TRELLO LITE API

Software Requirements Document

Field	Value
Project	Trello Lite API
Version	1.0
Status	Draft
Phase	Backend Phase 2 Portfolio Project
Stack	Python · FastAPI · PostgreSQL · SQLAlchemy · Alembic
Architecture	Clean Architecture (Controller → Service → Repository)
Auth	JWT Bearer Token · bcrypt Password Hashing
Document Type	Software Requirements Document (SRD)

1. Introduction

1.1 Purpose

This document defines the complete functional and non-functional requirements for the Trello Lite API, a RESTful backend service built as a portfolio-grade project. It serves as the authoritative reference for development, testing, and review.

1.2 Project Overview

Trello Lite is a project and task management API inspired by Trello. It allows authenticated users to create projects, manage tasks within those projects, assign tasks to other users, filter and search

content, and enforce role-based access control. The system exposes a clean, versioned REST API consumed by any frontend client.

1.3 Scope

This SRD covers:

- User registration, authentication, and authorization
- Project CRUD operations with ownership rules
- Task CRUD operations with status and priority management
- Task assignment to users
- Pagination, filtering, sorting, and full-text search
- Role-based permissions (admin vs. regular user)
- Input validation and error handling
- Unit testing coverage

1.4 Definitions

Term	Definition
Admin	A privileged user with system-wide read and management access
Member	A regular authenticated user with access scoped to their own resources
Project Owner	The user who created a project; has full control over it
Project Member	A user explicitly added to a project; can view and interact with its tasks
Assignee	A user to whom a specific task has been assigned
JWT	JSON Web Token used for stateless authentication
SRD	Software Requirements Document
Repository	Data access layer; abstracts all database queries
Service	Business logic layer; orchestrates repositories
Controller/Router	HTTP layer; handles request parsing and response formatting

2. System Context

2.1 Tech Stack

Layer	Technology	Purpose
Language	Python 3.11+	Primary development language

Layer	Technology	Purpose
Framework	FastAPI	HTTP routing, request validation, OpenAPI docs
Database	PostgreSQL 15+	Relational data persistence
ORM	SQLAlchemy 2.x	Database abstraction and query building
Migrations	Alembic	Schema versioning and migration management
Auth	PyJWT + python-jose	JWT token generation and verification
Hashing	bcrypt (passlib)	Secure password storage
Validation	Pydantic v2	Request/response schema validation
Testing	pytest + httpx	Unit and integration testing
Env Mgmt	python-dotenv	Environment variable management

3. Entity-Relationship Description

3.1 Entities Overview

The system contains four core entities: User, Project, Task, and Assignment. The following describes their attributes and relationships.

3.2 User

Column	Type	Constraints	Description
id	UUID / SERIAL	PK, NOT NULL	Primary identifier
username	VARCHAR(50)	UNIQUE, NOT NULL	Unique display name
email	VARCHAR(255)	UNIQUE, NOT NULL	Login email address
hashed_password	VARCHAR(255)	NOT NULL	bcrypt hash of password
role	ENUM	NOT NULL, DEFAULT 'user'	Values: admin, user
is_active	BOOLEAN	NOT NULL, DEFAULT TRUE	Soft-disable account
created_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Account creation time
updated_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Last update time

3.3 Project

Column	Type	Constraints	Description
id	UUID / SERIAL	PK, NOT NULL	Primary identifier
name	VARCHAR(100)	NOT NULL	Project display name
description	TEXT	NULLABLE	Optional longer description
owner_id	FK → users.id	NOT NULL, ON DELETE CASCADE	User who created the project
is_archived	BOOLEAN	NOT NULL, DEFAULT FALSE	Archive flag (soft delete)
created_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Project creation time
updated_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Last update time

3.4 Task

Column	Type	Constraints	Description
id	UUID / SERIAL	PK, NOT NULL	Primary identifier
title	VARCHAR(200)	NOT NULL	Short task title
description	TEXT	NULLABLE	Detailed task description
status	ENUM	NOT NULL, DEFAULT 'todo'	Values: todo, in_progress, done
priority	ENUM	NOT NULL, DEFAULT 'medium'	Values: low, medium, high
due_date	DATE	NULLABLE	Optional deadline
project_id	FK → projects.id	NOT NULL, ON DELETE CASCADE	Owning project
created_by	FK → users.id	NOT NULL	User who created the task
created_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Task creation time
updated_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Last update time

