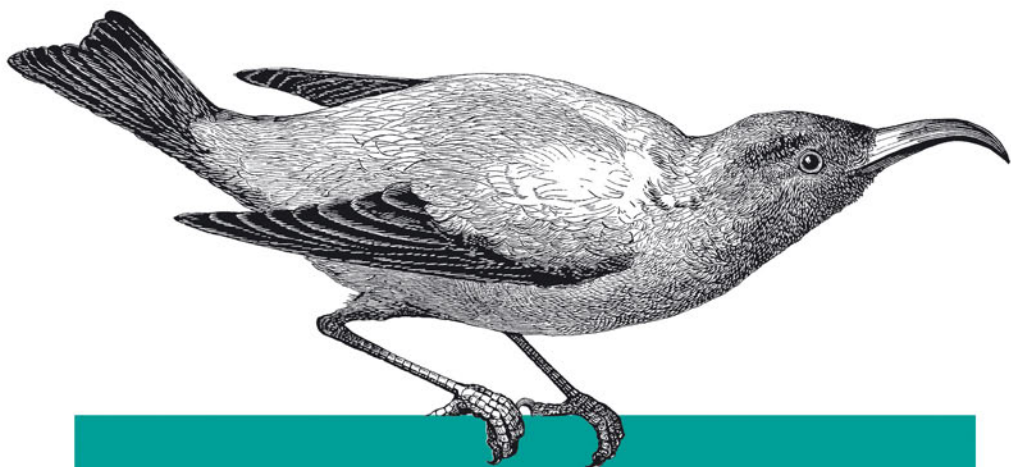


O'REILLY®



React

w działaniu

Tworzenie aplikacji
internetowych

Helion 

Stoyan Stefanov

Tytuł oryginału: React: Up & Running: Building Web Applications

Tłumaczenie: Joanna Zatorska

ISBN: 978-83-283-3301-7

© 2017 Helion SA

Authorized Polish translation of the English edition of: React: Up & Running, ISBN 9781491931820 © 2016 Stoyan Stefanov

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

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

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/reacwd>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

Wstęp	11
1. Witaj, świecie	17
Konfiguracja	17
Witaj, świecie React	18
Co tu się wydarzyło?	20
React.DOM.*	21
Specjalne atrybuty DOM	25
Rozszerzenia przeglądarki React DevTools	26
Co dalej: niestandardowe komponenty	27
2. Życie komponentu	29
Absolutne minimum	29
Właściwości	31
propTypes	32
Domyślne wartości właściwości	35
Stan	36
Komponent obszaru tekstowego ze stanem	37
Uwaga na temat zdarzeń DOM	40
Obsługa zdarzeń w dawnych czasach	40
Obsługa zdarzeń w bibliotece React	43
Props kontra state	44
Props w stanie początkowym: antywzorzec	44

Dostęp do komponentu z zewnątrz	45
Zmiana właściwości w locie	47
Metody cyklu życia	48
Przykład cyklu życia: zaloguj wszystko	49
Przykład cyklu życia: użycie domieszki	52
Przykład cyklu życia: użycie komponentu potomnego	54
Zysk wydajnościowy: zapobieganie aktualizacjom komponentów	56
PureRenderMixin	59
3. Excel — komponent eleganckiej tabeli	63
Przed wszystkim dane	63
Pętla nagłówków tabeli	64
Debugowanie ostrzeżeń konsoli	66
Dodawanie zawartości <td>	68
Jak ulepszyć komponent?	70
Sortowanie	71
Jak ulepszyć komponent?	72
Oznaczenia sortowania w interfejsie użytkownika	73
Edycja danych	74
Komórka edytowalna	75
Komórka z polem tekstowym	77
Zapisywanie	77
Konkluzje i różnice w wirtualnym drzewie DOM	78
Wyszukiwanie	80
Stan i interfejs użytkownika	81
Filtrowanie zawartości	84
Jak ulepszyć wyszukiwanie?	86
Natychmiastowa odpowiedź	86
Jak ulepszyć ponowne odtwarzanie?	88
Alternatywna implementacja?	88
Pobieranie danych tablicy	88

4. JSX	91
Witaj, JSX	91
Transpilacja kodu JSX	92
Babel	93
Po stronie klienta	94
Przekształcenia JSX	95
JavaScript w JSX	97
Białe znaki w JSX	100
Komentarze w JSX	101
Encje HTML	102
Zapobieganie XSS	103
Atrybuty rozszczepiania	104
Atrybuty rozszczepiania przekazywane przez obiekt nadrzędny do potomka	105
Zwracanie wielu węzłów w JSX	106
Różnice między JSX a HTML	108
Brak słów class i for	108
style jest obiektem	109
Znaczniki zamykające	109
Atrybuty w notacji camelCase	109
JSX i formularze	110
Obsługa zdarzenia onChange	110
value a defaultValue	110
Parametr value elementu <textarea>	111
Wartość elementu <select>	113
Komponent Excel w JSX	114
5. Konfiguracja na potrzeby rozwoju aplikacji	115
Aplikacja wzorcowa	116
Pliki i foldery	116
index.html	117
CSS	118
JavaScript	119
Zmodernizowany JavaScript	119

