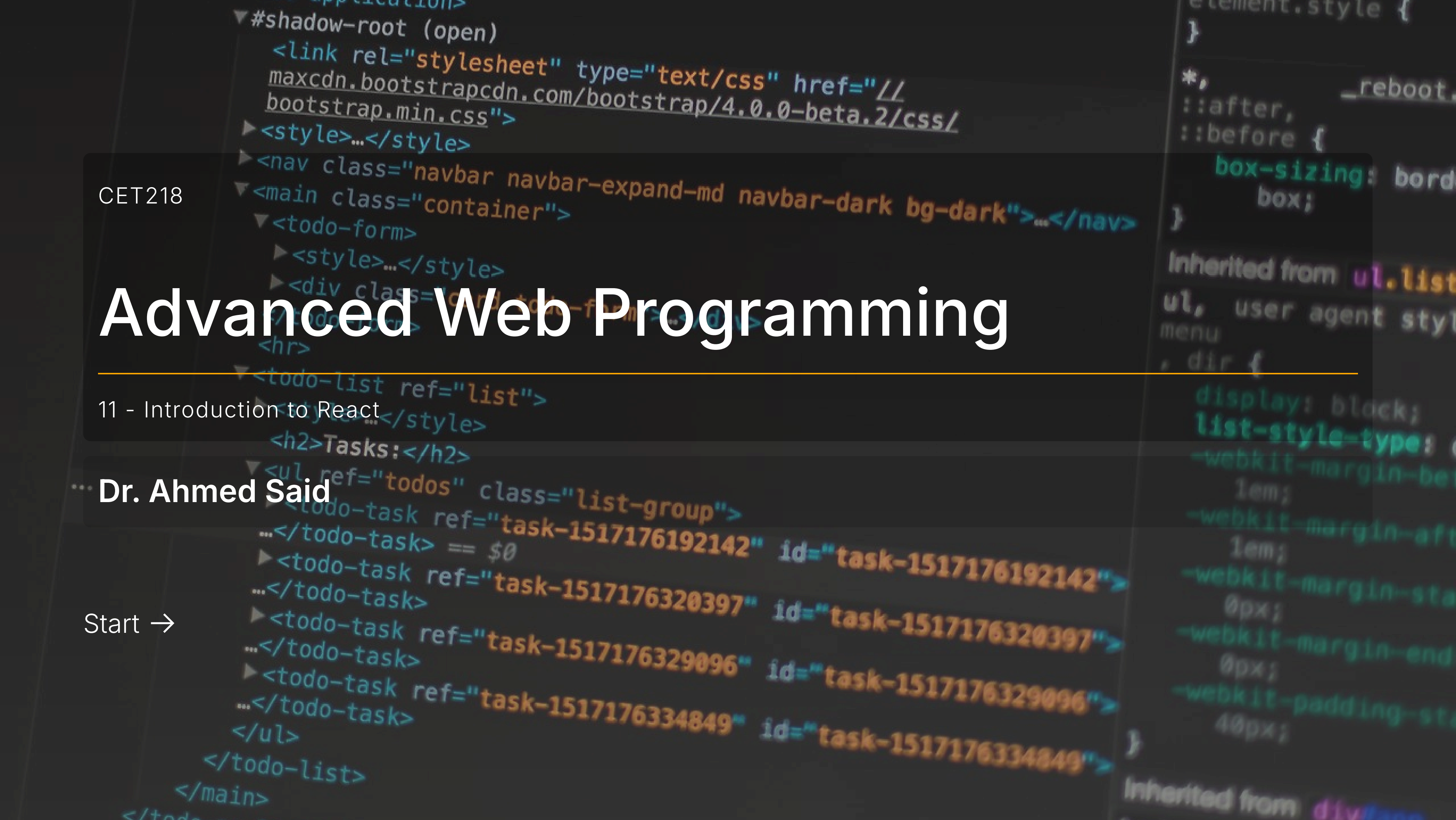


CET218

Advanced Web Programming

11 - Introduction to React

Dr. Ahmed Said

Start →

What is React?

- A JavaScript library for building user interfaces UI (front-end) using Components.
- React Created by Facebook in 2011 by Jordan Walke, open-sourced in 2013.
- React is also known as **React.js** or **ReactJS**.

How Does React Work?

- **Component-Based**: UI is built using reusable components.
- **React creates a VIRTUAL DOM in memory**.
- Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.
- When the state of an object changes, React updates only that object in the real DOM (**React only changes what needs to be changed!**).
- This makes React fast and efficient.
- **Unidirectional Data Flow**: Data flows in one direction, making it easier to understand and debug.

Why Use React?

- **Reusability**: Components can be reused, reducing code duplication.
- **Efficiency**: Virtual DOM minimizes browser updates for faster rendering.
- **Ecosystem**: Large community with tools like Redux and React Router.
- **Community**: Large community with extensive documentation, tutorials, and third-party libraries.
- **Cross-Platform**: React can be used for web (React), mobile (React Native), and desktop applications (Electron).

Getting Started with React

1. Directly in HTML

2. React Environment

Getting Started with React

- **Directly in HTML**: Include React and ReactDOM via CDN.
- The quickest way to start is to include React and ReactDOM via CDN in your HTML file.

```html
<script src="https://unpkg.com/react@17/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
```

- These two scripts are all you need to get started with React.
- These CDN links are for development purposes only. For production, use the minified versions.
- You will need to include Babel to transpile JSX (Write JSX syntax) into JavaScript.

```html
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

Getting Started with React

```html
<!DOCTYPE html>
<html>
  <head>
    <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  </head>
  <body>
    <div id="mydiv"></div>
    <script type="text/babel">
      function Hello() {
        return <h1>Hello World!</h1>;
      }

      const container = document.getElementById('mydiv');
      const root = ReactDOM.createRoot(container);
      root.render(<Hello />)
    </script>

  </body>
</html>
```

Setting Up React Environment

- This way of using React can be OK for testing purposes, but for production you will need to set up a React environment.
- **Prerequisites**:
  1. Install Node.js and npm.
  2. Install Create React App globally.
  3. Create a new React app.
- **Install Node.js and npm**:
  - Download and install from Node.js website.
  - Verify installation:

```bash
node -v
npm -v
```

Setting Up React Environment

- **Install Create React App**:
  - Create React App is a command-line tool to set up a new React project with a good default configuration.
  - Install it globally:

```bash
npm install -g create-react-app
```

  - Verify installation:

```bash
create-react-app --version
```

  - Uninstall it globally:

```bash
npm uninstall -g create-react-app
```

Setting Up React Environment

- **Create a New React App**:
  - Use Create React App to create a new project:

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

  - This command creates a new directory called `my-app` with all the necessary files and dependencies.

- **Run the React Application**:
  - Change to the project directory:

    ```bash
    cd my-app
    ```

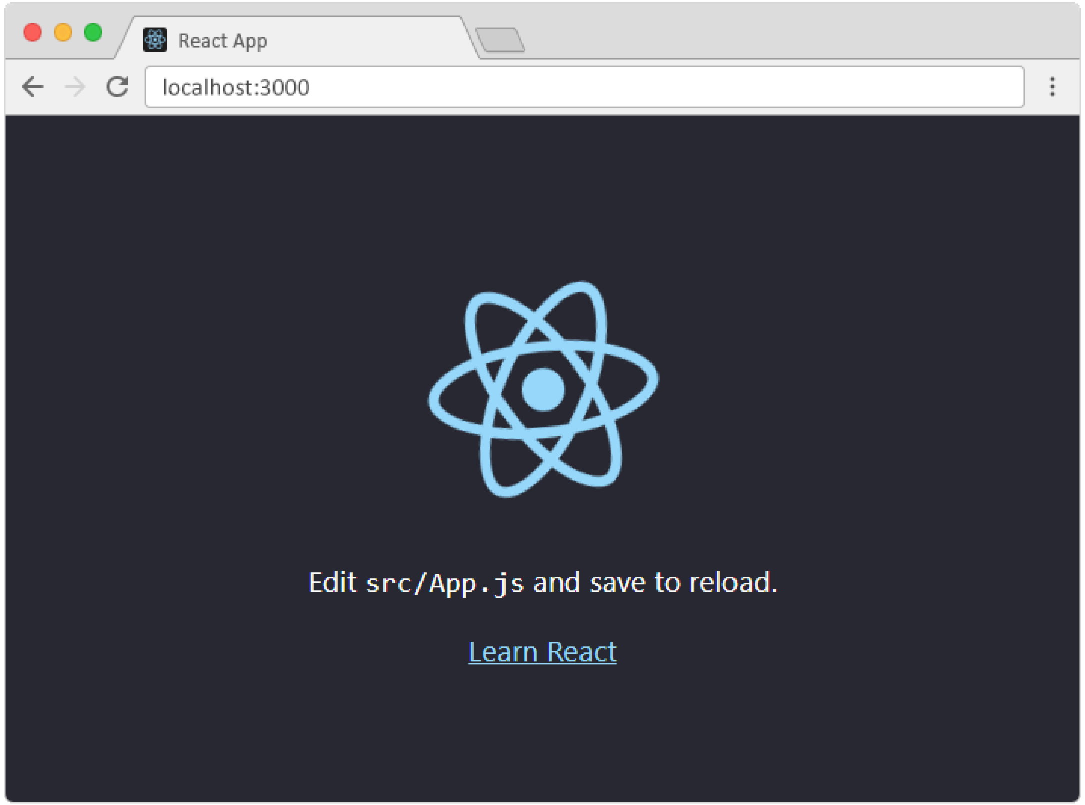
  - Start the development server:

    ```bash
    npm start
    ```

React Project Structure

- `node_modules/` : Contains all the dependencies. Don’t modify this folder.
- `package.json` : Lists project dependencies and scripts.
- `public/` : Contains static files (e.g., `index.html`).
    - `public/index.html` : Main HTML file.
    - `public/favicon.ico` : Favicon for the application.
    - `public/manifest.json` : Metadata for the application.
    - `public/robots.txt` : Instructions for web crawlers.
    - `public/logo192.png` : Logo for the application.
    - `public/logo512.png` : Logo for the application.
- `src/` : Contains React components and application logic.
    - `src/index.js` : Entry point for the React application.’
    - `src/App.js` : Main application component.
    - `src/App.css` : Styles for the application.
    - `src/index.css` : Global styles for the application.
    - `src/reportWebVitals.js` : Performance measurement.
    - `src/setupTests.js` : Setup for testing.
    - `src/logo.svg` : Logo for the application.
    - `src/App.test.js` : Test file for the main application component.

React Project Structure



Edit `src/App.js` and save to reload.

Learn React

React Project Structure

```js
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

React Project Structure

- `import`: Importing modules and styles.
- `function App()`: Main application component.
- `return`: JSX syntax for rendering UI.
- `export default App`: Exporting the component for use in other files.

Modifying the App Component

- Replace the content of `src/App.js` with the following code:

```js
function App() {
  return (
          <div className="App">
            <h1>Hello World!</h1>
          </div>
  );
}

export default App;
```

- This will render "Hello World!" on the page.`
- Notice that the changes are visible immediately after you save the file, you do not have to reload the browser!
- This is called **Hot Reloading** and is one of the features of Create React App.
- You can also use the `npm run build` command to create a production build of your application. This will create a `build` folder with all the necessary files for deployment.

index.js

```js
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
reportWebVitals();
```

- `ReactDOM.createRoot` : Creates a root for the React application.
- `root.render` : Renders the main application component.
- `<React.StrictMode>` : A wrapper for highlighting potential problems in an application.
- `reportWebVitals` : A function for measuring performance in the application.

