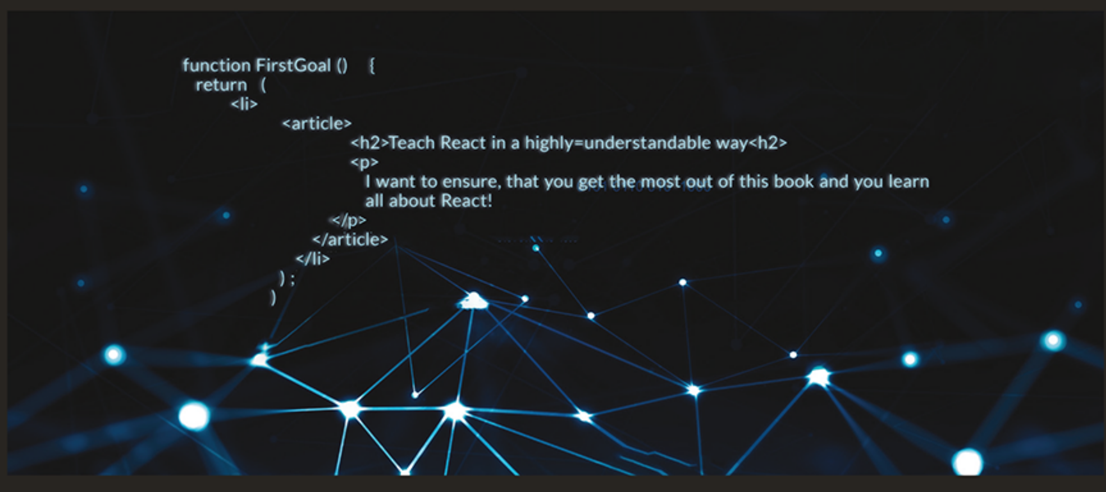


React
kluczowe koncepcje

Przewodnik po najważniejszych mechanizmach biblioteki React

MAXIMILIAN SCHWARZMÜLLER

Tytuł oryginału: React Key Concepts: Consolidate your knowledge of React’s core features

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-8322-884-6

Copyright © Packt Publishing 2022. First published in the English language
under the title ‘React Key Concepts – (9781803234502)’

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 jakiejkolwiek 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 wydawca 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 wydawca nie ponoszą również żadnej odpowiedzialności
za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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

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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Wprowadzenie **12**

Rozdział 1.
React — co, jak i dlaczego? **19**
- Wprowadzenie 19
- Czym jest React? 20
- Problem z czystym JavaScriptem 21
- React i kod deklaratywny 25
  - W jaki sposób React operuje modelem DOM? 29
- Wprowadzenie do aplikacji jednostronicowych 30
  - Tworzenie projektu Reacta 31
- Podsumowanie i najważniejsze informacje do zapamiętania 33
  - Co dalej? 34
  - Sprawdź swoją wiedzę! 34

Rozdział 2.
Komponenty Reacta i składnia JSX **35**
- Wprowadzenie 35
- Czym są komponenty? 36
  - Po co stosować komponenty? 37
  - Budowa komponentu 38
  - Czym dokładnie są funkcje tworzące komponenty? 40
- Co React robi z tymi wszystkimi komponentami? 41
  - Wbudowane komponenty 45
  - Konwencje nazewnicze 46
- JSX, HTML i czysty JavaScript 47
  - Używanie Reacta bez składni JSX 49
  - Elementy JSX są traktowane jak zwykłe wartości z JavaScriptu 50
  - Elementy JSX muszą być samozamykające 52
- Zwracanie dynamicznych treści 52
  - Kiedy należy dzielić komponenty? 53

Kup ksi k

Pole ksi k

Podsumowanie i najważniejsze informacje do zapamiętania 54
Co dalej? 55
Sprawdź swoją wiedzę! 55
Zastosuj zdobytą wiedzę 56
Ćwiczenie 2.1. Utwórz aplikację Reacta z własną prezentacją 56
Ćwiczenie 2.2. Tworzenie aplikacji Reacta do zapisywania celów związanych z tą książką 57

Rozdział 3.
Komponenty i propy 59
Wprowadzenie 59
To jeszcze nie koniec 59
Używanie propów w komponentach 60
Przekazywanie propów do komponentów 60
Używanie propów w komponencie 61
Komponenty, propy i możliwość wielokrotnego użytku 63
Specjalny prop children 63
W których komponentach potrzebne są propy? 64
Jak zarządzać wieloma propami? 65
Stosowanie operatora rozwijania do propów 66
Łańcuchy propów 68
Podsumowanie i najważniejsze informacje do zapamiętania 69
Co dalej? 69
Sprawdź swoją wiedzę! 70
Zastosuj zdobytą wiedzę 70
Ćwiczenie 3.1. Tworzenie aplikacji wyświetlającej Twoje cele związane z tą książką 70

Rozdział 4.
Praca ze zdarzeniami i stanem 72
Wprowadzenie 72
Na czym polega problem? 73
Jak nie rozwiązywać tego problemu 74
Lepsze, ale wciąż niepoprawne rozwiązanie 76
Poprawne reagowanie na zdarzenia 77
Poprawne aktualizowanie stanu 80
Więcej o haczyku useState() 81
Spojrzenie na wewnętrzne mechanizmy Reacta 83
Konwencje nazewnicze 84
Dozwolone typy wartości stanu 85

Kup ksi k Pole ksi k

Praca z wieloma wartościami stanu ... 86
Używanie wielu wycinków stanu ... 86
Zarządzanie połączonymi obiektami stanu ... 88
Poprawne aktualizowanie stanu na podstawie jego wcześniejszej wartości ... 90
Wiązanie dwukierunkowe ... 94
Generowanie wartości na podstawie stanu ... 95
Praca z formularzami i przesyłanie formularzy ... 97
Przenoszenie stanu wyżej ... 99
Podsumowanie i najważniejsze informacje do zapamiętania ... 102
Co dalej? ... 103
Sprawdź swoją wiedzę! ... 103
Zastosuj zdobytą wiedzę ... 104
Ćwiczenie 4.1. Tworzenie prostego kalkulatora ... 104
Ćwiczenie 4.2. Rozbudowywanie kalkulatora ... 105

Rozdział 5.
Wyświetlanie list i warunkowe wyświetlanie treści ... 107
Wprowadzenie ... 107
Czym są warunkowo wyświetlane treści i dane z list? ... 108
Warunkowe wyświetlanie treści ... 109
Różne sposoby warunkowego wyświetlania treści ... 112
Warunkowe ustawianie znaczników elementów ... 117
Wyświetlanie danych z list ... 118
Przekształcanie danych z list za pomocą metody map() ... 121
Aktualizowanie list ... 123
Problem z elementami list ... 125
Ratunek ze strony kluczy ... 128
Podsumowanie i najważniejsze informacje do zapamiętania ... 130
Co dalej? ... 130
Sprawdź swoją wiedzę! ... 131
Zastosuj zdobytą wiedzę ... 131
Ćwiczenie 5.1. Warunkowe wyświetlanie komunikatu o błędzie ... 132
Ćwiczenie 5.2. Wyświetlanie listy produktów ... 133

Rozdział 6.
Dodawanie stylów do aplikacji Reacta ... 134
Wprowadzenie ... 134
Jak działają style w aplikacjach Reacta? ... 135
Używanie stylów wewnątrzwierszowych ... 138
Ustawianie stylów za pomocą klas CSS ... 140

Kup ksi k Pole ksi k

Dynamiczne ustawianie stylów 142
Warunkowe dodawanie stylów 144
Łączenie kilku klas CSS w dynamiczny sposób 145
Scalanie obiektów zawierających style wewnątrzwierszowe 146
Tworzenie komponentów umożliwiających modyfikowanie stylów 147
Problem ze stylami bez ograniczenia zasięgu 149
Ograniczanie zasięgu stylów za pomocą modułów CSS 150
Biblioteka styled-components 153
Używanie innych bibliotek i platform stylów CSS lub JavaScriptu 155
Podsumowanie i kluczowe informacje do zapamiętania 156
Co dalej? 157
Sprawdź swoją wiedzę! 157
Zastosuj zdobytą wiedzę 158
Ćwiczenie 6.1. Wyświetlanie informacji o poprawności danych wejściowych po przesłaniu formularza 158
Ćwiczenie 6.2. Używanie modułów CSS do ograniczania zasięgu stylów 159

Rozdział 7.
Portale i referencje 161
Wprowadzenie 161
Świat bez referencji 162
Referencje a stan 165
Używanie referencji w celach innych niż dostęp do modelu DOM 167
Przekazywanie referencji 170
Komponenty kontrolowane i niekontrolowane 175
Gdzie elementy z Reacta trafiają w modelu DOM 178
Ratunek ze strony portali 181
Podsumowanie i najważniejsze informacje do zapamiętania 183
Co dalej? 183
Sprawdź swoją wiedzę! 184
Zastosuj zdobytą wiedzę 184
Ćwiczenie 7.1. Pobieranie wartości wejściowych od użytkownika 184
Ćwiczenie 7.2. Dodawanie menu bocznego 185

Rozdział 8.
Zarządzanie efektami ubocznymi 188
Wprowadzenie 188
Czego dotyczy problem? 189
Efekty uboczne 191
Efekty uboczne dotyczą nie tylko żądań HTTP 193