Instalowanie wymagań wstępnych	123
Node.js	123
Browserify	124
Babel	124
React itd.	125
Zabierzmy się do budowania	125
Transpilacja JavaScriptu	125
Pakowanie JavaScriptu	126
Pakowanie CSS	126
Efekty!	126
Wersja dla systemu Windows	127
Budowanie podczas rozwijania aplikacji	127
Wdrożenie	128
Dalsze kroki	129
6. Budowanie aplikacji	131
Whinepad v.0.0.1	132
Konfiguracja	132
Zacznij pisać kod	132
Komponenty	134
Konfiguracja	135
Wykrywanie	135
Komponent <Button>	137
Button.css	138
Button.js	139
Formularze	142
<Suggest>	143
Komponent <Rating>	145
„Fabryka” <FormInput>	149
<Form>	152
<Actions>	155
Okna dialogowe	156
Konfiguracja aplikacji	160
Nowy i ulepszony <Excel>	161
<Whinepad>	170
Czynności końcowe	174

7. Lint, przepływ, testowanie, powtarzanie	175
package.json	175
Konfiguracja narzędzia Babel	176
scripts	176
ESLint	177
Konfiguracja	177
Uruchamianie	178
Wszystkie reguły	179
Flow	180
Konfiguracja	180
Uruchamianie	180
Subskrypcja pod kątem weryfikacji typów	181
Poprawki w komponencie <Button>	182
app.js	183
Więcej właściwości i stanów związanych z weryfikacją typów	185
Typy eksportu i importu	187
Rzutowanie typów	188
Niezmienniki	189
Testowanie	190
Konfiguracja	190
Pierwszy test	192
Pierwszy test Reacta	192
Testowanie komponentu <Button>	194
Testowanie komponentu <Actions>	198
Więcej symulowanych interakcji	201
Testowanie kompletnych interakcji	202
Pokrycie	204
8. Flux	207
Wielka idea	208
Kolejne spojrzenie na Whinepad	208
Magazyn, czyli komponent Store	209
Zdarzenia magazynu	212
Użycie magazynu w <Whinepad>	214
Użycie magazynu w komponencie <Excel>	216

Użycie magazynu w komponencie <Form>	217
Nakreślanie granicy	218
Akcje	219
Akcje CRUD	219
Wyszukiwanie i sortowanie	220
Użycie akcji w module <Whinepad>	222
Użycie akcji w module <Excel>	224
Podsumowanie architektury Flux	226
Niezmienność	227
Niezmienny magazyn danych	228
Manipulowanie danymi niezmiennymi	229
Skorowidz	233

Witaj, świecie

Ruszajmy na wyprawę, podczas której opanujesz tworzenie aplikacji za pomocą technologii React. W tym rozdziale nauczysz się konfigurować Reacta, a także napiszesz swoją pierwszą aplikacją internetową „Witaj, świecie”.

Konfiguracja

Najpierw musisz uzyskać kopię biblioteki React. Na szczęście jest to bardzo proste.

Przejdź do strony <http://reactjs.com> (która powinna przekierować Cię na oficjalną stronę GitHub, znajdującą się pod adresem <http://facebook.github.io/react/>) i kliknij przycisk *Download*. Na kolejnej stronie kliknij przycisk *Download Starter Kit*, aby pobrać plik ZIP zawierający bibliotekę. Rozpakuj pobrany plik i skopiuj znajdujący się tam folder do miejsca, które łatwo odszukać.

