KanCLI: Command-Line Kanban

Backend Systems – Portfolio 01

Anna Gropp 5124124
Daniel Borgs 5124094
Emma Schüpfer 5124120

December 18, 2025

Contents

1 Project Overview

A terminal-based Kanban board that lets teams organize tasks across projects, assign work, and track progress through workflow states, all via CLI because context-switching to browsers breaks flow.

1.1 Relevance

Small development teams need a straightforward way to coordinate work without the overhead of complex web interfaces. Jira is overkill for simple task tracking and hampers staying in flow state. A terminal-native solution keeps developers in their natural workflow. Just `task move 5 --state done` and you're good.

Additionally, it teaches proper backend architecture patterns (hexagonal design, JPA relationships, concurrent access) while solving a real productivity problem for technical users who prefer the command line.

1.2 Essential Capabilities

1. **Users:** Each user has a username, password and email. No authorization complexity. Everyone can perform all actions.

2. **Project Organization:** Projects act as containers for tasks. Many-to-many user assignments are supported (users can belong to multiple projects; projects can have multiple users).

3. **Task Management:** Each task has a title, an optional description, a category `BUG`, `FEATURE`, `IMPROVEMENT`, or `RESEARCH`, and can be assigned to one user or remain unassigned. Tasks belong to a specific project.

4. **Kanban Workflow:** Tasks progress through three predefined states: `TODO`, `IN_PROGRESS` and `DONE`.

5. **Tagging System:** Tasks can be labeled with flexible tags. The system comes with pre-populated priority tags: high, medium, and low.

6. **Terminal Client:** A lightweight command-line interface (CLI) that communicates with the backend via HTTP requests. The client runs separately from the server.

7. **Concurrent Access:** Multiple users can interact with the system simultaneously using separate client instances.

8. **Testing:** The codebase includes unit tests to verify individual components of the domain logic and integration tests to ensure correct behavior of API endpoints.

1.3 Explicit Non-Goals (Out of Scope)

1. **Authentication and Authorization:** No roles. All users are trusted equally since the focus lies on backend architecture.

2. **Advanced Features:** No subtasks, due dates, time tracking, comments, file attachments, or notifications. The scope is limited to the core Kanban workflow.

2 Domain Model

2.1 The Entities (UML)

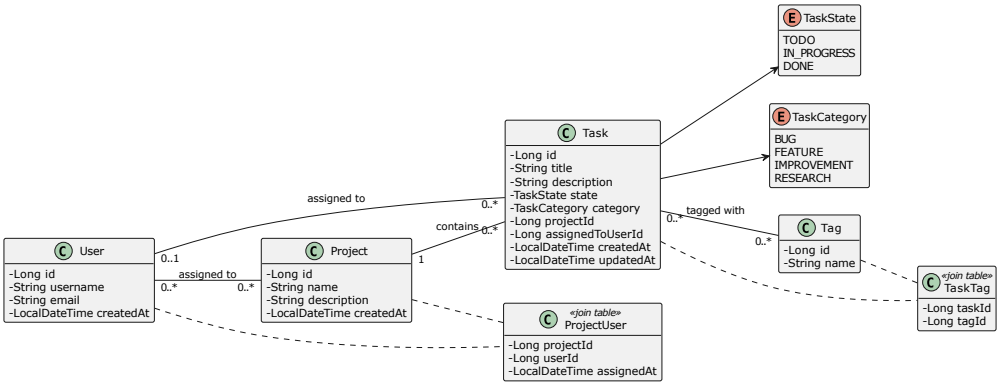

Figure 1: Domain model UML diagram

2.2 Description

1. **User:** Minimal entity for identification via `username` and `password`. Both username and email must be unique.

2. **Project:** Groups related tasks. Each project has a unique name and an optional description. Maintains a many-to-many relationship with users through the `ProjectUser` join table.

3. **Task:** The core entity. Every task belongs to exactly one project and can optionally be assigned to one user. Has a `state` (enum: `TODO`, `IN_PROGRESS`,

DONE) and a `category` (enum: `BUG`, `FEATURE`, `IMPROVEMENT`, `RESEARCH`). Title is required; description is optional.

4. **Tag:** Provides flexible labeling. Each tag has an ID and name. Tasks and tags are linked via the `TaskTag` join table. The system is pre-populated with priority tags (`high`, `medium`, `low`), though custom tags could be added later.

5. **ProjectUser (join table):** Links users and projects in a many-to-many relationship. Uses a composite key on (`projectId`, `userId`) and includes a timestamp indicating when the user was added to the project.

6. **TaskTag (join table):** Links tasks and tags via a composite key on (`taskId`, `tagId`).