3.5 Assignment

Column	Type	Constraints	Description
id	UUID / SERIAL	PK, NOT NULL	Primary identifier
task_id	FK → tasks.id	NOT NULL, ON DELETE CASCADE	The assigned task
user_id	FK → users.id	NOT NULL, ON DELETE CASCADE	The assignee
assigned_by	FK → users.id	NOT NULL	Who made the assignment
assigned_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Assignment timestamp
UNIQUE	(task_id, user_id)	Composite UNIQUE	Prevents duplicate assignments

3.6 Relationships

- User 1 → N Projects (owner_id). One user may own many projects.
- Project 1 → N Tasks (project_id). One project contains many tasks.
- User 1 → N Tasks (created_by). One user may create many tasks.
- Task N ↔ N User via Assignment. A task may be assigned to multiple users; a user may have multiple assignments.
- ProjectMember (optional join table): User N ↔ N Project for explicit project membership tracking.

4. Authentication & Authorization

4.1 Authentication Mechanism

The API uses JWT Bearer Token authentication. Tokens are stateless and stored client-side. There is no server-side session store.

- Token type: Bearer (Authorization header)
- Algorithm: HS256
- Access token expiry: 30 minutes (configurable via environment variable)
- Refresh token expiry: 7 days
- Payload claims: sub (user_id), role, exp, iat
- Passwords stored as bcrypt hashes with cost factor 12

4.2 Token Flow

1. Client POSTs credentials to POST /api/v1/auth/login
2. Server validates credentials, returns access_token and refresh_token

3. Client sends Authorization: Bearer <access_token> on all protected routes
4. Server decodes and validates JWT on every request via FastAPI dependency injection
5. Client uses POST /api/v1/auth/refresh to obtain new access token using refresh token

4.3 Roles

Role	Description	Typical Capabilities
admin	System administrator	Full read access to all resources; can deactivate users; can view all projects and tasks regardless of ownership
user	Regular authenticated user	CRUD on own projects and tasks; can be added to other projects; can see only their own and member projects

4.4 HTTP Status Codes for Auth

Scenario	Status Code
Valid credentials, token issued	200 OK
Missing or malformed Authorization header	401 Unauthorized
Valid token but insufficient permissions	403 Forbidden
Token expired	401 Unauthorized (with code: token_expired)
Invalid token signature	401 Unauthorized (with code: invalid_token)
Account inactive	403 Forbidden (with code: account_disabled)

5. Permission Matrix

5.1 User Management

Action	Admin	Owner (self)	Other User
Register new account	N/A (public)	N/A (public)	N/A (public)
View own profile	Yes	Yes	No
View any profile	Yes	No	No
Update own profile	Yes	Yes	No
Deactivate any user	Yes	No	No
Delete any user	Yes	No	No
List all users	Yes	No	No

5.2 Project Management

Action	Admin	Project Owner	Project Member	Non-Member
Create project	Yes	Yes	Yes	Yes
View project details	Yes	Yes	Yes	No
List all projects	Yes (all)	Yes (own+member)	Yes (own+member)	Own only
Update project	Yes	Yes	No	No
Archive project	Yes	Yes	No	No
Delete project	Yes	Yes	No	No
Add member to project	Yes	Yes	No	No
Remove member from project	Yes	Yes	No	No

5.3 Task Management

Action	Admin	Project Owner	Project Member	Task Creator
Create task in project	Yes	Yes	Yes	N/A
View task	Yes	Yes	Yes	Yes
List tasks in project	Yes	Yes	Yes	Yes
Update task (title/desc/status/priority)	Yes	Yes	Yes (own tasks)	Yes
Delete task	Yes	Yes	No	Yes
Change task due_date	Yes	Yes	No	Yes

5.4 Assignment Management

Action	Admin	Project Owner	Project Member	Assignee
Assign task to user	Yes	Yes	No	No
Unassign user from task	Yes	Yes	No	No
View task assignments	Yes	Yes	Yes	Yes
View own assignments (all projects)	Yes	Yes	Yes	Yes

6. Functional Requirements

6.1 User Authentication

- FR-AUTH-01: The system shall allow any visitor to register a new account with username, email, and password.
- FR-AUTH-02: The system shall hash all passwords using bcrypt before storage.
- FR-AUTH-03: The system shall issue a JWT access token and a JWT refresh token upon successful login.
- FR-AUTH-04: The system shall reject login attempts with invalid credentials with 401 and a generic error message (no enumeration of which field is wrong).
- FR-AUTH-05: The system shall allow token refresh via a dedicated endpoint without requiring re-authentication.
- FR-AUTH-06: The system shall return the authenticated user's profile at GET /api/v1/auth/me.
- FR-AUTH-07: The system shall support logout by invalidating the refresh token (blacklist stored in DB or by token version).

6.2 User Management

- FR-USER-01: Admins shall be able to list all users with pagination.
- FR-USER-02: Admins shall be able to retrieve any user's profile by ID.
- FR-USER-03: Any authenticated user shall be able to update their own username, email, and password.
- FR-USER-04: Admins shall be able to deactivate (soft-disable) any user account.
- FR-USER-05: Deactivated users shall receive 403 on all authenticated requests.

6.3 Project Management

- FR-PROJ-01: Authenticated users shall be able to create projects.
- FR-PROJ-02: Project creators automatically become the project owner.
- FR-PROJ-03: Owners shall be able to update project name and description.
- FR-PROJ-04: Owners shall be able to archive a project. Archived projects are excluded from default list views.
- FR-PROJ-05: Owners shall be able to hard-delete a project. All tasks cascade-delete.
- FR-PROJ-06: Owners shall be able to add and remove project members.
- FR-PROJ-07: The project list for a regular user shall return only projects where they are owner or member.
- FR-PROJ-08: Admins shall see all projects regardless of membership.

6.4 Task Management

- FR-TASK-01: Project owners and members shall be able to create tasks within a project.
- FR-TASK-02: Task creation shall capture: title, description (optional), status, priority, due_date (optional).
- FR-TASK-03: Task status shall follow this allowed set: todo, in_progress, done.
- FR-TASK-04: Task priority shall follow this allowed set: low, medium, high.
- FR-TASK-05: Project owners and task creators shall be able to update any field on a task.
- FR-TASK-06: Project members (non-owner) shall only update status on tasks assigned to them.
- FR-TASK-07: Project owners and task creators shall be able to delete a task.
- FR-TASK-08: The task list endpoint shall support filtering by status, priority, assignee, and due_date range.
- FR-TASK-09: The task list endpoint shall support sorting by created_at, due_date, and priority.
- FR-TASK-10: The task list endpoint shall support full-text search on title and description fields.

6.5 Task Assignment

- FR-ASSIGN-01: Project owners shall be able to assign any project member to any task within that project.
- FR-ASSIGN-02: A user must be a project member to be assignable to tasks within that project.
- FR-ASSIGN-03: Duplicate assignments (same task + same user) shall be rejected with 409 Conflict.
- FR-ASSIGN-04: Project owners shall be able to remove an assignment.
- FR-ASSIGN-05: An authenticated user shall be able to list all tasks assigned to them across all projects.
- FR-ASSIGN-06: Task detail responses shall include a list of assignees.

7. API Endpoint Reference

7.1 Base URL & Versioning

All endpoints are prefixed with /api/v1. Versioning is handled via URL path segment.

7.2 Authentication Endpoints

Method	Path	Auth Required	Description
POST	/api/v1/auth/register	No	Register a new user account
POST	/api/v1/auth/login	No	Login and receive JWT tokens
POST	/api/v1/auth/refresh	No (refresh token in body)	Issue new access token
POST	/api/v1/auth/logout	Yes	Invalidate refresh token
GET	/api/v1/auth/me	Yes	Get authenticated user profile

7.3 User Endpoints

Method	Path	Auth Required	Description
GET	/api/v1/users	Yes (Admin)	List all users with pagination
GET	/api/v1/users/{user_id}	Yes (Admin)	Get any user by ID
PATCH	/api/v1/users/me	Yes	Update own profile (username/email/password)
PATCH	/api/v1/users/{user_id}/deactivate	Yes (Admin)	Deactivate a user account
DELETE	/api/v1/users/{user_id}	Yes (Admin)	Hard delete a user account

