem` i dodajemy ją do tablicy `this.items`. Teraz trzeba powiązać wywołanie metody `addItem` z jakimś działaniem użytkownika. Użyjemy do tego dyrektywy `v-on`. Zastosujemy ją w polu tekstowym (uzależniając wywołanie metody od wystąpienia zdarzenia `keyup.enter`) oraz w przycisku *Dodaj!* (metoda zostanie wywołana po jego kliknięciu).

Wstaw przed listą podany kod HTML:

```

<div class="input-group">
  <input v-model="newItem" v-on:keyup.enter="addItem" placeholder="dodaj
  ↳produkt" type="text" class="form-control">
  <span class="input-group-btn">
    <button v-on:click="addItem" class="btn btn-default"
    ↳type="button">Dodaj!</button>
  </span>
</div>

```

Dyrektywa `v-on` dołącza do elementu detektor zdarzeń. Skrótowo można ją zapisać za pomocą znaku `@`. A więc zapis `v-on:keyup="addItem"` jest równoznaczny z zapisem `@keyup="addItem"`. Więcej informacji na temat dyrektywy `v-on` znajdziesz w oficjalnej dokumentacji pod adresem <http://vuejs.org/api/#v-on>.

Podsumujmy. Całość kodu aplikacji zaprezentowano poniżej.

Kod HTML:

```

<div id="app" class="container">
  <h2>{{ title }}</h2>
  <div class="input-group">
    <input v-model="newItem" @keyup.enter="addItem" placeholder="dodaj
    ↳produkt" type="text" class="form-control">
    <span class="input-group-btn">
      <button @click="addItem" class="btn btn-default"
      ↳type="button">Dodaj!</button>
    </span>
  </div>
</ul>
  <li v-for="item in items" :class="{ 'removed': item.checked }">
    <div class="checkbox">
      <label>

```

```

        <input type="checkbox" v-model="item.checked">{{ item.text }}
      </label>
    </div>
  </li>
</ul>
<div class="footer hidden">
<hr/>
<em>Zmień nazwę listy zakupów</em>
<input v-model="title"/>
</div>
</div>

```

Kod JavaScript:

```

var data = {
  items: [{ text: 'Banany', checked: true },
          { text: 'Jabłka', checked: false }],
  title: 'Moja lista zakupów',
  newItem: ''
};

new Vue({
  el: '#app',
  data: data,
  methods: {
    addItem: function() {
      var text;

      text = this.newItem.trim();
      if (text) {
        this.items.push({
          text: text,
          checked: false
        });
        this.newItem = '';
      }
    }
  }
});

```

Działanie aplikacji możesz sprawdzić w JSFiddle pod adresem <https://jsfiddle.net/chudaol/vxjfkxzk/3/>.

## Korzystanie z Vue.js w istniejącym projekcie

Podejrzewam, że widząc łatwość, z jaką można powiązać właściwości modelu z warstwą prezentacji, już zastanawiasz się, jak wykorzystać Vue we własnym projekcie. Jednak chwilę później przychodzi zwątpienie: przecież będę musiał zainstalować kilka rzeczy, uruchomić polecenie



npm install, załadować brakujące pakiety, dostosować strukturę projektu i dopiero wtedy dodać dyrektywy i zmodyfikować kod.

Tutaj muszę zaprotestować: nie! Żadnych instalacji, żadnych pakietów. Wystarczy pobrać plik *vue.js* i wstawić go w kodzie HTML strony. Tylko tyle! Nie ma konieczności dokonywania zmian w strukturze projektu ani podejmowania jakichkolwiek innych decyzji dotyczących architektury. Nie musisz się nad niczym zastanawiać! Pokażę Ci, jak zastosowaliśmy Vue.js w projekcie EdEra (<https://www.ed-era.com>) do utworzenia prostej sekcji „Sprawdź się” na końcu rozdziału książki GitBooka.

EdEra to ukraiński portal edukacyjny, który postawił sobie za cel unowocześnienie systemu oświaty, wprowadzając weń elementy interaktywnej zabawy. Jestem jednym z założycieli tej młodej firmy i osobą odpowiedzialną za techniczną stronę przedsięwzięcia. W EdEra mamy kilka kursów online zbudowanych na bazie otwartej platformy EdX (<https://open.edx.org/>), a także kilka interaktywnych książek edukacyjnych, które bazują na rewelacyjnym frameworku GitBook (<http://www.gitbook.com>). GitBook to platforma stworzona w technologii Node.js. Pozwala każdej osobie z elementarną znajomością języka Markdown i podstawowych poleceń Gita pisać książki i udostępniać je na serwerach GitBooka. Książki EdEra można znaleźć pod adresem <http://ed-era.com/books> (uwaga, wszystkie są napisane w języku ukraińskim).

Oto co zrobiliśmy w naszych książkach za pomocą Vue.js.

W pewnym momencie postanowiłam umieścić krótki quiz na końcu rozdziału o zaimkach osobowych w podręczniku do angielskiego. W związku z tym dołączyłam plik *vue.js* i dokonałam edycji odpowiedniego pliku *.md*, wstawiając taki oto kod HTML:

```
<div id="pronouns">
  <p><strong>Check yourself :)</strong></p>
  <textarea class="textarea" v-model="text" v-on:keyup="checkText">
    {{ text }}
  </textarea><i v-bind:class="{ 'correct': correct, 'incorrect': !correct }"></i>
</div>
```

Następnie dodałam własny plik JavaScript z podanym kodem:

```
$(document).ready(function() {
  var initialText, correctText;

  initialText = 'Me is sad because he is more clever than I.';
  correctText = 'I am sad because he is more clever than me.';

  new Vue({
    el: '#pronouns',
    data: {
      text: initialText,
      correct: false
    },
    methods: {
      checkText: function() {
        var text;
```

```

    text = this.text.trim();
    this.correct = text === correctText;
  }
});

```

Kod jest dostępny na stronie GitHuba <https://github.com/chudaol/ed-era-book-english>.

Stronę utworzoną w Markdownie z wstawionym fragmentem kodu HTML znajdziesz pod adresem [https://github.com/chudaol/ed-era-book-english/blob/master/2/osobovi\\_zaimenniki.md](https://github.com/chudaol/ed-era-book-english/blob/master/2/osobovi_zaimenniki.md).

A kod JavaScript dostępny jest pod adresem <https://github.com/chudaol/ed-era-book-english/blob/master/custom/js/quiz-vue.js>.

Możesz nawet pobrać całe repozytorium i wypróbować książkę na swoim komputerze za pomocą narzędzia gitbook-cli (<https://github.com/GitbookIO/gitbook/blob/master/docs/setup.md>).

