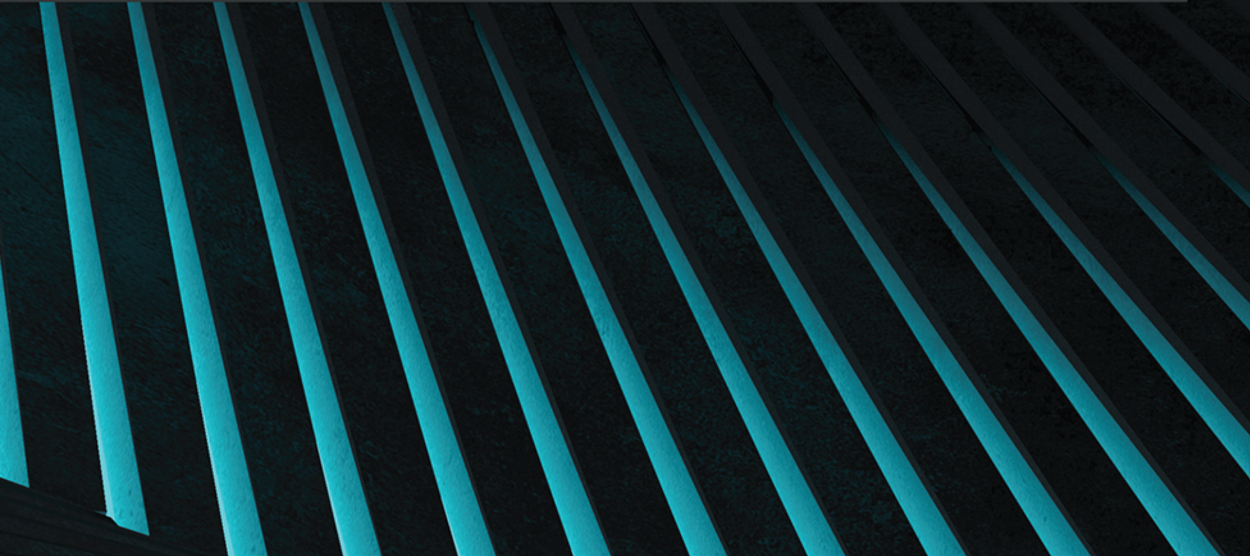
React 17
Wzorce projektowe i najlepsze praktyki

Projektowanie i rozwijanie nowoczesnych aplikacji internetowych

Wydanie III



Carlos Santana Roldán

Helion

Tytuł oryginału:React 17 Design Patterns and Best Practices: Design, build, and deploy production-ready web applications using industry-standard practices, 3rd Edition

Tłumaczenie: Piotr Pilch

ISBN: 978-83-283-8745-4

Copyright © Packt Publishing 2021. First published in the English language under the title 'React 17 Design Patterns and Best Practices - 3rd Edition – (9781800560444)'

Polish edition copyright © 2022 by Helion S.A.
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 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.

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/rea173.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/rea173
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

O autorze 11

O korektorach merytorycznych 13

Przedmowa 15

Część I. Witaj, React! 21

Rozdział 1. Pierwsze kroki z biblioteką React 23

- **Wymagania techniczne** 24
- **Różnica między programowaniem imperatywnym i deklaratywnym** 25
- **Sposób działania elementów biblioteki React** 27
- **Zapomnieć o wszystkim** 28
- **Problem znużenia kodem w JavaScripcie** 30
- **Wprowadzenie do języka TypeScript** 33
  - Elementy języka TypeScript 33
  - Przekształcanie kodu w JavaScripcie w kod w TypeScripcie 34
  - Typy 35
  - Interfejsy 36
- **Podsumowanie** 39

Rozdział 2. Czyszczenie kodu 41

- **Wymagania techniczne** 41
- **Zastosowanie składni JSX** 42
  - Babel 7 43
  - Tworzenie pierwszego elementu 44
  - Elementy modelu DOM i komponenty biblioteki React 44
  - Własności 45

Kup ksi k Pole ksi k

Elementy potomne 45
Różnice względem języka HTML 46
Atrybuty rozwinięcia 50
Literały szablonów 50
Typowe wzorce 50
Określanie stylu kodu **58**
EditorConfig 59
Prettier 59
ESLint 60
Programowanie funkcyjne **64**
Funkcje pierwszoklasowe 65
Czystość 65
Niezmienność 66
Rozwijanie funkcji 67
Kompozycja 67
Programowanie funkcyjne i interfejsy użytkownika 67
Podsumowanie **68**

Część II. Działanie biblioteki React 69

Rozdział 3. React Hooks 71

Wymagania techniczne **71**
Wprowadzanie do dodatku React Hooks **72**
Bez przełomowych zmian 72
Zastosowanie funkcji Hook useState 72
Reguły funkcji Hook 73
Migracja komponentu klasowego pod kątem użycia dodatku React Hooks **74**
Efekty biblioteki React **78**
Funkcja Hook useEffect 78
Warunkowe aktywowanie efektu 78
Funkcje useCallback, useMemo i memo **79**
Zapamiętywanie komponentu za pomocą funkcji memo 82
Zapamiętywanie wartości za pomocą funkcji useMemo 84
Zapamiętywanie definicji funkcji za pomocą funkcji useCallback 87
Zapamiętywanie funkcji przekazanej jako argument funkcji useEffect 92
Funkcja Hook useReducer **95**
Podsumowanie **99**

Rozdział 4. Przegląd popularnych wzorców kompozycji 100

Wymagania techniczne **100**
Zapewnienie komunikacji między komponentami **101**
Zastosowanie własności children 101
Wzorce kontenerowe i prezentacyjne **103**
Komponenty wyższego rzędu **108**
Komponent FunctionAsChild **111**
Podsumowanie **112**

Kup ksi k Pole ksi k

Rozdział 5. Omówienie języka GraphQL na przykładzie rzeczywistego projektu 113

Wymagania techniczne 114
Instalowanie systemu baz danych PostgreSQL 114
Najlepsze narzędzia do zarządzania bazą danych PostgreSQL 115
Tworzenie pliku .env i plików konfiguracyjnych 116
Konfigurowanie pliku .env 116
Tworzenie podstawowego pliku konfiguracyjnego 117
Konfigurowanie serwera Apollo Server 118
Definiowanie typów, zapytań i przekształceń języka GraphQL 120
Zapytania 121
Przekształcenia 121
Scalanie definicji typów 122
Tworzenie resolverów 122
Tworzenie zapytania getUsers 123
Tworzenie zapytania getUserData 124
Tworzenie przekształceń 125
Scalanie resolverów 125
Tworzenie modeli narzędzia Sequelize 126
Nawiązywanie połączenia przez narzędzie Sequelize z bazą danych PostgreSQL 127
Funkcje uwierzytelniania 128
Czym jest standard JSON Web Token? 128
Funkcje JWT 128
Tworzenie funkcji uwierzytelniania 130
Typy i interfejsy 131
Uruchamianie projektu po raz pierwszy 133
Testowanie zapytań i przekształceń języka GraphQL 135
Sprawdzanie poprawności 139
Wykonywanie operacji logowania 142
Tworzenie interfejsowego systemu logowania z wykorzystaniem klienta Apollo 146
Konfigurowanie narzędzia Webpack 5 146
Konfigurowanie interpretera języka TypeScript 149
Konfigurowanie serwera Express 150
Tworzenie konfiguracji interfejsu 151
Tworzenie elementu pośredniczącego użytkownika 153
Tworzenie funkcji tokenów JWT 155
Tworzenie zapytań i przekształceń języka GraphQL 156
Tworzenie kontekstu użytkownika do obsługi operacji logowania i połączonego użytkownika 157
Konfigurowanie klienta Apollo Client 159
Tworzenie ścieżek aplikacji 159
Tworzenie stron 160
Tworzenie komponentów procesu logowania 161
Tworzenie komponentów panelu sterowania 164
Testowanie systemu logowania 165
Podsumowanie 169

