**Objectives**

* Explain the need and Benefits of React Context API

In React, data is typically passed from parent to child using props. However, in large applications, many components may need access to the same data (like authentication status, theme, language settings). Passing data through many layers of components (prop drilling) becomes inefficient and difficult to manage.

To solve this, React provides the Context API, which allows sharing data across the entire component tree without passing it explicitly at every level.

**Benefits:**

* **Avoids Prop Drilling:** Eliminates the need to pass props through every nested component.
* **Global State Sharing:** Makes global data (e.g., user info, settings) available to any component.
* **Cleaner Code:** Reduces clutter and improves readability.
* **Integrated with React:** No need for external libraries like Redux for simpler state management.
* **Component Independence:** Components can access required data without dependency on the hierarchy.
* Working with createContext()

The createContext() function is part of React’s Context API and is used to create a Context object.

**Steps to Use createContext():**

1. **Create Context:**

import { createContext } from 'react';

const MyContext = createContext();

1. **Provide the Context using Provider:**

<MyContext.Provider value={someValue}>

<Component />

</MyContext.Provider>

1. **Consume the Context using:**
   * useContext(MyContext) – a React hook (simplest way).
   * <MyContext.Consumer> – traditional method using a render function.

This pattern allows data to be shared across components without having to pass it through props manually.

* List the types of Router Components

React Router is used for building single-page applications (SPAs) with navigation.

|  |  |
| --- | --- |
| Component | Description |
| <BrowserRouter> | Uses HTML5 history API to sync the UI with the URL. Commonly used in modern web apps. |
| <HashRouter> | Uses the URL hash (#) to keep track of routes. Useful for static file servers. |
| <Routes> | Replaces Switch in React Router v6. Wraps <Route> elements and renders the first match. |
| <Route> | Defines a route’s path and the component to render. |
| <Link> | Allows navigation without reloading the page. |
| <NavLink> | Works like Link but adds styling when active. |
| <Navigate> | Redirects to a new location programmatically. |
| <Outlet> | Used for rendering nested routes in parent routes. |
| useNavigate() | Hook used for navigation within components (e.g., on button click). |

## **Notes**

Estimated time to complete this lab: **30 minutes.**

Developers of Apps Centric Solutions have created an employee management application which supports light and dark themes for the buttons. The current solution uses the react state and props to provide the theme name to be used from App component to Employee List component and from there to Employee Card component. Quality assurance team analyzed the solutions and found the technique being used to be a substandard one. React architect suggested to use the react context API to share the theme name with nested child components instead of passing them down using props from the parent component.

You are assigned the task of converting the application form props only to React Context API.

Application can be downloaded from below



1. Unzip the application and open it using VS Code
2. Go to terminal and execute *npm install* command to restore all the node modules



Figure 1: Restore node modules

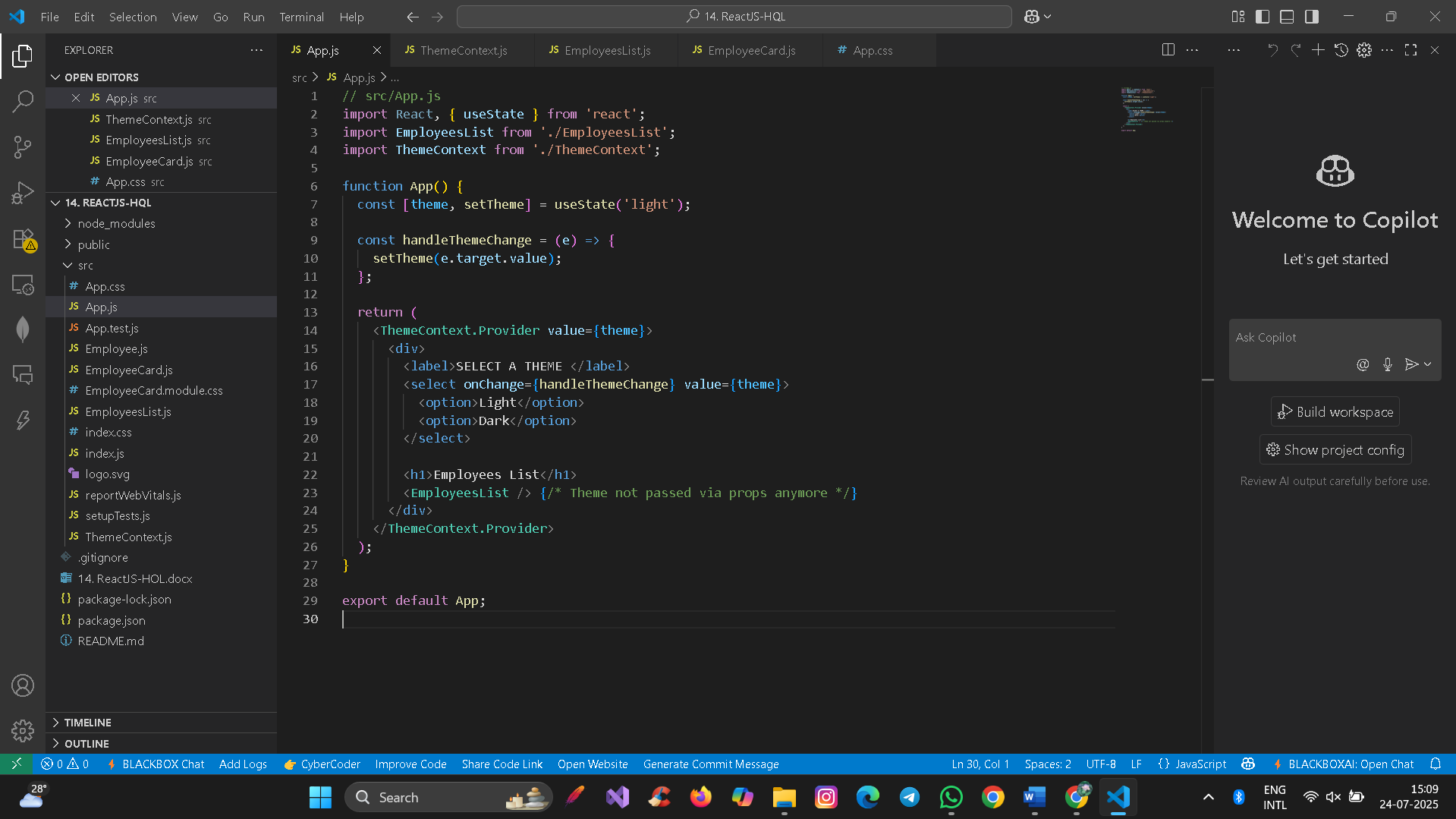