Kup ksi k Pole ksi k

Zarządzanie efektami ubocznymi za pomocą haczyka useEffect() 194
Jak używać haczyka useEffect()? 196
Efekty uboczne i zależności 197
Zbędne zależności 199
Porządkowanie stanu po wystąpieniu efektów ubocznych 202
Zarządzanie wieloma funkcjami powodującymi efekty uboczne 206
Zależności w postaci funkcji 207
Unikanie niepotrzebnego wykonywania funkcji powodującej efekty uboczne 212
Efekty uboczne i kod asynchroniczny 218
Reguły związane z haczykami 219
Podsumowanie i najważniejsze informacje do zapamiętania 220
Co dalej? 221
Sprawdź swoją wiedzę! 221
Zastosuj zdobytą wiedzę 222
Ćwiczenie 8.1. Tworzenie prostego bloga 222

Rozdział 9.
Na zapleczu Reacta i możliwości optymalizacji 224

Wprowadzenie 224
Jeszcze o przetwarzaniu i aktualizowaniu komponentów 225
Co się dzieje w momencie wywołania funkcji tworzącej komponent? 227
Wirtualna wersja modelu DOM a rzeczywisty model DOM 228
Grupowanie zmian stanu 230
Unikanie niepotrzebnego wykonywania komponentów podrzędnych 232
Unikanie kosztownych obliczeń 237
Haczyk useCallback() 241
Unikanie niepotrzebnego pobierania kodu 243
Zmniejszanie wielkości pakietów dzięki podziałowi kodu (i leniwemu wczytywaniu) 244
Tryb strict 250
Debugowanie kodu i narzędzia dla deweloperów Reacta 251
Podsumowanie i najważniejsze informacje do zapamiętania 255
Co dalej? 255
Sprawdź swoją wiedzę! 256
Zastosuj zdobytą wiedzę 256
Ćwiczenie 9.1. Optymalizowanie istniejącej aplikacji 256

Kup ksi k Pole ksi k

Rozdział 10.
Praca ze złożonym stanem 261
Wprowadzenie 261
Problem ze stanem używanym dla kilku komponentów 262
Korzystanie z kontekstu do zarządzania stanem używanym
przez wiele komponentów 265
Udostępnianie wartości kontekstu i zarządzanie nimi 267
Używanie kontekstu w komponentach zagnieżdżonych 270
Zmiana kontekstu z poziomu komponentów zagnieżdżonych 272
Lepsze automatyczne uzupełnianie kodu 273
Kontekst czy przenoszenie stanu wyżej? 274
Przenoszenie logiki zarządzania kontekstem do odrębnych komponentów 274
Łączenie wielu kontekstów 276
Ograniczenia funkcji useState() 277
Zarządzanie stanem za pomocą haczyka useReducer() 279
Funkcje redukujące 280
Zgłaszanie operacji 281
Podsumowanie i najważniejsze informacje do zapamiętania 284
Co dalej? 285
Sprawdź swoją wiedzę! 285
Zastosuj zdobytą wiedzę 285
Ćwiczenie 10.1. Modyfikowanie aplikacji, aby używała API kontekstu 286
Ćwiczenie 10.2. Zastępowanie haczyka useState() haczykiem useReducer() 287

Rozdział 11.
Tworzenie niestandardowych haczyków Reacta 290
Wprowadzenie 290
Po co tworzyć niestandardowe haczyki? 291
Czym są niestandardowe haczyki? 293
Pierwszy niestandardowy haczyk 294
Niestandardowe haczyki dają dużo swobody 297
Parametry w niestandardowych haczykach 298
Wartości zwracane przez niestandardowe haczyki 299
Bardziej złożony przykład 301
Podsumowanie i najważniejsze informacje do zapamiętania 309
Co dalej? 309
Sprawdź swoją wiedzę! 310
Zastosuj zdobytą wiedzę 310
Ćwiczenie 11.1. Utwórz niestandardowy haczyk
dla danych wejściowych z klawiatury 310

Kup ksi k Pole ksi k

Rozdział 12.
Aplikacje wielostronicowe oparte na bibliotece React Router 313

Wprowadzenie 313
Jedna strona to za mało 314
Rozpoczynanie pracy z pakietem React Router i definiowanie tras 315
Dodawanie nawigacji do strony 318
Od komponentu Link do komponentu NavLink 323
Komponenty Route a „zwykłe” komponenty 327
Od tras statycznych do dynamicznych 331
Pobieranie parametrów trasy 334
Tworzenie dynamicznych odsyłaczy 336
Programowe nawigowanie po witrynie 338
Przekierowania 341
Trasy zagnieżdżone 342
Obsługa niezdefiniowanych tras 345
Leniwe wczytywanie 346
Podsumowanie i najważniejsze informacje do zapamiętania 347
Co dalej? 348
Sprawdź swoją wiedzę! 349
Zastosuj zdobytą wiedzę 349
Ćwiczenie 12.1. Tworzenie prostej trzystronicowej witryny 349
Ćwiczenie 12.2. Wzbogacanie podstawowej witryny 351

Rozdział 13.
Zarządzanie danymi za pomocą biblioteki React Router 353

Wprowadzenie 353
Pobieranie danych i routing są ze sobą ściśle powiązane 354
Wysyłanie żądań HTTP bez biblioteki React Router 355
Wczytywanie danych z użyciem biblioteki React Router 358
Włączanie dodatkowych mechanizmów biblioteki React Router 362
Wczytywanie danych na potrzeby tras dynamicznych 364
Funkcje wczytujące dane, żądania i kod po stronie klienta 366
Jeszcze o układach stron 367
Ponowne używanie danych w trasach 371
Obsługa błędów 374
A teraz o przesyłaniu danych 376
Używanie funkcji action() i metody formData() 379
Zwracanie danych zamiast przekierowywania użytkownika 382
Kontrolowanie, które operacje mają być uruchamiane
przez poszczególne elementy <Form> 384

Kup ksi k Pole ksi k

Wyświetlanie obecnego stanu nawigacji 385
Programowe przesyłanie formularzy 387
Pobieranie i przesyłanie danych od kuchni 388
Odraczanie wczytywania danych 391
Podsumowanie i najważniejsze informacje do zapamiętania 394
Co dalej? 395
Sprawdź swoją wiedzę! 395
Zastosuj zdobytą wiedzę 396
Ćwiczenie 13.1. Aplikacja z listą zadań do wykonania 396

Rozdział 14.
Dalsze kroki i dodatkowe materiały 400
Wprowadzenie 400
Co teraz? 401
Ciekawe problemy do zbadania 401
Często używane i popularne biblioteki Reacta 405
Inne materiały 405
React umożliwia tworzenie nie tylko aplikacji internetowych 406
Zakończenie 407

Dodatek 408
Rozdział 2. Komponenty Reacta i składnia JSX 408
Ćwiczenie 2.1. Tworzenie aplikacji Reacta do zapisywania celów związanych z tą książką 408
Ćwiczenie 2.2. Tworzenie aplikacji Reacta do zapisywania celów związanych z tą książką 409
Rozdział 3. Komponenty i propy 412
Ćwiczenie 3.1. Tworzenie aplikacji wyświetlającej Twoje cele związane z tą książką 412
Rozdział 4. Praca ze zdarzeniami i stanem 414
Ćwiczenie 4.1. Tworzenie prostego kalkulatora 414
Ćwiczenie 4.2. Rozbudowywanie kalkulatora 417
Rozdział 5. Wyświetlanie list i warunkowe wyświetlanie treści 420
Ćwiczenie 5.1. Warunkowe wyświetlanie komunikatu o błędzie 420
Ćwiczenie 5.2. Wyświetlanie listy produktów 423
Rozdział 6. Dodawanie stylów do aplikacji Reacta 426
Ćwiczenie 6.1. Wyświetlanie informacji o poprawności danych wejściowych po przesłaniu formularza 426
Ćwiczenie 6.2. Używanie modułów CSS do ograniczania zasięgu stylów 432

Kup ksi k Pole ksi k

Rozdział 7. Portale i referencje 436
Ćwiczenie 7.1. Pobieranie wartości wejściowych od użytkownika 436
Ćwiczenie 7.2. Dodawanie menu bocznego 438
Rozdział 8. Zarządzanie efektami ubocznymi 441
Ćwiczenie 8.1. Tworzenie prostego bloga 441
Rozdział 9. Na zapleczu Reacta i możliwości optymalizacji 446
Ćwiczenie 9.1. Optymalizowanie istniejącej aplikacji 446
Rozdział 10. Praca ze złożonym stanem 453
Ćwiczenie 10.1. Modyfikowanie aplikacji, aby używała API kontekstu 453
Ćwiczenie 10.2. Zastępowanie haczyka useState() haczykiem useReducer() 459
Rozdział 11. Tworzenie niestandardowych haczyków Reacta 463
Ćwiczenie 11.1. Utwórz niestandardowy haczyk dla danych wejściowych z klawiatury 463
Rozdział 12. Aplikacje wielostronicowe oparte na bibliotece React Router 466
Ćwiczenie 12.1. Tworzenie prostej trzystronicowej witryny 466
Ćwiczenie 12.2. Wzbogacanie podstawowej witryny 473
Rozdział 13. Zarządzanie danymi za pomocą biblioteki React Router 477
Ćwiczenie 13.1. Aplikacja z listą zadań do wykonania 477

Odpowiedzi 488
Rozdział 1. 488
Rozdział 2. 490
Rozdział 3. 491
Rozdział 4. 492
Rozdział 5. 494
Rozdział 6. 495
Rozdział 7. 496
Rozdział 8. 497
Rozdział 9. 498
Rozdział 10. 499
Rozdział 11. 500
Rozdział 12. 501
Rozdział 13. 502

Kup ksi k Pole ksi k

Komponenty Reacta i składnia JSX

Rozdział

2

Cele nauki

Dzięki lekturze tego rozdziału nauczysz się:

- definiować, czym dokładnie są **komponenty**,
- efektywnie budować komponenty i korzystać z nich,
- posługiwać się standardowymi konwencjami nazewniczymi i wzorcami projektowymi,
- wyjaśniać relacje między komponentami a kodem JSX,
- pisać kod JSX i dowiesz się, dlaczego jest używany,
- pisać komponenty Reacta bez używania kodu JSX,
- jak napisać pierwszą aplikację Reacta.