Kup ksi k Pole ksi k

Rozdział 6. Zarządzanie danymi 170

Wymagania techniczne 170
Wprowadzenie do składnika React Context API 171
Tworzenie pierwszego kontekstu 171
Opakowywanie komponentów za pomocą dostawcy 173
Korzystanie z kontekstu za pomocą funkcji useContext 173
Wprowadzenie do stosowania składnika React Suspense z wykorzystaniem biblioteki SWR 175
Wprowadzenie do biblioteki SWR 175
Budowanie aplikacji Pokedex! 175
Testowanie składnika React Suspense 181
Podsumowanie 185

Rozdział 7. Tworzenie kodu dla przeglądarki 186

Wymagania techniczne 187
Zastosowanie formularzy 187
Komponenty niekontrolowane 187
Komponenty kontrolowane 191
Obsługa zdarzeń 193
Referencje 196
Implementowanie animacji 197
React Motion 199
Format SVG 200
Podsumowanie 202

Część III. Wydajność, ulepszenia i środowisko produkcyjne 203

Rozdział 8. Zapewnianie ładnego wyglądu komponentów 205

Wymagania techniczne 206
Style CSS w kodzie w JavaScripcie 206
Zastosowanie stylów lokalnych 208
Biblioteka Radium 211
Zastosowanie modułów stylów CSS 215
Webpack 5 215
Przygotowywanie projektu 216
Style CSS o zasięgu lokalnym 221
Moduły Atomic CSS 226
Moduły CSS biblioteki React 228
Zastosowanie biblioteki styled-components 229
Podsumowanie 231

Kup ksi k Pole ksi k

Rozdział 9. Renderowanie serwerowe dla zabawy i zarobku 233

Wymagania techniczne 234
Uniwersalne aplikacje 234
Powody implementowania opcji SSR 235
Implementowanie optymalizacji pod kątem wyszukiwarki internetowej 235
Wspólna baza kodu 236
Lepsza wydajność 237
Nie lekceważ złożoności 237
Tworzenie prostego przykładu użycia opcji SSR 238
Implementowanie pobierania danych 244
Zastosowanie środowiska Next.js do tworzenia aplikacji opartych na bibliotece React 247
Podsumowanie 251

Rozdział 10. Zwiększanie wydajności aplikacji 252

Wymagania techniczne 253
Uzgadnianie 253
Klucze 254
Techniki optymalizacji 255
Narzędzia i biblioteki 257
Niezmienność 257
Dodatki narzędzia Babel 258
Podsumowanie 259

Rozdział 11. Testowanie i debugowanie 260

Wymagania techniczne 260
Korzyści wynikające z testowania 261
Bezproblemowe testowanie kodu w JavaScripcie za pomocą środowiska Jest 262
Testowanie zdarzeń 267
Zastosowanie rozszerzenia React DevTools 269
Stosowanie rozszerzenia Redux DevTools 269
Podsumowanie 270

Rozdział 12. React Router 271

Wymagania techniczne 271
Instalowanie i konfigurowanie biblioteki React Router 272
Tworzenie sekcji 272
Dodawanie parametrów do ścieżek 277
Podsumowanie 284

Rozdział 13. Antywzorce, jakich należy unikać 285

Wymagania techniczne 285
Inicjalizowanie stanu za pomocą właściwości 286
Stosowanie indeksów jako klucza 288
Rozmieszczanie właściwości w elementach modelu DOM 290
Podsumowanie 292

Kup ksi k   Pole ksi k

Rozdział 14. Wdrażanie aplikacji w środowisku produkcyjnym 293

Wymagania techniczne 293
Tworzenie pierwszego Dropletu usługi DigitalOcean 294
Rejestrowanie w usłudze DigitalOcean 294
Tworzenie pierwszego Dropletu 297
Instalowanie środowiska Node.js 299
Konfigurowanie serwisów Git i GitHub 300
Wyłączanie Dropletu 303
Konfigurowanie serwera nginx, narzędzia PM2 i domeny 304
Instalowanie i konfigurowanie serwera nginx 305
Konfigurowanie serwera odwrotnego proxy 306
Dodawanie domeny do Dropletu 307
Implementowanie narzędzia CircleCI do zapewnienia ciągłej integracji 309
Dodawanie klucza SSH do narzędzia CircleCI 310
Konfigurowanie narzędzia CircleCI 312
Tworzenie zmiennych środowiskowych w narzędziu CircleCI 314
Podsumowanie 318

Rozdział 15. Kolejne kroki 319

Wymagania techniczne 319
Uczestniczenie w rozwoju biblioteki React 320
Dystrybucja kodu 321
Znajomość najlepszych praktyk związanych z publikowaniem kodu open source 322
Publikowanie pakietu za pomocą narzędzia npm 324
Podsumowanie 325

Kup ksi k Pole ksi k

1

ROZDZIAŁ

Pierwsze kroki z biblioteką React

Witaj, Czytelniku!

W książce przyjąłem, że wiesz już, czym jest biblioteka React i jakie problemy pozwala rozwiązywać. Być może utworzyłeś z wykorzystaniem tej biblioteki małą lub średniej wielkości aplikację i chcesz zwiększyć zakres umiejętności oraz uzyskać odpowiedzi na wszystkie postawione pytania. Powinieneś wiedzieć, że biblioteka React utrzymywana jest przez projektantów z firmy Facebook oraz setki członków społeczności języka JavaScript. React to jedna z najpopularniejszych bibliotek służących do tworzenia interfejsów użytkownika. Jest dobrze znana ze swojej szybkości, co zawdzięcza inteligentnemu sposobowi współpracy z modelem DOM (ang. *Document Object Model*). React oferuje rozszerzenie JSX, czyli nową składnię do tworzenia znaczników w kodzie pisanym w języku JavaScript, co wymaga zmiany myślenia o separacji zagadnień. Składnia JSX zapewnia wiele ciekawych funkcji, takich jak renderowanie po stronie serwera, co daje możliwość tworzenia uniwersalnych aplikacji.

W pierwszym rozdziale zaprezentuję wybrane podstawowe zagadnienia, których opanowanie jest niezbędne, aby móc efektywnie korzystać z biblioteki React. Zagadnienia te są jednak na tyle proste, że początkujący nie będą mieli problemu z poradzeniem sobie z nimi:

- Różnica między programowaniem imperatywnym i deklaratywnym.
- Komponenty biblioteki React i ich instancje oraz sposób stosowania przez nią elementów do sterowania przepływem w interfejsie użytkownika.
- Wpływ biblioteki React na sposób budowania aplikacji internetowych, wymuszanie zupełnie nowego postrzegania separacji zagadnień oraz powody, dla których biblioteka nie cieszy się popularnością wśród projektantów.
- Przyczyna znużenia projektantów kodem w JavaScripcie, a także rozwiązania umożliwiające uniknięcie najczęstszych błędów popełnianych przez programistów rozpoczynających korzystanie z ekosystemu biblioteki React.
- Wpływ języka TypeScript na obowiązujące zasady gry.

Kup ksi k Pole ksi k

Wymagania techniczne

Aby skorzystać z wiedzy zawartej w książce, musisz dysponować minimalnym doświadczeniem z zakresu używania terminala, w którym zostanie uruchomionych kilka poleceń uniksowych. Ponadto konieczne jest zainstalowanie środowiska Node.js. Dostępne są dwie opcje. Pierwszą z nich jest pobranie tego środowiska bezpośrednio z oficjalnej witryny internetowej (*https://nodejs.org/*). Druga opcja (zalecana) to zainstalowanie narzędzia **NVM** (ang. *Node Version Manager*) dostępnego pod adresem *https://github.com/nvm-sh/nvm*.

Jeśli zdecydujesz się na NVM, możesz zainstalować dowolne wersje środowiska Node.js i przełączać się między nimi za pomocą polecenia `nvm install`:

```
# node identyfikuje alias najnowszej wersji:
nvm install node

# Możliwa jest również instalacja wersji globalnej środowiska (w tym wypadku zostanie zainstalowana
# najnowsza wersja):
nvm install 10
nvm install 9
nvm install 8
nvm install 7
nvm install 6

# Inna opcja to instalacja konkretnej wersji:
nvm install 6.14.3
```

Po zainstalowaniu różnych wersji możesz przełączać się między nimi przy użyciu polecenia `nvm use`:

```
nvm use node # Dotyczy najnowszej wersji
nvm use 10
nvm use 6.14.3
```

I wreszcie masz możliwość określenia domyślnej wersji środowiska `node` za pomocą następujących poleceń:

```
nvm alias default node
nvm alias default 10
nvm alias default 6.14.3
```

Podsumowując, oto lista wymagań niezbędnych w przypadku niniejszego rozdziału:

- **Node.js w wersji 12 lub nowszej** (*https://nodejs.org/*),
- **NVM** (*https://github.com/nvm-sh/nvm*),
- **Visual Studio Code** (*https://code.visualstudio.com/*),
- **TypeScript** (*https://www.npmjs.com/package/typescript*).

Kod zamieszczony w tym rozdziale książki dostępny jest do pobrania pod adresem *https://ftp.helion.pl/przyklady/rea173.zip*.

Kup ksi k Pole ksi k

Różnica między programowaniem imperatywnym i deklaratywnym

Czytając dokumentację biblioteki React lub wpisy na poświęconym jej blogu, bez wątpienia spotkasz się z terminem **deklaratywny**. Jednym z powodów, dla których biblioteka ta ma tak duże możliwości, jest fakt wymuszania modelu programowania deklaratywnego.

A zatem aby opanować bibliotekę React, kluczowe jest zrozumienie, czym jest programowanie deklaratywne, a także jakie są podstawowe różnice między programowaniem imperatywnym i deklaratywnym. W tym wypadku najprościej potraktować programowanie imperatywne jako sposób opisu działania różnych rzeczy, natomiast programowanie deklaratywne jako metodę opisu tego, co zamierza się osiągnąć.

Wizyta w barze w celu zamówienia piwa to realny przykład świata imperatywnego, w którym barmanowi zostałyby normalnie przekazane następujące polecenia:

1. Znajdź szklankę i weź ją z półki.
2. Umieść szklankę pod kurkiem.
3. Ciągnij uchwyt w dół do momentu napełnienia szklanki.
4. Wręcz mi napełnioną szklankę.

W świecie deklaratywnym po prostu stwierdzono by: „Czy mogę prosić o piwo?”.

W wariancie deklaratywnym przyjmuje się, że barman wie już, jak podać piwo. Jest to ważny aspekt zasady działania programowania deklaratywnego.

Zajmijmy się przykładem kodu napisanego w JavaScripcie. W kodzie zostanie utworzona prosta funkcja, która dla danej tablicy łańcuchów z małymi literami zwraca tablicę tych samych łańcuchów w postaci dużych liter:

```
toUpperCase(['foo', 'bar']) // ['FOO', 'BAR']
```

Funkcja imperatywna służąca do rozwiązania tego problemu zostałaby zaimplementowana w następującej postaci:

```
const toUpperCase = input => {
  const output = []
  for (let i = 0; i < input.length; i++) {
    output.push(input[i].toUpperCase())
  }
  return output
}
```

Najpierw tworzona jest pusta tablica, w której zostanie umieszczony wynik. Dalej w ramach pętli funkcja przetwarza wszystkie elementy tablicy wejściowej i wstawia do pustej tablicy wartości z dużymi literami. Na końcu zwracana jest tablica wyjściowa.

Kup ksi k

Pole ksi k

Rozwiązanie deklaratywne miałoby następującą postać:

```
const toUpperCase = input => input.map(value => value.toUpperCase())
```

Elementy tablicy wejściowej są przekazywane funkcji `map`, która zwraca nową tablicę zawierającą wartości z dużymi literami. Godnych uwagi jest kilka znaczących różnic: pierwszy przykład kodu jest mniej elegancki, a zrozumienie go wymaga więcej trudu. Drugi przykład kodu jest bardziej zwięzły i czytelniejszy, co stanowi znaczącą różnicę w przypadku pokaźnych baz kodu, gdzie możliwości jego utrzymania mają kluczowe znaczenie.

Kolejnym aspektem wartym uwagi jest to, że w przykładzie deklaratywnym nie ma potrzeby używania zmiennych ani ciągłego aktualizowania ich wartości w czasie działania kodu. W przypadku programowania deklaratywnego dąży się zwykle do unikania tworzenia i przekształcania stanu.

W ramach ostatniego przykładu sprawdźmy, co oznacza, że biblioteka React ma być deklaratywna. Problem, jaki spróbujemy rozwiązać, to typowe zadanie podczas projektowania aplikacji internetowych. Mowa o tworzeniu przycisku przełączania.

Wyobraź sobie prosty komponent interfejsu użytkownika, taki jak przycisk przełączania. Po kliknięciu go staje się on zielony (aktywny), gdy wcześniej był szary (nieaktywny). Z kolei przycisk przyjmie kolor szary (nieaktywny), jeśli poprzednio był zielony (aktywny).

