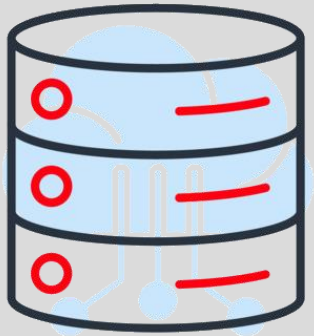
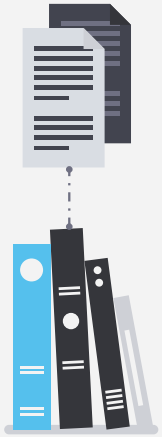
University Student Project Repository

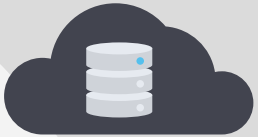


Angelica RINGS, Robert-Alexandru KISS, Enzo VERDIER--TURRIN, Louis COLBUS, Sven BARNICH

Table of contents



Why make a student project repository?



The Problem:

- Lost files
- Fragmented storage
  - Projects scattered across Moodle
  - Local hard drives
  - Descriptions in loose emails
- Inaccessible to all students

Our proposed Solution:

- Centralized DB
- Showcased projects
- Searchable by tags
- Accessible to all students and supervised by administrators

Users

Students	Professors and Recruiters	Administrator
Upload and manage their project	Browse projects	Approve new uploads
Manage profile	Rate projects	Remove inappropriate or outdated content
Rate other projects		
View portfolio		

Functional Requirements

Users/Students	Professors/Recruiters	Website Administrators
Log in	Log in	Log into Admin account
Upload Project Files	Browse Projects (with filter)	Approve new uploaded Projects
Edit / Delete uploaded Projects	View Student Profiles & Project Details	Remove inappropriate or outdated Projects
Manage Student Profile		View Student Profiles & Project Details
Manage Project Collaborators		
View other students' Profiles & Project Details		
Rate other students' Projects		
View average star rating per Project		

Non-Functional Requirements

Performance	Performance	Storage
Fully functional on screens from 360px to 1920px width	Support at least 200 concurrent users without performance degradation	Must support files up to 500 MB
Response time of < 3 seconds	99.5% uptime during Uni business hours	Store Project Data in a structured DB
	Modular architecture for simplicity	

Accessibility		Security
Adhere to the Web Content Accessibility Guidelines		Hash and salt all passwords before saving

Requirements: Not Met

Users/Students	Professors/Recruiters	Website Administrators
Log in using university email	Log in using email	Access content modification logs

Security	Storage	Accessibility
Maintain history of all content modifications and admin actions	Monthly backups retained for 60 Days	Adhere to the Web Content Accessibility Guidelines
Secure HTTPS access		
Scan for potentially malicious files		

Use Case

Upload Project Use Case

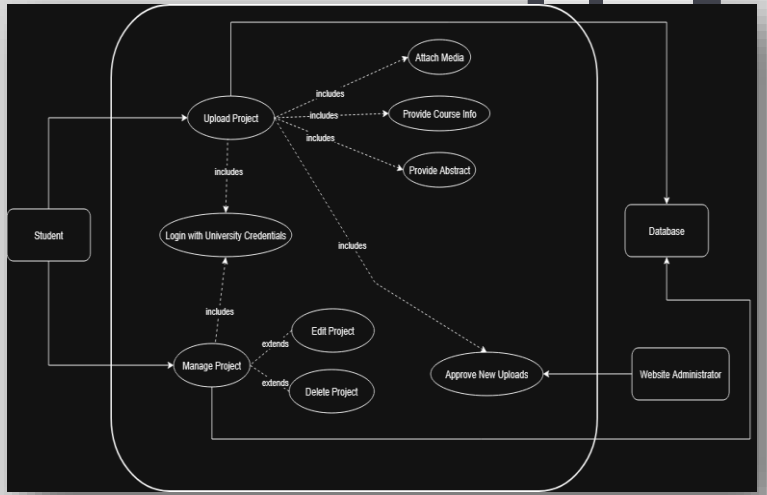
Core User actors: *Student*, *Administrator*

