AeroAspire

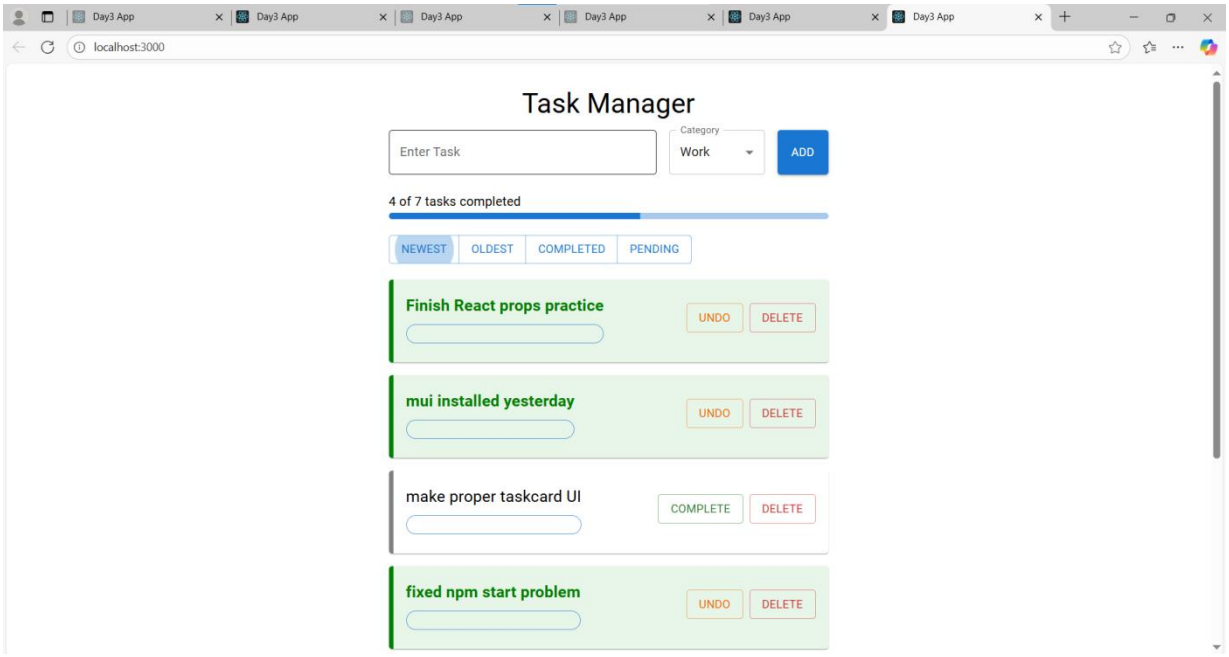
SDE Intern

Goutham V

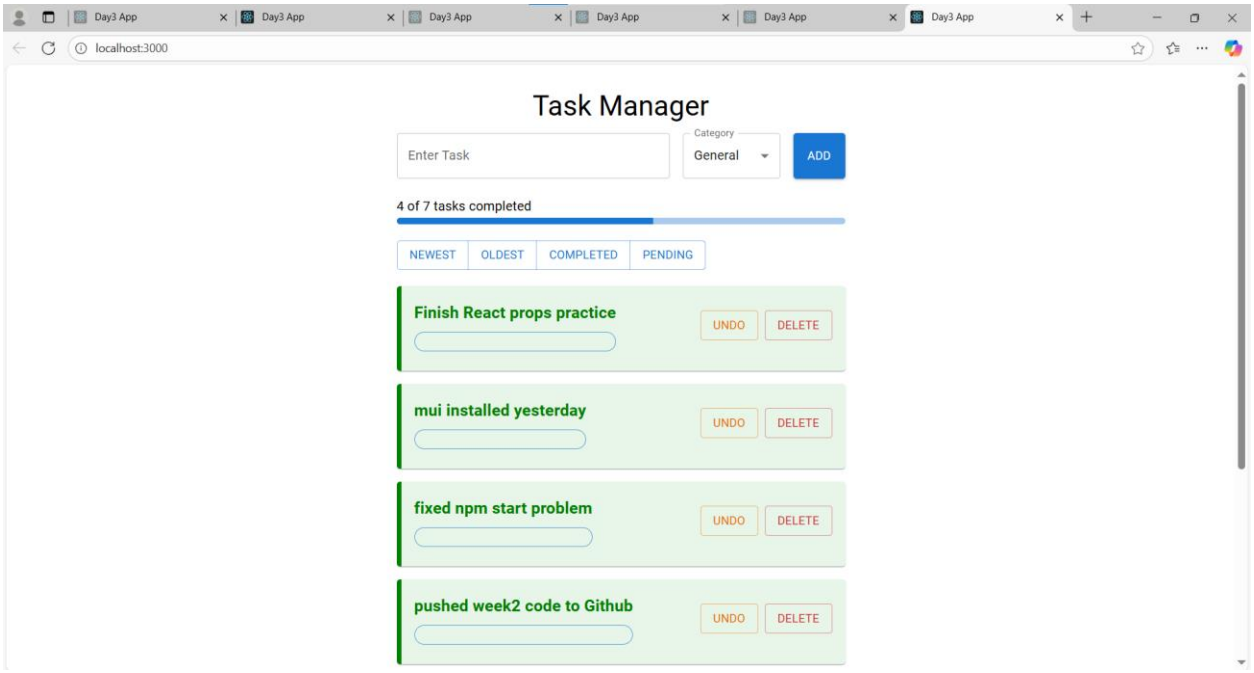
Week 2 – Day3 (01st October)

Task:

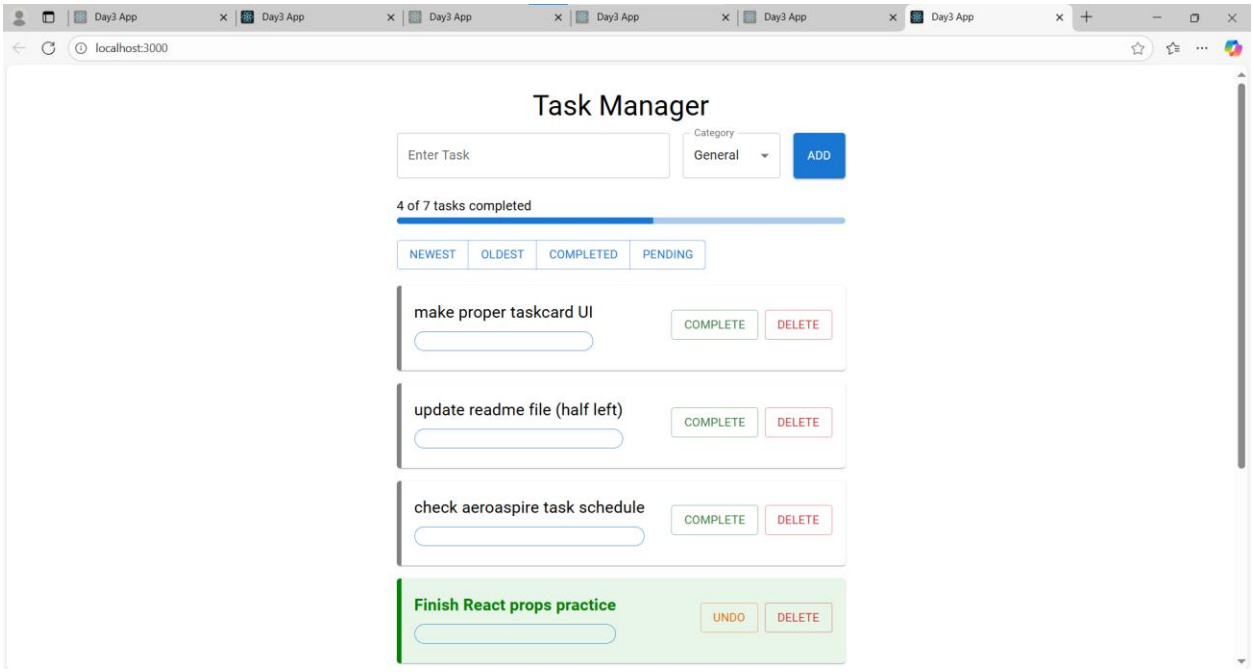
Add ability to add tasks; manage state; fetch effects if needed (dummy), Create form using MUI TextFields and Buttons; validate required inputs / length / number etc.