W wariancie imperatywnym kod miałby następującą postać:

```
const toggleButton = document.querySelector('#toggle')

toogleButton.addEventListener('click', () => {
  if (toggleButton.classList.contains('on')) {
    toggleButton.classList.remove('on')
    toggleButton.classList.add('off')
  } else {
    toggleButton.classList.remove('off')
    toggleButton.classList.add('on')
  }
})
```

Kod jest imperatywny z powodu wszystkich instrukcji niezbędnych do zmiany klas. Dla porównania w wariancie deklaratywnym kod oparty na bibliotece React miałby następującą postać:

```
// Powoduje włączenie przycisku przełączania
<Toggle on />

// Powoduje wyłączenie przycisku przełączania
<Toggle />
```

W przypadku programowania deklaratywnego projektanci opisują jedynie to, co zamierzają osiągnąć. Ponadto nie ma potrzeby wyszczególniania wszystkich kroków, które to umożliwią. To, że biblioteka React oferuje wariant deklaratywny, ułatwia korzystanie z niej, a w konsekwencji kod wynikowy jest prosty. Często prowadzi to do mniejszej liczby błędów i większych możliwości utrzymania kodu.

Kup ksi k Pole ksi k

Z następnego podrozdziału dowiesz się, jak działają elementy biblioteki React. Poza tym bardziej szczegółowo przybliżę, w jaki sposób własności `props` przekazywane są komponentowi biblioteki.

Sposób działania elementów biblioteki React

W książce przyjąłem, że jesteś zaznajomiony z komponentami i ich instancjami. Istnieje jednak inny obiekt, o którym powinno się wiedzieć, aby móc efektywnie korzystać z biblioteki React. Mowa o elemencie.

Niezależnie od tego, czy wywołujesz deklarację `createClass`, rozszerzasz klasę `Component` czy deklarujesz funkcję bezstanową, tworzony jest komponent. React zarządza wszystkimi instancjami komponentów w czasie działania kodu. W danej chwili czasu w pamięci może istnieć więcej niż jedna instancja tego samego komponentu.

Jak wcześniej wspomniałem, biblioteka React bazuje na modelu deklaratywnym, dlatego nie ma potrzeby instruowania jej, jak ma prowadzić interakcję z modelem DOM. Deklarujesz, co ma zostać wyświetlone na ekranie, a biblioteka zajmuje się zapewnieniem tego.

Jak być może miałeś już okazję się o tym przekonać, większość innych bibliotek interfejsów użytkownika działa inaczej. W ich przypadku odpowiedzialność za dbanie o aktualizowanie interfejsu pozostawiana jest projektantowi, który musi ręcznie zarządzać tworzeniem i usuwaniem elementów modelu DOM.

Aby sterować przepływem w interfejsie użytkownika, React używa określonego typu obiektu nazywanego **elementem**, który opisuje, co musi zostać pokazane na ekranie. Te trwałe obiekty są znacznie prostsze w porównaniu z komponentami i ich instancjami, a ponadto zawierają wyłącznie te informacje, które są ściśle wymagane do reprezentowania interfejsu.

Oto przykład kodu elementu:

```
{
  type: Title,
  props: {
    color: 'red',
    children: 'Witaj, Title!'
  }
}
```

Elementy dysponują atrybutem `type`, który jest najważniejszy z atrybutów, oraz kilkoma właściwościami. Istnieje również określona właściwość o nazwie `children`, która jest opcjonalna. Reprezentuje ona bezpośredni element potomny elementu.

Atrybut `type` jest istotny, gdyż informuje bibliotekę React, jak ma zajmować się samym elementem. Jeśli atrybut ten jest łańcuchem, element reprezentuje węzeł modelu DOM. Z kolei gdy atrybut ten to funkcja, element jest komponentem.

Kup ksi k Pole ksi k

W celu reprezentowania drzewa renderowania elementy i komponenty modelu DOM mogą być zagnieżdżane wzajemnie w następujący sposób:

```
{
  type: Title,
  props: {
    color: 'red',
    children: {
      type: 'h1',
      props: {
        children: 'Witaj, H1!'
      }
    }
  }
}
```

Gdy typem elementu jest funkcja, React wywołuje ją, przekazując własności `props` w celu otrzymania elementów bazowych. Biblioteka wykonuje względem wyniku cały czas tę samą operację rekurencyjnie aż do momentu uzyskania drzewa węzłów DOM, które mogą być przez nią renderowane na ekranie. Proces ten, nazywany **rekoncyliacją**, wykorzystywany jest zarówno przez platformę React DOM, jak i React Native do tworzenia ich interfejsów użytkownika.

Biblioteka React stanowi przełom, dlatego na początku jej składnia może wydać się dziwna, ale gdy zrozumiesz zasady jej działania, polubisz ją. Aby to osiągnąć, musisz zapomnieć o wszystkim, co poznałeś dotychczas.

Zapomnieć o wszystkim

Użycie biblioteki React za pierwszym razem wymaga zwykle otwartości umysłu, gdyż zapewnia ona nowy sposób projektowania aplikacji internetowych i mobilnych. W przypadku tej biblioteki podjęto próbę innowacji metody budowania interfejsów użytkownika w oparciu o ścieżkę postępowania, która narusza większość dobrze znanych, sprawdzonych rozwiązań.

W ostatnich dwóch dekadach okazało się, że separacja zagadnień jest ważną kwestią. Traktowano to jako oddzielanie logiki od szablonów. Naszym celem zawsze było umieszczanie kodu napisanego w JavaScripcie i HTML-u w różnych plikach. Aby ułatwić to projektantom, utworzono różne rozwiązania oferujące szablony.

Problem polega na tym, że przeważnie tego rodzaju separacja jest tylko iluzją, a prawda jest taka, że kod w JavaScripcie i kod w HTML-u są silnie ze sobą sprzężone niezależnie od tego, gdzie się znajdują.

Przeanalizujmy przykład szablonu:

```
{{#items}}
  {{#first}}
    <li><strong>{{name}}</strong></li>
  {{/first}}
  {{#link}}
```


Kup ksi k Pole ksi k

```
    <li><a href="{{url}}">{{name}}</a></li>
  {{/link}}
{{/items}}
```

Powyższy fragment kodu pochodzi z witryny internetowej Mustache, czyli jednego z najpopularniejszych systemów szablonowych.

W pierwszym wierszu kodu system Mustache instruowany jest do wykonania pętli dla kolekcji elementów. W obrębie pętli zastosowano logikę warunkową w celu sprawdzenia, czy istnieją właściwości `#first` i `#link`. Zależnie od ich wartości renderowana jest inna część kodu w HTML-u. Zmienne umieszczono w nawiasach klamrowych.

Jeśli aplikacja musi jedynie wyświetlić wartości zmiennych, biblioteka szablonów mogłaby stanowić dobre rozwiązanie, ale gdy w grę zaczyna wchodzić użycie złożonych struktur danych, sprawy ulegają zmianie. Systemy szablonowe i ich język **DSL** (ang. *Domain-Specific Language*) oferują podzbiór elementów, a ponadto próbują zapewnić funkcjonalności prawdziwego języka programowania bez osiągania takiego samego poziomu kompletności. Jak pokazałem w przykładach, w kwestii wyświetlania informacji szablony w dużym stopniu zależą od modeli otrzymywanych z warstwy logiki.

Z kolei kod w JavaScripcie prowadzi interakcję z elementami modelu DOM renderowanymi przez szablony w celu dokonania aktualizacji interfejsu użytkownika nawet wtedy, gdy są one ładowane z osobnych plików. Ten sam problem dotyczy stylów. Są one definiowane w innym pliku, ale wskazywane w szablonach. Selektory CSS przestrzegają struktury znaczników, dlatego prawie niemożliwa jest zmiana jednego selektora bez naruszenia innego (jest to określane przez definicję **sprzężenia**). Z tego właśnie powodu klasyczna separacja zagadnień stała się bardziej separacją technologii, co oczywiście nie jest niczym złym, lecz nie zapewnia rozwiązania żadnych faktycznych problemów.