Przyjrzyjmy się temu kodowi. Niektóre jego fragmenty powinny wyglądać znajomo:

- Obiekt data zawiera dwie właściwości:
  - text typu string,
  - correct typu Boolean.
- Metoda checkText pobiera wartość właściwości text, porównuje ją z poprawnym tekstem i nadaje odpowiednią wartość właściwości correct.
- Dyrektywa v-on powoduje wywołanie metody checkText po zwolnieniu klawisza (na skutek zdarzenia keyup).
- Dyrektywa v-bind wiąże klasę correct z właściwością correct.

W moim IDE ten kod wygląda tak jak na rysunku 1.9.

```

osobovi_zaimenniki.md | quiz-vue.js
50 <span class="p1"><b>Об'єктивний вірнінок (<i>Objective pronoun</i></span></b><br>
51 Об'єктивний вірнінок, хоча і здається трохи складнішим за назвний, але існує певні праг.
52 Ми вважало об'єктивний вірнінок.<br>
53 <b><i>. Нісна сніс <i>than/as</i></b><br>
54 She is more beautiful than <b>her</b><br>
55 Tom's as clever as <b>me</b><br>
56 <b>2. У коротких неформальних вірповідях:</b><br>
57 "I'm so tired.<br>
58 ~<b>Me</b> too.<br>
59 </li>
60 </ol>
61
62 <div id="pronouns">
63 <p><strong>Check yourself </strong></p>
64 <textarea class="textare" v-model="text" v-on:keyup="checkText">
65   {{ text }}
66 </textarea><i v-bind:class="{ 'correct': correct, 'incorrect': !correct }"></i>
67
68 </div>
69
70 <div class="centered-table-wrapper">
71 <table class="centered-table">
72 <tr>
73   Якщо раніше існувала думка, що для заміни назви тварин ми вживаємо лише <span class="p1">
74 </td>
75 </tr>
76 </table>

```

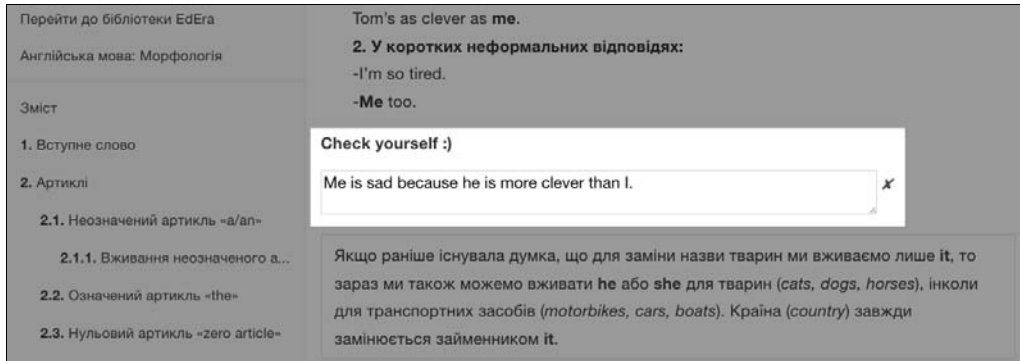
```

1 (document).ready(function() {
2   var initialText, correctText;
3
4   initialText = "He is sad because he is more clever than I.";
5   correctText = "I am sad because he is more clever than me.";
6
7   new Vue({
8     el: "#pronouns",
9     data: {
10      text: initialText,
11      correct: false
12    },
13    methods: {
14      checkText: function () {
15        var text;
16
17        text = this.text.trim();
18
19        this.correct = text === correctText;
20      }
21    }
22  });
23 });
24

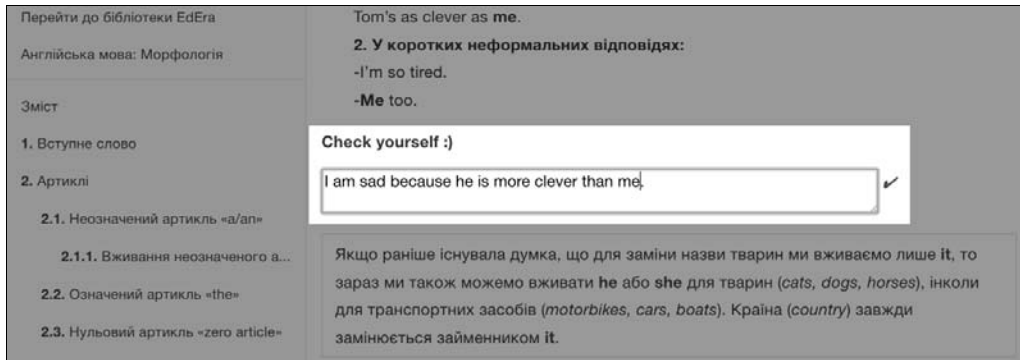
```

Rysunek 1.9. Korzystanie z Vue w projekcie bazującym na Markdownie

Rysunki 1.10 i 1.11 przedstawiają stronę wyświetloną w przeglądarce.



Rysunek 1.10. Vue.js w akcji na stronie GitBooka



Rysunek 1.11. Vue.js w akcji na stronie GitBooka

Wspomnianą stronę możesz zobaczyć pod adresem [http://english.ed-era.com/2/osobovi\\_zaimenniki.html](http://english.ed-era.com/2/osobovi_zaimenniki.html).

Imponująca, prawda? Prosta i do tego reaktywna!

## Vue.js 2.0!

Gdy pisałam tę książkę, zapowiedziano premierę Vue.js 2.0 (<https://vuejs.org/2016/04/27/announcing-2.0/>). Pierwsze recenzje nowej wersji frameworka znajdziesz na stronach:

- <http://www.infoworld.com/article/3063615/javascript/vuejs-lead-our-javascript-framework-is-faster-than-react.html>,
- [https://www.reddit.com/r/vuejs/comments/4gq2r1/announcing\\_vuejs\\_20/](https://www.reddit.com/r/vuejs/comments/4gq2r1/announcing_vuejs_20/).

Druga wersja Vue.js pod pewnymi względami znacząco różni się od pierwszej, zaczynając od sposobu wiązania danych, a kończąc na interfejsie API. Do renderowania używa lekkiej wirtualnej implementacji drzewa DOM, obsługuje renderowanie po stronie serwera, jest szybsza i mniejsza objętościowo.

W chwili, gdy pisałam te słowa, Vue 2.0 był we wczesnej wersji alfa. Ale nie martw się. Wszystkie przykłady omawiane w tej książce bazują na najnowszej stabilnej wersji Vue 2.0 i są w pełni kompatybilne z poprzednią.

## Projekty, w których wykorzystano Vue.js

Prawdopodobnie zastanawiasz się, jakie strony zostały zbudowane w całości lub częściowo na bazie Vue.js. W sieci istnieje ich wiele — od rozwiązań typu open source, przez projekty badawcze, po aplikacje klasy enterprise. Pełną i stale aktualizowaną listę tych projektów można znaleźć pod adresem <https://github.com/vuejs/awesome-vue#projects-using-vuejs>.

