University of Science
and Technology of Hanoi

Group Project

Students–Supervisors Collaboration Platform with Builtin Utilities for Academic Management

Authors
Đoàn Bá Cường
Ngô Ngọc Đức Huy
Đào Dương Hoàng Long
Ngô Xuân Minh
Nguyễn Gia Phong

Supervisor
Đoàn Nhật Quang, PhD

February 4, 2021

Contents

1 Introduction

1.1 Motivation

Group project and internship are annual activities for master students and last year bachelor students in University of Science and Technology of Hanoi (USTH). With the university's projected growth, managing them is becoming a complicated yet important task. The management is required for both students and the school staff. From students' and supervisors' perspective, there should be a tool that helps manage numerous tasks in their projects. From faculties' academic assistants' perspective, a system that helps collect projects' progress and evaluation is of great importance, since manual input for students' grade is an exhausting task and can lead to error.

Therefore, a collaborative platform that students and supervisors can use in their project, which also support academic evaluation and report, is needed. This tool should help students manage their projects seamlessly, and academic assistants collect statistical reports quickly.

1.2 Background

Project management is a common problem, which is why it is unsurprising that there have been a variety of project management systems. Students and supervisors have already been using some of these systems to manage their tasks. We studied these systems to see how projects are commonly and effectively managed.

Trello, a popular web service for keeping track of tasks, uses Kanban board. A Kanban board consists of several columns, each of which contains some tasks in a certain stage, such as *to-do*, *in progress*, *in review*, and *done*. User can move a task from a column to another. As this workflow is easy to learn, it is also used by other popular collaborative platforms, such as Jira, Asana, GitHub, or Tuleap.

Some other software, such as GanttProject, ProjectLibre, or Microsoft Project is modeled after another tool: Gantt chart. Gantt chart is a way of using bar chart to visualize the schedule for the project. This method simplifies scheduling independent tasks to be worked on at the same time to save wait time.

Although many of project management systems above are targeting software development projects, The application of Kanban board and Gantt chart as project management tools can benefit any other fields. In fact, using Kanban board in other fields is also known to improve productivity by

saving time and aid internal communication [1].

As effective and popular as existing project management systems are, they do not support academic evaluation, which is one of our main motivations. Extending existing free software could be an approach to achieve our target. However, the complexity of those systems makes it impractical to study within a short period. Therefore, we started this project to create from scratch an academic-oriented project management system.

1.3 Objectives

In this project, the overall aim is to develop a collaboration platform for students and mentors. The resulting system should be able to not only integrate well into their workflows and academic management tasks but also effectively perform on back-end servers often with limited resources and across a wide variety of end-user machines.

1.3.1 Building a Collaboration Platform

At the lowest level, discussion would be facilitated through comment threads, organized into logical tasks. The tasks would be categorized by stages such as *to-do*, *doing* and *done*. These stages would then be visualized as columns in a Kanban board to aid managing and prioritizing tasks.

1.3.2 Developing Utilities for Academic Management

In addition to discussion and job scheduling, the system-to-be would support academic tasks. To mentors and committees, these are grading and giving feedback. To academic assistants, these include (but not limited to) exporting transcripts and statistical reports, and tracking students' overall progress to provide help if necessary.

1.3.3 Constructing an Applicable Web Application

The system should be implemented as a web application for portability [2]. It should be easy to be adopted by users on different devices network connection quality, whilst being simple enough to be maintained by a few administrators to run on modest hardware.

Last but not least, for long-term maintainability and ethical concerns, such as protecting of digital freedom and promoting independence and cooperation in educational institutions, the system should be made available under a free software license [3].

1.4 Expected Outcomes

From the beginning, it was foreseeable[1] that a few students would not be able to complete a medium-sized system from requirement engineering to architecture, analysis and design as well as implementation, by working part-time in three months. Rather, we anticipated to finish the following:

1. Requirements clarification
2. Concrete system architecture
3. Initial system design for both functional objectives (*Building a Collaboration Platform* and *Developing Utilities for Academic Management*)
4. Implementation of a subset of the use cases, focusing on *Building a Collaboration Platform*, while satisfying non-functional requirements to be ready for real-world usage

1.5 Report Structure

The report contains following chapters:

1. **Introduction**: Introduce the motivation and background of the problem and some related works, and afterwards, define the objective and scope of the project.
2. **Requirements**: Specify use cases as well as non-functional requirements.
3. **Methodology**: Illustrate the development steps, from technical choices to architecture and system analysis and design.
4. **Results and Discussion**: Show what we did in this project and what we learned from it.
5. **Conclusion and Future Work**: Summarize the findings and suggest the work that can be done to improve the system.

[1] While we were overly ambitious and optimistic, our supervisor was more realistic and taken Hofstadter's Law into account:

> It always takes longer than you expect, even when you take into account Hofstadter's Law [4].

2 System Requirements

In this section, from the objective and context, we examined user categories and their behavior. After that, we analyzed the expected system to see which features therewill be. From there, we derived a list of use cases for the system. Finally, we add some supplementary specifications for describing nonfunctional requirements.

2.1 Types of Users

Firstly, to understand which use cases that need to be implemented, we analyzed which types of users there are and how they interact with our system.

The primary users are **students** and their **supervisors**. They can interact quite similarly with the system:

- Create a new project
- Edit public information about a project
- Invite new members to a project
- Create new tasks
- Evaluate the tasks

However, there are differences between what students and supervisors can do. Unlike supervisors, students are not allowed to evaluate the project. On the other hand, supervisors should not be able to do tasks nor to upload project reports and slides while students can do those tasks.

After the students submit the reports, there will be **judges** whose responsibility is to assess these reports. They also evaluate the groups' project oral defense (also called *presentation*). They should therefore be allowed to access and evaluate reports and slides of projects they are assigned.

During the process, **academic assistants** of each faculties should collect statistical reports of each project and print the results of the subject.

The system's user categories can be summarized in the table below

Type	Responsibility	Ability
Student	progress the project	manage projects, manage and complete tasks, upload reports and slides
Supervisor	support the group and evaluate the project	create tasks, evaluate projects
Judge	evaluate groups' reports and presentations	access and evaluate reports and presentation
Assistant	collect statistical report and evaluation	aggregate and export projects' evaluation

In the system, we assigned users of each group a role: `student`, `supervisor`, `assistant`, `judge`.

2.2 Functional Decomposition

From the project objectives, we applied divide-and-conquer strategy: decomposed each expected functionality and thereby outlined the use cases we need to implement.

The work breakdown structure (WBS) diagram in Figure 2.1 summarizes the decomposition we made.

In the following two sections, we will justify this breakdown.

2.2.1 Building a Collaboration Platform

For the users to collaborate, they should be able to create projects and tasks. They should be also able to edit them in case they make some mistakes or they need to update.

Each task should be assigned to some members. After it is done, the member should be able to complete it. As they do task, they may need to discuss the problem.

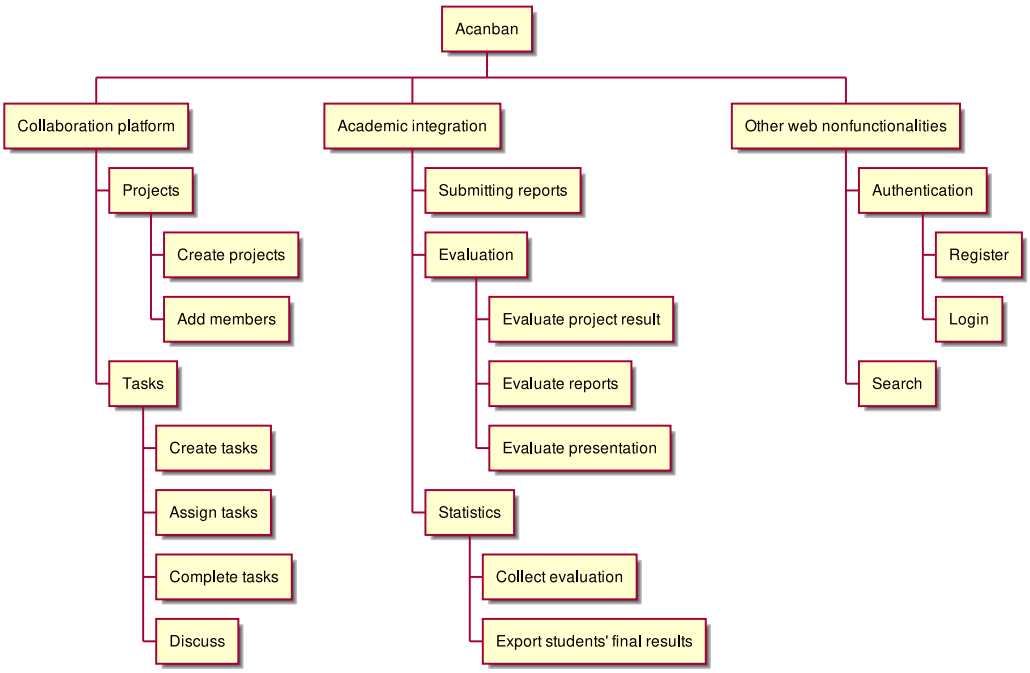


Figure 2.1: The features the system was expected to have.

2.2.2 Academic Integration

Group projects are evaluated according to three criteria:

1. The project outcome
2. The written reports
3. The presentation, also called oral defense

The first one is evaluated by the supervisor; the latter two are evaluated by the judge.

The system should therefore allow students to upload their reports, and supervisors and judges to evaluate them.

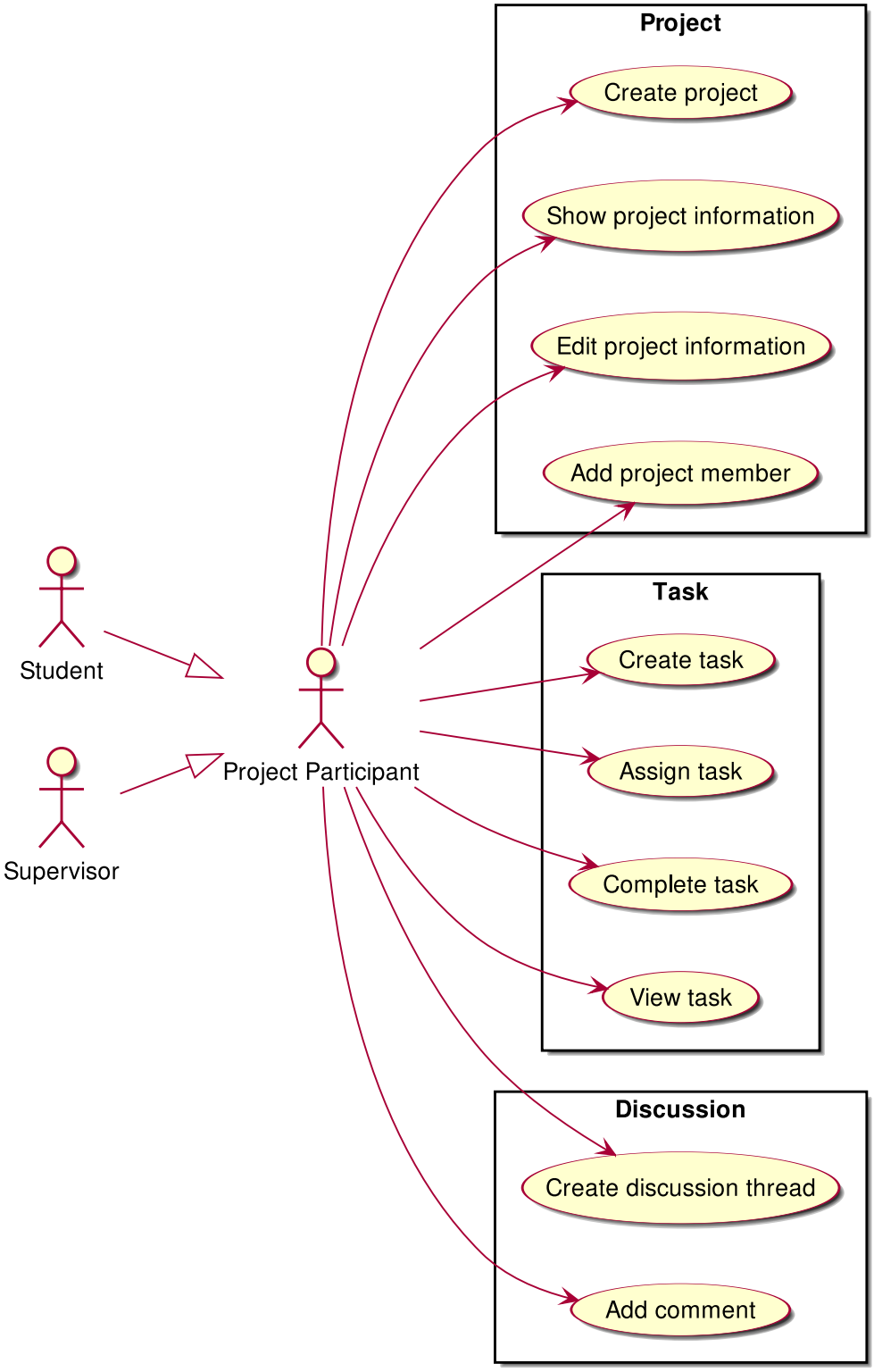
After the evaluation is made, the result is aggregated by each faculty's academic assistants, who then export the results or make statistical reports.

2.3 Use-Case Model

This section includes the list of use cases we expected to have in the system. Since the number of use cases is large, we divide the use cases to smaller groups for clarity.

2.3.1 Project Collaboration

This section aims to describe basic activities relating to the project. For *Building a Collaboration Platform*, we let users perform basic create-read-update-delete on projects, along with the help of tasks and discussion.


Project
Create project
Show project information
Edit project information
Add project member
Task
Create task
Assign task
Complete task
View task
Discussion
Create discussion thread
Add comment
Student
Supervisor
Project Participant

Create Project

This use case allows Student or Supervisor to create a new project. Its flow of events can be depicted as follows:

1. Student/Supervisor requests to create a project.
2. System receives the request and requests user to enter **name** and **description**.
3. The user provides necessary data.
4. System processes the data and create a new project.

Show Project Information

This use case allows a member to view the project's basic information. Its flow of events can be depicted as follows:

1. Participant requests to view a project.
2. System respond with the project's basic information.

Edit Project Information

This use case allows a member to edit a project's basic information. Its flow of events can be depicted as follows:

1. Participant requests to view a project.
2. System respond with a form for editing basic information.
3. Participant gives disired changes to the project's basic information.
4. System updates the project's information accordingly.

Add Project Members

This use case allows the creator of a project to add members to it. Its flow of events can be depicted as follows:

1. The creator selects other users to add to project.
2. The System adds the selected users as members of the project.

Create Tasks

This use case allows Student or Supervisor to generate tasks for the project. Its flow of events can be depicted as follows:

1. Student/Supervisor requests to generate tasks the project.
2. System receives the request and requests user to provide **name**, **assignees** and **deadline**.
3. User provides necessary data.
4. System processes the data and updates tasks list.

Assign Tasks

This use case allows a participant to assign a task to someone. Its flow of events can be depicted as follows:

1. Student selects the task and choose "Assign".
2. Student chooses the participant to assign to.
3. System receives the request and register the participant as assigned for that task.

Complete Tasks

This use case allows Student to complete task(s) in the project. Its flow of events can be depicted as follows:

1. Student requests to complete task(s) in the task list.
2. System receives the request and requests Student to hand in evidences.
3. Student submits a file or a link as evidence.
4. System receives the evidence and marks task(s) as completed.

Create Discussion Thread

This use case allows Student or Supervisor to create a discussion thread. Its flow of events can be depicted as follows:

1. User requests to create a new discussion thread.
2. System receives the request and requests user to enter **title** and **content**.
3. User provides necessary data.
4. System processes the data and create a new thread.

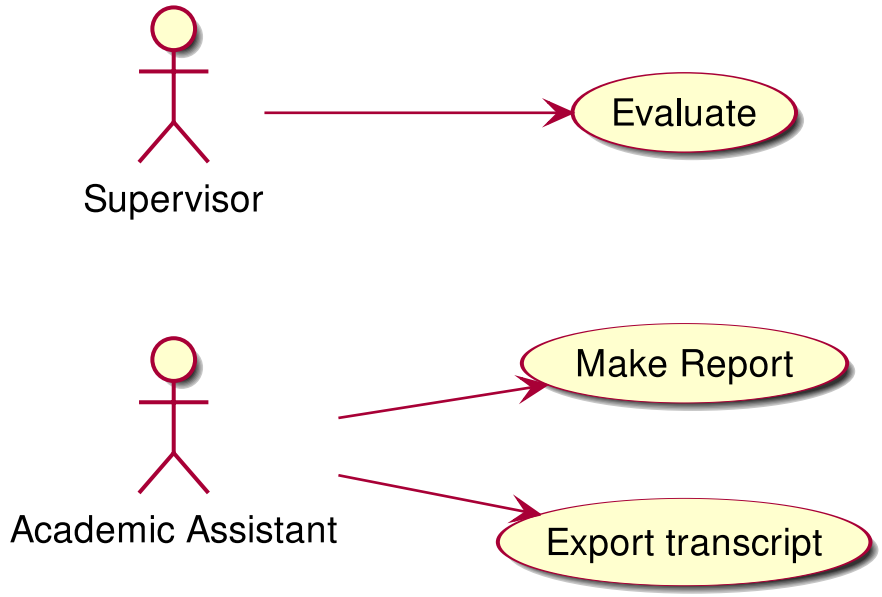
Add Comment

This use case allows Student or Supervisor to add a comment to a discussion thread. Its flow of events can be depicted as follows:

1. User requests to add a new comment to the discussion thread.
2. System receives the request and requests user to enter **comment**.
3. User enters a comment.
4. System processes the data and create a new comment in the thread.

2.3.2 Academic Evaluation

This section aims to describe basic activities relating to the evaluation. In order to build the academic integration, we let supervisors do evaluation. In addition, academic assistant could also track student progress. Overall, the use cases defined here are the functional requirements for *Developing Utilities for Academic Management*.



Evaluate

This use case allows Supervisor to evaluate the project. Its flow of events can be depicted as follows:

1. Supervisor requests to evaluate the project.
2. System receives the request provide an **evaluation** and **comment**.
3. Supervisor provides evaluation with comments.
4. System receives the data and terminate the project.

Make Report

This use case allows Academic Assistant to make reports based on the result of the project. Its flow of events can be depicted as follows:

1. Academic Assistant requests to make a report.
2. System receives the request and displays the result of the project.
3. Academic Assistant requests to make a hard copy.
4. System responds a document file format of the report.
5. Academic Assistant downloads the file for printing purpose later.

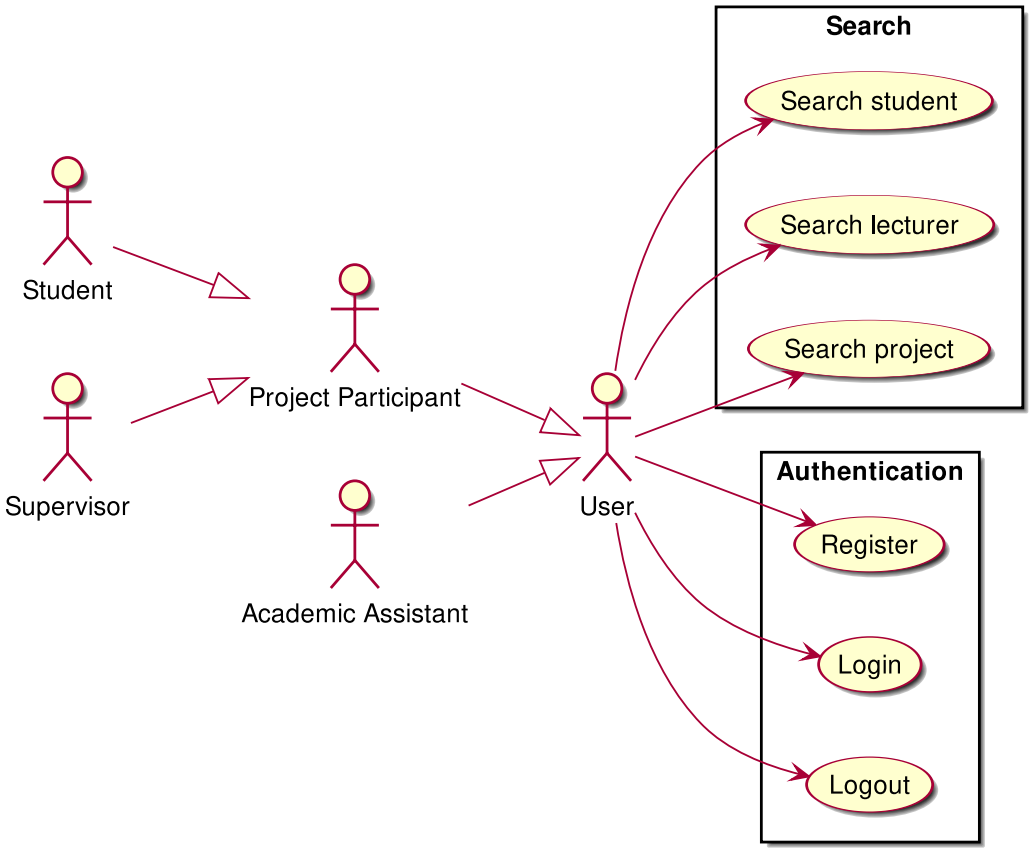
Export Transcript

This use case allows Academic Assistant to export transcript based on the evaluation of Supervisor. Its flow of events can be depicted as follows:

1. Academic Assistant requests to export the transcript.
2. System receives the request and displays the evaluation of Supervisor.
3. Academic Assistant requests to make a hard copy.
4. System responses by a document file format of the transcript.
5. Academic Assistant downloads the file for printing purpose later.

2.3.3 Other Use-Cases

As for most web applications, this system is required to perform basic tasks such as authentication and searching. In other words, those defined here are the bridges between the non-functional requirements (of *Constructing an Applicable Web Application*) and the use cases previously declared.



Register

This use case allows user to create his/her own account. Its flow of events can be depicted as follows:

1. User requests sign up for the system.
2. System receives the request and requests user to enter **username**, **password**, **name**, **role**, **and email**
3. User provides necessary information.
4. System requests user to verify by his/her provided email.
5. User verifies by his/her provided email.
6. System allows user to log in the system with newly created account.

Login

This use case allows user to log in the system.

1. User requests to log in the system.
2. System receives the request and requests user enter **username** and **password**.
3. User provides username and password.
4. System validates the entered username and password and allows user to log in the system.

Logout

This use case allows user to log out of the system. Its flow of events can be depicted as follows:

1. User requests to log in to the system.
2. System receives the request and allows user to logout the system.

Search Student

This use case allows User to search for students. Its flow of events can be depicted as follows:

1. User selects “Student” on search bar.
2. Sytem requests for the search query.
3. User enters the name of student that his/her wants to search.
4. System receives the search request including the name, and responds with a list of students matched with the provided name.

Search Supervisor

This use case allows user to search in list of lecturers. Its flow of events can be depicted as follows:

1. User selects “Supervisor” on search bar.
2. User enters name of lecturer that his/her wants to search.
3. System receives the search request including the name, and responds with a list of supervisors matching the provided name.

Search Project

This use case allows user to search in list of projects. Its flow of events can be depicted as follows:

1. User selects "Project" on search bar.
2. Sytem requests for the search query.
3. User enters name of project that his/her wants to search.
4. System receives the search request including the name, and responses with a list of projects matched with the provided name.

2.4 Supplementary Specification

Besides the functionalities specified in the Use-Case model, following are more specifications for non-functional requirements. Together, these are concrete realization of requisites for *Constructing an Applicable Web Application*.

2.4.1 Usability

The system should be intuitive to any users. Users should be able to use it with little to no training.

In order to achieve this, we derived specific design contraints:

- The user interface should not contain superfluous and distracting features.
- The user's projects should be easily found as soon as the user sign in (for students and supervisors).
- All bodies of text should be readable and legible. Text font should thus be decided by the user since each user finds a different font easier to read.
- Error messages should be informative

2.4.2 Reliability

The system must be available 99.99% of the time (that is, there must be no more than a few seconds of outage a day).

2.4.3 Performance

The system should be able to support up to 1000 users simultaneously performing different tasks at any given time.

2.4.4 Supportability

The user interface must be functional on both desktop or laptop computers, tablet, and on smartphones.

The file system must support uploading various file formats that are used for discussing and reporting project results:

- Document: docx, pdf, tex, md, rst, plain text, etc.
- Audiovisual: png, jpg/jpeg, tiff, ogg, mp3, mp4, etc.
- Presentation: pptx, tex, pdf

2.4.5 Security

The system must not allow internal information to be accessed and modified by an unauthorized user. Passwords should be all stored as hash.

The systems should not be vulnerable to common security threat, such as XSS, SQL injection, DDoS attack.

The connection to the server must use HTTPS protocol, that is, it must be encrypted with TLS/SSL.

2.4.6 Legal Constraints

The resulted software should be released under a copyleft free license, namely Affero General Public License version 3.0, in order to persist digital freedom for education.

3 Methodology

This section concerns technologies chosen for this project, user interface design and implementation.

3.1 Technical Choices

Before starting to work on the project, we had to decide on the technology we would use for constructing this system. In this section, we will introduce the tools we used for development in this project and justify those choices.

3.1.1 Concurrency

Since most, if not all, of the system's use cases are I/O intensive, it is natural to employ an asynchronous input/output framework to solve concurrency issues. Trio was picked for the ease of flow control offered by structured concurrency.

3.1.2 Web Server and Framework

As asynchronous input/output was chosen, *ASGI* was needed to cooperate the asynchronous routines in the application and the server. For server, Hypercorn was picked for its first-class support for Trio. The application was built on top of Quart, a microframework similar to Flask but for ASGI, for its flexibility.

3.1.3 Templating Language

We use Jinja2 for templating, which is the default choice for Quart. It provides us with various tools that help secure the website easier. For example, it allows HTML escaping [5], which helps avoid XSS injection.

3.1.4 Persistency

For persistency, there are two separate concerns: storing the system's metadata, which is a collection of small chunks of plain text, and storing the files uploaded by the users. In the former case, RethinkDB, a document-oriented database, was chosen for recursive and less-structured data support. Such database, however, is performance-wise unsuitable for larger files, so *IPFS*, which is a distributed file system that provides similar abstraction, is used instead in the latter case.

By choosing RethinkDB instead of SQL, we do not have to worry about SQL injection or similar database attack. This is because the query language ReQL is not based on string parsing [6], but built as methods in the implementing language.

3.2 System Architecture

As previously mentioned, Acanban implements an *ASGI* application. The general architecture is illustrated in Figure 3.1.

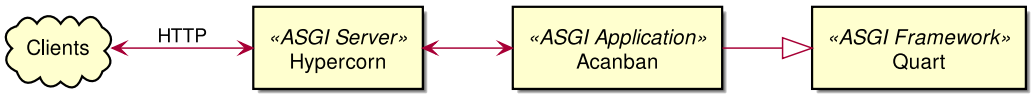


Figure 3.1: Acanban's role in the Asynchronous Server Gateway Interface

All the data and business logic handling is done on the server side in order to reduce workload for the clients, and thus the details involving the clients are not discussed in this section. *Requests* to and *responses* from the ASGI application are transfered through the ASGI server[2] Hypercorn. Typically, these messages are encrypted via TLS/SSL[3].

Persistency layers are provided by separate servers, which in our case are deployed on the same machine as the ASGI server and application. This has the following benefits:

- The network latency is significantly lower within a machine.
- We are spared from concerning with various security measures that would have been necessary were these servers are exposed to the Internet.
- It is more economic to maintain fewer physical (or virtual) machines.

While files uploaded by the users and generated by the system are stored in *IPFS*, other data are managed by RethinkDB. With these components, our test server, which is registered under the domain of acanban.ga, has the top-level view described in Figure 3.2.

As both IPFS and RethinkDB are designed to be distributed and the HTTP is distributed by nature, it is straightforward to scale the system. A load balancer such as nginx can be added to distribute the requests to multiple Acanban instances as is shown in Figure 3.3. IPFS and RethinkDB can also be run as *clusters*.

[2] Not to be confused with the physical machine that *serves* over HTTP that may run the ASGI application and other software.

[3] HTTP over TLS or SSL is often known as HTTPS.