7.4 Project Endpoints

Method	Path	Auth Required	Description
GET	/api/v1/projects	Yes	List accessible projects (paginated, filterable)
POST	/api/v1/projects	Yes	Create a new project
GET	/api/v1/projects/{project_id}	Yes	Get project details
PATCH	/api/v1/projects/{project_id}	Yes (Owner/Admin)	Update project name or description
DELETE	/api/v1/projects/{project_id}	Yes (Owner/Admin)	Hard delete project and tasks
PATCH	/api/v1/projects/{project_id}/archive	Yes (Owner/Admin)	Toggle project archived state
GET	/api/v1/projects/{project_id}/members	Yes (Member/Admin)	List project members
POST	/api/v1/projects/{project_id}/members	Yes (Owner/Admin)	Add a user to project

Method	Path	Auth Required	Description
DELETE	/api/v1/projects/{project_id}/members/{user_id}	Yes (Owner/Admin)	Remove a user from project

7.5 Task Endpoints

Method	Path	Auth Required	Description
GET	/api/v1/projects/{project_id}/tasks	Yes (Member/Admin)	List tasks in project (filtered/sorted/paginated)
POST	/api/v1/projects/{project_id}/tasks	Yes (Member/Admin)	Create a task in the project
GET	/api/v1/projects/{project_id}/tasks/{task_id}	Yes (Member/Admin)	Get task detail with assignees
PATCH	/api/v1/projects/{project_id}/tasks/{task_id}	Yes (Owner/Creator/Admin)	Update task fields
DELETE	/api/v1/projects/{project_id}/tasks/{task_id}	Yes (Owner/Creator/Admin)	Delete a task
GET	/api/v1/tasks/mine	Yes	List all tasks assigned to current user

7.6 Assignment Endpoints

Method	Path	Auth Required	Description
GET	/api/v1/projects/{project_id}/tasks/{task_id}/assignments	Yes (Member/Admin)	List assignments for a task
POST	/api/v1/projects/{project_id}/tasks/{task_id}/assignments	Yes (Owner/Admin)	Assign a user to a task
DELETE	/api/v1/projects/{project_id}/tasks/{task_id}/assignments/{user_id}	Yes (Owner/Admin)	Remove a user assignment

8. Filtering, Pagination, Sorting & Search

8.1 Pagination

All list endpoints implement limit/offset pagination via query parameters.

Parameter	Type	Default	Max	Description
limit	integer	20	100	Number of records to return
offset	integer	0	—	Number of records to skip

All paginated responses include a standard envelope:

Field	Type	Description
total	integer	Total matching records (ignoring pagination)
limit	integer	Applied limit
offset	integer	Applied offset
items	array	The result items for this page

8.2 Filtering — Tasks

Query Param	Type	Example	Description
status	string (enum)	?status=in_progress	Filter by task status
priority	string (enum)	?priority=high	Filter by task priority
assignee_id	integer/uuid	?assignee_id=5	Filter tasks by assignee
due_date_from	date (ISO 8601)	?due_date_from=2025-01-01	Tasks with due_date >= value
due_date_to	date (ISO 8601)	?due_date_to=2025-01-31	Tasks with due_date <= value
is_overdue	boolean	?is_overdue=true	Tasks past due date and not done
created_by	integer/uuid	?created_by=3	Filter tasks by creator

8.3 Filtering — Projects

Query Param	Type	Example	Description
is_archived	boolean	?is_archived=false	Include/exclude archived projects (default: false)
search	string	?search=mobile	Search by project name

8.4 Sorting

Tasks support sorting via the sort_by and sort_dir query parameters.

Parameter	Allowed Values	Default	Description
sort_by	created_at, due_date, priority, updated_at	created_at	Field to sort by
sort_dir	asc, desc	desc	Sort direction

Priority sorting uses semantic order: high > medium > low (not alphabetical).

8.5 Search

- Tasks support full-text search via the q query parameter: ?q=login bug
- Search is performed on title and description fields using ILIKE pattern matching (case-insensitive substring match).
- When q is provided, results are ordered by relevance (title match weighted above description match) before the sort_by order is applied.
- Search can be combined with any filter and pagination parameter.