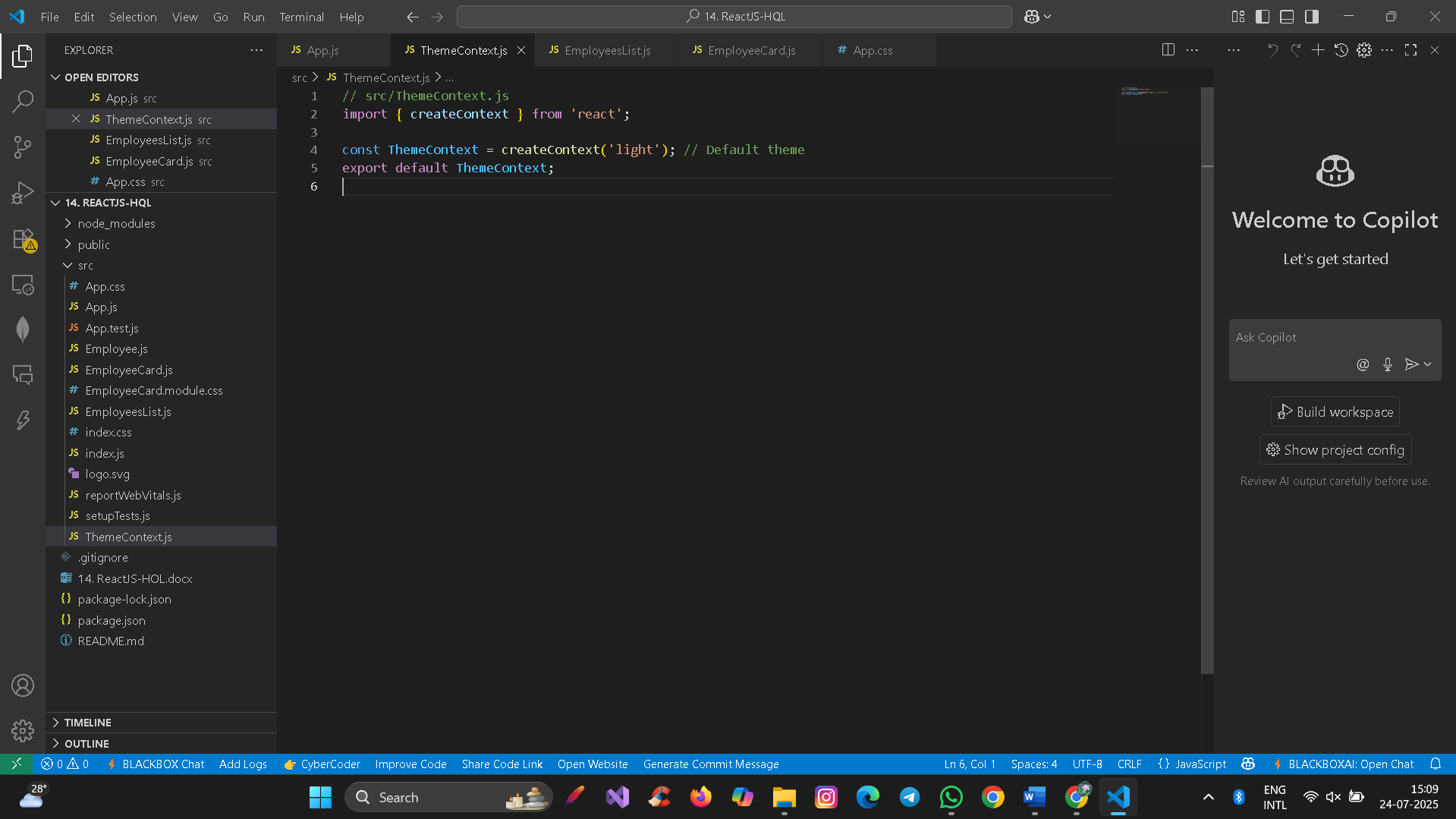
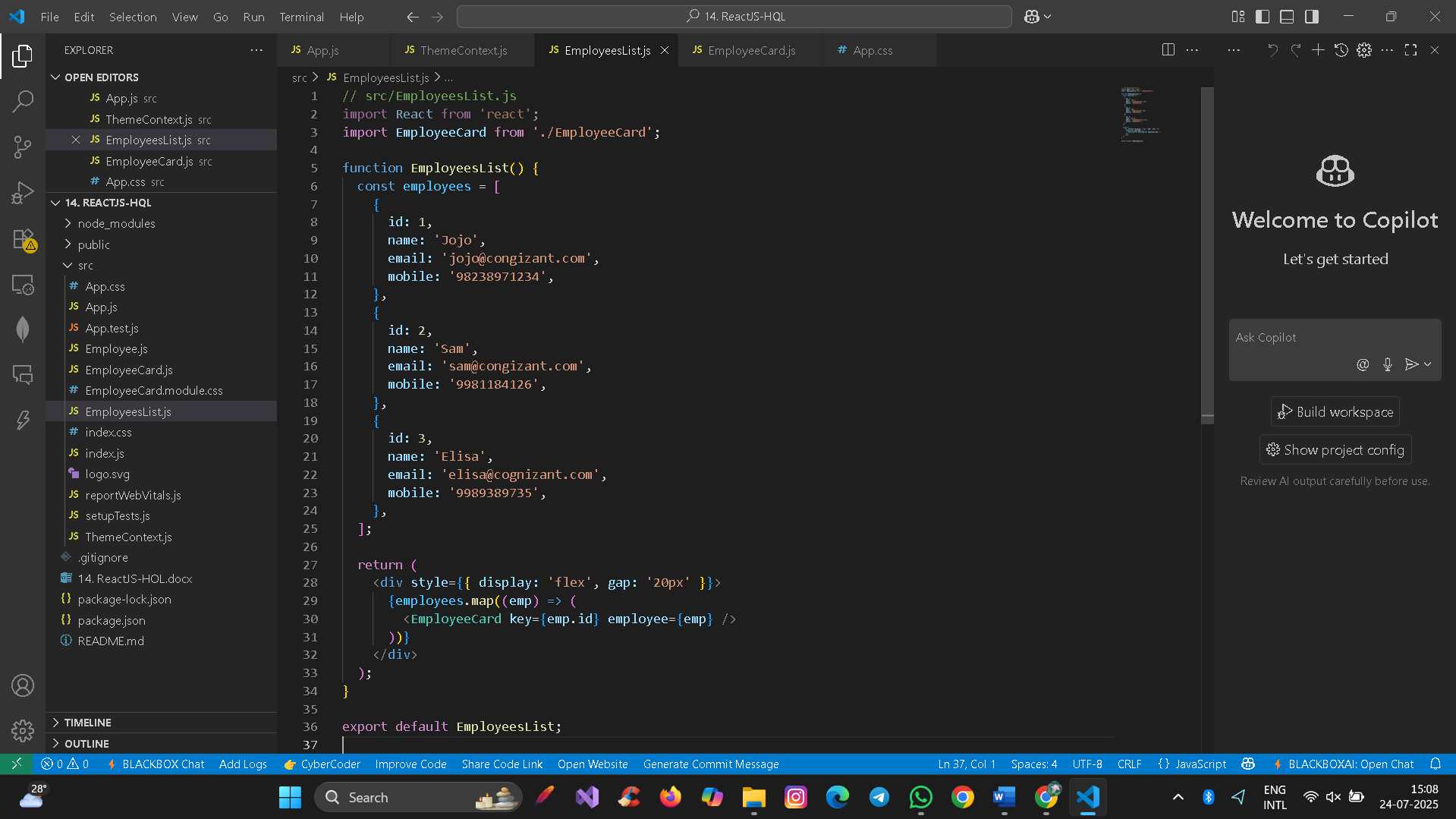
