# Zaawansowany Programista JS

## Dzień 7

Plan

- Wprowadzenie do React.JS
- Przewaga React nad Vanilla JS
- Cechy React.JS
  - Virtual DOM
  - One-way binding
- Create React App
- JSX
- Props
- State
- React Hooks
- Pisanie Aplikacji w React.JS przy uzyciu 1 komponentu

## Wprowadzenie do React.JS

React to biblioteka JavaScript, która służy do budowania interfejsów użytkownika (UI) w aplikacjach webowych. Została stworzona przez Facebooka i po raz pierwszy opublikowana w 2013 roku. React skupia się na deklaratywnym podejściu do tworzenia UI, co oznacza, że programista określa, jak powinien wyglądać interfejs, a React zajmuje się obsługą jego stanu i renderowaniem.

Jedną z głównych zalet Reacta jest jego modułowa struktura. Programista może tworzyć komponenty, które reprezentują fragmenty interfejsu użytkownika, a następnie łączyć je w większe komponenty, aby stworzyć cały interfejs. Komponenty są łatwe w użyciu, ponieważ każdy z nich ma własny stan i cykl życia, co pozwala na prostą kontrolę nad ich zachowaniem.

React wykorzystuje wirtualny DOM (Document Object Model), który umożliwia szybsze i wydajniejsze renderowanie UI. W przypadku zmiany stanu, React porównuje wirtualny DOM z rzeczywistym DOM i renderuje tylko te elementy, które wymagają zmiany, co przyspiesza proces renderowania i poprawia wydajność aplikacji.

## Przewaga React nad Vanilla JS

Kilka zalet ReactJS w porównaniu do natywnego JavaScript:

- Deklaratywność: ReactJS pozwala programistom deklarować, jak powinien wyglądać interfejs użytkownika w zależności od stanu aplikacji, a nie jak go aktualizować. To oznacza, że programiści mogą skupić się na definiowaniu stanu aplikacji i nie muszą się martwić o to, jak interfejs powinien się aktualizować w odpowiedzi na zmiany w stanie.
- Reużywalność komponentów: ReactJS składa się z wielu modularnych komponentów, które można łatwo przesunąć, zmienić lub usunąć, co ułatwia rozwijanie aplikacji. Ponadto, ponieważ React korzysta z języka JavaScript, który jest powszechnie znany, programiści mogą łatwo tworzyć własne komponenty i biblioteki, które mogą być używane w innych projektach.
- Wirtualny DOM: ReactJS wykorzystuje wirtualny DOM, który umożliwia szybsze renderowanie interfejsów użytkownika. Zamiast bezpośrednio manipulować rzeczywistym drzewem DOM, React manipuluje wirtualnym drzewem DOM, które jest lżejsze i szybsze. W rezultacie, gdy zmieni się stan aplikacji, React porównuje wirtualne drzewo DOM z rzeczywistym drzewem DOM i aktualizuje tylko te elementy, które wymagają zmiany.
- Duża społeczność i wiele bibliotek: ReactJS ma ogromną społeczność, która stale rozwija nowe narzędzia i biblioteki, co ułatwia prace z tą biblioteką i umożliwia tworzenie zaawansowanych interfejsów użytkownika.

## Virtual DOM

Virtual DOM (wirtualny DOM) to technika wykorzystywana w bibliotece ReactJS, która polega na tworzeniu w pamięci aplikacji wirtualnego drzewa DOM zamiast bezpośrednio manipulować rzeczywistym drzewem DOM.

Główną zaletą wirtualnego DOM jest to, że pozwala on na szybsze renderowanie interfejsów użytkownika. Zamiast odświeżać całe drzewo DOM po każdej zmianie w aplikacji, ReactJS porównuje wirtualne i rzeczywiste drzewo DOM i aktualizuje tylko te elementy, które wymagają zmiany. Dzięki temu aplikacja działa szybciej i zużywa mniej zasobów.