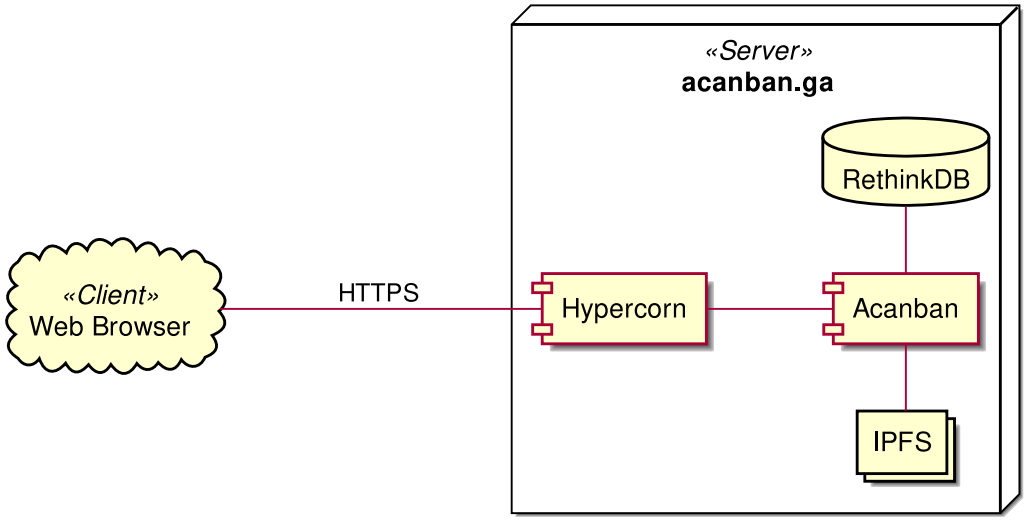


Figure 3.2: Deployment architecture

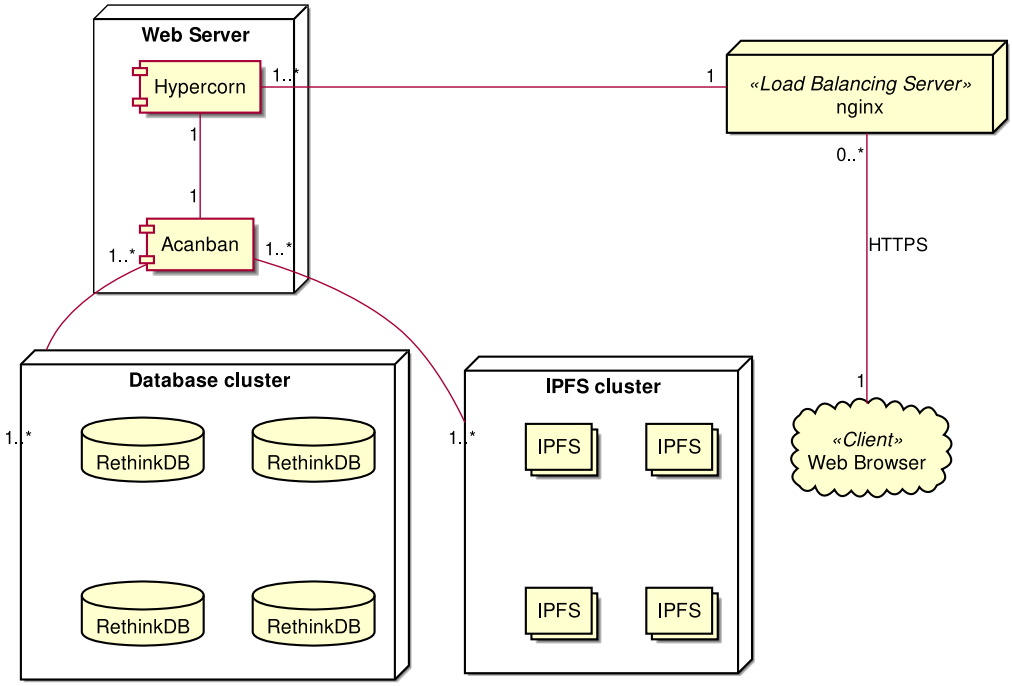


Figure 3.3: Alternative architecture with load balancer

However, we do not implement this architecture within the scope of this project, due to following reasons:

- We do not have several servers to implement.
- For intended use, the expected requests can go up to as many as 1000. Load balancing for such few requests is overhead.

3.3 Use-Case Analysis and Design

In this section, we will address the previously stated requirements. Each use case previously defined will be analyzed and from there we design the logical flow for the implementation. Due to the time limitation, we could only designed the system for some few use cases related to authentication and project management.

3.3.1 Security

Some designs that address security concerns—authentication, authorization, encryption, preventing attacks—are described in this part.

Authentication

In order to protect the data from unauthorized users, we must first verify their identity, i.e., authenticate the users.

To register for an account, the user should fill a form with personal information:

- real name
- username
- password
- the user's role, which can be student, supervisor, or assistant

When user submits a form to register for an account, the register controller will check the database to see if the username exists. If the username does not exist, the controller will continue to create an account in the database. The password is stored in the database as a hash for security.

After the successful account creation, the user will be redirected to the home page.

If the username or email is taken, the controller should inform so to the user.

Another exceptional flow happens when the user request an invalid role. This should not happen when the user submit the form via browser, but it can happen if someone is submitting it via a nonstandard way.

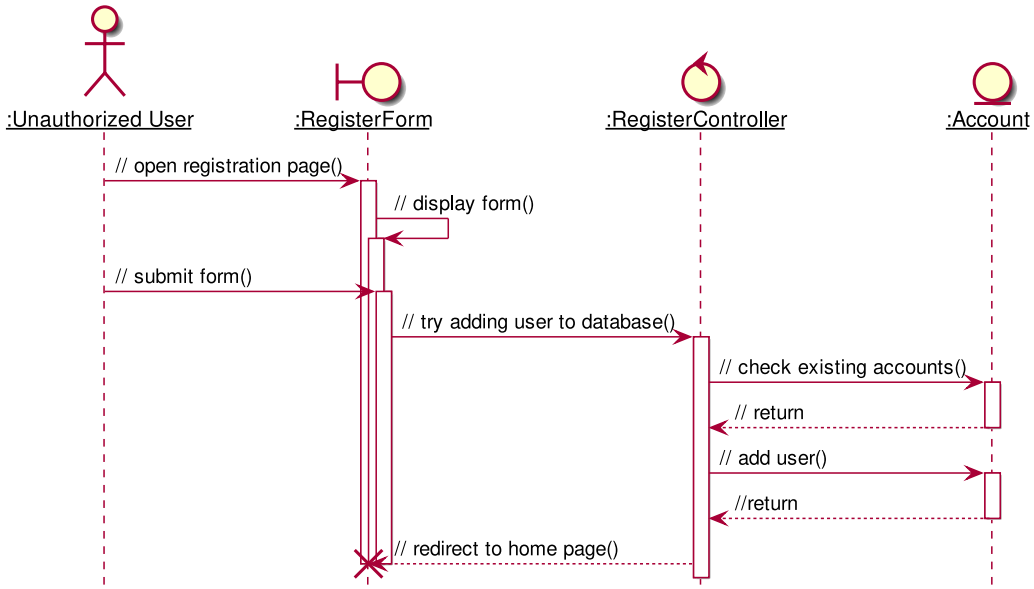


Figure 3.4: Analysis sequence diagram for the basic flow of registration process

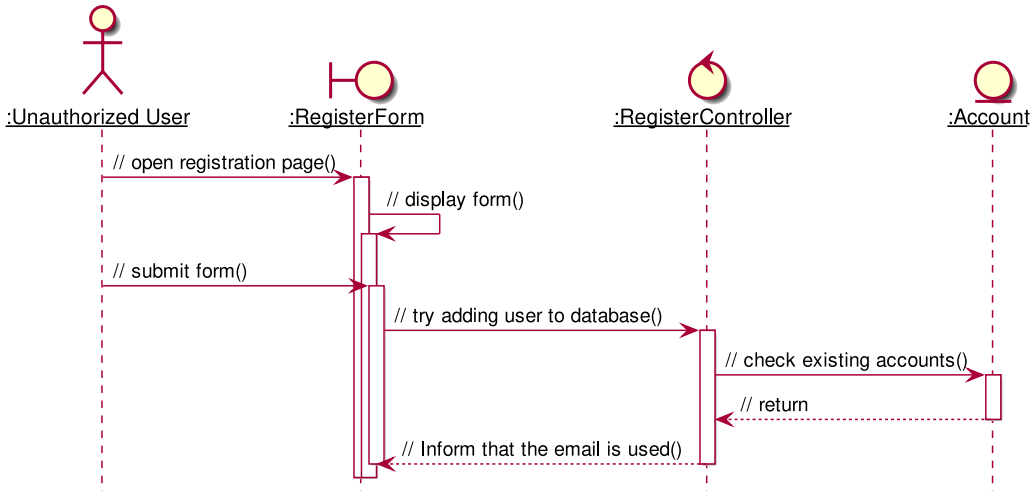


Figure 3.5: Analysis sequence diagram for the alternative flow of the registration process where the mail is taken

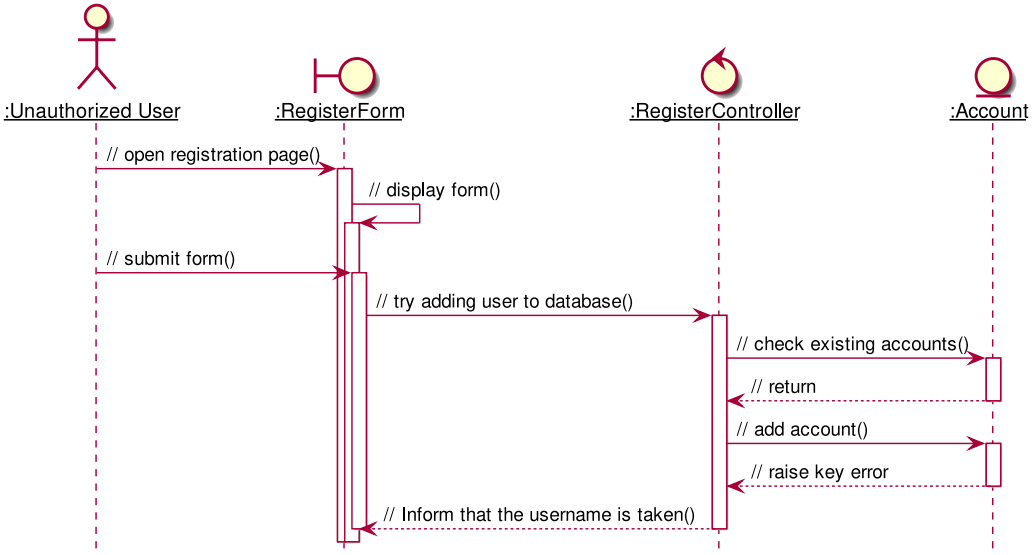


Figure 3.6: Analysis sequence diagram for the alternative flow of the registration process where the username is taken

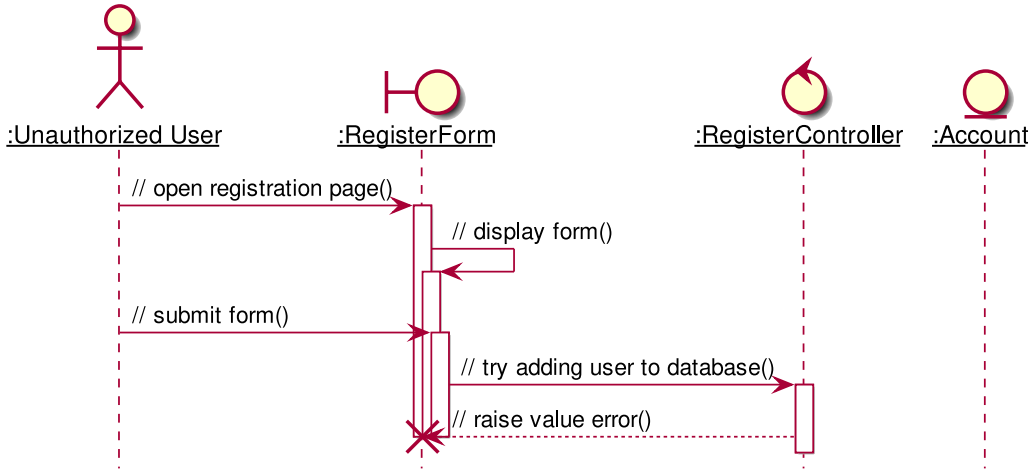


Figure 3.7: Analysis sequence diagram for the alternative flow of the registration process where the role is invalid

In order to prevent faking identity, we intended to require a token for registration. This token is provided by the system administrator and each person can only receive one token, so if they use it for faking identity, they cannot create their account anymore and have to suffer th consequence.

Alternatively, the administrator can remove the register endpoint and generate the accounts for each user.

However, we did not implement either of these schemes.

After that, the user can log in to the site in the login endpoint. After submitting the form with their username and password, they should be logged in.

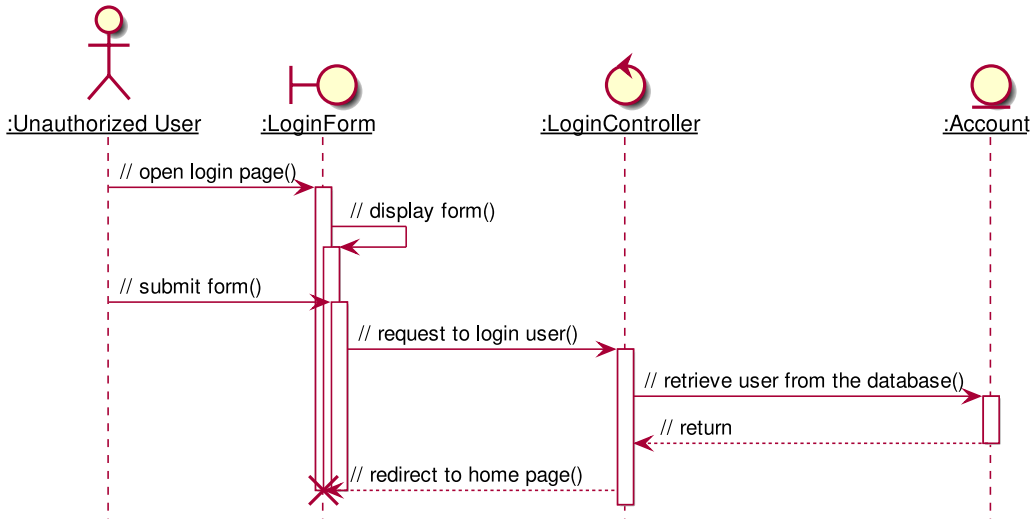


Figure 3.8: Analysis sequence diagram for the basic flow of login

However, if the user tries to log in with a non-existent account, the controller should raise an error and inform the user so.

If the user input wrong password, the user should also not be logged in and be informed of wrong password.

Authorization

After authenticated, the users are authorized according to their role and their identity. For example, a user with role “assistant” cannot participate in a project, or student cannots edit a projects they do not participate in.