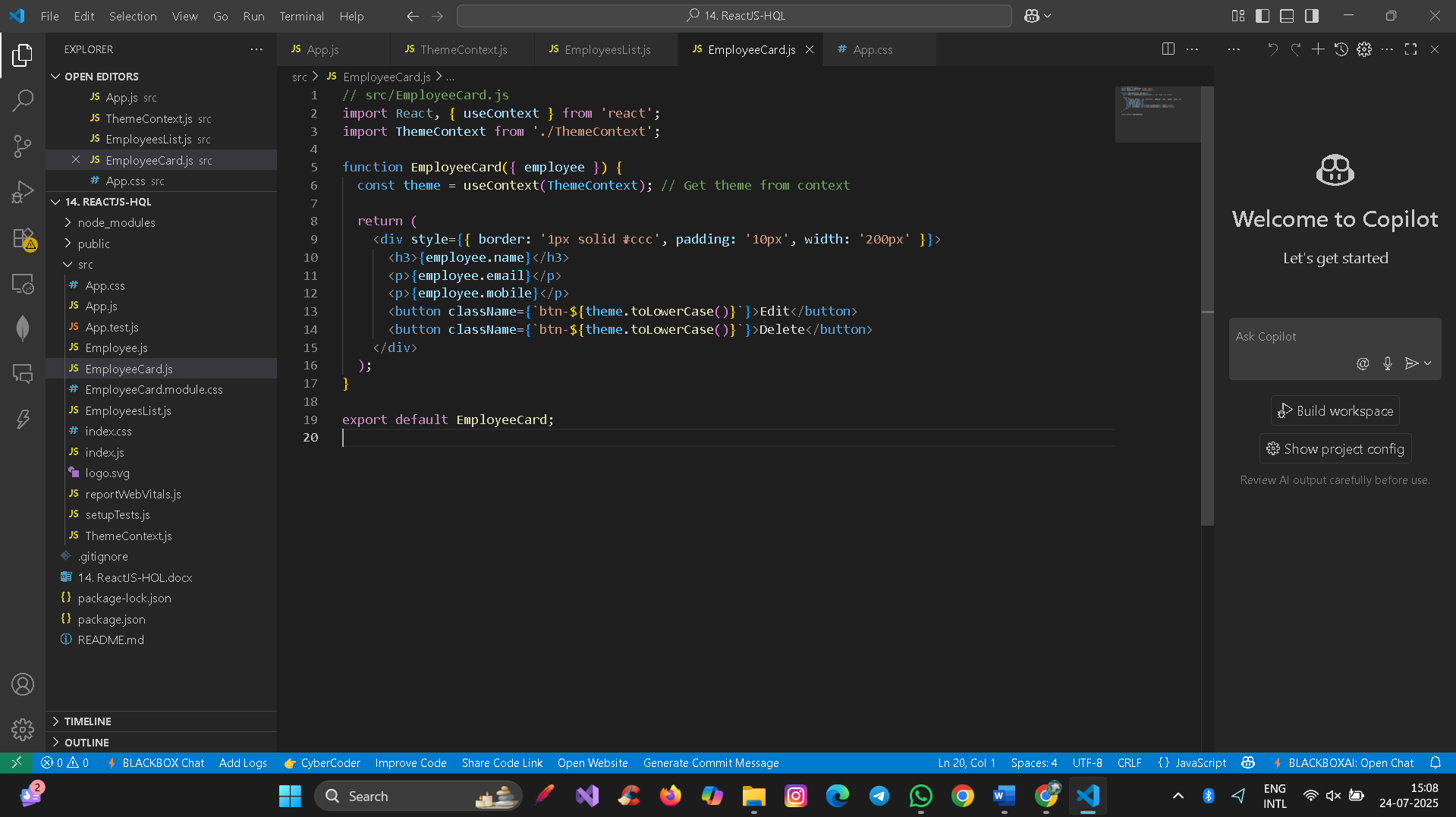
