

Team data

Team number: 10

GITHUB: [SE1-Research-Showcase-Portal](#)

| Student number | First name | Last name |
|----------------|------------|-------------------|
| 0231297359 | Yaraslaw | Akhramenka |
| 0231364350 | Adelaide | Danilov |
| 0220502202 | Demyan | Faguer |
| 0230901309 | Oleksandr | Marchenko Breneur |
| 023159503B | Oleksandr | Yeroftieiev |

Introduction

Vision of the project

The Research Showcase Portal is a web application designed to make research sharing and discovery easy, encourage discussion, and provide a semiformal environment where a broad audience can publish and engage with research outputs. The portal supports different roles (aka stakeholders): anonymous user, registered user, researcher, moderator, all of which are granted full access to view the content uploaded to the portal.

Tech stack

- **Frontend:** Next.js (App Router), TypeScript, Tailwind CSS.
- **Backend:** FastAPI (Python), REST API.
- **Database:** PostgreSQL with SQLAlchemy ORM models.
- **Uploads:** attachment upload endpoints and static serving of uploaded files.

Motivation

- **PostgreSQL:** PostgreSQL is a proven, widely supported database implementation that is ACID-compliant and comes with a complete ecosystem of supporting tools.
- **Relational DB:** The core entities of the portal (users, publications, projects, tags, reviews, permissions, etc.) are naturally represented as structured records with well-defined relationships. A relational model supports this structure directly and enables consistent integrity constraints and expressive querying.
- **SQLAlchemy:** SQLAlchemy provides an ORM that enables interacting with the database in an object-oriented style, supports dynamic updates, and allows using models both as database table definitions and as application data structures.
- **FastAPI:** FastAPI offers flexible development of GET/POST endpoints via decorators and integrates tightly with Pydantic for model validation.
- **REST over WebSockets:** Generally, real-time communications not required for the portal, instead a simple polling every couple of seconds is sufficient to update the page content.

Requirements

In general, we have closely followed the functional and non-functional requirements listed in the first deliverable.

| Requirement | Details | Status |
|-----------------------------|--|-------------|
| Account | Backend: <code>/users/*</code> endpoints; Frontend: <code>/register</code> , <code>/login</code> , <code>/reset-password</code> , <code>/verify-email</code> | Implemented |
| Profiles | Backend: <code>GET /users/{username}</code> , <code>PATCH /users/me</code> ; Frontend: <code>/[username]</code> , <code>/me/profile</code> | Implemented |
| Institutional Verification | Decided to drop this feature and instead give the verified plaque to every researcher upon promotion | Iterated |
| Create Research Post | <code>POST /posts/create</code> ; <code>POST /posts/attachments/upload</code> ; static <code>/attachments</code> | Implemented |
| Becoming a Researcher | Promotion logic in review flow (≥ 3 positive reviews) | Implemented |
| Comments & threaded replies | <code>POST/GET /posts/{post_id}/comments</code> ; thread via <code>parent_comment_id</code> | Implemented |
| Voting | <code>POST /posts/{post_id}/vote</code> , <code>/comments/{comment_id}/vote</code> , <code>/reviews/{review_id}/vote</code> | Implemented |
| Tagging & discovery | Tags stored and used for filtering in UI; dedicated tag browsing API | Implemented |
| Search | <code>GET /posts/</code> (query parameters) and search functionality on the main page | Implemented |
| Share/Cite | Post stores BibTeX; frontend supports share/citation copy | Implemented |
| Peer-reviews | <code>POST/GET /posts/{post_id}/reviews</code> and review UI | Implemented |
| Reporting & moderation | <code>POST/GET /posts/{post_id}/reports</code> ; moderator status updates | Implemented |

Table 1: Functional requirements' implementation status table

For the non-functional requirements, some of the notions, such as "clean UI" can't be assessed on the implemented/not implemented scale.

For the maintainability requirement (aim $\geq 80\%$ test coverage), due to the time constraints, most of the testing was done manually through FastAPI's `/docs` instead of automated methods.

For the security requirements, the rate limiting is not yet implemented either.

Project structure

The project is organized as a single repo with three main components (backend, frontend, database). This structure supports readability, testing, and future extension as the feature set grows.