W przypadku biblioteki React podejmowana jest próba wykonania kroku naprzód przez umieszczenie szablonów tam, gdzie ich miejsce, czyli obok logiki. Wynika to z tego, że biblioteka ta sugeruje organizację aplikacji opartą na tworzeniu niewielkich klocków nazywanych komponentami. Środowisko nie powinno instruować, w jaki sposób separować zagadnienia, ponieważ każda aplikacja ma swoje własne zagadnienia, a tylko projektanci powinni decydować o sposobie wyznaczenia granic w tworzonych aplikacjach.

Rozwiązanie oparte na komponentach w znaczący sposób zmienia sposób tworzenia aplikacji internetowych. Z tego powodu klasyczne pojmowanie separacji zagadnień jest stopniowo zastępowane przez znacznie bardziej nowoczesną strukturę. Model wymuszany przez bibliotekę React nie jest nowy ani nie został opracowany przez jej twórców. Niemniej jednak biblioteka ta ma udział w powszechnym rozpropagowaniu tego rozwiązania, a co najważniejsze, zostało ono spopularyzowane w taki sposób, że jest łatwiejsze do zrozumienia przez projektantów mających różny poziom umiejętności.

Kod renderowania komponentu Reacta wygląda następująco:

```
return (
  <button style={{ color: 'red' }} onClick={this.handleClick}>
    Kliknij mnie!
  </button>
)
```


Kup ksi k   Pole ksi k

Wszyscy zgodzimy się z tym, że początkowo wydaje się to trochę dziwne, ale wynika to właśnie stąd, że nie przywykliśmy do tego rodzaju składni. Od razu po zaznajomieniu się z nią i uświadomieniu sobie, jakie ma ona możliwości, zdamy sobie sprawę z jej potencjału. Zastosowanie kodu w JavaScripcie zarówno w przypadku logiki, jak i tworzenia szablonów nie tylko ułatwia separację zagadnień w lepszy sposób, ale też stwarza większy zakres możliwości i środków wyrazu. Właśnie to jest niezbędne do budowania złożonych interfejsów użytkownika.

Z tego powodu nawet wtedy, gdy z początku idea łączenia kodu w JavaScripcie i kodu w HTML-u brzmi dziwnie, ważne jest, by dać szansę bibliotece React. W przypadku nowej technologii najlepiej zacząć od wypróbowania jej w niewielkim projekcie pobocznym i przekonania się, jak się sprawdza. Ogólnie rzecz ujmując, właściwym podejściem zawsze jest zapomnienie o dotychczasowym stanie wiedzy i zmiana sposobu myślenia, jeśli długofalowe korzyści są tego warte.

Istnieje inne zagadnienie, które jest naprawdę kontrowersyjne i trudne do zaakceptowania. Inżynierowie tworzący bibliotekę React starają się wprowadzić je do społeczności. Zagadnienie to dotyczy tego, że w obrębie komponentu przenoszona jest również logika związana ze stylami. Końcowym celem jest hermetyzacja każdej osobnej technologii używanej do tworzenia komponentów i separacja zagadnień zgodnie z ich domeną i funkcjonalnością.

Oto przykład obiektu stylu wykorzystany z dokumentacji biblioteki React:

```
const divStyle = {
  color: 'white',
  backgroundImage: `url(${imgUrl})`,
  WebkitTransition: 'all', // Zwróć tutaj uwagę na dużą literę W
  msTransition: 'all' // ms to jedyny przedrostek dostawcy zapisany małymi literami
}
ReactDOM.render(<div style={divStyle}>Witaj, świecie!</div>, mountNode)
```

Taki zestaw rozwiązań, w przypadku których projektanci używają kodu w JavaScripcie do tworzenia własnych stylów, znany jest jako `#CSSinJS`. Szerzej będzie o nim mowa w rozdziale 8.

Z kolejnego podrozdziału dowiesz się, jak uniknąć znużenia językiem JavaScript, które powodowane jest dużą ilością konfiguracji niezbędnej do uruchomienia aplikacji opartej na bibliotece React (dotyczy to głównie narzędzia tworzącego pakiety).

Problem znużenia kodem w JavaScripcie

Dominuje opinia, że biblioteka React składa się z ogromnej liczby technologii i narzędzi, a jeśli chcesz z nich skorzystać, musisz sięgać po menedżery pakietów, transkompilatory, programy ładujące pakiety oraz nieskończoną liczbę różnych bibliotek. Takie postrzeganie jest tak powszechne i przekazywane przez różne osoby, że zostało wyraźnie zdefiniowane i określone mianem **znużenia kodem w JavaScripcie**.

Nie trudno zrozumieć przyczyny takiego stanu rzeczy. Wszystkie repozytoria i biblioteki w ekosystemie biblioteki React są tworzone z wykorzystaniem zupełnie nowych technologii, najnowszej wersji języka JavaScript oraz najbardziej zaawansowanych technik i modeli.

Kup ksi k Pole ksi k

Co więcej, w serwisie GitHub istnieje ogromna liczba szablonów standardowego kodu biblioteki React, z których każdy obejmuje dziesiątki zależności w celu zaoferowania rozwiązań wszelkich problemów. Można wprost pomyśleć, że wszystkie te narzędzia są wymagane do rozpoczęcia pracy z biblioteką React. Jest to jednak dalekie od prawdy. Pomimo takiego powszechnego sposobu myślenia React to naprawdę drobna biblioteka, która może być stosowana w obrębie dowolnej strony (lub nawet w środowisku JSFiddle) w taki sam sposób, w jaki każdy korzystał z biblioteki jQuery lub Backbone. Wystarczy jedynie umieścić skrypt na stronie przed domykającym elementem `body`.

Istnieją dwa skrypty, ponieważ bibliotekę React podzielono na dwa pakiety:

- `react` — implementuje podstawowe elementy biblioteki.
- `react-dom` — zawiera wszystkie elementy powiązane z przeglądarką.

Wynika to stąd, że pakiet podstawowy służy do obsługi różnych platform docelowych, takich jak React DOM w przeglądarkach i React Native w przypadku urządzeń przenośnych. Uruchamianie aplikacji opartych na Reakcie w obrębie pojedynczej strony napisanej w HTML-u nie wymaga żadnego menedżera pakietów ani złożonej operacji. Możesz po prostu pobrać pakiet dystrybucji i samemu go udostępnić (lub skorzystać z witryny o adresie *https://unpkg.com/*). W ciągu kilku minut będziesz gotowy do rozpoczęcia używania biblioteki React i jej składników.

Oto adresy URL, które zostaną dołączone do kodu w HTML-u w celu umożliwienia rozpoczęcia stosowania biblioteki React:

- *https://unpkg.com/react@17.0.1/umd/react.production.min.js*
- *https://unpkg.com/react-dom@17.0.1/umd/react-dom.production.min.js*

Jeśli zostanie dodana tylko podstawowa wersja biblioteki React, nie można używać składni JSX, gdyż nie jest to standardowy język obsługiwany przez przeglądarkę. W gruncie rzeczy chodzi o to, aby zacząć od zupełnie minimalnego zestawu składników i dodawać kolejne funkcjonalności od razu, gdy okażą się niezbędne. W wypadku prostego interfejsu użytkownika można by użyć jedynie składnika `createElement` (`_jsx` w bibliotece React 17). Tylko wtedy, gdy zaczniemy budować coś bardziej złożonego, można uwzględnić transkompilator w celu włączenia obsługi składni JSX i przekształcenia jej w kod w JavaScripcie. Gdy tylko aplikacja zwiększy się trochę bardziej, może się okazać konieczny router do obsługi różnych stron i widoków. To też może zostać dołączone.

W pewnym momencie może być wskazane załadowanie danych z wybranych punktów końcowych API. Jeśli aplikacja cały czas się powiększa, zostanie osiągnięty punkt, w którym niezbędne będą zależności zewnętrzne do zapewnienia abstrakcji złożonych operacji. Dokładnie tylko w takiej sytuacji powinno się wprowadzić menedżer pakietów. Później przyjdzie pora na podzielenie aplikacji na osobne moduły i uporządkowanie plików w odpowiedni sposób. Na tym etapie powinno się zacząć brać pod uwagę zastosowanie narzędzia tworzącego pakiety modułów.

Postępowanie zgodnie z tym prostym schematem nie spowoduje znużenia. Rozpoczęcie od szablonu zawierającego 100 zależności i dziesiątki pakietów narzędzia npm, o których nic nie wiemy, to najlepszy sposób na pogubienie się. Godne uwagi jest to, że każde zajęcie związane

Kup ksi k Pole ksi k

z programowaniem (a w szczególności z inżynierią interfejsów) wymaga ciągłej nauki. Naturą technologii internetowych jest rozwijanie się w zawrotnym tempie i zmienianie odpowiednio do potrzeby zarówno użytkowników, jak i projektantów. Właśnie temu podlega środowisko biblioteki React od samego początku, co sprawia, że jest ono tak ekscytujące.

Wraz ze zdobywaniem doświadczenia z zakresu technologii internetowych uczymy się, że nie można wszystkiego opanować. Z tego powodu należy znaleźć właściwy sposób ciągłego aktualizowania własnej wiedzy, aby uniknąć znużenia. Mamy możliwość podążania za wszystkimi nowymi trendami bez sięgania po nowe biblioteki wyłącznie dla zasady, chyba że dysponujemy czasem na realizowanie pobocznego projektu.

Zadziwiające jest to, jak w świecie języka JavaScript od razu po ogłoszeniu specyfikacji lub jej wersji roboczej ktoś w społeczności implementuje ją jako dodatek w postaci transkompilatora lub „łatki” zapewniającej zgodność wstecz, a dzięki temu wszyscy inni mają możliwość eksperymentowania z tym, gdy dostawcy przeglądarek dochodzą do porozumienia i zaczynają zapewniać obsługę specyfikacji.

Jest to coś, co sprawia, że język JavaScript i przeglądarka tworzą całkowicie odmienne środowisko w porównaniu z dowolnym innym językiem lub platformą. Mankamentem jest tutaj to, że wszystko szybko się zmienia. Jest to jednak tylko kwestia znalezienia odpowiedniej równowagi między postawieniem na nowe technologie i zapewnieniem sobie bezpiecznej sytuacji.

W każdym razie projektanci z firmy Facebook dużą wagę przykładają do kwestii **komfortu pracy projektantów** i uważnie analizują uwagi społeczności. A zatem nawet pomimo tego, że nie jest prawdą, iż stosowanie biblioteki React wymaga poznania setek różnych narzędzi, projektanci z firmy Facebook uświadomili sobie, że ludzie odczuwali znużenie, dlatego udostępnili narzędzie CLI, które niesamowicie ułatwia przygotowanie za pomocą szablonów i uruchomienie prawdziwej aplikacji opartej na bibliotece React.

Jedynym wymaganiem jest zastosowanie środowiska złożonego z Node.js i narzędzia npm oraz zainstalowanie narzędzia CLI globalnie w następujący sposób:

```
npm install -g create-react-app
```

Po zainstalowaniu pliku wykonywalnego możesz za jego pomocą utworzyć aplikację, przekazując nazwę folderu:

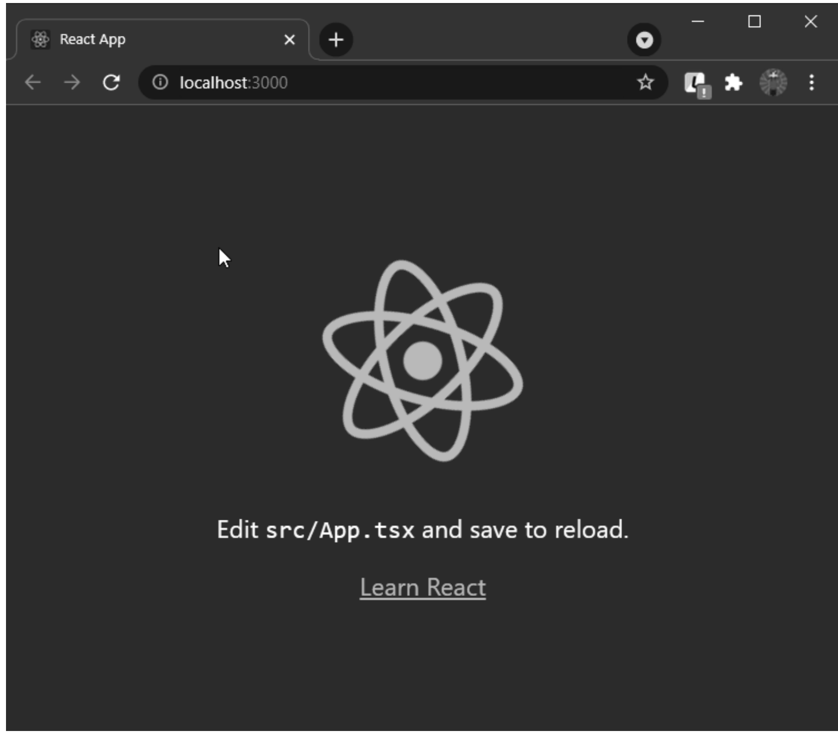
```
create-react-app witaj-swiecie --template typescript
```

Na koniec przy użyciu polecenia `cd witaj-swiecie` przechodzisz do folderu aplikacji i po prostu wykonujesz następujące polecenie:

```
npm start
```

W magiczny sposób przykładowa aplikacja działa z pojedynczą zależnością, ale z uwzględnieniem wszystkich elementów niezbędnych do zbudowania kompletnej aplikacji z użyciem Reacta wykorzystującej najbardziej zaawansowane rozwiązania. Na poniższym rysunku pokazałem domyślną stronę aplikacji utworzonej za pomocą polecenia `create-react-app`.

Kup ksi k Pole ksi k



W zasadzie jest to Twoja pierwsza aplikacja wykorzystująca bibliotekę React.

