AeroAspire

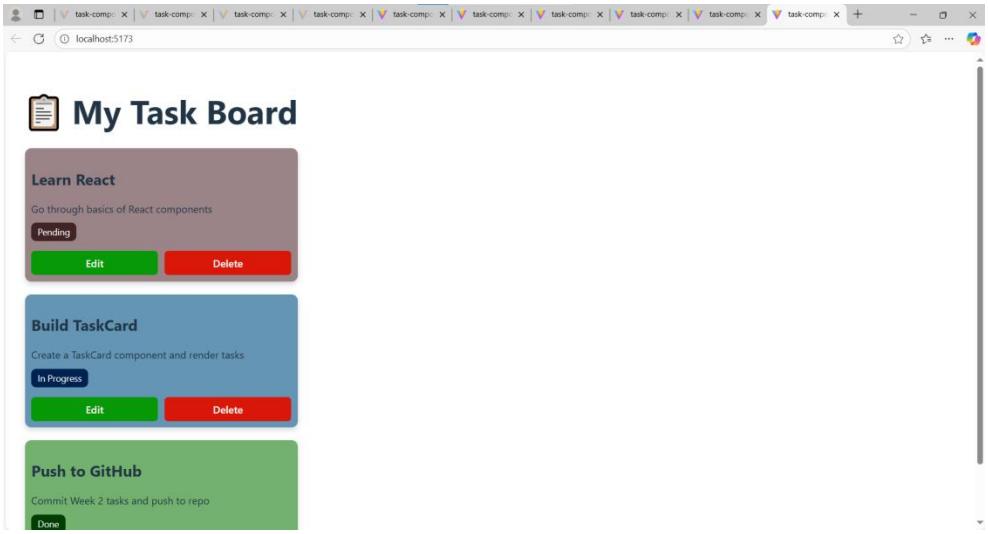
SDE Intern

Goutham V

Week 2 – Day2 (30th September)

Task:

- **TaskCard component; render list of dummy tasks via props**



Steps I Followed for Week 2 Task – React Components & Props

1. Created a new branch (week2-tasks) in GitHub to keep my work organized.
2. Made a reusable TaskCard component that accepts title, description, status, and color as props.
3. Added dummy task data inside App.jsx and used the map() function to render multiple cards dynamically.

4. Designed a colorful UI using inline CSS with pastel backgrounds, status badges, and hover effects.
5. Arranged the task cards in a responsive grid layout so they look compact and neat.
6. Added Edit and Delete buttons to each card for future functionality.
7. Tested the project locally with npm start to check the output.
8. Finally, committed and pushed the changes to GitHub.

Reflection,

1.How props are passed from parent to child; what happens if props change?

- In React, props are like function parameters. A parent component can send data down to its child components through props.
- The child component cannot change props; it can only use them. This makes props “read-only.”
- When the parent’s state or data changes, React re-renders the parent. During this process, the new values of props are sent again to the child.
- If the new props are different from the old ones, React will update the child’s UI accordingly.
- Example: If I pass status="Pending" to my TaskCard and later the parent updates it to "Done", React re-renders that card with the new status.

2.What is the virtual DOM in React and how does re-render happen when props change?

- The virtual DOM is like a blueprint of the real DOM that React keeps in memory. It’s a lightweight copy that React uses to decide how to update the page efficiently.
- When props or state change, React doesn’t blindly re-draw the whole UI. Instead:

    - React builds a new virtual DOM tree with the updated props.
    - It compares this new tree with the previous one (this process is called diffing or reconciliation).

- Only the elements that have actually changed are updated in the real DOM.

❖ This makes updates very fast because React avoids unnecessary work on parts of the UI that haven't changed.
❖ For example, if only the description of one task changes, React won't rebuild the whole task list — just that one card.

3.How to avoid unnecessary re-renders?

❖ React re-renders whenever props or state change, but sometimes this causes components to re-render even if the visible output hasn't changed. To optimize this, we can:

- **Use** React.memo: This tells React to remember the last rendered output of a component and skip re-rendering if the props are the same.
- Keep components pure: Don't mutate props or state directly. Instead, always create new objects/arrays so React can detect real changes.
- Split big components into smaller ones: This way, only the part that depends on the changing props re-renders instead of the entire UI.
- Avoid passing unnecessary props: If a child doesn't need certain props, don't pass them, because changes in those props would still trigger re-renders.

❖ By following these steps, React apps stay smooth even when handling lots of updates.