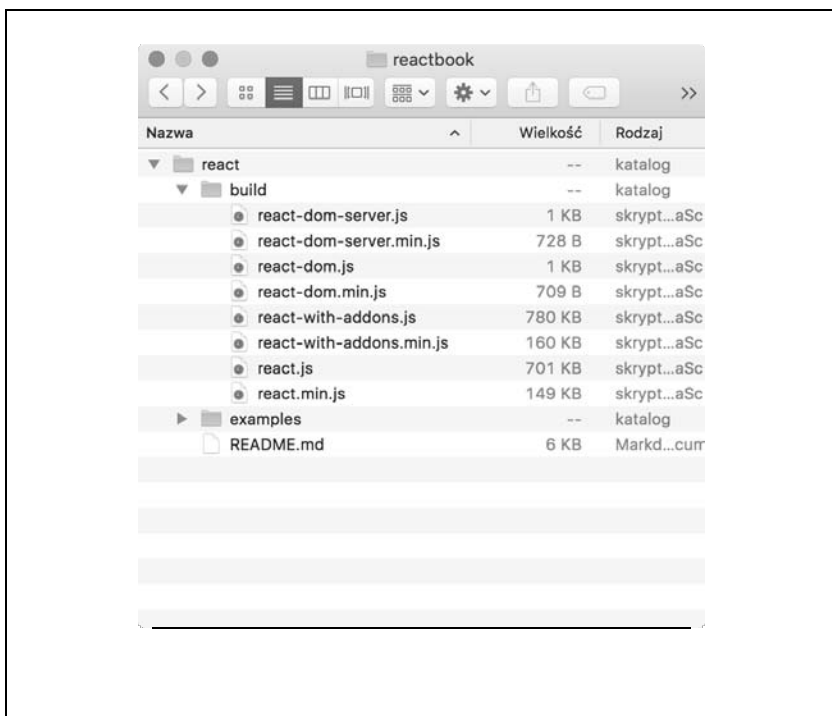
Możesz to zrobić na przykład następująco:

```
mkdir ~/reactbook
mv ~/Downloads/react-0.14.7/ ~/reactbook/react
```

Po wykonaniu tych czynności zawartość katalogu roboczego (o nazwie *reactbook*) powinna wyglądać jak na rysunku 1.1.

Do rozpoczęcia pracy wystarczy plik *~/reactbook/react/build/react.js*. Kolejne pliki poznasz w trakcie dalszej pracy.

Zwróć uwagę na to, że React nie narzuca żadnej struktury folderów. Wspomniany plik możesz umieścić w folderze o dowolnej nazwie, a nazwę samego pliku, *react.js*, możesz dowolnie zmienić.



Rysunek 1.1. Zawartość katalogu z biblioteką React

Witaj, świecie React

Zacznij od utworzenia prostej strony w folderze roboczym (`~/reactbook/01.01.hello.html`):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Witaj, React</title>
    <meta charset="utf-8">
  </head>
  <body>
    <div id="app">
      <!-- tutaj będzie renderowana moja aplikacja -->
    </div>
    <script src="react/build/react.js"></script>
```

```
<script src="react/build/react-dom.js"></script>
<script>
  // kod mojej aplikacji
</script>
</body>
</html>
```



Kody wszystkich przykładów znajdujących się w książce znajdziesz na stronie <ftp://ftp.helion.pl/przyklady/reactwd.zip>.

W tym pliku warto zwrócić uwagę na następujące aspekty:

- Dołączamy bibliotekę React i jej rozszerzenie DOM (korzystając ze znaczników `<script src=>`).
- Definiujemy miejsce, w którym aplikacja powinna być widoczna na stronie (`<div id="app">`).



W aplikacji tworzonej za pomocą Reacta można zawsze korzystać zarówno z kodu HTML, jak i z innych bibliotek JavaScriptu. Na jednej stronie może się znajdować kilka aplikacji napisanych z użyciem Reacta. Wystarczy umieścić odpowiedni kod w strukturze DOM i rozkazać mu, aby „czynił swoją magię”.

Dodaj teraz kod wyświetlający powitanie. W tym celu zmodyfikuj plik `01.01.hello.html` i zastąp wiersz `// kod mojej aplikacji` następującym kodem:

```
ReactDOM.render(
  React.DOM.h1(null, "Witaj, świecie!"),
  document.getElementById("app")
);
```

Wczytaj plik `01.01.hello.html` w przeglądarce i sprawdź działanie swojej nowej aplikacji (rysunek 1.2).

Gratulacje! Utworzyłeś swoją pierwszą aplikację za pomocą Reacta!

Na rysunku 1.2 widoczny jest też podgląd *wygenerowanego* kodu w narzędziu Chrome Developer Tools. Możesz zauważyć, że tymczasowa zawartość elementu `<div id="app">` została zastąpiona zawartością wygenerowaną przez aplikację utworzoną za pomocą Reacta.

Witaj, świecie!



