

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

© 2020 Helion SA

Authorized Polish translation of the English edition of JavaScript Everywhere ISBN 9781492046981 © 2020 Adam D. Scott

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

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

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

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

Autor oraz Helion SA 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 Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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.

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

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

Spis treści

Wstęp	13
Wprowadzenie	15
1. Środowisko programistyczne	19
Edytor tekstu	20
Terminal	20
Użycie dedykowanej aplikacji terminala	20
Użycie Visual Studio Code	20
Poruszanie się po systemie plików	20
Narzędzia powłoki i menedżer Homebrew (tylko w systemie macOS)	21
Node.js i menedżer pakietów npm	22
Instalacja Node.js i npm w systemie macOS	22
Instalacja Node.js i npm w systemie Windows	23
MongoDB	23
Instalacja i uruchomienie MongoDB w systemie macOS	23
Instalacja i uruchomienie MongoDB w systemie Windows	24
Git	24
Expo	25
Prettier	25
ESLint	26
Ładny wygląd kodu w edytorze	26
Podsumowanie	26
2. Wprowadzenie do API	27
Co będziemy budować?	27
Jak zbudujemy nasze API?	28
Rozpoczęcie pracy	28
Podsumowanie	30

3. Aplikacja internetowa utworzona za pomocą Node.js i frameworka Express	31
Witaj, świecie!	31
nodemon	32
Rozszerzone opcje portu	33
Podsumowanie	34
4. Pierwsze API GraphQL	35
Zmiana serwera na API (mniej więcej)	35
Podstawy GraphQL	39
Schemat	39
Funkcja resolvera	40
Dostosowanie API do naszych potrzeb	41
Podsumowanie	47
5. Baza danych	49
Rozpoczęcie pracy z MongoDB	50
Połączenie MongoDB z aplikacją	51
Odczytywanie i zapisywanie danych przez aplikację	55
Podsumowanie	61
6. Operacje CRUD	63
Rozdzielenie schematu GraphQL i funkcji resolverów	63
Tworzenie schematu CRUD GraphQL	66
Funkcje resolverów CRUD	67
Data i godzina	69
Podsumowanie	71
7. Konta użytkownika i uwierzytelnianie	73
Uwierzytelnianie w aplikacji	73
Szyfrowanie i tokeny	74
Szyfrowanie hasła	74
Tokeny JSON Web	75
Integracja uwierzytelniania z naszym API	77
Schematy użytkownika	77
Funkcje resolverów uwierzytelniania	78
Dodawanie użytkownika do kontekstu funkcji resolvera	82
Podsumowanie	84
8. Akcje użytkownika	85
Zanim zaczniesz	85
Dodawanie użytkownika do nowej notatki	85
Uprawnienia użytkownika w zakresie uaktualniania i usuwania notatek	88

Zapytania użytkownika	89
Oznaczanie notatki jako ulubionej	92
Zapytania zagnieżdżone	96
Podsumowanie	98
9. Istotne szczegóły	99
Najlepsze praktyki podczas tworzenia aplikacji internetowej i aplikacji opartej na Express.js	99
Express Helmet	99
CORS	100
Stronicowanie	100
Ograniczenia danych	103
Pozostałe aspekty tworzenia naszej aplikacji	104
Testowanie	104
Subskrypcje	104
Platforma Apollo GraphQL	104
Podsumowanie	104
10. Wdrożenie naszego API	105
Hosting bazy danych	105
Wdrożenie aplikacji	109
Konfiguracja projektu	111
Wdrożenie kodu aplikacji	112
Testowanie	113
Podsumowanie	113
11. Interfejsy użytkownika i React	115
JavaScript i interfejs użytkownika	116
Interfejsy deklaratywne tworzone za pomocą JavaScriptu	117
Minimalna znajomość biblioteki React	117
Podsumowanie	122
12. Budowa klienta internetowego za pomocą React	123
Co będziemy budować?	123
Jak będziemy budować naszą aplikację?	124
Rozpoczęcie pracy	125
Budowa aplikacji	126
Routing	127
Łączenie	131
Komponenty UI	131
Podsumowanie	134

13. Nadawanie stylu aplikacji	135
Tworzenie komponentu układu	135
CSS	137
Biblioteka typu CSS-in-JS	138
Utworzenie komponentu przycisku	139
Dodawanie stylów globalnych	140
Style komponentu	142
Podsumowanie	145
14. Praca z klientem Apollo	147
Konfiguracja klienta Apollo	148
Wykonywanie zapytań do API	149
Wybrane style	156
Zapytania dynamiczne	158
Stronicowanie	161
Podsumowanie	163
15. Uwierzelnianie i informacje o stanie	165
Utworzenie formularza rejestracyjnego	165
Formularze React i informacje o stanie	168
Mutacja signUp	170
Tokeny JWT i lokalne magazyny danych	173
Przekierowania	174
Dołączanie nagłówków do żądania	174
Zarządzanie lokalnymi informacjami o stanie	175
Wylogowanie	177
Utworzenie formularza logowania	180
Trasy chronione	185
Podsumowanie	186
16. Operacje tworzenia, odczytywania, uaktualniania i usuwania	187
Tworzenie nowych notatek	187
Odczytywanie notatek użytkownika	193
Uaktualnianie notatek	196
Usunięcie notatki	202
Zmiana ulubionych	204
Podsumowanie	208

17. Wdrożenie aplikacji internetowej	209
Statyczna witryna internetowa	209
Procedura wdrożenia	210
Hosting kodu źródłowego w repozytorium Git	211
Wdrożenie z użyciem Netlify	212
Podsumowanie	214
18. Aplikacje utworzone za pomocą frameworka Electron	215
Co zbudujemy?	215
Jak utworzymy aplikację?	215
Rozpoczęcie pracy	216
Nasza pierwsza aplikacja Electron	217
Szczegóły związane z oknem aplikacji w systemie macOS	218
Narzędzia programistyczne	219
API frameworka Electron	220
Podsumowanie	221
19. Integracja istniejącej aplikacji internetowej z frameworkiem Electron	223
Integracja aplikacji internetowej	223
Ostrzeżenia i błędy	224
Konfiguracja	226
Polityka CSP	227
Podsumowanie	229
20. Wdrożenie aplikacji frameworka Electron	231
Electron Builder	231
Konfigurowanie Electron Builder	232
Kompilacja dla bieżącej platformy	233
Ikony aplikacji	233
Kompilacja aplikacji dla różnych platform	234
Podpisywanie kodu	234
Podsumowanie	235
21. Tworzenie aplikacji mobilnych za pomocą React Native	237
Co będziemy tworzyć?	237
W jaki sposób utworzymy aplikację?	238
Rozpoczęcie pracy	239
Podsumowanie	242
22. Tworzenie aplikacji mobilnej	243
Elementy konstrukcyjne React Native	243
Style i biblioteka Styled Components	245
Biblioteka Styled Components.....	247

Routing	248
Nawigacja oparta na kartach i React Native	249
Nawigacja oparta na stosie i React Native	251
Dodawanie tytułów ekranów	255
Ikony	256
Podsumowanie	258
23. GraphQL i React Native	259
Utworzenie widoków listy i przewijanej treści	259
Zapewnienie routingu listy	265
GraphQL i klient Apollo	266
Tworzenie zapytań GraphQL	267
Dodanie paska postępu wczytywania danych	273
Podsumowanie	273
24. Uwierzytelnianie w aplikacji mobilnej	275
Mechanizm uwierzytelniania	275
Utworzenie formularza logowania	283
Uwierzytelnienie za pomocą mutacji GraphQL	288
Uwierzytelnione zapytania GraphQL	290
Dodanie formularza rejestracji	293
Podsumowanie	298
25. Dystrybucja aplikacji mobilnej	301
Konfiguracja pliku app.json	301
Ikony i ekrany wczytywania aplikacji	303
Ikony aplikacji	303
Winietka	304
Publikowanie aplikacji za pomocą Expo	305
Tworzenie kompilacji natywnych	306
iOS	306
Android	307
Dystrybucja aplikacji w sklepach Apple'a i Google'a	308
Podsumowanie	308
Zakończenie	309
A Lokalne uruchomienie API	311
B Lokalne uruchomienie aplikacji internetowej	313

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. Wszegobecną 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. Klienci Relay firmy Facebook (<https://relay.dev/>) i urql firmy Formidable (<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.

```
# Przejdźcie 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 `<div>`?

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 JavaScriptcie.

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

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

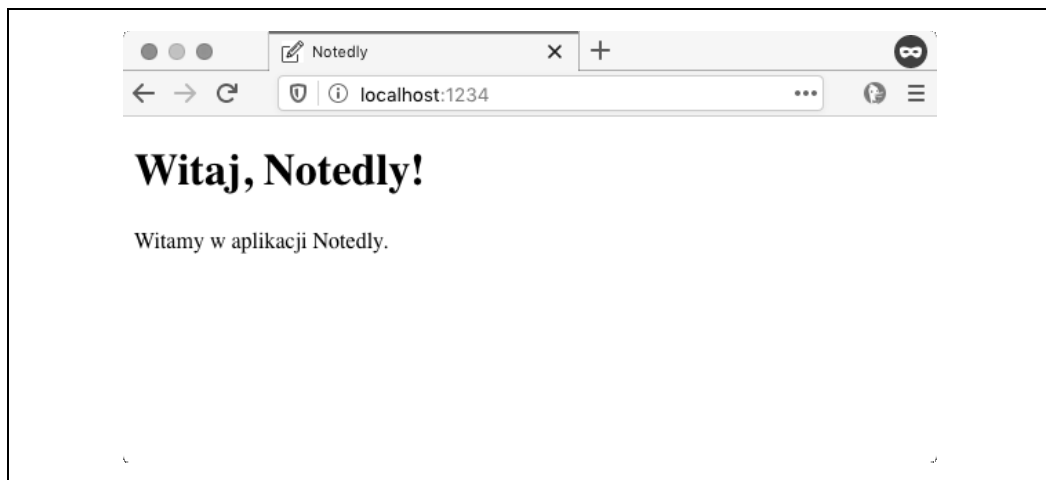
Pełna zawartość pliku `src/App.js` powinna mieć teraz przedstawioną tutaj postać.

```
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. nagłówki 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>
  );
};
```

```

    </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 zaczechu `React useEffect`. Ten zaczech 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 zaczechu, 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>
  );
};

```

```

    </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.

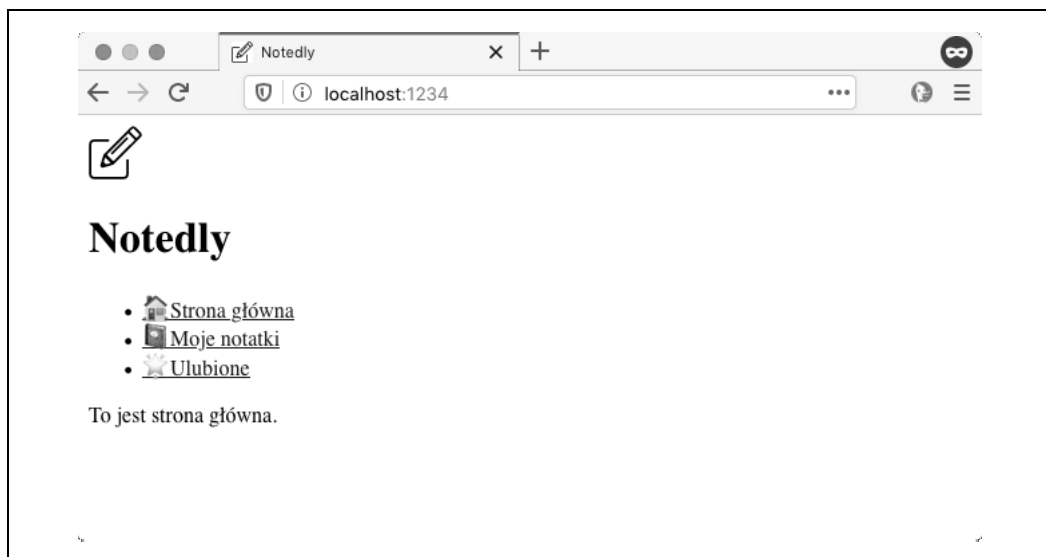
Po wprowadzeniu zmian kod w pliku `src/pages/home.js` ma następującą postać (rysunek 12.2):

```
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.

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>

GRUPA
Helion 

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 biurowych. 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 JavaScriptcie. 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

Sprawdź nasze szkolenia!

SZKOLENIA



AKADEMIA IT & BUSINESS

HELIONSZKOLENIA.PL

KOD KORZYŚCI
Sięgnij po więcej! ▶



ISBN 978-83-283-7114-9



9 788328 371149