React

1. What is React?

React is a JavaScript library for building user interfaces (UIs) based on components. It's efficient and declarative, making it easier to manage complex UIs.

2. Setting up a React Development Environment:

Use Create React App (CRA): `npx create-react-app my-react-app`. This sets up a new React project with a pre-configured development server.

3. Understanding JSX:

JSX is a syntax extension that allows HTML-like code within JavaScript. React uses JSX to describe the UI structure. Example: `const element = <h1>Hello, JSX!</h1>;`

4. Components: The Building Blocks:

Components are reusable pieces of UI. They can be functional (simple) or class-based (more complex).

5. Functional Components:

Simple JavaScript functions that accept props and return JSX. Example: `const MyComponent = (props) => { return <h1>Hello, {props.name}!</h1>; };`

6. Class Components (Less Common Now):

JavaScript classes that extend `React.Component`. They can manage state and handle more complex logic. Generally, functional components with hooks are preferred now.

7. Props: Passing Data to Components:

Props (short for properties) are used to pass data from a parent component to a child component.

8. State: Managing Data Within a Component:

State is data that can change within a component. Changes to state trigger re-renders. Use `useState` hook in functional components.

9. The `useState` Hook:

A hook that lets you add state to functional components. `const [count, setCount] = useState(0);`

10. Rendering Components:

Components are rendered using JSX within other components or directly using ReactDOM.render().

11. Component Composition:

Combining multiple components to create more complex UIs.

12. Conditional Rendering:

Displaying different content based on conditions. Use JavaScript's conditional operators (e.g., `&&`, `? :`) within JSX.

13. Lists and Keys:

When rendering lists of items, each item needs a unique "key" prop. This helps React efficiently update the list.

14. Event Handling:

Responding to user interactions like clicks, form submissions, etc. Use event handlers like `onClick`, `onChange`.

15. Forms:

Handling user input through form elements. Use controlled components where form data is stored in state.

16. The `useEffect` Hook:

A hook that lets you perform side effects (data fetching, subscriptions) in functional components.

17. Fetching Data (API Calls):

Use `fetch` or `axios` within `useEffect` to make API calls and update state with the data.

18. Routing (React Router):

Navigating between different pages or views in your application. React Router is a popular library for this.

19. Styling Components:

Styling your React components using CSS, CSS Modules, styled-components, or other styling solutions.

20. Deployment:

Building your React app for production and deploying it to a web server or hosting platform. CRA provides build scripts for this.

```
task-manager/
├── src/
│   ├── App.js          // Main component
│   ├── Task.js         // Task component
│   ├── index.js        // Entry point
│   └── styles.css     // (Optional) Stylesheet
├── public/
│   └── index.html      // HTML file
└── package.json       // Project configuration
```

Okay, let's build a simple "Task Manager" application step-by-step, illustrating the React concepts. I'll provide code snippets for each step, and you can ask questions along the way.

1. Project Setup:

npx create-react-app task-manager

cd task-manager

npm start

2. Initial Component Structure (App.js):

```
import React from 'react';

function App() {
```

```
  return (
    <div className="task-manager">
      <h1>Task Manager</h1>
      {/* Tasks will go here */}
    </div>
  );
}

export default App;
```

3. Creating a Task Component:

```
// Task.js
import React from 'react';

function Task(props) {
  return (
    <div className="task">
      <h3>{props.title}</h3>
      <p>{props.description}</p>
    </div>
  );
}

export default Task;
```

4. Rendering Tasks in App.js (using props):

```
import React from 'react';
import Task from './Task';

function App() {
  const tasks = [
    { title: 'Grocery Shopping', description: 'Buy milk, eggs, and bread'
},
    { title: 'Pay Bills', description: 'Electricity and internet' },
  ];
```

```
  return (
    <div className="task-manager">
      <h1>Task Manager</h1>
      {tasks.map((task, index) => (
        <Task key={index} title={task.title} description={task.description}
/>
      ))}
    </div>
  );
}

export default App;
```

5. Adding State for New Tasks:

```
import React, { useState } from 'react';
import Task from './Task';

function App() {
  const [tasks, setTasks] = useState([
    { title: 'Grocery Shopping', description: 'Buy milk, eggs, and bread'
},
    { title: 'Pay Bills', description: 'Electricity and internet' },
  ]);

  // ... (rest of the component)
}

export default App;
```

6. Input Field and Add Task Function:

```
import React, { useState } from 'react';
import Task from './Task';

function App() {
  const [tasks, setTasks] = useState([ /* ... */ ]);
  const [newTaskTitle, setNewTaskTitle] = useState('');
  const [newTaskDescription, setNewTaskDescription] = useState('');
```

```
  const addTask = () => {
    setTasks([
      ...tasks,
      { title: newTaskTitle, description: newTaskDescription },
    ]);
    setNewTaskTitle(''); // Clear input fields
    setNewTaskDescription('');
  };

  return (
    <div className="task-manager">
      <h1>Task Manager</h1>
      <input
        type="text"
        placeholder="Task Title"
        value={newTaskTitle}
        onChange={(e) => setNewTaskTitle(e.target.value)}
      />
      <input
        type="text"
        placeholder="Task Description"
        value={newTaskDescription}
        onChange={(e) => setNewTaskDescription(e.target.value)}
      />
      <button onClick={addTask}>Add Task</button>
      {/* ... (rendering tasks) */}
    </div>
  );
}

export default App;
```

We have:

- Project setup with Create React App.
- A `Task` component to display individual tasks.
- State to manage the list of tasks.
- Input fields and a button to add new tasks.