Wirtualny DOM jest też łatwiejszy w obsłudze, ponieważ pozwala na bardziej deklaratywne podejście do tworzenia interfejsów użytkownika. Zamiast manipulować elementami na drzewie DOM za pomocą bezpośrednich operacji, programista opisuje, jak interfejs powinien wyglądać, a React zajmuje się resztą.

## One-way binding

One-way binding (jednokierunkowe wiązanie danych) to technika, w której zmiany wprowadzone w komponencie rodzica są przekazywane do komponentów potomnych poprzez przekazywanie propsów. Komponenty potomne nie mają wpływu na stan i właściwości swojego rodzica, a jedynie odbierają je jako wartości propsów.

W ReactJS jednokierunkowe wiązanie danych jest stosowane jako standardowy sposób przekazywania danych między komponentami. Oznacza to, że dane przepływają z góry do dołu w hierarchii komponentów, a zmiany wprowadzone w komponencie rodzica są automatycznie odzwierciedlane w komponentach potomnych.

One-way binding jest uważane za bardziej przewidywalne i łatwiejsze w obsłudze niż two-way binding (dwukierunkowe wiązanie danych), które pozwala na edytowanie danych w

jednym komponencie, co automatycznie zmienia stan innych komponentów. W jednokierunkowym wiązaniu danych zmiany w danych muszą być wprowadzone w komponencie rodzica, co ułatwia śledzenie przyczyn błędów i ułatwia debugowanie.

## Create React App

Create React App to narzędzie, które pozwala szybko utworzyć i skonfigurować nowy projekt ReactJS bez konieczności ręcznego konfigurowania środowiska deweloperskiego. Jest to oficjalne narzędzie utworzone przez zespół ReactJS, które pozwala programistom na szybkie rozpoczęcie pracy nad projektem React bez konieczności ustawiania i konfigurowania webpacka, babel czy innych narzędzi.

Create React App oferuje wiele ułatwień dla programistów, takich jak wstępnie skonfigurowane ustawienia Webpacka, automatyczne odświeżanie strony po wprowadzeniu zmian, gotowe skrypty do budowania aplikacji oraz przyjazne dla dewelopera ustawienia środowiska. Ponadto, narzędzie oferuje szereg wbudowanych funkcjonalności, takich jak automatyczne ładowanie plików CSS czy obsługa hot-reloadingu.

Dzięki Create React App programiści mogą skoncentrować się na pisaniu kodu aplikacji zamiast na konfigurowaniu i ustawianiu środowiska deweloperskiego, co pozwala na szybszy start i bardziej produktywną pracę.

Aby postawić aplikację ReactJS za pomocą Create React App, należy wykonać następujące kroki:

Zainstaluj Node.js na swoim komputerze, jeśli nie jest jeszcze zainstalowany.

Utwórz nowy projekt ReactJS za pomocą Create React App. Aby to zrobić, otwórz terminal lub wiersz polecenia i wpisz następujące polecenie:

```
npx create-react-app my-app
```

gdzie `my-app` to nazwa twojego nowego projektu. Polecenie to utworzy nowy projekt ReactJS w katalogu o nazwie `my-app`.

Wejdź do katalogu twojego nowego projektu za pomocą polecenia:

```
cd my-app
```

Uruchom aplikację w trybie deweloperskim za pomocą polecenia:

```
npm start
```

Polecenie to uruchomi serwer deweloperski i otworzy twoją aplikację w przeglądarce internetowej pod adresem `http://localhost:3000/`. Teraz możesz edytować kod swojej aplikacji w dowolnym edytorze tekstu lub IDE i natychmiast zobaczyć wprowadzone zmiany na stronie internetowej.

Jeśli chcesz opublikować swoją aplikację na serwerze produkcyjnym, możesz zbudować aplikację w wersji produkcyjnej za pomocą polecenia:

```
npm run build
```

Polecenie to utworzy zoptymalizowaną wersję twojej aplikacji w katalogu build. Możesz ten katalog opublikować na dowolnym serwerze internetowym, aby udostępnić swoją

aplikację publicznie.

