Aplikacje mobilne

Podstawy JavaScript i React

Mateusz Pawełkiewicz

JavaScript i ES6:

JavaScript to język programowania, który jest podstawą większości interaktywnych stron internetowych. Jest to język interpretowany, co oznacza, że jest wykonywany przez przeglądarkę internetową bez potrzeby kompilacji. JavaScript jest językiem dynamicznym, co oznacza, że typy danych mogą być zmieniane w trakcie wykonywania programu.

ES6, znany również jako ECMAScript 2015, to wersja JavaScript, która wprowadziła wiele nowych funkcji i składni, które ułatwiają programowanie w JavaScript. Oto kilka kluczowych elementów składni ES6:

Zmienne let i const: ES6 wprowadził dwa nowe sposoby deklarowania zmiennych: let i const. let jest podobne do var, ale ma lepsze zarządzanie zakresem. const jest używane do deklarowania stałych.

```javascript
let name = "John";
const PI = 3.14159;
```

Funkcje strzałkowe: ES6 wprowadził nową składnię dla funkcji, czyli funkcje strzałkowe. Są one krótsze i mają lepsze zarządzanie kontekstem this.

```javascript
const add = (a, b) => a + b;
```

Łańcuchy szablonowe (Template Literal): ES6 wprowadził łańcuchy szablonowe, które umożliwiają wstawianie zmiennych bezpośrednio do ciągów znaków.

```javascript
let greeting = `Hello, ${name}!`;
```

Klasy: ES6 wprowadził składnię klasy, która jest bardziej zrozumiała dla programistów z doświadczeniem w innych językach obiektowych.

```javascript
class Person {
  constructor(name) {
    this.name = name;
  }

  greet() {
    return `Hello, ${this.name}!`;
  }
}
```

Promises: ułatwiają pracę z operacjami asynchronicznymi.

```javascript
let promise = new Promise((resolve, reject) => {
  // some async operation here
});
```

React

Komponenty: Są to niezależne jednostki kodu, które reprezentują część interfejsu użytkownika. Komponenty mogą być funkcjonalne lub klasowe.

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Stan: to dane, które są przechowywane w komponencie i mogą wpływać na to, co jest renderowane. W komponentach funkcyjnych, stan jest zarządzany za pomocą hooka useState.

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

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

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Props: to dane, które są przekazywane do komponentów od ich rodziców.

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

// Użycie komponentu z props
const element = <Welcome name="Sara" />;
```

Cykl życia komponentu: Są to metody, które są wywoływane w różnych punktach w życiu komponentu. W komponentach funkcyjnych w React, cykl życia komponentu jest zarządzany za pomocą hooka useEffect. Hook useEffect jest wywoływany po każdym renderowaniu komponentu, co odpowiada metodzie componentDidMount, componentDidUpdate, i componentWillUnmount w klasowych komponentach.

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

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

  useEffect(() => {
    document.title = `You clicked ${count} times`;

    return () => {
      document.title = 'React App';
    };
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Hooki: Są to funkcje, które pozwalają na korzystanie ze stanu i innych funkcji React bez pisania klasowego komponentu.

useState: Jest to najbardziej podstawowy hook, który pozwala na korzystanie ze stanu w komponentach funkcyjnych. Zwraca parę: bieżącą wartość stanu i funkcję, która pozwala go aktualizować.

```
const [count, setCount] = useState(0);
```

useEffect: Pozwala na wykonywanie efektów ubocznych w komponentach funkcyjnych. Może być używany do różnych celów, takich jak pobieranie danych, subskrypcje czy ręczne zmiany DOM. Jest to odpowiednik metod componentDidMount, componentDidUpdate, i componentWillUnmount w klasowych komponentach.

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
});
```

useContext: Pozwala na korzystanie z kontekstu bez konieczności opakowywania komponentu w Context.Consumer. Kontekst pozwala na przekazywanie danych przez drzewo komponentów bez konieczności przekazywania props przez każdy poziom ręcznie.

```
const value = useContext(MyContext);
```

useCallback: Zwraca memoizowaną wersję przekazanej funkcji, która zmienia się tylko, gdy zmieniają się zależności. Jest to przydatne, gdy przekazujemy funkcje jako props do zoptymalizowanych komponentów dzieci.

```
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```

useMemo: Zwraca memoizowaną wartość. Jest to przydatne do optymalizacji wydajności, gdy mamy kosztowne obliczenia.

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

Wszystkie te hooki są wbudowane w React i są dostępne bez konieczności instalowania dodatkowych pakietów.