Wprowadzenie

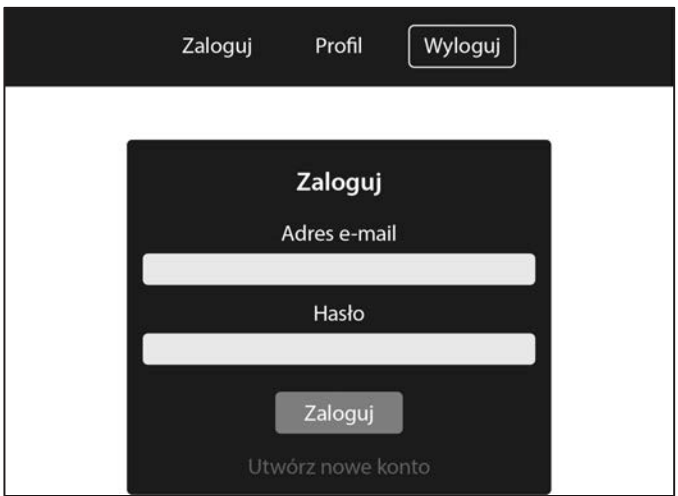
W poprzednim rozdziale omówiłem Reacta na ogólnym poziomie. Wiesz już, czym jest to narzędzie i dlaczego warto rozważyć korzystanie z niego do budowania interfejsów użytkownika. Wiesz już też, jak tworzyć projekty Reacta za pomocą polecenia `npx create-react-app`.

W tym rozdziale poznasz jedno z najważniejszych zagadnień związanych z Reactem — komponenty. Dowiesz się, że są to cegiełki wielokrotnego użytku służące do tworzenia interfejsów użytkownika. Ponadto szczegółowo omawiam kod JSX, dzięki czemu nauczysz się, jak za pomocą komponentu i kodu JSX zbudować swoją pierwszą prostą aplikację Reacta.

Kup ksi   k   Pole   ksi   k

Czym są komponenty?

Bardzo ważnym aspektem Reacta jest używanie tak zwanych komponentów. **Komponenty** są cegiełkami wielokrotnego użytku, które łączy się w celu uzyskania ostatecznego interfejsu użytkownika. Na przykład prosta witryna może się składać z nagłówka, który obejmuje pasek nawigacji, i sekcji głównej, gdzie znajduje się formularz na dane uwierzytelniające (zobacz rysunek 2.1).



Rysunek 2.1. Przykładowy ekran uwierzytelniania z paskiem nawigacji

Gdy przyjrzysz się tej przykładowej stronie, może uda Ci się zidentyfikować różne cegiełki (czyli komponenty). Niektóre z nich są nawet zastosowane kilkakrotnie.

W nagłówku z paskiem nawigacji znajdują się następujące komponenty:

- elementy nawigacyjne (*Zaloguj* i *Profil*),
- przycisk *Wyloguj*.

Poniżej, w sekcji głównej, wyświetlone są następujące informacje:

- kontener zawierający formularz uwierzytelniający,
- elementy na dane wejściowe,
- przycisk logowania,
- odsyłacz do strony *Utwórz nowe konto*.

Warto zauważyć, że niektóre komponenty są zagnieżdżone w innych. Komponenty mogą więc składać się z innych komponentów. Jest to bardzo ważna cecha Reacta i podobnych bibliotek.

Po co stosować komponenty?

Niezależnie od tego, którą stronę oglądasz, wszystkie one składają się z tego rodzaju cegiełek. Nie jest to koncepcja ani idea specyficzna dla Reacta. Co więcej, sam HTML „myśli" w kategoriach komponentów. Używane są w nim elementy `<img>`, `<header>`, `<nav>` itd. Możesz je łączyć ze sobą, aby opisywać i strukturyzować zawartość witryny.

W Reakcie *przyjęto* ideę podziału strony internetowej na cegiełki wielokrotnego użytku, ponieważ to podejście umożliwia programistom pracę nad niewielkimi, łatwymi w zarządzaniu porcjami kodu. Praca nad kodem i jego konserwacja są wtedy łatwiejsze niż w sytuacji, gdy masz jeden wielki plik HTML (lub plik z kodem Reacta).

To dlatego różne narzędzia (zarówno biblioteki do tworzenia frontendów, na przykład React lub Angular, jak i biblioteki backendowe i systemy zarządzania szablonami, takie jak **EJS** (ang. *Embedded JavaScript*), także wykorzystują komponenty, przy czym stosowana może być inna terminologia, na przykład *partials* lub *includes*.

Uwaga

EJS to popularny system zarządzania szablonami dla JavaScriptu. Jest on często stosowany przede wszystkim do tworzenia backendów witryn internetowych z użyciem narzędzia NodeJS.

W trakcie korzystania z Reacta bardzo ważna jest dbałość o to, by kod był łatwy w zarządzaniu. Należy pracować z małymi komponentami wielokrotnego użytku, ponieważ komponenty Reacta nie są tylko blokami kodu w HTML-u. Zamiast tego komponenty Reacta obejmują także logikę w JavaScripcie i często również style CSS. W skomplikowanych interfejsach użytkownika połączenie znaczników (JSX), logiki (JavaScript) i stylów (CSS) może szybko doprowadzić do powstania dużych porcji kodu, przez co staje się on trudny w konserwacji. Pomyśl o dużym pliku HTML, który obejmuje także kod JavaScript i style CSS. Praca z takim plikiem nie jest przyjemna.

W skrócie można napisać, że w trakcie pracy nad projektem Reacta będziesz stosować wiele komponentów. Kod należy dzielić na małe, łatwe w zarządzaniu cegiełki, a następnie łączyć komponenty w ogólny interfejs użytkownika. Jest to ważny aspekt Reacta.

Uwaga

W trakcie pracy z Reactem należy przyjąć koncepcję pracy z komponentami. Jednak z technicznego punktu widzenia są one opcjonalne. Teoretycznie możesz budować bardzo skomplikowane strony internetowe obejmujące pojedynczy komponent. Nie jest to ani przyjemne, ani praktyczne, jednak technicznie jest to wykonalne i nie powoduje żadnych problemów.

Kup ksi k Pole ksi k

Budowa komponentu

Komponenty są ważne. Jak jednak dokładnie wygląda komponent Reacta? Jak samodzielnie pisać komponenty Reacta?

Oto przykładowy komponent:

```
import { useState } from 'react';

function SubmitButton() {
  const [isSubmitted, setIsSubmitted] = useState(false);

  function submitHandler() {
    setIsSubmitted(true);
  };

  return (
    <button onClick={submitHandler}>
      { isSubmitted ? 'Wczytywanie…' : 'Wyślij' }
    </button>
  );
};

export default SubmitButton;
```

Zwykle fragment kodu tego rodzaju jest zapisywany w odrębnym pliku (na przykład *SubmitButton.js* w podkatalogu */components* w katalogu */src* projektu Reacta) i importowany do innych plików komponentów, w których dany komponent jest potrzebny. Na przykład poniższy komponent importuje zdefiniowany wcześniej komponent `Sub↪mitButton` i używa go w instrukcji `return`, by go wyświetlić:

```
import SubmitButton from './submit-button';

function AuthForm() {
  return (
    <form>
      <input type="text" />
      <SubmitButton />
    </form>
  );
};

export default AuthForm;
```

Instrukcje importu w tych przykładach to zwykłe javascriptowe instrukcje `import` z jednym dodatkowym aspektem — w większości projektów Reacta (na przykład tworzonych za pomocą instrukcji `npx create-react-app`) można pominąć rozszerzenie pliku (tu jest nim *.js*). `import` i `eksport` to standardowe słowa kluczowe z JavaScriptu pomocne przy podziale powiązanego kodu między wiele plików. Elementy takie jak zmienne,

Kup książkę Poleć książkę

stałe, klasy lub funkcje można eksportować za pomocą instrukcji `export` lub `export default`, dzięki czemu po zaimportowaniu tych elementów można z nich korzystać w innych plikach.

Uwaga

Jeśli koncepcja podziału kodu między wiele plików oraz stosowania instrukcji `import` i `eksport` jest dla Ciebie czymś nowym, możesz najpierw zapoznać się z bardziej podstawowymi materiałami na temat JavaScriptu z omówieniem tych zagadnień. Na przykład w witrynie MDN dostępny jest świetny artykuł z objaśnieniem podstaw: *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules*.

Komponenty przedstawione w tych przykładach są bardzo uproszczone, a ponadto obejmują mechanizmy, których jeszcze nie omówiłem (na przykład wywołanie `useState()`). Jednak ogólna idea tworzenia samodzielnych cegiełek, które można łączyć ze sobą, powinna być zrozumiała.