Przyjrzyjmy się niektórym z nich.

## Grammarly

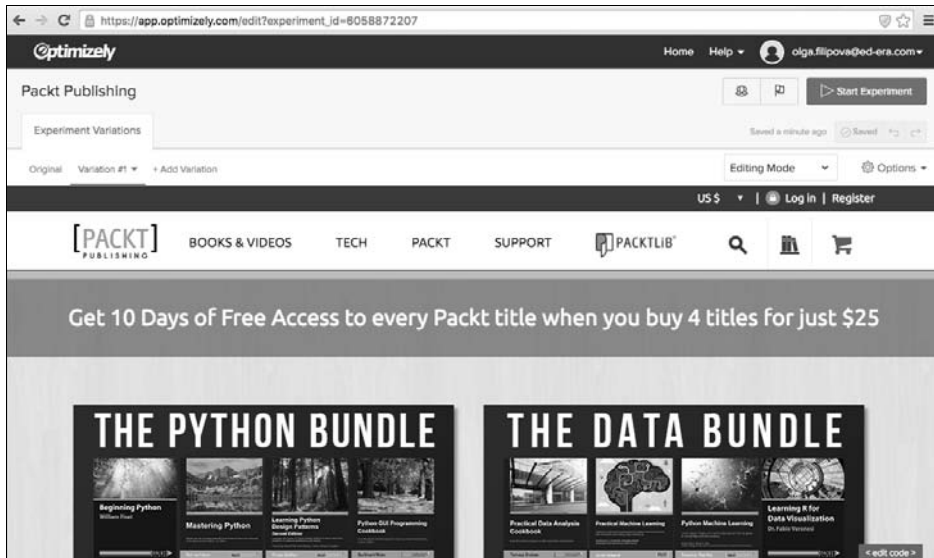
Grammarly (<https://www.grammarly.com/>) to usługa, która pomaga poprawnie pisać w języku angielskim. Obejmuje kilka aplikacji, w tym proste rozszerzenie przeglądarki Chrome, sprawdzające poprawność wprowadzanego tekstu. Jest także internetowy edytor przeznaczony do sprawdzania dłuższych fragmentów tekstu. Ten edytor został zbudowany przy użyciu Vue.js! Na rysunku 1.12 pokazano tekst, który właśnie czytasz (w oryginale — *przyp. tłum.*), edytowany w internetowym edytorze Grammarly:



Rysunek 1.12. Grammarly — projekt bazujący na frameworku Vue.js

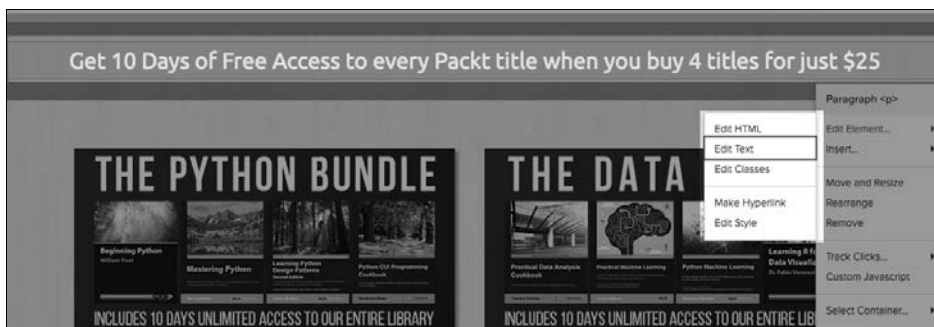
## Optimizely

Optimizely (<https://www.optimizely.com/>) to usługa, która pomaga testować, optymalizować i personalizować witryny. Aby sprawdzić, czy serwis rzeczywiście korzysta z Vue.js, utworzyłam nowy eksperyment, w którym wykorzystałam stronę wydawnictwa Packt (zob. rysunek 1.13).

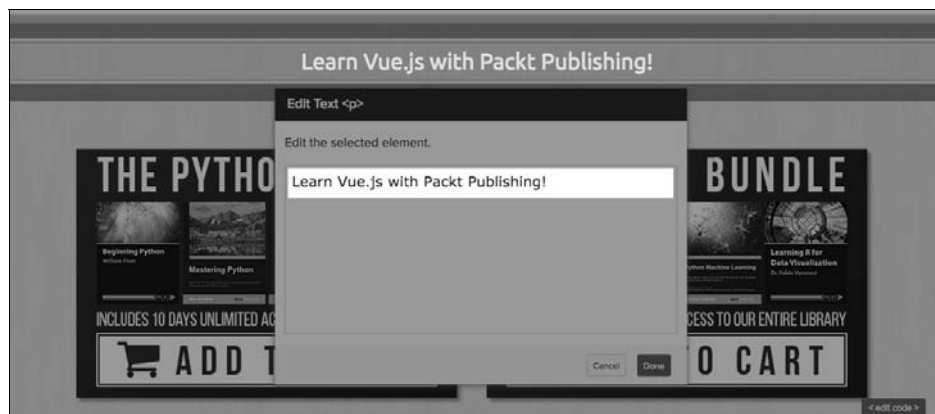


Rysunek 1.13. Optimizely: projekt bazujący na frameworku Vue.js

Po umieszczeniu kursora myszy nad danym elementem strony można otworzyć menu kontekstowe pozwalające wprowadzać na stronie różne zmiany. Najprostsza polega na edycji tekstu (zob. rysunki 1.14 i 1.15).



Rysunek 1.14. Vue.js w trakcie działania w serwisie Optimizely



Rysunek 1.15. Vue.js w trakcie działania w serwisie Optimizely

Na ekranie pojawiło się pole tekstowe. Wprowadzanie tekstu powoduje reaktywną zmianę tytułu. Widzieliśmy to już wcześniej, w naszej aplikacji Vue.

## FilterBlend

FilterBlend (<https://github.com/ilyashubin/FilterBlend>) to narzędzie typu open source, które służy do eksperymentowania z właściwościami CSS `background-blend-mode` oraz `filter`.

Pozwala załadować własne zdjęcia i stosować do nich różne kombinacje filtrów i trybów mieszania.

