O'REILLY®

Wszechstronny JavaScript

Technologie: GraphQL, React, React Native i Electron

Helion

Adam D. Scott

Tytuł oryginału: JavaScript Everywhere: Building Cross-Platform Applications with GraphQL, React, React Native, and Electron

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-7114-9

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

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

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

Printed in Poland.

Spis treści

ROZDZIAŁ 12.

Budowa klienta internetowego za pomocą React

Początkową ideą stojącą za hipertekstem było połączenie powiązanych ze sobą dokumentów: jeżeli dokument A odwoływał się do dokumentu B, wówczas kliknięcie pewnego elementu pozwalało na łatwe poruszanie się między nimi. W 1989 r. inżynier oprogramowania w CERN-ie Tim Berners-Lee opracował ideę połączenia hipertekstu z komputerami w sieci, co pozwalało na łatwe odwoływanie się do dokumentów niezależnie od miejsca położenia zawierających je komputerów. Każdy element danych w internecie — zdjęcie kota, artykuł w gazecie, wiadomość serwisu społecznościowego, strumieniowany klip wideo, witryna pomagająca w znalezieniu pracy, witryna zawierająca recenzje lokali gastronomicznych itd. — pozostaje zgodny z ideą globalnie połączonych dokumentów.

Sednem sieci jest medium pozwalające na łączenie dokumentów ze sobą. Każda strona internetowa składa się ze znaczników HTML-a, jest generowana przez przeglądarkę WWW, zawiera arkusze stylów CSS określające jej wygląd i używa JavaScriptu do usprawnienia sposobu jej działania. Obecnie korzystamy z technologii internetowych do tworzenia niemalże wszystkiego w sieci: od osobistych blogów, poprzez niewielkie broszury aż po skomplikowane i interaktywne aplikacje. Wszechobecną zaletą sieci jest to, że oferuje uniwersalny dostęp do swoich zasobów. Użytkownik potrzebuje jedynie przeglądarki WWW i urządzenia zapewniającego połączenie z internetem, co daje mu niezbędne środowisko pracy.

Co będziemy budować?

W kolejnych rozdziałach zajmiemy się utworzeniem klienta internetowego dla naszej społecznościowej aplikacji notatek Notedly. Użytkownik będzie miał możliwość utworzenia konta i zalogowania się do niego, tworzenia notatek w formacie Markdown, edytowania notatek, wyświetlania kanałów notatek innych użytkowników, a także oznaczania jako „ulubionych” notatek dodanych przez innych użytkowników. Do realizacji tego zadania wykorzystamy opracowane wcześniej API GraphQL.

W naszej aplikacji internetowej:

- użytkownik będzie miał możliwość tworzenia, odczytywania, uaktualniania i usuwania własnych notatek;

- użytkownik będzie mógł wyświetlać kanały notatek utworzonych przez innych użytkowników, odczytywać pojedyncze notatki innych użytkowników, choć bez prawa do uaktualniania lub usuwania;
- użytkownik będzie mógł utworzyć konto, zalogować się do niego i wylogować się z niego;
- użytkownik będzie mógł pobierać informacje o profilu, a także informacje dotyczące publicznych profili innych użytkowników;
- użytkownik będzie mógł oznaczać jako ulubione notatki innych użytkowników, a także pobierać listę ulubionych notatek innych użytkowników.

Implementacja tych funkcjonalności wymaga sporo pracy, ale zadanie zostanie podzielone na kilka mniejszych, którymi będziemy się zajmować w tej części książki. Gdy już opanujesz tworzenie aplikacji React zawierającej wszystkie wymienione funkcje, będziesz mógł wykorzystać poznane narzędzia i techniki do zbudowania dowolnej aplikacji internetowej.

Jak będziemy budować naszą aplikację?