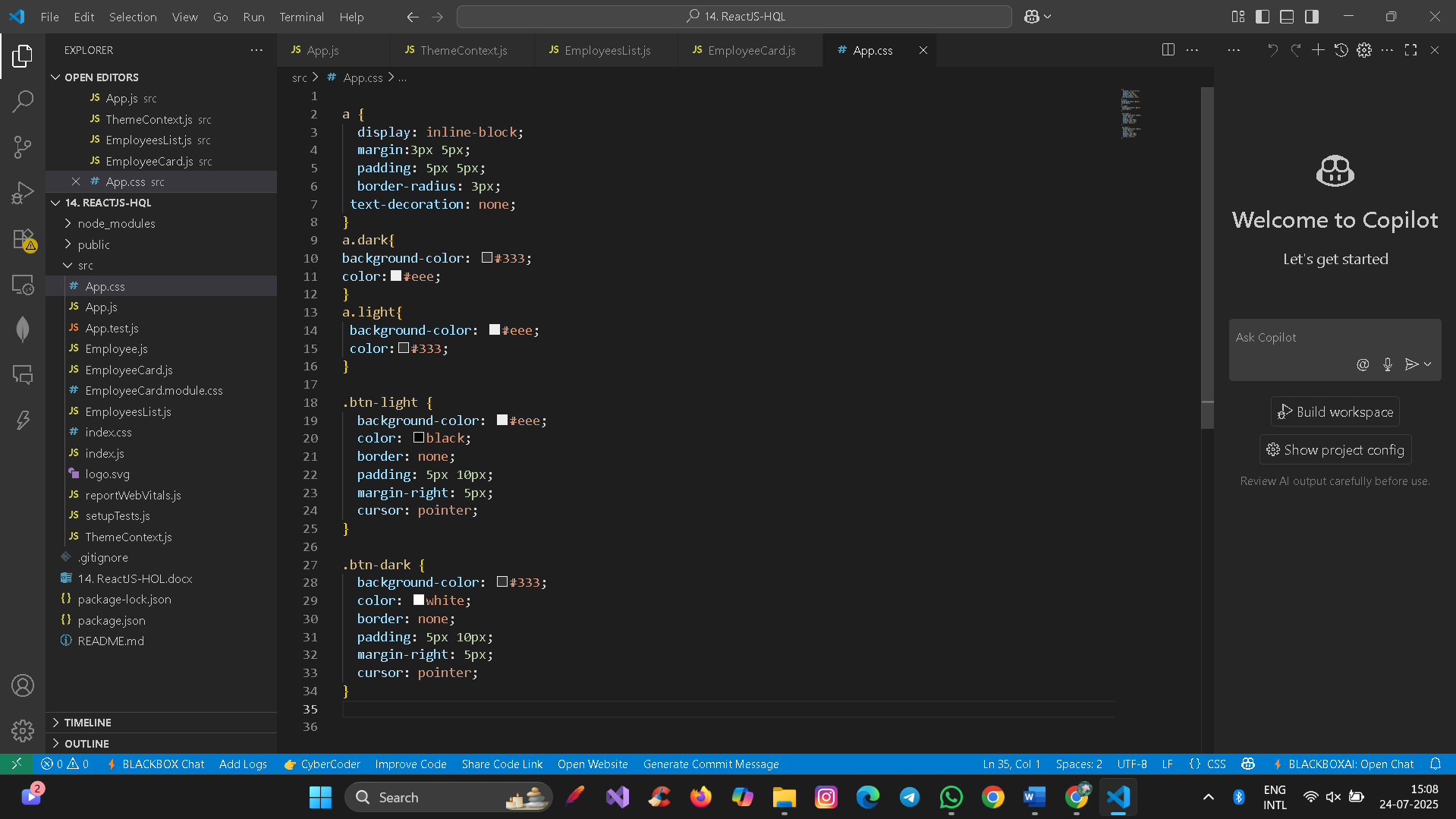
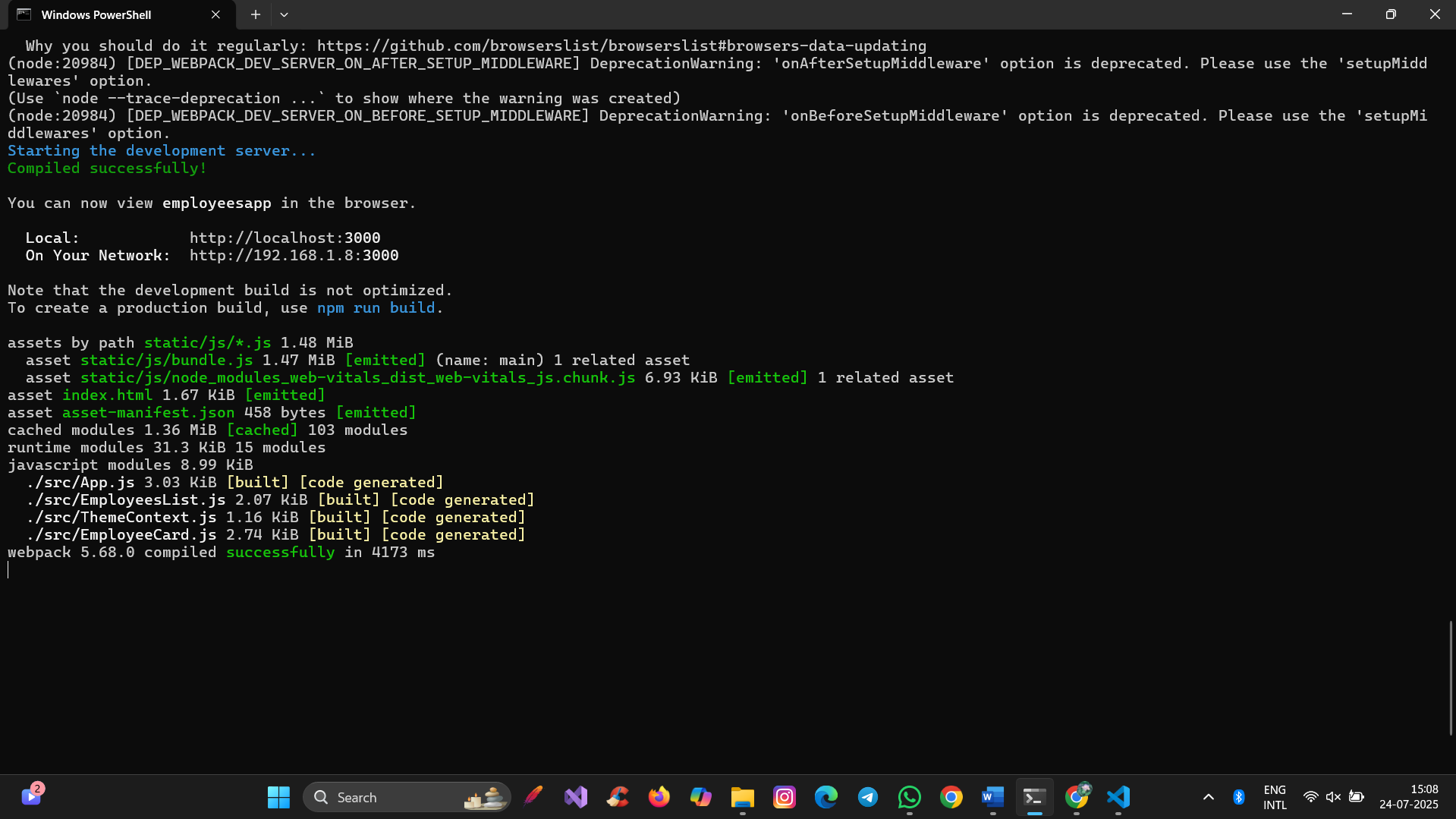
