MODUL PRAKTIKUM
MATAKULIAH PRAKTIKUM FULLSTACK DEVELOPER
S1 TEKNIK INFORMATIKA

PERTEMUAN 2
STATE, HOOKS DAN PROPS

SEKOLAH TINGGI ILMU KOMPUTER PGRI BANYUWANGI
(STIKOM PGRI BANYUWANGI)
2024

Materi

State, Hooks dan Props

1. State

Penjelasan Tentang State di React

State di React adalah objek khusus yang digunakan untuk menyimpan data atau informasi tentang komponen yang dapat berubah seiring waktu. State adalah elemen yang sangat penting dalam komponen berbasis kelas maupun komponen fungsional dengan hooks. Ketika state diubah, React akan merender ulang komponen untuk mencerminkan perubahan tersebut, membuat antarmuka pengguna tetap sinkron dengan data aplikasi.

State pada Komponen Kelas

Pada komponen berbasis kelas, state diinisialisasi dalam konstruktor dan diubah menggunakan metode `setState`.

Contoh:

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

class Counter extends Component {
  constructor(props) {
    super(props);
    // Inisialisasi state
    this.state = {
      count: 0
    };
  }
  // Metode untuk mengubah state
  increment = () => {
    this.setState({ count: this.state.count + 1 });
  }
  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}

export default Counter;
```

Penjelasan:
- Konstruktor: Menginisialisasi state dengan nilai awal.
- this.setState: Metode yang digunakan untuk mengubah nilai state. Ini akan memicu perenderan ulang komponen dengan state yang baru.

State pada Komponen Fungsional dengan Hooks

Pada komponen fungsional, kita menggunakan `useState` hook untuk mengelola state. Hook ini diperkenalkan di React 16.8 dan memungkinkan penggunaan state tanpa perlu menggunakan kelas.

Contoh:

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

const Counter = () => {
  // Menggunakan useState untuk mendeklarasikan state
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
};

export default Counter;
```

Penjelasan:
- useState: Hook yang mengembalikan sepasang nilai: state saat ini dan fungsi untuk memperbaruinya. Dalam contoh ini, `count` adalah nilai state dan `setCount` adalah fungsi untuk memperbarui `count`.
- setCount: Fungsi yang digunakan untuk mengubah nilai state. Ketika dipanggil, ini akan merender ulang komponen dengan state yang baru.

2. Hooks

Penjelasan Tentang Hooks di React

Hooks adalah fitur yang diperkenalkan di React 16.8 yang memungkinkan Anda menggunakan state dan fitur React lainnya dalam komponen fungsional. Hooks adalah fungsi yang dimulai dengan kata "use". Mereka memberikan cara baru untuk mengelola logika komponen tanpa harus menggunakan kelas.

Mengapa Hooks?

Sebelum Hooks, komponen fungsional hanyalah komponen "dumb" yang tidak memiliki state atau lifecycle methods. Semua logika state dan lifecycle harus ditulis dalam komponen kelas. Hooks memungkinkan komponen fungsional untuk memiliki kemampuan ini.

Jenis-jenis Hooks

Beberapa Hooks yang paling umum digunakan adalah:

1. useState: Mengelola state lokal dalam komponen fungsional.
2. useEffect: Menjalankan efek samping seperti data fetching, subscription, atau manual DOM manipulation.
3. useContext: Mengakses nilai dari Context API.
4. useReducer: Mengelola state yang lebih kompleks menggunakan reducer function.
5. useRef: Mengakses elemen DOM atau menyimpan nilai persisten yang tidak memicu rerender.

Contoh Penggunaan Hooks

1. useState
Hook `useState` digunakan untuk menambahkan state ke komponen fungsional.

Contoh:

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

const Counter = () => {
  // Mendeklarasikan state "count" dengan nilai awal 0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default Counter;
```

Penjelasan:
- useState: Mengembalikan sepasang nilai: nilai state saat ini (`count`) dan fungsi untuk memperbarui nilai tersebut (`setCount`).
- setCount: Fungsi yang digunakan untuk mengubah nilai state.

2. useEffect

Hook `useEffect` digunakan untuk menjalankan efek samping dalam komponen fungsional. Efek samping ini bisa berupa fetching data, mengatur timer, atau melakukan operasi DOM.

Contoh:

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

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

  useEffect(() => {
    const interval = setInterval(() => {
      setCount(prevCount => prevCount + 1);
    }, 1000);

    // Cleanup function untuk menghentikan interval saat komponen unmount
    return () => clearInterval(interval);
  }, []);

  return (
    <div>
      <p>Count: {count}</p>
    </div>
  );
};

export default Timer;
```

Penjelasan:

- useEffect: Hook yang menjalankan efek samping. Mengambil dua argumen: fungsi yang menjalankan efek samping dan array dependencies.
- Array dependencies: Daftar variabel yang digunakan oleh efek. Efek hanya akan dijalankan kembali jika salah satu dari variabel ini berubah. Jika array kosong (`[]`), efek hanya dijalankan sekali setelah komponen dirender.

3. useContext

Hook `useContext` digunakan untuk mengakses nilai dari Context API tanpa harus menggunakan komponen `Consumer`.

Contoh:

```
import React, { createContext, useContext } from 'react';

// Membuat Context
const UserContext = createContext();

const UserProvider = ({ children }) => {
```

```
  const user = { name: 'Alice', age: 25 };

  return (
    <UserContext.Provider value={user}>
      {children}
    </UserContext.Provider>
  );
};

const UserProfile = () => {
  const user = useContext(UserContext);

  return (
    <div>
      <p>Name: {user.name}</p>
      <p>Age: {user.age}</p>
    </div>
  );
};

const App = () => {
  return (
    <UserProvider>
      <UserProfile />
    </UserProvider>
  );
};

export default App;
```

Penjelasan:

- createContext: Membuat konteks baru.
- useContext: Mengakses nilai dari konteks tanpa menggunakan komponen `Consumer`.

4. useReducer

Hook `useReducer` digunakan untuk mengelola state yang kompleks, mirip dengan konsep Redux.

Contoh:

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

const initialState = { count: 0 };

const reducer = (state, action) => {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
```

```
    default:
      return state;
  }
};

const Counter = () => {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>Increment</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>
    </div>
  );
};

export default Counter;
```

Penjelasan:

- useReducer: Mengelola state dan memberikan cara untuk mengirimkan aksi untuk memperbarui state. Mengambil dua argumen: reducer function dan initial state.
- dispatch: Fungsi yang digunakan untuk mengirimkan aksi ke reducer.

5. useRef

Hook `useRef` digunakan untuk mengakses elemen DOM atau menyimpan nilai persisten yang tidak memicu rerender.

Contoh:

```
import React, { useRef } from 'react';
const TextInput = () => {
  const inputRef = useRef();

  const focusInput = () => {
    inputRef.current.focus();
  };
  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
};
export default TextInput;
```

Penjelasan:

- useRef: Mengembalikan objek ref yang dapat disematkan ke elemen DOM melalui atribut `ref`.
- inputRef.current: Menunjuk ke elemen input, memungkinkan kita untuk memanggil metode DOM langsung pada elemen tersebut.

3. Props

Penjelasan Tentang Props di React

Props (singkatan dari properties) adalah cara untuk mengirim data dari satu komponen ke komponen lainnya dalam React. Props memungkinkan komponen untuk menerima data yang dapat digunakan untuk merender konten secara dinamis. Props bersifat read-only, artinya komponen yang menerima props tidak dapat mengubah nilai props tersebut.

Props biasanya digunakan untuk:
1. Menyampaikan data dari komponen induk (parent) ke komponen anak (child).
2. Membuat komponen lebih modular dan reusable dengan memungkinkan mereka menerima input yang berbeda.

Cara Menggunakan Props

1. Mengirim Props dari Komponen Induk ke Komponen Anak

Props dikirim dari komponen induk ke komponen anak melalui atribut JSX.

Contoh:

```
import React from 'react';

// Komponen Anak
const Greeting = (props) => {
  return <h1>Hello, {props.name}!</h1>;
};

// Komponen Induk
const App = () => {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
    </div>
  );
};

export default App;
```

Penjelasan:
- Komponen `Greeting` menerima prop `name` dan menggunakan nilai tersebut untuk menampilkan pesan sapaan.
- Komponen `App` mengirimkan nilai `name` kepada `Greeting` dengan atribut JSX.

2. Mengakses Props dalam Komponen

Props dapat diakses dalam komponen anak menggunakan `props` yang diteruskan sebagai parameter ke fungsi komponen.

Contoh:

```
import React from 'react';

        // Komponen Anak
        const UserInfo = (props) => {
          return (
            <div>
              <p>Name: {props.name}</p>
              <p>Age: {props.age}</p>
            </div>
          );
        };

        // Komponen Induk
        const App = () => {
          return (
            <div>
              <UserInfo name="Alice" age={25} />
              <UserInfo name="Bob" age={30} />
            </div>
          );
        };

export default App;
```

Penjelasan:

- Komponen `UserInfo` menerima dua props: `name` dan `age`.
- Komponen `App` mengirimkan nilai `name` dan `age` kepada `UserInfo`.

3. **Default Props**

Anda dapat mendefinisikan props default yang akan digunakan jika tidak ada nilai props yang diberikan oleh komponen induk.

Contoh:

```
        import React from 'react';

        // Komponen Anak
        const Greeting = (props) => {
          return <h1>Hello, {props.name}!</h1>;
        };

        // Default props
        Greeting.defaultProps = {
          name: 'Guest',
        };
```

```
// Komponen Induk
const App = () => {
  return (
    <div>
      <Greeting />
      <Greeting name="Alice" />
    </div>
  );
};

export default App;
```

Penjelasan:
- `Greeting.defaultProps` mendefinisikan nilai default untuk prop `name`.
- Jika tidak ada nilai `name` yang diberikan, `Greeting` akan menggunakan nilai default `'Guest'`.

4. Prop Types

React menyediakan paket `prop-types` untuk memvalidasi tipe data props yang diterima oleh komponen.

Contoh:

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

// Komponen Anak
const Greeting = (props) => {
  return <h1>Hello, {props.name}!</h1>;
};

// Validasi tipe props
Greeting.propTypes = {
  name: PropTypes.string.isRequired,
};

// Komponen Induk
const App = () => {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting /> {/* Akan menghasilkan peringatan karena prop 'name' tidak diberikan */}
    </div>
  );
};

export default App;
```

Penjelasan:

- `Greeting.propTypes` mendefinisikan bahwa prop `name` harus berupa string dan wajib ada (`isRequired`).
- Jika `name` tidak diberikan atau bukan string, React akan menampilkan peringatan di konsol selama pengembangan.

Praktikum:

1. Buatlah folder baru Bernama pertemuan 2 didalam folder fullstack pada pertemuan 1.
2. Inisiasi folder dengan perintah "***npm init -y***"
3. Buatlah project react dengan vite.
4. Buatlah aplikasi todo list untuk implementasi ketiga hal diatas.
5. Silahkan upload ke github dengan perintah "***git push origin***"
6. Buatlah laporan dan upload ke elearning

<Selamat Mengerjakan>