Helion

<packt>

WYDANIE II

React i TypeScript

Reaktywne tworzenie stron internetowych dla początkujących

CARL RIPPON

Tytuł oryginału: Learn React with TypeScript: A beginner's guide to reactive web development with React 18 and TypeScript, 2nd Edition

Tłumaczenie: Radosław Słowiński

ISBN: 978-83-289-0770-6

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/reaty2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

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

Printed in Poland.

Spis treści

CZĘŚĆ 2. Podstawy tworzenia aplikacji

ROZDZIAŁ 6

CZĘŚĆ 3. Dane

CZĘŚĆ 4. Zaawansowany React

Stylizacja interfejsów w Reaccie

Rozdział 5

W niniejszym rozdziale zajmiemy się stylowaniem komponentu alertu, nad którym pracowaliśmy wcześniej. W tym celu zastosujemy cztery różne techniki. Zaczniemy od podstawowego CSS i zapoznamy się z jego ograniczeniami. Potem przyjrzymy się **modułom CSS**, które eliminują największe trudności podstawowego CSS. Na koniec skorzystamy z biblioteki **CSS w JS** o nazwie Emotion i biblioteki Tailwind CSS, a także omówię zalety tych rozwiązań.

Nauczysz się również używać grafik SVG w aplikacjach tworzonych za pomocą Reacta oraz zastosujesz je w komponencie alertu jako ikony informacji i ostrzeżenia.

W tym rozdziale omówię zatem następujące tematy:

- CSS;
- moduły CSS;
- CSS w JS;
- Tailwind CSS;
- grafiki SVG.

Wymagania techniczne

W tym rozdziale skorzystamy z następujących narzędzi:

- **Przeglądarka** — najlepiej nowoczesna, taka jak Google Chrome.
- **Node.js** i **npm** — dostępne do pobrania pod adresem *https://nodejs.org/en/download/*.
- **Visual Studio Code** — można pobrać ze strony *https://code.visualstudio.com/*.

Wszystkie kody źródłowe do tego rozdziału znajdziesz pod adresem *https://ftp.helion.pl/przyklady/reaty2.zip*, w folderze *r05*.

CSS

Na początek skonfigurujemy projekt w Reaccie i TypeScripcie z komponentem alertu z rozdziału 3. „Konfiguracja Reacta i TypeScriptu". Następnie dodamy ten komponent i nadamy mu styl za pomocą klasycznego CSS. Na zakończenie przyjrzymy się wyzwaniom związanym z klasycznym CSS i poszukamy rozwiązania tych problemów.

Tworzenie projektu

Będziemy korzystać z projektu, który zakończyliśmy w rozdziale 3. Jest on dostępny w folderze *r03\2-Tworzenie-projektu-przy-użyciu-Create-React-App\myapp*. Wykonaj następujące kroki:

1. Rozpakuj folder *r03\2-Tworzenie-projektu-przy-użyciu-Create-React-App\myapp* do wybranego folderu i otwórz ten folder w Visual Studio Code.
2. W terminalu Visual Studio Code wykonaj następujące polecenie, aby zainstalować wszystkie zależności:

   ```
   npm i
   ```

Teraz, gdy projekt jest gotowy, zastanówmy się, jak wykorzystać CSS w komponentach Reacta.

Jak odnieść się do CSS?

Funkcjonalność Create React App umożliwia wykorzystanie standardowego CSS w projekcie. Patrząc na plik *App.tsx*, zauważysz, że już wtedy wykorzystaliśmy tam tradycyjny CSS:

```
...
import './App.css';
...
function App() {
  return (
    <div className="App">
       ...
    </div>
  );
}
...
```

Style z pliku *App.css* są zaimportowane, a klasa CSS `App` jest przypisana do głównego elementu `div`.

W Reaccie stosujemy atrybut `className` zamiast `class`, gdyż `class` to słowo zastrzeżone w JavaScripcie. W trakcie transpilacji `className` zostaje przekształcone na `class`.

Mechanizm importowania CSS pochodzi z narzędzia webpack. Kiedy webpack przetwarza pliki, dołącza wszystkie zaimportowane style do gotowego pakietu.

Aby przyjrzeć się, jak wygląda gotowy pakiet CSS, wykonaj poniższe czynności:

1. Otwórz i przejrzyj plik *App.css*. Jak już zauważyliśmy, jest on używany w *App.tsx*. Jednak zawiera klasy, które obecnie nie są wykorzystywane, np. `App-header` i `App-logo`. Wcześniej były one używane w komponencie `App`, ale usunęliśmy je przy dodawaniu komponentu `Alert`. Teraz zostaw te klasy tam, gdzie są.
2. Gdy otworzysz plik *index.tsx*, zobaczysz, że importowany jest tam *index.css*. W tym pliku jednak nie ma odniesień do klas CSS. Spójrz na plik *index.css* — zauważysz, że zawiera on tylko reguły CSS, które są ukierunkowane na nazwy elementów, a nie na klasy CSS.
3. Aby utworzyć gotową, produkcyjną wersję projektu, wpisz w terminalu:

   ```
   npm run build
   ```

   W ciągu kilku chwil w katalogu głównym projektu pojawi się folder *build*.
4. Otwórz *index.html* w folderze *build*. Zauważysz zoptymalizowaną wersję pliku — bez niepotrzebnych spacji. Poszukaj teraz elementu `link` wskazującego na plik CSS i zanotuj jego ścieżkę. Powinna wyglądać mniej więcej tak: */static/css/main.073c9b0a.css* (rysunek 5.1).

Rysunek 5.1. Element link w pliku index.html

5. Otwórz wskazany plik CSS. Wszelkie spacje zostały usunięte, dzięki czemu plik jest przygotowany do wdrożenia. Możesz zauważyć, że zawiera on style z *index.css* i *App.css*, w tym zbędne klasy `App-header` i `App-logo` (rysunek 5.2).

Rysunek 5.2. Dołączony plik CSS, wraz ze zbędną klasą App-header

Ważne jest, aby zrozumieć, że webpack nie odrzuca zbędnych stylów — uwzględnia wszystkie style z zaimportowanych plików.

Następnie zajmiemy się stylizacją komponentu alertu przy użyciu CSS.

CSS w komponencie alertu

Skoro już wiesz, jak korzystać z CSS w Reaccie, zajmijmy się komponentem alertu. Postępuj zgodnie z poniższymi krokami:

1. W folderze *src* utwórz plik o nazwie *Alert.css*. Możesz go skopiować z repozytorium pobranego pod adresem *https://ftp.helion.pl/przyklady/reaty2.zip*, z folderu *r05\1-Używanie-czystego-CSS\app\src*.

2. Będziemy dodawać klasy CSS stopniowo, analizując style każdej z nich. Zacznijmy od dodania klasy `container` w pliku *Alert.css*:

```
.container {
  display: inline-flex;
  flex-direction: column;
  text-align: left;
  padding: 10px 15px;
  border-radius: 4px;
  border: 1px solid transparent;
}
```

   Zastosujemy ją dla głównego elementu `div`. Używamy tutaj mechanizmu flexbox, gdzie elementy są układane pionowo i wyrównane do lewej strony. Dla lepszego efektu wizualnego dodaliśmy zaokrąglone krawędzie oraz odstępy między ramką a wewnętrznymi elementami.

3. Wprowadź poniższe dodatkowe klasy, które możesz wykorzystać w klasie `container`:

```
.container.warning {
  color: #e7650f;
  background-color: #f3e8da;
}
.container.information {
  color: #118da0;
  background-color: #dcf1f3;
}
```

   Klasy te posłużą nam do odpowiedniego kolorowania różnorodnych alertów.