Core User flow:

- Upload Project(s) including *Media*, *Course Info*, *Abstract*
- Manage Project(s)

Crucial Dependency:

- Administrator needs to **approve** Project before it's publicly displayed



Use Case

Project Rating Use Case

Core User Actors: *Student*, *Professor/Recruiter*

Core User flow:

- Browse approved projects
- View project details
- Rate a project with a star-based rating

Crucial Dependency:

- User must be logged in to submit a rating
- User can't rate its own project

Use Case

Content Moderation Use Case

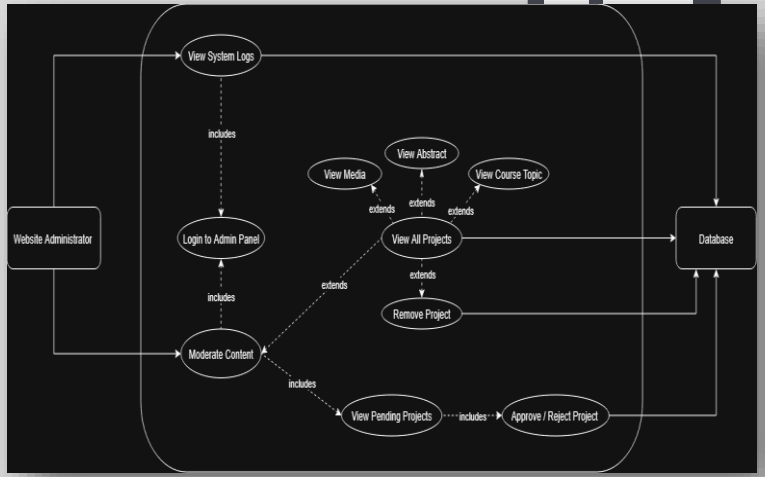
Core User Actor: *Administrator*

Core Admin flow:

- View System Logs
- Moderate Content
  - View pending Project(s) and decide approval
  - View existing Project(s) and delete if necessary

Crucial Dependency:

- Administrators needs to **log in** before being able to moderate



System Architecture

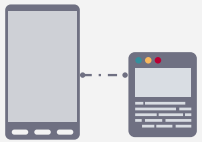
REACT Client — Rest API / Flask — SQLite DB + File Storage

3-Tier Modular Architecture

Frontend:
- Built with **React, TypeScript, Vite & Tailwind CSS**
- No page reloads, fast data fetching

Backend:
- Uses **Python** with **Flask, bcrypt, PyJWT**
- RESTful endpoints, handles logic, authentication, permission checks

Database:
- Utilizes **SQLite**
- Lightweight



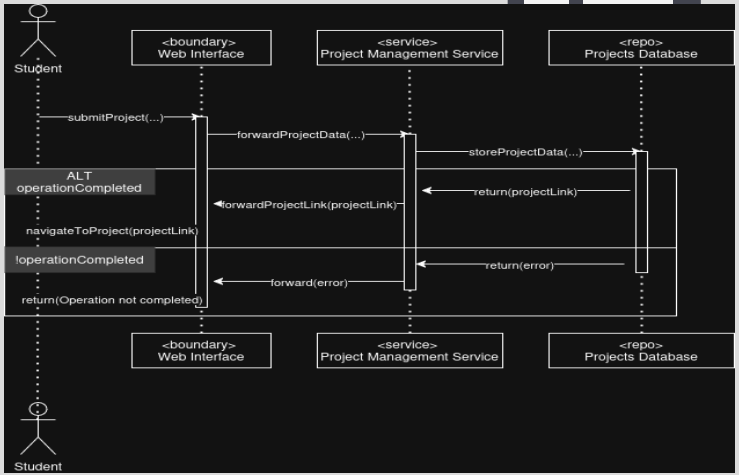
Sequence diagram

How data is processed by the system during an Upload:

- **The Boundary (React):** collects file and metadata
- **The Service (Flask):** validates token and file size
- **The Repo (DB):** stores the metadata, while file is written to disk