```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <div id="app">
      <h1 data-reactroot="Witaj, świecie!">Witaj, świecie!</h1>
    </div>
    <script src="react/build/react.js"></script>
    <script src="react/build/react-dom.js"></script>
    <script>
      ReactDOM.render(
        React.DOM.h1(null, "Witaj, świecie!"),
        document.getElementById("app")
      );
    </script>
  </body>
</html>
```

Rysunek 1.2. Działająca aplikacja Witaj, świecie

Co tu się wydarzyło?

Kod odpowiadający za działanie Twojej pierwszej aplikacji zawiera kilka ciekawych elementów.

Przede wszystkim należy zauważyć użycie obiektu React. Za jego pośrednictwem dostępne są wszystkie możliwe do wykorzystania metody API. API jest w rzeczywistości celowo uproszczone i zawiera niewielką liczbę metod, co ułatwia ich zapamiętanie.

Widoczny jest też obiekt ReactDOM. W obiekcie tym zdefiniowano tylko kilka metod, z których najbardziej użyteczna jest metoda render(). Metody te były wcześniej dostępne z poziomu obiektu React, lecz począwszy od wersji 0.14, zostały od niego odseparowane. W ten sposób podkreślono, że samo renderowanie aplikacji jest zupełnie inną kwestią. Można przecież utworzyć aplikację za pomocą Reacta przeznaczoną do renderowania w różnych środowiskach — na przykład w postaci kodu HTML (hierarchii DOM w przeglądarce), w elemencie canvas lub w natywnych aplikacjach Android lub iOS.

Kolejnym aspektem jest koncepcja *komponentów*. Interfejs użytkownika tworzy się za pomocą komponentów łączonych ze sobą w sposób, który najlepiej spełnia nasze potrzeby. W swoich aplikacjach będziesz w przyszłości tworzyć własne komponenty, lecz na początku możesz skorzystać z gotowych komponentów Reacta, wykorzystujących elementy DOM języka HTML. Masz do nich dostęp poprzez obiekt `React.DOM`. W pierwszym przykładzie używasz komponentu `h1`. Odpowiada on elementowi HTML `<h1>` i jest dostępny poprzez wywołanie funkcji `React.DOM.h1()`.

Wreszcie można zauważyć dostęp do elementu DOM za pomocą starej dobrej składni `document.getElementById("app")`. W ten sposób informujemy Reacta, w którym miejscu strony powinna zostać wyświetlona nasza aplikacja. Jest to pomost między znanym już sposobem przetwarzania elementów DOM a nowym sposobem stosowanym w świecie Reacta.

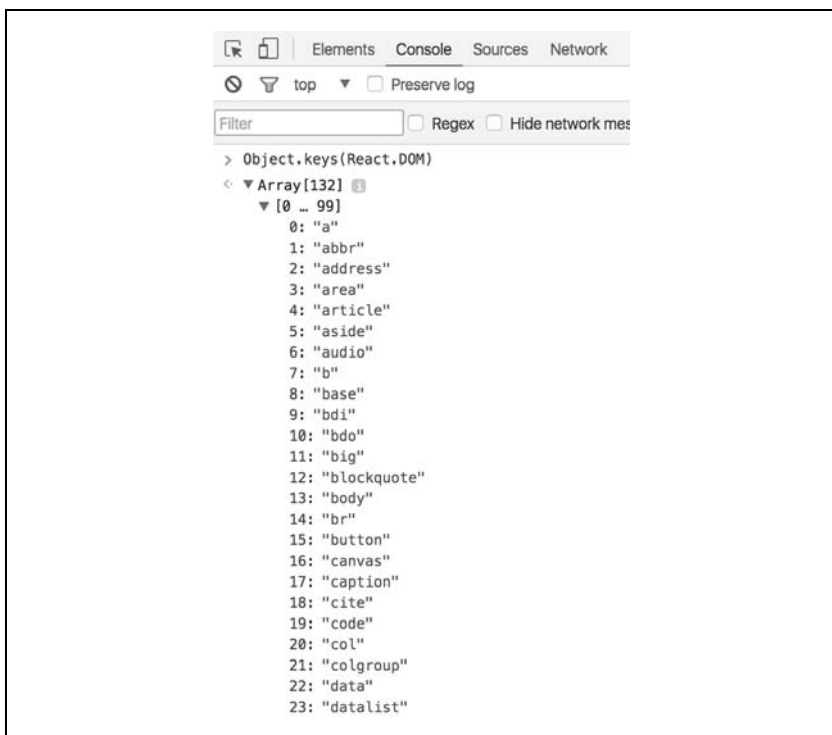


Po przekroczeniu mostu łączącego DOM ze światem Reacta nie musisz się już przejmować przetwarzaniem elementów DOM, ponieważ React przetwarza komponenty w elementy stosowane w docelowej platformie (w przeglądarce DOM, w elemencie `canvas`, w aplikacji natywnej). Nie musisz się przejmować strukturą DOM, ale oczywiście możesz. W bibliotece React znajdują się furtki, które w razie potrzeby umożliwiają powrót do świata DOM.

Wiesz już, za co odpowiada każdy wiersz, i możesz spojrzeć na kod całościowo. Oto co się stało: wyrenderowałeś komponent Reacta w wybranym miejscu drzewa DOM. Zawsze renderowany jest jeden komponent wysokiego poziomu, który może mieć dowolną liczbę komponentów potomnych (które z kolei mogą mieć swoich potomków itd.). Nawet w tym prostym przykładzie komponent `h1` ma potomka — napis „Witaj, świecie!”.

React.DOM.*

Jak już wiesz, poprzez obiekt `React.DOM` możesz skorzystać z wielu elementów HTML reprezentowanych przez komponenty Reacta (rysunek 1.3 przedstawia sposób, który umożliwia wyświetlenie całej listy w konsoli przeglądarki). Przyjrzyjmy się teraz API tego obiektu.



Rysunek 1.3. Lista właściwości React.DOM



Zwróć uwagę na różnice między obiektami `React.DOM` a `ReactDOM`. Pierwszy z nich jest kolekcją gotowych elementów HTML, a drugi służy do renderowania aplikacji w przeglądarce (przypomnij sobie metodę `ReactDOM.render()`).

Przyjrzyjmy się parametrom, jakie przyjmują wszystkie metody `React.DOM.*`. Pamiętaj, że aplikacja „Witaj, świecie!” wygląda następująco:

```
ReactDOM.render(  
  React.DOM.h1(null, "Witaj, świecie!"),  
  document.getElementById("app")  
);
```

Pierwszym parametrem przyjmowanym przez metodę `h1()` (w tym przypadku ma on wartość `null`) jest obiekt określający wszelkie właściwości (pomyśl o atrybutach DOM), które można przekazać do komponentu. Możesz na przykład napisać taki kod:

```

React.DOM.h1(
  {
    id: "my-heading",
  },
  "Witaj, świecie!"
),

```

Na rysunku 1.4 widoczny jest kod HTML wygenerowany w powyższym przykładzie.



Rysunek 1.4. Kod HTML wygenerowany przez wywołanie metody obiektu `React.DOM`

Drugi parametr (w tym przypadku "Witaj, świecie!") definiuje element potomny komponentu. Najprostszym przypadkiem jest zwykły potomek tekstowy (węzeł `Text` w terminologii DOM), widoczny w poprzednim przykładzie. Można jednak zagnieździć dowolną liczbę elementów potomnych, które należy przekazać jako dodatkowe parametry funkcji. Na przykład:

```

React.DOM.h1(
  {id: "my-heading"},
  React.DOM.span(null, "Witaj,"),
  " świecie!"
),

```

Kolejny przykład zawiera zagnieżdżone komponenty (uzyskany efekt jest przedstawiony na rysunku 1.5):

```
React.DOM.h1(  
  {id: "my-heading"},  
  React.DOM.span(null,  
    React.DOM.em(null, "Wita"),  
    "j,"  
  ),  
  " świecie!"  
),  
),
```



Rysunek 1.5. Kod HTML wygenerowany poprzez zagnieżdżone wywołania React.DOM



Jak widać, zagnieżdżanie komponentów może szybko doprowadzić do kodu zawierającego wiele wywołań funkcji i nawiasów, które należy uważnie śledzić. Problem ten można rozwiązać, korzystając ze *składni JSX*. JSX jest techniką wartą omówienia (zajmiemy się nią w rozdziale 4.), lecz na razie pozostawmy przy zwykłej składni JavaScriptu, ponieważ JSX wzbudza pewne kontrowersje. Otóż użytkownicy często na początku reagują na tę składnię dość alergicznie (hmm, XML w skrypcie napisanym w JavaScriptcie!), ale później okazuje się ona niezastąpiona. Oto poprzedni przykład, tym razem napisany z użyciem składni JSX:

```
ReactDOM.render(  
  <h1 id="my-heading">  
    <span><em>Wita</em></span> świecie!  
  </h1>,  
  document.getElementById("app")  
)
```

Specjalne atrybuty DOM

Musisz poznać kilka specjalnych atrybutów DOM, a mianowicie `class`, `for` i `style`.

Atrybutów `class` i `for` nie można używać w JavaScriptcie, ponieważ są to słowa zarezerwowane. Zamiast nich używaj elementów `className` i `htmlFor`:

```
// Antyprzykład  
// to nie działa  
ReactDOM.h1(  
  {  
    class: "pretty",  
    for: "me",  
  },  
  "Witaj, świecie!"  
);  
// Poprawny przykład  
// to działa  
ReactDOM.h1(  
  {  
    className: "pretty",  
    htmlFor: "me",  
  },  
  "Witaj, świecie!"  
);
```

Przejdźmy teraz do atrybutu `style`, określającego styl. Otóż nie można do niego przekazać zwykłego łańcucha tekstowego, jak w kodzie HTML. W tym przypadku musi to być obiekt JavaScriptu. Zawsze warto unikać stosowania łańcuchów tekstowych, aby zmniejszyć ryzyko ataków typu cross-site scripting (XXS). Z tego względu wspomniana zmiana jest pożądana.