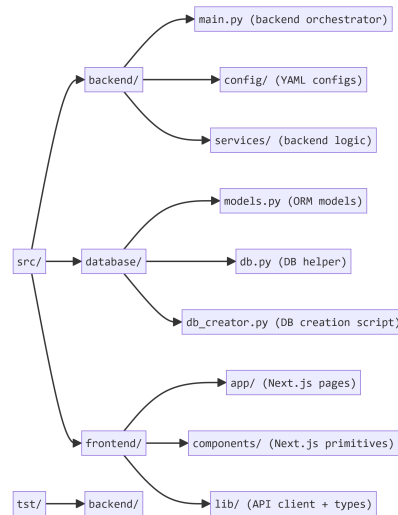


Figure 1: Project structure

Database and Domain Model

The database layer uses SQLAlchemy to define the ORM model entities and map them to the tables inside PostgreSQL DB. Additionally, we use psycopg2 python interface to interact with the SQL relations in the object-oriented manner.

The final DB schema (Fig 2) is closely related to the in initially envisioned domain model. The core entities of the DB are:

- **User & Profile:** authentication data, roles (user/researcher/moderator), and user profile fields.
- **Post:** research posts (title/body/metadata), optional BibTeX, timestamps.
- **Tag:** many-to-many tagging for posts to support discovery and filtering.
- **Attachment:** uploaded files associated with posts.
- **Comment:** comments and threaded replies via `parent_comment_id`.
- **Review:** structured researcher reviews on posts (with separate voting).
- **Report:** reports for Terms & Conditions violations, with moderation status lifecycle.
- **Votes:** vote tables for posts/comments/reviews to enforce one vote per user per target.



Figure 2: DB schema

Subsystem Interaction

- Authentication sits in front of everything: it protects access to Posts, Comments, Reviews, Votes, and Reports.
- Posts are the central content: a post has Comments and has Reviews.
- Voting is a shared service: it applies to Posts, Comments, and Reviews.
- Reports are filed against content: a report targets either a Post or a Comment (then handled/decided within the report workflow).

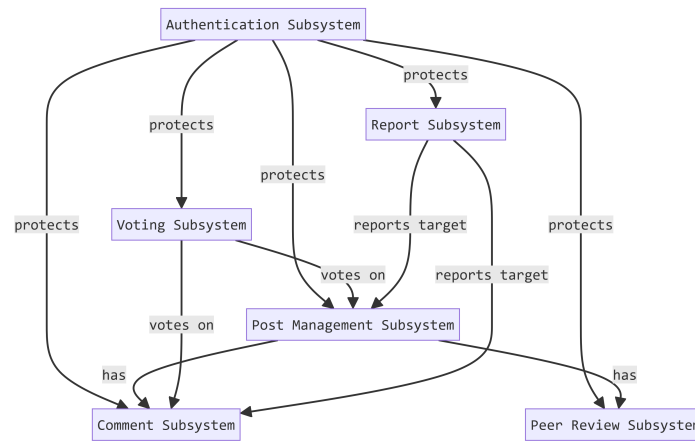


Figure 3: Subsystem interaction diagram

Use Cases Supported End-to-end

For the UML diagrams with specifics of the implementation of core UCs, please refer to the Appendix.

- **UC-01 Search:** anonymous or logged-in users search and filter posts by keywords/tags.
- **UC-02/UC-03 Register/Login:** users create accounts, verify email, and authenticate.
- **UC-04/UC-05 Account/Profile settings:** users modify their profile and account data such as passwords.
- **UC-07 Publish:** registered users publish posts with tags, BibTeX, and attachments.
- **UC-08/UC-09 Comment/Vote:** users comment (threaded) and vote on content.
- **UC-10 Sharing:** users can share a post by copying BibTeX or a plain URL.
- **UC-11 Review:** researchers add formal reviews to posts and vote on reviews.
- **UC-12/UC-13 Report/Moderate:** users report violations; moderators manage report status.