Jeżeli chcesz wypróbować narzędzie FilterBlend, możesz je zainstalować lokalnie. W tym celu wykonaj następujące kroki:

1. Sklonuj repozytorium:

```
git clone https://github.com/ilyashubin/FilterBlend.git
```

2. Przejdź do katalogu FilterBlend:

```
cd FilterBlend
```

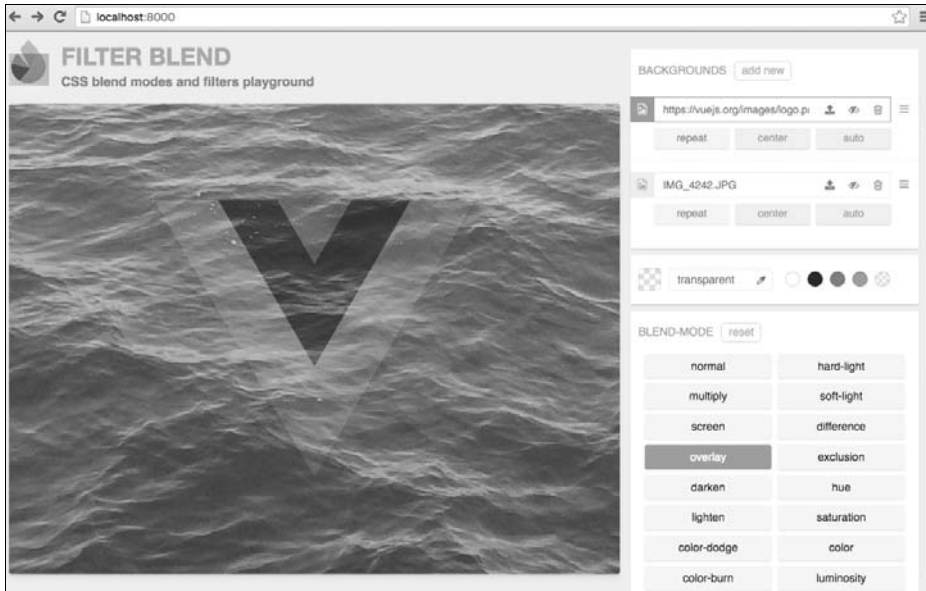
3. Zainstaluj zależności:

```
npm install
```

4. Uruchom projekt:

```
gulp
```

Wpisz w przeglądarce adres `localhost:8000` i spróbuj wygenerować jakiś ciekawy efekt. Zauważ, że każda zmiana w menu po prawej stronie zostanie natychmiast odzwierciedlona na obrazie po lewej. Jest to zasługa Vue.js. Jeżeli nie wierzysz, zajrzyj do kodu na GitHubie. Narzędzie FilterBlend pokazano na rysunku 1.16.



Rysunek 1.16. FilterBlend: projekt zbudowany na bazie Vue.js

## PushSilver

PushSilver (<https://pushsilver.com>) to eleganckie i proste narzędzie dla zapracowanych, wspomagające obsługę faktur. Umożliwia tworzenie, wysyłanie i ponowne przesyłanie faktur do klientów, a także prowadzenie ich ewidencji. Jego autor, niezależny konsultant, był zmęczony koniecznością nieustannego tworzenia faktur przy okazji każdego małego projektu. To przydatne narzędzie także powstało dzięki Vue.js (zob. rysunki 1.17 i 1.18).

## Organizacja książki

Większości książek technicznych nie trzeba czytać od początku do końca. Tak jest i w tym przypadku. Możesz wybrać fragmenty, które interesują Cię najbardziej, i pominąć całą resztę.

Książka została podzielona na następujące rozdziały:

- Rozdział 1.: ponieważ przeczytałeś już prawie cały, zakładam, że nie ma potrzeby omawiania jego zawartości.
- Rozdział 2., „Podstawy — instalacja i użytkowanie”: ma charakter teoretyczny, objaśnia, co się dzieje za kulisami Vue.js, oraz zawiera omówienie budowy frameworka. Jeśli nie przepadasz za teorią i chcesz od razu przejść do programowania, pominiń ten rozdział. Omówimy w nim również instalację i konfigurację Vue.

Rysunek 1.17. PushSilver: aplikacja do zarządzania fakturami bazująca na frameworku Vue

QUANTITY	DESCRIPTION	RATE	TOTAL
5	Drawing a kitten	\$10.00	\$50.00
2	Drawing a dog	\$5.00	\$10.00
		Taxes:	\$24.00
		Stripe Fees:	\$2.82
		Sub Total:	\$66.82
		<b>Total:</b>	<b>\$66.82</b>

Rysunek 1.18. PushSilver: aplikacja do zarządzania fakturami bazująca na frameworku Vue

- W rozdziałach 3. – 8. zbudujemy kilka aplikacji. W każdym z nich skoncentrujemy się na innym obszarze funkcjonalności Vue.js:
- W rozdziale 3., „Komponenty — zasada działania i zastosowanie”, wprowadzimy komponenty Vue. Zdobytą wiedzę na ich temat wykorzystamy przy budowie naszych aplikacji.



- W rozdziale 4., „Reaktywność — wiązanie danych”, poznamy wszystkie mechanizmy wiązania danych dostępne w Vue.js.
- W rozdziale 5., „Vuex — zarządzanie stanem aplikacji”, omówimy system zarządzania stanem Vuex i wyjaśnimy, jak z niego korzystać w aplikacjach.
- W rozdziale 6., „Wtyczki — buduj dom ze swoich własnych cegieł”, utworzymy kilka wtyczek Vue i dowiemy się, jak z nich korzystać w aplikacjach, aby zwiększyć ich funkcjonalność.
- W rozdziale 7., „Testy — sprawdzanie poprawności działania aplikacji”, poznamy niestandardowe dyrektywy Vue.js i sami utworzymy oraz wykorzystamy kilka z nich w naszej aplikacji.
- W rozdziale 8., „Wdrażanie — startujemy w sieci!”, nauczymy się testować i wdrażać aplikacje JavaScript napisane w Vue.js.
- W rozdziale 9., „Co dalej?”, podsumujemy to, czego się nauczyliśmy, i zastanowimy się, w jaki sposób możemy jeszcze ulepszyć nasze aplikacje.

## Zarządzamy czasem!

Teraz wiem już na pewno, że jesteś entuzjastycznie nastawiony do tej książki i że chcesz ją przeczytać jednym tchem. Ale nie o to chodzi. Sztuka zarządzania czasem polega na tym, aby umiejętnie dzielić go między pracę i odpoczynek. Utwórzmy małą aplikację, zegar, który pomoże nam efektywnie zarządzać czasem pracy za pomocą techniki Pomodoro.

Technika **Pomodoro** to technika zarządzania czasem, której nazwa pochodzi od kuchennego minutnika w kształcie pomidora (Pomodoro to po włosku „pomidor”). Istotą tej techniki są częste i krótkie przerwy w pracy co określoną liczbę minut. Więcej informacji o technice Pomodoro znajdziesz na jej oficjalnej stronie: <http://pomodorotechnique.com/>.

Nasze zadanie jest więc łatwe. Wystarczy, że utworzymy bardzo prosty licznik, który będzie odliczał sekundy pozostałe do końca czasu pracy, a następnie uruchomi się ponownie, tym razem odliczając sekundy pozostałe do końca odpoczynku, i tak dalej.