4. Wprowadź klasę dla nagłówka elementu container:

```
.header {
  display: flex;
  align-items: center;
  margin-bottom: 5px;
}
```

   Będzie ona przypisana do części zawierającej ikonę, tytuł oraz przycisk zamykania. Korzysta z flexboxa w układzie poziomym z pionowo wycentrowanymi elementami wewnątrz. Ponadto wprowadza mały odstęp pomiędzy nagłówkiem a treścią alertu.

5. Dla ikony wprowadź klasę określającą jej szerokość na 30 px:

```
.header-icon {
  width: 30px;
}
```

6. Dodaj klasę, która pogrubi tekst nagłówka:

```
.header-text {
  font-weight: bold;
}
```

7. Wprowadź klasę dla przycisku zamykania:

```
.close-button {
  border: none;
  background: transparent;
  margin-left: auto;
  cursor: pointer;
}
```

Usuwa to zarówno obramowanie, jak i tło przycisku. Ponadto przycisk jest umieszczany po prawej stronie nagłówka i otrzymuje wskaźnik kursora.

8. Wprowadź następującą klasę dla treści elementu:

```
.content {
  margin-left: 30px;
  color: #000;
}
```

Dzięki temu dodawany jest lewy margines, co sprawia, że treść jest w linii z nagłówkiem, a kolor tekstu zmienia się na czarny.

W ten sposób zakończyliśmy wszystkie definicje klas w CSS.

9. Otwórz plik *Alert.tsx* i dodaj komendę importu dla nowo utworzonego pliku CSS:

```
import './Alert.css';
```

10. Teraz będziemy odnosić się do nowo zdefiniowanych klas CSS w ramach komponentu alertu. Wprowadź wyróżnione pogrubieniem klasy CSS do kodu JSX komponentu alertu:

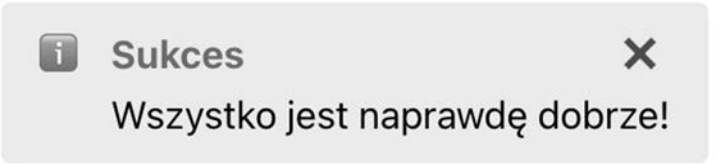
```
<div className={`container ${type}`}>
  <div className="header">
    <span
      ...
      className="header-icon"
    >
      {type === "warning" ? "⚠" : "i"}
    </span>
    <span className="header-text">{heading}</span>
  </div>
  {closable && (
    <button
      ...
      className="close-button"
    >
      ...
    </button>
    )}
  <div className="content">{children}</div>
</div>
```

Teraz dzięki klasom CSS z dołączonego pliku elementy w komponencie alertu mają style.

11. Przenieś przycisk zamykania tak, aby znalazł się w kontenerze nagłówka, dokładnie poniżej elementu `header`:

```
<div className={`container ${type}`}>
  <div className="header">
    ...
    <span className="header-text">{heading}</span>
    {closable && (
      <button
        aria-label="Close"
        onClick={handleCloseClick}
        className="close-button"
      >
        <span role="img" aria-label="Close">
          ✖
        </span>
      </button>
    )}
  </div>
  <div className="content">{children}</div>
</div>;
```

12. Uruchom aplikację w trybie dla deweloperów, wpisując komendę `npm start` w terminalu. Po chwili w przeglądarce zobaczysz ładniejszą wersję komponentu `Alert` (rysunek 5.3).



Rysunek 5.3. Komponent Alert ostylowany przy użyciu standardowego CSS

Stylizacja komponentu alertu została zakończona, ale kontynuujmy pracę, aby zobaczyć wady tej techniki stylowania.

Kolizje w CSS

Za chwilę zobaczysz, jak style CSS mogą kolidować w różnych komponentach. Pozostaw aplikację uruchomioną w trybie dla programistów i postępuj według następujących instrukcji:

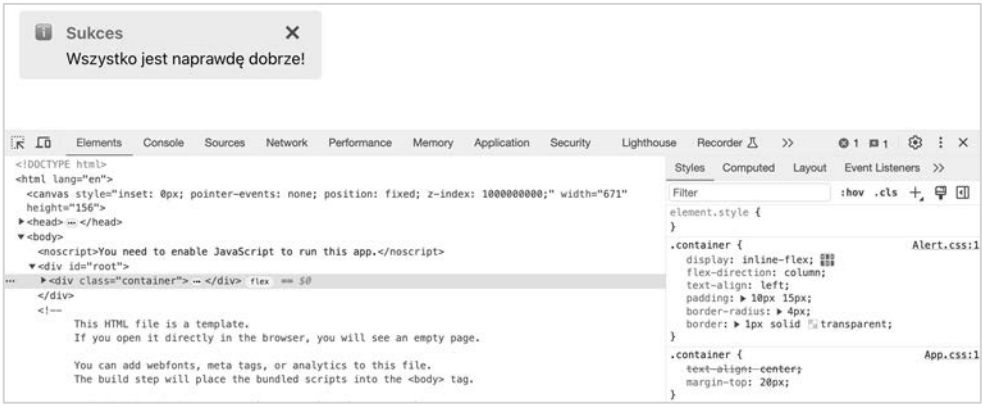
1. Otwórz plik *App.tsx* i zamień używaną klasę CSS `App` na `container` w elemencie `div`:

```
<div className="container">
  <Alert ...>
    ...
  </Alert>
</div>
```

2. Następnie otwórz plik *App.css*, zmień nazwę klasy CSS z `App` na `container` oraz dodaj do niej `20px` paddingu:

```
.container {
  text-align: center;
  padding: 20px;
}
```

3. Teraz, obserwując działającą aplikację, zauważysz, że alerty nie są już wyśrodkowane na stronie. Sprawdź elementy przy użyciu narzędzi deweloperskich w przeglądarce (rysunek 5.4). Gdy przyjrzysz się bliżej elementowi `div` z komponentu `App`, dostrzeżesz, że style z klasy CSS `container`, pochodzące z komponentu `Alert`, zostały również na niego nałożone, tak samo jak nowo dodana przez nas klasa `container`. W efekcie właściwość CSS `text-align` jest ustawiona na wartość `left`, a nie `center`.



Rysunek 5.4. Kolizja klas CSS

4. Zatrzymaj działającą aplikację, używając kombinacji klawiszy *Ctrl+C*.

   Istotą problemu jest to, że standardowe klasy CSS działają na całą aplikację, a nie tylko dla konkretnego zaimportowanego pliku. To sprawia, że klasy mogą ze sobą kolidować, jeżeli mają takie same nazwy, co właśnie zaobserwowaliśmy.

   Aby uniknąć takich kolizji, warto stosować konwencję nazywania BEM. Na przykład klasa `container` w `App` mogłaby nosić nazwę `App__container`, a w `Alert` — `Alert__container`. Jednak to podejście wymaga konsekwencji od całego zespołu deweloperów.

Uwaga

BEM (akronim od *Blok, Element, Modyfikator*) to metoda nazywania klas w CSS. Więcej na ten temat można przeczytać pod adresem *https://css-tricks.com/bem-101/*.

Oto krótkie podsumowanie tej części rozdziału:

- Narzędzie Create React App konfiguruje webpack w taki sposób, by umożliwić import plików CSS do komponentów Reacta.

- Wszystkie style z zaimportowanego pliku CSS są stosowane globalnie w aplikacji — nie ma izolowania ani eliminowania niepotrzebnych stylów.

W kolejnym kroku zapoznasz się z metodą stylizacji eliminującą konflikty stylów CSS między komponentami.

Moduły CSS

Moduły CSS (ang. *modules CSS*) to sposób stylizacji w aplikacjach tworzonych z użyciem Reacta. Zacznijmy od wyjaśnienia, czym dokładnie są, a potem zastosujmy je w komponencie alertu.

Co to są moduły CSS?

Moduły CSS to otwartoźródłowa biblioteka dostępna na GitHubie pod adresem *https://github.com/css-modules/css-modules*. Można ją zintegrować z webpackiem, aby ułatwić automatyczne określanie zakresu nazw klas CSS.

Moduł CSS to, podobnie jak wcześniej omawiane pliki, po prostu plik ze stylami. Jednak ma on rozszerzenie *.module.css*, a nie *.css*. Dzięki temu webpack wie, jak go inaczej przetworzyć.

Import modułu CSS do komponentu Reacta wygląda następująco:

```
import styles from './styles.module.css';
```

Składnia importu jest zbliżona do składni CSS, ale wprowadza się zmienną przechowującą informacje o mapowaniu klas CSS. W podanym fragmencie kodu informacje dotyczące klas CSS zaimportowano do zmiennej `styles`. Nazwa może być dowolna.

Taka zmienna mapująca nazwy klas CSS jest obiektem, w którym nazwy właściwości odpowiadają klasom CSS. Każda taka właściwość wskazuje na unikalną nazwę klasy do użycia w komponencie Reacta. Oto przykład, jak może wyglądać taki obiekt mapowania w komponencie o nazwie `MyComponent`:

```
{
  container: "MyComponent_container__M7tzC",
  error: "MyComponent_error__vj80j"
}
```

Nazewnictwo klas w module CSS składa się z nazwy pliku komponentu, pierwotnej nazwy klasy CSS oraz losowego ciągu. Dzięki temu unikamy kolizji nazw.

W komponencie odnosimy się do stylów z modułu CSS w atrybucie `className` w następujący sposób:

```
<span className={styles.error}>A bad error</span>
```

W efekcie nazwa klasy CSS dla danego elementu zostanie przetłumaczona na odpowiednią, unikalną nazwę. W podanych przykładach `styles.error` zostanie zamienione na `MyComponent_error__vj80j`. Ostatecznie w aplikacji używane są nazwy stylów, a nie oryginalne nazwy klas.

Projekty tworzone za pomocą Create React App mają już moduły CSS zintegrowane z webpackiem. Dlatego nie trzeba instalować modułów CSS, by móc je wykorzystać.

Dalej zajmiemy się wdrażaniem modułów CSS w komponencie alertu.

Zastosowanie modułów CSS w komponencie alertu

Znając już ideę modułów CSS, zastosujmy je w komponencie alertu. Postępuj zgodnie z poniższymi krokami:

1. Przemianuj plik *Alert.css* na *Alert.module.css*, by mógł funkcjonować jako moduł CSS.
2. Otwórz plik *Alert.module.css* i zmień nazewnictwo klas z notacji kebabcase na camelCase. Ułatwi to odwoływanie się do nich w komponencie — na przykład `styles.headerText` zamiast `styles["header-text"]`. Oto jakie zmiany trzeba wprowadzić:

```
...
.headerIcon {
  ...
}
.headerText {
  ... }
.closeButton {
  ...
}
```

3. Następnie otwórz *Alert.tsx* i dostosuj instrukcję importowania CSS tak, by wczytać moduł CSS w następujący sposób:

```
import styles from './Alert.module.css';
```

4. W składni JSX dostosuj odwołania do klas, by używały nazw z modułu CSS:

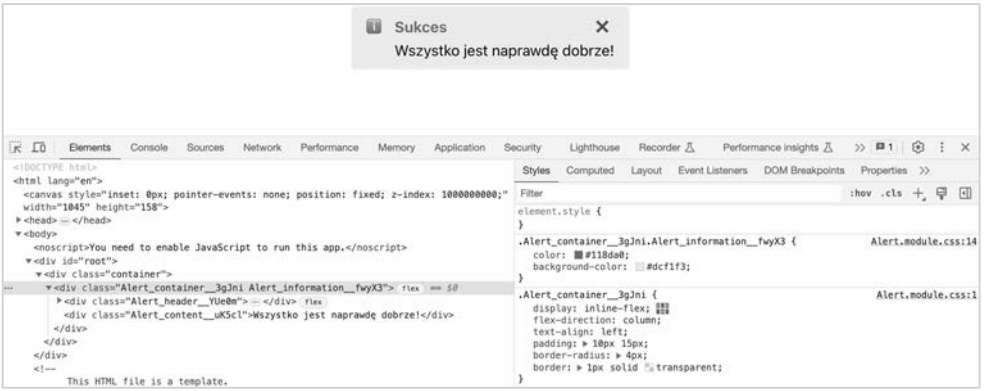
```
<div className={`${styles.container} ${styles[type]}`}>
  <div className={styles.header}>
    <span
      ...
      className={styles.headerIcon}
    >
      {type === "warning" ? "⚠" : "i"}
    </span>
    {heading && (
      <span className={styles.headerText}>{heading}</ span>
    )}
    {closable && (
      <button
        ...
        className={styles.closeButton}
      >
        ...
      </button>
```

```
      )}
    </div>
    <div className={styles.content}>{children}</div>
  </div>
```

5. Włącz aplikację, wpisując komendę `npm start` w terminalu.

   Po chwili na ekranie wyświetli się sformatowany alert. Teraz będzie on odpowiednio wyśrodkowany, co świadczy o tym, że style nie kolidują ze sobą.

6. Skorzystaj z narzędzi deweloperskich przeglądarki, aby przyjrzeć się elementom w DOM (rysunek 5.5). Zauważysz, że komponent korzysta z nazw klas z modułu CSS. Dzięki temu style alertu nie kolidują z głównymi stylami aplikacji.



Rysunek 5.5. Nazwy klas w zakresie modułu CSS

7. Zanim przejdziesz dalej, zatrzymaj aplikację, naciskając kombinację klawiszy *Ctrl+C*.

8. Aby lepiej zrozumieć moduły CSS, przyjrzyjmy się, jak zachowują się style w wersji produkcyjnej. Ale zanim to zrobimy, dodajmy niepotrzebną klasę CSS do pliku *Alert.module.css*:

```
...
.content {
  margin-left: 30px;
  color: #000;
}
.redundant {
  color: red;
}
```

9. Utwórz wersję produkcyjną aplikacji, wpisując komendę `npm run build` w terminalu. Po kilku sekundach w folderze *build* pojawią się gotowe pliki.

10. Kiedy otworzysz skompilowany plik CSS, możesz zauważyć kilka rzeczy:

    - Znajduje się w nim cały CSS z plików *index.css*, *App.css* oraz z modułu CSS, który właśnie utworzyliśmy.

- Klasy w module CSS są ograniczone. Dzięki temu style w wersji produkcyjnej nie będą ze sobą kolidowały, podobnie jak miało to miejsce w trybie deweloperskim.
- Plik zawiera również zbędną nazwę klasy CSS z modułu CSS (rysunek 5.6).

Rysunek 5.6. Niepotrzebna klasa CSS w zestawie CSS

Zakończyliśmy adaptację komponentu alertu do wykorzystania modułów CSS.

Uwaga

Jeśli chcesz dowiedzieć się więcej o modułach CSS, odwiedź repozytorium na GitHubie pod adresem *https://github.com/css-modules/css-modules*.

Oto krótkie podsumowanie modułów CSS:

- Za sprawą modułów CSS nazwy klas są automatycznie przypisane do konkretnego komponentu Reacta. Dzięki temu style różnych komponentów nie kolidują ze sobą.
- Moduły CSS to nie funkcja wbudowana w przeglądarki, lecz otwarte źródło, które możesz dodać do webpacka.
- W projektach utworzonych przy użyciu Create React App moduły CSS są już domyślnie zainstalowane i skonfigurowane.
- Podobnie jak standardowy CSS, dodatkowe klasy nie są usuwane z produkcyjnej wersji CSS.