React Render

- React uses a virtual DOM to optimize rendering.
- React renders HTML to the web page using the `createRoot()` and its method `render()`.
- The `createRoot` Function
  - The `createRoot()` function takes one argument, an HTML element.
  - The purpose of the function is to define the HTML element where a React component should be displayed.
- The `render` Method
  - The `render()` method takes one argument, a React component that should be rendered.
  - The purpose of the method is to display the React component in the HTML element defined by the `createRoot()` function.

React Render

- React uses a virtual DOM to optimize rendering.
- React renders HTML to the web page using the `createRoot()` and its method `render()`.
- The `createRoot` Function
  - The `createRoot()` function takes one argument, an HTML element.
  - The purpose of the function is to define the HTML element where a React component should be displayed.
- The `render` Method
  - The `render()` method takes one argument, a React component that should be rendered.
  - The purpose of the method is to display the React component in the HTML element defined by the `createRoot()` function.
- But render where?

React Render

- React uses a virtual DOM to optimize rendering.
- React renders HTML to the web page using the `createRoot()` and its method `render()`.
- The `createRoot` Function
  - The `createRoot()` function takes one argument, an HTML element.
  - The purpose of the function is to define the HTML element where a React component should be displayed.
- The `render` Method
  - The `render()` method takes one argument, a React component that should be rendered.
  - The purpose of the method is to display the React component in the HTML element defined by the `createRoot()` function.
- But render where?
- There is another folder in the root directory of your React project, named `public`. In this folder, there is an `index.html` file.

index.html

```html
...
<link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
<title>React App</title>
</head>
<body>
<noscript>You need to enable JavaScript to run this app.</noscript>
<div id="root"></div>

</body>
...
```

- The root node is the HTML element where you want to display the result.
- It is like a container for content managed by React.
- **It does NOT have to be a** `<div>` **element** and **it does NOT have to have the** `id='root'`:

Core Concepts

1. **Components**: Reusable UI building blocks.
2. **JSX**: HTML-like syntax for writing UI in JavaScript.
3. **Props**: Data passed between components.
4. **State**: Dynamic data managed within components.
5. **Hooks**: Functions for adding state and features to functional components.

JSX

React JSX

- **JSX** stands for JavaScript XML.
- JSX allows us to write HTML in React.
- JSX makes it easier to write and add HTML in React.

Why Use JSX?

- JSX looks like HTML, but it is actually syntactic sugar for `React.createElement`.
- JSX makes code easier to understand and write.
- Browsers cannot read JSX directly; it must be transpiled (e.g., by Babel) to JavaScript.

JSX Example

```jsx
const myElement = <h1>Hello, world!</h1>;
```

- This JSX code is transformed to:

```js
const myElement = React.createElement('h1', null, 'Hello, world!');
```

- The `React.createElement` function creates a React element.
    - The first argument is the type of element (e.g., `h1`), the second argument is the props (attributes), and the third argument is the children (content).
    - The `null` value indicates that there are no props for this element.
    - The `Hello, world!` string is the content of the `h1` element.

Embedding Expressions in JSX

- You can embed JavaScript expressions inside curly braces `{}` in JSX.

```jsx
const name = "Alice";
const greeting = <h1>Hello, {name}!</h1>;
```

JSX Must Have One Parent Element

- JSX expressions must have one parent element.

```jsx
// Correct
return (
  <div>
    <h1>Title</h1>
    <p>Paragraph</p>
  </div>
);

// Incorrect
return (
  <h1>Title</h1>
  <p>Paragraph</p>
);
```

JSX Attributes and Styling

- Use `className` instead of `class`.
- Use camelCase for event handlers and style properties.

```jsx
const element = <h1 className="header" style={{color: "blue"}}>Hello</h1>;
```

- **JSX is Optional**
- You do not have to use JSX, but it is recommended for readability and convenience.

```js
// Without JSX
const element = React.createElement('h1', {className: 'header'}, 'Hello');
```

Components

What is a Component?