I to wszystko! Dzięki Create React App możesz szybko i łatwo uruchomić i rozwijać swoją aplikację ReactJS.

## JSX

JSX (JavaScript XML) to rozszerzenie składni JavaScript, które umożliwia programistom pisanie kodu HTML wewnątrz kodu JavaScript. Jest to składnia używana w ReactJS do tworzenia interfejsów użytkownika.

JSX pozwala na pisanie kodu, który wygląda jak HTML, ale jest parsowany jako kod JavaScript. Przykładowo, zamiast pisać:

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

można napisać:

```
const element = <h1 className="greeting">Hello, world!</h1>;
```

JSX wygląda jak HTML, ale jest kompilowany do wywołań funkcji JavaScript. Przykładowo, kod JSX `<h1>Hello, world!</h1>` jest kompilowany do wywołania funkcji `React.createElement('h1', null, 'Hello, world!')`.

Dzięki JSX programiści ReactJS mogą pisać kod interfejsu użytkownika w bardziej intuicyjny sposób, co ułatwia ich pracę i zwiększa produktywność.

JSX jest podobny do HTML, ale jest to rozszerzenie składni JavaScript, które pozwala na bezpośrednie osadzanie kodu HTML wewnątrz kodu JavaScript. Oto kilka różnic między JSX a HTML:

- Sposób zapisu atrybutów: W JSX atrybuty są zapisywane w notacji camelCase, podczas gdy w HTML są one zapisywane w notacji kebab-case. Na przykład, w JSX piszemy `className` zamiast `class`, a `onClick` zamiast `onclick`.
- Wymagany zamknięty tag: W JSX każdy tag musi być zamknięty, nawet jeśli nie ma wewnątrz żadnych elementów. Na przykład, `<div></div>` jest poprawny, ale `<div>` nie jest.
- Właściwości: W JSX można przekazywać właściwości do komponentów w sposób podobny do HTML. Jednak w JSX atrybuty, które nie są właściwościami JavaScript, takie jak class lub for, muszą być napisane z użyciem notacji camelCase. Przykładowo, w JSX zamiast `class`, należy użyć `className`, a zamiast `for`, należy użyć `htmlFor`.
- Zmienne JavaScript: W JSX można używać zmiennych JavaScript, umieszczając je wewnątrz klamerek {}. Na przykład, `<div>{nazwaZmiennej}</div>` spowoduje wypisanie zawartości zmiennej `nazwaZmiennej`.

## Props

Propsy (lub właściwości) to mechanizm przekazywania danych z komponentów rodziców do ich komponentów potomnych w ReactJS. Propsy to obiekty, które są przekazywane do komponentu jako argumenty funkcji. Wewnątrz komponentu można odwoływać się do propsów za pomocą notacji kropkowej.

Przykładowo, jeśli mamy komponent o nazwie `MyComponent`, który chcemy wywołać i przekazać do niego propsy name i age, możemy to zrobić w następujący sposób:

```
<MyComponent name="Jan" age={30} />
```

Wewnątrz komponentu `MyComponent` można odwołać się do przekazanych propsów za pomocą notacji kropkowej:

```
const MyComponent = (props) => {
  return (
    <div>
      <p>Imię: {props.name}</p>
      <p>Wiek: {props.age}</p>
    </div>
  );
}
```

W powyższym przykładzie propsy `name` i `age` są przekazywane do komponentu `MyComponent`, który wyświetla wartości tych propsów za pomocą elementów `<p>`. Propsy pozwalają na przekazywanie danych między komponentami i na tworzenie komponentów reużywalnych, które mogą być konfigurowane za pomocą przekazywanych propsów.

## State

State to mechanizm w ReactJS służący do przechowywania danych, które mogą się zmieniać w trakcie działania aplikacji. W przeciwieństwie do propsów, które są przekazywane do komponentów z zewnątrz, state jest wewnętrznym stanem komponentu i jest zarządzany przez sam komponent.

