AeroAspire

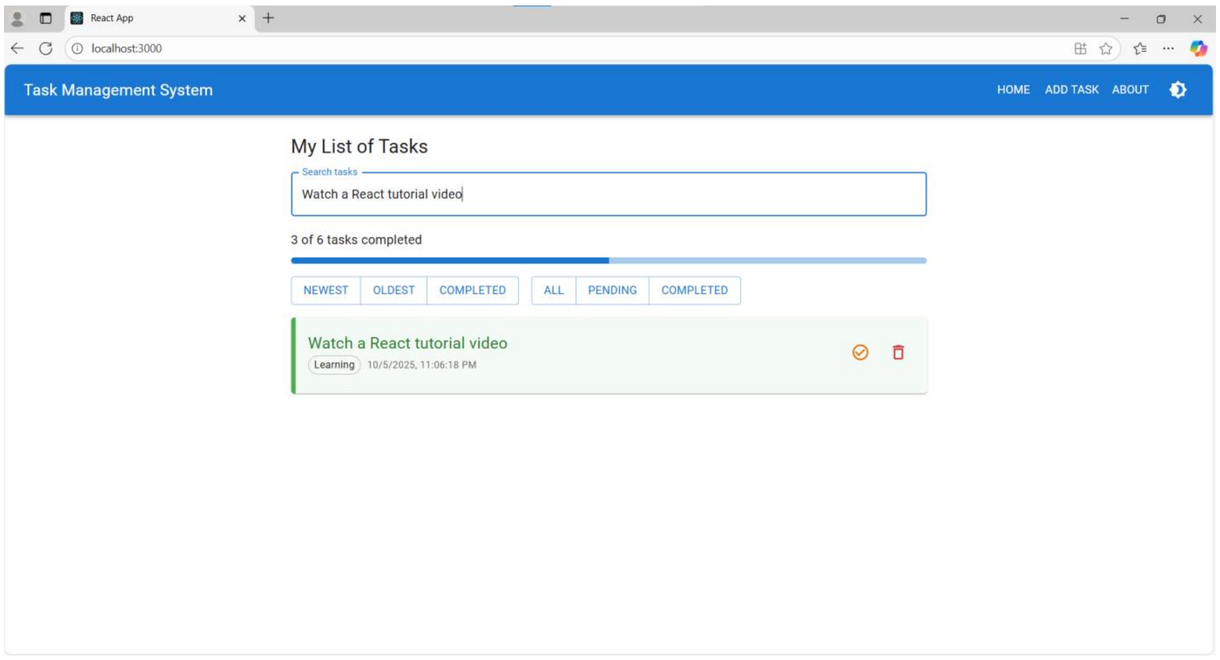
SDE Intern

Goutham V

Week 2 – Day5 (04rd October)

Task:

Add search/filter UI; theme toggle (light/dark) if possible; polish



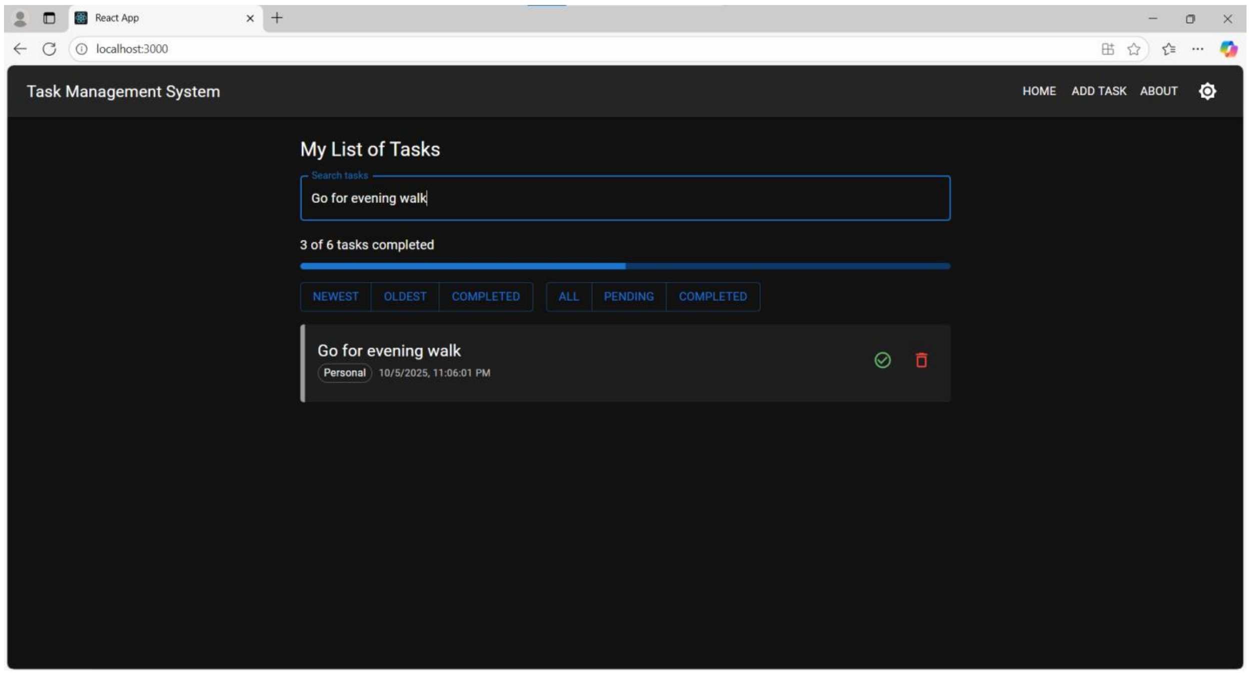
My List of Tasks

Search tasks: Watch a React tutorial video

3 of 6 tasks completed

Watch a React tutorial video
Learning 10/5/2025, 11:06:18 PM

Light Theme with search bar.