W kolejnym kroku przyjrzymy się innemu sposobowi stylizowania aplikacji tworzonych z użyciem Reacta.

CSS w JS

W tej części omówię CSS w JS i jego zalety. Następnie dostosujemy komponent alertu tak, aby zastosować tę technikę i zobaczyć, jak różni się od modułów CSS.

Koncepcja CSS w JS

CSS w JS (ang. *CSS-in-JS*) nie jest cechą wbudowaną w przeglądarki ani konkretną biblioteką. To rodzaj biblioteki. Przykłady popularnych bibliotek wykorzystujących CSS w JS to ***styled-components*** i **Emotion**. Różnice między *styled-components* a Emotionem nie są duże — obie cieszą się popularnością i mają zbliżone API. W tym rozdziale zajmiemy się Emotionem.

Emotion tworzy style o ograniczonym zakresie, podobnie jak moduły CSS. Ale zamiast pisać CSS w oddzielnym pliku, robisz to wewnątrz kodu w JavaScripcie — stąd termin *CSS w JS*. W praktyce możesz definiować style bezpośrednio dla elementów JSX w następujący sposób:

```
<span
  css={css`
    font-weight: 700;
    font-size: 14;
  `}
>
  {text}
</span>
```

Każda biblioteka typu CSS w JS ma nieco inną składnię — podany przykład pochodzi ze stylizacji przy użyciu Emotiona.

Posiadanie stylów bezpośrednio w kodzie komponentu ułatwia programiście pełne zrozumienie jego działania, bez potrzeby przeskakiwania między plikami. To oczywiście powiększa objętość pliku, co może komplikować jego czytelność. Można jednak wyodrębnić z niego komponenty podrzędne, by uprościć strukturę. Ewentualnie style można przenieść do osobnej funkcji JavaScriptu i zaimportować ją do komponentu.

Dużą zaletą CSS w JS jest możliwość wplecenia logiki w definicję stylów, co jest bardzo pomocne w aplikacjach o dużym stopniu interaktywności. Kolejny przykład pokazuje warunkowe formatowanie tekstu, gdzie zastosowanie `font-weight` zależy od propsa `important`, a `font-size` od propsa `mobile`:

```
<span
  css={css`
    font-weight: ${important ? 700 : 400};
    font-size: ${mobile ? 15 : 14};
  `}
>
  {text}
</span>
```

Interpolacja ciągu znaków w JavaScripcie służy do definiowania instrukcji warunkowej.

W standardowym CSS odpowiednik wyglądałby mniej więcej tak jak w poniższym przykładzie, gdzie dla różnych warunków tworzone są osobne klasy CSS:

```
<span
  className={`${important ? "text-important" : ""} ${
    mobile ? "text-important" : ""
  }`}
>
  {text}
</span>
```

Jeżeli stylizacja elementu jest mocno zależna od pewnych warunków, CSS w JS jest niewątpliwie bardziej przejrzysty i prostszy w pisaniu.

Teraz zajmiemy się wykorzystaniem Emotiona w komponencie alertu, nad którym ostatnio pracowaliśmy.

Stosowanie Emotiona w komponencie alertu

Skoro już znamy koncepcję CSS w JS, zastosujmy Emotiona w komponencie alertu. Aby to zrobić, postępuj zgodnie z poniższymi instrukcjami. (Wszystkie fragmenty kodu dostępne są w pliku *r05\3-Używanie-CSS-w-JS\app\src\Alert.tsx*):

1. Środowisko Create React App domyślnie nie ma zainstalowanego Emotiona, dlatego najpierw musimy zainstalować tę bibliotekę. Wprowadź następujące polecenie w terminalu:

   ```
   npm i @emotion/react
   ```

   Proces instalacji zajmie chwilę.

2. Przejdź do pliku *Alert.tsx* i usuń import modułu CSS.

3. Na początku pliku zaimportuj props `css` z Emotiona i dodaj do niego specjalny komentarz:

   ```
   /** @jsxImportSource @emotion/react */
   import { css } from '@emotion/react';
   import { useState } from 'react';
   ```

   Ten komentarz powoduje, że elementy JSX są przetwarzane przez funkcję `jsx` z Emotiona, a nie przez standardową funkcję `createElement` z Reacta. Funkcja `jsx` w Emotionie pozwala dodawać style do elementów, które korzystają z właściwości `css` tej biblioteki.

4. Teraz trzeba zastąpić wszystkie propsy `className` w JSX ich odpowiednikami z Emotiona, czyli atrybutami `css`. Style są bardzo podobne do tych, które zdefiniowaliśmy wcześniej w pliku CSS, więc nie będę ich ponownie opisywać.

   Stylizować będziemy po jednym elemencie, zaczynając od zewnętrznego elementu `div`:

   ```
   <div
     css={css`
       display: inline-flex;
       flex-direction: column;
       text-align: left;
       padding: 10px 15px;
       border-radius: 4px;
       border: 1px solid transparent;
       color: ${type === "warning" ? "#e7650f" : "#118da0"};
       background-color: ${type === "warning"
         ? "#f3e8da"
         : "#dcf1f3"};
     `}
   >
     ...
   </div>
   ```

W tym fragmencie kodu jest kilka ważnych kwestii do omówienia:

- Atrybut `css` standardowo nie jest akceptowany w elementach JSX. Specjalny komentarz na górze pliku (`/** @jsxImportSource @emotion/react */`) to umożliwia.
- Atrybut `css` jest przypisany do otagowanego literału szablonu. **Otagowany literał szablonu** (ang. *tagged template literal*) to specjalny rodzaj ciągu znaków, który jest przetwarzany przez funkcję wskazaną przed nim — w tym przypadku jest to funkcja o nazwie `css`. Więcej informacji o otagowanych literałach szablonu można znaleźć na stronie *https://developer.mozilla.org/pl/docs/Web/JavaScript/Reference/Template_literals*.
- Otagowany literał szablonu przekształca styl w klasę CSS podczas wykonywania kodu. Zobaczysz to w punkcie 14.
- Interpolacja ciągów znaków jest wykorzystywana do warunkowej stylizacji kolorów. Pamiętaj, że w zwykłym CSS lub przy użyciu modułów CSS musieliśmy zdefiniować trzy różne klasy. Ta wersja oparta na CSS w JS jest niewątpliwie bardziej przejrzysta i zwięzła.

5. Teraz zajmiemy się stylizacją nagłówka elementu:

```
<div
  css={css`
    display: flex;
    align-items: center;
    margin-bottom: 5px;
`}
>
  <span role="img" ... > ... </span>
  <span ...>{heading}</span>
  {closable && ...}
</div>
```

6. Następnie przeprowadzimy stylizację ikony:

```
<span
  role="img"
  aria-label={type === "warning" ? "Warning" :
    "Information"}
css={css`
    width: 30px;
  `}
>
  {type === "warning" ? "⚠" : "i"}
</span>
```

7. A teraz kolej na stylizację nagłówka:

```
<span
  css={css`
    font-weight: bold;
  `}
```

```
>
  {heading}
</span>
```