W trakcie pracy z Reactem możesz stosować dwa różne sposoby definiowania komponentów:

- **komponenty klasowe** są definiowane z użyciem słowa kluczowego `class`,
- **komponenty funkcyjne** są definiowane za pomocą zwykłych funkcji JavaScriptu.

We wszystkich przykładach przedstawionych do tego miejsca komponenty były budowane za pomocą funkcji JavaScriptu. Programista korzystający z JavaScriptu musi stosować jedno z dwóch wymienionych podejść, ponieważ React oczekuje, że komponenty będą funkcjami lub klasami.

Uwaga

Do drugiej połowy 2018 roku niektóre rodzaje zadań wymagały stosowania komponentów klasowych. Mam tu na myśli konkretnie komponenty korzystające wewnętrznie ze stanu (stan omawiam w dalszym miejscu książki). Jednak w drugiej połowie 2018 roku wprowadzono nową koncepcję — **Haczyki Reacta**. Umożliwiają one wykonywanie wszystkich operacji i zadań za pomocą komponentów funkcyjnych. Dlatego komponenty klasowe wychodzą z użycia i nie omawiam ich w tej książce.

W przedstawionych przykładach warto zwrócić uwagę na kilka aspektów:

- nazwy funkcji tworzących komponenty są pisane wielką literą (na przykład `SubmitButton`),
- w funkcjach tworzących komponenty można definiować inne, wewnętrzne funkcje (na przykład `submitHandler`),

Kup ksi k Pole ksi k

- funkcje tworzące komponenty zwracają kod *podobny do HTML-a* (kod JSX),
- w funkcjach tworzących komponenty można używać mechanizmów takich jak `useState()`,
- funkcje tworzące komponenty można eksportować (za pomocą instrukcji `export default`),
- niektóre mechanizmy (na przykład `useState` lub niestandardowy komponent `SubmitButton`) są importowane, do czego służy słowo kluczowe `import`.

W dalszych punktach szczegółowo omawiam różne zagadnienia związane z komponentami i ich kodem.

Czym dokładnie są funkcje tworzące komponenty?

W Reakcie komponenty są funkcjami (lub klasami, które jednak, jak już wspomniałem, wychodzą z użycia).

Funkcja jest zwykłą konstrukcją z JavaScriptu; nie jest to koncepcja specyficzna dla Reacta. Warto to zapamiętać. React jest biblioteką javascriptową, dlatego *używane są w niej mechanizmy JavaScriptu* (na przykład funkcje). *React nie jest nowym językiem programowania*.

Gdy korzystasz z Reacta, możesz używać zwykłych funkcji javascriptowych do ukrywania kodu HTML (ściślej rzecz biorąc, kodu JSX) i powiązanej ze znacznikami logiki w JavaScripcie. Od kodu zapisanego w funkcji zależy, czy będzie ją można potraktować jako komponent Reacta. Na przykład we wcześniejszych fragmentach kodu `submit↪Handler` jest zwykłą funkcją javascriptową, ale nie jest komponentem Reacta. W następnym przykładzie znajduje się inna zwykła funkcja javascriptowa niebędąca komponentem Reacta:

```
function calculate(a, b) {
  return {sum: a + b};
};
```

Funkcja jest traktowana jak komponent i może być używana jak element HTML w kodzie JSX, jeśli zwraca *wyświetlaną* wartość (zwykle w formie kodu JSX). To bardzo ważne. Funkcji możesz używać jako komponentu Reacta w kodzie JSX tylko w sytuacji, gdy zwraca ona coś, co można wyświetlić w Reakcie. Zwracana wartość technicznie nie musi być kodem JSX, jednak przeważnie ma właśnie taką postać. Przykład ze zwracaniem wartości innej niż kod JSX znajdziesz dalej w książce, w rozdziale 7. „Portale i referencje”.

We fragmencie kodu, w którym zdefiniowane są funkcje `SubmitButton` i `AuthForm`, te dwie funkcje są komponentami Reacta, ponieważ obie zwracają kod JSX (czyli kod, który

Kup ksi k Pole ksi k

może być wyświetlany przez Reacta). Gdy funkcja spełnia warunek bycia komponentem Reacta, można jej używać jak elementu HTML w kodzie JSX. Użyłem na przykład funkcji `<SubmitButton />` jak samozamykającego elementu HTML.

W czystym JavaScripcie oczywiście zwykle wywołujesz funkcje, by je wykonać. Jednak z komponentami funkcyjnymi jest inaczej. React wywołuje je za Ciebie, dlatego programista może używać ich jak elementów HTML w kodzie JSX.

Uwaga

Jeśli chodzi o wartości z możliwością wyświetlania, warto zauważyć, że najczęściej zwracanym lub używanym rodzajem takich wartości jest kod JSX, czyli znaczniki zdefiniowane za pomocą składni JSX. Jest to zrozumiałe, ponieważ w JSX można definiować strukturę treści i interfejs użytkownika w sposób podobny jak za pomocą HTML-a.

Jednak obok znaczników JSX jest też kilka innych ważnych wartości, które umożliwiają wyświetlanie, dlatego mogą być zwracane przez niestandardowe komponenty (zamiast kodu JSX). Najważniejsze jest to, że możesz zwracać także łańcuchy znaków lub liczby, jak również tablice przechowujące elementy JSX, łańcuchy znaków lub liczby.

Co React robi z tymi wszystkimi komponentami?

Jeśli prześledzisz sekwencję wszystkich komponentów i instrukcji `import` oraz `export` do samego początku, znajdziesz instrukcję `root.render(...)` w głównym skrypcie wejściowym projektu React. Zwykle ten skrypt jest umieszczony w pliku *index.js* w katalogu *src/* projektu. Metoda `render()` udostępniana przez bibliotekę React (a konkretnie przez pakiet `react-dom`) przyjmuje fragment kodu JSX, po czym interpretuje go i wykonuje.

Kompletny fragment kodu w głównym pliku wejściowym (*index.js*) zazwyczaj wygląda mniej więcej tak:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Dokładny kod, jaki znajdziesz w nowym projekcie Reacta, może wyglądać nieco inaczej.

Kup ksi k Pole ksi k

Może się w nim znajdować na przykład dodatkowy element `<StrictMode>` obejmujący element `<App>`. Element `<StrictMode>` aktywuje dodatkowe testy, które pomagają wykryć subtelne błędy w kodzie Reacta. Jednocześnie może też prowadzić do zaskakujących zachowań i nieoczekiwanych komunikatów o błędach, co dzieje się zwłaszcza w trakcie eksperymentowania z Reactem lub poznawania tego narzędzia. Ponieważ w tej książce chcę przede wszystkim opisać podstawowe mechanizmy i kluczowe koncepcje związane z Reactem, nie omawiam tu elementu `<StrictMode>`.

Aby dalsza praca przebiegała płynnie, warto uporządkować nowo utworzony plik *index.js*, by wyglądał podobnie jak przedstawiony wcześniej fragment kodu.

Metoda `createRoot()` informuje Reacta, że należy utworzyć nowy **punkt wejścia**, który posłuży do wstrzyknięcia wygenerowanego interfejsu użytkownika do dokumentu HTML przekazywanego użytkownikom witryny. Tak więc argument przekazywany do metody `createRoot()` to wskaźnik do elementu modelu DOM z pliku *index.html*. Ten plik zawiera jedną stronę udostępnianą użytkownikom witryny.

W wielu sytuacjach wspomnianym argumentem jest wywołanie `document.getElement` `↪ById('root')`. Ta wbudowana metoda z czystego JavaScriptu zwraca referencję do elementu modelu DOM, który to element występuje już w dokumencie *index.html*. Dlatego programista musi się upewnić, że element o podanej wartości atrybutu `id` (tu jest nią `root`) istnieje w pliku HTML, do którego wczytywany jest skrypt aplikacji Reacta. W domyślnym projekcie Reacta utworzonym za pomocą instrukcji `npx create-react-app` potrzebny element jest dostępny. W pliku *index.html* z katalogu *public/* znajdziesz element `<div id="root">`.

Wspomniany plik *index.html* zawiera niewiele kodu i stanowi tylko szkielet aplikacji Reacta. React potrzebuje tylko punktu wejścia (zdefiniowanego za pomocą wywołania `createRoot()`), który posłuży do dołączenia wygenerowanego interfejsu użytkownika do wyświetlanej witryny. Tak więc plik HTML i jego zawartość nie definiują bezpośrednio zawartości witryny. Ten plik jest tylko punktem początkowym dla aplikacji Reacta. React może następnie przejąć kontrolę i sterować rzeczywistym interfejsem użytkownika.

Po zdefiniowaniu głównego punktu wejścia można wywołać metodę `render()` obiektu `root` utworzonego za pomocą wywołania `createRoot()`:

```
root.render(<App />);
```

Metoda `render()` informuje Reacta, jaką treść (na przykład który komponent Reacta) należy wstrzyknąć do głównego punktu wejścia. W większości aplikacji Reacta tą treścią jest komponent o nazwie `App`. React generuje następnie odpowiednie instrukcje do operowania modelem DOM, aby odzwierciedlić na stronie internetowej znaczniki zdefiniowane w kodzie JSX komponentu `App`.

Kup książkę Poleć książkę

Komponent App jest funkcją tworzącą komponent zaimportowaną z innego pliku. W domyślnym projekcie Reacta tworząca komponent funkcja App jest zdefiniowana w pliku *App.js*, który także znajduje się w katalogu */src*, i eksportowana za pomocą tego pliku.

Przekazywany do funkcji render() komponent (zwykle <App />) można nazwać **komponentem głównym** aplikacji Reacta. Jest to podstawowy komponent wyświetlany w modelu DOM. Wszystkie pozostałe komponenty są zagnieżdżone w kodzie JSX tego komponentu App lub w kodzie JSX głębiej zagnieżdżonych komponentów podrzędnych. Możesz sobie wyobrazić, że wszystkie te komponenty tworzą drzewo komponentów analizowane przez Reacta i przekształcane na instrukcje operujące modelem DOM.

Uwaga

W poprzednim rozdziale wspomniałem, że Reacta można stosować w różnych platformach. Za pomocą pakietu react-native można tworzyć aplikacje mobilne dla systemów iOS i Android. Pakiet react-dom, który udostępnia metodę createRoot() (i tym samym pośrednio metodę render()), jest przeznaczony dla przeglądarek. Stanowi „most" między możliwościami Reacta i instrukcjami przeglądarki potrzebnymi do wyświetlania i działania w niej interfejsu użytkownika (opisanego za pomocą kodu JSX i komponentów Reacta). Jeśli chcesz zbudować aplikację na inne platformy, trzeba zastąpić wywołania ReactDOM.createRoot() i render() (oczywiście istnieją potrzebne analogiczne metody).

Niezależnie od tego, czy używasz tworzącej komponent funkcji jak elementu HTML w kodzie JSX innych komponentów, czy korzystasz z niej jak z elementu HTML przekazywanego jako argument do metody render(), React odpowiada za interpretowanie i wykonywanie tej funkcji.

Oczywiście nie jest to nowa koncepcja. W JavaScripcie funkcje są obiektami pierwszoklasowymi, co oznacza, że możesz przekazywać funkcje jako argumenty do innych funkcji. To dzieje się w omawianym scenariuszu, przy czym tu pojawia się zmiana w postaci zastosowania składni JSX, która nie jest domyślnie dostępna w JavaScripcie.

React wykonuje za programistę funkcje tworzące komponenty i przekształca zwrócony kod JSX na instrukcje operujące modelem DOM. Dokładnie rzecz biorąc, React porusza się po zwróconym kodzie JSX i przechodzi do każdego niestandardowego komponentu, jaki może występować w kodzie JSX, do momentu uzyskania kodu JSX składającego się wyłącznie z natywnych, wbudowanych elementów HTML (w ujęciu technicznym nie jest to jeszcze kod HTML, co jednak wyjaśniam dalej w tym rozdziale).

Przyjrzyj się tym dwóm komponentom:

```
function Greeting() {
  return <p>Witaj w tej książce!</p>;
```

Kup ksi k Pole ksi k

```
};

function App() {
  return (
    <div>
      <h2>Witaj, świecie!</h2>
      <Greeting />
    </div>
  );
};

const root = ReactDOM.createRoot(document.getElementById('app'));
root.render(<App />);
```

Komponent `App` używa komponentu `Greeting` w kodzie JSX. React analizuje całą strukturę znaczników JSX i zwraca poniższy końcowy kod JSX:

```
root.render((
  <div>
    <h2>Witaj, świecie!</h2>
    <p>Witaj w tej książce!</p>
  </div>
), document.getElementById('app'));
```

Ten kod nakazuje pakietom `react` i `react-dom` wykonanie następujących operacji na modelu DOM:

- utworzenie elementu `<div>`,
- utworzenie w `<div>` dwóch elementów podrzędnych: `<h2>` i `<p>`,
- ustawienie tekstu elementu `<h2>` na `'Witaj, świecie!'`,
- ustawienie tekstu elementu `<p>` na `'Witaj w tej książce!'`,
- wstawienie elementu `<div>` wraz z elementami podrzędnymi do już istniejącego elementu modelu DOM o identyfikatorze `'app'`.

Jest to nieco uproszczony opis, jednak możesz przyjąć, że React obsługuje komponenty i kod JSX w taki właśnie sposób.

Uwaga

W rzeczywistości React wewnętrznie nie używa kodu JSX, jednak programistom łatwiej jest posługiwać się takim kodem. Dalej w rozdziale dowiesz się, na jaką postać przekształcany jest kod JSX i jak naprawdę wygląda kod używany przez Reacta.

Kup ksi k Pole ksi k

Wbudowane komponenty

We wcześniejszych przykładach pokazałem, że możesz tworzyć własne niestandardowe komponenty w postaci funkcji zwracających kod JSX. Co więcej, jest to jedno z głównych zadań, jakie będziesz nieustannie wykonywać jako programista Reacta — będziesz tworzyć funkcje tworzące komponenty. Dużo takich funkcji.

Jednak ostatecznie po scaleniu całego kodu JSX w jeden duży blok, tak jak w ostatnim przykładzie, otrzymasz porcję kodu JSX zawierającą wyłącznie standardowe elementy HTML: `<div>`, `<h2>`, `<p>` itd.

Gdy używasz Reacta, nie musisz tworzyć nowych elementów HTML, które przeglądarka potrafi wyświetlić i obsługiwać. Zamiast tego tworzysz komponenty *działające tylko w środowisku Reacta*. Zanim dotrą one do przeglądarki, są analizowane przez Reacta i „tłumaczone” na instrukcje w JavaScripcie operujące modelem DOM (na przykład `document.append(…)`).

Pamiętaj przy tym, że JSX nie jest częścią języka JavaScript. Jest to wyłącznie **lukier składniowy** (czyli uproszczenie składni kodu) zapewniany przez bibliotekę React i projekt, w którym piszesz kod Reacta. Tak więc w kodzie JSX elementy takie jak `<div>` także *nie są zwykłymi elementami HTML*, ponieważ *nie piszesz kodu w HTML-u*. Kod pozornie może tak wyglądać, jednak znajduje się w pliku *.js* i nie jest kodem HTML. Jest to specjalny kod JSX. Należy o tym pamiętać.

Dlatego elementy `<div>` i `<h2>` występujące w przykładach to tak naprawdę także komponenty Reacta. Nie są one jednak komponentami zbudowanymi przez Ciebie, tylko komponentami udostępnianymi przez Reacta (a konkretnie przez pakiet ReactDOM).

W trakcie pracy z Reactem zawsze ostatecznie korzystasz z tych podstawowych mechanizmów — z wbudowanych tworzących komponenty funkcji, które są później przekształcane na instrukcje przeglądarki generujące i dodające lub usuwające zwykłe elementy modelu DOM. Niestandardowe komponenty buduje się w celu pogrupowania elementów w taki sposób, aby otrzymać cegiełki wielokrotnego użytku, których można używać do budowania interfejsu użytkownika. Jednak ostatecznie interfejs użytkownika składa się ze zwykłych elementów HTML.

Uwaga

Jeśli znasz się na programowaniu frontendów stron internetowych, możliwe, że wiesz, czym jest technologia **Web Components**. Umożliwia ona tworzenie nowych elementów HTML za pomocą czystego JavaScriptu.

Wspomniałem już, że w Reakcie ten mechanizm nie jest używany. W Reakcie nie tworzysz nowych, niestandardowych elementów HTML.

Konwencje nazewnicze

Wszystkie tworzące komponenty funkcje, jakie znajdziesz w tej książce, noszą nazwy takie jak `SubmitButton`, `AuthForm`, `Greeting` itd.

Dla funkcji w Reakcie możesz stosować dowolne nazwy (przynajmniej w pliku, w którym je definiujesz). Jednak zwyczajowo używana jest NotacjaPascalowa, zgodnie z którą pierwsza litera jest wielka, a kolejne słowa są pogrupowane w jedno słowo (na przykład `SubmitButton` zamiast `Submit Button`), przy czym każde „podsłowo” rozpoczyna się wielką literą.

W miejscu, w którym definiujesz funkcję tworzącą komponent, taki zapis jest tylko konwencją nazewniczą, a nie sztywną regułą. Jednak ta notacja *jest* wymagana w miejscu, w którym *używasz* funkcji tworzących komponenty, czyli w kodzie JSX z osadzonymi własnymi niestandardowymi komponentami.

Nie możesz użyć własnego niestandardowego komponentu w następujący sposób:

```
<greeting />
```

React wymaga używania wielkiej początkowej litery w nazwach własnych niestandardowych komponentów, gdy są one stosowane w kodzie JSX. Ta reguła ma umożliwić Reactowi jednoznaczne i łatwe odróżnianie nazw niestandardowych komponentów od nazw komponentów wbudowanych (`<div>` itd.). React musi sprawdzić tylko pierwszą literę, aby stwierdzić, czy jest to element wbudowany, czy niestandardowy.

Ważne jest, aby oprócz konwencji nazewniczych dla funkcji tworzących komponenty znać także konwencje nazewnicze dotyczące plików. Niestandardowe komponenty zwykle umieszcza się w odrębnych plikach w katalogu *src/components/*. Nie jest to jednak sztywna reguła. Możesz zmienić dokładną lokalizację i nazwę tego katalogu, przy czym powinien on znajdować się w katalogu *src/*. Standardowo stosowana jest jednak nazwa *components/*.

Choć dla funkcji tworzących komponenty standardowo używana jest NotacjaPascalowa, nie istnieje analogiczna domyślna notacja dla nazw plików. Część programistów stosuje NotacjęPascalową także dla nazw plików. Na przykład w nowych projektach Reacta tworzonych w sposób opisany w tej książce komponent `App` znajduje się w pliku *App.js*. Mimo to natrafisz na wiele projektów Reacta, w których komponenty są zapisane w plikach o nazwach zgodnych z notacją-kebab. W tej notacji używane są same małe litery, a kolejne słowa są łączone w jedno za pomocą dywizu. Funkcje tworzące komponenty mogą być wtedy zapisane w plikach mających nazwy w formacie *submit-button.js*.

Ostatecznie to Ty i Twój zespół decydujecie, jaką konwencję nazewniczą stosować dla plików. W tej książce dla nazw plików używam NotacjiPascalowej.

Kup ksi k Pole ksi k

JSX, HTML i czysty JavaScript

Wcześniej wspomniałem, że projekty Reacta zwykle zawierają dużo kodu JSX. Większość niestandardowych komponentów zwraca fragmenty kodu JSX. Dzieje się tak we wszystkich dotychczas zaprezentowanych przykładach, a także w prawie każdym projekcie Reacta, z jakim się zetkniesz. Nie ma przy tym znaczenia, czy używasz Reacta dla przeglądarki, czy dla innych platform, na przykład z użyciem pakietu `react-native`.

Ale czym dokładnie jest kod JSX? Na czym polegają różnice między nim i HTML-em? I jak jest powiązany z czystym JavaScriptem?

JSX to technika, która nie występuje w czystym JavaScripcie. Bardziej zaskakujące jest jednak to, że nie jest ona bezpośrednio częścią także samej biblioteki React.

JSX jest lukrem składniowym stosowanym w procesie budowania całego projektu Reacta. Gdy uruchamiasz roboczy serwer WWW za pomocą polecenia `npm start` lub budujesz aplikację Reacta do użytku produkcyjnego (na przykład w celu wdrożenia) przy użyciu instrukcji `npm run build`, rozpoczynasz proces, który przekształca kod JSX na zwykłe instrukcje w JavaScripcie. Programista nie widzi tych końcowych instrukcji, jednak biblioteka React otrzymuje je i analizuje.

Do jakiej postaci przekształcany jest kod JSX?

Ostatecznie wszystkie fragmenty kodu JSX są przetwarzane w wywołania metody `React.createElement(…)`.

Oto konkretny przykład:

```
function Greeting() {
  return <p>Witaj, świecie!</p>;
};
```

Kod JSX zwracany przez ten komponent jest przekształcany na następujący kod w czystym JavaScripcie:

```
function Greeting() {
  return React.createElement('p', {}, 'Witaj, świecie!');
};
```

Metoda `createMethod()` jest wbudowana w bibliotekę React. W tym miejscu nakazuje ona Reactowi utworzenie elementu reprezentującego akapit z wewnętrzną zagnieżdżoną treścią `'Witaj, świecie!'`. Następnie ten element jest w pierwszej kolejności tworzony wewnętrznie (w **wirtualnej wersji modelu DOM** omówionej dalej w książce, w rozdziale 9. „Na zapleczu Reacta i możliwości optymalizacji”). Po utworzeniu wszystkich elementów na podstawie całego kodu JSX wirtualna wersja modelu DOM jest przekształcana na wykonywane przez przeglądarkę instrukcje operujące rzeczywistym modelem DOM.

Kup ksi k Pole ksi k

Uwaga

Wcześniej wspomniałem, że React (w przeglądarce) jest połączeniem dwóch pakietów: `react` i `react-dom`.

Po opisaniu metody `React.createElement(...)` łatwiej jest objaśnić, w jaki sposób te dwa pakiety współdziałają ze sobą. React wewnętrznie tworzy wirtualną wersję modelu DOM, a następnie przekazuje ją do pakietu `react-dom`. Wtedy ten pakiet generuje instrukcje operujące rzeczywistym modelem DOM, które muszą zostać wykonane, aby zaktualizować stronę i wyświetlić na niej oczekiwany interfejs użytkownika.

Jak już napisałem, ten proces omawiam szczegółowo w rozdziale 9.

Wartość środkowego parametru (w tym przykładzie jest nią `{}`) to obiekt JavaScriptu, który może zawierać dodatkową konfigurację dla tworzonego elementu.

Oto przykład, w którym ten środkowy argument jest istotny:

```
function Advertisement() {
  return <a href="https://my-website.com">Odwiedź moją witrynę</a>;
};
```

Ten kod jest przekształcany na następującą postać:

```
function Advertisement() {
  return React.createElement(
    'a',
    { href: ' https://my-website.com ' },
    'Odwiedź moją witrynę'
  );
};
```

Ostatnim argumentem przekazywanym do metody `React.createElement(...)` jest zagnieżdżona zawartość elementu, czyli treść, która powinna znajdować się między znacznikiem otwierającym a znacznikiem zamykającym. Dla zagnieżdżonych elementów z kodu JSX generowane są zagnieżdżone wywołania `React.createElement(...)`:

```
function Alert() {
  return (
    <div>
      <h2>To jest alert!</h2>
    </div>
  );
};
```

Ten kod jest przekształcany w następujący sposób:

```
function Alert() {
  return React.createElement(
    'div', {}, React.createElement('h2', {}, 'To jest alert!')
  );
};
```

Kup ksi k Pole ksi k

Używanie Reacta bez składni JSX

Ponieważ cały kod JSX i tak jest przekształcany na wywołania metod w natywnym JavaScripcie, możesz tworzyć aplikacje Reacta i oparte na Reakcie interfejsy użytkownika bez kodu JSX.

Jeśli chcesz, możesz całkowicie pominąć kod JSX. Zamiast pisać kod JSX w komponentach i wszystkich miejscach, gdzie taki kod jest oczekiwany, możesz stosować wywołania `React.createElement(…)`.

Na przykład dwa następne fragmenty kodu prowadzą do uzyskania w przeglądarce identycznego interfejsu użytkownika:

```
function App() {
  return (
    <p>Zachęcam do odwiedzin mojego <a href="https://my-blog-site.com"
    ↪>bloga</a></p>
  );
};
```

Powyższy fragment daje ten sam efekt co poniższy kod:

```
function App() {
  return React.createElement(
    'p',
    {},
    [
      'Zachęcam do odwiedzin mojego ',
      React.createElement(
        'a',
        { href: 'https://my-blog-site.com' },
        'bloga'
      )
    ]
  );
};
```

Oczywiście zupełnie inną kwestią jest to, czy warto to robić. Jak widać w tym przykładzie, stosowanie samych wywołań `React.createElement(…)` jest zdecydowanie mniej wygodne. Musisz pisać znacznie więcej kodu, a struktura z głęboko zagnieżdżonymi elementami prowadzi do powstawania wysoce nieczytelnego kodu.

To dlatego programiści korzystający z Reacta zwykle używają kodu JSX. Jest to świetne rozwiązanie, dzięki któremu tworzenie interfejsów użytkownika za pomocą Reacta staje się znacznie przyjemniejsze. Należy jednak pamiętać, że JSX nie jest częścią ani HTML-a, ani czystego JavaScriptu. Jest to jedynie lukier składniowy przekształcany na zapleczu na wywołania `React.createElement(…)`.

Elementy JSX są traktowane jak zwykłe wartości z JavaScriptu

Ponieważ kod JSX jest tylko lukrem składniowym przekształcanym na wywołania `React.createElement()`, warto wiedzieć o kilku istotnych zagadnieniach i regułach:

- elementy JSX są ostatecznie **zwykłymi wartościami z JavaScriptu** (a konkretnie funkcjami),
- reguły dotyczące wszystkich wartości z JavaScriptu obowiązują także dla elementów JSX,
- dlatego w miejscu, w którym oczekiwana jest tylko jedna wartość (na przykład po słowie kluczowym `return`), można użyć tylko jednego elementu JSX.

Poniższy kod spowoduje błąd:

```
function App() {
  return (
    <p>Witaj, świecie!</p>
    <p>Poznaj React!</p>
  );
};
```

Ten kod na pozór wygląda poprawnie, jednak w rzeczywistości jest błędny. W tym przykładzie zwracane są dwie wartości zamiast jednej. W JavaScripcie jest to niedozwolone.

Na przykład poniższy kod, w którym biblioteka React nie jest używana, także jest błędny:

```
function calculate(a, b) {
  return (
    a + b
    a - b
  );
};
```

Nie możesz zwracać więcej niż jednej wartości niezależnie od tego, jak je zapiszesz.

Oczywiście możesz zwrócić tablicę lub obiekt. Na przykład poniższy kod jest poprawny:

```
function calculate(a, b) {
  return [
    a + b,
    a - b
  ];
};
```

Taki kod jest dozwolony, ponieważ zwracana jest tylko jedna wartość — tablica. Ta tablica zawiera kilka wartości, co jest typowe dla tablic. Jest to dopuszczalne i tak samo będzie w kodzie JSX:

```
function App() {
  return [
    <p>Witaj, świecie!</p>,
    <p>Poznaj React!</p>
  ];
};
```

Tego rodzaju kod jest dozwolony, ponieważ zwracasz jedną tablicę zawierającą dwa elementy. W tym przykładzie są to elementy JSX, jednak — jak wcześniej wspomniałem — elementy JSX to zwykłe wartości z JavaScriptu. Możesz więc używać ich w dowolnym miejscu, w jakim oczekiwane są zwykłe wartości.

