Task Manager API

Student Name: Sanket Jadhav

Roll No: 192

Year & Section: 3rd Year, AI–ML

Project Title: Task Manager API

Project Type: Application Developer

Stack / Framework: FastAPI, SQLModel/SQLAlchemy, Pydantic

1. Problem Understanding

1.1 What is the problem statement in your own words?

Teams and developers need a simple, efficient backend system to manage tasks, comments, and labels in a collaborative workflow. The problem is the lack of a lightweight, structured API that supports relations, filtering, and communication between collaborators.

1.2 Why does this problem exist or matter?

- Helps developers build task-based systems quickly
- Useful for collaboration tools, productivity apps, and team workflows
- Makes backend API design, database relations, and filtering easy to integrate
- Provides hands-on experience for real-world API development

1.3 Key inputs and expected outputs

Inputs	Process	Expected Outputs
Task data, comment data, label data (JSON)	CRUD operations, validation, relational mapping	Created/updated tasks, comments, labels
Filter requests (label, status, priority)	Query filtering, DB queries	Filtered task list

Seed command	Populate demo data	Ready-to-use DB with sample tasks

2. Functional Scope

2.1 Core Features (Must-haves)

- Task CRUD (create, read, update, delete)
- Comments attached to tasks
- Label system with many-to-many relations
- Filters (labels, status)
- Database relations: Task ↔ Comments ↔ Labels
- API documentation with Swagger
- Database seed script

2.2 Stretch Goals

- Activity log for all task operations
- Sorting & advanced filtering(modify a query)

2.3 Libraries or tools used

- **FastAPI** — main framework
- **SQLModel / SQLAlchemy** — ORM + DB models
- **SQLite / PostgreSQL** — storage
- **Pydantic** — schema validation
- **Uvicorn** — server
- **pytest** — testing

- **FastAPI AutoDocs** — Swagger / Redoc
- **python-dotenv** — env management

3. System & Design Thinking

3.1 App flow / pipeline (bullet flow)

1. Client sends request (Task / Comment / Label)
2. FastAPI router receives request
3. Validation via Pydantic
4. Service layer processes logic
5. ORM interacts with DB
6. API returns structured JSON response
7. Optional email mock triggers
8. Seed script used to pre-populate DB

3.2 Key data structures / algorithms

- Relational database tables (Task, Comment, Label)
- Many-to-many mapping table for labels
- Hash maps for label filtering
- Query filtering algorithms
- CRUD workflow patterns

3.3 Testing approach

- Unit tests for each CRUD operation
- Integration tests for relations (task–comments–labels)
- Tests for validation & error states

- Expected test count: **6+ tests**

4. Timeline & Milestones (4 Weeks)

Week	Planned Deliverables	Mentor Checkpoint
W1	Schema design, DB setup, models (Task, Comments, Labels)	☐
W2	CRUD for all entities, relations & seed script	☐
W3	Filters, testing (6 tests)	☐
W4	Documentation, README, final demo, polishing	☐

5. Risks & Dependencies

5.1 Hardest technical challenges

- Understanding SQLModel relations
- Implementing many-to-many labels
- Writing unit/integration tests

5.2 Mentor help required

- Review data models & relations
- Suggestions for efficient filtering
- Guidance for writing clean test cases
- Review API documentation & demo video

6. Evaluation Readiness

6.1 Proof that the project works

- Working REST API (via Swagger)
- Seed data loaded successfully
- Test cases passing
- Demo video (2–3 mins)
- GitHub repo (code + docs)
- Screenshots / Postman collections

6.2 Success metrics / goals

- 100% CRUD functionality
- Labels filter working correctly
- API returns correct relational data
- Minimum 6 tests pass
- Complete README + documentation

7. Responsibilities

7.1 Responsibility Table

Task	Student Name	Mentor Notes
Task 1: Schema & models	Sanket Jadhav	☐
Task 2: CRUD APIs	Sanket Jadhav	☐
Task 3: Relations + Filters	Sanket Jadhav	☐
Task 4: Extras (activity log)	Sanket Jadhav	☐
Task 5: Docs + Demo	Sanket Jadhav	☐

Signatures (Students):

Mentor Approval:

Date: