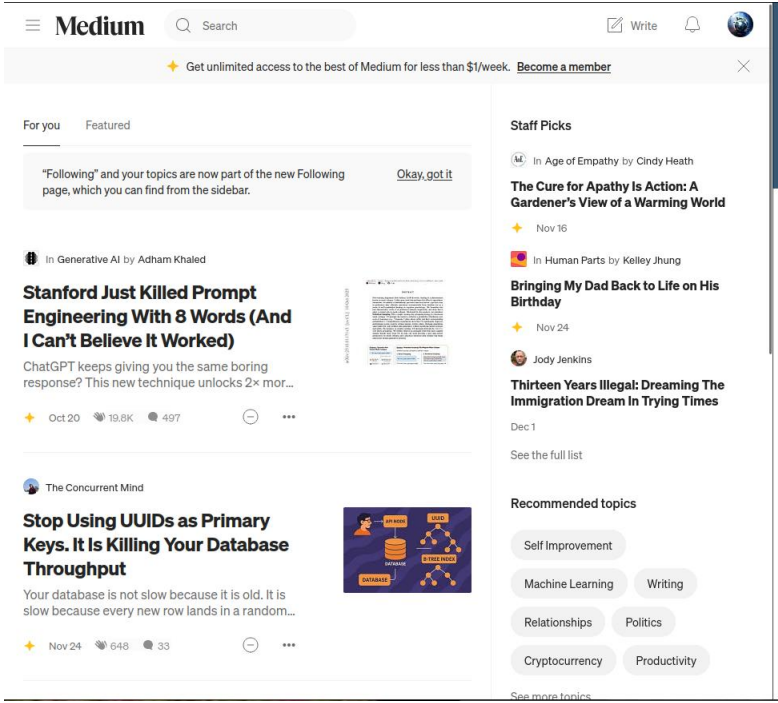
Research Showcase Portal

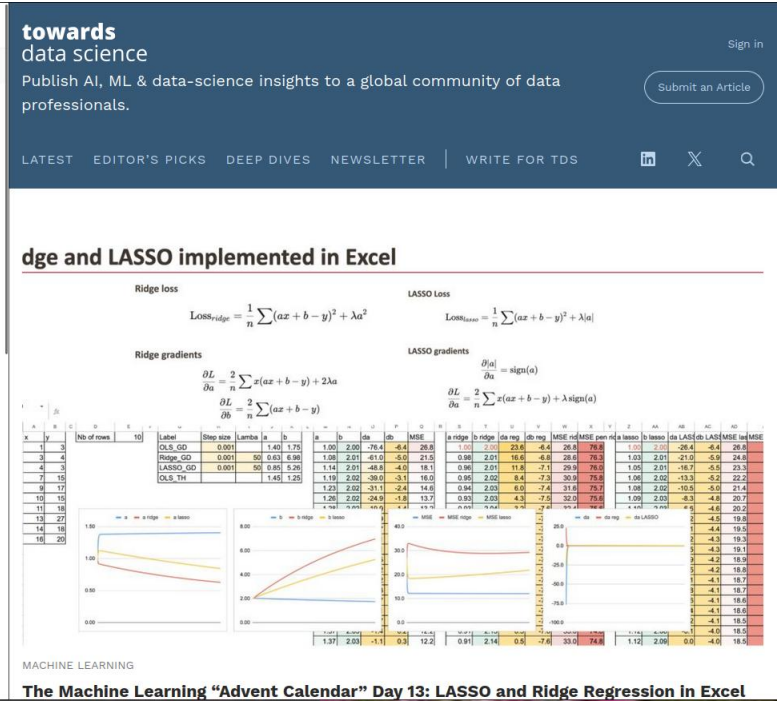
Software Engineering 1
University of Luxembourg
Group 10

Our Vision

- Simple, functional website
- Easy research sharing and discovery
- Endorse discussions
- Voice for everyone
- Semi-formal “vibe”

Inspirations





Stakeholder Roles

- **Anonymous User:** search, read, share/cite published research
- **Registered User:** all of the above + comment, publish research, vote, report T&C violations
- **Researcher:** all of the above + review other users' research, have a special plaque when posting
- **Moderator:** handles T&C reports; can approve initial researchers (cold-start problem)

Requirements

Functional	Non-functional
Accounts: register, login, reset password	**Performance:** search ≤ 1s, page render ≤ 2s
Profiles: editable profile and post history; optional social links	**Scalability:** API supports horizontal scaling
Create Research Post: allow text and attachment files	**Reliability:** durable storage for posts/comments
Comments: comments and replies for posts	**Security:** secure credentials storage, rate limits
Voting: up/down votes on posts and comments	**Usability:** intuitive UI and clear error messages
Reviews: researchers review published posts (3 positive reviews to become a researcher)	**Maintainability:** clean module boundaries, automated and manual testing, target 80% line coverage
Moderation: users report T&C violations; moderators delete/allow	

Use Cases (Core)

- **UC-01** Search + Filtering
- **UC-02** Registration + Authentication
- **UC-03** Login
- **UC-07** Publish Research Post
- **UC-08** Comment
- **UC-09** Vote
- **UC-11** Review Research Post
- **UC-12/13** T&C + Moderation

DEMO TIME!

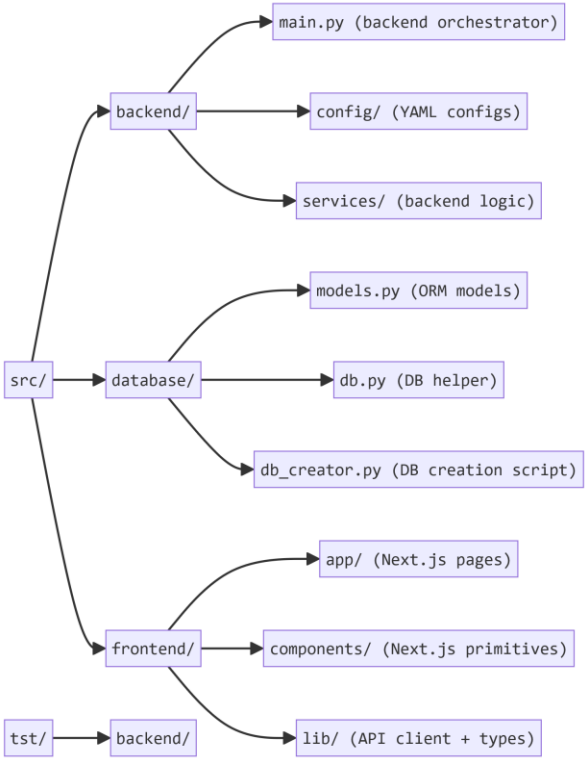
Tech Stack

Backend

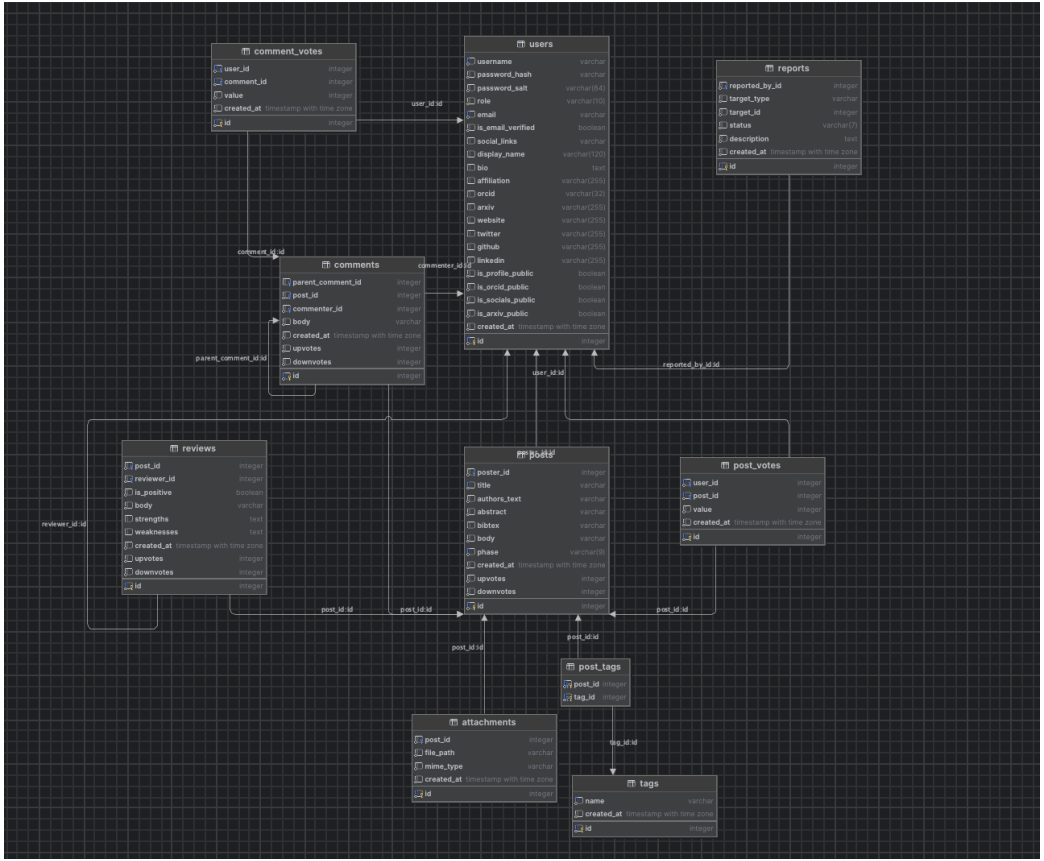
Frontend

Database

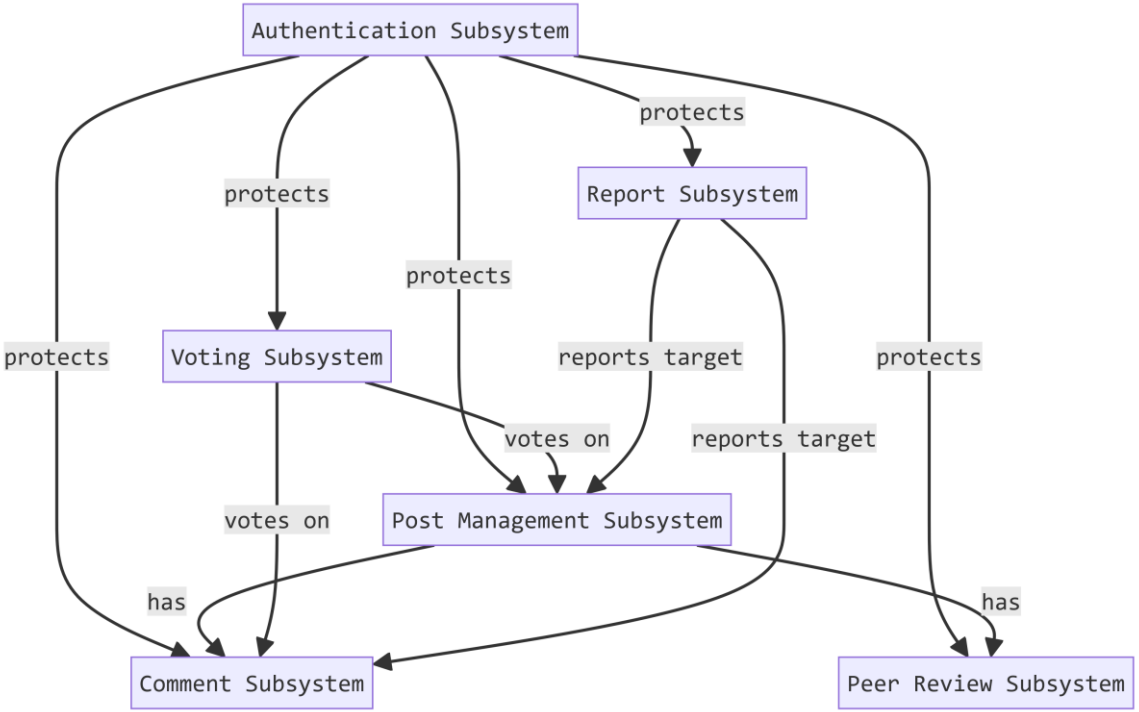
Project Structure

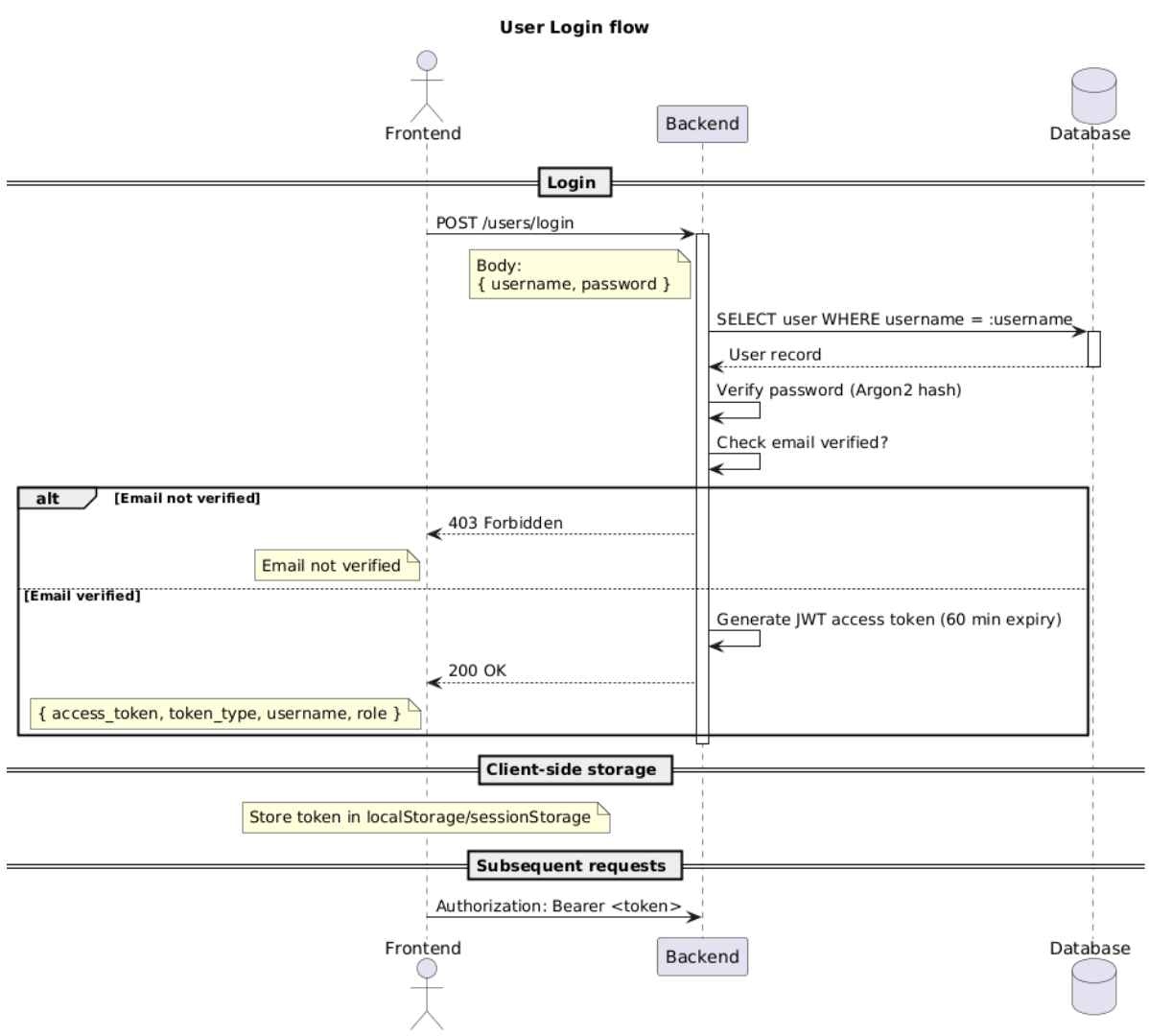


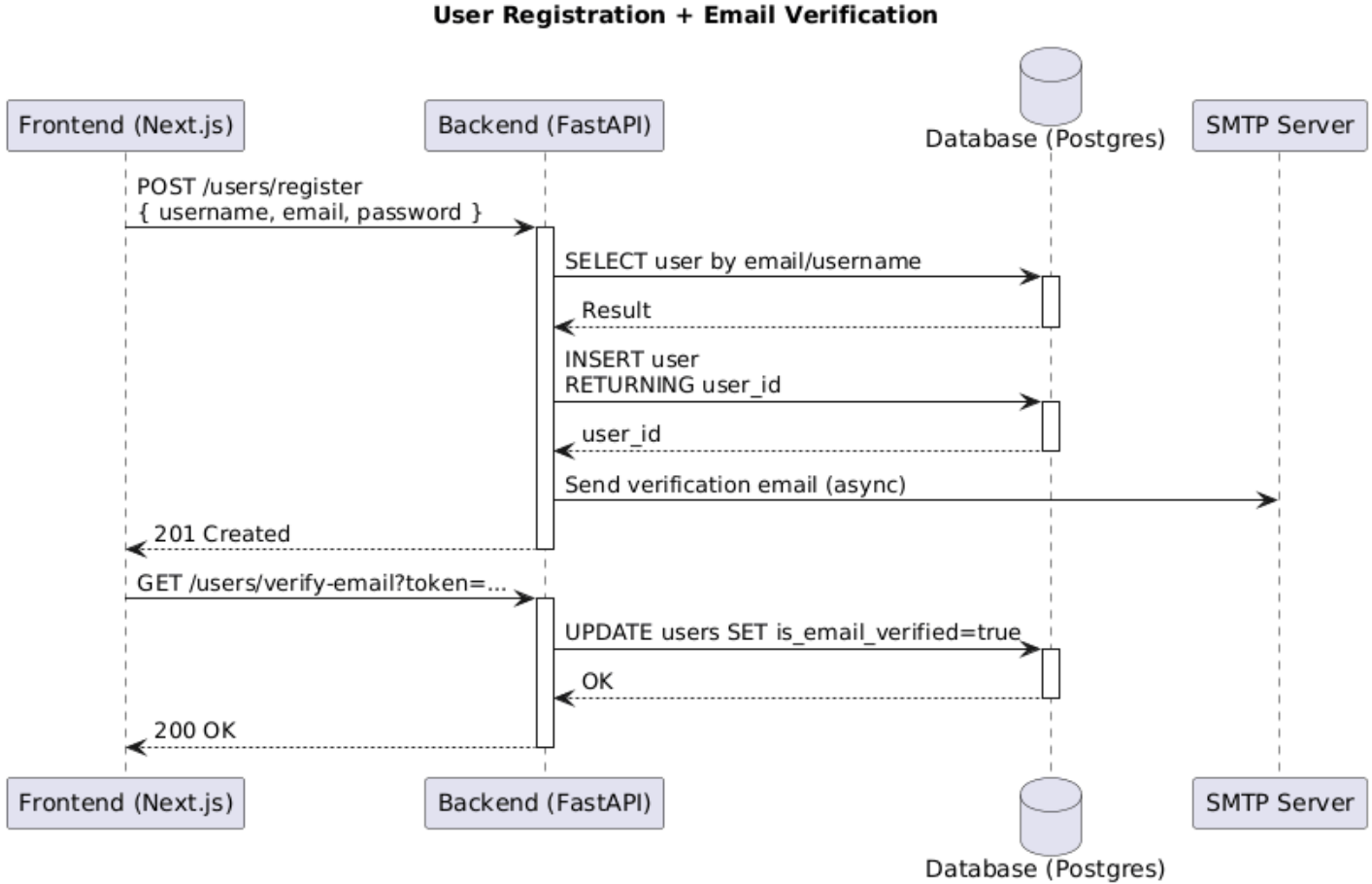
DB Schema

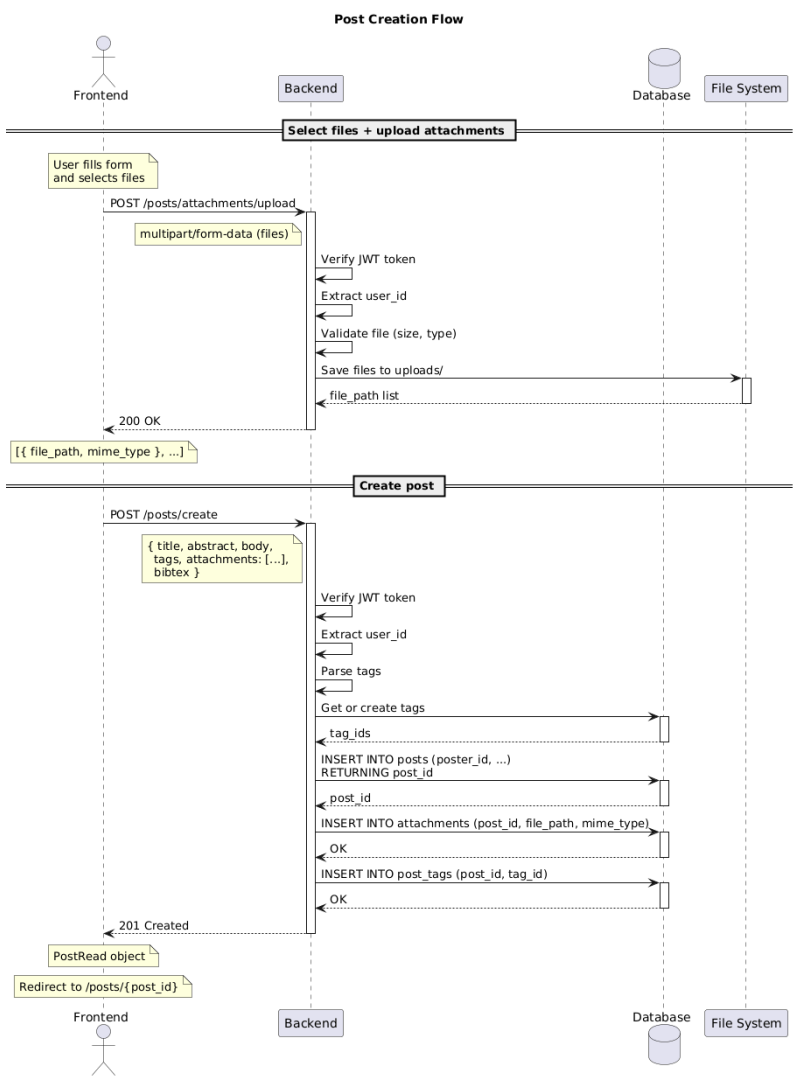

comment_votes
user_id integer
comment_id integer
value integer
created_at timestamp with time zone
id integer
user_id:id
users
username varchar
password_hash varchar
password_salt varchar(64)
role varchar(10)
email varchar
is_email_verified boolean
social_links varchar
display_name varchar(120)
bio text
affiliation varchar(255)
orcid varchar(32)
arxiv varchar(255)
website varchar(255)
twitter varchar(255)
github varchar(255)
linkedin varchar(255)
is_profile_public boolean
is_orcid_public boolean
is_socials_public boolean
is_arxiv_public boolean
created_at timestamp with time zone
id integer
reports
reported_by_id integer
target_type varchar
target_id integer
status varchar(7)
description text
created_at timestamp with time zone
id integer
comment_id:id
comments
parent_comment_id integer
post_id integer
commenter_id integer
body varchar
created_at timestamp with time zone
upvotes integer
downvotes integer
id integer
commenter_id:id
parent_comment_id:id
user_id:id
reported_by_id:id
reviews
post_id integer
reviewer_id integer
is_positive boolean
body varchar
strengths text
weaknesses text
created_at timestamp with time zone
upvotes integer
downvotes integer
id integer
reviewer_id:id
posts
poster_id integer
title varchar
authors_text varchar
abstract varchar
bibtex varchar
body varchar
phase varchar(9)
created_at timestamp with time zone
upvotes integer
downvotes integer
id integer
post_votes
user_id integer
post_id integer
value integer
created_at timestamp with time zone
id integer
post_id:id
post_id:id
post_id:id
post_id:id
post_id:id
post_tags
post_id integer
tag_id integer
attachments
post_id integer
file_path varchar
mime_type varchar
created_at timestamp with time zone
id integer
tag_id:id
tags
name varchar
created_at timestamp with time zone
id integer