8. W kolejnym kroku zajmujemy się przyciskiem zamykania:

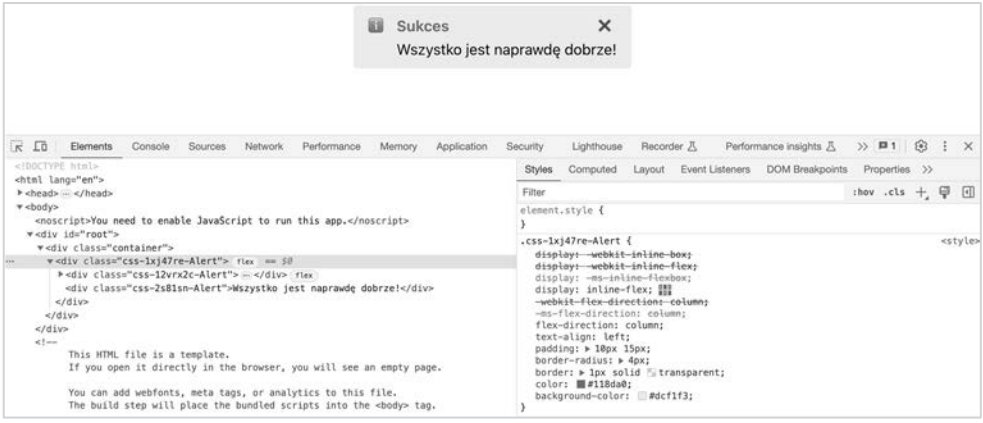
```
{closable && (
  <button
    ...
    css={css`
      border: none;
      background: transparent;
      margin-left: auto;
      cursor: pointer;
    `}
  >
    ...
  </button>
)}
```

9. Na koniec stylizujemy element zawierający wiadomość:

```
<div
  css={css`
    margin-left: 30px;
    color: #000;
  `}
>
   {children}
</div>
```

10. Włącz aplikację, wpisując w terminalu `npm start`. Alert będzie prezentował się dokładnie tak jak poprzednio.

11. Zbadaj elementy w DOM, korzystając z narzędzi deweloperskich w przeglądarce. Komponent alertu używa specjalnych nazw klas CSS, podobnych do modułów CSS (rysunek 5.7).



Rysunek 5.7. Nazwy klas Emotiona z podziałem na zakresy

12. Przed kontynuacją zakończ działanie aplikacji, naciskając kombinację klawiszy *Ctrl+C*.
13. Aby pogłębić wiedzę o Emotionie, przyjrzyjmy się zachowaniu CSS w wersji produkcyjnej. Utwórz wersję produkcyjną, wpisując w terminalu `npm run build`.

    Po chwili w folderze *build* zostaną utworzone odpowiednie pliki.
14. Otwórz skompilowany plik CSS z katalogu *build/static/css*. Zobaczysz, że brakuje tam stylów Emotiona. Jest to spowodowane tym, że Emotion generuje style w trakcie działania aplikacji, a nie podczas jej kompilowania. Style nie mogą być tworzone na etapie kompilacji, bo mogą zależeć od zmiennych w JavaScripcie, których wartości są znane tylko podczas działania aplikacji.

Na tym kończymy proces refaktoryzacji komponentu alertu — zaimplementowaliśmy CSS w JavaScripcie.

Uwaga

Więcej na temat Emotiona znajdziesz na jego oficjalnej stronie: *https://emotion.sh/docs/introduction*.

Oto podsumowanie zdobytej wiedzy o Emotionie i CSS w JavaScripcie:

- Style dla bibliotek typu CSS w JS określamy w JavaScripcie, nie w plikach CSS.
- Możemy bezpośrednio definiować style Emotiona w elemencie JSX, używając atrybutu `css`.
- Dużym plusem jest możliwość wprowadzania logiki warunkowej bezpośrednio do stylu, co przyspiesza stylizowanie interaktywnych elementów.
- Style Emotiona są dodawane podczas działania aplikacji, nie w trakcie kompilowania, ponieważ bazują na zmiennych z JavaScriptu. Choć pozwalają w łatwy sposób wprowadzić logikę warunkowego stylowania komponentu, obniża to wydajność, gdyż style są tworzone i dodawane podczas działania aplikacji.

Teraz przyjrzymy się innemu sposobowi stylizacji w Reaccie.

Tailwind CSS

W tej części poznasz narzędzie Tailwind CSS i jego zalety. Następnie dostosujemy komponent alertu do korzystania z Tailwinda i zobaczymy, jak różni się to od innych metod, które wcześniej testowaliśmy.

Co to jest Tailwind CSS?

Tailwind to zbiór gotowych klas CSS do stylizowania aplikacji. Nazywany jest też **ramką narzędziową CSS** (ang. *utility-first CSS framework*), gdyż predefiniowane klasy działają jak uniwersalne narzędzia.

Przykładową klasą CSS jest `bg-white`, która nadaje elementowi białe tło (`bg` to skrót od słowa *background*). Kolejnym przykładem jest klasa `bg-orange-500`, która nadaje tłu kolor pomarańczowy w odcieniu 500. Tailwind oferuje bogatą paletę kolorów, którą można dostosowywać.

Można też łączyć klasy narzędziowe, by stylizować elementy. Oto jak wygląda stylizacja przycisku w JSX:

```
<button className="border-none rounded-md bg-emerald-700 text-white cursor-
↪pointer">
  ...
</button>
```

Oto wyjaśnienie klas z poprzedniego przykładu:

- `border-none` eliminuje obramowanie danego elementu.
- `rounded-md` zaokrągla narożniki elementu. Skrót `md` oznacza „średnie”; można też wybrać `lg` (duże) lub `full`, jeśli chcemy mocniej zaokrąglone krawędzie.
- `bg-emerald-700` nadaje tłu elementu odcień szmaragdu 700.
- `text-white` sprawia, że tekst elementu jest biały.
- `cursor-pointer` zmienia kursor na wskaźnik.

Klasy narzędziowe są specyficzne i skoncentrowane na jednoznacznej stylizacji. Są bardzo elastyczne, co umożliwia ich wielokrotne wykorzystanie.

W Tailwindzie można zdefiniować, że dana klasa stanie się aktywna po najechaniu na element, dodając przed nią `hover:`. W poniższym przykładzie tło przycisku zmienia się na ciemniejszy odcień szmaragdu, kiedy użytkownik najedzie kursorem na przycisk:

```
<button className="md border-none rounded-md bg-emerald-700 text-white cursor-
↪pointer hover:bg-emerald-800">
  ...
</button>
```

Istotą Tailwinda jest fakt, że nie tworzymy nowych klas CSS dla każdego elementu, który chcemy stylizować, lecz używamy szerokiego zakresu istniejących już klas. Dzięki temu aplikacja prezentuje się schludnie i spójnie.

Uwaga

Więcej informacji o Tailwindzie znajdziesz na stronie *https://tailwindcss.com/*. Jest to kluczowe miejsce, w którym możesz poznać wszystkie klasy narzędziowe, jakie oferuje Tailwind.

W kolejnym kroku dodamy i skonfigurujemy Tailwinda w naszym projekcie z komponentem alertu.

Jak zainstalować i ustawić narzędzie Tailwind CSS?

Skoro już wiemy, czym jest Tailwind, dodajmy go do projektu z komponentem alertu. W tym celu wykonaj poniższe kroki:

1. W projekcie w Visual Studio zacznij od zainstalowania Tailwinda — wpisz w terminalu:

   ```
   npm i -D tailwindcss
   ```

   Tailwinda dodajemy jako zależność deweloperską, bo nie jest potrzebny podczas działania aplikacji.

2. Tailwind współpracuje z Create React App dzięki bibliotece **PostCSS**. To narzędzie przekształca CSS za pomocą JavaScriptu, a Tailwind działa w nim jako dodatek. Zainstaluj PostCSS następującą komendą:

   ```
   npm i -D postcss
   ```

3. Ponadto Tailwind zaleca użycie **Autoprefixera** — narzędzia dodającego prefiksy do CSS. Aby je zainstalować, wpisz:

   ```
   npm i -D autoprefixer
   ```

4. Teraz, aby wygenerować pliki konfiguracyjne dla Tailwinda i PostCSS, wpisz:

   ```
   npx tailwindcss init -p
   ```

   Po chwili w projekcie pojawią się dwa nowe pliki konfiguracyjne: *tailwind.config.js* dla Tailwinda i *postcss.config.js* dla PostCSS.

5. Teraz otwórz plik *tailwind.config.js* i określ ścieżkę do komponentów Reacta w następujący sposób:

   ```
   module.exports = {
     content: [
       './src/**/*.{js,jsx,ts,tsx}'
       ],
       theme: {
         extend: {},
       },
     plugins: [],
   }
   ```

6. Teraz otwórz plik *index.css* znajdujący się w folderze *src* i na jego początku dodaj trzy następujące linie:

   ```
   @tailwind base;
   @tailwind components;
   @tailwind utilities;
   ```

   To **dyrektywy**, które podczas procesu budowy generują odpowiednie style CSS dla Tailwinda.

Tailwind jest już zainstalowany i gotowy do działania.

W następnym kroku użyjemy Tailwinda, aby dodać style do komponentu alertu.

Korzystanie z narzędzia Tailwind CSS

Przyszedł czas, by skorzystać z Tailwinda i za jego pomocą dodać style do komponentu alertu. Zamieńmy `css` Emotiona na klasy narzędzi Tailwinda w atrybucie `className`. Aby to zrobić, wykonaj następujące kroki:

1. Otwórz plik *Alert.tsx*. Na początek usuń specjalny komentarz związany z Emotionem oraz linijkę importującą `css`.
2. W najbardziej zewnętrznym elemencie `div` zamień atrybut `css` na `className`:

```
<div
  className={`inline-flex flex-col text-left px-4 py-3
    rounded-md border-1 border-transparent`}
>
  ...
</div>
```

   Oto co oznaczają użyte klasy narzędziowe:

   - `inline-flex` i `flex-col` tworzą flexbox w linii, który będzie układał elementy w kolumnach.
   - `text-left` wyrównuje treść do lewej.
   - `px-4` dodaje 4 jednostki przestrzeni jako margines z lewej i prawej.
   - `py-3` dodaje 3 jednostki przestrzeni jako margines na górze i na dole.
   - Już wcześniej spotkaliśmy się z klasą `rounded-md` — zaokrągla ona narożniki elementu `div`.
   - Klasy `border-1` i `border-transparent` dodają przezroczystą obramówkę o szerokości 1 px.

Uwaga

W Tailwindzie jednostki przestrzeni opierają się na skali proporcjonalnej. Jedna jednostka to `0,25rem`, co mniej więcej odpowiada `4px`.

3. W najbardziej zewnętrznym elemencie `div` dodaj warunkowe stylowanie, korzystając z interpolacji:

```
<div
  className={`inline-flex flex-col text-left px-4 py-3 rounded-md border-1
  ↪border-transparent ${
    type === 'warning' ? 'text-amber-900' : 'text-teal-900'
  } ${type === 'warning' ? 'bg-amber-50' : 'bg-teal-50'}`}
>
  ...
</div>
```

   Dla alertów ostrzegawczych kolor tekstu przyjmuje odcień bursztynowy 900, a dla alertów informacyjnych — turkusowy 900. Tło dla alertów ostrzegawczych ma odcień bursztynowy 50, a dla alertów informacyjnych — turkusowy 50.

4. W nagłówku zamiast atrybutu `css` użyj atrybutu `className`:

   Oto co oznaczają użyte klasy:

   - `flex` i `items-center` tworzą poziomy flexbox z elementami wycentrowanymi pionowo.
   - `mb-1` daje 1 jednostkę przestrzeni jako dolny margines elementu.

5. W przypadku ikony zamień atrybut `css` na `className`:

```
<span role="img" ... className="w-7">
  {type === 'warning' ? '⚠' : 'i'}
</span>
```

   `w-7` definiuje szerokość elementu jako 7 jednostek odstępu.

6. W nagłówku zmień atrybut `css` na `className`:

```
<span className="font-bold">{heading}</span>
```

   `font-bold` nadaje elementowi pogrubioną czcionkę.

7. Dla przycisku zamykającego zmień atrybut `css` na `className`:

```
{closable && (
  <button
    ...
    className="border-none bg-transparent ml-auto cursor-pointer"
  >
    ...
  </button>
)}
```

   Tutaj `border-none` usuwa obramowanie, `bg-transparent` nadaje przezroczyste tło, `ml-auto` wyjustowuje element do prawej strony, a `cursor-pointer` zmienia kursor na wskaźnik.

8. Dla elementu zawierającego wiadomość zamień atrybut `css` na `className`:

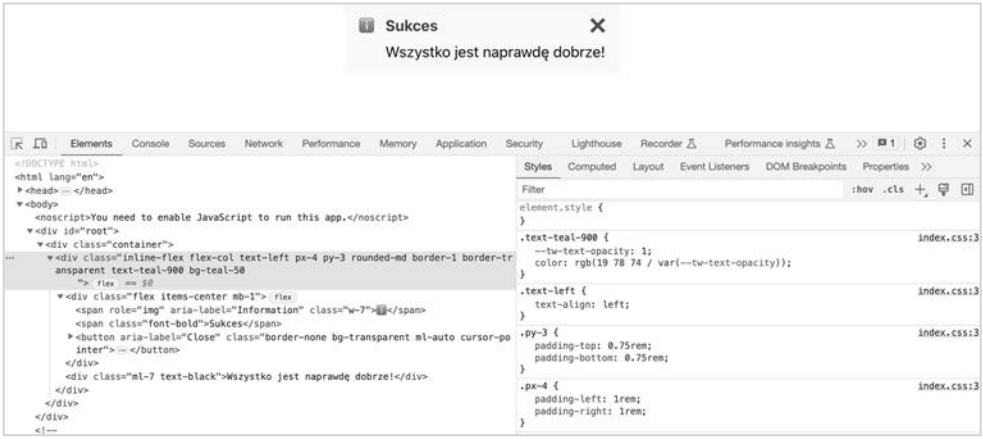
```
<div className="ml-7 text-black">
  {children}
</div>
```

   `ml-7` nadaje elementowi lewy margines, 7 jednostek odstępu, a `text-black` ustawia tekst na kolor czarny.

9. Uruchom aplikację, wpisując `npm start` w terminalu. Po chwili aplikacja zostanie wyświetlona w przeglądarce.

   Dzięki Tailwindowi komponent alertu prezentuje się lepiej, głównie za sprawą domyślnych kolorów i spójnych odstępów.

10. Skorzystaj z narzędzi deweloperskich w przeglądarce, aby przyjrzeć się elementom. Dostrzeżesz klasy narzędzi Tailwinda oraz to, że odstępy korzystają z jednostek rem (rysunek 5.8).

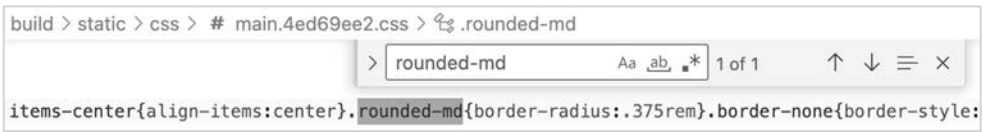


Rysunek 5.8. Alert z zastosowanymi stylami z Tailwinda

Ważne jest, że klasy CSS nie są ograniczone dla konkretnych elementów, dzięki czemu są uniwersalne i wielokrotnego użytku.

11. Aby kontynuować, zatrzymaj aplikację, naciskając kombinację klawiszy *Ctrl+C*.
12. Aby pogłębić wiedzę o Tailwindzie, zastanówmy się nad zachowaniem CSS w wersji produkcyjnej. Na początek utwórz wersję produkcyjną, używając komendy `npm run build` w terminalu.