The system returns a success message, flags projects internally as “approved=False”

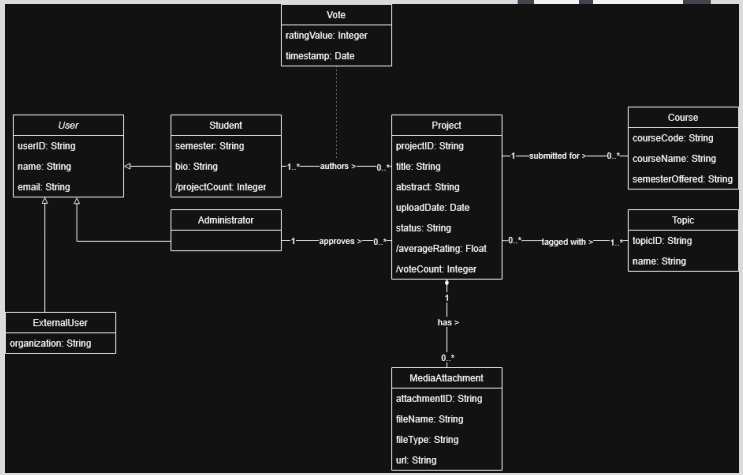
In case of error → returns error instead



Class diagram

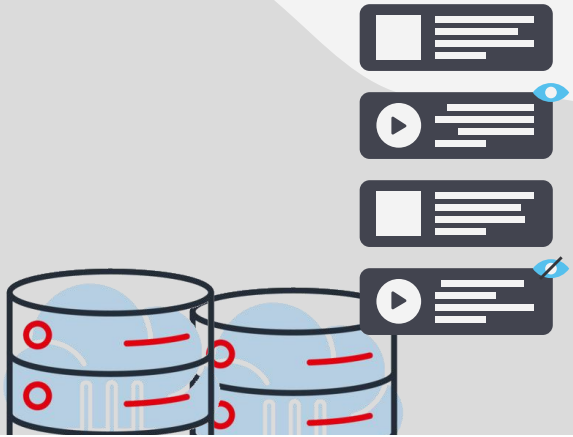
The “brain” of our Repository

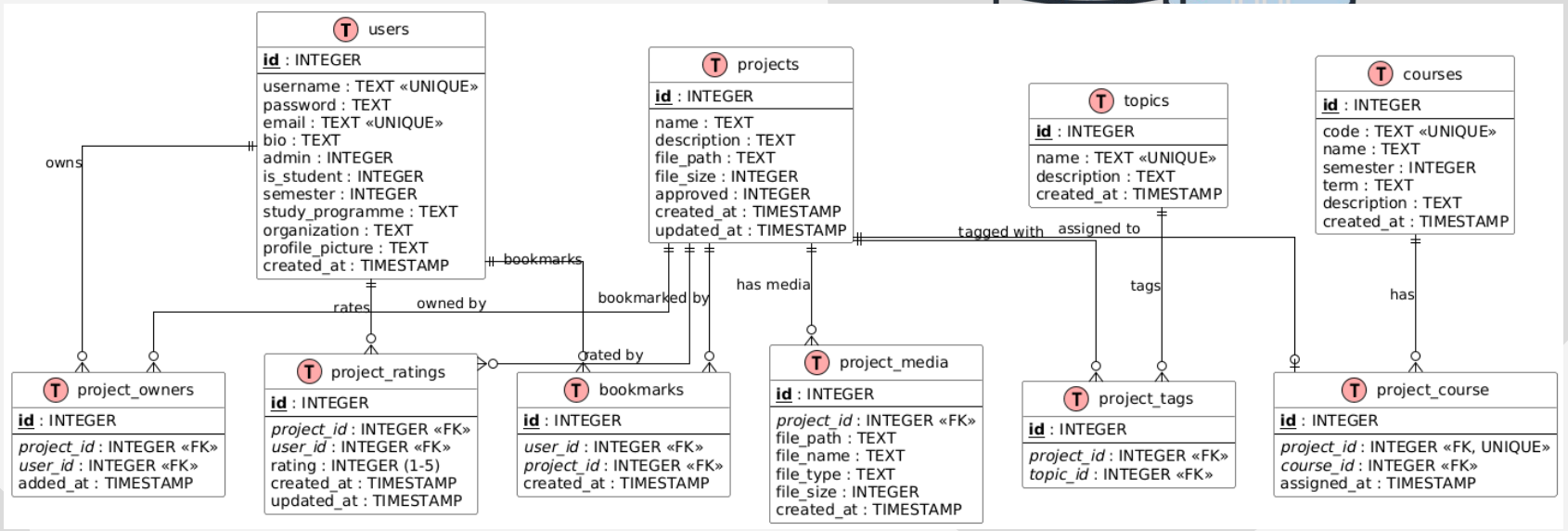
- User is a **superclass** for Student and Admin
- Created a relationship that allows for **multiple** authors per project
- Projects are tagged with Topics. Also N-to-N relationship, allowing for **flexible** categorization