Jak już prawdopodobnie się domyśliłeś, podczas budowy aplikacji będziemy korzystać z React jako działającej po stronie klienta biblioteki JavaScriptu. Dane będą pobierane z opracowanego wcześniej API GraphQL. Aby pomóc w wykonywaniu zapytań, mutacji i buforowaniu danych, użyjemy klienta Apollo (*https://www.apollographql.com/docs/react/*), czyli kolekcji narzędzi typu *open source* przeznaczonych do pracy z GraphQL. Wprawdzie będziemy korzystać z wersji React biblioteki, ale zespół w firmie Apollo opracował również wersje przeznaczone dla frameworków Angular, Vue, Scala.js, Native iOS i Native Android.

Inne działające po stronie klienta biblioteki GraphQL

W tej książce zdecydowałem się na użycie Apollo, ale warto w tym miejscu dodać, że nie jest to jedyna działająca po stronie klienta biblioteka dla GraphQL. Klienty Relay firmy Facebook (*https://relay.dev/*) i urql firmy Formiddable (*https://formidable.com/open-source/urql/*) to dwie popularne alternatywy.

Do łączenia kodu w paczkę wykorzystamy Parcel (*https://parceljs.org/*). Narzędzie do łączenia kodu w paczkę pozwala na tworzenie kodu JavaScriptu, który niekoniecznie będzie obsługiwany przez przeglądarki WWW (np.: nowszych funkcji języka, modułów kodu, minimalizacji), i przygotowanie go do użycia w środowisku przeglądarki WWW. Parcel to niewymagająca konfiguracji alternatywa dla narzędzi kompilowania aplikacji, takich jak Webpack (*https://webpack.js.org/*). Oferuje użyteczny zestaw funkcji, np. dzielenie kodu i automatyczne uaktualnienie przeglądarki WWW podczas pracy nad aplikacją (z ang. *hot module replacement*), i nie wymaga przy tym konfigurowania zestawu narzędzi. Jak dowiedziałeś się z poprzedniego rozdziału, `create-react-app` (*https://create-react-app.dev/*) to narzędzie również niewymagające konfiguracji początkowej i używające w tle Webpacka. Natomiast Parcel pozwala na tworzenie aplikacji całkowicie od zera w sposób, który uznałem za idealny podczas poznawania tematu tworzenia aplikacji React.

Rozpoczęcie pracy

Zanim przystąpimy do pracy, musimy utworzyć w komputerze lokalnym kopię plików początkowych projektu. Kod źródłowy projektu (*https://github.com/javascripteverywhere/web*) zawiera wszystkie skrypty i odwołania do bibliotek zewnętrznych, które będą potrzebne podczas budowania aplikacji. Aby utworzyć kopię tego kodu w komputerze lokalnym, otwórz aplikację terminala, przejdź do katalogu używanego do przechowywania projektów, a następnie za pomocą polecenia `git clone` utwórz kopię repozytorium. Jeżeli wykonywałeś kolejne operacje podczas pracy nad API we wcześniejszych rozdziałach, powinieneś mieć już katalog *notedly* przeznaczony dla projektu.

```
# Przejście do katalogu zawierającego projekty.
$ cd
$ cd Projekty
# Jeżeli jeszcze nie masz katalogu notedly, wydaj polecenie `mkdir notedly`.
$ cd notedly
$ git clone git@github.com:javascripteverywhere/web.git
$ cd web
$ npm install
```

Instalowanie zależności zewnętrznych

Dzięki utworzeniu kopii kodu startowego dla projektów przedstawionych w książce i wydaniu polecenia `npm install` w katalogu projektu unikniesz konieczności wydawania tego polecenia dla każdej z zależności zewnętrznych.

Struktura kodu przedstawia się następująco:

/src

Jest to katalog, w którym powinieneś tworzyć nowy kod podczas lektury książki.

/solutions

Ten katalog zawiera gotowe aplikacje w postaci, w jakiej znajdują się na końcu wielu rozdziałów. Jeżeli utkniesz, zajrzyj do zamieszczonego tutaj rozwiązania.

/final

Ten katalog zawiera ostateczną wersję działającego projektu.

Skoro masz kod umieszczony w katalogu lokalnym, powinieneś utworzyć kopię pliku *.env* projektu. Ten plik jest przeznaczony dla różnych informacji środowiskowych projektu, takich jak hasła, adres URL bazy danych, identyfikatory klientów itd. Z powodu tych danych wymieniony plik nigdy nie powinien trafić do systemu kontroli wersji. Będziesz potrzebował własnej wersji tego pliku. Aby ją utworzyć, z poziomu powłoki przejdź do katalogu *web*, a następnie wydaj przedstawione tutaj polecenie.

```
$ cp .env.example .env
```

W katalogu powinieneś zobaczyć nowy plik *.env*. W tym momencie nie musisz nic robić z wymienionym plikiem, ale wraz z postępem prac nad API back-endu do tego pliku będziemy dodawać informacje. W projekcie znajduje się również plik o nazwie *.gitignore*, który ma gwarantować, że plik *.env* nigdy przypadkowo nie trafi do systemu kontroli wersji.

Pomocy! Nie widzę pliku .env!

Domyślnie systemy operacyjne ukrywają pliki o nazwach zaczynających się od kropki, ponieważ zwykle takie pliki są wykorzystywane przez system operacyjny, a nie przez użytkownika końcowego. Jeżeli nie widzisz pliku *.env*, spróbuj otworzyć katalog projektu w edytorze kodu źródłowego. Teraz plik powinien być widoczny w panelu plików edytora tekstu. Ewentualnie możesz w oknie terminala wydać polecenie `ls -a`, które wyświetla wszystkie pliki znajdujące się w katalogu bieżącym.

Budowa aplikacji

Po skopiowaniu kodu startowego do komputera lokalnego możemy przystąpić do tworzenia aplikacji internetowej za pomocą React. Zacznij od pliku *src/index.html*. Zawartość tego pliku wygląda na standardowy, choć jeszcze zupełnie pusty dokument HTML-a. Zwróć uwagę na dwa wiersze kodu.

```
<div id="root"></div>
<script src="./App.js"></script>
```

Mają one ogromne znaczenie dla aplikacji React. Element `<div>` o identyfikatorze `root` jest kontenerem dla całej aplikacji. Natomiast plik *App.js* będzie punktem wyjścia do naszej aplikacji JavaScriptu.

Teraz można rozpocząć faktyczne tworzenie aplikacji React w pliku *src/App.js*. Jeżeli wykonałeś wprowadzenie do React przedstawione w poprzednim rozdziale, kolejne kroki nie będą dla Ciebie nowością. Kod w pliku *src/App.js* zaczyna się od poleceń importujących biblioteki `react` i `react-dom`.

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

Następnym krokiem jest utworzenie funkcji o nazwie `App` zwracającej zawartość naszej aplikacji. W tym momencie ta funkcja będzie zawierała jedynie dwa wiersze kodu umieszczone w elemencie `<div>`.

```
const App = () => {
  return (
    <div>
      <h1>Witaj, Notedly!</h1>
      <p>Witamy w aplikacji Notedly</p>
    </div>
  );
};
```

O co chodzi z tymi wszystkimi elementami ?

Jeżeli dopiero zaczynasz programowanie z użyciem biblioteki React, być może zastanawiasz się nad tendencją do ujmowania komponentów w znaczniki `<div>`. Komponent React musi znajdować się w elemencie nadrzędnym, którym często jest właśnie znacznik `<div>`, choć równie dobrze może to być inny prawidłowy znacznik HTML-a, taki jak `<section>`, `<header>` lub `<nav>`. Jeżeli stosowanie tych znaczników wydaje Ci się niewłaściwe, zamiast nich możesz skorzystać z `<React.Fragment>` lub pustego `<>` i umieścić w nim komponenty w JavaScripcie.

Następnym krokiem jest nakazanie bibliotece React wygenerowania naszej aplikacji w elemencie o identyfikatorze `root`. Wymaga to następującego polecenia:

```
ReactDOM.render(<App />, document.getElementById('root'));
```

Pełna zawartość pliku *src/App.js* powinna mieć teraz przedstawioną tutaj postać.

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

const App = () => {
  return (
    <div>
      <h1>Witaj, Notedly!</h1>
      <p>Witamy w aplikacji Notedly.</p>
    </div>
  );
};

ReactDOM.render(<App />, document.getElementById('root'));
```

Skoro mamy pełny kod źródłowy, warto przetestować go w przeglądarce WWW. W aplikacji terminala uruchom serwer programistyczny za pomocą polecenia `npm run dev`. Po przygotowaniu kodu do uruchomienia przejdź pod adres *http://localhost:1234*, aby wyświetlić stronę internetową (rysunek 12.1).

Rysunek 12.1. Nasza początkowa aplikacja React uruchomiona w przeglądarce WWW

Routing

Jedną z podstawowych funkcji sieci jest możliwość łączenia dokumentów ze sobą. Również w przypadku naszej aplikacji chcemy zapewnić użytkownikom możliwość poruszania się między stronami. W kodzie HTML-a wygenerowanej aplikacji oznacza to utworzenie wielu dokumentów HTML-a. Gdy użytkownik przejdzie do nowego dokumentu, cały dokument zostanie odświeżony, nawet jeśli na dwóch stronach istnieją jakieś jego współdzielone elementy, np. nagłówek i stopka.

W aplikacjach JavaScriptu można wykorzystać routing po stronie klienta. Pod wieloma względami jest to podobne do łączenia dokumentów HTML-a. Użytkownik kliknie łącze, adres URL zostaje uaktualniony i następuje przejście na nową stronę. Różnica będzie polegała na tym, że w budowanej tutaj aplikacji uaktualnienie strony zostanie przeprowadzone po zmianie treści. Ta operacja zostanie przeprowadzona płynnie, jak w tradycyjnej aplikacji komputerowej, więc użytkownik nawet może się nie zorientować, że nastąpiło odświeżenie strony.

W aplikacji React najczęściej stosowaną biblioteką routingu jest React Router (*https://reacttraining.com/react-router/*). Pozwala ona na dodanie routingu do aplikacji internetowej React. W celu zaimplementowania routingu w aplikacji zacznij od utworzenia katalogu *src/pages* z następującymi plikami:

- */src/pages/index.js*,
- */src/pages/home.js*,
- */src/pages/mynotes.js*,
- */src/pages/favorites.js*.

Pliki *home.js*, *mynotes.js* i *favorites.js* będą komponentami poszczególnych stron. Każdy z nich utworzymy z pewną treścią początkową i z zaczepem `effect`, który pozwoli na uaktualnienie tytułu dokumentu, gdy użytkownik przejdzie na daną stronę.

Oto kod pliku */src/pages/home.js*:

```
import React from 'react';

const Home = () => {
  return (
    <div>
      <h1>Notedly</h1>
      <p>To jest strona główna.</p>
    </div>
  );
};

export default Home;
```

Oto kod pliku */src/pages/mynotes.js*:

```
import React, { useEffect } from 'react';

const MyNotes = () => {
  useEffect(() => {
    // Uaktualnienie tytułu dokumentu.
    document.title = 'Moje notatki — Notedly';
  });

  return (
    <div>
      <h1>Notedly</h1>
      <p>To są moje notatki.</p>
```

```
    </div>
  );
};

export default MyNotes;
```

Oto kod pliku */src/pages/favorites.js*:

```
import React, { useEffect } from 'react';

const Favorites = () => {
  useEffect(() => {
    // Uaktualnienie tytułu dokumentu.
    document.title = 'Ulubione – Notedly';
  });

  return (
    <div>
      <h1>Notedly</h1>
      <p>To są moje ulubione notatki.</p>
    </div>
  );
};

export default Favorites;
```

Zaczep useEffect

W poprzednich przykładach do określenia tytułu strony użyliśmy zaczepu React `useEffect`. Ten zaczep pozwala na stosowanie pewnych efektów w komponencie i uaktualnienie tego, co nie jest powiązane z samym komponentem. Jeżeli chcesz dowiedzieć się więcej na temat tego zaczepu, odpowiednie informacje znajdziesz w dokumentacji React na stronie *https://reactjs.org/docs/hooks-effect.html*.

Teraz z pliku *src/pages/index.js* zaimportujesz React Router i metody niezbędne do obsługi routingu w przeglądarce WWW za pomocą pakietu `react-router-dom`.

```
import React from 'react';
import { BrowserRouter as Router, Route } from 'react-router-dom';
```

Następnym krokiem jest zaimportowanie utworzonych przed chwilą komponentów stron.

```
import Home from './home';
import MyNotes from './mynotes';
import Favorites from './favorites';
```

Każdy z przygotowanych komponentów strony będzie zdefiniowany jako trasa z określonym adresem URL. Zwróć uwagę na użycie `exact` dla trasy strony głównej, co gwarantuje, że dany komponent będzie wygenerowany jedynie dla głównego adresu URL.

```
const Pages = () => {
  return (
    <Router>
      <Route exact path="/" component={Home} />
      <Route path="/mynotes" component={MyNotes} />
      <Route path="/favorites" component={Favorites} />
```

```
    </Router>
  );
};

export default Pages;
```

Pełny kod źródłowy w pliku *src/pages/index.js* przedstawia się teraz następująco:

```
// Import biblioteki React i zależności routingu.
import React from 'react';
import { BrowserRouter as Router, Route } from 'react-router-dom';

// Import tras.
import Home from './home';
import MyNotes from './mynotes';
import Favorites from './favorites';

// Definicje tras.
const Pages = () => {
  return (
    <Router>
      <Route exact path="/" component={Home} />
      <Route path="/mynotes" component={MyNotes} />
      <Route path="/favorites" component={Favorites} />
    </Router>
  );
};

export default Pages;
```

Teraz można już uaktualnić plik *src/App.js* w celu użycia naszych tras przez ich zaimportowanie i wygenerowanie komponentów.

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

// Import tras.
import Pages from '/pages';

const App = () => {
  return (
    <div>
      <Pages />
    </div>
  );
};

ReactDOM.render(<App />, document.getElementById('root'));
```

Jeżeli ręcznie zmienisz adres URL w przeglądarce WWW, będziesz mógł wyświetlać poszczególne komponenty. Na przykład po wpisaniu adresu *http://localhost:1234/favorites* nastąpi wygenerowanie strony ulubionych notatek.

Łączenie

Wprawdzie utworzyliśmy strony, ale zapomnieliśmy o kluczowym komponencie połączenia ich ze sobą. Na stronie głównej dodamy więc łącza pozwalające na przejście do poszczególnych stron. W tym celu wykorzystamy komponent `Link` z biblioteki React Router.

Kod w pliku *src/pages/home.js* zmodyfikuj do następującej postaci:

```
import React from 'react';
// Import komponentu Link z react-router.
import { Link } from 'react-router-dom';

const Home = () => {
  return (
    <div>
      <h1>Notedly</h1>
      <p>To jest strona główna.</p>
      {/* Miejsce na listę łączy. */}
      <ul>
        <li>
            <Link to="/mynotes">Moje notatki</Link>
        </li>
        <li>
            <Link to="/favorites">Ulubione</Link>
        </li>
      </ul>
    </div>
  );
};

export default Home;
```

Po wprowadzeniu omówionych tutaj zmian można poruszać się po aplikacji. Kliknięcie łącza na stronie głównej spowoduje przejście do komponentu odpowiedniej strony. Podstawowe funkcje przeglądarki WWW dotyczące poruszania się po stronach, np. przyciski wstecz i do przodu, działają zgodnie z oczekiwaniami.

Komponenty UI

W ten sposób sukcesem zakończyła się operacja utworzenia komponentów poszczególnych stron i nawigacji między nimi. Podczas przygotowywania stron będziesz pracować z wieloma współdzielonymi elementami interfejsu użytkownika, np. nagłówkami i nawigacją po witrynie. Tworzenie ich od początku za każdym razem nie zalicza się do efektywnych rozwiązań, a ponadto może stać się dość irytujące. Zamiast tego najlepiej będzie przygotować wielokrotnego użycia komponenty interfejsu, które następnie będą importowane, gdy zajdzie taka potrzeba. Tak naprawdę traktowanie interfejsu użytkownika jako składającego się z małych komponentów jest jedną z zalet biblioteki React i przekonało mnie do tego frameworka.

