Project Setup :

- Create a new React application and paste this code into your **App.jsx** file to handle the switching of different tasks without implementing router logic. Replace Task1 and Task2 with your component names.

JavaScript

```
import React, { useState } from "react";
import Task1 from "./tasks/task1/Task1";
import Task2 from "./tasks/task2/Task2";

const App = () => {
  const [taskItem, setTaskItem] = useState("Task1");

  const TaskItemComponent = {
    Task1: <Task1 />,
    Task2: <Task2 />,
  };

  return (
    <div>
      <div>
        <div>
          <input type="radio" id="task1" checked={taskItem === "Task1"}
onChange={() => setTaskItem("Task1")} />
          <label htmlFor="task1">Task 1</label>
        </div>
        <div>
          <input type="radio" id="task2" checked={taskItem === "Task2"}
onChange={() => setTaskItem("Task2")} />
          <label htmlFor="task2">Task 2</label>
        </div>
      </div>
      {TaskItemComponent[taskItem]}
    </div>
  );
};

export default App;
```

Additional Instructions:

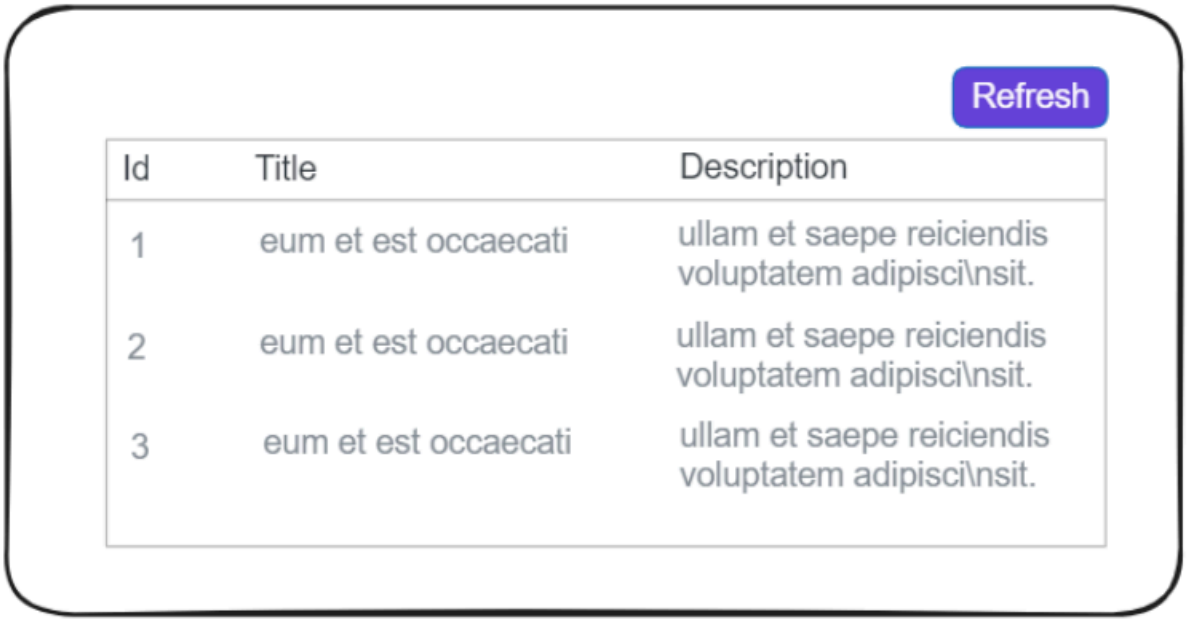
- You may refer to the library documentation only if necessary.
- You can use any CSS libraries to simplify the styling process, but styling is not important.
- Avoid using any GPT model for references or code generation.

Task 1 :

Time duration : 0.5 hr

Implementation Details:

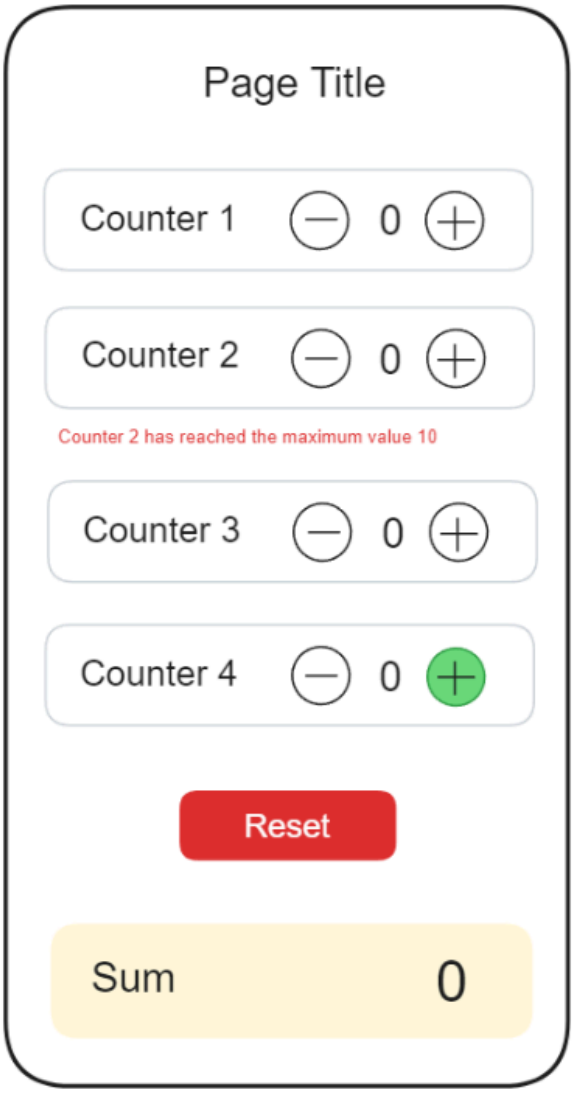
- Create a custom hook “useFetch” for data fetching. The hook should accept the api URL and return states for loading, error, data, and onRefresh function. Add a 2 second delay before initiating the data fetch.
- Fetch data from the API url “https://jsonplaceholder.typicode.com/posts” using the custom hook.
- Develop a Table component to display the fetched data. Pass the data into Table through the props object.
- The table should include columns for Id, Title, and Description. Map the fetched data into the table body and display all items from the data.
- Add a refresh button above the table. On clicking the refresh button, trigger a new data fetch to update the displayed information.(Use custom hook’s onRefresh function).
- Show a loading text while fetching data.
- Display an error message if the API fails to fetch data.



Task 2 :

Time duration : 1.5 hr

Implementation Details:



- Develop a React application based on the provided design.
- Create separate components for the counter card, reset button, and sum card.
- Use Redux or Redux Toolkit or any other library (except context api) for state management.
- The plus and minus button of the counter card should be clickable and the value in between them should change accordingly.
- Set the counter minimum value as zero and maximum value as ten. Implement this logic in the reducer. Once the counter reaches the maximum value an error message with the counter name should appear just below the corresponding counter. (eg: Counter 2 has reached the maximum value 10). This message should automatically disappear when the value decremented from the maximum.
- Display the Page Title only after 3 seconds from the initial page render.
- On clicking the reset button, reset all counter values to zero.
- Update the sum card dynamically with the sum of all counter values.
- Ensure that interacting with one counter card does not trigger the re-rendering of other counter cards. Assign a unique id to each counter card and log/print this id within the respective counter card component to identify re-render.
- Remove React.StrictMode from the main file to prevent second re-render in the development mode.