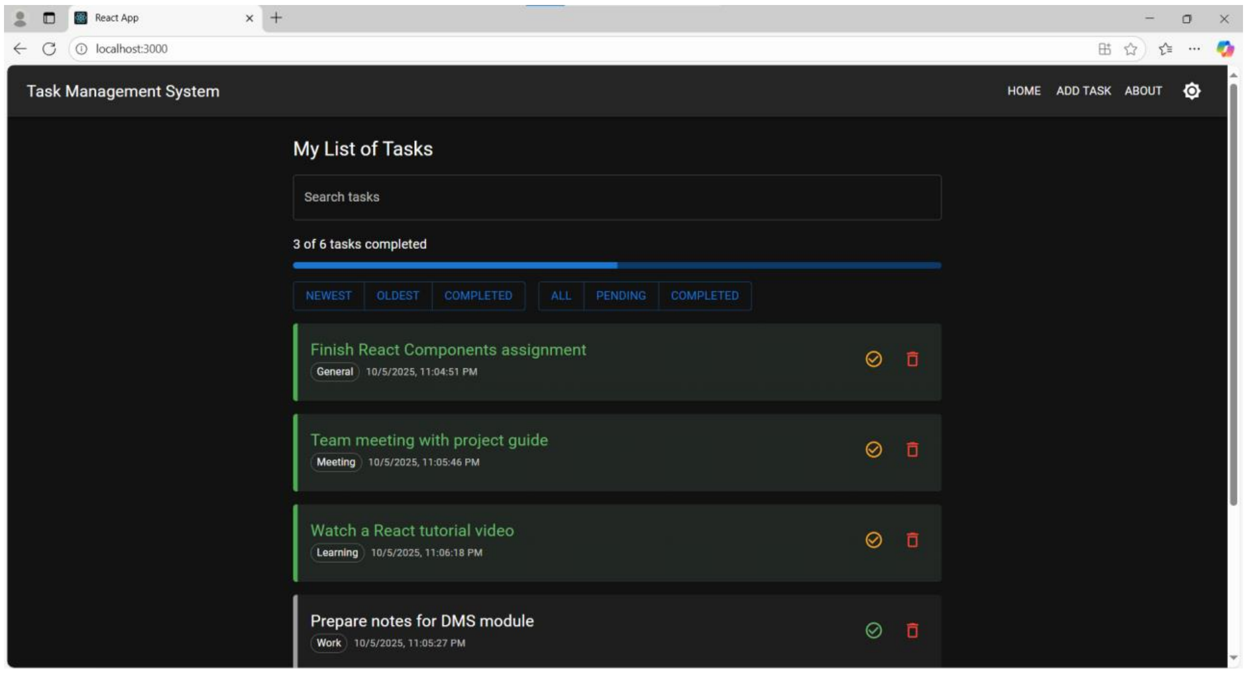
My List of Tasks

Go for evening walk

3 of 6 tasks completed

Go for evening walk
Personal 10/5/2025, 11:06:01 PM

Dark Theme with search bar.



My List of Tasks

3 of 6 tasks completed

Finish React Components assignment
General 10/5/2025, 11:04:51 PM

Team meeting with project guide
Meeting 10/5/2025, 11:05:46 PM

Watch a React tutorial video
Learning 10/5/2025, 11:06:18 PM

Prepare notes for DMS module
Work 10/5/2025, 11:05:27 PM

Polished Add task UI.

Steps I Followed

1) First, I created a new React app inside my *Day5* folder.
2) I installed the needed packages like *Material UI* and *React Router***.**
3) I added a *search bar* so that tasks can be searched easily by typing their name.
4) Then I worked on a *theme toggle button* to switch between light and dark mode.
5) I polished the UI a bit so that it looks better and smoother compared to the earlier days.
6) I used *localStorage* so that even after refreshing, the tasks do not disappear.
7) Finally, I tested the app with some dummy tasks and took screenshots of light mode, dark mode, and the polished Add Task UI.

Reflection,

1. Explain the full data flow: user action → state → props → UI update.

-When I was working on this task, the best example of the data flow was when I added a task. First, I typed something into the input box (user action). That updated the state in my component using useState. From there, the state was passed as props to the TaskCard component, so each card knew what to display. Once the state was updated, React automatically re-rendered the UI and I could immediately see my new task appearing in the list.
The same thing happened when I toggled a task as complete or deleted it. The action → state change → props update → UI render loop is very clear in these

cases. This flow made me realize how React keeps everything in sync without me manually touching the DOM.

2. What are common anti-patterns in React you noticed or want to avoid?

-While doing my tasks, I ran into some issues with **react-scripts**. Every day, I had to delete node_modules and reinstall because of version issues, and that felt like a recurring blocker. I'd call this an anti-pattern because relying on deleting and reinstalling repeatedly shows that the project setup isn't stable. Instead, pinning versions and using a proper package manager lock file would have been better.

Other anti-patterns I came across or realized I should avoid:

- Putting too much code inside one file like App.js instead of breaking it down.
- Forgetting to add a dependency array in useEffect, which once gave me repeated re-renders.
- Not handling localStorage properly — at first my tasks disappeared on refresh, which showed me how easy it is to misuse persistence.
- Ignoring errors in the console (at times I just skipped them, but later I realized they pointed to actual mistakes like wrong imports).

3. If this app grows big, what architectural patterns would you use?

-Looking at how my project evolved from Day3 to Day5, I can already see that if it keeps growing, everything in one place will get messy. If I were to scale it:

- I would separate components more clearly — for example, TaskList, TaskCard, SearchBar, and ThemeToggle could all live in their own folders.
- I would definitely need a *state management library* (like Redux or Zustand) because right now props are being passed around and that will become confusing as more features get added.

- I’d also think about *file structure*: keeping pages/ for routing pages (Home, About, Add Task), components/ for reusable UI, and maybe hooks/ for custom hooks.
- Since I faced problems with react-scripts and packages, I’d probably set up a more modern build system like Vite if I were starting fresh.
- Finally, I would plan for performance — maybe lazy loading routes, memoizing expensive components, and keeping localStorage usage minimal (because it’s not designed for huge data).