***Conference Management System***

**Team:** Crooked Raven

**Members:**

Mierlă Cristina

Mihance Luca

Mihoc Adelin

Moldovan Andrei

Moldovan Ioana (Team Leader)

**Contents**

**1. Introduction**

**1.1 Purpose**

**1.2 Intended audience**

**2. Proposed System**

**2.1 Overview**

**2.2 Used technologies**

**2.3 Functional Requirements**

**2.4 Nonfunctional Requirements**

**2.5 System models**

**2.5.1 Scenarios**

**2.5.2 Use Case model**

**2.5.3 Data model**

**2.5.4 Database schema**

**2.5.5 Sequence diagrams**

**2.6 Workflow**

**1. Introduction**

**1.1 Purpose**

The purpose of this web application is to help its users organize conferences in an easy manner. Also, people from the science world can find here any topic of interest and attend a presentation related to it or share their research with the public.

**1.2 Intended audience**

This project serves as a tool for people who want to organize a conference. Also, it is very helpful for anyone interested in attending or publishing a research paper in science related areas.

**2. Proposed System**

**2.1 Overview**

The Conference management systems allows users to manage, organize and participate in their own scientific conference. In other words, this system stores every information about any given conference; its participants with their unique roles, the proposed papers, and their respective reviews. Each conference is split up in five phases: registering as an author or listener, submitting an abstract/paper as an author, bidding as a pc member to let the chairs know if you are willing to review any papers, reviewing the chosen papers as a chair or pc member and presenting your work as an author or overseeing the flow of a session as a session chair. The listeners will be able to specify the sessions they wish to attend, but it is not mandatory.

**2.2 Used technologies**

Database

* MySql Server
* H2 Database

Data persistence

* Spring Data JPA
* Hibernate

Backend

* + - Java
    - Spring Boot

Testing

* Mockito
* Postman

Build tools and package managers

* Maven

Frontend

* Html
* Css
* JavaScript
* jQuery

Diagrams

* Star UML
* Visual Paradigm
* Diagrams.net

Version control system

* GitHub

**2.3 Functional Requirements**

* All people have to create an account and sign up onto our website
* A steering committee member must contact the admin of the website to create a conference
* The steering committee creates the phases of the conference and sets their deadlines, also starting every phase
* A person registers as an author at a conference and submits an abstract or a paper
* The program committee members bid to let the chairs know if they are willing to review any papers
* Chairs of a conference choose reviewers for each paper
* The program committee members leave reviews for the papers they have been assigned to
* The papers are automatically evaluated based on their reviews and in case of conflicts the chairs intervene in order to accept or decline a paper
* Sessions are created with an assigned room, session chair and a list of speakers
* If the paper is accepted, it can be presented by a speaker during a session
* Each participant of the conference who is not a member of any committee must pay a registration fee

**2.4 Nonfunctional Requirements**

* The website’s load time should not be more than two seconds for users
* The website’s interface must be user-friendly and easy to use
* The users shall be notified of any operations which have assigned new responsibilities to them or any operations which have removed some of the old ones, with respect to the conferences they participate in
* All monetary amounts must be accurate to two decimal places
* Using the third normal form for the database schema design approach
* Database queries must be efficient, timeout after 200ms
* To ensure data integrity in a multi-user multi-threaded environment database transactions must be carefully handled, keeping in mind the appropriate isolation level, and locking mechanism
* Security being of the utmost importance, the access of users to resources and the operations they can perform on these is restricted based on the roles a user holds for a given conference, or a given paper
* To ensure system wide application and network security, the program was fortified against the top 10 OWASP most common web vulnerabilities
* Test driven development
* Documenting the endpoints

**2.5 System models**

**2.5.1 Scenarios**

**Conference**

Each conference is split up into five phases: registering to the conference, submitting an abstract or paper, bidding, reviewing of the papers, and presenting/attending at the established sessions

**Submitting**

When someone wants to submit a reference paper to the conference he needs to create an account and add it. Before the submitting phase ends, he can edit the paper, change the keywords, or adjust its title.