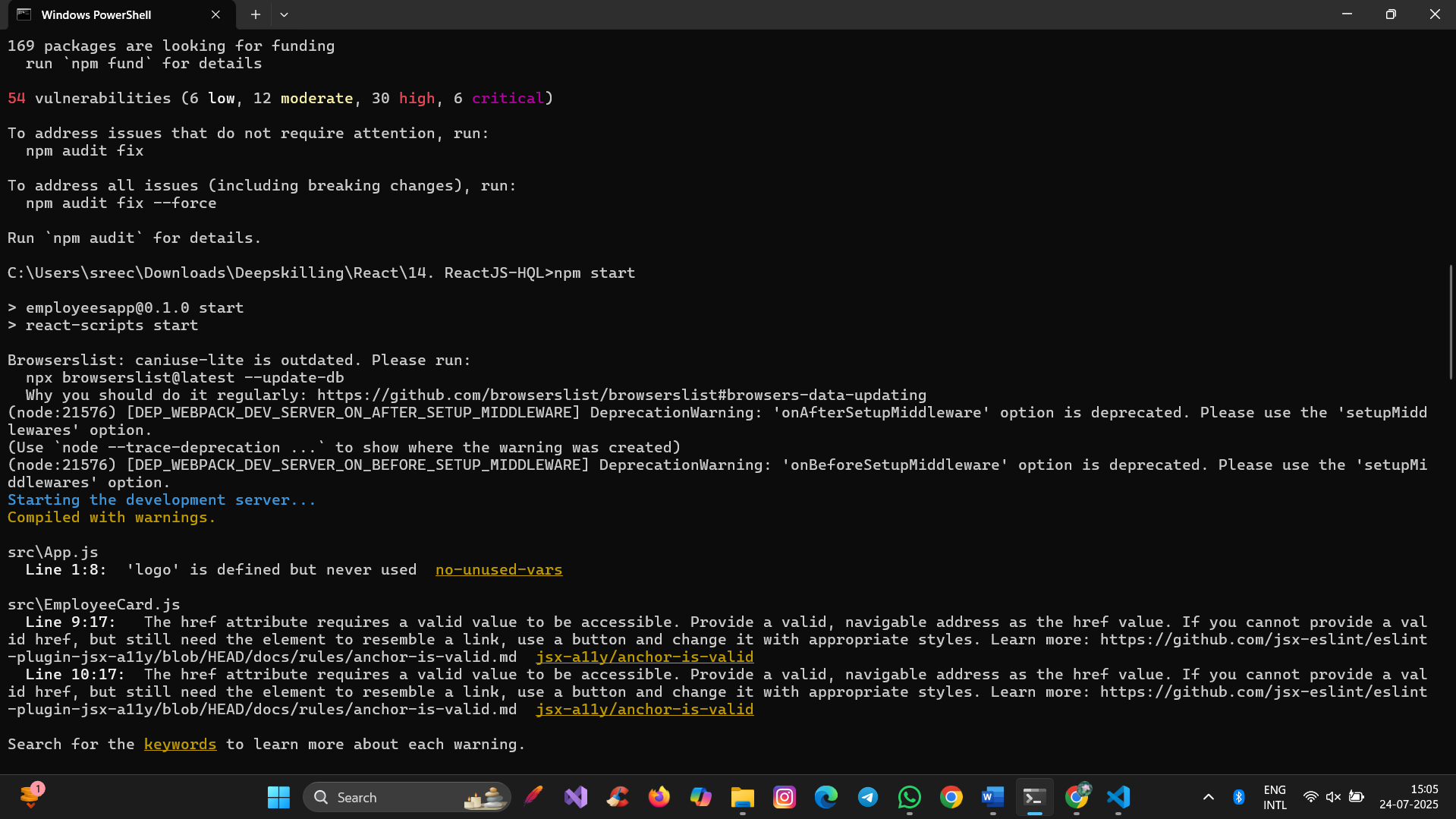
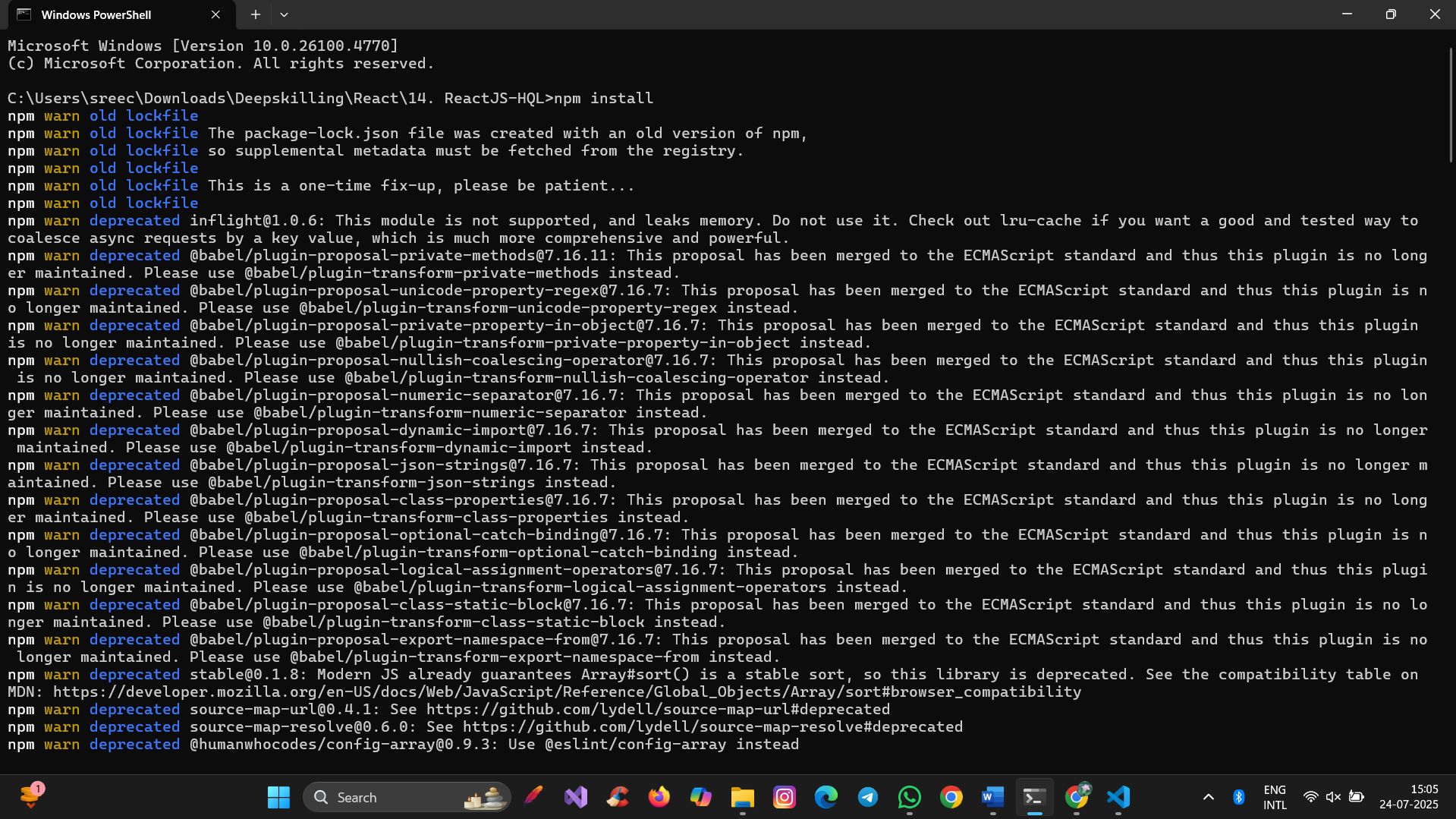
1. Run the application once to see the output. Use npm start command.

