NAME: JANHVI SINGH

REGISTRATION NUMBER: 22BCE8544

Assignment 11

App.css

/\* src/App.css \*/

/\* Basic reset for body and root element \*/

body {

  margin: 0;

  padding: 0;

  box-sizing: border-box; /\* Include padding and border in the element's total width and height \*/

  background-color: #eef2f6; /\* Light background for the entire page \*/

  min-height: 100vh; /\* Ensure body takes full viewport height \*/

  display: flex; /\* Use flexbox to center content if needed \*/

  justify-content: center; /\* Center horizontally \*/

  align-items: flex-start; /\* Align to the top of the viewport \*/

}

#root {

  /\* The div where your React app is mounted \*/

  width: 100%; /\* Take full width \*/

  display: flex;

  justify-content: center;

}

/\* Add any other global styles here \*/

App.jsx

import { useState } from 'react'

import reactLogo from './assets/react.svg'

import viteLogo from '/vite.svg'

import './App.css'

function App() {

  const [count, setCount] = useState(0)

  return (

    <>

      <div>

        <a href="https://vite.dev" target="\_blank">

          <img src={viteLogo} className="logo" alt="Vite logo" />

        </a>

        <a href="https://react.dev" target="\_blank">

          <img src={reactLogo} className="logo react" alt="React logo" />

        </a>

      </div>

      <h1>Vite + React</h1>

      <div className="card">

        <button onClick={() => setCount((count) => count + 1)}>

          count is {count}

        </button>

        <p>

          Edit <code>src/App.jsx</code> and save to test HMR

        </p>

      </div>

      <p className="read-the-docs">

        Click on the Vite and React logos to learn more

      </p>

    </>

  )

}

export default App

EmployeeCard.jsx

// src/EmployeeCard.jsx

import React from 'react';

/\*\*

 \* EmployeeCard Component

 \* Displays individual employee details.

 \*

 \* @param {object} props - The component's props.

 \* @param {string} props.name - The name of the employee.

 \* @param {string} props.position - The position of the employee.

 \* @param {string} props.department - The department of the employee.

 \*/

function EmployeeCard({ name, position, department }) {

  return (

    <div style={{

      backgroundColor: '#f9f9f9', // Light background for the card

      border: '1px solid #e0e0e0', // Light gray border

      borderRadius: '8px', // Rounded corners

      padding: '20px',

      marginBottom: '15px', // Space between cards

      boxShadow: '0 2px 5px rgba(0,0,0,0.05)', // Subtle shadow for depth

      display: 'flex',

      flexDirection: 'column',

      gap: '8px' // Space between text elements

    }}>

      <h3 style={{ margin: '0', color: '#007bff' }}>{name}</h3> {/\* Employee Name \*/}

      <p style={{ margin: '0', fontSize: '0.95em', color: '#555' }}>

        <strong>Position:</strong> {position} {/\* Employee Position \*/}

      </p>

      <p style={{ margin: '0', fontSize: '0.95em', color: '#555' }}>

        <strong>Department:</strong> {department} {/\* Employee Department \*/}

      </p>

    </div>

  );

}

export default EmployeeCard;

EmployeeDashboardjsx

// src/EmployeeDashboard.jsx

import React from 'react';

import Header from './Header';         // Import the Header component

import EmployeeCard from './EmployeeCard'; // Import the EmployeeCard component

/\*\*

 \* EmployeeDashboard Component

 \* Serves as the main container for displaying employee information.

 \* It holds the employee data and composes Header and EmployeeCard components.

 \*/

function EmployeeDashboard() {

  // Hardcoded array of employee objects

  const employees = [

    { id: 1, name: 'Alice Johnson', position: 'Software Engineer', department: 'Engineering' },

    { id: 2, name: 'Bob Williams', position: 'Product Manager', department: 'Product' },

    { id: 3, name: 'Charlie Brown', position: 'UX Designer', department: 'Design' },

    { id: 4, name: 'Diana Miller', position: 'QA Engineer', department: 'Engineering' },

    { id: 5, name: 'Eve Davis', position: 'HR Specialist', department: 'Human Resources' },

  ];

  return (

    <div style={{

      fontFamily: 'Arial, sans-serif',

      maxWidth: '900px', // Max width for the dashboard container

      margin: '40px auto', // Center the dashboard horizontally with top/bottom margin

      padding: '30px',

      border: '1px solid #c0c0c0', // Overall container border

      borderRadius: '10px', // Rounded corners for the main dashboard container

      boxShadow: '0 6px 12px rgba(0,0,0,0.15)', // More pronounced shadow

      backgroundColor: '#ffffff', // White background

      color: '#333' // Default text color

    }}>

      <Header /> {/\* Render the Header component \*/}

      <h2 style={{ color: '#333', textAlign: 'center', marginBottom: '25px' }}>Employee Directory</h2>

      {/\* Grid layout for employee cards \*/}

      <div style={{

        display: 'grid',

        gridTemplateColumns: 'repeat(auto-fill, minmax(280px, 1fr))', // Responsive grid columns

        gap: '20px' // Space between grid items (cards)

      }}>

        {/\* Loop through the employees array and render an EmployeeCard for each \*/}

        {employees.map(employee => (

          <EmployeeCard

            key={employee.id} // Unique key for each list item (important for React)

            name={employee.name}

            position={employee.position}

            department={employee.department}

          />

        ))}

      </div>

    </div>

  );

}

export default EmployeeDashboard;

README

**# React Employee Dashboard - Props and Component Composition**

This project enhances the previous employee dashboard by focusing on passing data using React `props` and leveraging `component composition` for a more modular and reusable application structure.

**## Project Structure**

\* `src/App.jsx`: The root component, responsible for rendering the `EmployeeDashboard`.

\* `src/EmployeeDashboard.jsx`: The main container component that manages the hardcoded employee data and renders `Header` and multiple `EmployeeCard` components.

\* `src/EmployeeCard.jsx`: A reusable component designed to display the details of a single employee. It receives employee data as `props`.

\* `src/Header.jsx`: A static component that displays the application's title.

\* `src/App.css`: Contains basic global styles for the application's layout.

**## How to Run**

1.  **\*\*Ensure Node.js and npm are installed\*\*** on your system.

2.  **\*\*Create the project directory (if not already done):\*\***

    Open your terminal and run:

    ```bash

    npm create vite@latest react-employee-dashboard --template react

    cd react-employee-dashboard

    ```

3.  **\*\*Install dependencies:\*\***

    ```bash

    npm install

    ```

4.  **\*\*Copy the provided `.jsx` and `.css` files\*\*** into the `src` directory of your `react-employee-dashboard` project, overwriting existing `App.jsx` and `App.css` if necessary. Ensure `Header.jsx` and `EmployeeCard.jsx` are created in `src`.

5.  **\*\*Start the development server:\*\***

    ```bash

    npm run dev

    ```

6.  **\*\*Open your browser\*\*** and navigate to the `Local` address provided in your terminal (e.g., `http://localhost:5173/`).

**## Explanation of Approach**

**### How Props are Passed to Child Components**

\* **\*\*`EmployeeDashboard.jsx` to `EmployeeCard.jsx`\*\***:

    \* In `EmployeeDashboard.jsx`, we have an array of `employees`.

    \* We use the `map()` function to iterate over this array:

        ```jsx

        {employees.map(employee => (

          <EmployeeCard

            key={employee.id} // Essential for React list rendering

            name={employee.name}

            position={employee.position}

            department={employee.department}

          />

        ))}

        ```

    \* For each `employee` object, an `EmployeeCard` component is rendered.

    \* The `employee.name`, `employee.position`, and `employee.department` values are passed as distinct `props` named `name`, `position`, and `department` to the `EmployeeCard` component.

\* **\*\*Receiving Props in `EmployeeCard.jsx`\*\***:

    \* The `EmployeeCard` component receives these props as a single `props` object in its function signature. We use object destructuring `{ name, position, department }` to directly access these values:

        ```jsx

        function EmployeeCard({ name, position, department }) {

          // ... use name, position, department directly in JSX

        }

        ```

    \* These destructured variables (`name`, `position`, `department`) are then directly used within the JSX of `EmployeeCard` to display the specific employee's information.

**### Component Composition for Modular Applications**

Component composition is the practice of building complex UIs by combining smaller, independent, and reusable components. This assignment showcases:

\* **\*\*`App.jsx` composing `EmployeeDashboard.jsx`\*\***: `App` is the top-level component that brings in `EmployeeDashboard`, which is a self-contained unit responsible for the entire employee display.

\* **\*\*`EmployeeDashboard.jsx` composing `Header.jsx` and `EmployeeCard.jsx`\*\***:

    \* The `EmployeeDashboard` component acts as a parent that orchestrates the layout and data flow.

    \* It includes the `Header` component directly (`<Header />`) to display the application's title. The `Header` is a presentational component with no data dependencies.

    \* It then composes multiple `EmployeeCard` components. Instead of having all the display logic for each employee within `EmployeeDashboard`, that responsibility is delegated to the `EmployeeCard`. `EmployeeDashboard` is responsible for *\*what\** data to show, while `EmployeeCard` is responsible for *\*how\** to display a single employee.

\* **\*\*Benefits of Composition\*\***:

    \* **\*\*Reusability\*\***: `EmployeeCard` can be used anywhere else in the application where individual employee details need to be displayed, simply by passing the relevant props.

    \* **\*\*Modularity\*\***: Each component focuses on a single responsibility, making the codebase easier to understand, test, and maintain.

    \* **\*\*Separation of Concerns\*\***: Data management (in `EmployeeDashboard`) is separated from presentation logic (in `EmployeeCard` and `Header`).

    \* **\*\*Scalability\*\***: As the application grows, new features can be added by creating new components and composing them, rather than modifying large, monolithic components.

**### Challenges Faced in Creating Reusable Components**

\* **\*\*Identifying Clear Component Boundaries\*\***: Initially, it can be tempting to put all logic and rendering in one large component. The challenge was to identify distinct, logical units (like an individual employee card or a header) that could be separated into their own components. The rule of thumb is "single responsibility."

\* **\*\*Prop Drilling (Minor Here)\*\***: For this simple app, passing `name`, `position`, `department` directly is fine. In larger applications, if data needs to pass through many layers of components that don't directly use the data, it can lead to "prop drilling." While not explicitly solved here (as it's not a major issue yet), this assignment lays the groundwork for understanding why state management solutions (like Context API or Redux) become necessary in more complex scenarios.

\* **\*\*Unique `key` Prop for Lists\*\***: Forgetting the `key` prop when mapping lists is a common beginner mistake that leads to React warnings and potential performance/rendering issues. Remembering that each item in a dynamic list *\*must\** have a stable, unique `key` (like `employee.id` in this case) is crucial for React's efficient reconciliation process.