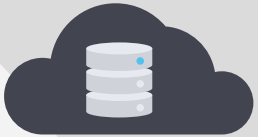
Database

- Normalize the DB to **3NF** to avoid data redundancy
- We *store the path* to files in the DB, and the actual file on the server to avoid performance degradation
- We use JSON Web Tokens for authentication, this enables the backend to be "**stateless**" – no data stores in server memory





From idea to project: be publicly available



Oracle Cloud

- Ampere A1 VPS (Arm processor):
  - 4 OCPUs
  - 24 GB of RAM
  - 200 GB block volume (storage)
  - Ubuntu 25.10

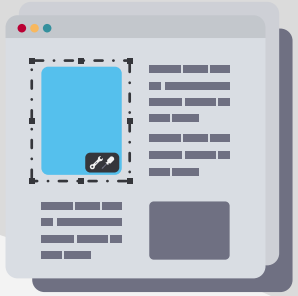
Virtual Cloud Network

- No HTTPS for now
- Subdomain *se1.insects.ltd* mapped to public IP *130.61.214.41* using CNAME record
- Ingress rule to allow traffic on TCP port 3000 (for Vite server)

Conclusion & retrospective

Overall, the MVP has been properly implemented... but, the project is **not** done yet:

- Still requires clean up, refactor, finding bugs and fixing them
- Switch to production WSGI server, add HTTPS
- Start rotating JWT secret key, think about vectors of attack like XSS and SQL injection



LIVE DEMO

You can try for yourselves!

http://se1.insects.ltd:3000/

Contribution: Implementation

Feature	who implemented	what contributed
Frontend setup & core UI	Angelica	Front end setup general UI structure, layout styling, light mode
Explore page	Angelica	Explore (main) page, project listing filtering, project cards
Project pages	Angelica	View project page, upload project page, edit project functionality
User profiles	Angelica	Profile page, visibility, pending projects view, upload profile picture
Admin features	Angelica	Admin logs, pending project review, user management, notifications, topic and course management

Contribution: Implementation

Feature	who implemented	what contributed
Notifications	Angelica	Notification (edit, delete, approval, ratings) tab and page
Bookmarks	Angelica	Bookmarking projects
Reporting system	Angelica	Report user and report projects
Authentication	Angelica	Login/sign up UI, account info handling
Unit tests	Angelica	Backend unit tests (DB and API endpoints)
Backups	Angelica	Script for the automated backups
Rating system	Angelica	Rating UI

Contribution: Implementation

Feature	who implemented	what contributed
Authentication	Robert	Login/ sign up logic
Admin moderation	Robert	Admin approval of projects topic
Upload Projects	Robert	Edit projects and edit profile UI
Backend database & API	Robert	CRUD database, API endpoints, database integration
UI styling tweaks	Robert	Colour changes and minor UI tweaks

Contribution: Implementation

Feature	who implemented	what contributed
User Profiles	Louis	Public user profiles
Misc fixes & config	Louis	File extension fixes, API, URL fixes
CI / DevOps	Louis	Backend CI workflow test automation
Hosting	Louis	Deploying and hosting application