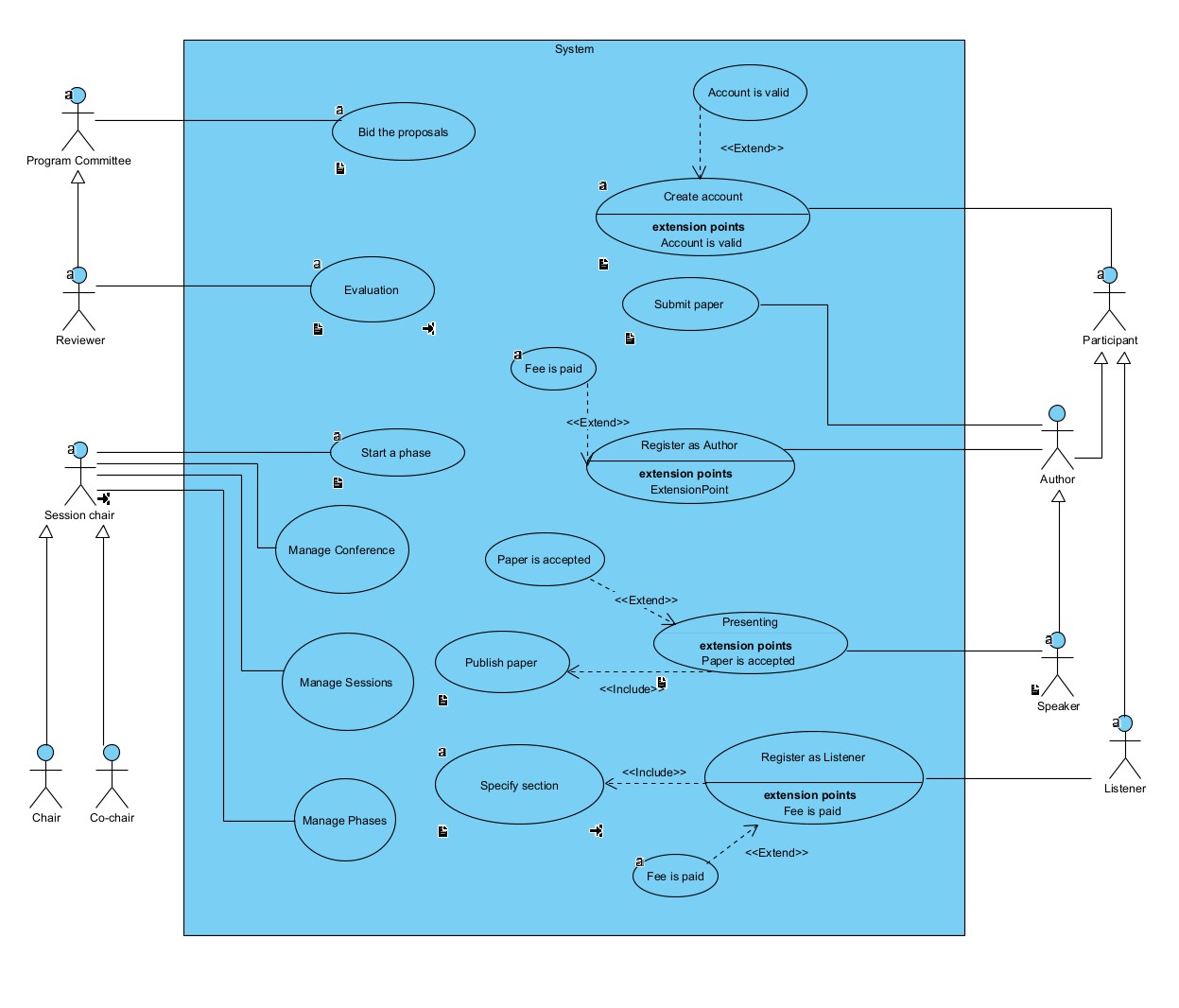
**Bidding**

In this phase the program committee members specify if they wish to review any papers and are assigned by the chairs of the conference to review some of the papers. After that, they become reviewers must review the papers they have been assigned to.

**Reviewing**

Each paper gets an evaluation from “Strong Reject” to “Strong Accept”. Reviewers are also invited to leave a feedback for the authors. After all papers are reviewed, their status is automatically calculated. In case of conflicting reviews, the chairs must intervene and manually accept or decline the papers. All accepted papers can be improved, and a speaker can present it at the conference.