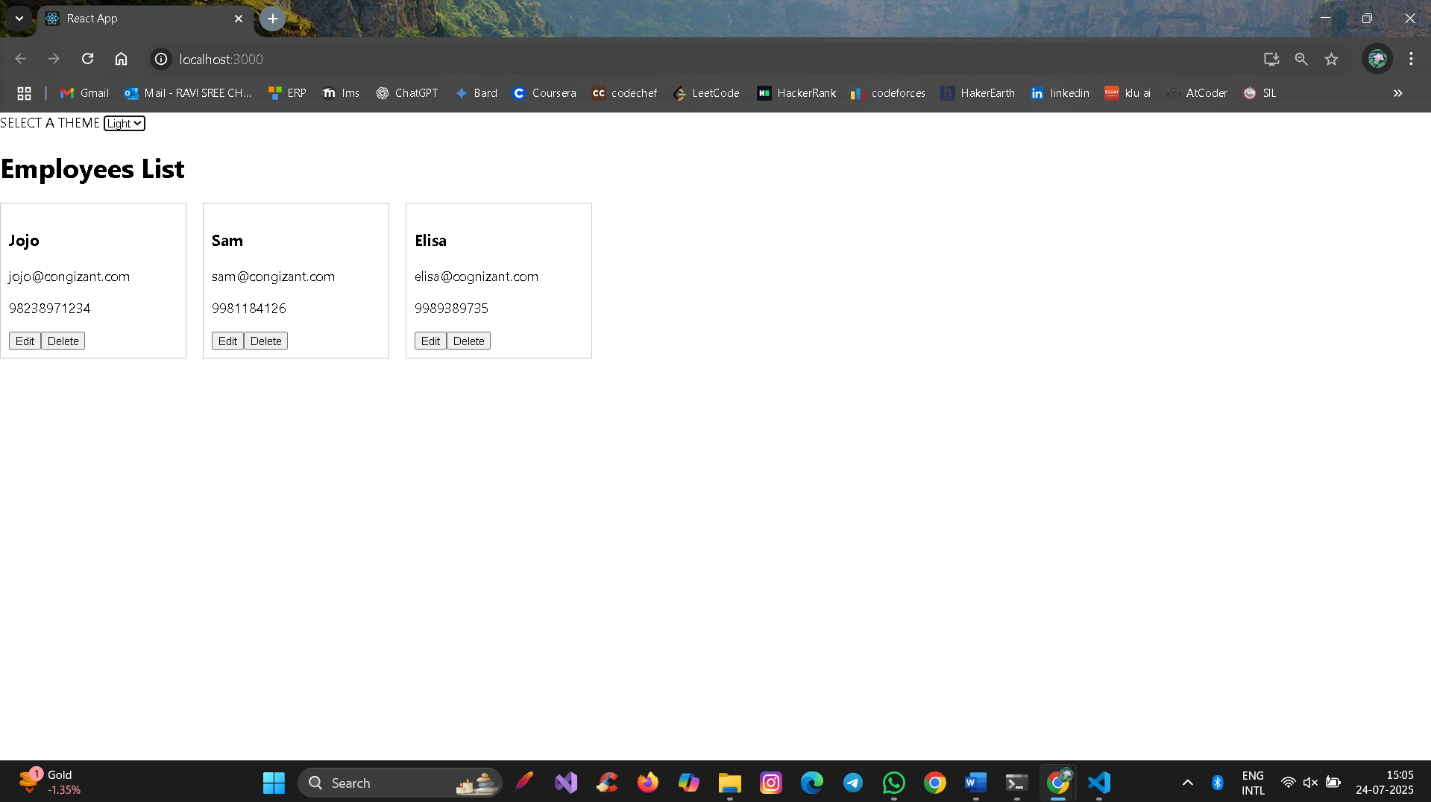


Figure 2: Starting application

1. Explore the components present in **App.js**, **EmployeesList.js** and **EmployeeCard.js** files.
2. Create a new file with the name as **ThemeContext.js**. Define a new context in the file with the name as ThemeContext and assign it a default value of ‘light’ and export it as default form the module.
3. Open App component present in **App.js** file.
   1. Import the ThemeContext in App component.
   2. Define the theme context provider to be the entire JSX of the App component.
   3. Assign the value for the theme provider from the state of the component.
   4. Modify the call to EmployeeList component so that theme name is no longer passed as props.
4. Go to EmployeeList component present in **EmployeeList.js** file and modify it so that theme name is not passed explicitly to its child component.
5. Go to **EmployeeCard** component inside **EmployeeCard.js** file
   1. Import the ThemeContext into the component file
   2. Retrieve the value of the context with the help of **useContext()** and store it in a variable
   3. Use the variable to pass the className for the buttons.

**HandsOn:**

****

****