React and Tailwind CSS

A Practical Guide to Modern Web Development Frameworks

Learning Objectives

By the end of this session, students will be able to:

- Understand the core concepts of React, including components, props, and state
- Create and structure a React application using Vite as a build tool
- Develop functional components and pass data between them
- Implement styling using Tailwind CSS
- Convert traditional CSS styling to Tailwind CSS utility classes
- Build a simple, interactive todo list application

Technical Requirements

Before starting, ensure you have the following software/tools installed and set up:

- **Node.js (Version 18.0+)** - The JavaScript runtime environment needed to run React applications
- **npm (Node Package Manager)** - Comes with Node.js and is used to install dependencies
- **Visual Studio Code (VSCode)** - Recommended code editor =.

Overview

This section takes a different approach by emphasizing collaboration and teamwork, key elements in any real-world business environment. Working in teams fosters creativity, leverages diverse perspectives, and mirrors the dynamic nature of professional settings. In business, teams combine individual strengths, leading to more innovative solutions and efficient problem-solving.

The goal here is to work together to build a simple Todo List application using React. You are encouraged to form groups of **two** or **three**, allowing for lively discussion, shared insights, and a richer learning experience.

Section 1: Introduction to React

This section will walk you step by step on how to create a simple todo list application using React. This builds upon what we have discussed in Lesson 4 and gives you hands-on experience with React.

Each task will begin with a brief explanation of the task and what is required. It may also contain an explanation of the main concepts of the section and then a task for you to complete. Please read this carefully before attempting the task.

Like in our previous sessions, I strongly recommend you get used to looking at documentation for information. Remember everything you need to know about React can be found online or in the official React documentation. So if you get stuck in a task, try to look it up before asking questions.

1. What is Node.js?

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It allows developers to run JavaScript on the server side, outside of a browser environment. For React development, Node.js is essential as it provides the **npm (Node Package Manager)** which we use to install and manage packages/dependencies for our projects.

2. What is Vite?

Vite (French for "quick") is a modern build tool that significantly improves the development experience for React applications. Vite offers faster startup times and hot module replacement, making development more efficient. In summary, Vite is used to bootstrap (create) react applications.

Task 1: Create a new React Application

1. Open your terminal or command prompt.

2. Navigate to the `'03_react_and_tailwindcss'` directory in the practicals folder.

3. Run the following command to create a new React project named **"my-todo-app".** Feel free to give it another name if you wish.

```
npm create vite@latest my-todo-app -- --template react
```

4. Navigate into your project directory (i.e the react app you just created):

```
cd my-todo-app
```

5. Install the app's dependencies using npm :

```
npm install
```

6. Start the project on a development server:

```
npm run dev
```

7. Open your browser and navigate to the URL shown in the terminal (typically http://localhost:5173). You have successfully created **your first react application**!

3. An Explanation of the Project Structure

```
my-todo-app
├── .gitignore           # Specifies files to ignore in version control
├── README.md            # Project documentation
├── eslint.config.js     # ESLint configuration for code linting
├── index.html           # Entry HTML file
├── package.json         # Project dependencies and scripts
├── public/              # For Static assets e.g images, svgs, etc
│   └── vite.svg
├── src/                 # Main Source code
│   ├── App.css          # Styles for App component
│   ├── App.jsx          # Main App component
│   ├── assets/          # Project assets
│   │   └── react.svg    # React logo
│   ├── index.css        # Global styles
│   └── main.jsx         # Application entry point
└── vite.config.js       # Vite configuration
```

The "src /" folder

Lets ignore all the other folders for now and focus on the **src** directory. This folder contains the source code for your entire web application. All new components you will be creating should be added to this folder.

Familiarise yourself with the code in the src folder. As you may have noticed, Vite automatically creates a simple counter program on the App component. Feel free to play around with the code in this folder, then– replace the code in App.jsx with the following;:

```
import './App.css'

function App() {

  return (
    <div>Your code goes here</div>
  )
}

export default App
```

Task 2: Create your first Component

4. Understanding React Components

Components are the building blocks of React applications. They are reusable, self-contained pieces of code that return JSX (JavaScript XML) to describe what should appear on the screen – it is easier to just think of it as HTML. Components can be either **function** components or **class** components, but modern React development primarily uses function components.

Why Use Components?

- **Modularity:** Break down complex UIs into manageable, independent pieces.
- **Reusability:** Use the same UI elements across different parts of your application. For example, a **<Button>** component can be reused on multiple pages without duplicating code.
- **Separation of Concerns:** Keep your code organized by isolating functionality and presentation.

How to Define a Component

When working with JavaScript in React, components are typically defined as Javascript functions and stored in files with a **`.jsx`** extension. These functions **return** JSX, which is then rendered as HTML on the page.