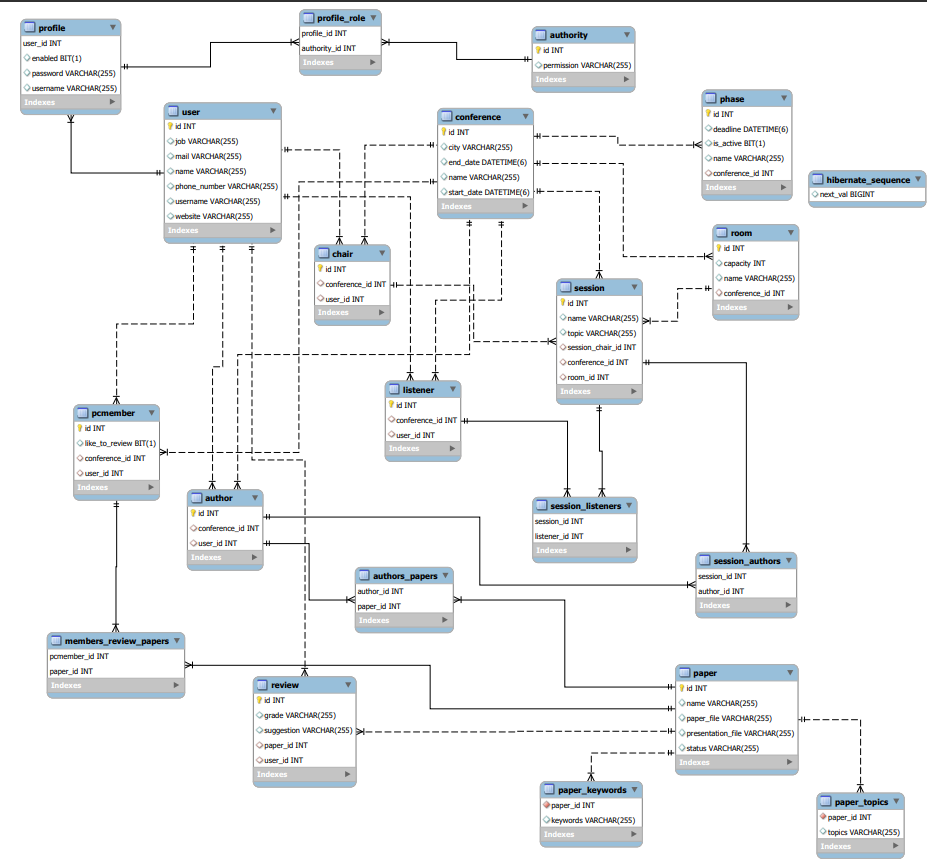
**2.5.2 Use Case model**



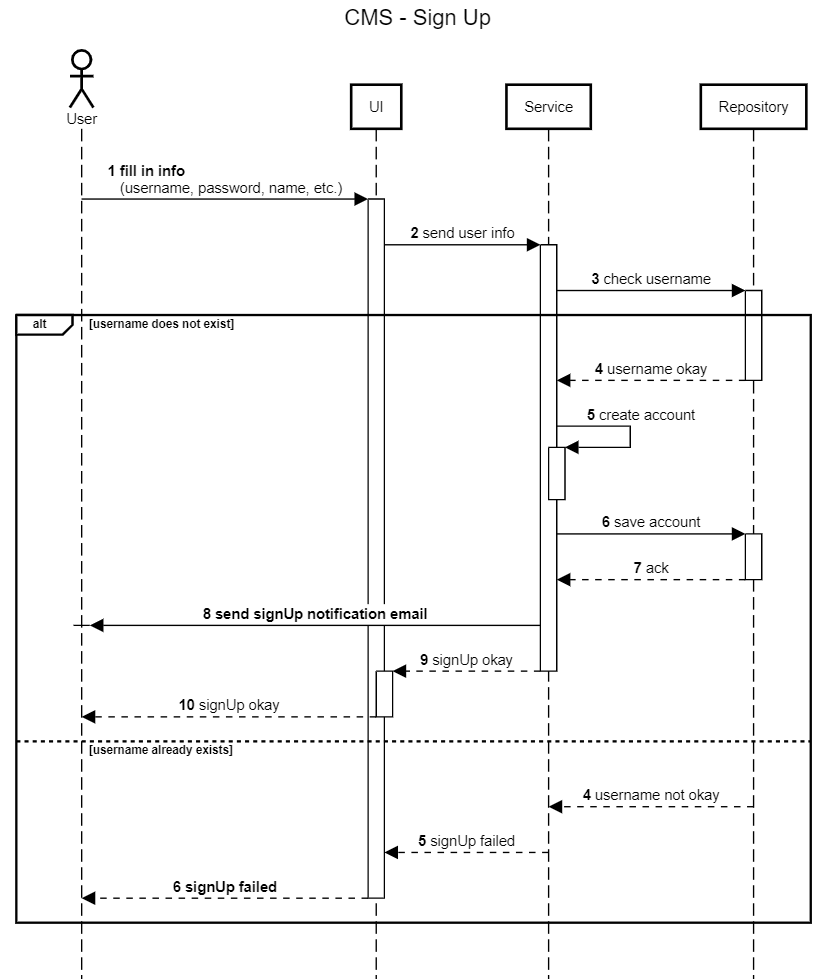
**2.5.3DataModelDiagram, engineering drawing

