People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of

the Requirements for the Degree of the

MASTER

In **Electronics**

Option: **Computer Engineering**

Title:

DESIGN AND IMPLEMENTATION OF A STUDENT COMMUNICATION PLATFORM "IGEE SPACE"

Presented by:

- **RAHALI Fares**

Supervisor:

Mr. ZITOUNI Abdelkader

Registration number :………./2023

Dedication

With genuine honor and gratitude, I dedicate this work to:

First and foremost, His Majesty Allah, whose guidance made everything possible.

My beloved family —my parents, my brothers Yakoub and Abd Al-Raouf— who are an integral part of me and deserve to share in my achievements.

My friends and my teacher of the Qur'an whom have been a major source of motivation and encouragement for me during my journey. Your assistance will live on in my memories forever.

Thank you all. Your unwavering support will never be forgotten.

Acknowledgment

Praise to Allah, the almighty, for providing me with everything I needed to reach this significant milestone. Alhamdulillah for granting me the patience, wisdom, strength, and unwavering support of the people that enabled me to successfully complete this project.

It is my privilege and deep pleasure to express my sincerest gratitude to my respected supervisor, Mr. Zitouni, for his support and patience. I am genuinely grateful for the invaluable assistance he provided during a challenging phase.

My profound gratitude goes to all the teachers who dedicated themselves to their students, guiding us and contributing to our journey to this point. Additionally, I am thankful to all my colleagues who offered their assistance throughout this entire period.

Thank you.

Abstract

In recent years, social media platforms have become one of the main communication tools between students and faculty. However, because of its side effects including distractions, privacy, data security, and fake news, a new Student Communication Platform has been designed and implemented to provide students, teachers, and administrations with a reliable, and safe environment to communicate, collaborate, and interact with each other. The project's aim is to create a dedicated space that is free from the distractions of social media platforms and keep the students' information private. This platform offers features such as creating posts, sharing files, group chat, notifications, project proposals, and personalized profiles making it an all-in-one solution for student communication. The Student Communication Platform is a web-based application, that makes it accessible through many different devices and browsers. The process of building this platform went through two phases, the system design phase is demonstrated through the use of UML and entity-relationship diagrams, while the implementation showcases the utilization of Next.js, Tailwind CSS, and Supabase as programming tools. The Student Communication Platform project fulfilled all requirements, delivering a dependable, focused, and protected space for student communication.

Table of contents

List of Figures

List of Tables

List of Abbreviations

API: Application Programming Interface

App: Application

CRUD: Create, Read, Update and Delete

CSS: Cascading Style Sheets

DB: Database

DBMS: Database Management System

DOM: The Document Object Model

ERD: Entity Relationship Diagram

FK: Foreign Key

HTML: HyperText Markup Language

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol Secure

ID: identifier

IDE: Integrated development environment

IGEE: Institut de Génie Electrique et Electronique

JS: JavaScript

JSON: JavaScript object notation

JWT: JSON Web Token

MPA: Multi Page Applications

ORM: Object-Relational Mapping

PK: Primary Key

SCP: Student Communication Platform

SEO: Search Engine Optimization

SPA: Single Page Applications

SQL: Structured Query Language

SSR: Server-side Rendering

TS: TypeScript

UI: User Interface

UML: Unified Modeling Language

URL: Uniform Resource Locator

UX: User experience

VS Code: Visual Studio Code

General Introduction

The internet evolution has impacted a significant role in various aspects of our lives, and one of the most affected fields is the communication field, where the world has become fully interconnected by the appearance of social media platforms, and people can now connect, interact, and exchange information with ease across long distances.

Social media platforms, like Facebook, have gained huge popularity in the educational environment. Students, teachers, and administrations started using them as a way to share information, collaborate on projects, and receive announcements. However, concerning the downside of these platforms from privacy, data security, and most significantly distractions where a student starts wasting time after checking announcements without him realizing that, or sometimes he even forgets his first intent because of the persuasive design the social media use.

This project (Student Communication Platform) has been implemented to bridge the gap between traditional social media platforms and the specific needs of educational environments, by creating a reliable and dedicated space for students, teachers, and administrations to communicate, collaborate, and share information. The SCP provides a free-distraction space where students focus on their studies and connect with their classmates in a meaningful way which increases their productivity, not to mention that the student's information is secured and kept private. The SCP provides a range of features, including the ability to create, update, and delete posts and comment on them, share files, engage in group chat, receive real-time notifications, submit project proposals, and maintain personalized profiles and more, which offers an all-in-one solution for student communication.

In this report, we cover the key aspects of our project. Chapter 1 identifies the problems, and the objectives of the project, as well as explores the concept of web development. Chapter 2 focuses on the essential tools and technologies used to build the project. In Chapter 3, we dive into system design, utilizing UML and ERDs to create a visual representation. Finally, Chapter 4 showcases the implementation phase by presenting the user interface (UI) of our platform. The report concludes with a summary of the presented work and offers suggestions for future works.

Chapter one:

General Background

1.1 Introduction

The World Wide Web has revolutionized our lives, work, and social connections, particularly with the advent of social media platforms. This chapter provides a brief overview of the project's subject, highlighting the issues arising from the integration of social media into the education system. Furthermore, it outlines the objectives aimed at addressing these problems. Finally, some of the main concepts are introduced.

1.2 Subject

Effective communication is essential in educational institutions, However, not any communication approach is valid especially if we talk about social media. To address this, our project focuses on developing a student communication platform, a centralized web application that aims to make collaboration, information sharing, and engagement easier. We aim to overcome the limitations and the downsides of social media platforms as an educational environment by providing a user-friendly hub where students can connect, communicate, and access specialized forums based on the year and specialties. By doing so, we seek to enhance the educational experience, foster community, and empower students in their learning journey.

1.3 Problem statement

The problem we address is the use of social media as an educational environment, which poses the following challenges:

- **Inadequate tools:** Social media platforms lack the necessary tools for effective educational communication and collaboration.
- **Distractions and time wastage:** Students are often distracted by unrelated content on social media platforms, leading to wasted time that could be spent on educational activities.
- **Limited access:** Not all students have social media accounts, hindering inclusive communication and collaboration.
- **Account Security and Shutdowns:** Hacking or platform policies can lead to account compromises or closures, disrupting communication and resource sharing.
- **Lack of content supervision:** There is inadequate oversight of shared content, raising concerns about inappropriate or unreliable information.

- **Use of fake names:** Students may use fake names, compromising identity verification and creating an untrustworthy environment.
- **Exclusion of non-students:** Educational groups may lack strict verification, allowing unauthorized individuals to join.

1.4 Objectives

The project aims to solve the problems mentioned above, by providing a set of solutions as follows:

- **Develop a student communication platform:** Create a user-friendly web application that serves as a dedicated communication platform and free-distraction space for students, teachers, and administrators, fostering interaction and collaboration.
- **Minimize distractions:** Develop a platform that solely caters to educational needs, ensuring no irrelevant content that distracts students.
- **Secure registrations:** Implement a registration process where only institute students can sign up using their student card ID. The admin will verify registrations to exclude non-student accounts.
- **Content supervision:** Enable students to report posts, while granting the admin full control to monitor and remove suspicious or unrelated content.
- **Real names:** Enforce the use of students' real first and last names on the platform, promoting transparency and accountability.

1.5 Concepts

1.5.1 Web application

Web application development refers to the process of creating dynamic applications that are accessed through web browsers over the internet, it involves multiple components, including front-end development, back-end development, and database integration, it’s built using web technologies like HTML, CSS, and JavaScript, it offers cross-platform compatibility and eliminating the need for installation, it enables seamless collaboration and accessibility from anywhere, simplifying deployment and reducing maintenance efforts. [1]

1.5.2 Single-Page Application

A Single-Page Application (SPA) is a type of web application that operates within a single web page, where the content is dynamically updated without requiring reloading the page, meaning instead of navigating to different pages, the application dynamically updates specific sections or components based on user interactions, providing a smoother and more responsive user experience. SPAs can offer fast and interactive web applications that feel more like native desktop or mobile applications. Most JavaScript frameworks like React, Angular, and Vue use SPAs. [2]

1.5.3 Multi-Page Application

A Multi-Page Application (MPA) is a traditional type of website where each page represents a separate HTML document. When you click on a link or interact with the website, it loads a new page from the server. MPAs follow a more traditional web browsing experience, where each page refreshes entirely, and the whole content is replaced. MPAs rely on server-side rendering to generate and serve the HTML content to the user's browser. Each page in an MPA can have its own HTML, CSS, and JavaScript, and the navigation between pages involves a round trip to the server. MPAs are commonly used for simpler websites or content-driven applications that don't require real-time updates or extensive interactivity. [3]

1.5.4 Front-end

Front-end development involves creating and implementing web applications' UI (user interface) and UX (user experience) using HTML, CSS, and JavaScript. It focuses on designing and coding the visual and interactive elements that users see and interact with directly in their web browsers. Front-end developers work to ensure a visually appealing design, responsive layout, and interactive functionality, often utilizing frameworks like React, Angular, or Vue.js. [4]

1.5.5 Back-end

Back-end development involves creating and maintaining the server-side logic, databases, and infrastructure that power web applications. It includes writing code in server-side programming languages like PHP, Python, Ruby, or Node.js, and using frameworks to handle tasks such as data management, security, integration with external services, and scalability. Back-end developers ensure the smooth functioning of the web application, process user requests, interact with databases and implement security measures to protect data and ensure optimal performance. [4]

1.5.6 Serverless

Serverless computing, or serverless, is a cloud computing model where developers can build and run applications without managing the underlying server infrastructure. It allows developers to focus on writing code while the cloud provider handles server scaling and management. Serverless platforms execute code in response to events or triggers, providing benefits like automatic scaling, reduced operational overhead, and cost efficiency. It promotes agility and scalability for applications. [6]

1.5.7 Framework

Frameworks are pre-made structures or libraries that offer a foundation and a set of tools for building applications. They offer functionalities, design patterns, and reusable components to develop software. By using frameworks, developers can save time and effort by leveraging pre-existing code and methods to optimize the development process and promote best practices, they often include features such as database access, user authentication, templating engines, and routing mechanisms, allowing developers to focus on application-specific logic rather than reinventing the wheel. [7]

1.5.8 State Management

State management refers to how data is stored and managed within a web application. It involves keeping track of important information and the current state of the application. In web development, state management techniques are used to ensure that data is preserved and accessible across different pages. Effective state management is crucial for creating responsive and interactive web applications. [5]

1.5.9 HTTP Protocol and Status code

1.5.9.1 HTTP Protocol

HTTP (Hypertext Transfer Protocol) is an application-layer protocol, it’s the main protocol for communication between web browsers and servers. It enables the request-response model where a client sends a request to a server, and the server responds with the requested data or performs the requested action [8]. other web protocols are:

- **HTML (Hypertext Markup Language):** Standard markup language for creating web pages and structuring content.
- **HTTPS (Hypertext Transfer Protocol Secure):** Secure version of HTTP that encrypts communication for data confidentiality and integrity.
- **DNS (Domain Name System):** Protocol for translating domain names into IP addresses, facilitating server location and communication.
- **FTP (File Transfer Protocol):** Protocol for transferring files between clients and servers, enabling file management on remote servers.
- **WS (Web Sockets):** Protocol allowing real-time bidirectional communication between clients and servers, facilitating interactive web applications.

1.5.9.2 Status codes

The following tables shows some of the most used HTTP status codes: [9]

Status code	**Description**
100	Continue
200	OK
301	Moved Permanently
400	Bad Request
401	Not Authorized
404	Not Found
500	Internal Server Error
503	Service Unavailable

Table 1: Common status codes

1.5.10 URL

A URL (Uniform Resource Locator) is a web address that specifies the location of a resource on the Internet [10]. Its components include:

- **Protocol:** Specifies the communication protocol used, such as HTTP or HTTPS.
- **Domain**: The domain name or IP address of the server hosting the resource.
- **Path:** The specific location or file path on the server where the resource is located.
- **Query Parameters:** Optional parameters that provide additional information to the server.

An example of a URL is "**https://www.example.com/blog/article?id=123**":

- The protocol is **“https://**”.

- The domain is “**www.example.come**”.
- The path is “**/blog/article**”.
- The query parameter is “**id=123**”.

1.5.11 Database & Database Management Systems

1.5.11.1 Database

A database is a carefully organized collection of data that is designed to efficiently store, retrieve, and manipulate information, they can be relational databases also known as SQL databases, which organize data into tables with defined relationships, or NoSQL databases that store data in a non-tabular format (document-oriented format) which is more flexible than the SQL one. The role of databases in web applications is essential, ensuring effective data storage, retrieval, and management. [11]

1.5.11.2 Database Management Systems

A DBMS (Database Management System) is a software application that helps users store, organize, manage, and retrieve data efficiently. It serves as a bridge between users and databases, enabling tasks such as data creation, modification, deletion, querying, and reporting. Examples of popular DBMSs include Oracle Database, MySQL, Microsoft SQL Server, and PostgreSQL for SQL databases. MongoDB, Redis, and Firebase for NoSQL databases. [12]

1.5.12 Object-Relational Mapping

ORM (Object-Relational Mapping) is a programming technique that connects object-oriented code with relational databases. It enables developers to work with databases using object-oriented concepts and eliminates the need for writing raw and complex SQL queries. ORMs automate data mapping, retrieval, and manipulation, simplifying database interactions and making application development more efficient. Examples of popular ORMS are Mongoose and Prisma. [13]

1.5.13 Application Programming Interface

An API (Application Programming Interface) is a set of rules that allows different software applications to talk to each other and share information. It provides a way for developers to use specific features or data from one application in another application, without needing to know all the details of how the first application works. [14]

1.5.14 Authentication

Authentication is the process of verifying the identity of a user. It involves validating the provided credentials, to determine if the user is who they claim to be. Authentication mechanisms include techniques like username/password authentication, biometric authentication (fingerprint or facial recognition), and multi-factor authentication (combining multiple authentication factors). By authenticating users, web applications can ensure that only authorized individuals can access protected resources.