2.3 Example Records

User:

```
{
  "id": 1,
  "username": "alice",
  "email": "alice@uni.edu",
  "hash": "<argon2id-hash>",
  "createdAt": "2025-10-15T09:00:00"
}
```

Project with users:

```
{
 "id": 1,
 "name": "Backend Systems Course",
 "description": "Final project for CS-401",
 "assignedUsers": [1, 2, 3],
 "createdAt": "2025-10-20T12:00:00"
}
```

Task with tags:

```
{
  "id": 5,
  "title": "Setup database schema",
  "description": "Create tables for User, Project, Task, Tag, and join
  ↪  tables",
```

```
  "state": "IN_PROGRESS",
  "category": "FEATURE",
  "projectId": 1,
  "assignedToUserId": 2,
  "tags": ["priority:high", "database"],
  "createdAt": "2025-10-21T08:30:00",
  "updatedAt": "2025-10-22T10:15:00"
}
```


3 Use Cases

3.1 UC-01: Create Project with Users

UC-01: Create Project with Users

Actor:	Any user
Goal:	Create a new project and assign people to it
Preconditions:	Users you want to assign must already exist
Flow:	1. Provide project name, description (optional), list of user IDs 2. System checks name is unique and all user IDs are valid 3. Creates project + ProjectUser records 4. Returns project with ID
Failure Cases:	• Project name already exists → system returns an error • One or more user IDs are invalid → system returns an error • No users specified → system creates project without assigned users
Acceptance Criteria:	• Project can be retrieved by its ID • Assigned users appear in the ProjectUser table • Duplicate project names are rejected • Invalid user IDs produce an error
Priority:	Must Have

3.2 UC-02: Assign User to Project

UC-02: Assign User to Project	
Actor:	Any user
Goal:	Add someone to an existing project
Preconditions:	Project and user both exist
Flow:	1. Provide project ID + user ID 2. Check both exist and user isn't already on project 3. Create ProjectUser record 4. Done
Failure Cases:	• Project doesn't exist → error • User doesn't exist → error • Already assigned → error (no duplicates)
Acceptance Criteria:	• User linked to project • Timestamp recorded • Duplicate check works
Priority:	Must Have

3.3 UC-03: Create Task

UC-03: Create Task

Actor:	Any user
Goal:	Create a work item in a project
Preconditions:	Project exists (+ user exists if assigning)
Flow:	1. Provide project ID, title, description, category, optional assignee 2. Validate project exists, title is valid (3-200 chars), category is valid enum 3. If assignee given, validate user exists 4. Create task with state = `TODO` 5. Return task with ID
Failure Cases:	• Invalid project → error • Invalid title → error • Invalid category → error • Invalid assignee → error • No assignee → fine, create unassigned
Acceptance Criteria:	• Task created with correct state (`TODO`) • Linked to project • Optional assignee works • Validation catches bad input
Priority:	Must Have

3.4 UC-04: Assign/Reassign Task

UC-04: Assign/Reassign Task

Actor:	Any user
Goal:	Set who's responsible for a task
Preconditions:	Task and user both exist
Flow:	1. Provide task ID + user ID 2. Validate both exist 3. Update task.assignedToUserId 4. Update task.updatedAt 5. Return updated task 6. Return task with ID
Failure Cases:	• Task doesn't exist → error • User doesn't exist → error • Reassigning → works fine, replaces old assignee • Unassigning (null user ID) → also fine
Acceptance Criteria:	• Assignee updated • Timestamp updated • Can reassign/unassign
Priority:	Must Have

3.5 UC-05: Move Task State

UC-05: Move Task State

Actor:	Any user
Goal:	Update task through Kanban workflow
Preconditions:	Task exists
Flow:	1. Provide task ID + new state (`TODO`, `IN_PROGRESS`, `DONE`) 2. Validate task exists and state is valid enum 3. Update task.state 4. Update task.updatedAt 5. Return updated task
Failure Cases:	• Task doesn't exist → error • Invalid state → error • Same state → technically fine (updates timestamp anyway)
Acceptance Criteria:	• State updated • Timestamp updated • All transitions allowed (no workflow enforcement) • Concurrent updates handled safely
Priority:	Must Have

3.6 UC-06: Add/Remove Tags

UC-06: Add/Remove Tags

Actor:	Any user
Goal:	Label tasks for organization/filtering
Preconditions:	Task exists, tag exists
Flow (Add):	1. Provide task ID + tag name 2. Validate task + tag exist 3. Check TaskTag doesn't already exist 4. Create TaskTag record 5. Update task.updatedAt 6. Return task with tags
Flow (Remove):	1. Provide task ID + tag name 2. Validate task + tag exist 3. Delete TaskTag record 4. Update task.updatedAt 5. Return task with tags
Failure Cases:	• Task/tag doesn't exist → error • Tag already on task (add) → error or ignore • Tag not on task (remove) → error or ignore
Acceptance Criteria:	• Tags added/removed • Timestamp updated • No duplicates • Pre-seeded priority tags available
Priority:	Should Have

3.7 Priority Summary

Use Case	Priority	Reason
UC-01: Create Project with Users	Must Have	Foundation
UC-02: Assign User to Project	Must Have	Team collaboration
UC-03: Create Task	Must Have	Core functionality
UC-04: Assign Task	Must Have	Responsibility tracking
UC-05: Move Task State	Must Have	Kanban workflow
UC-06: Manage Tags	Should Have	Nice to have, not critical