If got this error while setting up off the React application?

```
Module not found: Error: Can't resolve 'web-vitals' in 'D:\Bhaskar\06-02-2025\task-manager\src'
ERROR in ./src/reportWebVitals.js 5:4-24
Module not found: Error: Can't resolve 'web-vitals' in 'D:\Bhaskar\06-02-2025\task-manager\src'
```

Follow as said below:

The "Can't resolve 'web-vitals'" error usually arises in Create React App (CRA) projects because the `web-vitals` package is referenced in `reportWebVitals.js` (or sometimes directly in `index.js`) but isn't installed by default. It's related to performance tracking. Here's how to address it and prevent similar issues:

1. Install web-vitals (Recommended):

The simplest and best solution is to install the package. This allows you to use web vitals if you want to track performance. [1]

```
npm install web-vitals
# or
yarn add web-vitals
```

After installing, the error should disappear.

2. Remove or Comment Out the Import (If you don't need web vitals):

In `src/index.js` (or `src/reportWebVitals.js` if it exists):

```
// import reportWebVitals from './reportWebVitals'; // Comment out or
remove this line

// ... rest of your index.js code

// reportWebVitals(console.log);  // Comment out or remove this line as
well if it exists.
```

If the `reportWebVitals` file exists and you've commented out the import in `index.js`, you can optionally delete the `reportWebVitals.js` file.

3. Restart the Development Server:

After installing or removing the import, restart your development server to ensure the changes take effect.

Preventing Similar Errors in the Future:

- **Pay attention to console warnings:** React and CRA often give helpful warnings in the console. Address them promptly.
- **Check package.json:** If you encounter a "cannot resolve" error for a module, make sure it's listed as a dependency in your `package.json` file. If it's not there, you'll need to install it.
- **Typos:** Double-check for typos in your import statements and package names. A simple misspelling can lead to these errors.

Clean installs: If you're still having trouble, try deleting your `node_modules` folder and reinstalling your dependencies:

```
rm -rf node_modules  # or delete the node_modules folder manually
npm install         # or yarn install
```

- **Version Compatibility:** Make sure your dependencies are compatible with each other and with your Node.js and npm/yarn versions. Check the documentation for the libraries you're using.

Finally the clubbed code for Task Manager application:

Key changes:

- **File Structure:** Organized into `src` with `App.js`, `Task.js`, and `index.js`. Added a styles.css file for basic styling.
- **Delete Functionality:** Added a `deleteTask` function in `App.js` and passed it as a prop to the `Task` component. The `Task` component now has a delete button that calls this function.
- **Empty Task Prevention:** Added a check in `addTask` to prevent adding tasks with empty titles.
- **CSS Styling:** Added basic CSS for layout and appearance. The task list now uses a responsive grid.

- **Comments:** Added comments to explain the code.

```
//src/App.js

import React, { useState } from 'react';
import Task from './Task';  // Import the Task component
import './styles.css' //import the css file

function App() {
  const [tasks, setTasks] = useState([
    { title: 'Grocery Shopping', description: 'Buy milk, eggs, and bread'
},
    { title: 'Pay Bills', description: 'Electricity and internet' },
  ]);

  const [newTaskTitle, setNewTaskTitle] = useState('');
  const [newTaskDescription, setNewTaskDescription] = useState('');

  const addTask = () => {
    if (newTaskTitle.trim() !== "") { //Prevent adding empty tasks
      setTasks([
        ...tasks,
        { title: newTaskTitle, description: newTaskDescription },
      ]);
      setNewTaskTitle('');
      setNewTaskDescription('');
    } else {
      alert("Task Title cannot be empty!")
    }
  };

  const deleteTask = (index) => {
    const updatedTasks = tasks.filter((_, i) => i !== index);
    setTasks(updatedTasks);
  };

  return (
    <div className="task-manager">
      <h1>Task Manager</h1>

      <div className="add-task"> {/* Container for adding tasks */}
```

```
        <input
          type="text"
          placeholder="Task Title"
          value={newTaskTitle}
          onChange={(e) => setNewTaskTitle(e.target.value)}
        />
        <input
          type="text"
          placeholder="Task Description"
          value={newTaskDescription}
          onChange={(e) => setNewTaskDescription(e.target.value)}
        />
        <button onClick={addTask}>Add Task</button>
      </div>

      <div className="task-list"> {/* Container for the task list */}
        {tasks.map((task, index) => (
          <Task
            key={index}
            title={task.title}
            description={task.description}
            onDelete={() => deleteTask(index)} // Pass the delete function
          />
        ))}
      </div>
    </div>
  );
}

export default App;
```

```
// src/Task.js

import React from 'react';

function Task(props) {
  return (
    <div className="task">
      <h3>{props.title}</h3>
```

```
      <p>{props.description}</p>
      <button onClick={props.onDelete}>Delete</button> {/* Delete button
*/}
    </div>
  );
}

export default Task;
```

```
// src/styles.css


.task-manager {
  font-family: sans-serif;
  margin: 20px;
}

.add-task {
  display: flex;
  gap: 10px; /* Space between input fields and button */
  margin-bottom: 20px;
}

.add-task input {
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.add-task button {
  padding: 8px 16px;
  background-color: #4CAF50;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.task-list {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); /*
```

```
Responsive grid */
    gap: 20px;
}

.task {
  border: 1px solid #ccc;
  padding: 10px;
  border-radius: 4px;
  display: flex;
  flex-direction: column; /* Align title, description, and button
vertically */
}

.task h3 {
  margin-top: 0; /* Remove default margin for h3 */
}

.task button {
  margin-top: 10px; /* Space between description and button */
  padding: 5px 10px;
  background-color: #f44336; /* Red color for delete button */
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  align-self: flex-end; /* Align button to the right */
}
```

Output: