SPEC-1: Secure, Scalable FastAPI-Based ToDo App

# Background

The goal of this project is to create a secure, modular, and scalable ToDo web application using FastAPI. It follows OWASP secure coding practices, and is structured for progressive, maintainable development. This system includes registration, login, RBAC, user-specific ToDos, admin role management, logout, testing, Docker deployment, and CI/CD support.

# Requirements

Must Have:

- Secure registration system with email/password and hashed credentials

- Login system with role-based access control (RBAC)

- User-specific ToDo creation, listing, and deletion

- Admin dashboard for role management

- Logout and session clearing

- Unit testing using `pytest`

- Dockerization and `.env` support

- GitHub Actions CI/CD

Should Have:

- HTTPS setup guide (external Nginx + Certbot)

- UI enhancements using Tailwind or Bootstrap

Could Have:

- REST API for external clients

- JWT-based auth as an alternative to signed cookies

# Method

Architecture Overview:

- Web Frontend includes Home Page, Register Form, Login Form, Dashboard

- FastAPI App includes routes for registration, login, dashboard, todos, admin, and auth

Database Schema (SQLModel):

User Table:

- id: int (PK)

- email: str (unique)

- hashed\_password: str

- role: str (default: "user")

- created\_at: datetime

ToDo Table:

- id: int (PK)

- title: str

- description: str

- completed: bool

- created\_at: datetime

- user\_id: int (FK -> User.id)

Security Features:

- Passwords hashed using `passlib[bcrypt]`

- Signed cookies for session via `itsdangerous`

- RBAC: Admin-only access to `/admin`

- Form validation on both frontend and backend

- Cookie flags: `secure=True`, `httponly=True`

Configuration:

- `.env` used for secret and DB connection

- Dockerized using a multi-stage build

- GitHub Actions for CI (lint, type-check, test, coverage)

# Implementation

1. Scaffold project structure with FastAPI, Jinja2 templates

2. Create landing page with navbar and branding

3. Add `/register` route with form validation and secure persistence

4. Add `/login` route, hashed credential check, signed cookie session

5. Implement `/dashboard` with user greeting

6. Add `/todos` (CRUD) linked to logged-in user ID

7. Create `/admin` for role management (admin only)

8. Add `/logout` to clear cookies

9. Write unit tests for auth and ToDos

10. Containerize with Dockerfile and `.env` support

11. Add GitHub Actions CI pipeline

# Milestones

1. ✅ Project scaffold and landing page

2. ✅ Secure registration module

3. ✅ Login with session and RBAC

4. ✅ User dashboard and ToDo CRUD

5. ✅ Admin panel with role management

6. ✅ Logout and UI updates

7. ✅ Unit test coverage and testability

8. ✅ Dockerization and `.env`

9. ✅ CI/CD with GitHub Actions

# Gathering Results

- Use `pytest --cov` to measure test coverage (target: >90%)

- Manually verify access control via browser tests

- Check login/logout flows across roles

- Monitor production containers for memory and CPU usage

- Ensure secure headers and cookies in browser dev tools

generate a gpt prompt for a fastapi based todo app. it should not have security flaws and should follow owasp secure coding guidlenes. the app should keep scalability in mind and all the dependecies should be listed in requirements.txt readme file should contain details about the app with running instructions. you should also generate a prompt to generate project structure and commands to create that structure. The projects should have a landing home page which displays a logo in left and few navigation links in right. in the content area of landing page it should contain few details about appwill register and login screen. The prompts should be generated in step by step manner so that when first prompt is run it creates project sturucture and other basic details listed above and second module create another module and third prompt generates anopther modules etc etc. the prompts should be progressive prompts will when run keeps updating the main app accordngly. the app should have following modules registration module with role based access control registration page displays a beatutiful form with email and password field. the email should be vallidated for correctness and password length should be atleast 8 chacraters. when form is submitted the users is regsitered and stored in sqlite database. the password should be hashed before storing in database. the backend should also validate email and password length. Users is registered with default role of user and date of registration is also stored in database Login system with authorization the app should display a login page with email and password field. here also we should validate email and password length in frontend accordingly. once the login form is submitted, it verfies users against information stored in sqlite table if the users is verfied he is redirected to members dashboard and wlecome message is shown. if user is not verfied, login failure message should be displayed