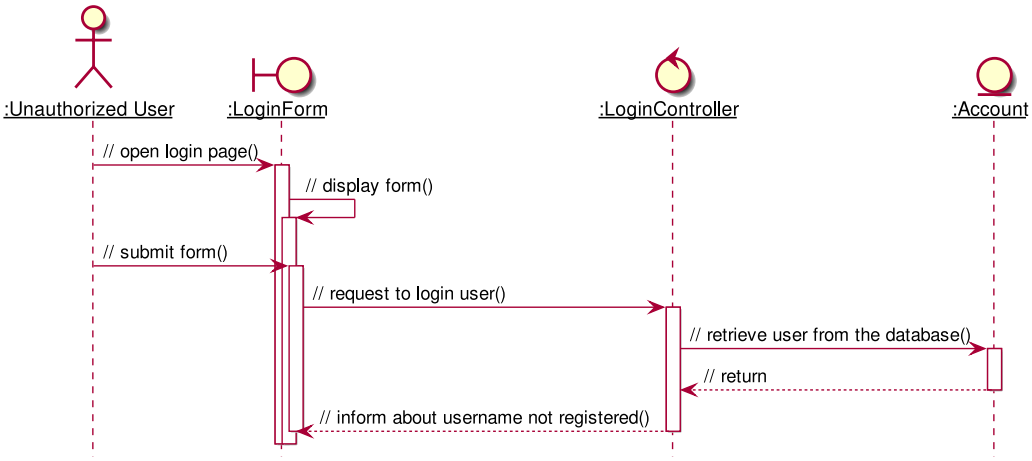


Figure 3.9: Analysis sequence diagram for the alternative flow of login where there is no user with the username

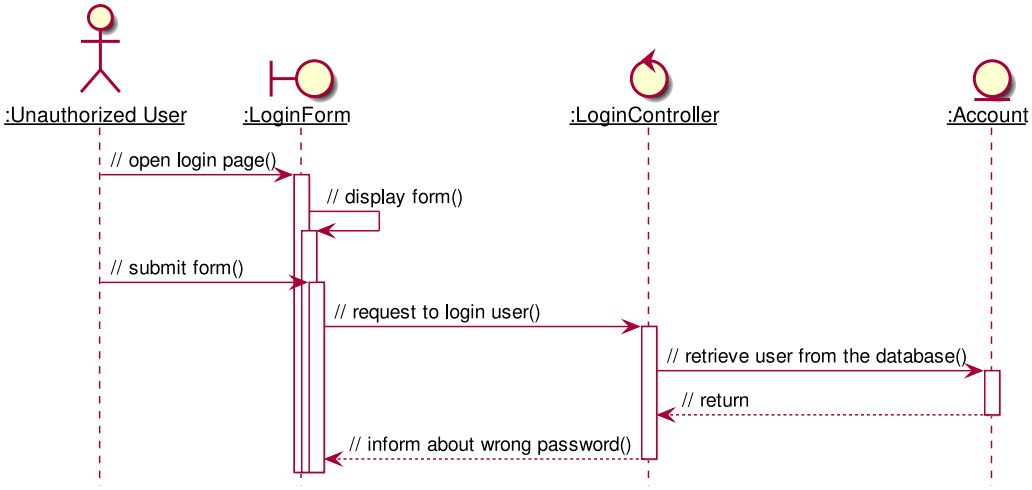


Figure 3.10: Analysis sequence diagram for the alternative flow of login where the

Encrypted Connection

To protect the data sent through HTTP, we upgraded it to HTTPS by creating a TLS certificate on the server side. Furthermore, the server is configured to use secure cookies, that is, cookies that can only be sent via HTTPS.

Injection Attacks

XSS Attack

Jinja by default escapes all HTML tags. This means that if an attacker tries to inject a script into the content, for example, setting project description as `<script>sendSensitiveData()</script>`, the script tags would appear as is and not parsed as a script element.

Moreover, the server is configured to use same-site and HTTP-only cookies, which renders any cookie-stealing JavaScript useless.

3.3.2 Project Management

In this section, we present the system design for project management use cases. Task management use cases, while indeed are part of project management, are discussed in the next.

Create Project

The function allow creating projects.

The implementation involves two database tables:

- `users` database table, where `role` index is checked.
- `projects` database table, where new document is created.

When user requests to create project, the project controller checks with `quart_auth` whether the person is authenticated. The controller then checks in `users` database that if the current user is not assistant. When both conditions are satisfied, controller returns a form for user to create project. After user fills in the form, the controller requests updating `project` database accordingly.

If user is not authenticated, controller should inform that user has not logged in.

If user is an assistant, controller should inform that user are not allowed to create projects.

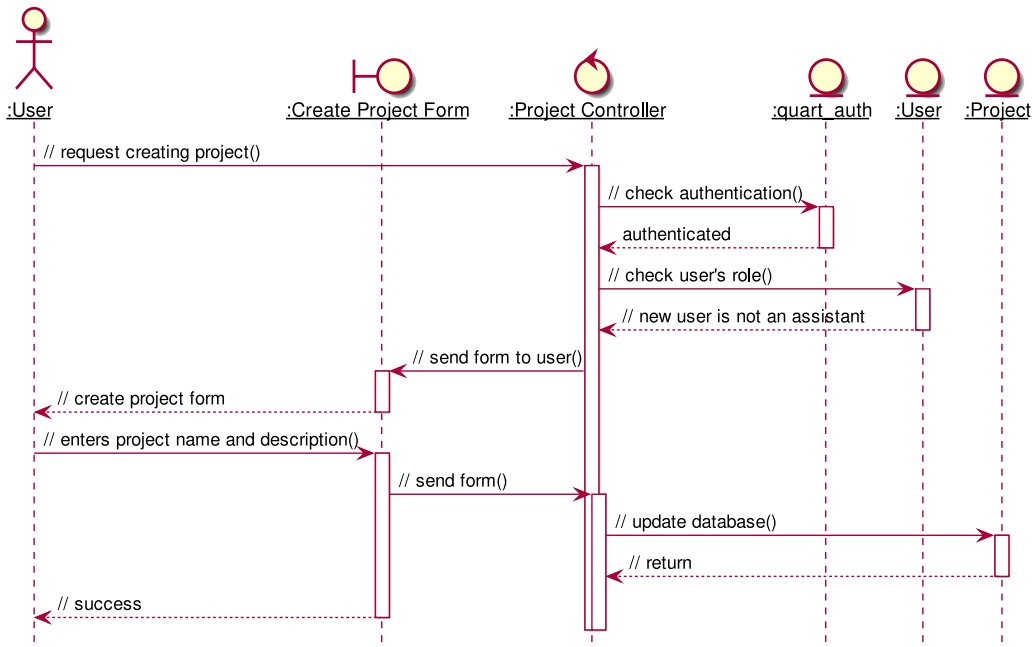


Figure 3.11: Activity diagram illustrating the successful project creation

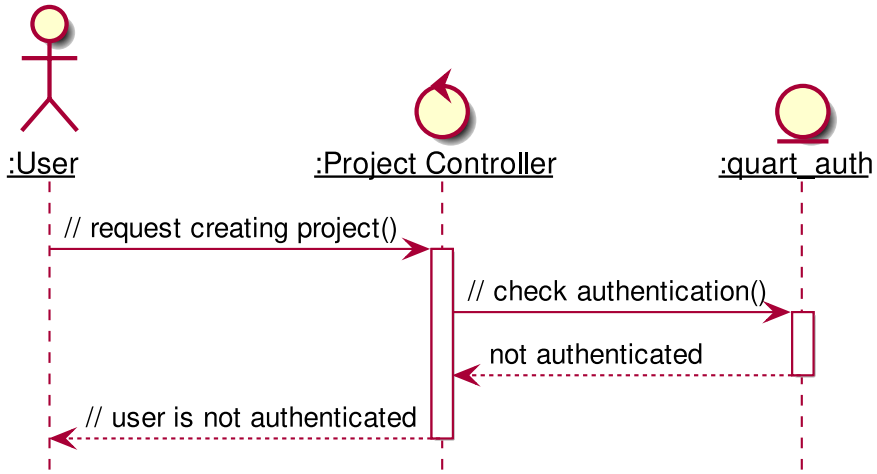


Figure 3.12: Analysis sequence diagram for create project by non-authenticated user.

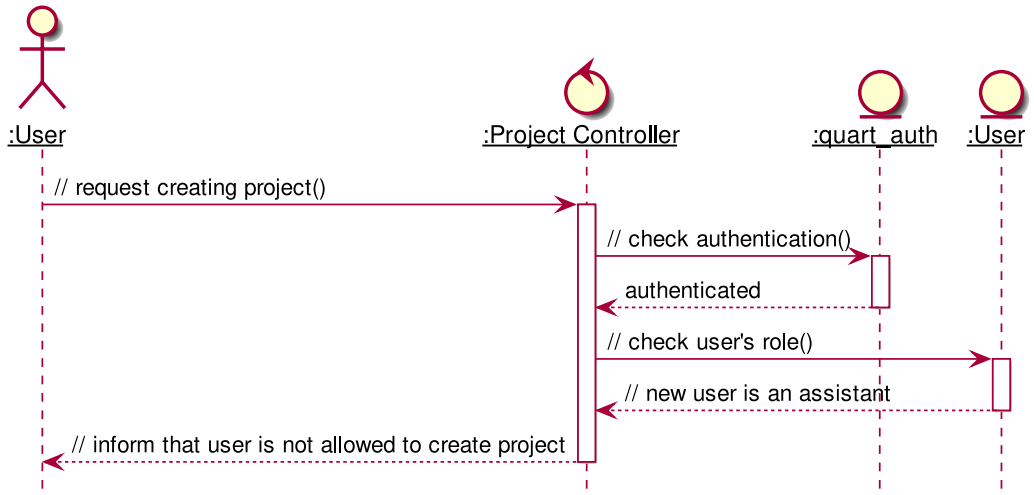


Figure 3.13: Analysis sequence diagram for create project by assistant.

Show Project Information

The function allow showing project information.

The implementation involves only `project` database table, where `id`, `students` and `supervisors` indexes are visited

When user requests to create project, the project controller checks with `quart_auth` whether the person is authenticated. The controller then checks in `project` database that if the current user is a member of project, and if the project id is existed in database table. When three conditions are satisfied, controller show the project information.

If `quart_auth` returns user is not authenticated, controller should inform accordingly.

If project id is not in the database, controller must show the error

If user is not in the project, controller must show the error

Edit Project Information

The function allow editing project information.

The implementation involves only `project` database table, where `id`, `students` and `supervisors` indexes are visited

When user navigate to edit tab, the project controller checks whether or not the person is authenticated. After that, it checks if the current user is a member of project, and if the project id is existed in database table. If all conditions are satisfied, controller shows the form for user to fill in. When

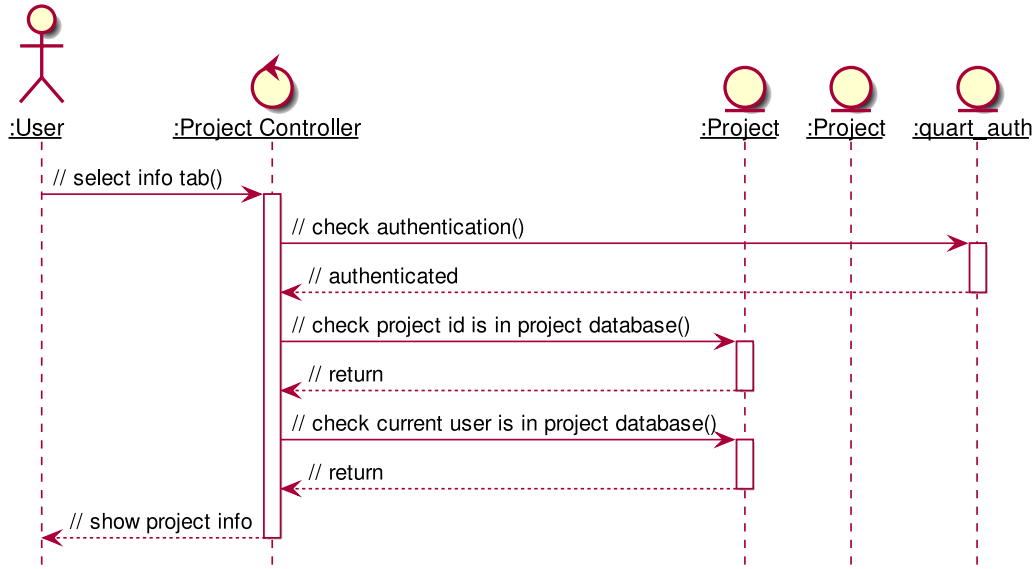


Figure 3.14: Analysis sequence diagram for successfully show project information.

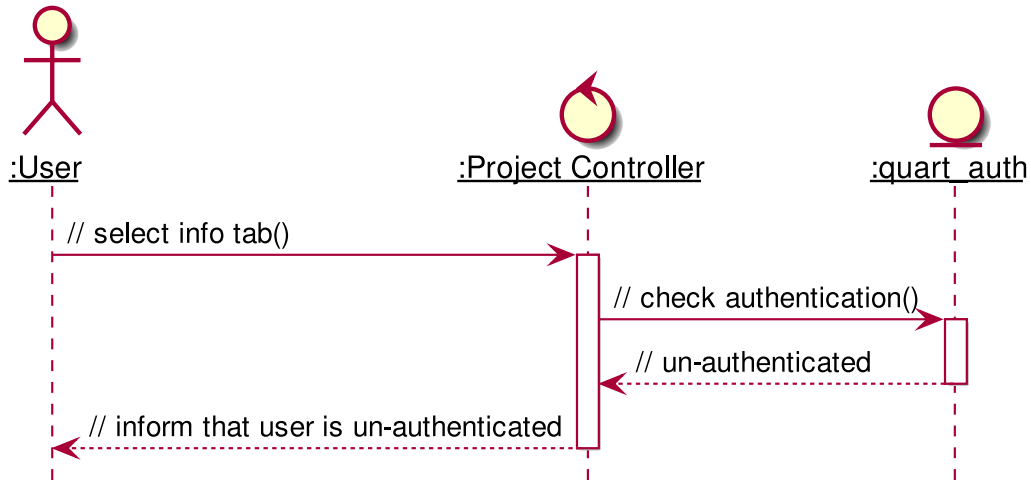


Figure 3.15: Analysis sequence diagram for showing project information when user is not authenticated.