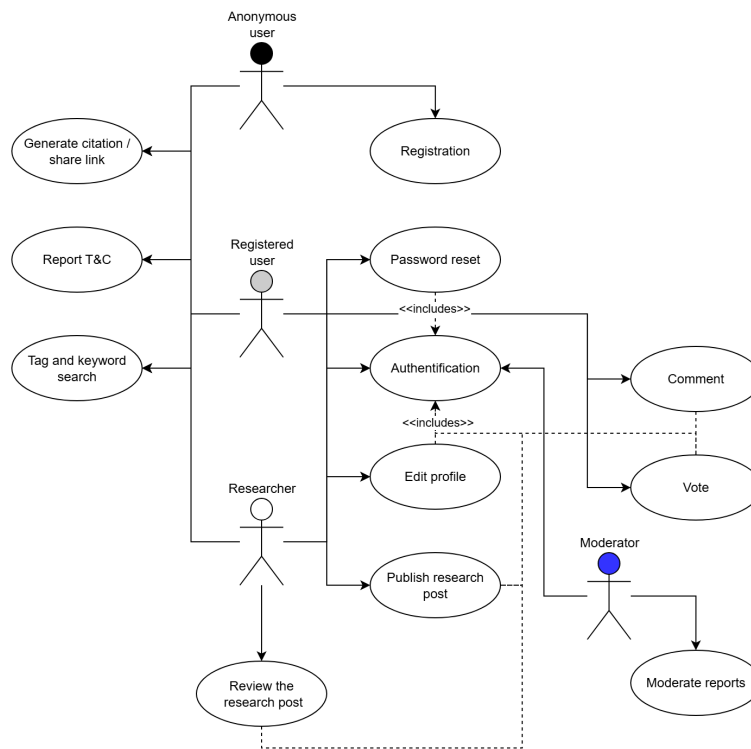


Figure 4: Use-case model updated according to the delivered features.

Implementation path

This section summarizes the implementation path in four sprints and provides direct [GitHub link](#) to follow the evolution of the codebase.

Sprint 1: Architecture & environment setup

Work started by establishing the repository layout, preparing local development, and introducing the initial database and backend scaffolding. The goal of this sprint was to make the project runnable end-to-end (database → backend → frontend) with a reproducible setup.

- Finalized the choice of technological stack for the project
- Defined repository structure under `src/` (backend, database, frontend).
- Added environment/setup documentation and configuration files.
- Introduced database schema and its implementation under SQLAlchemy ORM subset.
- Wired the FastAPI application to support frontend requests (e.g., CORS and basic routing).

Sprint 2: Core auth flow: Registration, login, verification

After the baseline was set up, we started to implement the core auth flow: authentication and access control as a gateway for publishing and interacting with content. Use cases UC-02, UC-03, UC-06 were used to validate the correctness of the authentication flow.

- Implemented secure user registration and login endpoints and connected them to the frontend authentication pages.
- Added email verification logic to activate accounts before allowing full access.
- Enforced role-based access control for restricted actions after login.

- Implemented password security using Argon2 hashing and JWT-based authentication tokens.
- Validated the flow based on UC-02, UC-03, UC-06 by checking successful login, restricted access before authentication, and ability to post after authentication.

Sprint 3: Posting, search, user interactions and UI

The portal received its main features such as paper posting, search capabilities and user interactions, so that users can publish their work, find relevant ones and engage with it. Along with the progress on the UI side, UC-01, UC-07, UC-08, UC-09 were delivered

- Implemented post listing, as well as search and filtering functionality.
- Added commenting and threaded replies using `parent_comment_id`.
- Implemented user voting on posts and comments, enforcing one vote per user per target.
- A qualitative transition of the front-end from mock-up test points to the formed user interface

Sprint 4: Moderation, reports, QoL and polishing

Finally, moderation and QoL features were added to bring the user experience closer to the intended product vision. Respective delivered Use Cases: UC-05, UC-10, UC-11, UC-12, UC-13

- Added reporting flows for Terms & Conditions violations and moderator handling of reports.
- Implemented an ability to change the user status according to moderation and platform rules.
- Added generating citations with share links and BibTeX display and copy.
- Provided the ability to update profile information.
- Gave researchers a possibility to leave a structured feedback by means of reviews.
- Added promotion to researcher after at least three positive reviews.

Future work

Potential extensions based on the initial scope and observed gaps:

- **Institutional verification workflow:** implement university email verification and a verified badge for confirmed institutional accounts.
- **Testing and CI:** add automated unit and integration tests and a CI/CD pipeline with coverage targets and quality gates.
- **Content quality and moderation:** improve review templates, add spam and abuse prevention measures, provide richer moderation tools and analytics.
- **Security improvement:** iterate input validation and rate limiting, restrict file upload handling, enforce limits on repeated server queries - for example, restricting report spamming.
- **Scraping policy:** Develop a stance regarding search engines permissions and the extent of the allowable scraping of the portal content.