This screen shows the newest added tasks



This screen shows the completed tasks



This screen shows the incompleted tasks

Steps I Followed

1. I created a *React* app for my Day 3 task of Week 2.
2. I made two components inside src/components:
    - TaskForm.js → to add new tasks
    - TaskCard.js → to display each task
3. I installed *Material UI (MUI)* to style the app with TextFields, Buttons, Cards, Stack, etc.
4. I implemented features for the tasks:
    - Add tasks with a name and category
    - Mark tasks as completed or undo them
    - Delete tasks
    - Show progress of completed tasks with a progress bar
    - Sort tasks by newest, oldest, completed, or pending
5. I created a README.md file in the root of my project.
6. I made a folder called images in the root and added screenshots of my app there.
7. In README.md, I added the images using this format:
8. ![Alt text](images/filename.png)
9. *Caption describing the image*
10. I made sure the image filenames and folder names matched exactly (case sensitive).
11. I committed all changes in VS Code using Source Control**:**
    - Staged the files (code, README, images)
    - Added a commit message
    - Clicked Push to upload to GitHub
12. I checked GitHub to make sure the images appeared in the README correctly

Reflection,

1. Walk through flow: user types, state updates, component re-renders, effect runs (if any)

When a user types into an input, the onChange updates the state variable associated with that input. React then re-renders the component to display the latest state in the UI. If there's a useEffect monitoring that state, it runs after the re-render to perform actions, like saving to localStorage. This flow **— user input → state update → render → effect —** is exactly how React keeps the UI in sync with data.

2. How useEffect works: dependencies, cleanup, initial render

useEffect lets you perform actions in function components.

• **Initial render**: If you provide an empty dependency array [], the effect runs once after the first render.

• **Dependencies**: Passing [tasks] or any variable makes the effect run whenever that variable changes.

• **Cleanup**: Returning a function from useEffect lets you clean up resources before the next effect runs or before the component unmounts. In my task app, I used useEffect to load tasks from localStorage when the component mounted and to save tasks whenever they changed.

3. What pitfalls exist (e.g. stale closures, infinite loops)?

Some common issues from the documentation:

• **Stale closures**: If an effect captures an old value of state, it may act on outdated data.

• **Infinite loops**: Updating state inside useEffect without proper dependencies can cause React to re-render endlessly.

• **Missing dependencies**: Effects may not run as expected if you forget to include state variables in the dependency array. I avoided these issues by carefully specifying [tasks] as the dependency whenever I wanted to save tasks.

4. What is a controlled vs uncontrolled component?

• **Controlled component**: React state owns the value of the input. You update the state via onChange and use the state as the value.

- Example: value={task} in TextField.

• **Uncontrolled component**: The input manages its own value internally, and React only reads it through a ref. In my app, I used controlled components to handle validation and clear the field after submission.

5. Describe event handling in forms (onChange, onSubmit)

• **onChange**: Triggered every time the input changes. I use it to update the state with what the user typed.

• **onSubmit**: Triggered when the form is submitted (button click or Enter key).

- I call *preventDefault()* so the page does not reload.
- Then I validate input and update the task list state. This is how React handles form events while keeping the UI in sync with the state.

6. How does MUI help: theming, form helpers, error display?

Material UI makes building React UIs faster:

• **Theming**: Easily manage colors, spacing, and typography consistently across the app.

• **Form helpers**: Components like TextField have error and helperText props that make input validation simpler.

• **Layout components**: Stack, Card, Button, etc., help create a clean design without writing extra CSS.

It fits well with React's declarative style, allowing me to focus on state and logic instead of styling every detail manually.