    Po chwili w folderze *build* pojawiają się utworzone pliki.
13. Przejrzyj scalony plik CSS znajdujący się w katalogu *build/static/css*. Na początku pliku znajdują się podstawowe style Tailwinda, a dalej wszystkie klasy Tailwinda, które zastosowaliśmy (rysunek 5.9).



Rysunek 5.9. Klasy CSS Tailwinda w złączonym pliku CSS

Uwaga

Istotne jest, by wiedzieć, że Tailwind nie dodaje wszystkich swoich klas CSS, gdyż spowodowałoby to powstanie olbrzymiego pliku CSS. W rzeczywistości dodawane są jedynie te klasy, które faktycznie zostały użyte w aplikacji.

Na tym kończymy refaktoryzację komponentu alertu, by korzystał z Tailwinda.

Podsumujmy wiedzę na temat Tailwinda:

- Tailwind to starannie opracowane klasy CSS do wielokrotnego użytku, które możemy zastosować w elementach Reacta.
- Tailwind oferuje atrakcyjną paletę kolorów oraz 4-pikselową skalę odstępów, które można dostosowywać do swoich potrzeb.

- Tailwind działa jako wtyczka dla PostCSS i jest aktywowany podczas kompilacji.
- W odróżnieniu od Emotiona Tailwind nie zmniejsza wydajności w czasie działania, gdyż style nie są generowane i stosowane w trakcie działania aplikacji.
- Do paczki CSS dodawane są tylko te klasy, które są używane w elementach Reacta.

Teraz zajmiemy się wizualnym ulepszeniem ikony w komponencie alertu.

Grafiki SVG

W tej części wyjaśnię, jak pracować z plikami SVG w Reaccie oraz jak je zastosować dla ikon w naszym komponencie.

Jak korzystać z SVG w Reaccie?

SVG, czyli **skalowalne grafiki wektorowe** (ang. *Scalable Vector Graphics*), składają się z punktów, linii i kształtów opartych na wzorach matematycznych, a nie na pojedynczych pikselach. Dzięki temu można je skalować bez utraty jakości. Dobra jakość ikon jest kluczem — ich zniekształcenie sprawia, że cała aplikacja wygląda na mniej profesjonalną. Stosowanie SVG do tworzenia ikon to standard w nowoczesnym tworzeniu aplikacji.

Create React App domyślnie konfiguruje webpack do współpracy z plikami SVG. Przykładowo w domyślnym komponencie `App` odniesienie do *logo.svg* wygląda następująco:

```
import logo from './logo.svg';
...
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" /> ...
      </header>
    </div>
  );
}
export default App;
```

W przytoczonym przykładzie `logo` jest zaimportowane jako ścieżka dostępu do pliku SVG, a potem wykorzystane jako wartość atrybutu src w elemencie img, by wyświetlić obrazek SVG.

Innym sposobem korzystania z SVG jest ich traktowanie jako komponentów:

```
import { ReactComponent as Logo } from './logo.svg';

function SomeComponent() {
  return (
```

```
      <div>-
        <Logo />
      </div>
    );
  }
```

Komponenty SVG w Reaccie dostępne są pod nazwą importu `ReactComponent`. W tym przypadku komponent SVG ma alias `Logo`, który jest używany w kodzie JSX.

Za chwilę dowiesz się, jak używać SVG w alercie.

Dodawanie SVG do alertu

Aby zastąpić ikonki emoji w komponencie alertu ikonami SVG, wykonaj następujące kroki:

1. Zacznij od utworzenia trzech plików — *cross.svg*, *info.svg* oraz *warning.svg* — w katalogu *src*. Skopiuj i wklej ich treść z plików dostępnych pod adresem *r05\5-Używanie-SVG\app\src*.
2. Otwórz plik *Alert.tsx* i dodaj odpowiednie linie importu, by dodać SVG jako komponenty Reacta:

   ```
   import { ReactComponent as CrossIcon } from './cross.svg';
   import { ReactComponent as InfoIcon } from './info.svg';
   import { ReactComponent as WarningIcon } from './warning.svg';
   ```

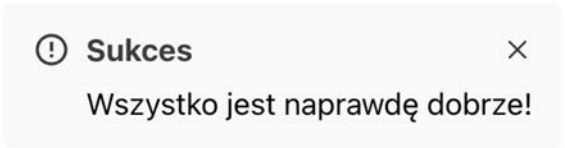
   Przypisaliśmy odpowiednie nazwy dla komponentów SVG.
3. Zaktualizuj element `span`, by zamiast ikonek emoji wykorzystać SVG:

   ```
   <span
     role="img"
     aria-label={type === 'warning' ? 'Warning' :
       'Information'}
     className="inline-block w-7"
   >
     {type === 'warning ' ? (
       <WarningIcon className="fill-amber-900 w-5 h-5" />
     ): (
       <InfoIcon className="fill-teal-900 w-5 h-5" />
     )}
   </span>;
   ```

   Dzięki Tailwindowi ikony mają teraz odpowiedni rozmiar i kolor.
4. Teraz zamień emoji służącą do zamykania komponentu na odpowiednik w SVG:

   ```
   <button
     aria-label="Close"
     onClick={handleCloseClick}
     className="border-none bg-transparent ml-auto cursor-pointer"
   >
     <CrossIcon />
   </button>
   ```

5. Aby zobaczyć efekty, uruchom aplikację komendą `npm start`. Po chwili zobaczysz ją w przeglądarce z nowym, lepszym komponentem alertu (rysunek 5.10).



Rysunek 5.10. Alert z ikoną SVG

Na tym kończymy uzupełnianie komponentu alertu, który teraz wygląda znacznie lepiej.

Oto czego się nauczyłeś o SVG w aplikacjach tworzonych z użyciem Reacta:

- Webpack musi być odpowiednio skonfigurowany do obsługi SVG, ale Create React App robi to za nas.
- Standardowo importowany plik SVG to po prostu ścieżka dostępu do niego, którą możemy wykorzystać w elemencie `img`.
- Można użyć importu nazwanego, `ReactComponent`, aby traktować SVG jako komponent w Reaccie.

Teraz podsumujemy najważniejsze kwestie omówione w tym rozdziale.

Podsumowanie

W tym rozdziale zgłębiliśmy cztery techniki stylizacji.

Najpierw pokazałem, jak wykorzystywać CSS do stylizowania aplikacji w Reaccie. Cechą tego rozwiązania jest fakt, że wszystkie style z zaimportowanego pliku są dołączane, bez względu na to, czy faktycznie są używane. Co więcej, te style nie są przypisane wyłącznie do jednego komponentu — klasy CSS z tymi samymi nazwami dla różnych komponentów wywołują konflikt, co zaobserwowaliśmy dla klasy `container`, która wywoływała kolizję z klasami w komponentach `App` i `Alert`.

Później omówiłem moduły CSS. To narzędzie pozwala tworzyć standardowe pliki CSS, które są importowane w taki sposób, że style są przypisane tylko do danego komponentu. Dowiedziałeś się także, że moduły CSS to otwartoźródłowa biblioteka, która jest domyślnie zainstalowana i skonfigurowana w Create React App. Rozwiązanie to eliminuje problem powstawania kolizji nazw w CSS, jednak nie usuwa nieużywanych stylów.

Następnie przeszliśmy do bibliotek typu CSS w JS, które umożliwiają tworzenie stylów bezpośrednio w komponencie. Wykorzystaliśmy props `css` z biblioteki Emotion, by nadać styl komponentowi alertu bez konieczności tworzenia zewnętrznego pliku CSS. Atutem tego rozwiązania jest szybkość wdrażania logiki warunkowego stylowania elementu. Style z biblioteki Emotion działają podobnie do modułów CSS, tak jak one są ograniczone, jednak przypisanie stylów odbywa się w czasie rzeczywistym, a nie podczas kompilacji. Dostrzegliśmy także niewielki spadek wydajności, który wynika właśnie z tworzenia stylów podczas działania aplikacji.

Czwarte podejście do stylizacji polegało na użyciu narzędzia Tailwind CSS. Tailwind oferuje zestaw uniwersalnych klas CSS, które można wykorzystać w elementach Reacta. Zwróciliśmy uwagę na przyjemną paletę kolorów oraz 4-pikselową skalę odstępów, które można dostosować do własnych potrzeb. Zauważyliśmy też, że do finalnej wersji aplikacji trafiają tylko te klasy z Tailwinda, których faktycznie używamy.

Na koniec przedstawiłem konfigurację Create React App z webpackiem, która umożliwia korzystanie z plików SVG. Można je traktować jako ścieżki dostępu w elemencie `img` lub jako komponenty Reacta, korzystając z importu nazwanego `ReactComponent`.

W kolejnym rozdziale zajmiemy się tworzeniem wielostronicowych aplikacji w Reaccie przy użyciu znanej biblioteki — React Router.

Pytania

Odpowiedz na poniższe pytania, aby sprawdzić wiedzę zdobytą w tym rozdziale:

1. Dlaczego stosowanie standardowego CSS może być kłopotliwe?

   ```
   <div className="wrapper"></div>
   ```

2. Posiadamy komponent, któremu nadano style za pomocą modułów CSS w następujący sposób:

   ```
   import styles from './styles1.module.css';
   function ComponentOne() {
   return <div className={styles.wrapper}></div>;
   }
   ```

   Mamy też inny komponent ostylowany w podobny sposób przy użyciu modułów CSS:

   ```
   import styles from './styles2.module.css';

   function ComponentTwo() {
     return <div className={styles.wrapper}></div>;
   }
   ```

   Czy wystąpi konflikt tych elementów `div`, skoro oba korzystają z klasy o nazwie `wrapper`?

3. Oto komponent, któremu nadano style za pomocą modułów CSS:

   ```
   import styles from './styles3.module.css';

   function ComponentThree() {
     return <div className={styles.wrapper}>
   </div> }
   ```

   Style zdefiniowane w pliku `styles3.module.css` prezentują się następująco:

   ```
   .wrap {
     display: flex;
     align-items: center;
     background: #e7650f;
   }
   ```

   Mimo uruchomienia aplikacji style nie są widoczne. Co jest przyczyną?

4. Tworzymy komponent wielokrotnego użytku przycisku z propsem `kind`, który może przyjmować wartości `"square"` lub `"rounded"`. Przycisk zaokrąglony powinien mieć promień 4 px, a kwadratowy nie powinien mieć promienia wcale. Jak zastosować ten warunkowy styl za pomocą propsa `css` z Emotiona?
5. Nadajemy styl przyciskowi, korzystając z Tailwinda. Obecnie jego styl wygląda tak:

   ```
   <button className="bg-blue-500 text-white font-bold py-2 px-4 rounded">
     Przycisk
   </button>
   ```

   Jak ulepszyć styl, aby tło przycisku zmieniało się na niebieski odcień 700, gdy użytkownik na niego najedzie?
6. Logo w SVG jest załączane w taki sposób:

   ```
   import Logo from './logo.svg';

   function LogoComponent() {
     return <Logo />;
   }
   ```

   Ale logo nie jest wyświetlane. Dlaczego?
7. Nadaliśmy styl przyciskowi za pomocą Tailwinda, używając propsa `color`, który określa jego kolor. Wygląda to tak:

   ```
   <button className={`bg-${color}-500 text-white font-bold py-2 px-4
   ↪rounded`}>
     Przycisk
   </button>
   ```

   Kolor przycisku nie jest taki, jakiego oczekiwaliśmy. Co jest nie tak?

Odpowiedzi

1. Klasa `"wrapper"` może kolidować z innymi klasami. Aby uniknąć takich sytuacji, można ręcznie ograniczyć klasę do konkretnego komponentu:

   ```
   <div className="card-wrapper"></div>
   ```
2. Nie dojdzie do kolizji styli, ponieważ każda klasa jest ograniczona do jednego komponentu przez zastosowanie unikatowych nazw klas w modułach CSS.
3. W komponencie użyto błędnej nazwy klasy — zamiast `"wrapper"` powinno być `"wrap"`:

   ```
   import styles from './styles3.module.css';

   function ComponentThree() {
     return <div className={styles.wrap}>
   </div>
   }
   ```

4. Atrybut `css` dla przycisku mógłby wyglądać następująco:

```
<button
  css={css`
    border-radius: ${kind === "rounded" ? "4px" : "0px"};
  `}
>
  ...
</button>
```

5. Aby dodać akcję wywołaną przez najechanie kursorem, `hover`, można dostosować styl w następujący sposób:

```
<button className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2
↪px-4 rounded">
  ...
</button
```

6. `Logo` zawiera ścieżkę do SVG, a nie komponent. Aby zaimportować komponent logo, należy dostosować poniższą instrukcję importu:

```
import { ReactComponent as Logo } from './logo.svg';

function LogoComponent() {
  return <Logo />;
}
```

7. Użycie klasy `bg-${color}-500` jest problematyczne, ponieważ jej wartość jest określana w trakcie działania aplikacji ze względu na zmienną `color`. Używane klasy Tailwinda są określane i dodawane do zestawu stylów podczas kompilowania, co sprawia, że klasy odpowiedzialne za dodanie koloru tła nie zostaną włączone do pliku. Skutkuje to brakiem zastosowania wybranego koloru tła dla przycisku.

React: efektywne narzędzie dla najlepszych projektantów!

React został zaprojektowany specjalnie na potrzeby Facebooka, w celu tworzenia interfejsów graficznych aplikacji internetowych. Dzisiaj jest powszechnie używany do budowania front-endów interaktywnych UI. Z kolei TypeScript, język napisany przez Microsoft, stanowi rozszerzenie JavaScriptu i cechuje się bogatym systemem typów. Ułatwia to wczesne wykrywanie błędów i refaktoryzację kodu. React i TypeScript, używane razem, pozwalają na efektywne tworzenie dużych, zaawansowanych i łatwych w utrzymaniu front-endów.

To drugie wydanie przewodnika dla programistów, którzy chcą efektywnie budować atrakcyjne złożone front-endy aplikacji. Uwzględniono tu nowe elementy Reacta 18, w tym hooki, biblioteki do zarządzania stanem, jak również najnowszą wersję języka TypeScript. Dzięki tej książce nauczysz się tworzyć przejrzyste i łatwe w utrzymaniu komponenty React, które mogą być wielokrotnie używane — dobrze zorganizowane, bazujące na nowoczesnych wzorcach projektowych. Dowiesz się, jak projektować bezpiecznie typowane komponenty oraz zarządzać złożonymi stanami, a także jak interaktywnie korzystać z web API GraphQL. Poznasz też metody pisania solidnych testów jednostkowych dla komponentów React za pomocą JEST.

W książce między innymi:

- gruntowny wstęp do Reacta i wprowadzenie do TypeScriptu
- tworzenie komponentów Reacta wielokrotnego użytku
- stosowanie hooków Reacta
- czym jest web API REST i GraphQL
- zarządzanie stanem aplikacji
- tworzenie automatycznych testów komponentów

Carl Rippon od ponad dwudziestu lat tworzy skomplikowane aplikacje biznesowe dla różnych branż. Ostatnie osiem lat poświęcił na budowanie aplikacji jednostronicowych przy użyciu szerokiej gamy technologii JavaScriptu, w tym Angulara, ReactJS i TypeScriptu. Prowadzi bloga, na którym porusza różnorodną tematykę z zakresu technologii.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-0770-6
9 788328 907706
Cena: 89,00 zł
packt