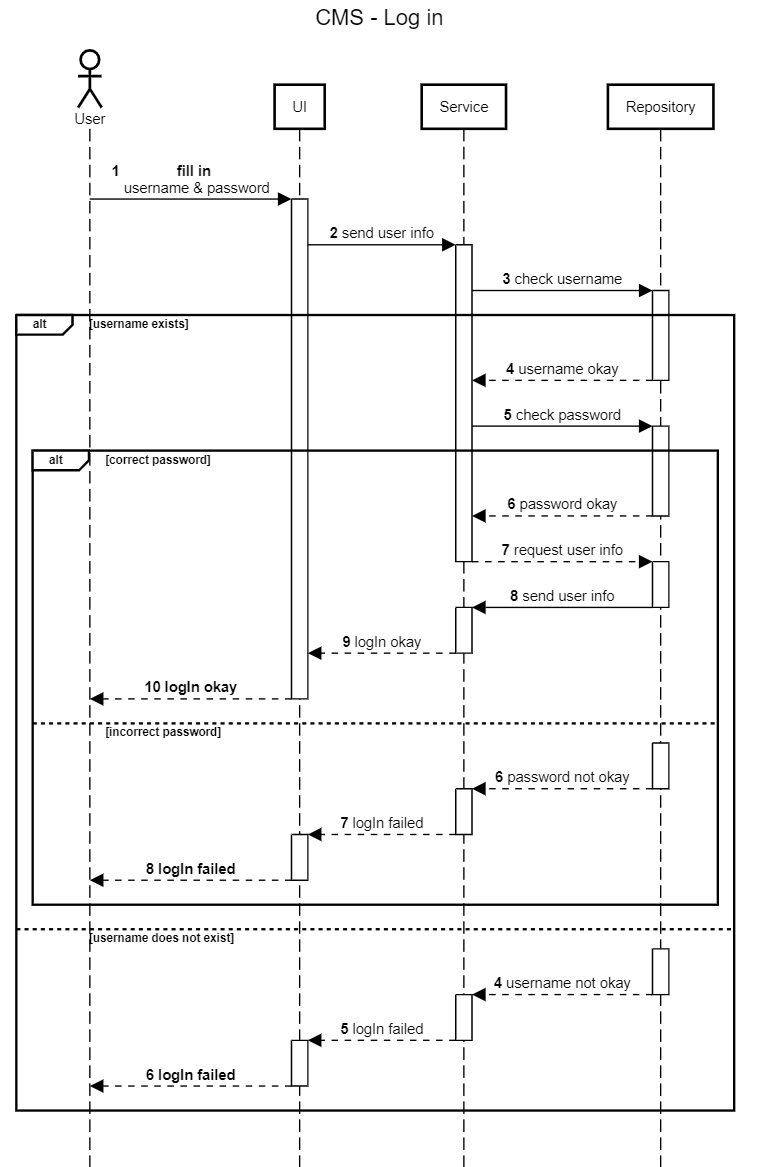
Description automatically generated**

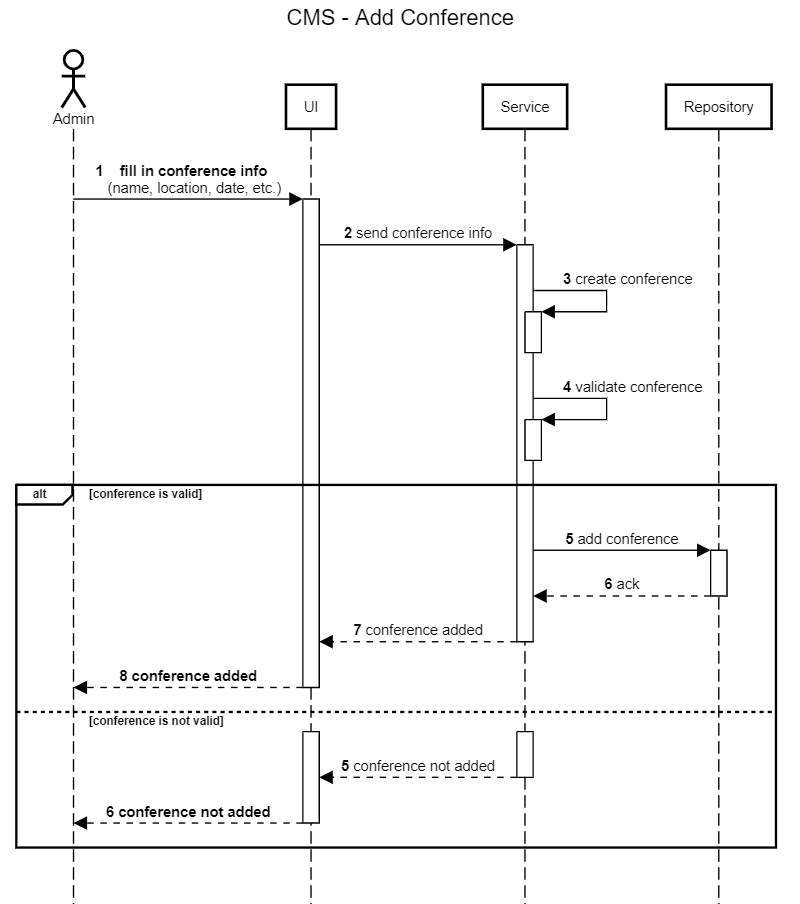
**2.5.4 Database schema**

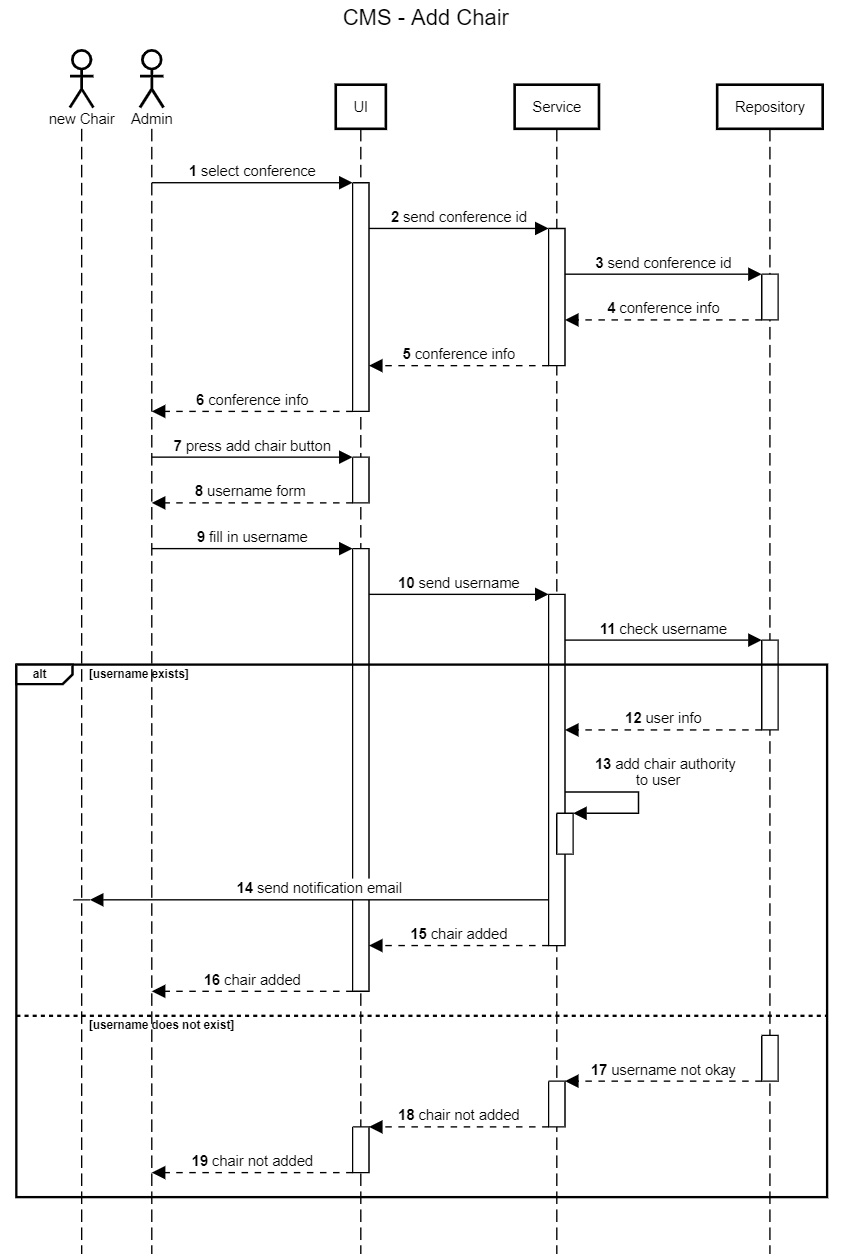
****

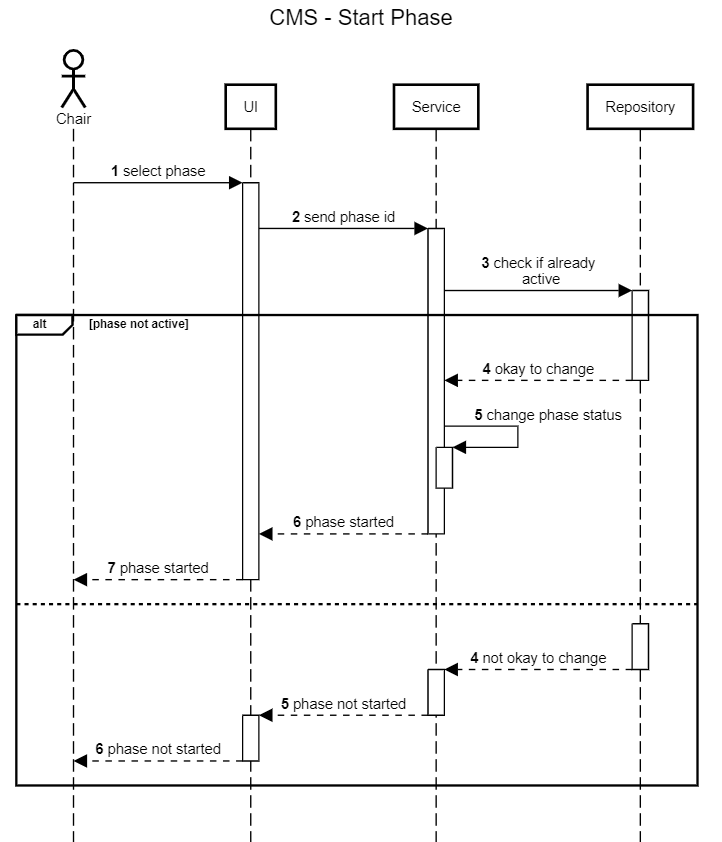
**2.5.5 Sequence diagrams**

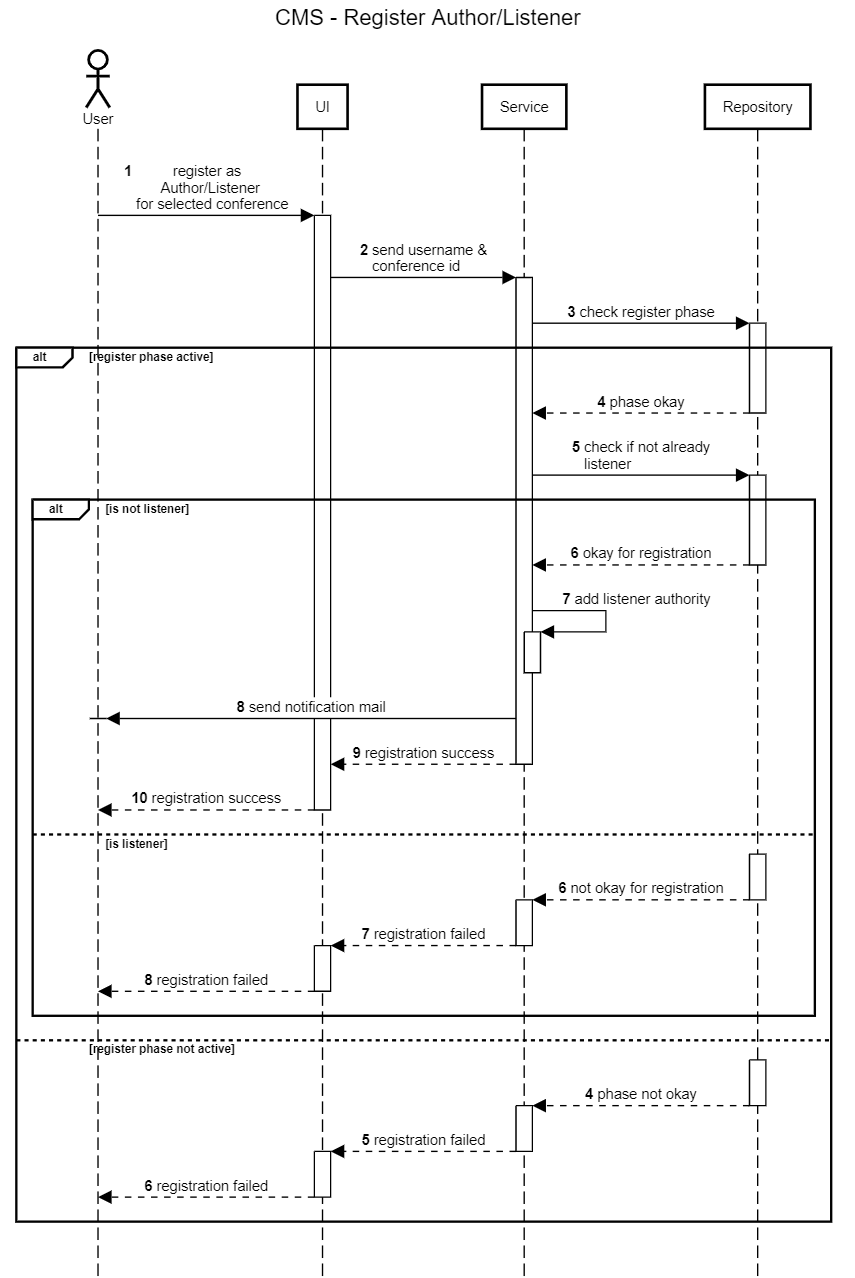


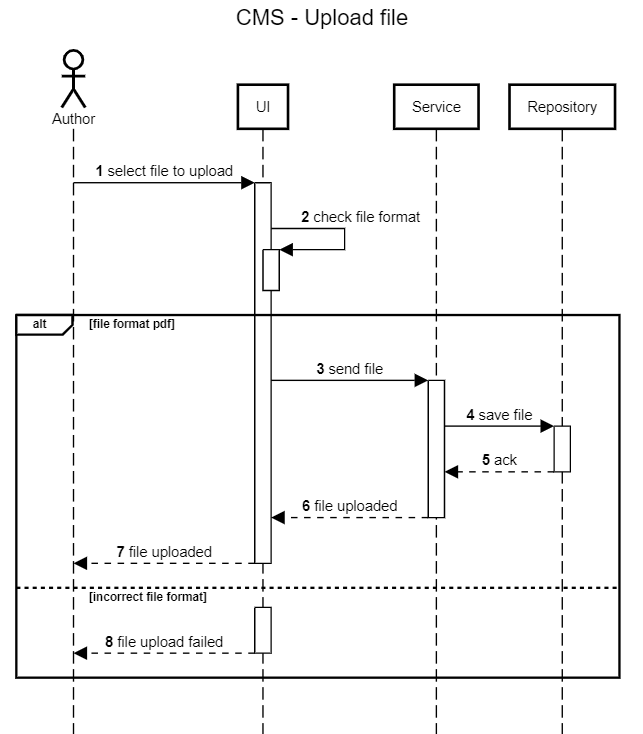


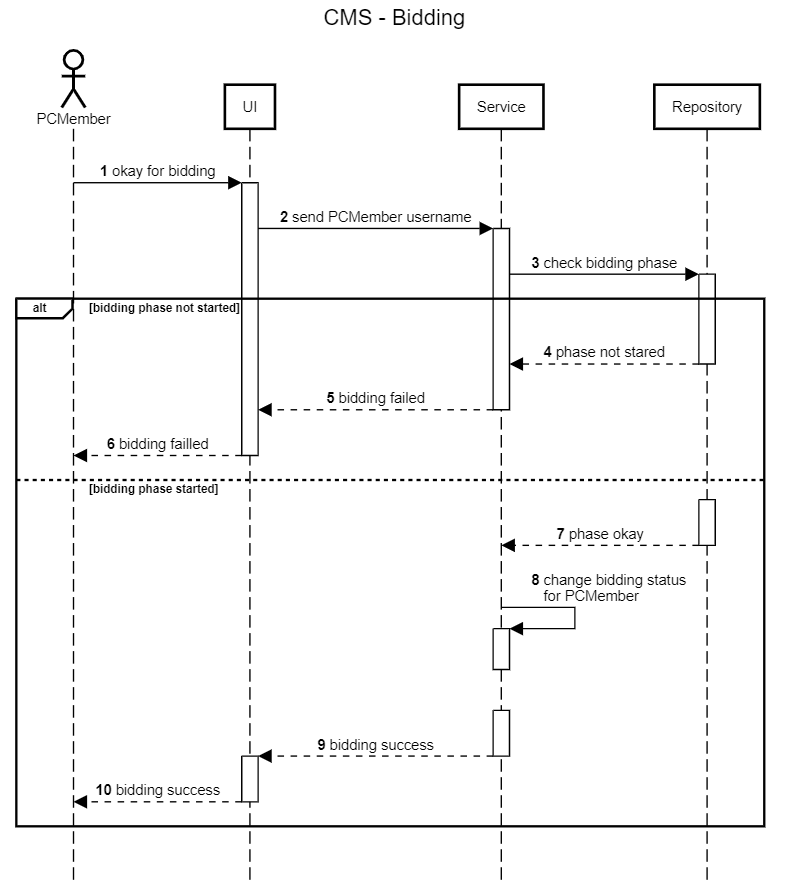


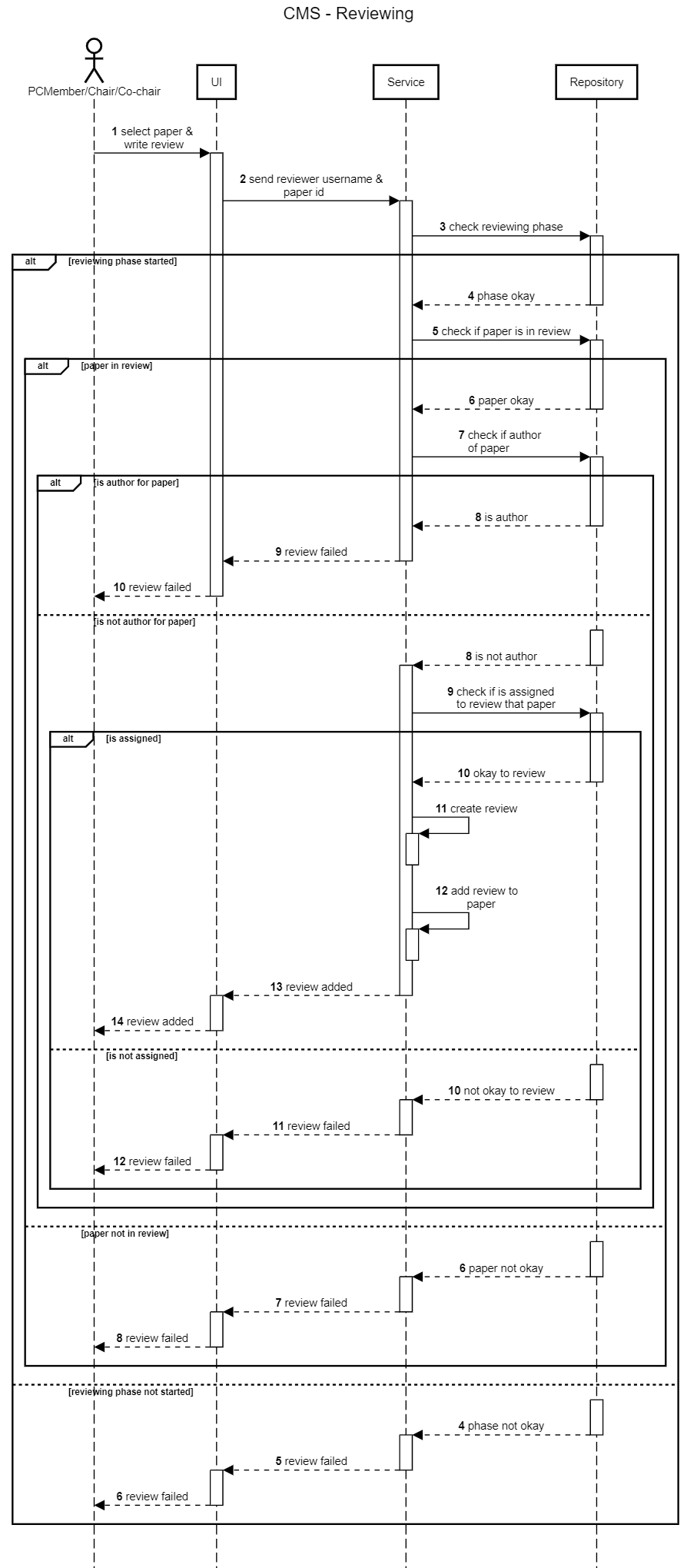


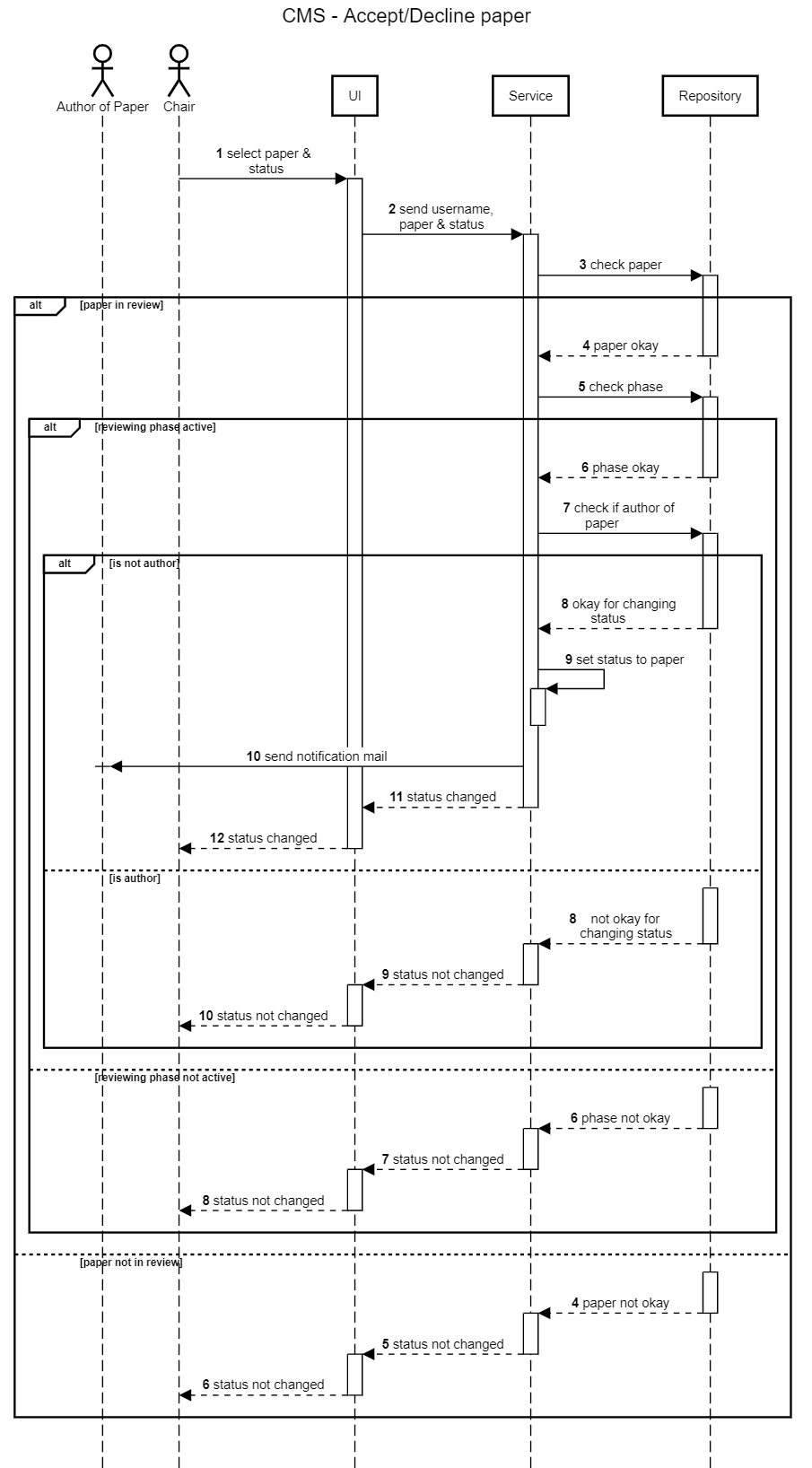


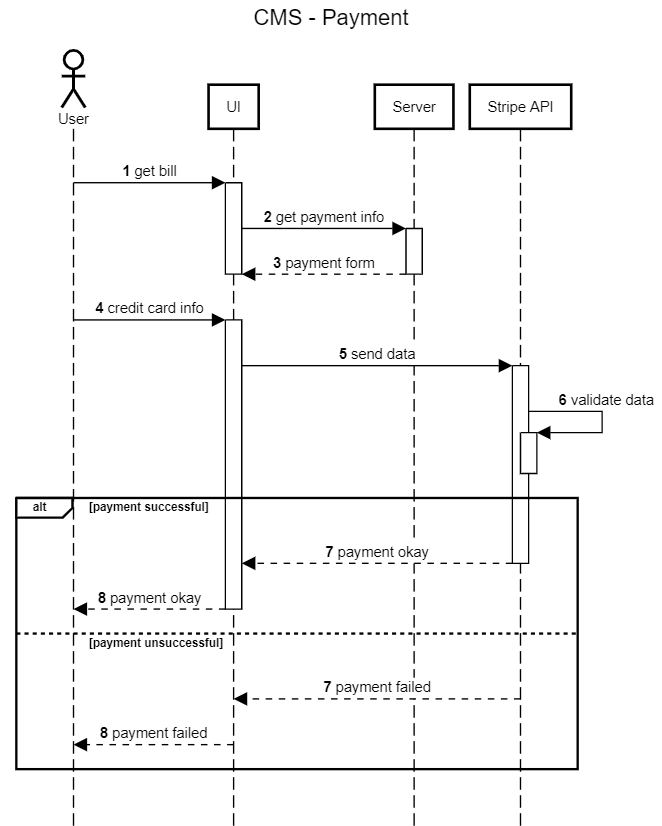












**2.6 Workflow**

* + - **Subteams, planning and communication**

The first step towards implementing this application was deciding upon a clear separation of responsibilities based on different parts of the system. The natural choice was dividing the team into 2 subteams, one concerned with the backend which later