For example, the following code defines a simple **`Welcome`** component in a file named **`Welcome.jsx`:**

```
function Welcome(props) {
   return <h1>Hello! React is cool right?</h1>;
}
```

Your Tasks:

1. In your project's `src` directory, create a new file called `Header.jsx`.

2. Add the following code to create a simple Header component:

```
function Header() {
  return (
    <header>
      <h1>My Todo List Application</h1>
      <p>A simple React application for managing tasks</p>
    </header>
  );
}

export default Header;
```

3. Now, import and use this component in your `App.jsx`:

```
import Header from "./Header";
import "./App.css";

function App() {
  return (
    <div>
      <Header />
    </div>
  );
}

export default App;
```

4. See your changes in your browser: http://localhost:5173/. React has a **hot reload** feature, so whenever you make changes to the application, your changes are automatically reflected in the browser.

Task 3: Understanding Props

Props (short for **"properties"**) are a way to pass data from parent components to child components. They are read-only and help make your components reusable by allowing them to receive different data.

Props are passed to components like HTML attributes. For example:

```
<Header title="My Todo List" subtitle="Manage your tasks efficiently" />
```

And received in the component as an argument (in the shape of a javascript object).

```
function Header(props) {
  return (
    <header>
      <h1>{props.title}</h1>
      <p>{props.subtitle}</p>
    </header>
  );
}
```

Your Tasks:

1. Update the Header component to effectively use props.

2. Notice how this approach makes the Header component configurable. You can reuse the same component across multiple pages in your application without duplicating code – this is one of the major benefits of React components.

3. Create two new components **TaskItem.jsx** and **TodoList.jsx**.

4. Replace the contents of these components with the ones in the **solutions/my-todo-app** folder. The instructor will demonstrate how **State** and **Events** work in these components.

Section 2: TailwindCSS (Optional)

1. Introduction to TailwindCSS

TailwindCSS is a modern, utility-first CSS framework that enables rapid development of custom user interfaces. Unlike traditional CSS, which relies on separate stylesheets, TailwindCSS allows you to directly apply predefined classes to your HTML elements, simplifying the styling process significantly.

To understand the advantages of Tailwind, let's compare it with traditional CSS through a practical example.

Traditional CSS Approach:

Using traditional CSS often involves creating a separate CSS file and importing it into your React component.

Button.css file:

```css
.primary-button {
  padding: 0.5rem 1rem;
  background-color: #3b82f6;
  color: white;
  border-radius: 0.25rem;
  font-weight: 600;
}

.primary-button:hover {
  background-color: #2563eb;
}
```

Button.jsx component:

```jsx
// Button.jsx Component
import './Button.css';

function Button({ text }) {
  return <button className="primary-button">{text}</button>;
```

```
}
```

TailwindCSS Approach:

With TailwindCSS, you directly apply predefined utility classes within the JSX itself. There is no need for a separate CSS file.

Button.jsx component:

```
function Button({ text }) {
  return (
    <button className="px-4 py-2 bg-blue-500 text-white rounded
font-semibold hover:bg-blue-600">
      {text}
    </button>
  );
}
```

Notice how TailwindCSS eliminates the need for separate CSS files, simplifying your workflow by keeping styling directly within your HTML.

2. Installing Tailwind

For the tasks in this section, TailwindCSS is pre-installed. However, when setting up a new project, follow these steps to install Tailwind using Vite:

Detailed installation instructions can be found in the Tailwind documentation.

3. Understanding Tailwind's Utility Classes

TailwindCSS provides a comprehensive set of predefined utility classes that enable quick and efficient styling directly within HTML. These classes eliminate the need for writing custom CSS, making development faster and more streamlined.

The **`tailwind_tutorial.md`** file included in today's session materials contains an in-depth explanation of Tailwind's utility classes. Make sure to review this file for a deeper understanding of how to use Tailwind effectively.

Note: This is only a brief introduction — Tailwind offers many more utility classes beyond what we'll cover here. For a complete reference, explore the official Tailwind documentation.

Task 4: Styling Todolist with TailwindCSS

In this section, we will style our todo list application using TailwindCSS. The `styled-todo-list` folder currently contains your todo list application styled using Vanilla CSS. Your task is to convert this styling to use TailwindCSS utility classes.

Your Tasks:

1. Replace Vanilla CSS classes with appropriate Tailwind utility classes.
2. Ensure your todo list app remains visually consistent after migration.
3. Experiment with responsive designs using Tailwind classes to improve your application's usability across devices.

Recommended Approach to Completing the Task

Focus on restyling one component at a time. For example, when restyling `Header.jsx`:

1. Review and understand the current CSS styles in `Header.css`.
2. Replace each CSS style with equivalent Tailwind utility classes directly in the JSX.
3. Remove the original CSS file (`Header.css`) after successfully converting all styles.

Example:

Equivalent TailwindCSS implementation (Header.jsx):

```
function Header() {
  return (
    <header className="text-center mb-8 pb-4 border-b border-gray-200">
      <h1 className="text-4xl leading-tight text-slate-600 mb-2">
        My Todo List Application
      </h1>
      <p className="text-gray-600 text-lg">
        A simple React application for managing tasks
      </p>
    </header>
  );
```

```
}
```