Zajmijmy się tym!

W obiekcie Vue wprowadzimy dwie zmienne z danymi: `minute` oraz `second`. Dzięki pierwszej wyświetlimy na naszej stronie minuty, a dzięki drugiej — sekundy. Metoda `_tick` wraz z upływem czasu będzie zmniejszać wartość zmiennej `second`. Kiedy ta ostatnia będzie równa 0, zmniejszy również wartość zmiennej `minute` o 1. Kiedy obie zmienne osiągną wartość 0, aplikacja przełączy się między czasem pracy a odpoczynkiem:

Kod JavaScript będzie wyglądać następująco:

```

const POMODORO_STATES = {
  WORK: 'work',
  REST: 'rest'
};
const WORKING_TIME_LENGTH_IN_MINUTES = 25;
const RESTING_TIME_LENGTH_IN_MINUTES = 5;

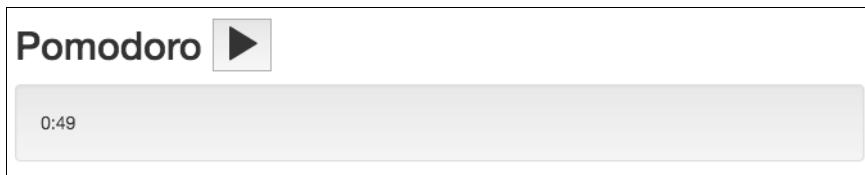
new Vue({
  el: '#app',
  data: {
    minute: WORKING_TIME_LENGTH_IN_MINUTES,
    second: 0,
    pomodoroState: POMODORO_STATES.WORK,
    timestamp: 0
  },
  methods: {
    start: function() {
      this._tick();
      this.interval = setInterval(this._tick, 1000);
    },
    _tick: function() {
      // Jeżeli wartość second jest większa od 0, zmniejsz ją o 1.
      if (this.second !== 0) {
        this.second--;
        return;
      }
      // Jeżeli second ma wartość 0, a minute nie,
      // zmniejsz minute o 1 i nadaj second wartość 59.
      if (this.minute !== 0) {
        this.minute--;
        this.second = 59;
        return;
      }
      // Jeżeli second i minute są równe 0,
      // przełącz między czasem pracy a odpoczynkiem.
      this.pomodoroState = this.pomodoroState ===
        POMODORO_STATES.WORK ? POMODORO_STATES.REST :
        POMODORO_STATES.WORK;
      if (this.pomodoroState === POMODORO_STATES.WORK) {
        this.minute = WORKING_TIME_LENGTH_IN_MINUTES;
      } else {
        this.minute = RESTING_TIME_LENGTH_IN_MINUTES;
      }
    }
  }
});

```

W kodzie HTML zarezerwujemy miejsce na minutę i sekundę oraz utworzymy przycisk startu naszego zegara Pomodoro:

```
<div id="app" class="container">
  <h2>
    <span>Pomodoro</span>
    <button @click="start()">
      <i class="glyphicon glyphicon-play"></i>
    </button>
  </h2>
  <div class="well">
    <div class="pomodoro-timer">
      <span>{{ minute }}</span>:<span>{{ second }}</span>
    </div>
  </div>
</div>
```

Do stylizacji ponownie użyjemy Bootstrapa, co pozwoli nadać zegarowi elegancki wygląd (zob. rysunek 1.19).



Rysunek 1.19. Zegar odliczający czas, zbudowany za pomocą Vue.js

Nasza aplikacja prezentuje się dobrze, ale ma przynajmniej trzy wady:

- Nic nie wiemy o jej aktualnym stanie. Nie wiemy zatem, czy powinniśmy pracować, czy odpoczywać. Dodajmy tytuł, który będzie się zmieniać przy każdej zmianie stanu aplikacji.
- Minuty i sekundy też nie są wyświetlane prawidłowo. Na przykład wartość 24 minuty i 5 sekund powinna być prezentowana na stronie jako 24:05, a nie jako 24:5. Naprawimy to, zastępując zwykłą wartość wartością obliczaną.
- Przycisk startu można klikać wielokrotnie, a po każdym kliknięciu tworzony jest nowy zegar. Spróbuj go kliknąć kilka razy i zobacz, co się dzieje z zegarem. Naprawimy to, dodając przyciski pauzy i stopu. Następnie uzależnimy stan poszczególnych przycisków (aktywny, nieaktywny) od stanu aplikacji.

## Zmiana tytułu przy użyciu właściwości obliczanych

Zacznijmy od rozwiązania pierwszego problemu. Dodamy właściwość obliczaną `title` i użyjemy jej w kodzie HTML.

**Właściwości obliczane** (ang. *computed properties*) to właściwości wewnątrz obiektu `data`, które pozwalają uniknąć osadzania obliczeń bezpośrednio w kodzie szablonu. Więcej informacji na temat właściwości obliczanych znajdziesz w oficjalnej dokumentacji na stronie <http://vuejs.org/guide/computed.html>.

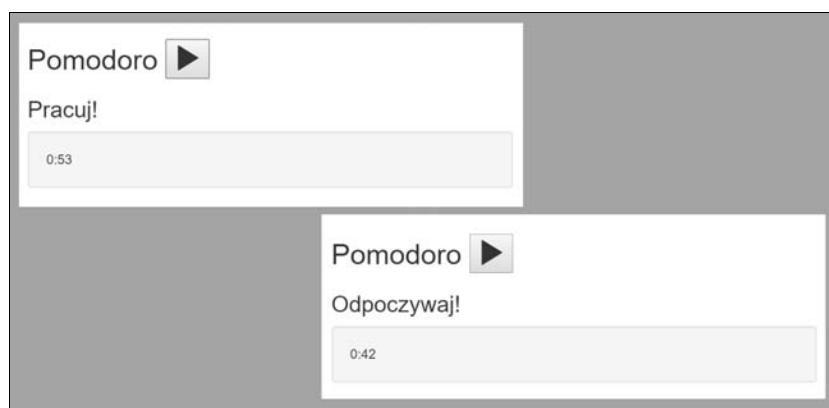
Dodaj w obiekcie opcji `Vue` sekcję `computed` i umieść w niej właściwość `title`:

```
data: {
  //...
},
computed: {
  title: function() {
    return this.pomodoroState ===
      ↪ POMODORO_STATES.WORK ? 'Pracuj!' : 'Odpoczywaj!'
  }
},
methods: {
  //...
```

Teraz wystarczy użyć właściwości w kodzie HTML tak jak zwykłej właściwości modelu danych `Vue`:

```
<h2>
  <span>Pomodoro</span>
  <!--!>
</h2>
<h3>{{ title }}</h3>
<div class="well">
```

Gotowe! Tytuł zmienia się teraz po każdym przełączeniu stanu zegara (zob. rysunek 1.20).



Rysunek 1.20. Automatyczna zmiana tytułu na podstawie stanu zegara

Prawda, że fajne?

## Dopełnienie wartości za pomocą właściwości obliczanych

Podobne rozwiązanie zastosujemy, aby dopełnić liczbę minut i sekund zerem po lewej stronie. Dodamy w sekcji `computed` dwie obliczane właściwości, `min` i `sec`, i zastosujemy prosty algorytm wstawiający 0 po lewej stronie. Oczywiście mogliśmy użyć słynnej biblioteki `left-pad` (<https://github.com/stevemao/left-pad>), ale aby zachować prostotę i nie popsuć całego internetu (<http://www.spidersweb.pl/2016/03/domek-z-kart-left-pad.html>), zastosujemy własne rozwiązanie:

```
computed: {
  title: function() {
    return this.pomodoroState ===
      ↪ POMODORO_STATES.WORK ? 'Pracuj!' : 'Odpoczywaj!'
  },
  min: function() {
    if (this.minute < 10) {
      return '0' + this.minute;
    }

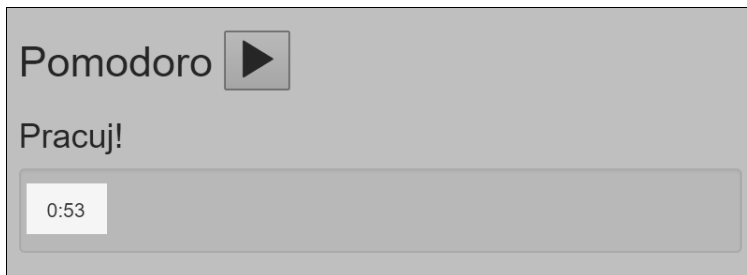
    return this.minute;
  },
  sec: function() {
    if (this.second < 10) {
      return '0' + this.second;
    }

    return this.second;
  }
}
```

Nowe właściwości wstawimy w kodzie HTML w miejsce zmiennych `minute` i `second`:

```
<div class="pomodoro-timer">
  <span>{{ min }}</span>:<span>{{ sec }}</span>
</div>
```

Odśwież stronę i sprawdź, czy liczby wyglądają teraz tak jak na rysunku 1.21.



Rysunek 1.21. Dopełnienie zerem za pomocą właściwości obliczanych

## Kontrola stanu przy użyciu przycisków startu, pauzy i stopu

Aby rozwiązać trzeci problem, wprowadzimy trzy stany aplikacji: started (uruchomiona), paused (wstrzymana) i stopped (zatrzymana) oraz trzy odpowiadające im metody, których zadaniem będzie przełączanie między tymi stanami. Ponieważ mamy już metodę start, odpowiedzialną za uruchomienie aplikacji, wystarczy dodać w niej instrukcję przełączającą stan na wartość started. Utworzymy jeszcze dwie dodatkowe metody, pause i stop, których zadaniem będzie, odpowiednio, wstrzymanie i zatrzymanie zegara, a także przełączenie aplikacji do odpowiedniego stanu:

```
const POMODORO_STATES = {
  WORK: 'work',
  REST: 'rest'
};
const STATES = {
  STARTED: 'started',
  STOPPED: 'stopped',
  PAUSED: 'paused'
};
//<...>
new Vue({
  el: '#app',
  data: {
    state: STATES.STOPPED,
    //<...>
  },
  //<...>
  methods: {
    start: function() {
      this.state = STATES.STARTED;
      this._tick();
      this.interval = setInterval(this._tick, 1000);
    },
    pause: function() {
      this.state = STATES.PAUSED;
      clearInterval(this.interval);
    },
    stop: function() {
      this.state = STATES.STOPPED;
      clearInterval(this.interval);
      this.pomodoroState = POMODORO_STATES.WORK;
      this.minute = WORKING_TIME_LENGTH_IN_MINUTES;
      this.second = 0;
    },
    //<...>
  }
});
```

Dodajmy też w kodzie HTML dwa przyciski wraz z detektorami kliknięcia, które wywołają odpowiednie metody:

```
<button :disabled="state==='started'"
  @click="start()">
  <i class="glyphicon glyphicon-play"></i>
</button>
<button :disabled="state!=='started'"
  @click="pause()">
  <i class="glyphicon glyphicon-pause"></i>
</button>
<button :disabled="state!=='started' && state !== 'paused'"
  @click="stop()">
  <i class="glyphicon glyphicon-stop"></i>
</button>
```

Teraz aplikacja nie tylko ładnie wygląda, ale pozwala również uruchamiać, wstrzymywać i zatrzymać zegar (zob. rysunek 1.22).



Rysunek 1.22. Stan przycisków startu, pauzy i stopu jest zgodny ze stanem aplikacji

Działanie aplikacji na tym etapie budowy możesz sprawdzić w serwisie JSFiddle pod adresem <https://jsfiddle.net/chudaol/b6vmtzq1/1/>.

Po tak intensywnej pracy i przyswojeniu tylu nowych pojęć zasługujesz na ładnego kota! Też uwielbiam koty, więc, proszę, oto losowy zwierzak z niesamowitego generatora <http://thecatapi.com/>:



## Ćwiczenie

Na koniec tego rozdziału chciałabym zaproponować małe ćwiczenie. Zegar Pomodoro, który utworzyliśmy w tym rozdziale, jest fantastyczny, ale jego możliwości są jak na razie dosyć skromne. Jak go uatrakcyjnić? Mógłby na przykład wyświetlać podczas odpoczynku losowe koty z <http://thecatapi.com/>. Potrafisz to zaimplementować? Wierzę, że tak! Proszę jednak nie myl pracy z odpoczynkiem! Twój szef nie będzie zachwycony, kiedy odkryje, że zamiast pracować, oglądasz koty.

Rozwiązanie tego ćwiczenia znajdziesz w dodatku A „Rozwiązania ćwiczeń”.

## Podsumowanie

Bardzo się cieszę, że jesteś w tym miejscu, bo to oznacza, że masz już pewne pojęcie o Vue.js i jeśli ktoś Cię kiedyś zapyta, czy to narzędzie, biblioteka, czy framework, będziesz w stanie udzielić poprawnej odpowiedzi. Wiesz też, jak uruchomić aplikację Vue.js i jak korzystać z Vue w istniejących projektach. Zajrzałeś w głąb kilku poważnych aplikacji, które powstały na bazie tego frameworka, i zacząłeś tworzyć własne! Teraz, idąc na zakupy, nie zapomnij o liście zakupów, którą sam utworzyłeś za pomocą Vue.js! I nie musisz już wykradać minutnika-pomidora z kuchni, żeby odmierzać czas pracy. Możesz do tego użyć własnego cyfrowego zegara Pomodoro. Ale i tak najważniejsze jest to, że dzięki Vue.js możesz teraz wstawiać losowe zdjęcia kotów w swoich aplikacjach JavaScript.

W następnym rozdziale omówimy sposób działania frameworka, a także wykorzystywane przezeń wzorce architektoniczne. Wszystkie nowe pojęcia zilustrujemy odpowiednimi przykładami. Dopiero wtedy będziemy gotowi do pracy z kodem i udoskonalimy nasze aplikacje, przenosząc je w zupełnie inny wymiar.



# Skorowidz

## A

adres IP, 256  
akcje, 169, 230  
aktualizowanie danych na serwerze, 201  
analiza wiązania danych, 28  
Angular, 58  
animacje, 285  
API typu REST, 201  
aplikacja  
    Lista zakupów, 82, 108, 278  
    Pomodoro, 279  
aplikacje  
    dodawanie funkcji, 281  
    obsługa minut i sekund, 185  
    repozytorium GitHuba, 258  
    stan paused, 183  
    stan started, 183  
    stan stopped, 183  
architektura Vuex, 156  
asynchroniczne testy, 237  
automatyczne odświeżanie, 103  
AWS, Amazon Web Services, 257

## B

Bootstrap, 20  
Bower, 71

## C

CDN, Content Distribution Network, 20

chmurowa platforma Heroku, 258  
ciągła integracja, 260  
CRUD, 198  
CSP, Content Security Policy, 72  
CSS, 102

## D

debugowanie, 70, 80  
definiowanie  
    komponentów, 99  
    szablonów, 97  
deklaracja szablonów, 88  
deklaratywne widoki, 21  
detektory zdarzeń, 141  
dodawanie  
    funkcji, 281  
    nowej listy zakupów, 208  
DOM, Document Object Model, 21, 55, 120  
dopelnienie wartości, 47  
drzewo DOM, 21, 55, 120  
dwukierunkowe wiązanie danych, 21, 129  
dyrektywa, 21, 63, 127  
    v-bind, 129  
    v-for, 30, 134  
    v-if, 131  
    v-model, 128  
    v-on, 32, 141  
    v-show, 131  
dziennik zdarzeń, 266

## E

E2E, end-to-end, 249  
efekty przejścia, 285  
    CSS, 115

## F

FilterBlend, 40  
filtry, 126  
format HTML, 88  
framework  
    Angular, 58  
    Protractor, 249  
    React, 56  
    Selenium, 249  
    Weex, 278  
funkcje, 281

## G

getter, 53, 164, 230  
GitHub, 258  
globalny magazyn stanu, 156  
Grammarly, 38  
GUI, 262

## H

Heroku, 258  
    lokalne uruchamianie aplikacji, 272  
    tworzenie aplikacji, 265  
    uruchomienie aplikacji, 267

### I

IDE, 70, 102  
 implementacja listy, 23, 27  
 instalacja, 51, 70  
     magazynu Vuex, 174  
     samodzielna, 70  
     vue-resource, 198  
     za pomocą npm, 74  
     za pomocą vue-cli, 77  
 interfejs  
     graficzny, 262  
     użytkownika, UI, 21  
 interpolacja danych, 120  
 istniejący projekt, 34

### J

jednokierunkowe wiązanie  
 danych, 21  
 jQuery  
     implementacja listy, 23

### K

kaskadowe arkusze  
 stylów, CSS, 21  
 komponent  
     ItemComponent, 109  
     ItemsComponent, 109, 136  
     ShoppingListComponent, 136  
 komponenty, 21, 59  
     definiowanie, 99  
     dwukierunkowe wiązanie,  
         129  
     jednoplikowe, 100, 105, 111  
     komunikacja typu  
         rodzic – dziecko, 149  
     konfiguracja, 109  
     przebudowa aplikacji, 96  
     rejestrowanie, 99  
     styl, 102  
     właściwość data, 89  
     właściwość el, 89  
     zasada działania, 87  
     zasięg, 90, 102  
     zastosowanie, 87  
     zawierające inne  
         komponenty, 92

komunikacja typu  
     rodzic – dziecko, 149

konfiguracja  
     komponentów, 109  
     procesu ciągłej integracji,  
         260  
 kontrola stanu, 48  
 kryteria dobrego testu, 233

### L

Lista zakupów, 23, 27  
     przebudowa aplikacji, 96,  
         105  
     testowanie jednostkowe, 229  
     wdrażanie aplikacji, 270  
     wtyczka vue-resource, 196  
 listy  
     dodawanie elementów, 32  
     implementacja przy użyciu  
         jQuery, 23  
         Vue.js, 27  
     wyświetlanie, 30  
 logotyp, 284  
 lokalne uruchamianie aplikacji,  
     272

### M

magazyn  
     stanu, 156  
     Vuex, 157, 174  
 Markdown, 21  
 menedżer pakietów npm, 21  
 metoda, 198  
     defineProperty, 53  
     Object.defineProperty, 85  
     toUpperCase, 167  
 model DOM, 21, 55, 120  
 modyfikacja modelu, 29  
 mutacja, 168, 230  
     CHANGE\_TITLE, 236  
     DELETE\_SHOPPING\_LI  
         ST, 236  
     POPULATE\_SHOPPING\_  
         LISTS, 236  
 MVVM, 52

### N

nagłówek ze stanem aplikacji,  
     121  
 narzędzia programisty, 28  
 narzędzie  
     Bower, 71  
     Travis, 258  
     vue-cli, 100, 159  
 niestandardowa wtyczka, 212  
 notacja  
     CamelCase, 97  
     kebab-case, 97  
 nowa lista zakupów, 206

### O

obsługa  
     minut i sekund, 185  
     właściwości data, el, 89  
 odświeżanie automatyczne, 103  
 operacje CRUD, 198  
 Optimizely, 39

### P

panel Vue devtools, 80  
 plik  
     AddItemComponent.vue,  
         108  
     App.vue, 137  
     config.js, 190  
 pliki JSON, 197  
 pobieranie list zakupów, 199  
 polecenie  
     ifconfig, 256  
     vue init, 68  
 polityka bezpieczeństwa  
     treści, CSP, 72  
 Pomodoro  
     animacje, 285  
     efekty przejścia, 285  
     logotyp, 284  
     niestandardowa wtyczka,  
         212  
     obsługa minut i sekund, 185  
     przebudowa aplikacji, 111  
     testy E2E, 250  
     testy jednostkowe, 245

tożsamość i styl, 284  
 tworzenie zegara, 188  
 wdrażanie aplikacji, 265  
 zastosowanie wtyczki, 216

preprocesory  
 CSS, 104  
 HTML, 104  
 JavaScriptu, 104

projekt  
 FilterBlend, 40  
 Grammarly, 38  
 Optimizely, 39  
 PushSilver, 41

Protractor, 249

przebudowa aplikacji  
 Lista zakupów, 96, 105  
 Pomodoro, 111

przejście, 115

przełączanie dźwięku, 218

przepływ danych, 56

przetwarzanie tablicy, 134

przycisk  
 do przełączania dźwięku,  
 218  
 pauza, 48, 180  
 start, 48, 180  
 stop, 48, 180

punkt wstrzymania, 55

PushSilver, 41

**R**

React, 56

reaktywne wiązanie efektów  
 przejścia, 115

reaktywność, 21, 119

rejestrowanie komponentów, 99

repozytorium GitHuba, 258

**S**

Selenide, 249

Selenium, 249

serwer  
 CDN, 71  
 HTTP, 197

setter, 53

sieci dystrybucji treści, CDN,  
 20

skrót, 145

specyfika wtyczek, 195

sprawdzanie adresu IP, 256

stan  
 aplikacji, 20, 67, 121, 149  
 magazynu, 164

standard ES2015, 106

stopień pokrycia kodu, 226, 234

styl, 102, 284

symulowanie odpowiedzi  
 serwera, 237

szablony  
 dla wszystkich  
 komponentów, 97  
 w formacie HTML, 88

szkielet aplikacji, 82  
 Lista zakupów, 82  
 Pomodoro, 85

szybkie prototypowanie, 21

**Ś**

środowiska IDE, 70, 102

**T**

tablice, 134

technika Pomodoro, 43

testowanie  
 akcji, 230  
 getterów, 230  
 komponentów, 243  
 mutacji, 230  
 wydajności, 277

testy, 225  
 asynchroniczne, 237  
 E2E, 249, 250  
 jednostkowe, 225, 228, 245  
 kryteria, 233

Travis, 258, 260

tworzenie  
 asynchronicznych testów,  
 237  
 metod, 198  
 nowej listy zakupów, 206  
 prostego serwera, 197  
 szkieletu aplikacji, 82  
 testów E2E, 250  
 testów jednostkowych, 245

zasobów, 198  
 zegara Pomodoro, 188

**U**

usługa Amazona, 257

usuwanie list zakupów, 210

użytkowanie, 70

używanie  
 Heroku lokalnie, 272  
 komponentów, 88

**V**

v-bind, 129

v-for, 134

v-if, 131

v-model, 128

v-on, 141

v-show, 131

Vue 2.0, 277

Vue devtools, 80

Vue.js, 19  
 implementacja listy, 27

Vue.js 2.0, 37

vue-cli, 68, 159

vue-resource, 196  
 instalacja, 198

Vuex, 21, 67, 149, 156  
 instalacja magazynu, 174  
 mutacje, 157  
 stany, 157  
 zastosowanie magazynu,  
 176, 180

**W**

wdrażanie oprogramowania, 255  
 Lista zakupów, 270  
 Pomodoro, 265

Weex, 278

wersja developer, 79

wiązanie  
 atrybutów, 129  
 danych, 28, 119  
 dwukierunkowe, 128  
 efektów przejścia CSS, 115  
 właściwości, 95

widoki deklaratywne, 21

- właściwości obliczane, 46, 47
- właściwość
  - data, 89
  - el, 89
  - timestamp, 191
- wtyczka, 64, 70, 195
  - NoiseGenerator, 213
  - vue-resource, 196, 198
- wtyczki
  - dla środowisk IDE, 102
  - niestandardowe, 212
- wyrażenia, 122
- wyświetlanie
  - listy, 30
  - warunkowe, 131

- wzorzec architektoniczny
  - MVC, 21
  - MVVM, 21, 52

### Z

- zarządzanie
  - czasem Pomodoro, 43
  - stanem aplikacji, 149
- zasięg, 102
  - komponentów, 90
- zasoby, 198
- zastosowanie
  - magazynu Vuex, 176, 180
  - wtyczki, 216

- zaznaczanie elementów listy, 31
- zdarzenia, 141, 149
- zegar Pomodoro, 188, 290
- zmiana tytułu, 45

### Ż

- żądanie utworzenia zasobu, 209

# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

## Vue.js 2: nowe narzędzie, większe możliwości, najlepsze wdrożenia!

Vue.js jest jednym z najnowszych frameworków. Dzięki swojej prostocie i wszechstronności wzbudza ogromne zainteresowanie twórców aplikacji internetowych. Służy do budowy efektywnych, reaktywnych, złożonych i skalowalnych aplikacji WWW, przy czym pozwala na korzystanie z komponentów wielokrotnego użytku. Framework Vue.js powstał jako narzędzie do szybkiego prototypowania, a teraz rozwija się i ewoluuje, dzięki czemu liczba dostępnych funkcji stale rośnie. Vue.js jest na tyle elastyczny i neutralny względem struktury aplikacji, że z całą pewnością można go uznać za framework wspierający budowę kompleksowych aplikacji WWW.

Dzięki tej książce dowiesz się, jak rozpocząć pracę z tym znakomitym narzędziem. Zrozumiesz, czym jest Vue.js, i sprawdzisz, jakie ma możliwości. Będziesz też mieć okazję do przetestowania ich podczas tworzenia kilku ciekawych aplikacji. W trakcie lektury kolejnych rozdziałów przekonasz się, jak wykorzystać potencjał Vue.js do tworzenia niezwykle wydajnych, reaktywnych interfejsów WWW. Nie musisz być przy tym wybitnym deweloperem. Dzięki tej książce poznasz wszystkie etapy budowania interaktywnej aplikacji WWW za pomocą Vue.js: od planowania struktury aż po pełne wdrożenie!

### W tej książce między innymi:

- zarys budowy frameworka, jego instalacja i sposoby wykorzystywania
- system jednoplikowych komponentów oraz wiązanie danych
- architektura aplikacji w Vue.js
- korzystanie z istniejących wtyczek i tworzenie własnych
- testy i wdrażanie aplikacji

**Olga Filipova** — urodziła się w Kijowie. Dorastała w rodzinie fizyków, naukowców i profesorów. Ma duże doświadczenie jako programistka aplikacji WWW, szczególnie w zakresie języka JavaScript. Równocześnie od wielu lat rozwija swoje talenty dydaktyczne w dziedzinie technologii internetowych. Obecnie mieszka w Berlinie, gdzie pracuje jako lider zespołu w firmie Meetrics.

<b>Helion</b> helion.pl 0 801 339900 0 601 339900	Sprawdź nasze szkolenia SZKOLENIA AKADEMIA IT & BUSINESS WWW.SZKOLENIA.HELION.PL	KOD KORZYŚCI Sięgnij po więcej! ▶ ISBN 978-83-283-3874-6 9 788328 338746	
INFORMATYKA W NAJLEPSZYM WYDANIU			Cena: 59,00 zł

**Packt**