would become a REST api, and one concerned with the frontend. Then we drew the first draft of the timeline, trying to allocate a time slice for different parts of the system such as the data access layer, the endpoints, the security context or the actual components the user would be interacting with. We had weekly meetings in order to keep each other updated with the progress of certain components, to create and assign more tasks or to review and maybe redraft the timeline. This took place over Microsoft Teams and GitHub

* + - **Development**
      * + **Backend**
* Each sub-team had full autonomy within itself but would have to coordinate with the other based upon which features of the application were next in line to be implemented. For the backend we started with the data model, which we had to modify quite a few times. After reaching an appropriate data model we proceeded with modelling the database using ORM technologies such as Spring Data JPA and Hibernate. Before proceeding with the endpoints, we had to implement a proper authentication and authorization system, based on roles and authorities, which were required due to the way the application was supposed to be structured. We used JSON Web Tokens for this, making the server stateless, as per RESTful characteristics. Then we started having regular meetings with the frontend sub-team to choose what endpoints should be implemented next and discussing the details related to those.
* After finishing the main features of the app, we refactored the code to make it cleaner and increase reusability, we added mail notification and integrated payment systems to improve the user experience, and focused on increasing the performance and decreasing the wait-time for our database queries by configuring isolation levels and lock mechanisms for transactions and providing custom implementations for data fetching.
  + - * + **Frontend**

First, we started by creating templates to decide how the app should look and feel. After deciding upon one we proceeded with the html pages, creating the layouts, and styling them carefully. When the backend team was ready to start implementing the endpoints, we began to add functionality to our client, using Vanilla JS and jQuery.

Always keeping in mind the user experience and the fact that each interaction with our system should be intuitive and natural, we had to make the DOM of our pages dynamic, in order to provide each user with a custom experience, with regard to the roles held by him in the context of each conference. After implementing the requests and making sure each feature is functional, we started hiding certain elements or adding others based on the above-mentioned criteria.

Pre-loading scripts or pre-fetching data was also crucial in our implementation in order to minimize the delay between the user’s interaction with our system and the effects of those interactions. As a benchmark for that, we chose a maximum of 2s delay time.

* + - **Logging, debugging, testing**
      * + Testing was thorough, our integration and unit tests having a coverage of 90% of our endpoints and services. For full system tests we have used postman and later on manual testing using our frontend client.
        + To ease debugging, we have added our custom loggers on the backend, helping us also with keeping an audit of all interactions between our API and our client.