

PMA: a multi-workspace project-management application, delivered by my team

Several of the businesses I run technology for needed a lightweight, self-hosted alternative to Jira or Asana: multiple companies, each with their own workspaces, projects and people, without paying per seat for features nobody...

Role: Technical lead and product owner; built by a developer on my team under my direction and review

Stack: Node and TypeScript API (Express, Mongoose, Passport with local and Google OAuth, Zod validation), React client (TanStack Query and Table, Radix UI, react-hook-form)

Scope: Workspaces, projects, tasks, members, roles and permissions, invitations

SIGN-IN SCREEN, BUILT CLIENT FROM THE REPOSITORY

Context

Several of the businesses I run technology for needed a lightweight, self-hosted alternative to Jira or Asana: multiple companies, each with their own workspaces, projects and people, without paying per seat for features nobody used. I scoped and specified PMA and had it built by a developer on my team, reviewing architecture and code as it went. This page describes what was delivered and the decisions I made about how it should be built.

What was built

The data model

Workspace contains projects, projects contain tasks, and membership is per workspace. Tasks carry a status lifecycle (Backlog, To Do, In Progress, In Review, Done) and a priority (Low, Medium, High). Every entity is validated at the edge with Zod schemas so the API rejects malformed input before it reaches a service.

Roles and permissions, not just roles

Three roles (Owner, Admin, Member) map onto fourteen explicit permissions: create, edit and delete for workspaces, projects and tasks, member management, settings, and a view-only permission. A role guard resolves the member's role in the current workspace and checks the specific permission on each route, so adding a fourth role later means editing a table, not the controllers. Roles are seeded at first run.

Authentication

Passport handles both email-and-password and Google OAuth, with cookie sessions. Accounts are modelled separately from users, so one person can hold more than one sign-in provider.

The client

A React application with sign-in and sign-up (including the Google failure path), a workspace dashboard, project details, a tasks view built on TanStack Table, members management with role changes, workspace settings, and an invitation flow that lets a user join a workspace from a link.

Structure

Controllers, services, models, routes, middlewares and utilities in separate layers, an async-handler wrapper, a central error handler and a typed error-code enum, so failures come back to the client in a consistent shape.

Decisions worth noting

- ◆ **Permissions as data.** I insisted on the explicit permission table rather than role checks scattered through the code. It is the difference between a tool a company can adapt and one it has to fork.
- ◆ **Validation at the boundary.** Zod on every request body means the services can trust their inputs and stay small.
- ◆ **Own it rather than rent it.** For a small group of companies the cost of a SaaS project tool compounds; a self-hosted tool with exactly the features needed does not.

Why it is in the portfolio

It is the clearest example on this record of how I run a team build: define the model and the permission rules up front, hold the line on structure, review the code, and ship something the businesses actually use. The commits are my developer's; the architecture and the accountability are mine.