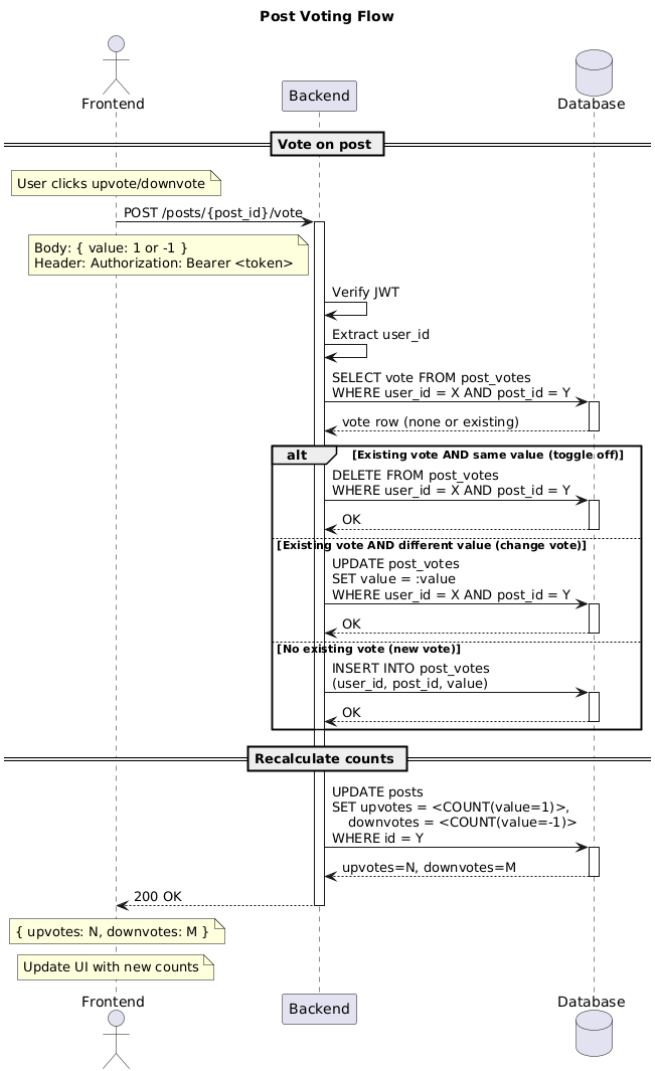
Subsystem Interactions

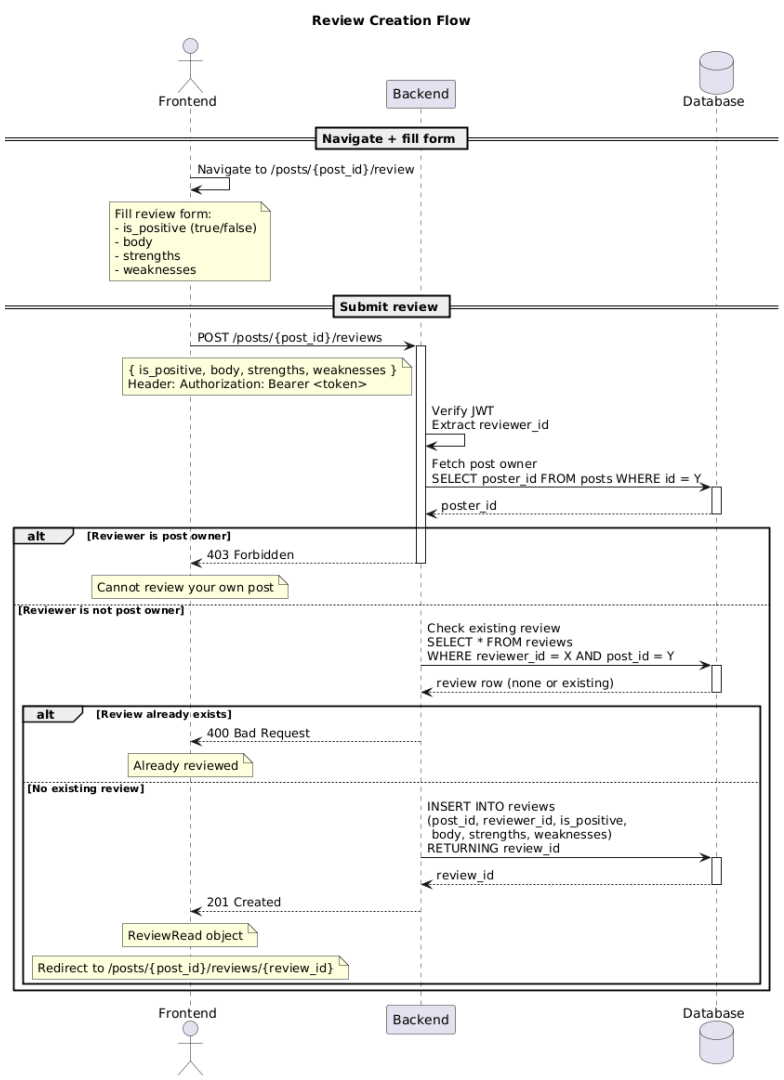



User Login flow
Frontend
Backend
Database
Login
POST /users/login
Body:
{ username, password }
SELECT user WHERE username = :username
User record
Verify password (Argon2 hash)
Check email verified?
alt
[Email not verified]
403 Forbidden
Email not verified
[Email verified]
Generate JWT access token (60 min expiry)
200 OK
{ access_token, token_type, username, role }
Client-side storage
Store token in localStorage/sessionStorage
Subsequent requests
Authorization: Bearer <token>
Frontend
Backend
Database


User Registration + Email Verification
Frontend (Next.js)
Backend (FastAPI)
Database (Postgres)
SMTP Server
POST /users/register
{ username, email, password }
SELECT user by email/username
Result
INSERT user
RETURNING user_id
user_id
Send verification email (async)
201 Created
GET /users/verify-email?token=...
UPDATE users SET is_email_verified=true
OK
200 OK
Frontend (Next.js)
Backend (FastAPI)
Database (Postgres)
SMTP Server


Post Creation Flow
Frontend
Backend
Database
File System
Select files + upload attachments
User fills form
and selects files
POST /posts/attachments/upload
multipart/form-data (files)
Verify JWT token
Extract user_id
Validate file (size, type)
Save files to uploads/
file_path list
200 OK
[{ file_path, mime_type }, ...]
Create post
POST /posts/create
{ title, abstract, body,
tags, attachments: [...],
bibtex }
Verify JWT token
Extract user_id
Parse tags
Get or create tags
tag_ids
INSERT INTO posts (poster_id, ...)
RETURNING post_id
post_id
INSERT INTO attachments (post_id, file_path, mime_type)
OK
INSERT INTO post_tags (post_id, tag_id)
OK
201 Created
PostRead object
Redirect to /posts/{post_id}
Frontend
Backend
Database
File System


Post Voting Flow
Frontend
Backend
Database
Vote on post
User clicks upvote/downvote
POST /posts/{post_id}/vote
Body: { value: 1 or -1 }
Header: Authorization: Bearer <token>
Verify JWT
Extract user_id
SELECT vote FROM post_votes
WHERE user_id = X AND post_id = Y
vote row (none or existing)
alt
[Existing vote AND same value (toggle off)]
DELETE FROM post_votes
WHERE user_id = X AND post_id = Y
OK
[Existing vote AND different value (change vote)]
UPDATE post_votes
SET value = :value
WHERE user_id = X AND post_id = Y
OK
[No existing vote (new vote)]
INSERT INTO post_votes
(user_id, post_id, value)
OK
Recalculate counts
UPDATE posts
SET upvotes = <COUNT(value=1)>,
downvotes = <COUNT(value=-1)>
WHERE id = Y
upvotes=N, downvotes=M
200 OK
{ upvotes: N, downvotes: M }
Update UI with new counts
Frontend
Backend
Database


Review Creation Flow
Frontend
Backend
Database
Navigate + fill form
Navigate to /posts/{post_id}/review
Fill review form:
- is_positive (true/false)
- body
- strengths
- weaknesses
Submit review
POST /posts/{post_id}/reviews
{ is_positive, body, strengths, weaknesses }
Header: Authorization: Bearer <token>
Verify JWT
Extract reviewer_id
Fetch post owner
SELECT poster_id FROM posts WHERE id = Y
poster_id
alt
[Reviewer is post owner]
403 Forbidden
Cannot review your own post
[Reviewer is not post owner]
Check existing review
SELECT * FROM reviews
WHERE reviewer_id = X AND post_id = Y
review row (none or existing)
alt
[Review already exists]
400 Bad Request
Already reviewed
[No existing review]
INSERT INTO reviews
(post_id, reviewer_id, is_positive,
body, strengths, weaknesses)
RETURNING review_id
review_id
201 Created
ReviewRead object
Redirect to /posts/{post_id}/reviews/{review_id}
Frontend
Backend
Database

Contributions

- **Yaraslaw Akhramenka:** profile page, report feature
- **Adelaide Danilov:** DB modelling, registration feature
- **Demyan Faguer:** review feature, report
- **Oleksandr Marchenko Breneur:** search, post, comment and vote features
- **Oleksandr Yeroftieiev:** personal page, report feature

ANY QUESTIONS?