Wprowadzenie do języka TypeScript

TypeScript to nadzbiór z typowaniem języka JavaScript, który kompilowany jest do postaci kodu w JavaScripcie. Oznacza to, że język **TypeScript** jest językiem **JavaScript** z kilkoma dodatkowymi elementami. TypeScript to język otwartoźródłowy, stworzony przez Andersa Hejlsberga (projektanta języka C#) w firmie Microsoft.

Dowiedzmy się, jakie są elementy języka TypeScript, a także w jaki sposób przekształcić kod w JavaScripcie w kod w TypeScripcie.

Elementy języka TypeScript

W tym podrozdziale postaram się przedstawić zestawienie najważniejszych elementów. Oto one:

- **Język TypeScript to język JavaScript**. Dowolny kod napisany w JavaScripcie będzie współdziałać z kodem w TypeScripcie. Oznacza to, że jeśli wiesz już, jak posługiwać się językiem JavaScript, zasadniczo dysponujesz wszystkim, co jest niezbędne do tworzenia kodu w TypeScripcie. Musisz jedynie dowiedzieć się,

Kup ksi k Pole ksi k

w jaki sposób dodać typy do swojego kodu. Cały kod w TypeScripcie transformowany jest ostatecznie do postaci kodu w JavaScripcie.

- **Język JavaScript to język TypeScript**. Oznacza to po prostu, że możesz zmienić rozszerzenie dowolnego poprawnego pliku *.js* na rozszerzenie *.ts* i nie spowoduje to żadnych problemów.
- **Sprawdzanie błędów**. W przypadku języka TypeScript ma miejsce kompilowanie kodu i sprawdzanie pod kątem błędów, co znacząco ułatwia zidentyfikowanie błędów przed uruchomieniem kodu.
- **Typowanie silne**. Domyślnie JavaScript nie zapewnia typowania silnego. W przypadku TypeScriptu masz możliwość dodania typów do wszystkich zmiennych i funkcji, a ponadto możesz nawet określić typy wartości zwracanych.
- **Obsługa programowania obiektowego**. Obsługiwane są takie zagadnienia jak klasy, interfejsy, dziedziczenie itp.

Przekształcanie kodu w JavaScripcie w kod w TypeScripcie

W niniejszym podrozdziale pokażę, jak przekształcić kod w JavaScripcie w kod w TypeScripcie.

Załóżmy, że musisz sprawdzić, czy słowo jest palindromem. Napisany w JavaScripcie kod odpowiedniego algorytmu ma następującą postać:

```
function isPalindrome(word) {
  const lowerCaseWord = word.toLowerCase()
  const reversedWord = lowerCaseWord.split('').reverse().join('')

  return lowerCaseWord === reversedWord
}
```

Plik z tym kodem może mieć nazwę *palindrom.ts*.

Jak widać, uzyskiwana jest zmienna `word` typu `string`, a ponadto zwracana jest wartość typu `boolean`. Jak zatem kod zostanie przekształcony w kod w TypeScripcie?

```
function isPalindrome(word: string): boolean {
  const lowerCaseWord = word.toLowerCase()
  const reversedWord = lowerCaseWord.split('').reverse().join('')

  return lowerCaseWord === reversedWord
}
```

Prawdopodobnie pomyślisz: „Wspaniale, właśnie określiłem typ `string` dla zmiennej `word` oraz typ `boolean` dla wartości zwracanej funkcji. Ale co teraz?”.

Jeżeli spróbujesz uruchomić funkcję z wartością, która nie jest łańcuchem, uzyskasz błąd języka TypeScript:

Kup ksi k   Pole ksi k

```
console.log(isPalindrome('Radar')) // Prawda
console.log(isPalindrome('Anna')) // Prawda
console.log(isPalindrome('Robert')) // Fałsz
console.log(isPalindrome(101)) // Błąd języka TypeScript
console.log(isPalindrome(true)) // Błąd języka TypeScript
console.log(isPalindrome(false)) // Błąd języka TypeScript
```

A zatem jeśli spróbujesz przekazać funkcji liczbę, zostanie wygenerowany błąd widoczny na poniższym rysunku.

```
 8   console.log(isPalindrome('Radar')) // prawda
 9   console.log(isPalindrome('Anna')) // prawda
10   console.log(isPalindrome('Robert')) // fałsz
11   console.log(isPalindrome(101)) // błąd języka TypeScript
12   console.log(isPalindrome(true)) // błąd języka TypeScript
13   console.log(isPalindrome(false)) // błąd języka TypeScript

PROBLEMS 3   OUTPUT   DEBUG CONSOLE   TERMINAL   Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS palindrom.ts F:\ 3
  Argument of type 'number' is not assignable to parameter of type 'string'. ts(2345) [11, 26]
```

Z tego właśnie powodu język TypeScript okazuje się bardzo przydatny, gdyż wymusi w przypadku kodu większą ścisłość i jawność.

Typy

W ostatnim przykładzie pokazałem, jak określić niektóre typy podstawowe dla parametru funkcji i jej wartości zwracanej. Prawdopodobnie zastanawiasz się jednak, w jaki sposób możesz opisać obiekt lub tablicę z zastosowaniem większej ilości szczegółów. **Typy** mogą ułatwić lepsze opisanie obiektów lub tablic. Dla przykładu załóżmy, że zamierzasz opisać typ `User` w celu zapisania informacji w bazie danych:

```
type User = {
  username: string
  email: string
  name: string
  age: number
  website: string
  active: boolean
}

const user: User = {
  username: 'jnowak',
  email: 'jnowak@abc.com',
  name: 'Jan Nowak',
  age: 33,
  website: 'http://www.js.edukacja',
```

```
  active: true
}

// Załóżmy, że dane te zostaną wstawione za pomocą środowiska Sequelize...
models.User.create({ ...user })
```

Jeżeli zapomnisz dodać jednego z węzłów lub umieścisz niepoprawną wartość w jednym z nich, zostanie wygenerowany komunikat o błędzie widoczny na poniższym rysunku.

```
1   type User = {
2     username: string
3     email: string
4     name: string
5     age: number
6     website: string
7     active: boolean
8   }
9
10  const user: User = {
11    username: 'jnowak',
12    email: 'jnowak@abc.com',
13    name: 'Jan Nowak',
14    website: 'http://www.js.edukacja',
15    active: true
16  }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, **/*.ts, !**/node_modules/**)

TS plik.ts C:\Appz 1

Property 'age' is missing in type '{ username: string; email: string; name: string; website: string; active: true; }' but required in type 'User'. ts(2741) [10, 7]

plik.ts[5, 3]: 'age' is declared here.

Jeśli wymagane są opcjonalne węzły, zawsze możesz wstawić znak ? obok nazwy węzła w sposób zaprezentowany w następującym bloku kodu:

```
type User = {
  username: string
  email: string
  name: string
  age?: number
  website: string
  active: boolean
}
```

> Choć możesz podać jako typ dowolną nazwę, dobrą praktyką wartą przestrzegania jest dodawanie przedrostka `T`. Oznacza to na przykład, że nazwa typu `User` przyjmie postać `TUser`. W ten sposób możesz szybko stwierdzić, że masz do czynienia z typem, a ponadto nie pomyślisz błędnie, że jest to klasa lub komponent biblioteki React.

Interfejsy

Interfejsy w dużym stopniu przypominają typy. Czasami projektanci nie wiedzą, czym one się różnią. Tak jak typy, interfejsy mogą posłużyć do opisania postaci sygnatury obiektu lub funkcji, ale używana jest odmienna składnia:

Kup ksi k   Pole ksi k

