Day 28 — Final Project: Small Dashboard (Auth + CRUD + Styled UI)

Week 4 — Final Project

Goal: Build a small but complete dashboard app that includes authentication, CRUD operations (connected to your FastAPI backend), routing, state management with Redux Toolkit + RTK Query, a responsive sidebar ("burger"/drawer), and polished UI. Follow the step-by-step plan, implement features incrementally, and use the recommended hooks (`useEffect`, `useMemo`, `useCallback`, `useRef`) in the places shown.

NOTE about "Xeos" / "Xeos API"

You mentioned **Xeos API**. I checked — there isn't a common JS HTTP client named Xeos. The industry-standard choice for HTTP in 2025 is **Axios** (or RTK Query's `fetchBaseQuery` when using RTK Query). If by "Xeos" you meant something else, tell me and I will adapt. For this project we'll use **Axios** for custom services and **RTK Query** for server-state.

Project Overview (High level)

1. Auth (mock -> JWT later)
2. App shell (Navbar + Sidebar / Drawer + Content area)
3. Pages: Dashboard (stats), Tasks (CRUD list), Add/Edit Task, Profile
4. State: Redux Toolkit + RTK Query for tasks + small UI slices
5. Styling: Tailwind + Material UI for components (cards, buttons)
6. Hooks: `useEffect`, `useMemo`, `useCallback`, `useRef` where appropriate
7. Extras: Optimistic UI, toasts, loading & error states, responsive sidebar
8. Deploy frontend + backend when ready (Vercel / Netlify + any FastAPI host)

Folder Structure (recommended)

```
/src
 ├─ /app
 │   ├─ store.js
 │   └─ api.js            // RTK Query api slice
 ├─ /features
 │   ├─ /auth
 │   │   ├─ authSlice.js
 │   │   └─ Login.jsx
 │   ├─ /tasks
 │   │   ├─ TasksRTK.jsx
 │   │   └─ taskSlice.js (optional)
 │   └─ /ui
 │       └─ uiSlice.js    // sidebar open/close, theme
 ├─ /components
 │   ├─ Navbar.jsx
 │   ├─ Sidebar.jsx
 │   ├─ ProtectedRoute.jsx
 │   └─ Spinner.jsx
 ├─ /pages
 │   ├─ Dashboard.jsx
 │   ├─ TasksPage.jsx
 │   ├─ TaskEdit.jsx
 │   └─ Profile.jsx
 ├─ /api
 │   └─ axios.js         // optional axios instance for non-RTK calls
 ├─ App.jsx
 └─ main.jsx
```

Step-by-step Implementation Plan (do these in order)

Step 0 — Setup & Dependencies

- Initialize React app (Vite recommended) and start dev server.
- Install dependencies:

```
npm install @reduxjs/toolkit react-redux @mui/material @mui/icons-material @emotion/react @emotion/styled axios react-router-dom react-hot-toast tailwindcss
```

- Configure Tailwind (if using).

Why first? - Get the skeleton and tooling ready so later steps are plug-and-play.

Step 1 — Global Store + RTK Query

1. Create `src/app/api.js` using `createApi` + `fetchBaseQuery` (or use Axios adapter if you prefer).
2. Add endpoints for tasks and auth (login) mapped to your FastAPI paths.
3. Configure `store.js` to include `api.reducer` and `api.middleware`.
4. Wrap your app with `<Provider store={store}>` in `main.jsx`.

Why: RTK Query will replace manual fetch/useEffect for tasks and provide caching & invalidation.

Step 2 — Authentication Flow + Protected Routes

1. Create `features/auth/authSlice.js` that stores `user` (or token). For now, implement **mock login** that stores user in slice.

2. Create `Login.jsx` page with form and validation (use local UI state and form validation).
3. Create `ProtectedRoute.jsx` (or wrapper) that checks `useSelector(state => state.auth.user)` and redirects to `/login` if missing.
4. Add routes in `App.jsx`: `/login`, `/dashboard`, `/tasks`, `/tasks/:id/edit`, `/profile`.

Where to use hooks: - `useEffect` in Login only if you need to check existing auth token on mount.

Step 3 — App Shell: Navbar + Sidebar (Drawer / Burger)

1. Build `Navbar.jsx` (logo, user menu, logout button).
2. Build `Sidebar.jsx` as a responsive drawer. Use MUI `<Drawer>` or Tailwind + CSS for mobile burger toggle.
3. Control open/close with a UI slice (`uiSlice.js`) or local state in a top-level layout component.
4. Use `useRef` for focus management (e.g., close drawer and focus first link), and `aria` for accessibility.

Tips: - Sidebar items: Dashboard, Tasks, Add Task, Profile, Logout - On small screens show burger icon in Navbar that toggles drawer

---

Step 4 — Tasks Page (RTK Query-driven)

1. Replace manual fetch component with RTK Query hooks: `useGetTasksQuery`, `useAddTaskMutation`, `useUpdateTaskMutation`, `useDeleteTaskMutation`.
2. Show loading state via `isLoading`; show error via `error`.
3. Use MUI `<Card>` for each task or Tailwind cards.
4. Add inline edit button that navigates to `/tasks/:id/edit` (or open inline modal).

Where to use hooks: - `useEffect`: rarely needed; RTK Query handles cache — only for side effects (e.g., focus input on mount). - `useMemo`: memoize derived values (e.g., filtered task lists, expensive computations) — `const filtered = useMemo(() => tasks.filter(...), [tasks, filter])`. - `useCallback`: memoize event handlers passed down to memoized child components (`TaskItem`) — `onDelete = useCallback((id)=> deleteTask(id), [deleteTask])`.

Optimistic UI: implement optimistic updates in RTK Query `onQueryStarted` or do manual state patching for immediate feel.

Step 5 — Add / Edit Task Forms

1. Build `TaskEdit.jsx` with controlled inputs and validation. Use `useRef` for first input focus.
2. On submit, call `updateTask` mutation (RTK Query) or `addTask` for new.
3. After success, navigate back to `/tasks` using `useNavigate()`.

Where to use hooks: - `useEffect`: to load task data into form when editing (if not preloaded). - `useRef`: focus, or keep a ref to the form element.

Step 6 — Dashboard Page (stats + summaries)

1. Use `useGetTasksQuery` to read data and compute summary numbers: total tasks, completed, pending.
2. Use `useMemo` to compute heavy aggregations (only recompute when `tasks` change).
3. Show charts/cards (MUI or simple SVG) summarizing status.

Step 7 — Profile & User Menu

- Show logged-in user info from `authSlice`.
- Implement logout which clears the slice and redirects to `/login`.

Step 8 — UI Polish & Accessibility

- Replace plain buttons with MUI for consistent look (`Button`, `IconButton`, `Drawer`, `AppBar`).
- Add toasts using `react-hot-toast` for success & error messages.
- Ensure keyboard accessibility (focus trap in drawer, `aria` attributes).

Step 9 — Testing & QA

- Add small unit tests for critical components (Login, Tasks list) using Jest + RTL. Test behavior: add, delete, edit flows.
- Manually test on mobile sizes and keyboard navigation.

Step 10 — Deployment Checklist

- Build backend (FastAPI): ensure CORS safely configured for production origin.
- Build frontend (`npm run build`) and deploy to Vercel/Netlify. Or host frontend on static host and FastAPI on Railway/Render/DigitalOcean.
- Add environment variables (API URL) in deployment settings.

Where to use specific hooks — quick guide

- **useEffect**: side-effects (initial data fetch if not using RTK Query; subscribe/unsubscribe; focus management). Avoid overuse.
- **useMemo**: memoize derived data / expensive computations (filtering, sorting, aggregates).
- **useCallback**: memoize functions passed to children to avoid re-renders (use when child is memoized or you see re-renders). Example: handlers in a `TaskItem` list.
- **useRef**: store DOM refs (focus inputs), or keep mutable values across renders (e.g., debounce timer id), or to access previous value.