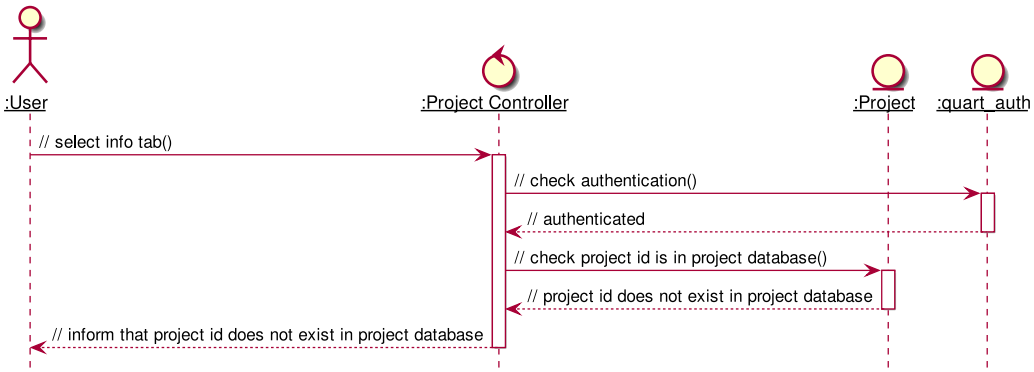


Figure 3.16: Analysis sequence diagram for showing project information when project id does not exist.

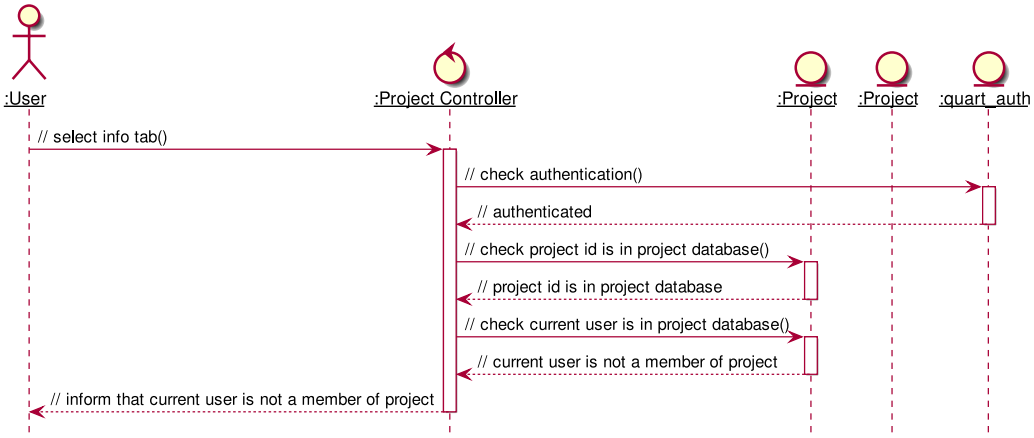


Figure 3.17: Analysis sequence diagram for showing project information when user is not a project's member.

user fills the form, controller updates the `project` table with the extracted data

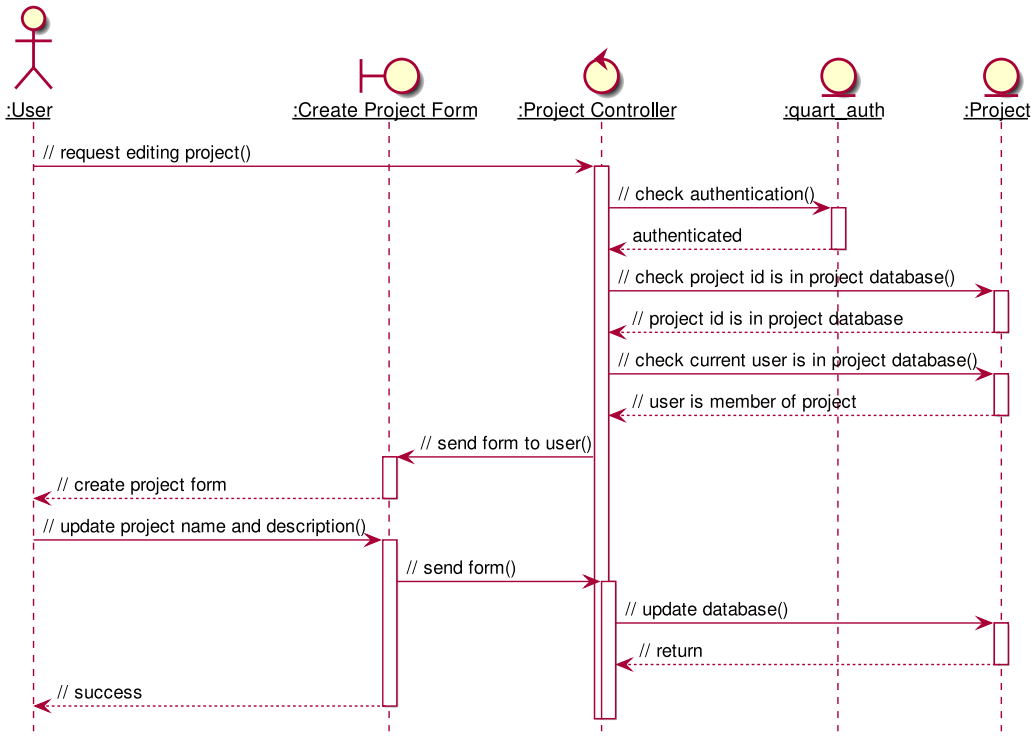


Figure 3.18: Analysis sequence diagram for successfully edit project information.

If user is not authenticated
If project not exists in `projects` database table
If user is not a member in the project

List member

The function allow listing the members in the projects.

The implementation involves only `projects` database table, in which we get the member list of members. Two fields are being called is `supervisors` and `students`.

When user navigate to member tab, the list of members in the project, classified as `supervisors` and `students` is shown in a form.

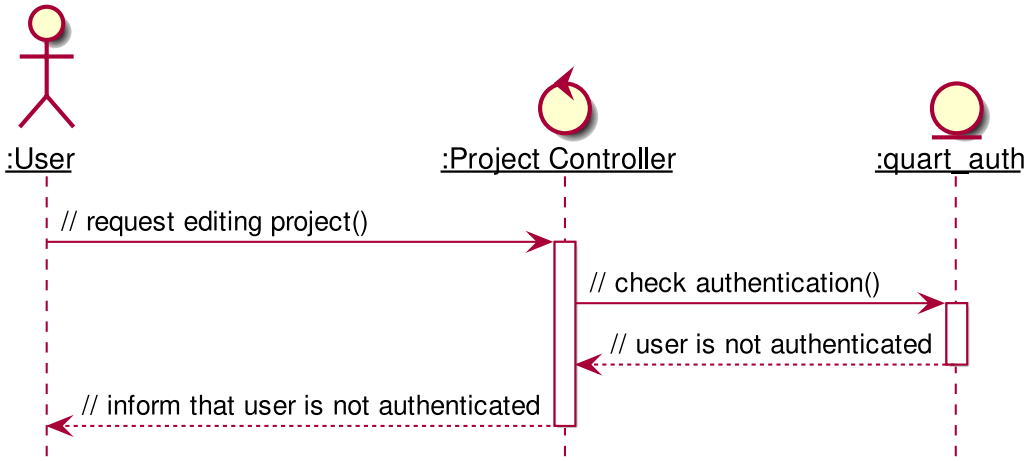


Figure 3.19: Analysis sequence diagram for editing project with unauthenticated user.

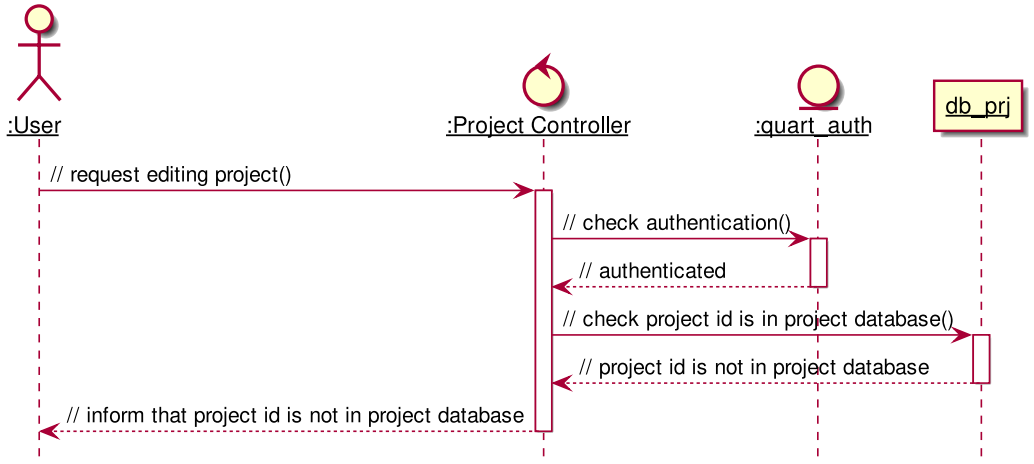


Figure 3.20: Analysis sequence diagram for editing projects when project id does not exist.

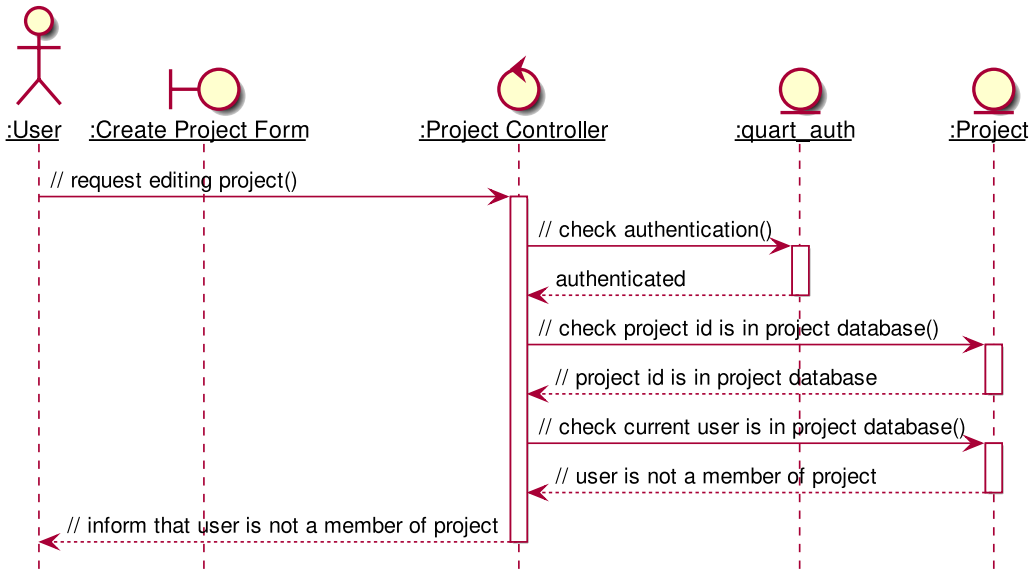


Figure 3.21: Analysis sequence diagram for editing projects when user is not a member of the project.

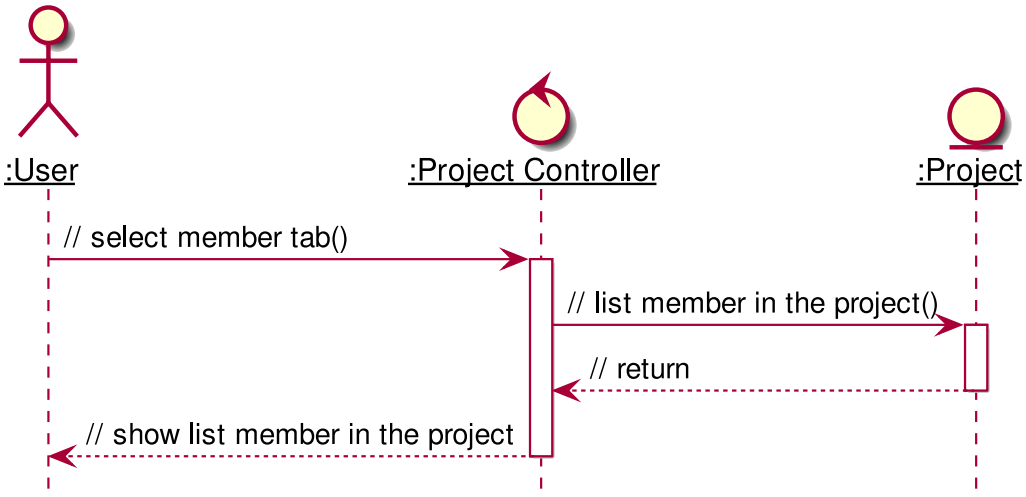


Figure 3.22: Analysis sequence diagram for the member listing process

Invite member

Since the project is initialized with only the creator, we need a function to invite members. Only who is in the project could introduce a new member.

The design involves the `projects` database table, where the *members* index is to be updated.

When user enters the invited member's name, the project controller checks the `projects` table in the database whether the person is already in the project, or that user is an assistant, or that user has not registered. If all three conditions is satisfied, two databases is updated accordingly.

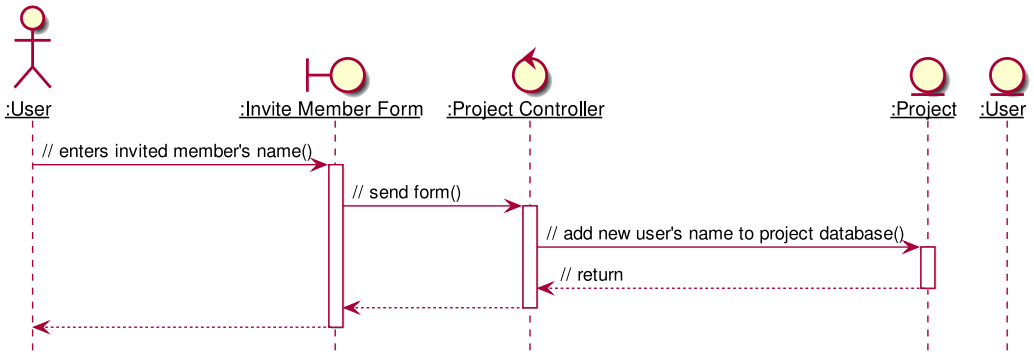


Figure 3.23: Analysis sequence diagram for successfully invite member.

