M VARUN REDDY – PROJECT MANAGEMENT SYSTEM

Project: Project Management System (Console-Based, Python + SQLite)

1. **Introduction**

   The purpose of this document is to provide a detailed Low-Level Design (LLD) for a simplified, console-based Project Management System. The system facilitates streamlined work management, including user access control, project tracking, task handling, comments, labels, and basic reporting. It operates entirely via a terminal interface and persists data using a local SQLite database. This design is implemented with Python and SQL only, without any web frontends or network APIs.

2. **Module Overview**

   The project consists of the following modules:

   2.1 User and Authentication

- Manages user accounts, minimal roles (admin, member), login/logout, and in-session identity.

   2.2 Project Management

- Handles creation, updating, listing, viewing, and archival of projects with status and schedule metadata.

   2.3 Task Management

- Provides CRUD for tasks within projects, assignment to users, status transitions, priorities, due dates, and basic time estimates.

   2.4 Comments

- Supports adding and listing textual comments on tasks to enable collaboration and context.

   2.5 Labels and Tagging

- Enables creating labels per project and attaching labels to tasks for categorization and filtering.

   2.6 Activity Log

- Records lightweight audit events (who, what, when) for key actions across core entities.

   2.7 Reporting

- Generates console reports for project status summaries, overdue tasks, and user workload.

3. **Architecture Overview**

   3.1 Architectural Style

- Interface: Console-based CLI with command routing and interactive prompts.
- Application: Modular Python services orchestrated by a command dispatcher.
- Database: Relational database using SQLite with SQL scripts and Python's standard DB-API.

3.2 Component Interaction

1. The console command layer parses input and dispatches to the relevant service.
2. Services perform validation and business rules, then call repository functions.
3. Repository functions execute SQL against SQLite and return results to services.
4. Services format and print results back to the console.

Module-Wise Design

4.1 User and Authentication Module

4.1.1 Features

- Register users (admin-only), login, logout, whoami.
- Enforce minimal role-based checks for protected actions (e.g., archival, user management).

4.1.2 Data Flow

- User issues auth commands in the console.
- Credentials are validated against stored password hashes in SQLite.
- In-memory session state tracks the current user/role during the process lifetime.

4.1.3 Entities

- User
- UserID
- Username
- Email
- PasswordHash
- Role (ADMIN, MEMBER)
- Active
- CreatedAt

4.2 Project Management Module

4.2.1 Features

- Create, update, list, show, and archive projects.
- Filter by status, owner, priority, and date range.

4.2.2 Data Flow

- User runs project commands.
- Service validates fields and owner existence, persists via repository, and prints details or lists.

4.2.3 Entities

- Project
- ProjectID
- Name
- Description
- OwnerID (User)
- Status (Planned, Active, On Hold, Completed, Archived)
- Priority (Low, Medium, High, Critical)
- StartDate
- EndDate
- Archived
- CreatedAt

4.3 Task Management Module

4.3.1 Features

- Create, update, assign tasks; move between Backlog, In Progress, Done.
- Set priority, due date, estimate/actual hours; list/filter by project, assignee, status, due date.

4.3.2 Data Flow

- User runs task commands (create/update/assign/move/list/show/delete).
- Service validates project and assignee, enforces status transitions, updates DB, logs activity, and prints output.

4.3.3 Entities

- Task
- TaskID

- ProjectID
- Title
- Description
- AssigneeID (User, nullable)
- Status (Backlog, In Progress, Done)
- Priority (Low, Medium, High, Critical)
- EstimateHours
- ActualHours
- DueDate
- CreatedAt

4.4 Comments Module

4.4.1 Features

- Add comments to tasks; list comments in chronological order.

4.4.2 Data Flow

- User issues comment add/list commands.
- Service validates task and user, persists comment, and prints the conversation.

4.4.3 Entities

- Comment
- CommentID
- TaskID
- UserID
- Body
- CreatedAt

4.5 Labels and Tagging Module

4.5.1 Features

- Create labels per project; attach/detach labels on tasks; filter tasks by label.

4.5.2 Data Flow

- User manages labels and mappings via console commands.
- Service validates label and task association and updates mapping table.

4.5.3 Entities

- Label
- LabelID
- ProjectID
- Name
- TaskLabel
- TaskID
- LabelID

4.6 Activity Log Module

4.6.1 Features

- Record minimal audit events: CREATE, UPDATE, ASSIGN, MOVE, ARCHIVE.
- Query activity per entity for traceability.