Appendix

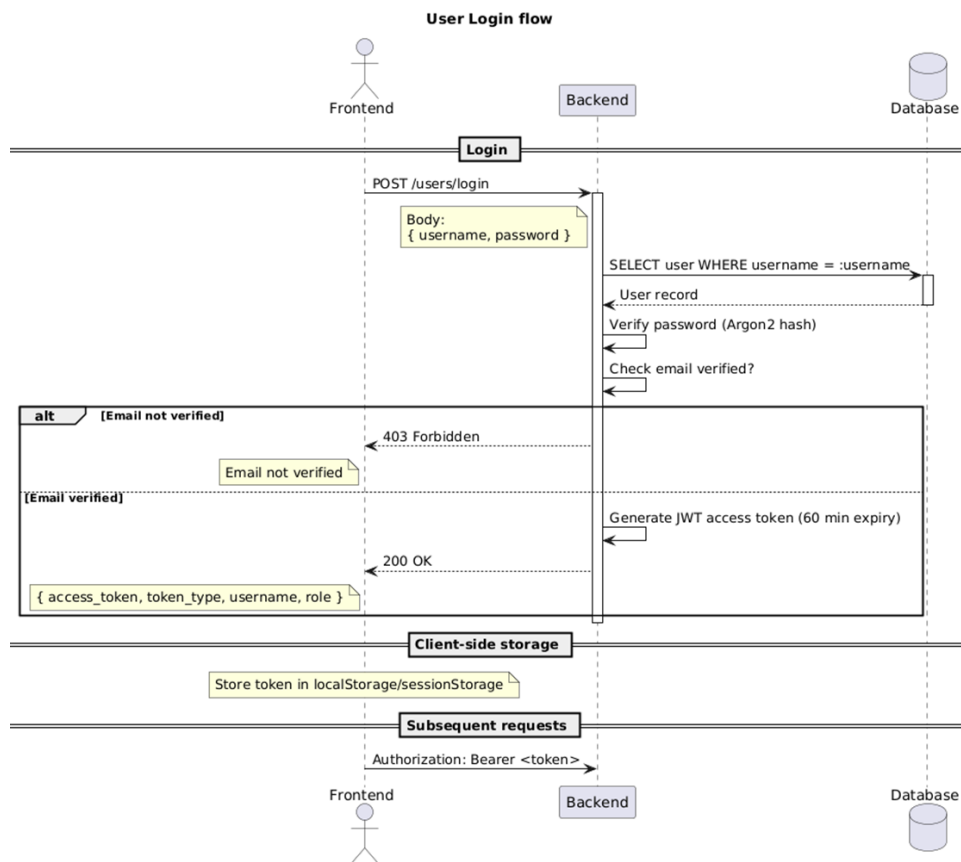


Figure 5: User Login

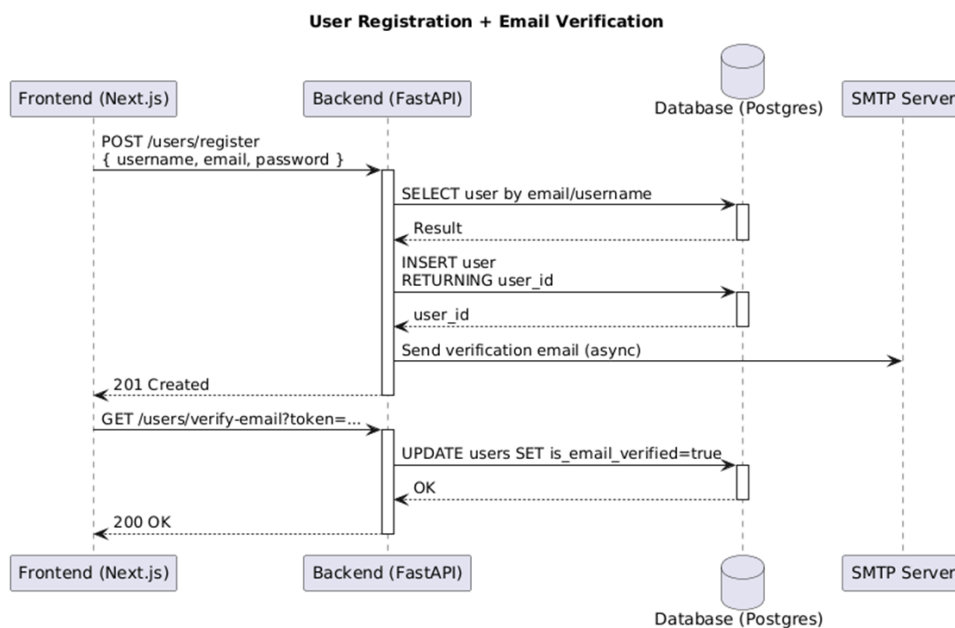


Figure 6: User Registration and Email Verification

