**LAB 03**

**NAME: Trương Mạnh Nguyên**

**Student CODE: 23521065**

**Link GitHub:** https://github.com/AdeptCodee/UIT-WebProject/tree/main/LAB3

**Exercise 1: The React Paradigm**

**1. Conceptual Questions**

**a. Difference between Imperative and Declarative Approach (1 mark)**

- **Imperative approach:** You tell the computer *how* to do something, step by step.
- **Declarative approach:** You tell the computer *what* you want, and it figures out how to do it.

**Example (non-code):**

- *Imperative:* You tell the barista exactly how to make your coffee “Grind the beans, boil water, pour it over, add milk, stir."
- *Declarative:* You just say, "I'd like a cappuccino," and the barista handles all the steps for you.

→ **React uses the declarative approach**, letting developers describe *what the UI should look like* based on the state, rather than managing every DOM change manually.

**b. Three Key Benefits of Component-Based Architecture (5 marks)**

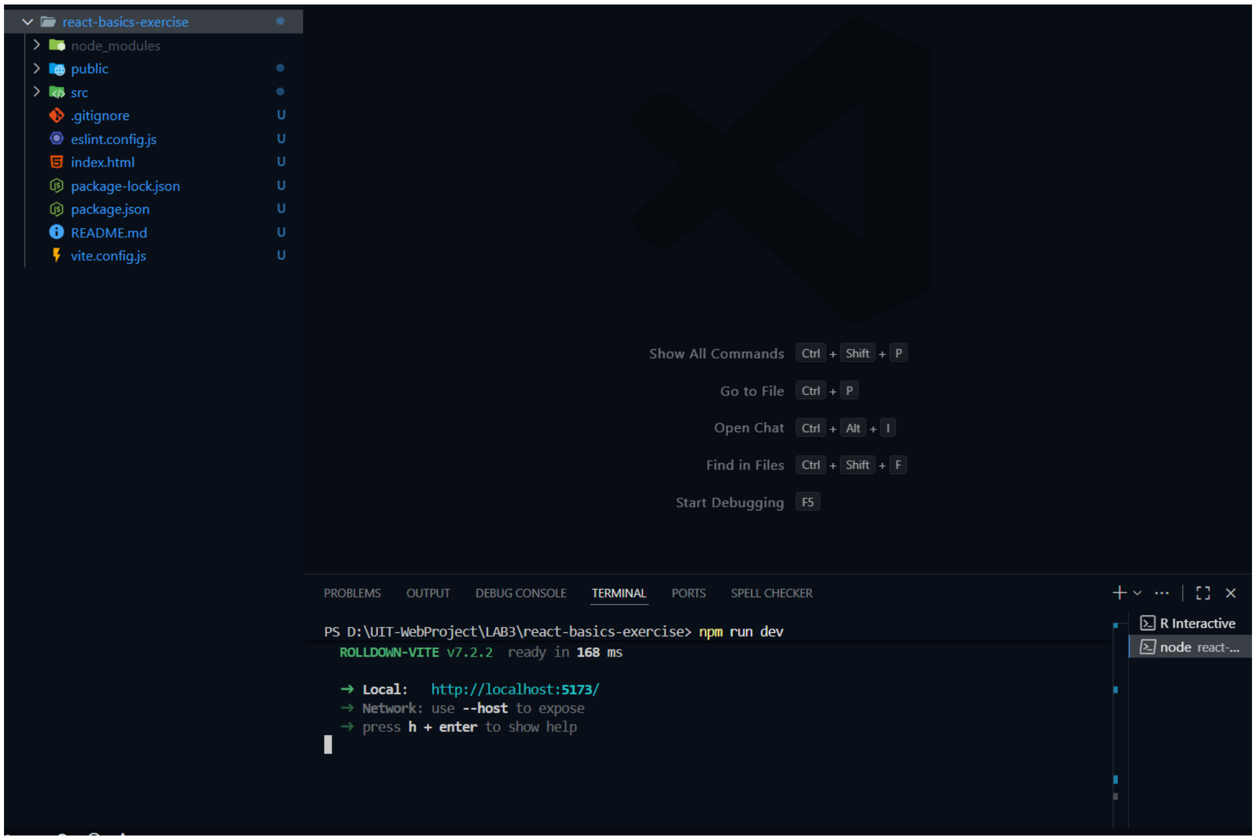
1. **Reusability:**
   Components can be reused across different parts of the app (e.g., buttons, forms).
   → Saves development time and ensures consistency.
2. **Maintainability:**
   Each component is self-contained, managing its own logic and style.
   → Easier to debug, update, or replace parts of the app without affecting others.
3. **Scalability:**
   Complex UIs can be built by combining smaller, simpler components.
   → Makes large applications easier to organize and extend over time.

**c. Role of the Virtual DOM & Reconciliation Process (1 mark)**

- The **Virtual DOM** is an in-memory copy of the real DOM that React uses to track changes.
- When the app's state updates, React first updates the Virtual DOM, then compares it to the previous version in a process called **reconciliation**.
- **Reconciliation** identifies only the parts of the real DOM that changed and updates them efficiently.

→ **This reduces direct manipulation of the real DOM**, which is slow, and results in **better performance** and smoother UI updates.

**Exercise 2: Setting Up a Modern React Development Environment**



```
PS D:\UIT-WebProject\LAB3\react-basics-exercise> npm run dev
  ROLLDOWN-VITE v7.2.2  ready in 168 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

|> [http://localhost:5173](http://localhost:5173) <|

That's the address where your React app runs in the browser.

- index.html: The main HTML file that loads your React app. It has a <div id="root"></div> where your React components will be rendered.
- src/main.jsx: The entry point of the React app. It tells React to render the <App /> component into the #root div in index.html. It also imports React, ReactDOM, and CSS.

- src/App.jsx: The main React component. It defines what content is displayed on the screen — like text, buttons, and other components. You can edit this file to change what appears in the browser.

**Exercise 5: Managing Component Memory with State**

**2. Conceptual Question:**

Use state.

Reasoning:

- The "Online / Offline" status changes inside the component based on user interaction (clicking a button). This means the data needs to be managed and updated within the component itself — which makes state the correct choice.
- Props are values passed from a parent component to a child and should not be modified by the child. If you tried to change a prop inside UserProfile, it would break React's one-way data flow.
- If multiple components need to share the same status (for example, a user list and a header showing the number of online users), you should lift the state up to a common parent and pass it down as props.
  But if only the UserProfile component needs to handle this status, managing it locally with state is simpler and more appropriate.