If added user is already in the project

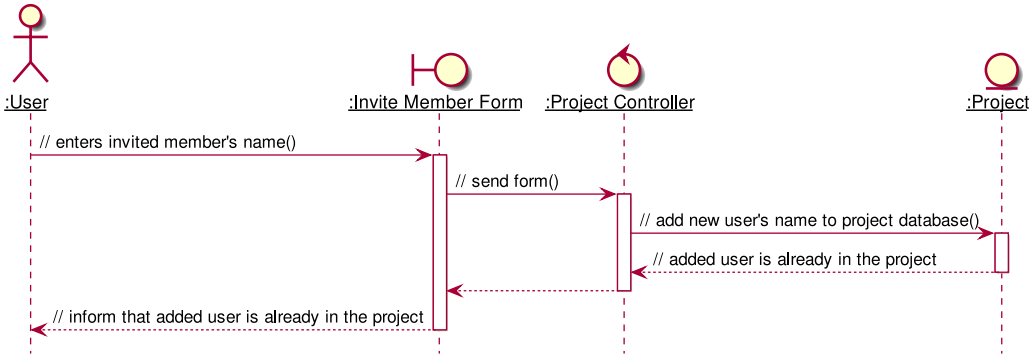


Figure 3.24: Analysis sequence diagram for adding member when user is existed in project.

If added user is an assistant

If the name is not in `users` database

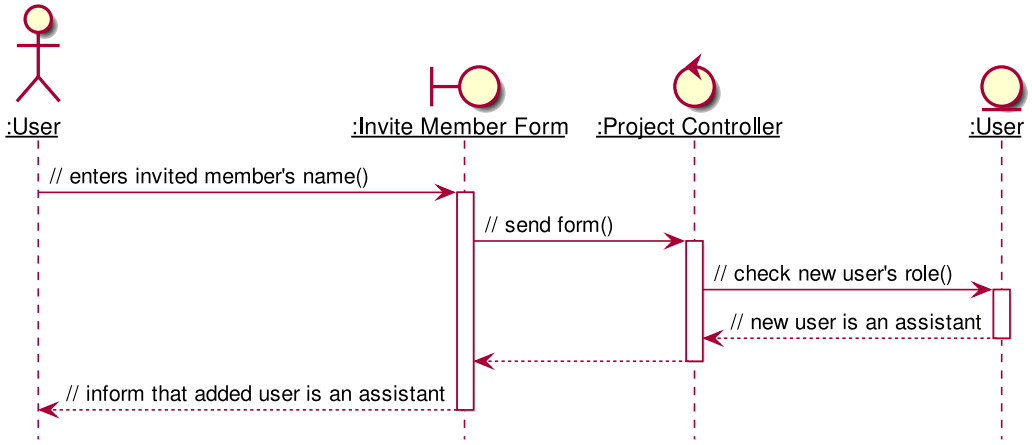


Figure 3.25: Analysis sequence diagram for adding member when user is an assistant.

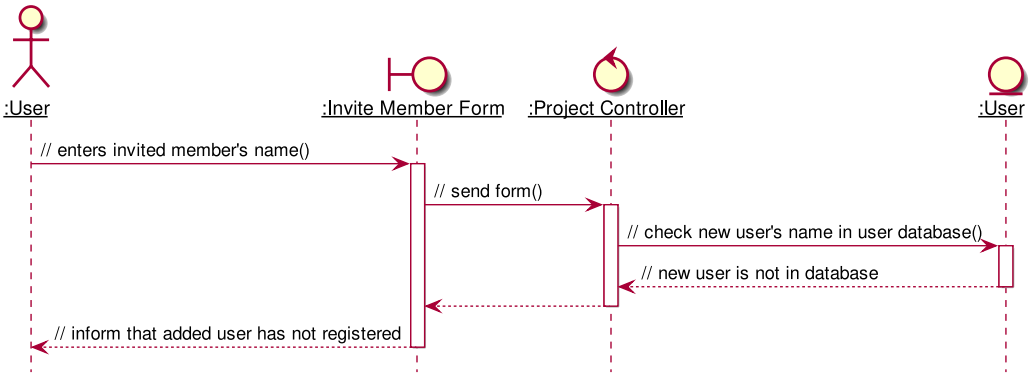


Figure 3.26: Analysis sequence diagram for adding non-registered user.

3.3.3 Artifact Upload

File uploading is necessary for report submission and slides sharing. These files are called "artifact". Only students of a project can upload these artifacts.

The process involves three storages:

- `projects` database table
- `files` database table
- IPFS

Uploading artifacts starts with user submitting a file via browser. A file controller receives it and add to the file system, IPFS. After adding the file, a CID is generated by IPFS and returned to the controller. Consequently, metadata about the file, including the CID, is added as a row in the table `files`. The database management system, RethinkDB, generates a primary key for this row, which is then added to project artifact revisions.

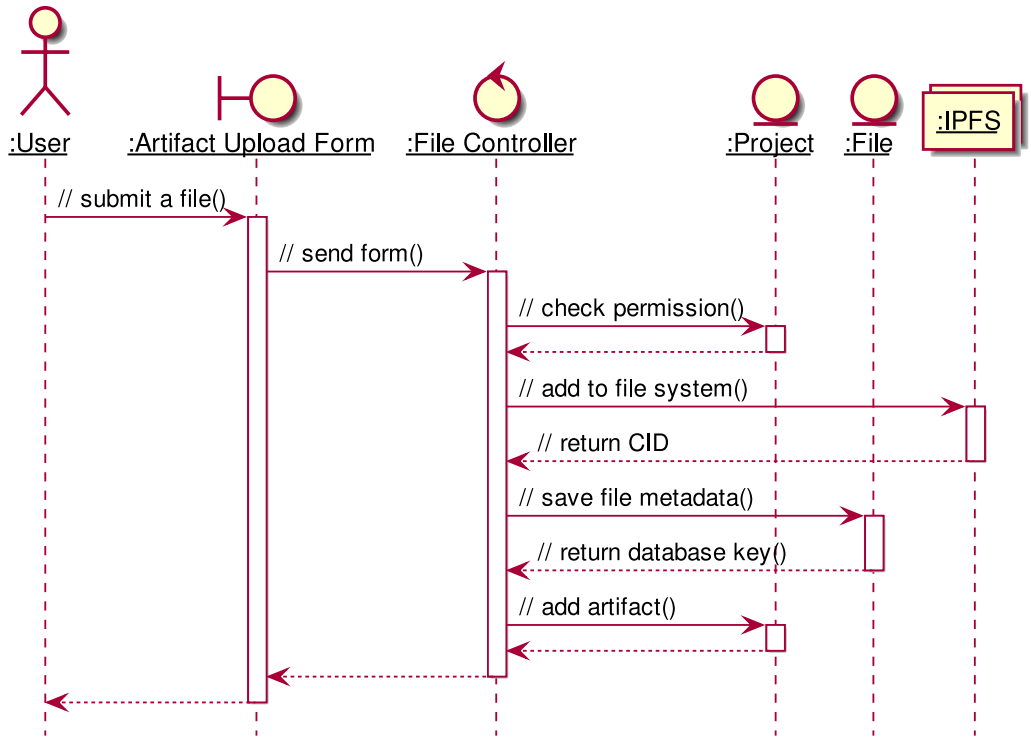


Figure 3.27: General message sequence for artifact uploading.

In case of uploading files for other purposes, such as including a picture in a comment, is similar to the flow described above. However, the primary key is not stored in the project, and the link to the stored file is returned

instead.

3.4 Database Design

This section is dedicated to designing the database for the system. Designing a non-relational database can be tricky due to its flexibility, as opposed to for traditional relational database. However, we followed a similar approach: we first analyze which entities need to be represented and from there, we derive tables for the database accordingly.

3.4.1 Entity Set

Since each user can have zero, one, or several projects and a group project can have several participating users, we model the relation between them as many-to-many.

In each project, there could be many tasks which participants would complete to advance the progress of the project. There should be thus a one-to-many relation between `Project` and `Task`, and between user and task.

For each task, we designed a comment thread, in which students can discuss their problems, or ask their supervisors for help, to resolve them faster. Since there can be many comments in a task, the relation between `Task` and `Comment` should be one-to-many.

There can be comments replying to other comments, constructing a tree data structure. This structure is described by a self-referential one-to-many relation.

Additionally, there should be a `Artifact` entity for storing reports and slides. Each project can have one report and one slide.

We allowed uploading multiple revisions of reports and slides, considering that the group can continually update their reports and presentations. Each `Artifact` entity thus can have multiple associated `File` as revisions.

However, a file does not necessarily tie to any project, since they can be referred in comments or by academic assistants in their statistical reports.

Visually, the relations between entities described above can be represented in Figure 3.28.

In each entity, there are other attributes and metadata which assists the management. They are described in the following sections.

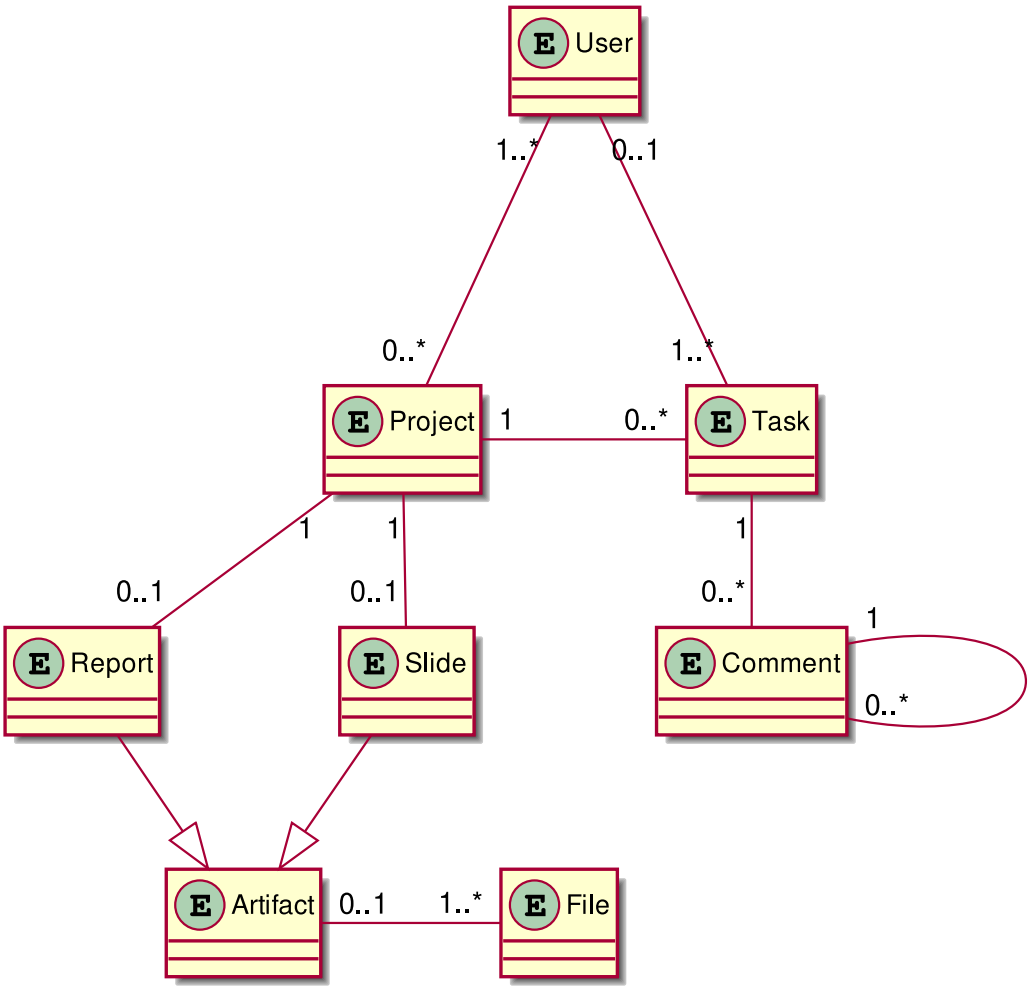


Figure 3.28: Relationship between entities in the database

User

A `User` object represents either a student, a supervisor, or an academic assistant. Since all users of this system have similar basic information, such as username and email, we represented them all as `User` object, and restrict their behaviors according to a field `role`.

Each `User` object has following attributes:

`username` [`string`] A unique name with which the user can refer to one another. It is also the primary key and allows the user to sign in.

`name` [`string`] The legal name of the user. This is required for group members to recognize each other, and for the academic assistants to collect their results.

`email` [`string`] The email address that is used to contact with the user.

`password` [`string`] The password for the user's account, encrypted with a hash function.

`role` [`string`] The role of the user in this system. It can be:

- `student`: Students who are participating in a group project or internship.
- `supervisor`: The supervisor of project(s).
- `assistant`: The academic assistant of a department.

`department` [`string`, *optional*] The department of the user, such as `ICT`, `SA`, or `LS`. Required for students and assistants.

`student-id` [`string`, *optional*] Only applicable for users with role `student`: A unique identifier assigned to students to be used outside this system.

`bio` [`object`, *optional*] A markup text describing the user. This is not necessary, but it can be helpful for a supervisor to have a biography to show their credibility in their respective field.

Project

A `Project` object includes the project description as well as the links to participants.

`name` [`string`] The name of the project.

`description` [`string`] The summary of a project, which gives outsiders a brief idea of the objective or the scope of the project.

We separated the list of the members as supervisors and students so that the view functions do not have to check for their roles all the time.

`supervisors` [`array` of `string`] Usernames of supervisors of the project.

`students` [`array` of `string`] Usernames of students participating in the

project.

As described above, a project should have several tasks, some revisions for reports and slides:

`tasks` [`array` of `Task`] List of `Task`s created for this project.

`reports` [`Artifact`] An object representing the reports of the project.

`slides` [`Artifact`] An object representing the presentation slides of the project.

Each project must be evaluated by the supervisor(s); the evaluation should therefore be stored as a property of the project.

`evaluation` [`number`] The evaluation of the work for the project that is provided by the supervisor. It can be based on previous evaluation of individual tasks.

Additionally, information about the timeframe of the project is also needed. It can be used for sorting, filtering and reminding participants of the deadline.

`created_on` [`time`] The date and time the project is created.

`deadline` [`time`] The date and time for the deadline of the project.

Task

To identify the tasks, the tasks' names and descriptions are needed.

`name` [`string`] The name of the task

`creator` [`User`] The user who created the task

`description` [`string`] The summary of a task

Since tasks should be assigned to someone to perform the tasks, there should be a field for linking to the assignee. Moreover, there should be a status to track if the task is in progress or has been done. The user can upload a report or add a link to show their work to address the task.

`assigned_to` [`User`] The assignee of the task. Must have role `student`.

`status` [`integer`] The status of the project in the Kanban board: to-do, in progress, or done.