9. Validation Rules

9.1 User Registration / Update

Field	Rules
username	Required. 3–50 characters. Alphanumeric + underscores only. Must be unique (case-insensitive).
email	Required. Valid email format. Max 255 characters. Must be unique (case-insensitive).
password	Required on register. Min 8 characters. Must contain at least one uppercase letter, one lowercase letter, and one digit. Max 128 characters.
new_password	Optional on update. Same rules as password. Cannot be the same as the current password.

9.2 Project

Field	Rules
name	Required. 1–100 characters. Must not be blank/whitespace only.
description	Optional. Max 2000 characters.

9.3 Task

Field	Rules
title	Required. 1–200 characters. Must not be blank/whitespace only.
description	Optional. Max 5000 characters.
status	Required. Must be one of: todo, in_progress, done.
priority	Required. Must be one of: low, medium, high.
due_date	Optional. ISO 8601 date (YYYY-MM-DD). Must not be in the past on creation.

9.4 Pagination & Query Params

Parameter	Rules
limit	Integer. Min 1, Max 100. Invalid type returns 422.
offset	Integer. Min 0. Invalid type returns 422.
sort_dir	Must be 'asc' or 'desc'. Invalid value returns 422.
sort_by	Must be one of the allowed sortable fields. Invalid value returns 422.
due_date_from / to	Must be valid ISO 8601 date. due_date_from must not be after due_date_to.

9.5 Validation Error Response

Validation errors return HTTP 422 Unprocessable Entity. The response body follows this structure:

Field	Type	Description
detail	array	List of validation error objects
detail[].loc	array	Location path (e.g., ['body', 'email'])
detail[].msg	string	Human-readable error message
detail[].type	string	Pydantic error type code

10. Error Handling Strategy

10.1 Error Response Format

All error responses return a consistent JSON body regardless of the error type:

Field	Type	Description
error	object	Container for error details
error.code	string	Machine-readable error code (e.g., 'resource_not_found')
error.message	string	Human-readable description
error.details	object | null	Additional context (e.g., field names for validation errors)

10.2 HTTP Status Code Strategy

Status Code	When to Use
200 OK	Successful GET, PATCH, DELETE (returning resource)
201 Created	Successful POST that creates a resource (return resource in body)
204 No Content	Successful DELETE that returns no body
400 Bad Request	Malformed request body (non-validation, e.g., invalid JSON)
401 Unauthorized	Missing, invalid, or expired authentication token
403 Forbidden	Authenticated but insufficient permissions for this resource
404 Not Found	Resource does not exist or is inaccessible to the user
409 Conflict	Duplicate resource (e.g., duplicate assignment, duplicate email)
422 Unprocessable Entity	Input fails validation rules (Pydantic errors)
500 Internal Server Error	Unexpected server-side error (never expose stack traces in production)

10.3 Error Codes Reference

Error Code	HTTP Status	Description
invalid_credentials	401	Wrong email or password on login
token_expired	401	JWT access token has expired
invalid_token	401	JWT signature invalid or malformed
token_required	401	Authorization header missing
account_disabled	403	User account is deactivated
permission_denied	403	Action not allowed for user's role/membership
resource_not_found	404	Requested resource does not exist
not_a_member	403	User is not a member of the project

Error Code	HTTP Status	Description
duplicate_assignment	409	User is already assigned to the task
duplicate_email	409	Email already registered
duplicate_username	409	Username already taken
validation_error	422	Request body fails schema validation
internal_error	500	Unexpected server error

10.4 Global Exception Handling

- All unhandled exceptions are caught by a global FastAPI exception handler.
- Stack traces are logged server-side but never returned in API responses.
- SQLAlchemy IntegrityError is caught and mapped to 409 Conflict with appropriate error code.
- SQLAlchemy NoResultFound is caught and mapped to 404 Not Found.
- FastAPI RequestValidationError is caught and reformatted into the standard error envelope.

11. Non-Functional Requirements

11.1 Performance

- NFR-PERF-01: All API endpoints (excluding complex search) shall respond within 300ms under normal load.
- NFR-PERF-02: Pagination must be implemented using LIMIT/OFFSET at the database layer, not in-memory slicing.
- NFR-PERF-03: Frequently filtered columns (status, priority, project_id, user_id) shall have database indexes.
- NFR-PERF-04: N+1 query problems shall be eliminated via SQLAlchemy eager loading (joinedload/selectinload) where appropriate.