Aby skorzystać ze state w komponencie, musimy użyć hooka useState, który jest dostępny od wersji React 16.8. Hook useState przyjmuje początkową wartość stanu i zwraca tablicę, w której pierwszy element to aktualna wartość stanu, a drugi element to funkcja, która pozwala na zmianę wartości stanu.

Przykładowo, jeśli chcemy stworzyć komponent `Counter`, który będzie wyświetlał licznik z możliwością zwiększania i zmniejszania jego wartości, możemy to zrobić za pomocą hooka useState:

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

const Counter = () => {
  const [count, setCount] = useState(0);

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);

  return (
    <div>
      <p>Licznik: {count}</p>
```

```
      <button onClick={increment}>Zwiększ</button>
      <button onClick={decrement}>Zmniejsz</button>
    </div>
  );
}
```

W powyższym przykładzie używamy hooka `useState` do przechowywania stanu `count`, który początkowo ustawiamy na wartość 0. Następnie definiujemy funkcje `increment` i `decrement`, które modyfikują wartość stanu za pomocą funkcji `setCount`. W renderowaniu komponentu wyświetlamy wartość stanu `count` oraz przyciski, które umożliwiają zwiększanie i zmniejszanie wartości stanu.

Dzięki hookowi useState możemy łatwo korzystać ze stanu w ReactJS i tworzyć interaktywne komponenty, które reagują na zmiany wartości stanu.

## React Hooks

React Hooks to funkcje, które pozwalają na używanie stanu i innych funkcjonalności React w funkcjonalnych komponentach. Wcześniej, przed wprowadzeniem Hooksów, stan i lifecycle metody były dostępne tylko w klasowych komponentach.

Hooks umożliwiają korzystanie z wielu funkcjonalności React, takich jak stan, efekty uboczne (side effects), kontekst, referencje do elementów DOM i wiele innych, bez potrzeby pisania klasowych komponentów. Hooksy zostały wprowadzone w ReactJS 16.8 i są często wykorzystywane przez deweloperów jako alternatywa dla klasowych komponentów.

Najczęściej używanym hookiem jest `useState`, który umożliwia używanie stanu w funkcjonalnych komponentach. Innymi przykładami hooków są:

- `useEffect`: pozwala na wykonywanie efektów ubocznych, takich jak pobieranie danych z API, podczas renderowania komponentu.
- `useContext`: umożliwia korzystanie z kontekstu w funkcjonalnych komponentach.
- `useRef`: pozwala na tworzenie referencji do elementów DOM.
- `useReducer`: umożliwia tworzenie reduktorów stanu, podobnie jak w Reduxie.
- `useCallback` i `useMemo`: optymalizują wydajność komponentów poprzez unikanie niepotrzebnych renderowań.

Przykład użycia hooka useEffect:

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

const Example = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    // wykonywane po renderowaniu komponentu
    document.title = `Licznik: ${count}`;
  }, [count]);

  return (
    <div>
      <p>Licznik: {count}</p>
      <button onClick={() => setCount(count + 1)}>Zwiększ</button>
    </div>
```

```
  );
}
```

W powyższym przykładzie używamy hooka useEffect do zmiany tytułu strony na podstawie wartości stanu count. useEffect jest wykonywany po każdym renderowaniu komponentu, ale dzięki drugiemu argumentowi `[count]` jest wykonywany tylko wtedy, gdy wartość stanu count się zmienia.

## Pisanie Aplikacji w React.JS przy uzyciu 1 komponentu

Napisz aplikacje Chat, skladajaca sie z formularza i listy. Formularz ma zawierac 2 pola input na author i message. Nastepnie po wcisnieciu przycisku Wyslij, wyswietl wiadomosci w liscie. Dane pobieraj i zapisuj do `localStorage` lub `json-server`