Componentization recommendations

- `Layout` (wraps Navbar + Sidebar + main content)
- `TaskItem` (memoized) — single task card with `onDelete`, `onToggleComplete`, `onEdit`
- `TaskList` — maps tasks to `TaskItem`s
- `TaskForm` — used for add & edit
- `ProtectedRoute` — wrapper for authenticated routes
- `Spinner` — small loading component

Performance tips

- Memoize lists (use `useMemo` for derived filtered arrays).
- Use `React.memo` on `TaskItem` to prevent re-render when unrelated state changes.
- Use `useCallback` for handlers passed to memoized children.
- Keep components small and focused.

Small Implementation Schedule (4 sessions / days)

- **Session 1:** App shell + store + RTK Query endpoints + simple TasksRTK list
- **Session 2:** Auth, ProtectedRoute, Login page, Sidebar + Navbar responsive
- **Session 3:** Task forms (Add/Edit), optimistic updates, toasts
- **Session 4:** Dashboard stats, polish UI, tests, deploy

Deliverables (what to push to repo)

- README with setup steps (how to run frontend & backend)
- `src/app` with store & api
- `src/features/tasks/TasksRTK.jsx`, `TaskForm.jsx`
- `src/features/auth/Login.jsx`, `authSlice.js`
- `src/components/Navbar.jsx`, `Sidebar.jsx`, `ProtectedRoute.jsx`
- Basic tests for `TasksRTK` and `Login`

Example code snippets (short)

RTK Query base (app/api.js)

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';
export const api = createApi({
  reducerPath: 'api',
  baseQuery: fetchBaseQuery({ baseUrl: process.env.VITE_API_URL }),
  tagTypes: ['Tasks'],
  endpoints: (builder) => ({
    getTasks: builder.query({ query: () => '/gettasks', providesTags:
```

```
['Tasks'] }),
    addTask: builder.mutation({ query: (task) => ({ url: '/addtask', method:
'POST', body: task }), invalidatesTags: ['Tasks'] }),
    updateTask: builder.mutation({ query: ({ id, ...data }) => ({ url: `/
updatetask/${id}`, method: 'PUT', body: data }), invalidatesTags: (result,
error, { id }) => [{ type: 'Tasks', id }] }),
    deleteTask: builder.mutation({ query: (id) => ({ url: `/deltask/${id}`,
method: 'DELETE' }), invalidatesTags: ['Tasks'] }),
  }),
});
export const { useGetTasksQuery, useAddTaskMutation, useUpdateTaskMutation,
useDeleteTaskMutation } = api;
```

ProtectedRoute.jsx

```
import { useSelector } from 'react-redux';
import { Navigate } from 'react-router-dom';
export default function ProtectedRoute({ children }){
  const user = useSelector(state => state.auth.user);
  return user ? children : <Navigate to="/login" replace />;
}
```

Exercise — Final Project (Step-by-step)

Follow this checklist and implement each item one-by-one. Mark as done in your README.

1. [ ] Initialize project + install deps
2. [ ] Create RTK Query `api.js` + configure store
3. [ ] Implement `TasksRTK.jsx` to list tasks
4. [ ] Add Add/Delete/Update actions using RTK Query hooks
5. [ ] Create Login page + `authSlice` (mock ok)
6. [ ] Add ProtectedRoute and protect `/dashboard` and `/tasks`
7. [ ] Build responsive Navbar + Sidebar (burger for mobile)
8. [ ] Implement `TaskForm` for add/edit; focus first input with `useRef`
9. [ ] Add toasts for success & error (react-hot-toast)
10. [ ] Implement optimistic updates for add/delete (onQueryStarted)
11. [ ] Add Dashboard stats page (useMemo for derived data)
12. [ ] Add minimal tests for Tasks list and Login
13. [ ] Deploy frontend & backend; set `VITE_API_URL` in env

If you want, I can now generate:
- A full code scaffold (copy/paste-ready) for this project, or
- A smaller starter repo zip with files, or
- Walk you through Session 1 step-by-step and produce the code live in the canvas.

Which do you want next?