11.2 Security

- NFR-SEC-01: JWT secret keys shall be sourced exclusively from environment variables, never hardcoded.
- NFR-SEC-02: Password fields shall never appear in any API response payload.
- NFR-SEC-03: Error messages on login shall not reveal whether the email or password was incorrect.
- NFR-SEC-04: All database inputs shall be parameterized via the ORM. Raw SQL string interpolation is prohibited.

- NFR-SEC-05: CORS origins shall be configurable via environment variables.

11.3 Reliability

- NFR-REL-01: All database schema changes shall be managed exclusively through Alembic migration scripts.
- NFR-REL-02: The application shall fail fast with a clear error message if required environment variables are missing at startup.
- NFR-REL-03: Database connections shall use a connection pool (SQLAlchemy default pool settings).

11.4 Maintainability

- NFR-MAINT-01: The codebase shall follow the three-layer clean architecture: Router (HTTP) → Service (Business Logic) → Repository (Data Access).
- NFR-MAINT-02: No database queries shall exist in Router or Service layers.
- NFR-MAINT-03: No business logic shall exist in Repository layers.
- NFR-MAINT-04: All API routes shall be documented via FastAPI's auto-generated OpenAPI/Swagger UI at /docs.
- NFR-MAINT-05: Environment configuration shall be managed via a Pydantic Settings class.

11.5 API Standards

- NFR-API-01: All endpoints shall use JSON request/response bodies with Content-Type: application/json.
- NFR-API-02: All timestamps in responses shall be formatted as ISO 8601 UTC strings.
- NFR-API-03: Resource IDs shall be consistent in type (all UUID or all integer) across the entire API.
- NFR-API-04: HTTP methods shall be used semantically: GET for reads, POST for creation, PATCH for partial update, DELETE for removal.

12. Folder Structure — Clean Architecture

Path	Description
trello_lite/	Project root
├── app/	Application package
│ ├── main.py	FastAPI app factory, middleware, router registration
│ ├── config.py	Pydantic Settings — all env var configuration

Path	Description
│ ├── database.py	SQLAlchemy engine, session factory, Base
│ ├── dependencies.py	Shared FastAPI dependencies (get_db, get_current_user, require_admin)
│ ├── exceptions.py	Custom exception classes and global exception handlers
│ ├── models/	SQLAlchemy ORM model definitions
│ │ ├── user.py	User model
│ │ ├── project.py	Project model
│ │ ├── task.py	Task model
│ │ ├── assignment.py	Assignment model
│ │ └── project_member.py	ProjectMember association model
│ ├── schemas/	Pydantic request/response schemas
│ │ ├── auth.py	Login, token, register schemas
│ │ ├── user.py	User request/response schemas
│ │ ├── project.py	Project request/response schemas
│ │ ├── task.py	Task request/response schemas
│ │ ├── assignment.py	Assignment request/response schemas
│ │ └── common.py	Shared schemas (PaginatedResponse, ErrorResponse)
│ ├── routers/	FastAPI router definitions (HTTP layer only)
│ │ ├── auth.py	Auth routes
│ │ ├── users.py	User routes
│ │ ├── projects.py	Project routes
│ │ ├── tasks.py	Task routes
│ │ └── assignments.py	Assignment routes
│ ├── services/	Business logic layer
│ │ ├── auth_service.py	Token generation, password verification, refresh logic
│ │ ├── user_service.py	User business logic
│ │ ├── project_service.py	Project business logic, membership logic
│ │ ├── task_service.py	Task business logic, filtering/sorting/search
│ │ └── assignment_service.py	Assignment business logic
│ └── repositories/	Data access layer (SQLAlchemy queries only)
│ ├── user_repository.py	User CRUD queries

