Zaawansowany Programista JS

Dzień 8

Plan

- Komponenty w React.JS
- Korzystanie z propsów
- Przekazywanie state przez propsy
- Przekazywanie funkcji przez propsy
- Atomic Design Structure
- Pisanie aplikacji w React.JS przy uzyciu wielu komponentow

Komponenty w React.JS

Komponenty w ReactJS to podstawowe bloki konstrukcyjne aplikacji. Są to funkcje lub klasy, które przyjmują dane wejściowe zwane propsami i zwracają elementy interfejsu użytkownika w postaci JSX. Mogą one być zagnieżdżone i składać się na bardziej złożone struktury UI.

Komponenty w ReactJS umożliwiają łatwe tworzenie wielokrotnego użytku interfejsu użytkownika oraz uporządkowanej struktury kodu. Mogą być wykorzystywane do tworzenia prostych elementów UI, takich jak przyciski, formularze czy pola tekstowe, ale także do bardziej skomplikowanych struktur, takich jak tabele, listy, menu nawigacyjne i wiele innych.

Przykład prostego komponentu funkcyjnego:

```
import React from 'react';

const Button = (props) => {
  return (
    <button onClick={props.onClick}>
      {props.label}
    </button>
  );
}
```

W powyższym przykładzie definiujemy prosty komponent funkcyjny `Button`, który przyjmuje propsy `onClick` i `label`. Ten komponent renderuje przycisk, który po kliknięciu wywołuje funkcję podaną w propsie `onClick`. Tekst przycisku jest zdefiniowany w propsie `label`.

Możemy użyć tego komponentu w innym komponencie, przekazując do niego właściwe propsy:

```
import React from 'react';
import Button from './Button';

function App() {
  return (
    <div>
      <h1>Witaj w mojej aplikacji!</h1>
      <Button label="Kliknij mnie!" onClick={() => alert('Kliknięto!')} />
    </div>
  );
}
```

W tym przypadku używamy komponentu Button w komponencie App i przekazujemy do niego propsy `label` i `onClick`. Komponent `Button` renderuje przycisk z tekstem `"Kliknij mnie!"`, który po kliknięciu wywołuje funkcję `alert()`.

Korzystanie z propsów

Propsy w React.JS to właściwości przekazywane do komponentów, dzięki którym możemy dostarczać dane z komponentu nadrzędnego do podrzędnego. Oto kilka zasad poprawnego korzystania z propsów w React.JS:

- Propsy są przekazywane do komponentu jako atrybuty JSX w momencie, gdy ten jest renderowany. Aby korzystać z propsów w komponencie, należy przekazać je jako parametry funkcji lub jako właściwości klasy.

```
function MyComponent(props) {
  return <h1>{props.title}</h1>;
}
```

- Propsy są tylko do odczytu, więc nie należy ich zmieniać w komponencie. Jeśli chcemy zmienić stan komponentu, powinniśmy użyć mechanizmu stanu (state) w React.JS.

```
function MyComponent(props) {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>{props.text}</p>
      <button onClick={() => setCount(count + 1)}>Kliknij mnie</button>
      <p>{count}</p>
    </div>
  );
}
```

W powyższym przykładzie używamy hooka useState do przechowywania licznika w stanie komponentu, który może być zmieniany przez kliknięcie przycisku. Props text jest wykorzystywany tylko do wyświetlenia tekstu na stronie.

- Należy zawsze deklarować typy propsów, które przekazujemy do komponentów. Jest to szczególnie przydatne, gdy pracujemy w większym zespole lub korzystamy z typowych schematów nazewnictwa.

```
import PropTypes from 'prop-types';

function MyComponent(props) {
  return <h1>{props.title}</h1>;
}

MyComponent.propTypes = {
  title: PropTypes.string.isRequired,
};
```

W powyższym przykładzie używamy biblioteki `PropTypes`, aby zadeklarować, że props title musi być typu string i jest wymagany. Dzięki temu, jeśli ktoś spróbuje użyć komponentu bez podania propsa title lub przekaże do niego coś innego niż string, zostanie wyświetlony błąd w konsoli.

Przekazywanie state przez propsy

Przekazywanie stanu (state) przez propsy w React.JS jest jednym z podstawowych sposobów komunikacji między komponentami. Możemy przekazywać stan z komponentu nadrzędnego do podrzędnego przez propsy, a następnie używać go w podrzędnym komponencie.