Pracę zaczniemy od utworzenia komponentów nagłówka i nawigacji. W katalogu *src* utwórz nowy podkatalog o nazwie *components*. Następnie w *src/components* utwórz dwa pliki o nazwach

Header.js i *Navigation.js*. Nazwy komponentów React muszą zaczynać się wielką literą, więc stosujemy się do konwencji wielkich liter także w nazwach plików.

Na początek przygotujemy komponent nagłówka w pliku *src/components/Header.js*. W tym celu należy zaimportować plik *logo.svg* i dodać do komponentu przedstawiony tutaj fragment kodu.

```
import React from 'react';
import logo from '../img/logo.svg';

const Header = () => {
  return (
    <header>
      <img src={logo} alt="Logo Notedly" height="40" />
      <h1>Notedly</h1>
    </header>
  );
};

export default Header;
```

Na potrzeby komponentu nawigacji zaimportujemy funkcjonalność `Link` z React Router i zastosujemy nieuporządkowaną listę łączy. Zawartość pliku *src/components/Navigation.js* przedstawia się następująco:

```
import React from 'react';
import { Link } from 'react-router-dom';

const Navigation = () => {
  return (
    <nav>
      <ul>
        <li>
          <Link to="/">Strona główna</Link>
        </li>
        <li>
          <Link to="/mynotes">Moje notatki</Link>
        </li>
        <li>
          <Link to="/favorites">Ulubione</Link>
        </li>
      </ul>
    </nav>
  );
};

export default Navigation;
```

Na rysunku w dalszej części rozdziału możesz zauważyć, że użyłem emotikonów jako ikon nawigacyjnych. Jeżeli chcesz zrobić to samo, kod znaczników obejmujący emotikony przedstawia się następująco:

```
<span aria-hidden="true" role="img">
  <!-- Emotikony.-->
</span>
```

Po przygotowaniu komponentów nagłówka i nawigacji można je wykorzystać w naszej aplikacji. Uaktualnij plik *src/pages/home.js* w celu dołączenia tych komponentów. Najpierw komponent trzeba zaimportować, a dopiero potem można go dołączyć do kodu znaczników JSX-a.

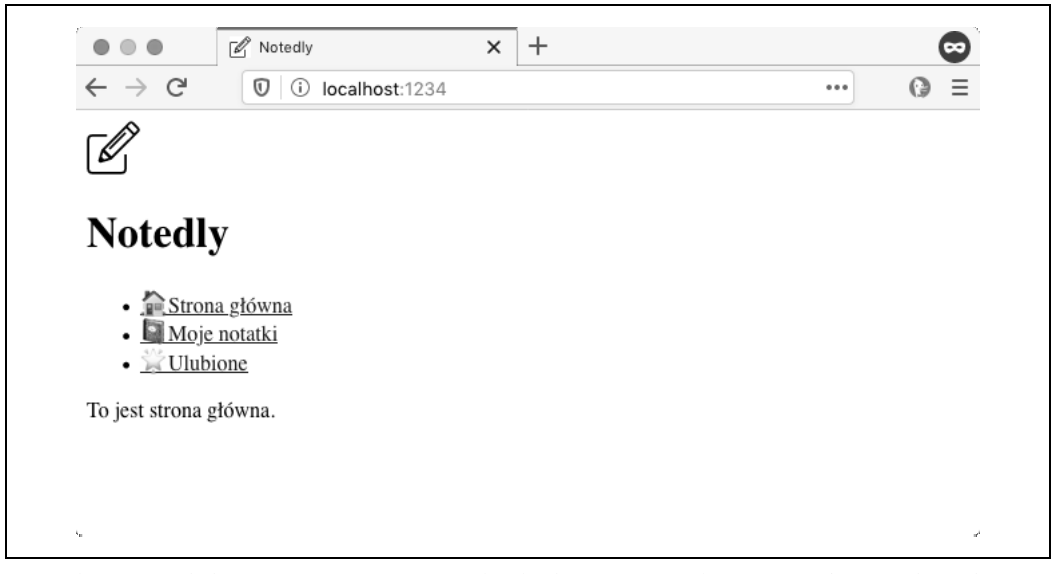
Po wprowadzeniu zmian kod w pliku *src/pages/home.js* ma następującą postać (rysunek 12.2):

```
import React from 'react';

import Header from '../components/Header';
import Navigation from '../components/Navigation';

const Home = () => {
  return (
    <div>
      <Header />
      <Navigation />
      <p>To jest strona główna.</p>
    </div>
  );
};

export default Home;
```



Rysunek 12.2. Dzięki komponentom React można bardzo łatwo tworzyć elementy interfejsu użytkownika wielokrotnego użycia

To już wszystko, co jest wymagane w celu utworzenia komponentów współdzielonych w aplikacji. Więcej informacji na temat używania komponentów w interfejsie użytkownika znajdziesz w sekcji *Thinking in React* dokumentacji biblioteki React na stronie *https://reactjs.org/docs/thinking-in-react.html*.

Podsumowanie

Sieć pozostaje niezrównanym medium umożliwiającym rozpowszechnianie aplikacji. Pozwala na połączenie ogólnego dostępu i możliwości wdrażania uaktualnień w czasie rzeczywistym. W tym rozdziale opracowaliśmy podstawy dla naszej aplikacji internetowej JavaScriptu utworzonej za pomocą biblioteki React. W następnym rozdziale zajmiemy się zdefiniowaniem układu i stylów aplikacji, używając do tego komponentów React i CSS-in-JS.

JavaScript — wykorzystaj prawdziwą wszechstronność!

Dawno temu młodziutki JavaScript służył do ozdabiania stron WWW. Dziś, choć wciąż jest niewielkim językiem skryptowym, jego możliwości są imponujące. Można go użyć do pisania dużych aplikacji na różne platformy, a nawet oprogramowania obsługującego urządzenia w IoT. Jest też świetnym narzędziem do tworzenia API dla interfejsów użytkownika aplikacji internetowej, aplikacji natywnej na urządzenia mobilne i aplikacji do komputerów biurkowych. Pracę ułatwiają nowe technologie, w tym React, React Native i GraphQL, a także framework Electron. Okazuje się, że aby tworzyć wiele różnych rodzajów oprogramowania, wystarczy dobrze poznać JavaScript i towarzyszące mu narzędzia.

Ta książka jest przeznaczona dla średnio zaawansowanych i początkujących programistów, którzy chcą poznać narzędzia ułatwiające tworzenie różnych aplikacji w JavaScripcie. Przedstawiono tu kompletne instrumentarium, dzięki któremu można pisać kod aplikacji w stylu CRUD działającej na każdej platformie. Omówiono, w jaki sposób należy przygotować sobie środowisko programistyczne do pracy, omówiono sposób tworzenia API za pomocą Node i Express, bazy danych MongoDB oraz serwera Apollo. Sporo miejsca poświęcono wykonywaniu interfejsów użytkownika niezależnych od platformy za pomocą różnych narzędzi. Poszczególne zagadnienia zilustrowano praktycznymi przykładami działającego kodu. Dzięki tej publikacji nawet początkujący programista zacznie szybko podejmować świadome decyzje technologiczne.

W książce między innymi:

- procesy GraphQL do pracy z danymi
- wspólny mechanizm uwierzytelniania dla API, aplikacji internetowej i aplikacji natywnych
- wykorzystywanie React i Styled Components
- tworzenie aplikacji dla iOS i Androida za pomocą React Native
- praca z frameworkiem Electron

Adam D. Scott jest inżynierem, programistą aplikacji internetowych i wykładowcą. Obecnie pracuje jako kierownik do spraw aplikacji internetowych w biurze ochrony konsumentów, gdzie wraz ze swoim utalentowanym zespołem koncentruje się na tworzeniu aplikacji internetowych typu open source. Przez ponad dekadę pracował w szkolnictwie — uczył i tworzył programy nauczania wielu tematów technicznych. Wraz z rodziną mieszka w Connecticut.

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

ISBN 978-83-283-7114-9
9 788328 371149
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 69,00 zł