4.6.2 Data Flow

- On significant actions, the service writes an activity log entry.
- Users list activity by entity to review recent changes.

4.6.3 Entities

- ActivityLog
- ActivityID
- UserID
- EntityType (User, Project, Task, Label, Comment)
- EntityID
- Action (CREATE, UPDATE, DELETE, ASSIGN, MOVE, ARCHIVE)
- CreatedAt

4.7 Reporting Module

4.7.1 Features

- Project status summary: counts by task status and completion ratio.
- Overdue tasks per project and per user.
- User workload: assigned task counts and sum of estimates/actuals.

4.7.2 Data Flow

- User runs report commands.
- Service aggregates via SQL queries and prints tabular summaries.

5. Command-Line Interface Desig

5.1 Command Structure

- auth: register, login, logout, whoami
- project: create, update, list, show, archive
- task: create, update, assign, move, list, show, delete
- comment: add, list
- label: create, list, add-to-task, remove-from-task
- report: project-status, overdue, user-workload

5.2 Interaction Patterns

- Commands support flags; interactive prompts fill missing arguments.
- Inline validation with clear error messages; confirmations for destructive actions.
- Tabular console output for lists and reports; concise detail views.

6. Database Design

6.1 Schema

6.1.1 User

Primary Key: UserID

Unique: Username, Email

Fields: PasswordHash, Role, Active, CreatedAt

6.1.2 Project

Primary Key: ProjectID

Foreign Key: OwnerID → User(UserID)

Fields: Name, Description, Status, Priority, StartDate, EndDate, Archived, CreatedAt

6.1.3 Task

Primary Key: TaskID

Foreign Keys: ProjectID → Project(ProjectID), AssigneeID → User(UserID, nullable)

Fields: Title, Description, Status, Priority, EstimateHours, ActualHours, DueDate, CreatedAt

6.1.4 Comment

Primary Key: CommentID
Foreign Keys: TaskID → Task(TaskID), UserID → User(UserID)
Fields: Body, CreatedAt

6.1.5 Label and TaskLabel

Label: Primary Key: LabelID; Foreign Key: ProjectID → Project(ProjectID)
TaskLabel: Composite Primary Key: (TaskID, LabelID); Foreign Keys: TaskID → Task(TaskID), LabelID → Label(LabelID)

6.1.6 ActivityLog

Primary Key: ActivityID
Foreign Key: UserID → User(UserID)
Fields: EntityType, EntityID, Action, CreatedAt

6.2 Constraints and Indexing

- CHECK constraints on Status and Priority fields.
- NOT NULL on required fields; Active defaults to true.
- Indexes: Task(ProjectID, Status), Task(AssigneeID, Status), Task(DueDate), ActivityLog(EntityType, EntityID).
- PRAGMA foreign_keys = ON on connection.

7. User Interface Design

7.1 Screens/Flows (Console)

- Authentication flow: register, login, session banner (current user).
- Project listing and detail view with quick stats.
- Task list grouped by status; task detail with comments and labels.
- Reports printed as tables with totals and highlights for overdue items.

7.2 Usability

- Consistent command naming and help text.
- Clear error messages and confirmation prompts.
- Optional colorized output for status and priority to improve readability.

8. Non-Functional Requirements

8.1 Performance

- Responsive operation on local datasets up to tens of thousands of tasks; transactions for bulk operations.

8.2 Scalability

- Designed for single-machine use; suitable for individual or small team workflows on a shared machine.
- **8.3 Security**
- Passwords stored as strong hashes; least-privilege via role checks; rely on OS file permissions for the SQLite file.

8.4 Reliability

- Transactions around multi-table updates; defensive validation and error handling.

8.5 Usability

- Comprehensive help for each command; consistent tabular outputs; informative validation feedback.

8.6 Maintainability

- Layered structure (CLI, services, repositories); centralized enums and validation; schema SQL for migrations.

9. Assumptions and Constraints

9.1 Assumptions

- Local, single-process execution using a SQLite database file.
- All user interactions occur via the console.
- System clock and locale provided by the host environment.

9.2 Constraints

- No web UI, desktop GUI, or network APIs.
- No cloud deployment or client-server components.
- SQLite is the only persistence layer.
- Developed and executed with standard Python tooling in a local environment.