```
interface User {
  username: string
  email: string
  name: string
  age?: number
  website: string
  active: boolean
}
```

> Masz możliwość podania jako interfejsu dowolnej nazwy, ale dobrą praktyką wartą przestrzegania jest dodawanie przedrostka `I`. Oznacza to na przykład, że nazwa interfejsu `User` przyjmie postać `IUser`. Dzięki temu możesz szybko stwierdzić, że masz do czynienia z interfejsem, a ponadto nie pomyślisz błędnie, że jest to klasa lub komponent biblioteki React.

Interfejs może być też rozszerzany, implementowany i scalany.

Rozszerzanie

Interfejs lub typ zapewnia również możliwość rozszerzania, ale różna będzie składnia, co widać w następującym bloku kodu:

```
// Rozszerzanie interfejsu
interface IWork {
  company: string
  position: string
}

interface IPerson extends IWork {
  name: string
  age: number
}

// Rozszerzanie typu
type TWork = {
  company: string
  position: string
}

type TPerson = TWork & {
  name: string
  age: number
}

// Rozszerzanie interfejsu do postaci typu
interface IWork {
  company: string
  position: string
}

type TPerson = IWork & {
  name: string
  age: number
}
```


Kup ksi k	Pole ksi k

Jak widać, za pomocą znaku & możesz rozszerzyć typ, natomiast rozszerzenie interfejsu umożliwia słowo kluczowe `extends`.

Implementowanie

Klasa może implementować alias typu lub interfejsu w dokładnie taki sam sposób. Nie może jednak zaimplementować (lub rozszerzyć) aliasu typu identyfikującego typ w postaci unii. Oto przykład:

```
// Implementowanie interfejsu
interface IWork {
  company: string
  position: string
}

class Person implements IWork {
  name: 'Jan'
  age: 33
}

// Implementowanie typu
type TWork = {
  company: string
  position: string
}

class Person2 implements TWork {
  name: 'Krystyna'
  age: 32
}

// Nie można zaimplementować typu w postaci unii
type TWork2 = { company: string; position: string } | { name: string; age: number }

class Person3 implements TWork2 {
  company: 'Google'
  position: 'Starszy inżynier oprogramowania'
}
```

Jeśli użyjesz tego kodu, w edytorze zostanie wygenerowany komunikat o błędzie widoczny na poniższym rysunku.

```
1  // Nie można zaimplementować typu w postaci unii
2  type TWork2 = { company: string; position: string } | { name: string; age: number }
3
4  class Person3 implements TWork2 {
5    company: 'Google'
6    position: 'Starszy inżynier oprogramowania'
7  }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, **/*.ts, !**/node_modules/**)

TS plik.ts F:\ 1

ⓧ A class can only implement an object type or intersection of object types with statically known members. ts(2422) [4, 26]

Kup ksi k Pole ksi k

Jak widać, nie ma możliwości zaimplementowania typu w postaci unii.

Scalanie deklaracji

W przeciwieństwie do typu interfejs może być definiowany wielokrotnie i będzie traktowany jako pojedynczy interfejs (wszystkie deklaracje będą scalane). Zaprezentowałem to w następującym bloku kodu:

```
interface IUser {
  username: string
  email: string
  name: string
  age?: number
  website: string
  active: boolean
}

interface IUser {
  country: string
}

const user: IUser = {
  username: 'jnowak',
  email: 'jnowak@abc.com',
  name: 'Jan Nowak',
  country: 'Polska',
  age: 33,
  website: 'http://www.js.edukacja',
  active: true
}
```

Jest to bardzo przydatne, gdy okaże się konieczne rozszerzenie interfejsów w różnych wariantach przez samo ponowne zdefiniowanie tego samego interfejsu.

Podsumowanie

W pierwszym rozdziale omówiłem podstawowe zagadnienia, które będą bardzo ważne w trakcie lektury reszty książki. Ponadto są one kluczowe przy codziennym korzystaniu z biblioteki React. Wiesz już, jak pisać kod deklaratywny. W pełni rozumiesz różnicę między tworzonymi komponentami i elementami używanymi przez bibliotekę React do wyświetlania instancji komponentów na ekranie.

Przedstawiłem powody, dla których kod logiki i szablony umieszczane są w jednym miejscu, a także wyjaśniłem, dlaczego ta niepopularna decyzja zapewniła wielki sukces bibliotece React. Znasz już również powody, dla których powszechne jest poczucie znużenia ekosystemem języka JavaScript, i wiesz, jak uniknąć tego rodzaju problemów dzięki korzystaniu z rozwiązania iteracyjnego.

Kup ksi k Pole ksi k

Pokazałem, w jaki sposób używać języka TypeScript do tworzenia niektórych typów podstawowych i interfejsów. Przybliżyłem, czym jest nowe narzędzie CLI z poleceniem `create-react-app`. Możesz już zacząć pisać prawdziwy kod.

Z następnego rozdziału dowiesz się, jak korzystać z kodu ze składnią JSX/TSX i stosować bardzo przydatne konfiguracje w celu poprawienia stylu kodu.

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: nowy wymiar efektywnego programowania aplikacji!

JavaScript pozwala na tworzenie znakomitego kodu, ale wymaga od programisty sporych umiejętności. Szczególnym uznaniem cieszy się React — popularna biblioteka *open source* służąca do tworzenia dynamicznych aplikacji na bazie niewielkich komponentów wielokrotnego użytku. React jest narzędziem, które w stosunkowo prosty sposób pozwala w pełni skorzystać z możliwości nowoczesnych przeglądarek i urządzeń mobilnych.

To książka przeznaczona dla średnio zaawansowanych użytkowników biblioteki React, którzy chcą tworzyć elastyczniejsze i łatwiejsze w utrzymaniu aplikacje. Zaprezentowano, jak należy budować komponenty możliwe do wielokrotnego użycia, jak projektować strukturę aplikacji, a także omówiono zasady tworzenia poprawnych formularzy. Opisano też procesy definiowania stylów dla komponentów biblioteki React, jak również ich optymalizowania w celu przyspieszenia aplikacji i zwiększenia ich responsywności. Nie zabrakło szczegółowego omówienia technik tworzenia zestawów efektywnych testów, ponadto znalazły się tutaj informacje na temat korzystania z takich narzędzi jak służący do testowania Enzyme, React Router czy ułatwiający ciągłą integrację CircleCI.

W książce między innymi:

- składniki biblioteki React, w tym Context API i React hooks
- tworzenie i optymalizacja komponentów
- stosowanie języka GraphQL w projektach
- renderowanie po stronie serwera
- tworzenie wydajnego zestawu testów

Carlos Santana Roldán od kilkunastu lat projektuje aplikacje internetowe, obecnie pracuje jako główny inżynier oprogramowania w firmie Snapchat. Jest twórcą serwisu *http://js.education/*, w którym publikuje materiały do nauki nowoczesnych technologii internetowych, takich jak React, Node.js, JavaScript i TypeScript.

ISBN 978-83-283-8745-4

Cena: 79,00 zł