Oto przykład, jak przekazywać stan przez propsy:

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

function ParentComponent() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <ChildComponent count={count} />
      <button onClick={() => setCount(count + 1)}>Zwiększ licznik</button>
    </div>
  );
}

function ChildComponent(props) {
  return <p>Licznik: {props.count}</p>;
}
```

W powyższym przykładzie mamy dwa komponenty - `ParentComponent` i `ChildComponent`. W `ParentComponent` przechowujemy stan licznika w zmiennej count i przekazujemy go jako props do `ChildComponent`. W `ChildComponent` wykorzystujemy przekazany przez props stan i wyświetlamy go w elemencie `p`.

W ten sposób możemy przekazywać dowolny stan z komponentu nadrzędnego do komponentu podrzędnego. Jednak należy pamiętać, że zbyt wiele przekazywanych propsów może prowadzić do tzw. "prop drilling", czyli nadmiernego przekazywania propsów przez wiele

warstw komponentów, co może utrudnić utrzymanie i testowanie kodu. W takim przypadku lepszym rozwiązaniem może być użycie innych mechanizmów, takich jak context lub Redux.

Przekazywanie funkcji przez propsy

Przekazywanie funkcji przez propsy w React.JS jest bardzo podobne do przekazywania stanu. Funkcja przekazywana jako props jest wywoływalna w komponencie podrzędnym, co pozwala na wykonywanie pewnych działań w komponencie nadrzędnym po kliknięciu lub w inny sposób aktywując tę funkcję w komponencie podrzędnym.

Oto przykład, jak przekazywać funkcję przez propsy:

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

function ParentComponent() {
  const [message, setMessage] = useState('Hello, World!');

  function handleClick() {
    setMessage('Witaj, Świecie!');
  }

  return (
    <div>
      <ChildComponent onClick={handleClick} />
      <p>{message}</p>
    </div>
  );
}

function ChildComponent(props) {
  return <button onClick={props.onClick}>Zmień wiadomość</button>;
}
```

W tym przykładzie mamy dwa komponenty - `ParentComponent` i `ChildComponent`. W `ParentComponent` przechowujemy stan wiadomości w zmiennej message i definiujemy funkcję `handleClick`, która zmienia tę wiadomość na `"Witaj, Świecie!"` po kliknięciu na przycisk w `ChildComponent`. Następnie przekazujemy funkcję `handleClick` jako props do `ChildComponent`.

W `ChildComponent` definiujemy przycisk, który wywołuje przekazaną przez props funkcję po kliknięciu.

W ten sposób możemy przekazywać dowolne funkcje z komponentu nadrzędnego do komponentu podrzędnego. Dzięki temu możemy umożliwić interakcję użytkownika z naszą aplikacją i wykonywanie określonych działań w komponencie nadrzędnym w zależności od akcji użytkownika w komponencie podrzędnym.

Atomic Design Structure

Atomic Design to podejście do projektowania interfejsów użytkownika, które polega na dzieleniu projektu na mniejsze, bardziej złożone komponenty. Zostało ono stworzone przez Brandona Satroma, a od tego czasu stało się popularnym narzędziem dla projektantów i programistów.

W React.JS Atomic Design Structure oznacza, że komponenty są projektowane jako zbiór małych, niezależnych elementów, które mogą być łatwo ponownie wykorzystane w innych projektach. Ta metoda opiera się na pięciu poziomach, które odpowiadają pięciu typom elementów:

- Atomy - najmniejsze elementy projektowe, takie jak przyciski, pola tekstowe i ikony.
- Cząsteczki - złożone elementy składające się z kilku atomów, na przykład formularze lub przyciski nawigacyjne.
- Organizmy - złożone grupy cząsteczek, tworzące kompletny interfejs użytkownika, na przykład nagłówek strony lub sekcja z formularzem rejestracyjnym.
- Szablony - wykorzystujące organizmy, definiują układ elementów na stronie.
- Strony - w pełni funkcjonalne strony internetowe.

W ten sposób, Atomic Design pozwala na łatwe przetestowanie i wdrożenie komponentów oraz na szybkie tworzenie bardziej skomplikowanych elementów interfejsu. Jest to bardzo korzystne podejście przy projektowaniu interfejsów użytkownika, ponieważ pozwala na zachowanie jednolitego stylu i wzorca projektowego, co z kolei przyspiesza proces tworzenia oprogramowania i ułatwia utrzymanie aplikacji w przyszłości.

Pisanie aplikacji w React.JS przy uzyciu wielu komponentow

Przerób zadanie z poprzedniego dnia, korzystając z wielu komponentów