- Components are the building blocks of a React application.
- They are reusable pieces of code that return a React element to be rendered to the page.
- Components are like JavaScript functions.
- They accept arbitrary inputs (called **props**) and return React elements describing what should appear on the screen.
- Components let you split the UI into independent, reusable pieces.
- **Types**:
  - **Functional Components**: JavaScript functions returning JSX, preferred for simplicity.
  - **Class Components**: ES6 classes, used in legacy code.
- **Key Features**:
  - Can accept **props** for customization.
  - Can manage **state** for dynamic behavior.
  - Support **composition** for building complex UIs.

Types of Components

- **Functional Components**
  - The most common and recommended way to write components in modern React.
  - Simple JavaScript functions that return JSX.
  - Can use React Hooks for state and lifecycle features.
- **Class Components**
  - Use ES6 classes.
  - Used in older React codebases.
  - Can have state and lifecycle methods.

Functional Component Example

```jsx
function Greeting(props) {
  return <h2>Hello, {props.name}!</h2>;
}

// Usage:
<Greeting name="Alice" />
```

- **Best Practice**: Use functional components for all new code.

Class Component Example

```jsx
import React from 'react';

class Welcome extends React.Component {
  render() {
    return <h2>Welcome, {this.props.name}!</h2>;
  }
}

// Usage:
<Welcome name="Bob" />
```

- **Note**: Prefer functional components unless you need legacy features.

Rendering Multiple Components

- You can use components inside other components.

```jsx
function App() {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
      <Greeting name="Charlie" />
    </div>
  );
}
```

- This will render three greetings on the page.
- Each component can have its own props and state.

Component Props

- **Props** are inputs to components.
- Passed as attributes in JSX.
- Props are **read-only**.

```jsx
function Car(props) {
  return <h2>I am a {props.brand}!</h2>;
}

// Usage:
<Car brand="Toyota" />
```

- **Destructuring Props**: You can destructure props for cleaner code.

```jsx
function Car({ brand }) {
  return <h2>I am a {brand}!</h2>;
}
```

Default Props

- You can set default values for props.

```jsx
function Button({ label = "Click Me" }) {
  return <button>{label}</button>;
}
```

- If no `label` prop is passed, it defaults to "Click Me".

- You can also set default props for class components.

```jsx
class Button extends React.Component {
  static defaultProps = {
    label: "Click Me"
  };

  render() {
    return <button>{this.props.label}</button>;
  }
}
```

Component State

- **State** is data managed within a component.
- Use the `useState` hook in functional components.

```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>
  );
}
```

- **State** is mutable and can change over time.
- State updates trigger re-renders of the component.

Passing Functions as Props

- You can pass functions to child components as props.

```jsx
function Child({ onButtonClick }) {
  return <button onClick={onButtonClick}>Click Me</button>;
}

function Parent() {
  const handleClick = () => alert('Button clicked!');
  return <Child onButtonClick={handleClick} />;
}
```

- This allows child components to communicate with parent components.
- The parent component can pass a function to the child, which the child can call when an event occurs.
- This is useful for handling events and managing state in a parent component based on actions in a child component.

Composing Components

- Build complex UIs by combining simple components.

```jsx
function Page() {
  return (
    <div>
      <Header />
      <Content />
      <Footer />
    </div>
  );
}
```

- Each component can be developed and tested independently.
- This promotes reusability and maintainability.

Best Practices for Components

- **Use functional components** and hooks for new code.
- **Keep components small and focused** on a single responsibility.
- **Use props for configuration** and state for dynamic data.
- **Name components with PascalCase** (e.g., `MyComponent`).
- **Extract repeated code** into reusable components.
- **Avoid side effects in render**; use hooks like `useEffect` for side effects.
- **Document your components** with comments or PropTypes.

Splitting Components into Files

- Place each component in its own file for better organization.

```text
src/
  components/
    Header.js
    Footer.js
    Content.js
```

- Example: `Header.js`

```jsx
function Header() {
  return <header>My App Header</header>;
}
export default Header;
```

Props vs State

- **Props**:
  - Immutable data passed from parent to child.
  - Example: `<Button label="Click" />`
- **State**:
  - Mutable data within a component, managed with `useState` or `this.state`.
  - Triggers re-renders on update.
- **Example**:

```jsx
function UserProfile(props) {
  const [age, setAge] = useState(20);
  return (
    <div>
      <p>Name: {props.name}</p>
      <p>Age: {age}</p>
      <button onClick={() => setAge(age + 1)}>Birthday</button>
    </div>
  );
}
```

Hooks

- **Purpose**: Add state and lifecycle features to functional components.
- **Common Hooks**:
    - `useState`: Manages state.
    - `useEffect`: Handles side effects.
- In React, side effects are operations that affect something outside the scope of the current function/component, such as:
    - Fetching data from an API
    - Subscribing to events
    - Manually manipulating the DOM
    - Setting up timers

Hooks

- **Example**:

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

function Timer() {
  const [seconds, setSeconds] = useState(0);
  useEffect(() => {
    const interval = setInterval(() => setSeconds(s => s + 1), 1000);
    return () => clearInterval(interval);
  }, []);
  return <p>Seconds: {seconds}</p>;
}
```

Full Working Example - Todo List

A Todo List app demonstrating components and JSX:

- **Features**: Add, delete, and toggle todos.
- **Components**: `App`, `TodoList`, `AddTodo`, `TodoItem`.

Todo List - App Component

jsx

```jsx
import React, { useState } from 'react';
import AddTodo from './AddTodo';
import TodoList from './TodoList';

function App() {
  const [todos, setTodos] = useState([]);

  const addTodo = (text) => {
    setTodos([...todos, { text, done: false }]);
  };

  const deleteTodo = (index) => {
    setTodos(todos.filter((_, i) => i !== index));
  };

  const toggleDone = (index) => {
    setTodos(
      todos.map((todo, i) =>
        i === index ? { ...todo, done: !todo.done } : todo
      )
    );
  };
```

```jsx
  return (
    <div className="container">
      <h1>Todo List</h1>
      <AddTodo addTodo={addTodo} />
      <TodoList todos={todos} deleteTodo={deleteTodo} toggleDone={toggleDone}
    </div>
  );
}

export default App;
```

Todo List - AddTodo Component

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

function AddTodo({ addTodo }) {
  const [text, setText] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    if (text.trim()) {
      addTodo(text);
      setText('');
    }
  };

  return (
    <div>
      <input
        type="text"
        value={text}
        onChange={(e) => setText(e.target.value)}
        placeholder="Add a new todo"
      />
      <button onClick={handleSubmit}>Add</button>
    </div>
  );
}
```

Todo List - TodoList and TodoItem Components

```jsx
// TodoList.js
import React from 'react';
import TodoItem from './TodoItem';

function TodoList({ todos, deleteTodo, toggleDone }) {
  return (
    <ul>
      {todos.map((todo, index) => (
        <TodoItem
          key={index}
          todo={todo}
          index={index}
          deleteTodo={deleteTodo}
          toggleDone={toggleDone}
        />
      ))}
    </ul>
  );
}

export default TodoList;
```

```jsx
// TodoItem.js
import React from 'react';

function TodoItem({ todo, index, deleteTodo, toggleDone }) {
  return (
    <li style={{ textDecoration: todo.done ? 'line-through' : 'none' }}>
      <input
        type="checkbox"
        checked={todo.done}
        onChange={() => toggleDone(index)}
      />
      {todo.text}
      <button onClick={() => deleteTodo(index)}>Delete</button>
    </li>
  );
}

export default TodoItem;
```

Resources for Further Learning

- React Documentation
- W3Schools React Tutorial