Jednak w kodzie JSX to podejście stosuje się dość rzadko. Wynika to z tego, że kłopotliwa jest konieczność pamiętania o opakowywaniu elementów JSX w nawiasy kwadratowe. Ponadto kod w mniejszym stopniu przypomina wtedy HTML, co niweczy cel stosowania kodu JSX (wymyślono go po to, aby umożliwić programistom pisanie kodu HTML w plikach javascriptowych).

Gdy potrzebne są równorzędne elementy, tak jak w przedstawianym przykładzie, używany jest specjalnego rodzaju komponent nakładkowy — **fragment** z biblioteki React. Jest to wbudowany komponent, który umożliwia zwracanie lub definiowanie równorzędnych elementów JSX.

```
function App() {
  return (
    <>
      <p>Witaj, świecie!</p>
      <p>Poznaj React!</p>
    </>
  );
};
```

Ten specjalny element `<>...</>` jest dostępny w większości nowych projektów Reacta (na przykład w projektach tworzonych za pomocą instrukcji `npx create-react-app`). Możesz sobie wyobrazić, że taki kod na zapleczu umieszcza elementy JSX w tablicy. Inna możliwość to zastosowanie składni `<React.Fragment>...</React.Fragment>`. Ten wbudowany element jest zawsze dostępny.

Nawiasy () otaczające kod JSX we wszystkich prezentowanych przykładach są niezbędne, aby umożliwić czytelne formatowanie wielowierszowego kodu. W ujęciu technicznym cały kod JSX można zapisać w jednym wierszu, jednak będzie wtedy mało czytelny. Aby zapisać elementy JSX w wielu wierszach, jak robi się to w typowym kodzie HTML

w plikach *.html*, potrzebujesz nawiasów. Są one dla JavaScriptu informacją, gdzie zwracana wartość się zaczyna i gdzie kończy.

Ponieważ elementy JSX są zwykłymi wartościami z JavaScriptu (przynajmniej po przekształceniu na wywołania `React.createElement(…)`), możesz używać elementów JSX we wszystkich miejscach, w których mogą pojawiać się takie wartości.

Do tego miejsca elementy JSX podawałem w instrukcjach `return`, ale takie elementy można też zapisywać w zmiennych lub przekazywać jako argumenty do innych funkcji.

```
function App() {
  const content = <p>Zapisane w zmiennej!</p>; // Poprawne!
  return content;
};
```

Będzie to istotne, gdy przejdę do bardziej zaawansowanych zagadnień, na przykład warunkowych lub powtarzanych treści. Omawiam je w dalszych miejscach książki.

Elementy JSX muszą być samozamykające

Inna ważna reguła związana z elementami JSX dotyczy tego, że muszą być one samozamykające, jeśli między znacznikiem otwierającym a znacznikiem zamykającym nie występuje żadna treść.

```
function App() {
  return <img src="some-image.png" />;
};
```

W zwykłym HTML-u ukośnik na końcu nie jest niezbędny. Zwykły HTML dopuszcza puste elementy, na przykład `<img src="…">`. Możesz dodać ukośnik, ale nie jest on wymagany.

W JSX ukośniki są konieczne, jeśli element nie zawiera żadnej wewnętrznie zagnieżdżonej treści.

Zwracanie dynamicznych treści

Do tej pory we wszystkich przykładach zwracana była statyczna treść, na przykład `<p>Witaj, świecie!</p>`. Jest to treść, która oczywiście nigdy się nie zmienia. Zawsze powoduje ona wyświetlenie akapitu z tekstem *Witaj, świecie!*.

Na tym etapie książki nie znasz jeszcze żadnych narzędzi umożliwiających generowanie dynamicznych treści. Przejdźmy do konkretów — React wymaga stanu (który omawiam w dalszym rozdziale) do modyfikowania wyświetlanych treści na przykład w reakcji na wprowadzenie danych przez użytkownika lub inne zdarzenie.

Kup ksi k Pole ksi k

Ale ponieważ ten rozdział dotyczy składni JSX, warto przyjrzeć się składni zwracania dynamicznych treści, choć na razie tak naprawdę nie będą się one zmieniać.

```
function App() {
  const userName = 'Max';
  return <p>Cześć, mam na imię {userName}!</p>;
};
```

Ten przykład z technicznego punktu widzenia zwraca statyczne dane wyjściowe, ponieważ wartość stałej `userName` nigdy się nie zmienia. Pokazana jest tu jednak składnia zwracania dynamicznych treści w kodzie JSX. Należy użyć otwierającego i zamykającego nawiasu klamrowego, `{…}`, i umieścić między nimi wyrażenie w JavaScripcie, na przykład nazwę zmiennej lub, jak w tym przykładzie, stałej.

W nawiasie klamrowym możesz podać dowolne poprawne wyrażenie w JavaScripcie. Możesz wywołać funkcję, na przykład `{getMyName()}`, lub wykonać wewnątrzwierszowo proste obliczenia, na przykład `{1 + 1}`.

Nie możesz natomiast umieścić w takim nawiasie klamrowym złożonych poleceń takich jak pętle czy instrukcje `if`. Obowiązują tu standardowe reguły z JavaScriptu. Zwracasz (potencjalnie) dynamiczną wartość, dlatego dozwolony jest każdy kod, który daje w wyniku pojedynczą wartość.

Kiedy należy dzielić komponenty?

Gdy będziesz używać Reacta, coraz lepiej poznawać to narzędzie i pracować nad bardziej wymagającymi projektami Reacta, prawdopodobnie natkniesz się na bardzo często zadawane pytanie: *„Kiedy należy rozdzielić jeden komponent React na kilka odrębnych komponentów?”*.

Jak wspomniałem wcześniej w tym rozdziale, w Reakcie najważniejsze są komponenty, dlatego bardzo często w jednym projekcie Reacta występują dziesiątki, setki, a nawet tysiące komponentów.

Jeśli chodzi o podział jednego komponentu Reacta na kilka mniejszych komponentów, nie istnieją sztywne reguły, których trzeba przestrzegać. Możliwe jest nawet umieszczenie całego kodu interfejsu użytkownika w jednym dużym komponencie. Inna możliwość to utworzenie odrębnego niestandardowego komponentu dla każdego elementu HTML i każdego fragmentu treści z interfejsu użytkownika. Jednak oba te podejścia zwykle nie są optymalne. Zamiast tego dobrą praktyczną regułą jest tworzenie odrębnego komponentu Reacta dla każdej **encji danych**, jaką można zidentyfikować.

Jeżeli na przykład generujesz listę zadań do zrobienia, możesz zidentyfikować dwie główne encje: pojedynczy element listy zadań i całą listę. W tym przykładzie sensowne jest utworzenie dwóch odrębnych komponentów zamiast jednego większego komponentu.

Kup ksi k Pole ksi k

Zaletą podziału kodu na kilka komponentów jest to, że poszczególne komponenty są łatwiejsze w zarządzaniu, ponieważ każdy z nich, jak również i każdy plik, zawiera wtedy mniej kodu.

Ale gdy dzielisz komponent na kilka mniejszych, pojawia się nowy problem — jak sprawić, by komponenty były konfigurowalne i nadawały się do wielokrotnego użytku?

```
import Todo from './todo';

function TodoList() {
  return (
    <ul>
      <Todo />
      <Todo />
    </ul>
  );
};
```

W tym przykładzie wszystkie zadania są identyczne, ponieważ używany jest dla nich komponent `<Todo />`, który nie umożliwia konfigurowania. Możliwe, że zechcesz pozwolić na jego konfigurację. W tym celu dodaj niestandardowe atrybuty, na przykład `<Todo text="Poznaj React!" />`, lub przekaż treść między znacznikiem otwierającym a znacznikiem zamykającym, na przykład `<Todo>Poznaj React!</Todo>`.

React oczywiście to umożliwia. W następnym rozdziale poznasz ważną technikę stosowania **propów**, które umożliwiają konfigurowanie komponentów w ten sposób.

Podsumowanie i najważniejsze informacje do zapamiętania

- W Reakcie używane są **komponenty**, czyli cegiełki wielokrotnego użytku łączone w celu zdefiniowania ostatecznego interfejsu użytkownika.
- Komponenty muszą zwracać **wyświetlaną** treść. Zwykle jest nią kod JSX definiujący kod HTML, który ma zostać ostatecznie wygenerowany.
- React udostępnia wiele wbudowanych komponentów. Obok komponentów specjalnych, na przykład <>…</>, dostępne są komponenty odpowiadające wszystkim standardowym elementom HTML.
- Aby umożliwić Reactowi odróżnienie komponentów niestandardowych od wbudowanych, nazwy komponentów niestandardowych w kodzie JSX muszą się zaczynać wielką literą. Dlatego zwykle używane są nazwy w NotacjiPascalowej.

Kup ksi k Pole ksi k

- JSX nie jest ani HTML-em, ani standardowym mechanizmem JavaScriptu. Jest natomiast **lukrem składniowym** dostępnym w procesie budowania witryny we wszystkich projektach Reacta.
- Kod JSX możesz zastąpić wywołaniami `React.createElement(...)`, ale ponieważ prowadzi to do powstawania zdecydowanie mniej czytelnego kodu, zwykle warto unikać tej techniki.
- Gdy używasz elementów JSX, nie możesz podawać wielu równorzędnych elementów w miejscach, gdzie oczekiwana jest pojedyncza wartość, na przykład bezpośrednio po słowie kluczowym `return`.
- Elementy JSX zawsze muszą być samozamykające, jeśli między znacznikiem otwierającym a znacznikiem zamykającym nie ma żadnej treści.
- Treści dynamiczne można zwracać za pomocą nawiasu klamrowego, na przykład `<p>{someText}</p>`.
- W większości projektów Reacta kod interfejsu użytkownika jest dzielony między dziesiątki lub setki komponentów, które są następnie eksportowane i importowane, by je ponownie połączyć.

