Pet Project: Task Management System

Technical Requirements and Development Plan (no sprints)

Project idea: Build a task management system (mini Trello/Jira) including a Web API, gRPC services, React frontend, and CI/CD automation.

Backend — .NET 8 / ASP.NET Core

- Architecture: Clean Architecture or Onion Architecture
- Layers: API, Application, Infrastructure, Domain
- REST API with CRUD operations, JWT authentication, FluentValidation, and error handling middleware

gRPC

- Use for inter-service communication (e.g., Tasks ↔ Notifications)
- Define .proto files, implement server and client in .NET
- Support gRPC-Web for browser access (optional) and unit tests for gRPC methods

Entity Framework / Dapper

- Primary ORM: EF Core; use Dapper for optimized complex queries
- Code-first approach, migrations, target DB: PostgreSQL or SQL Server
- Domain entities examples: User, Project, Task

Unit Tests — xUnit / NUnit

- Aim for at least 70% coverage of business logic; mock dependencies with Moq or NSubstitute
- Separate test project (e.g., ProjectName.Tests); use in-memory DB for integration tests

Logging

- Use Serilog; log exceptions, API requests/responses, and gRPC calls
- Outputs: console for dev, file or Seq/Elasticsearch for production
- Configure log levels via appsettings.json

CI/CD — Azure DevOps / AWS

- CI: build and run tests on every push; include linting and static analysis
- CD: deploy container to Azure App Service or AWS Elastic Beanstalk / ECS
- Use Docker and docker-compose; store secrets in Azure Key Vault or AWS Secrets Manager

Frontend — ReactJS

- React 18 + TypeScript; UI toolkit: MUI, Ant Design, or TailwindCSS
- Use React Query or RTK Query for data fetching; JWT auth stored securely (consider HttpOnly cookies)
- Pages: Login/Registration, Projects list, Tasks list, Task details

Autogenerated Documentation

- Use Swagger / Swashbuckle for Web API (OpenAPI spec available at /swagger)
- Generate gRPC documentation with protoc-gen-doc
- Provide README.md with setup and docker-compose for local development

Additional Requirements

- Containerization: Docker; environment-specific configuration files
- Caching: in-memory and option for Redis; health checks endpoint (/health)
- Recommended project structure: Api, Application, Domain, Infrastructure, GrpcServices, WebClient, Tests

Development Plan

- Step 1: Initialize repository, define project structure and basic CI pipeline
- Step 2: Implement core domain and database models, migrations, and basic CRUD API
- Step 3: Add authentication (JWT), validation, and error handling
- Step 4: Implement gRPC services and integrate with backend
- Step 5: Build React frontend and connect to the API (and gRPC-web if used)
- Step 6: Add logging, monitoring, and finalize CI/CD pipelines; prepare autogenerated documentation