Path	Description
│ ├── project_repository.py	Project CRUD + membership queries
│ ├── task_repository.py	Task CRUD + filtered list queries
│ └── assignment_repository.py	Assignment CRUD queries
├── migrations/	Alembic migration directory
│ ├── env.py	Alembic environment config
│ ├── script.py.mako	Migration template
│ └── versions/	Generated migration scripts
├── tests/	Test suite
│ ├── conftest.py	pytest fixtures (test DB, test client, auth headers)
│ ├── unit/	Unit tests (service and repository functions in isolation)
│ │ ├── test_auth_service.py	
│ │ ├── test_user_service.py	
│ │ ├── test_project_service.py	
│ │ └── test_task_service.py	
│ └── integration/	Integration tests (HTTP-level via TestClient)
│ ├── test_auth.py	
│ ├── test_users.py	
│ ├── test_projects.py	
│ ├── test_tasks.py	
│ └── test_assignments.py	
.env	Local environment variables (never committed to VCS)
.env.example	Template with required variable names (committed to VCS)
alembic.ini	Alembic configuration file
requirements.txt	Python dependencies
pytest.ini	pytest configuration

13. Testing Scope

13.1 Testing Strategy

The project uses a two-tier testing approach: unit tests for service-layer logic and integration tests for full HTTP request/response flows.

13.2 Test Infrastructure

- Test database: Separate PostgreSQL database (trello_lite_test) or SQLite in-memory for speed.
- Test client: FastAPI TestClient (synchronous) or httpx.AsyncClient for async routes.
- Fixtures: conftest.py provides reusable fixtures for DB session, authenticated user, admin user, test projects, test tasks.
- Isolation: Each test function runs in a rolled-back transaction or truncated tables to ensure independence.

13.3 Unit Test Coverage

Module	What to Test
auth_service	Password hashing, password verification, JWT creation, JWT decoding, expired token detection
user_service	User creation validation, duplicate detection, profile update logic, password change validation
project_service	Project creation, ownership check, member add/remove, archive toggle, permission checks
task_service	Task creation, status transition rules, filter building, sort direction, search query construction
assignment_service	Member-only assignment validation, duplicate detection, unassignment logic

13.4 Integration Test Coverage

Endpoint Group	Scenarios to Cover
POST /auth/register	Success, duplicate email, duplicate username, weak password, missing fields
POST /auth/login	Success, wrong password, unknown email, inactive account
GET /auth/me	Success with valid token, 401 with no token, 401 with expired token
POST /auth/refresh	Success, invalid refresh token
Project CRUD	Create, read, update, delete; verify 403 for non-owners; verify 404 for missing projects
Project members	Add member, remove member, 409 on duplicate, 404 on non-existent user

Endpoint Group	Scenarios to Cover
Task CRUD	Create, read, update, delete; verify non-members get 403; verify task creator permissions
Task list	Pagination (limit/offset), filter by status, filter by priority, search by keyword, sort by due_date
Assignments	Assign user, 409 on duplicate, unassign, verify assignee must be project member
Admin endpoints	List all users (admin only), deactivate user, verify regular user gets 403

13.5 Test Naming Convention

All test functions follow the pattern: test_<action>_<scenario>. Examples:

- test_login_with_valid_credentials_returns_tokens
- test_login_with_wrong_password_returns_401
- test_create_task_as_non_member_returns_403
- test_list_tasks_filtered_by_status_returns_correct_items
- test_assign_user_duplicate_returns_409

13.6 Coverage Target

Minimum target: 80% line coverage across service and repository layers. Coverage is measured via pytest-cov and reported on each test run.

14. Environment Configuration

Variable	Required	Default	Description
DATABASE_URL	Yes	—	PostgreSQL connection string
SECRET_KEY	Yes	—	JWT signing secret (min 32 chars)
ACCESS_TOKEN_EXPIRE_MINUTES	No	30	Access token TTL in minutes
REFRESH_TOKEN_EXPIRE_DAYS	No	7	Refresh token TTL in days
BCRYPT_ROUNDS	No	12	bcrypt cost factor
CORS_ORIGINS	No	*	Comma-separated allowed origins
APP_ENV	No	development	Environment: development, production

Variable	Required	Default	Description
LOG_LEVEL	No	INFO	Logging verbosity

15. Appendix — Glossary of HTTP Methods Used

Method	Semantics in This API
GET	Read-only retrieval. No side effects. Idempotent.
POST	Create a new resource or trigger an action (login, logout, assign).
PATCH	Partial update of an existing resource. Only provided fields are changed.
DELETE	Remove a resource permanently (hard delete) or remove a relationship (unassign).

— End of Document —