```
// Antyprzykład
// to nie działa
ReactDOM.h1(
  {
    style: "background: black; color: white; font-family: Verdana",
  },
  "Witaj, świecie!"
);

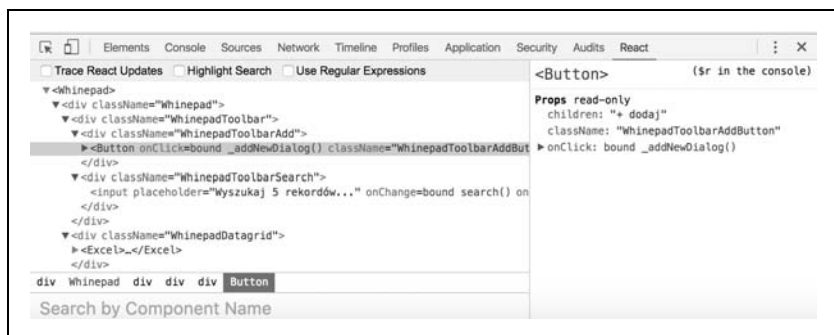
// Poprawny przykład
// to działa
ReactDOM.h1(
  {
    style: {
      background: "black",
      color: "white",
      fontFamily: "Verdana",
    }
  },
  "Witaj, świecie!"
);
```

Zauważ też, że przy określaniu właściwości CSS musisz użyć nazw API JavaScriptu. Innymi słowy: użyj właściwości `fontFamily` zamiast `font-family`.

Rozszerzenia przeglądarki React DevTools

Jeśli podczas przeglądania niektórych przykładów z tego rozdziału otworzyłeś konsolę przeglądarki, zapewne ujrzałeś informację o treści: „Download the React DevTools and use an HTTP server (instead of a file: URL) for a better development experience: <https://fb.me/react-devtools>”. Na stronie o podanym adresie dostępne są łącza umożliwiające zainstalowanie rozszerzenia przeglądarki, które może się okazać przydatne w debugowaniu aplikacji utworzonej z użyciem Reacta (rysunek 1.6).

Na początku rozszerzenie to może się wydawać przytłaczające, lecz wrażenie to minie, zanim dotrzesz do rozdziału 4.



Rysunek 1.6. Rozszerzenie React DevTools

Co dalej: niestandardowe komponenty

Na tym zakończyliśmy pracę nad prostą aplikacją „Witaj, świecie”. Wiesz już jak:

- zainstalować, skonfigurować i stosować bibliotekę React (jest to kwestia umieszczenia dwóch znaczników `<script>`);
- wyrenderować komponent Reacta w wybranej lokalizacji DOM (na przykład `ReactDOM.render(reactCo, domGdzie)`);
- używać wbudowanych komponentów, które wykorzystują zwykłe elementy DOM (na przykład `ReactDOM.div(atrybuty, dzieci)`).

Jednak prawdziwa potęga Reacta objawia się podczas używania niestandardowych komponentów budujących (i uaktualniających!) interfejs aplikacji. Z kolejnego rozdziału dowiesz się, jak je wykorzystać.

A

- addytywne wyszukiwanie, 86
- akcje, 155, 208, 219, 222
 - CRUD, 219
- aktualizacje komponentów, 56
- antywzorzec, 44
- API
 - tworzenie komponentu, 29
- aplikacja Whinepad, 132
- architektura Flux, 207, 226
- argument props, 182
- atak XSS, 103
- atrapa, 195
 - funkcji, 199
- atrybut
 - class, 108
 - for, 108
 - ref, 145
 - style, 109
- atrybuty
 - DOM, 25
 - rozszczepiania, 104, 105

B

- Babel, 93, 124, 176
- bezystanowy komponent, 38
 - funkcyjny, 141
- białe znaki, 100

- biblioteka

- Flow, 180
 - immutable, 228
- Browserify, 124
- budowanie, 127
 - aplikacji, 131

C

- CRUD, 131, 219
- CSS, 118
- cykl życia, 48
 - użycie domieszki, 52
 - użycie komponentu potomnego, 54
 - zaloguj wszystko, 49

D

- dane, 63
 - niezmienne, 229
 - tablicy, 88
- DDD, discovery-driven development, 138
- debugowanie, 66
- dodatek PureRenderMixin, 59, 61
- dodawanie zawartości <td>, 68
- DOM, 19, 78
 - specjalne atrybuty, 25
 - zdarzenia, 40
- domieszka, 53

domyślne wartości właściwości, 35
dostęp
 do komponentu, 46
 do komponentu z zewnątrz, 45
 do obiektów, 44
drzewo DOM, 78
dyspozytor, 226

E

ECMAScript, 120
edycja danych, 74, 168
eksport, 187
element
 <Form>, 152
 <select>, 113
 <Suggest>, 143
 <textarea>, 111
elementy wejściowe formularza, 150
encje HTML, 102
ESLint
 konfiguracja, 177
 uruchamianie, 178
Excel, 63

F

fabryka, 149, 152
filtrowanie zawartości, 84
Flow
 app.js, 183
 konfiguracja, 180
 rzutowanie typów, 188
 uruchamianie, 180
Flux, 207, 226
foldery, 116
formularz
 elementy wejściowe, 150
formularze, 110, 142
funkcja
 find, 200
 scry, 200

H

HTML, 108

I

import, 187
instalowanie wymagań wstępnych, 123
interakcje, 201
interfejs użytkownika, 81

J

JavaScript, 119
 moduły, 120
JSX, 25, 91, 108
 białe znaki, 100
 encje HTML, 102
 komentarze, 101
 komponent Excel, 114
 przekształcenia, 95
 transpilacja kodu, 92
 zwracanie węzłów, 106

K

klasa Whinepad, 170
klasy, 121
klienty, 94
komórka
 edytowalna, 75
 z polem tekstowym, 77
komponent, 21, 134
 Excel, 63
 odmontowywanie, 57
 renderowanie, 58
 uaktualnianie, 51
komponent
 <Actions>, 198
 <Button>, 137, 138, 182, 194
 <Excel>, 114, 134, 161, 216
 <Form>, 217

- <FormInput>, 149, 187
- <Rating>, 145
- <Whinepad>, 214
- Store, 209
- TextAreaCounter, 39, 49

komponenty

- bezstanowe, 38
- funkcyjne, 141
- niestandardowe, 27, 29
- potomne, 54
- tabeli, 63
- ze stanem, 37

konfiguracja, 17

- aplikacji, 160
- narzędzia Babel, 176
- procesu budowania, 115
- skryptów, 176

L

Lint, 175

M

magazyn, 208, 214–217, 228

manipulowanie danymi niezmiennymi, 229

metoda, 121

- componentDidMount(), 48
- componentDidUpdate(), 48
- componentWillMount(), 48
- componentWillUnmount(), 49
- componentWillUpdate(), 48, 50
- getInitialState(), 44
- render(), 20, 44
- shouldComponentUpdate(), 49, 59
- splice(), 231

metody cyklu życia, 48

moduł

- <Excel>, 224
- <Whinepad>, 222

moduły

ECMAScriptu, 120

JavaScriptu, 120

montowanie komponentu, 50

MVP, minimum viable product, 135

N

nagłówek tabeli, 64

narzędzie

- Babel, 124, 176
- Browserify, 124
- ESLint, 175, 177
- Flow, 175, 180
- Flux, 207
- Jest, 175, 190
- npm, 123, 175

wykrywania komponentów, 135

niestandardowe komponenty, 29

właściwości, 31

niezmienniki, 189

niezmiennność, 227

niezmienny magazyn danych, 228

Node.js, 123

notacja camelCase, 109

NPM, Node Package Manager, 175

O

obiekt

- React.DOM, 21
- this.props, 44
- this.state, 44

obsługa

- zdarzenia onChange, 110
- zdarzeń, 40, 43

obszar tekstowy, 37

odmontowywanie komponentu, 57

okna dialogowe, 156

opakowywanie komponentów, 198

operator rozszczepiania, 104

oznaczenia sortowania, 73

P

- pakiet classnames, 139
- pakowanie
 - CSS, 126
 - JavaScriptu, 126
 - projektu, 175
- parametr value, 111
- pierwsza
 - aplikacja, 18
 - specyfikacja, 196
- pierwszy test Reacta, 192
- plik, 116
 - app.css, 118
 - app.js, 183
 - Button.css, 138
 - Button.js, 139
 - index.html, 117
 - package.json, 175
- pobieranie danych tablicy, 88
- podgląd danych, 169
- pokrycie kodu testami, 204
- polecenia budowania, 127
- powtarzanie, 175
- programowanie sterowane wykrywaniem, DDD, 138
- przekształcenia JSX, 95
- przepływ, 175
 - danych, 208
 - jednokierunkowy, 227
- przycisk wyszukiwania, 80
- przypisanie destrukuryzacyjne, 141
- PureRenderMixin, 59

R

- raport pokrycia kodu testami, 205
- React DevTools, 26
- React.DOM, 21
 - właściwości, 22
- reguły biblioteki React, 179

- renderowanie
 - aplikacji, 20
 - komponentów, 58
 - tabeli, 69
- rozszcepianie, 104
- rozszerzenia
 - przeglądarki, 26
 - DOM, 19
- rozwijanie aplikacji, 127
- rzutowanie typów, 188

S

- składnia JSX, 25
- sortowanie, 71, 220
- SPA, single-page application, 116
- specyfikacja, 196
- stan, 36, 81
- symulowane interakcje, 201
- szablon aplikacji, 116

T

- tabela, 63
- testowanie, 175, 190