Co dalej?

Dzięki lekturze tego rozdziału wiesz już dużo o komponentach i kodzie JSX. Następny rozdział stanowi rozwinięcie tej wiedzy. Dowiesz się z niego, jak umożliwić ponowne wykorzystanie komponentów dzięki zapewnieniu ich konfigurowalności.

Zanim jednak przejdziesz dalej, możesz też sprawdzić wiedzę zdobytą do tego miejsca. W tym celu odpowiedz na poniższe pytania i wykonaj opisane ćwiczenia.

Sprawdź swoją wiedzę!

Sprawdź swoją wiedzę z zakresu zagadnień omówionych w tym rozdziale. W tym celu odpowiedz na poniższe pytania. Później możesz porównać swoje odpowiedzi z przykładowymi rozwiązaniami, które znajdziesz na stronie *https://packt.link/iSHGL* i w witrynie wydawnictwa Helion.

1. Po co używane są komponenty?
2. W jaki sposób można utworzyć komponent Reacta?
3. Co powoduje, że zwykła funkcja staje się funkcją tworzącą komponent Reacta?
4. O jakich podstawowych zasadach dotyczących elementów JSX należy pamiętać?
5. W jaki sposób kod JSX jest obsługiwany przez Reacta i ReactDOM?

Kup ksi k Pole ksi k

Zastosuj zdobytą wiedzę

Ten i poprzedni rozdział zapewniły Ci całą wiedzę potrzebną do utworzenia projektu Reacta i zapełnienia go pierwszymi podstawowymi komponentami.

Poniżej znajdziesz dwa pierwsze ćwiczenia w tej książce.

Ćwiczenie 2.1. Utwórz aplikację Reacta z własną prezentacją

Załóżmy, że tworzysz stronę z osobistym portfolio. Na tej stronie chcesz wyświetlać podstawowe informacje na swój temat, na przykład imię i nazwisko lub wiek. Użyj Reacta i zbuduj komponent Reacta, który zwraca tego rodzaju informacje w sposób opisany w ćwiczeniu.

Celem jest utworzenie aplikacji Reacta w sposób przedstawiony w poprzednim rozdziale. Należy więc utworzyć aplikację za pomocą instrukcji `npx create-react-app`, a następnie uruchomić polecenie `npm start`, by uruchomić serwer roboczy. Zmodyfikuj plik *App.js* w taki sposób, aby wyświetlał podstawowe informacje o Tobie. Możesz na przykład wyświetlać imię i nazwisko, adres, stanowisko i inne dane. Możesz samodzielnie wybrać, jakie treści chcesz wyświetlać i których elementów HTML użyjesz.

W tym pierwszym ćwiczeniu chodzi o to, aby przećwiczyć tworzenie projektu i pracę z kodem JSX.

Oto kroki, jakie należy wykonać:

1. Utwórz projekt Reacta za pomocą polecenia `npx create-react-app`.
2. Zmodyfikuj plik *App.js* w katalogu */src* utworzonego projektu i zwróć w nim kod JSX z wybranymi elementami HTML, aby wyświetlić podstawowe informacje na swój temat.

Ostatecznym celem jest uzyskanie danych wyjściowych takich jak na rysunku 2.2.

Cześć, to ja, Max!

Obecnie mam 32 lata i mieszkam w Monachium.

Moje pełne nazwisko to Maximilian Schwarzmüller. Jestem twórcą stron internetowych, a także wysokiej klasy wykładowcą na kursach internetowych.

Rysunek 2.2. Ostateczny wynik ćwiczenia — informacje o użytkowniku wyświetlone na ekranie

Kup ksi k Pole ksi k

Uwaga

Rozwiązanie tego ćwiczenia znajdziesz w dodatku na końcu książki.

Ćwiczenie 2.2. Tworzenie aplikacji Reacta do zapisywania celów związanych z tą książką

Załóżmy, że chcesz dodać do witryny z portfolio nową sekcję, w której planujesz śledzić swoje postępy w nauce. Na tej stronie chcesz zdefiniować i wyświetlać główne cele dotyczące tej książki, na przykład *Poznać najważniejsze funkcje biblioteki React*, *Wykonać wszystkie ćwiczenia* itd.

W tym ćwiczeniu utworzysz nowy projekt Reacta, w którym dodasz *wiele nowych komponentów*. Każdy cel ma być reprezentowany przez odrębny komponent, a wszystkie takie komponenty należy zgrupować w innym komponencie z listą wszystkich celów. Oprócz tego możesz utworzyć dodatkowy komponent nagłówka zawierający tytuł strony internetowej.

Oto kroki potrzebne do wykonania tego ćwiczenia:

1. Utwórz nowy projekt Reacta za pomocą polecenia `npx create-react-app`.
2. W tym projekcie utwórz katalog `components` zawierający pliki z komponentami (dla poszczególnych celów, a także dla listy celów i dla nagłówka strony).
3. W plikach różnych komponentów zdefiniuj i wyeksportuj potrzebne funkcje tworzące komponenty (`FirstGoal`, `SecondGoal`, `ThirdGoal` itd.) dla poszczególnych celów. W każdym pliku umieść jeden komponent.
4. Ponadto zdefiniuj jeden komponent dla całej listy celów (`GoalList`) i inny dla nagłówka strony (`Header`).
5. W komponentach reprezentujących poszczególne cele zwracaj kod JSX z tekstem opisującym cel i odpowiednią strukturą elementów HTML do przechowywania tej treści.
6. W komponencie `GoalList` importuj i zwracaj komponenty reprezentujące poszczególne cele.
7. W komponencie głównym, `App`, importuj i zwracaj komponenty `GoalList` i `Header`. Zastąp tymi operacjami pierwotny kod JSX.

Ostateczne dane wyjściowe przedstawia rysunek 2.3.

Uwaga

Rozwiązanie tego ćwiczenia znajdziesz w dodatku na końcu książki.

Moje cele związane z tą książką

- **Uczyć Reacta w łatwy do zrozumienia sposób**

  Chcę zadbać o to, aby Czytelnicy wynieśli jak najwięcej z tej książki i nauczyli się wszystkiego o Reakcie.

- **Umożliwić Ci wypróbowanie zdobytej wiedzy**

  Uczenie się przez lekturę jest ciekawe i pomocne, jednak w pełni opanować zagadnienie możesz tylko poprzez praktykę! Dlatego też chcę przygotować wiele ćwiczeń, które umożliwią Ci wypróbowanie zdobytej wiedzy.

- **Zachęcić Cię do kontynuowania nauki**

  Programiści uczą się przez całe życie. Chcę zadbać o to, aby nauka była dla Ciebie przyjemnością, a także zmotywować Cię do sięgnięcia po materiały dla zaawansowanych po zakończeniu lektury tej książki. Może będzie to mój kompletny kurs wideo na temat Reacta?

Rysunek 2.3. Ostateczne dane wyjściowe na stronie. Widoczna jest tu lista celów

Kup ksi k Pole ksi k

PROGRAM PARTNERSKI

GRUPY 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

React: interaktywne i dynamiczne frontendy w zasięgu ręki!

Spośród bibliotek JavaScriptu na szczególną uwagę zasługuje React, który służy do tworzenia nowoczesnych, interaktywnych interfejsów użytkownika. W relatywnie prosty sposób pozwala wykorzystywać możliwości współczesnych przeglądarek internetowych. Niestety, pierwsze kroki w obsłudze tej technologii bywają trudne i łatwo się zniechęcić. Właśnie dlatego warto podczas pracy z Reactem mieć przy sobie ten przewodnik!

Dzięki niemu szybko odnajdziesz się w najnowszej, 18. edycji biblioteki React. Książka zawiera informacje o jego najważniejszych narzędziach, podane w przystępny sposób, z naciskiem na wymiar praktyczny. Dowiesz się, jak przebiega tworzenie projektów i z jakich opcji można skorzystać podczas tego procesu. Przejrzyste objaśnienia, zilustrowane starannie opracowanymi przykładami, ułatwią usystematyzowanie wiedzy każdemu zapracowanemu programiście. To atrakcyjna propozycja dla osób, które korzystają z wielu różnych materiałów i chcą mieć wszystkie istotne informacje o bibliotece React zebrane w jednym miejscu.

W książce między innymi:

- budowa nowoczesnych aplikacji internetowych
- komponenty, zdarzenia i warunkowe zarządzanie wyświetlanymi treściami
- warunkowe stosowanie dynamicznych stylów
- zaawansowane techniki zarządzania stanem
- biblioteki Reacta, dobre praktyki i optymalizacja aplikacji

Maximilian Schwarzmüller jest twórcą stron internetowych, szkoleniowcem i autorem kursów internetowych. Uczy, jak korzystać z nowoczesnych platform i bibliotek frontendowych, takich jak Angular i React, jest też ekspertem z dziedziny technologii chmurowych. Posiada liczne certyfikaty związane z platformą AWS, w tym AWS Certified Solutions Architect Professional.