`work` [`string` (URL)] The URL to the work by the assignee that addresses the task. It can be a link to a file hosted on the acanban server (if the user uploads it to acanban) or an external link (e.g. to a GitHub pull request, or to a document hosted by some other services)

To facilitate collaboration, there should be a discussion thread for participants to discuss the problems of the tasks:

`discussion` [`array` of `Comment`] List of `Comment`s created for this task

Like for projects, creation date and deadline are added so that partici-

pants can keep their progress. It can also help sort the tasks by timeline.
`created_on` [`time`] The date and time the task is created.
`deadline` [`time`] The date and time for the deadline of the task.

Even though individual tasks are not required to be evaluated and the evaluation does not add to the final evaluation, an evaluation field was designed for tasks so that the assignees can receive a measurable feedback on their work.
`evaluation` [`number`] The evaluation of the task

Artifact

The `Artifact` object is used for storing artifacts with multiple revisions and can be academically evaluated. The evaluation can come with a comment so that the students can improve their skills in the future.
`comment` [`string`] Evaluation comment.
`grade` [`float`] Evaluated grade.
`revisions` [`array` of `string`] UUIDs of previously uploaded revisions.

Comment

`Comment` object represent a comment insides a task's discussion. It should contain the identifier for the commenter and its content.
`creator` [`User`] The user who created the comment.
`content` [`string`] The content of the comment.

Since the discussion follows a forest structure, each comment recursively contains a list of replying comment.
`comments` [`array` of `Comment`] List of `Comment`s replying to it.

To let the user know if the comment is new or old, creation time is also added.
`created_on` [`time`] The date and time the comment is created.

File

The `File` object is needed to store the metadata about the files used in a project. The object indicates the IPFS link to the file, besides other metadata such as uploader, uploaded time, file name, ...
`id` [`string`] UUID unique to the upload.
`cid` [`string`] Content identifier of the file in CID v1.
`name` [`string`] Name of the file.

`size` [`integer`] Size of the file, in bytes.
`time` [`datetime`] The time when the file was uploaded.
`user` [`string`] Username of the uploader.

3.4.2 Data Tables

From the relationship between data tables we analyzed above, we derive three data tables: `users`, `projects`, `files`.

The `User` entity set is mapped to `users` table. Likewise, `File` is mapped to `files` table.

table	User
department	ICT
email	adaml@example.edu
name	Adam Lahtinen
password	(hashed password)
role	student
student-id	BI9-xxx
username	adaml

Figure 3.29: An example database row for student.[1]

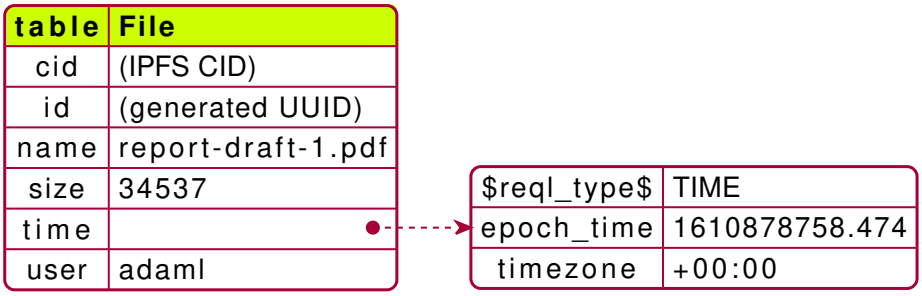


Figure 3.30: The structure of database table `file`.[1]

However, most other entities are stored within a single table `projects`, forming a typical tree-like structure of document-oriented database. For example, two entities `Artifacts` storing revisions for report and slides are included within a table row. `Tasks` are stored in an array in the project.

[3] The JSON structure is visualized by PlantUML. Source text used can be found at https://github.com/Huy-Ngo/acanban/tree/main/docs/source/meth/database/tables/images

The many-to-many relation between `User` and `Project` is realized using foreign keys with secondary index in this table as well:

- Each row in `projects` has a `supervisors` and a `students` fields. We model both of these fields as lists, since there can be occasions where two or more supervisors co-supervise a project.
- A secondary index named `members` was created for these fields for the ease of querying projects a certain user participates in.

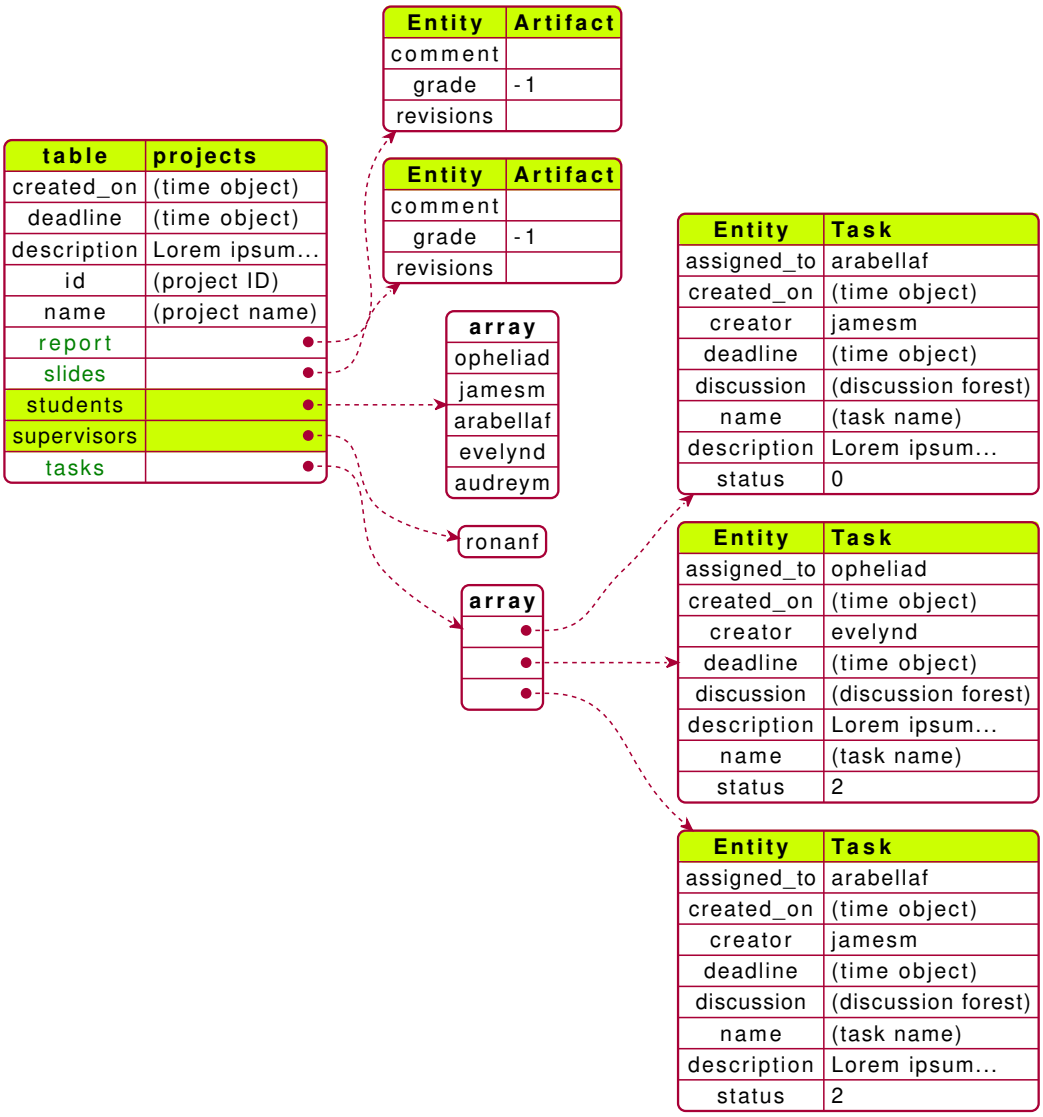


Figure 3.31: The structure of database table `projects`.[1]

Within each task is a discussion thread with a forest structure of `Comment`

entities. There are several root comments within the discussions. Each comment can contain some child comments.

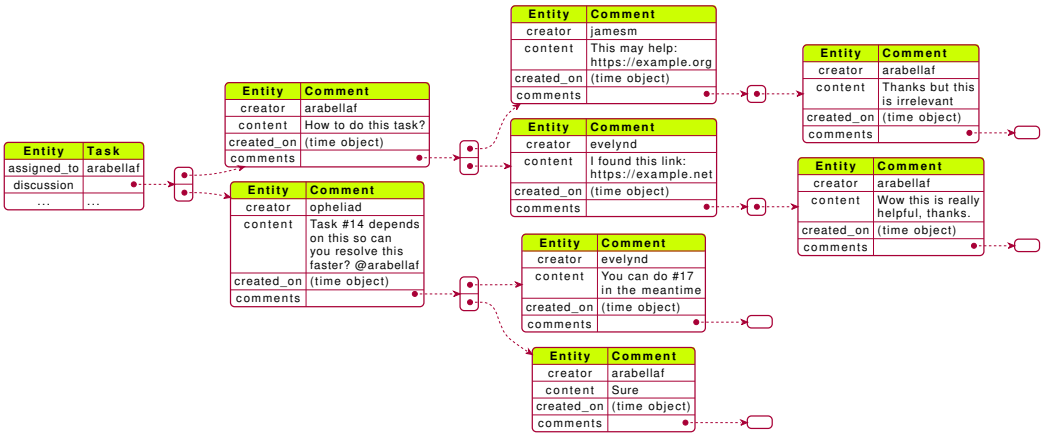


Figure 3.32: The forest structure of `discussion` field.[1]

3.5 User Interface and User Experience

This section is dedicated to the design of user interface and user experience, in order to support the goal of achieving *Constructing an Applicable Web Application*. Generally, the application should be portable across different input/output devices, as well as being easy to adopt by the end-users.

3.5.1 Navigation

For the ease of navigation, the organization of web pages must be well-structured. Furthermore, it is important to avoid having deadends, which negatively impact the traversability. With these principles im mind, we then designed the navigation graph. For every user, the following auxiliary endpoints are available, as illustrated in Figure 3.33.

- `/register`: Registration form
- `/login`: Login form
- `/u/<username>`: User information, including per project
- `/u/<username>/edit`: Form for updating user information
- `/`: User dashboard containing the list of projects perse participates in
- `/p`: List of public projects
- `/p/create`: Form for project creation
- `/p/<uuid>`: Project's page

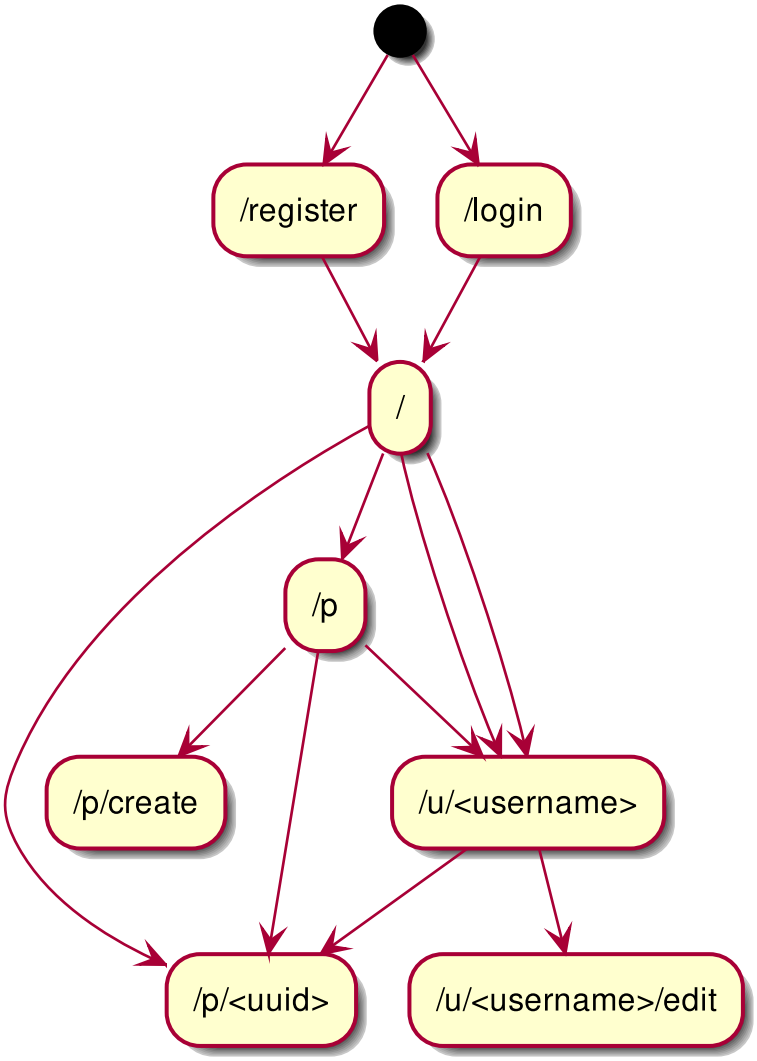


Figure 3.33: Auxiliary endpoints

Each project's page is divided into several tabs, namely `info`, `edit`, `members`, `tasks`, `report` and `slides`, as shown in Figure 3.34.