 - kompletnych interakcji, 202
 - komponentu <Actions>, 198
 - komponentu <Button>, 194
- transpilacja
 - JavaScriptu, 125
 - kodu JSX, 92
- tworzenie komponentu, 29
- typy eksportu i importu, 187

U

- uaktualnianie komponentu, 51
- użycie
 - akcji, 222, 224
 - domieszki, 52
 - komponentu potomnego, 54

- magazynu, 214, 216, 217
- niestandardowego komponentu, 29
- właściwości komponentu, 32

W

- wartości domyślne właściwości, 35
- wdrożenie, 128
- weryfikacja typów, 181, 185
- Whinepad, 132, 170, 208
 - użycie akcji, 222
 - użycie magazynu, 214
- widok, 208
- widżet oceny, 146
- właściwości
 - niestandardowych komponentów, 31
 - React.DOM, 22
- właściwość
 - props, 185
 - propTypes, 32, 141
 - value, 110

- wydajność, 56
- wykrywanie komponentów, 135
- wyszukiwanie, 80, 86, 220

Z

- zapisywanie, 77
- zapobieganie aktualizacjom komponentów
 - w, 56
- zdarzenia, 43
 - DOM, 40
 - magazynu, 212
 - syntetyczne, 40, 43
- zdarzenie onChange, 110
- zmiana właściwości w locie, 47
- znaczniki zamykające, 109
- zwracanie węzłów, 106

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**

React – programowanie może być przyjemne!

Interfejs użytkownika musi działać w przewidywalny sposób na różnych przeglądarkach i urządzeniach. Powinien zapewniać adekwatną obsługę wprowadzanych danych oraz umożliwiać płynne i niezawodne komunikowanie się aplikacji z użytkownikiem. Dziś programiści mogą skupić się na samym działaniu aplikacji, gdyż interfejs użytkownika można szybko, łatwo i przyjemnie zbudować za pomocą biblioteki React.

Poznasz ją dzięki temu przewodnikowi i zaczniesz wykorzystywać w tworzeniu aplikacji internetowych. Dowiesz się, jak można jednorazowo zdefiniować elastyczny interfejs, nauczysz się tworzyć rozbudowane aplikacje z wykorzystaniem małych, łatwych w zarządzaniu komponentów. Zapoznasz się również z JSX – technologią świetnie uzupełniającą React. Nauczysz się też korzystać z dodatkowych narzędzi, takich jak Browserify, Jest, ESLint, Flow, Flux czy Immutable.js. Wzbogać swój warsztat programisty aplikacji WWW!

Stoyan Stefanov – jest inżynierem w Facebooku. Wcześniej pracował dla Yahoo!, gdzie stworzył smush.it – narzędzie online służące do optymalizacji obrazów. Brał też udział w tworzeniu narzędzia o nazwie YSlow 2.0, przeznaczonego do pomiaru wydajności aplikacji. Jest autorem i współautorem wielu książek dotyczących programowania i tworzenia aplikacji internetowych. Przewodzi bloga dostępnego pod adresem <http://phpied.com>. Często zabiera głos na prestiżowych konferencjach, takich jak Velocity, JSConf, Fronteers i wielu innych.

Niektóre zagadnienia omówione w książce:

- przygotowywanie biblioteki React do pracy
- komponenty, ich właściwości, stan i cykl życia
- wykorzystywanie komponentów do budowy UI
- korzystanie z narzędzi pomocniczych przy budowie aplikacji
- diagnostyka i testowanie kodu aplikacji

Helion

księgarnia Internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Informatyka w najlepszym wydaniu

Helion SA
ul. Kościuski 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Sprawdź najnowsze promocje:
● <http://helion.pl/promocje>
Książki najchętniej czytane:
● <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
● <http://helion.pl/nowosci>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-283-3301-7



9 788328 333017

cena: 39,90 zł