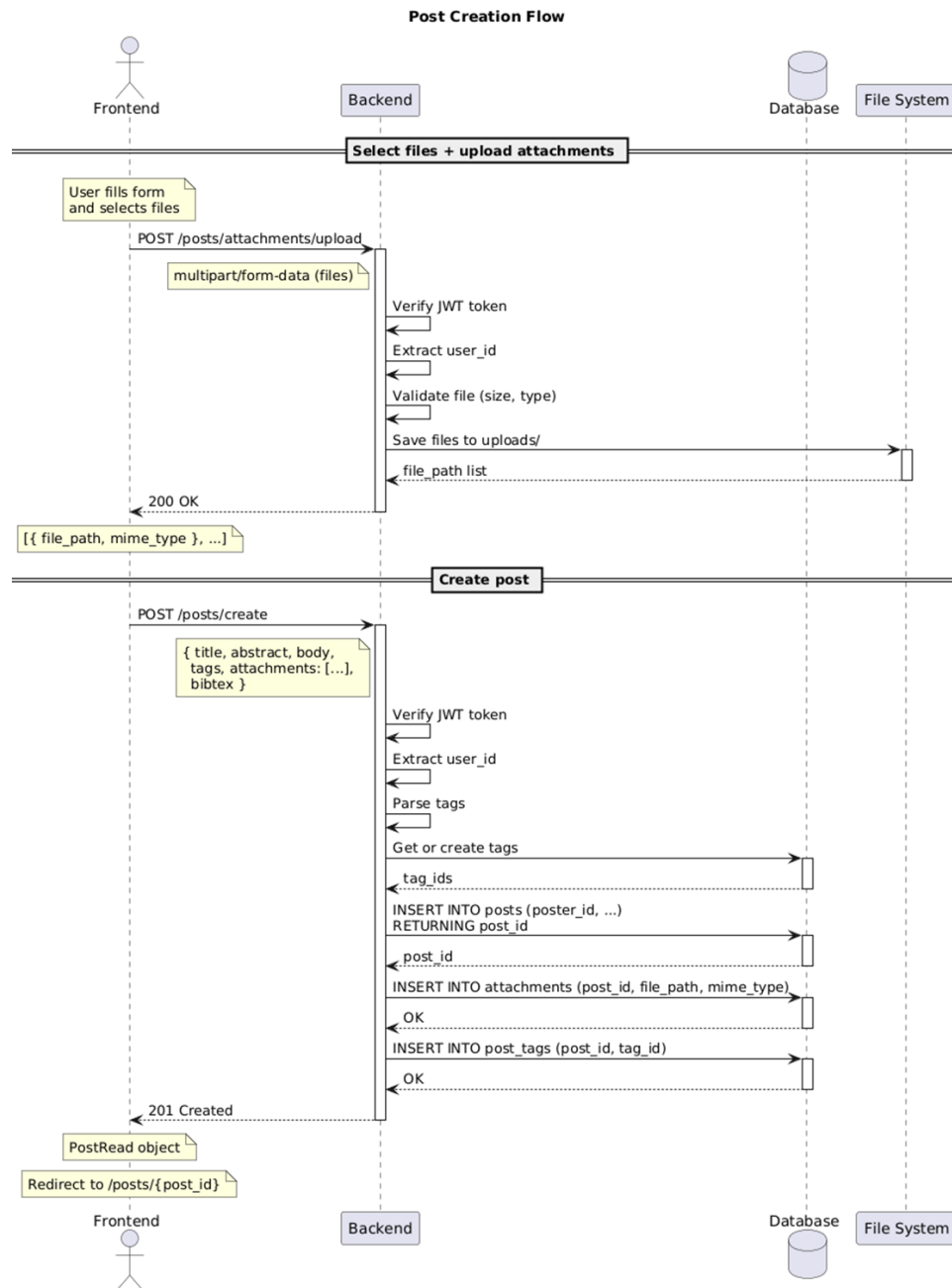


Figure 7: Post Creation

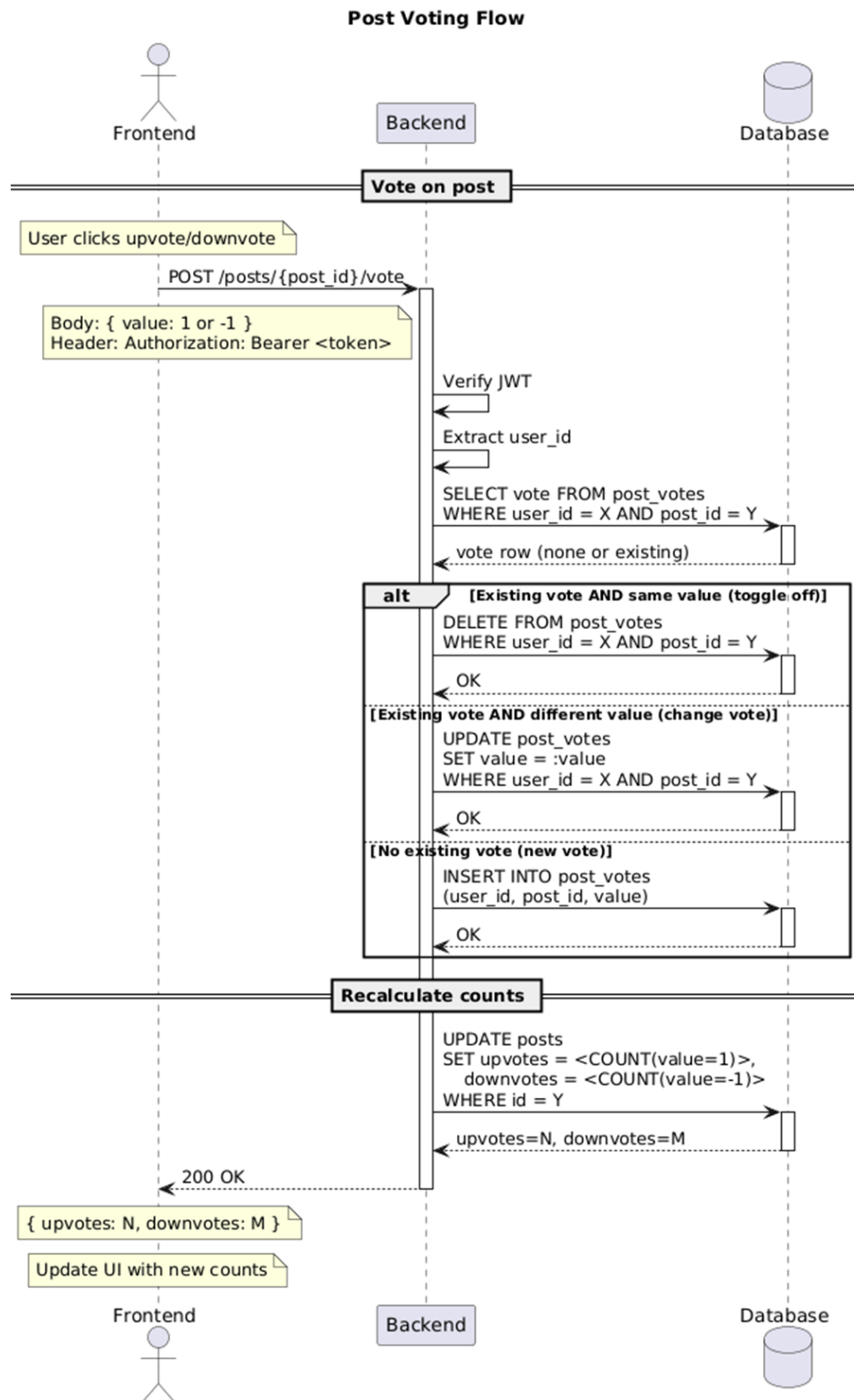


Figure 8: Post Voting

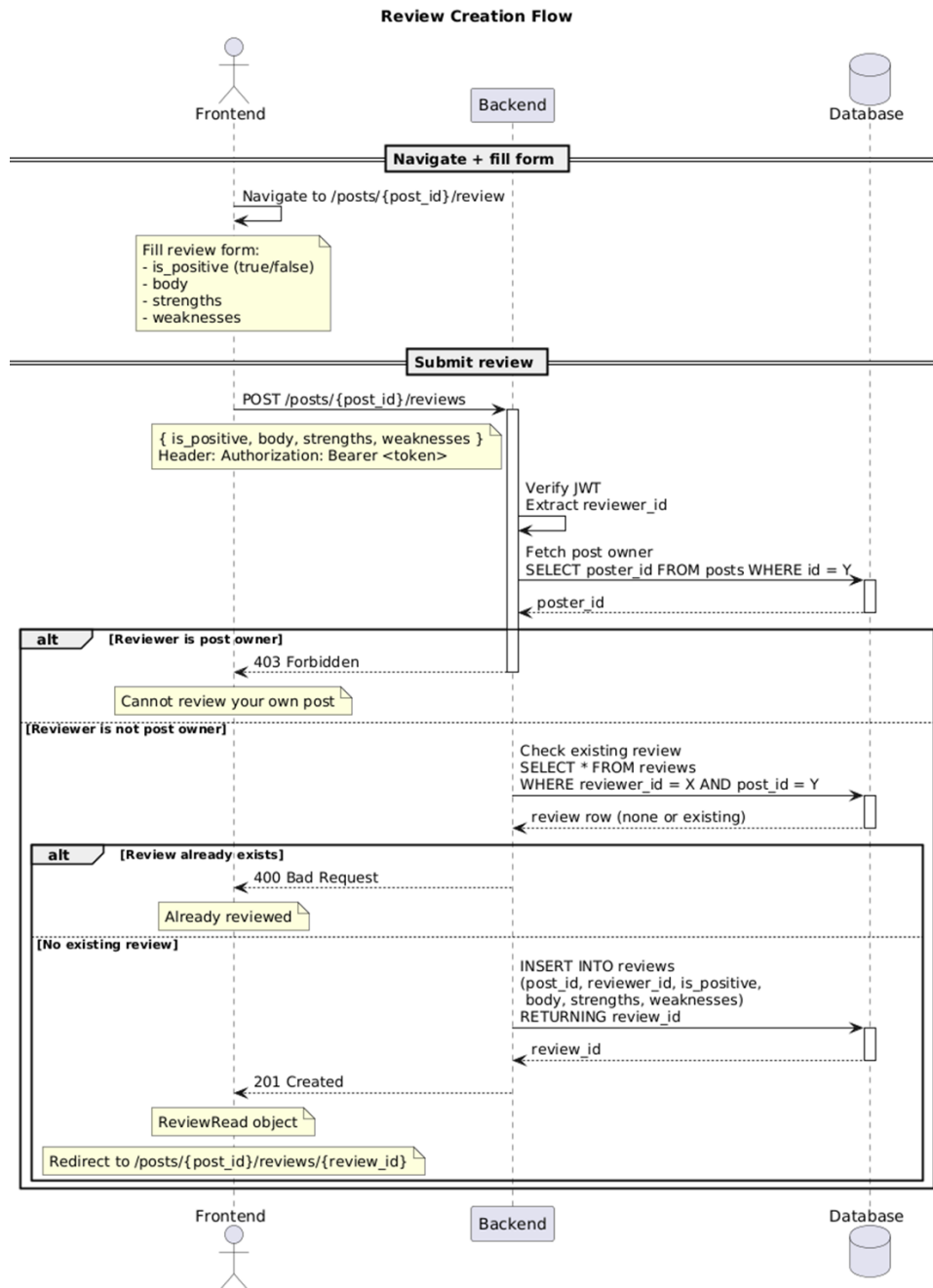


Figure 9: Review Creation