- `/p/<uuid>/info` (GET): Project's basic information
- `/p/<uuid>/edit` (GET and POST): Form for updating project's basic information
- `/p/<uuid>/members` (GET): Project's member listing
- `/p/<uuid>/invite` (POST): Form for adding a member
- `/p/<uuid>/leave` (POST): Form for leaving the project
- `/p/<uuid>/tasks` (GET): Tasks overview (Kanban board)
- `/p/<uuid>/report` (GET): Report revisions and evaluation
- `/p/<uuid>/report/upload` (POST): Form for uploading report revision
- `/p/<uuid>/report/eval` (POST): Form for evaluating report
- `/p/<uuid>/slides` (GET): Slides revisions and evaluation
- `/p/<uuid>/slides/upload` (POST): Form for uploading slides revision
- `/p/<uuid>/slides/eval` (POST): Form for evaluating report
- `/ipfs/<cid>` (GET): IPFS gateway proxy for file downloading

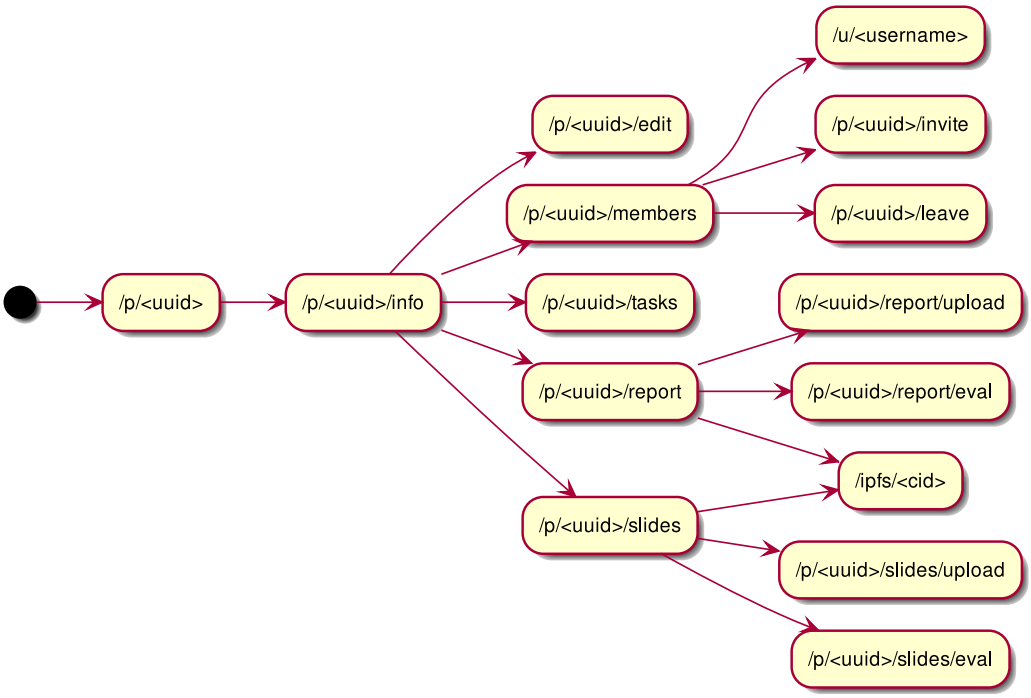


Figure 3.34: Project's endpoints

Project's tabs are mutually interlinked but for brevity they are not con-

nected in the figure. Additionally, POST-only endpoints redirects back to referrer upon success.

Due to complexity, task-related endpoints are documented separately in Figure 3.35, which consists of the ones listed below. Pages in `/p/<uuid>/tasks`, including the Kanban board, exclusively serves *Building a Collaboration Platform*.

- `/p/<uuid>/tasks/<index>` (GET): Task's description and discussion
- `/p/<uuid>/tasks/<index>/comment` (POST): Form for posting a comment
- `/p/<uuid>/tasks/<index>/upload` (POST): Form for uploading a file
- `/p/<uuid>/tasks/eval` (POST): Form for evaluating all tasks

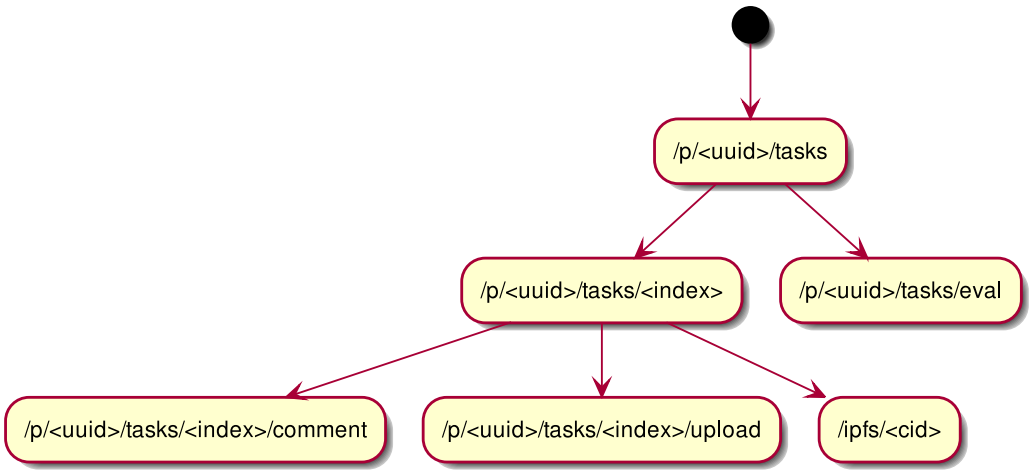


Figure 3.35: Task endpoints

3.5.2 Graphical User Interface

At the higher level, we concentrated in making user interface that works well for mobile devices. Mobile first design might not always make the best use of screen estate on larger devices, however we were confident that such trade-off is worth the high portability and compability that we can bring to the wide range of users. That being said, we actively tried to compensate through responsive design. One case in point is the Kanban board, which is rendered as a single column of different colors on handheld devices, while expands to multiple columns (as often seen traditionally) on a wider monitor.

We also strongly focused on accessibility. The colors of relatively high contrast and appropriate line length were chosen. We also made use of markup

elements in the standardized manner with the hope of providing compability for less common web browsers and supporting software for visually impaired people.

3.6 Development Process

From requirement, analysis and design documents to implementation were version controlled through git. For achieving better productivity and quality, we attempted to define a rigorous yet flexible workflow, detailed in the following subsections.

3.6.1 Collaboration Model

The development mostly took place on GitHub, except for realtime communication which occured either on a Matrix chat room or in the physical world.

At the high level, the works were split into multiple subprojects[4], where each project contained multiple self-contained tasks, which were to be resolved by patches separated by logical changes [7]. Projects are visually presented as Kanban boards including at the minimum of three columns: *To do*, *In progress* and *Done*.

In the time dimension, we divided the development period into multiple short iterations, whose length varied based on our schedule[5]. At the beginning of every iteration, we selected tasks from the *To do* column to *In progress*. In addition to inital planning, tasks were also added to *To do* as we discovered new issues during discussions or development, and were democratically and appropriately assigned to the group members.

At a lower level, each task was resolved via a self-contained patch. This means implementation patches must be accompanied by the tests coverring the change. How patches were checked (including executing automated tests) are detailed in the next subsection.

At the end of an iteration, we publish a (pre-)release to PyPI and deploy it to a test deployment server[6] kindly provided by USTH ICTLab.

[4] https://github.com/Huy-Ngo/acanban/projects

[5] Unfortunate for us, the group project took place during the examination season, and thus our time pool shrunk on weeks with higher density of exams for other courses.

[6] https://acanban.ga

3.6.2 Quality Assurance

In order to ensure the correctness of the implementation, we tried to take quality assurance as seriously as we can. To begin with, a git branch was chosen to be the `main` one for patches to base on. It was protected from being pushed directly onto without the reviewing process, which comprised of automated checks and peer reviews.

From the beginning, continuous integration (CI) was set up for both the implementation and this report itself[7]. For the paper, it simply tried to compile the original source written in reStructuredText into PDF for the ease of viewing for the peer reviewers. The software, however, were subjected to the following assertions:

- Style checker (flake8 and isort), which statically analysed the Python source files for errors and inconsistencies
- Type checker (mypy), which examined the Python AST for static typing issues to detect common bugs
- Testing and coverage, which automatically ran the tests and report the test coverage

We actively worked to enforce 100% *branch coverage*, most of which are covered by unit tests, which helped discover mistakes as well as regressions in later modifications. Integration testing, however, could often be tedious [8], and thus we chose to not imposing it. In compensation, we examined the test deployment at the end of iterations for bugs not having been caught by the test suite.

At the same time, patches were reviewed manually by at least one other team member. This is not only for maintaining quality standards but also to make the team more well aware of changes happening to the shared code base. Once approval was granted and automated checks were passing, the patch was rebase on top of the `main` branch. Continuous integration was then run again, for continuous integration.

[7] https://builds.sr.ht/~huyngo/acanban

4 Results and Discussion

4.1 Results

In this project, we clarified the requirements for the software, defined the system architecture, and drafted some analysis and design for the required use cases, based on which we have successfully implemented:

- Authentication
- Create and edit group projects
- Invite new members to a project
- Upload reports and slides
- Supervisors can evaluate students' work

All of these use cases have been implemented in a simplistic manner, which satisfied our goal of accessibility from all devices. The web pages can load with extremely low latency thanks to minimal assets: Our home page is only 21.8KB, and it only takes less than 0.1 seconds to load.

The current implemented source code is published on GitHub[8] under Affero General Public License. It is deployed on USTH ICTLab's server[9].

4.2 Discussion

We have achieved almost all our goals, as proven by our results above. Unfortunately, due to some difficulties that will be explained below, we failed to design and implement important some collaboration-related use cases as well as academic integration use cases. We will discuss our flaws and difficulties in this section.

4.2.1 System's flaws

- Important features that have not been implemented:
  - Notification
  - Create, view, and complete tasks on a simple Kanban board
  - Discussion
- Actors whose functionality have not been implemented: judges and assistants
- Registration currently allows anyone to have as many accounts as they want, which is not secure.

[8] https://github.com/Huy-Ngo/acanban
[9] https://acanban.ga

- There has not been checking for maximum grade and minimum grade fraction.
- File types for reports and slides are not ensured yet.
- There has not been a graphical user interface for administration.
- The user interface is not very attractive.

4.2.2 Difficulties

During the project, we met several difficulties in different aspects.

The primary obstacle for us was the lack of time spent for the project. This lack of time dedicated to the project was because of poor time management. Moreover, there had been other time-consuming projects and labworks in other courses, which make the management harder.

Another problem was the collaboration: There were several conflicts among some members' expectations of the project and workflows. People tended to work at different time in day, which made code reviews take a longer time than it should. Some members were not too keen on the project and did not spend enough time for it.

There were also some technical struggles: Not everyone in the group was familiar with web development in Python, let alone the framework. This revealed another issue, which is communication – some members did not reach out to others, which made it hard for the others to help.

As our project aimed to develop a project collaboration platform, we can analyze these difficulties to add features that can mitigate them.

5 Conclusion and Future Work

5.1 Conclusion

We accomplished two out of our three main objectives: developing an accessible website and create a collaborative platform. However, there are rooms for improvement in the implementation.

Through rigorous *Development Process*, we were able to not only maintain tight collaboration and desirable work quality, but also have better control over regressions and gain confidence in rolling out new changes, and improve productivity and performance throughout the three-month period. We wish to continue ameliorating the process when working on the remaining tasks of this project as well as in future software development.

The resulted system runs as a web service, allowing clients to run on any

platforms without installation.

Thanks to the simplistic user interface, it is accessible and easy to use.

5.2 Future Work

In the future, we will realize features that have not been implemented:

- Task management
- Jury's role and evaluation
- Notification
- Task discussion
- Statistical reports for academic assistant

On top of that, we need to refine existing features:

- Implement a method that prevents user.
- Add configuration for maximum and minimum grade.
- Check file types for reports and slides.
- Improve user interface and user experience.

A Acknowledgement

We would like to express our gratitude to Dr. Đoàn Nhật Quang for providing us instruction and feedback during his supervision. We would also like to extend our sincere thanks to Dr. Trần Giang Sơn for his support with USTH ICTLab's server that we used for testing our deployment.

We would also like to thank Philip Jones, the author and maintainer of the framework Quart, as well as the maintainers of RethinkDB for their timely technical support and bug fixes. The following organizations are also appreciated for providing their services free of charge:

- Matrix.org (instant messaging server)
- GitHub (code hosting and collaboration)
- SourceHut (automated CI/CD)
- PlantUML web server (diagrams generation)

B Glossary

Accessibility Accessibility means that the technology is developed so that people with disabilities can use them. It also benefits people without disabilities, such as people using devices with small screens or using slow Internet connection.

In this document, *accessibility* usually refers to the latter sense.

Asynchronous Server Gateway Interface (ASGI) ASGI is a spiritual successor to **WSGI**, the long-standing Python standard for compatibility between web servers, frameworks, and applications.

WSGI succeeded in allowing much more freedom and innovation in the Python web space, and ASGI's goal is to continue this onward into the land of asynchronous Python.

Branch coverage A coverage criteria where each control structure's branch has been executed by the test suite.

Cluster A set of machines that are connected and work together to be viewed as a single system.

Cross-site scripting (XSS) attack Cross-site scripting (XSS) is a code injection attack where the attacker inserts client-side scripts into web pages.

Distributed Denial of Service (DDoS) attack Denial of Service (DoS) attack is an attack where the attacker overloads the system with requests. DDoS is the distributed DoS attack, that is, the requests are flooding from multiple sources.

Hypertext Transfer Protocol (HTTP) An application-level protocol for distributed, collaborative, hypermedia information systems[10].

InterPlanetary File System (IPFS) IPFS is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. It uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices.

Metadata Data that describes the information rather than containing the information itself. This can include primary key of a table, creation date, etc.

Participant People who participate in a project.

Request A *HTTP* message from a client to a server including the protocol version in use, the method to be applied to and the identifier of the resource[11].

[10] **RFC 2616#section-1**
[11] **RFC 2616#section-5**

Response A *HTTP* messge a server responds with, after receiving and interpreting a *request* message[12].

SQL injection SQL injection is an attack where a malicious piece of code is passed in an SQL query, which changes the query to the query the attacker wants.

For example, the attacker may try to create a student with name `John' DROP TABLE Students; --` and delete the whole table in doing so, if the vulnerability is not handled.

Supervisor A person, normally a lecturer, that supervises, supports, and evaluate one or several project groups.

[12] **RFC 2616#section-6**

C References

[1] Marian H.H. Willeke, “Agile in Academics: Applying Agile to Instructional Design”. *2011 Agile Conference*, p. 246–251, Salt Lake City, UT, 2011. doi: 10.1109/AGILE.2011.17.

[2] Tommi Mikkonen and Antero Taivalsaari. “Web Applications—Spaghetti Code for the 21st Century”. *2008 Sixth International Conference on Software Engineering Research, Management and Applications*, p. 319–328, Prague, 2008. doi: 10.1109/SERA.2008.16.

[3] Richard Stallman. “Why Schools Should Exclusively Use Free Software”. *Free Software and Education*. GNU Project. Retrieved 2021-02-01. https://www.gnu.org/education/edu-schools.html

[4] Douglas Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. 20th anniversary ed., 1999, p. 152. ISBN 0-465-02656-7.

[5] The Pallets Project. “Template Designer Documentation”. *Jinja Documentation (2.11.x)*. Retrieved 2021-01-18. https://jinja.palletsprojects.com/en/2.11.x/templates/#html-escaping

[6] RethinkDB. “Introduction to ReQL”. Retrieved 2021-01-18. https://rethinkdb.com/docs/introduction-to-reql/#reql-embeds-into-your-programming-language

[7] The kernel development community. *Submitting patches: the essential guide to getting your code into the kernel*. The Linux Kernel documentation. Retrieved 2021-02-04. https://www.kernel.org/doc/html/latest/process/submitting-patches.html#separate-your-changes

[8] Michael Steindl and Juergen Mottok. “Optimizing software integration by considering integration test complexity and test effort”. *Proceedings of the 10th International Workshop on Intelligent Solutions in Embedded Systems*, Klagenfurt, 2012, p. 63-68.