1.5.15 Authorization

Authorization is the process of determining a user's access rights and permissions after authentication. It defines what actions and resources a user can access, such as reading, writing, or modifying specific data.

1.5.16 Hashing and Salting

1.5.16.1 Hashing

Hashing is a way of taking an input (like a password) and converting it into a unique string of characters. This resulting string (called a hash) has a fixed length and looks completely different from the original input. Hashing is a one-way process, meaning you can't obtain the original password from the hash. When a user creates an account, the password is turned into a hash and saved in a database, when he wants to log in later, the password he enters is turned into a hash and compared to the stored hash. If they match, the login is successful. [15]

1.5.16.2 Salting

Salting is an extra security step used together with hashing. It involves adding a random string of characters, called a salt, to a password before it gets hashed. The salt is unique for each user and stored alongside the hashed password in the database. Salting provides protection against attacks where hackers try to figure out the original password from the hash. [15]

1.6 Conclusion

Through this chapter, the issues of integrating social media in the education system were addressed, and how we could solve these problems using a web application. It also introduces key concepts of the web that help to understand the next parts of this project.

Chapter Two:

Used Tools and Technologies

2.1 Introduction

Utilizing tools and technologies effectively has become crucial for success as software development continues to advance at a rapid rate. This chapter explores the fundamental components that form the development process, including programming languages and development and design tools.

2.2 Programming Languages

2.2.1 HTML

HTML (Hypertext Markup Language) is the standard markup language for creating the structure and presentation of web pages. It consists of tags and elements that define the content and layout of a webpage. HTML documents use opening and closing tags to enclose elements, such as headings (<h1></h1>), paragraphs (<p></p>), inputs (<input />), and more. It also supports images, links, multimedia, and interactive elements. Web browsers interpret HTML documents to display the visual layout of web pages. [16]

2.2.2 CSS

CSS (Cascading Style Sheets) is a style sheet language used to define the visual styling of HTML (Hypertext Markup Language) documents. It allows web developers to control the appearance of HTML elements on web pages by applying styles such as colors, fonts, sizes, margins, paddings, borders, and more. It also enables advanced layout techniques like positioning, floats, flexbox, and grid systems, which facilitate the creation of responsive and dynamic web designs. It is an essential part of modern web development and is supported by all major web browsers. [17]

2.2.3 JavaScript

JavaScript, or JS, is a versatile programming language used for creating interactive behavior on websites. It works on both the client-side and server-side, allowing developers to modify page elements, handle user interactions, and perform asynchronous operations (API calls). JavaScript supports OOP (Object Oriented Programming) and has a rich ecosystem of libraries and frameworks. Because JavaScript can be used for creating interactive web applications, games, mobile apps, server-side applications, and much more, it’s considered the foundation of modern web development. [18]

2.2.4 TypeScript

TypeScript (TS) is a typed superset of JavaScript developed by Microsoft. It adds static typing and modern JavaScript features to enhance the development experience and maintainability of large-scale applications. It seamlessly integrates with JavaScript codebases and popular frameworks. TypeScript enables developers to catch type-related errors early in the development process rather than the runtime, it also helps to add auto-completion to enhance the development experience. TypeScript gets translated into plain JavaScript, allowing it to be executed in any JavaScript runtime environment. [19]

2.3 Development Tools

2.3.1 Visual Studio Code

Visual Studio Code, or simply VS Code, is an integrated development environment (IDE) developed by Microsoft. Because of its customizable interface, extensive plugin ecosystem, and powerful features, VS Code is widely used by developers for writing, editing, and debugging code efficiently, it offers a range of productivity tools, like terminal access, and built-in support for various frameworks and technologies, making it the best choice for many programmers and software engineers. [20]

2.3.2 Tabby Terminal

Tabby Terminal is a modern terminal emulator designed to enhance the command-line experience. It provides a customizable interface that allows users to work with command-line applications and shell environments efficiently. Tabby Terminal offers various features such as multiple tabs, split panes, and session management, enabling users to organize and switch between different terminal sessions easily. It also supports theming, customization of keyboard shortcuts, and integration with external tools, which helps improve developers' productivity and optimize their workspace environment.

2.3.3 React

React, or React.js, is a popular and free front-end JavaScript library for building user interfaces. It enables developers to create reusable UI components and write efficient code. It uses a virtual DOM (Document Object Model) to update only the necessary parts of the user interface, leading to fast and responsive applications. React.js has a vast ecosystem of libraries and strong community support. It is suitable for developing single-page, mobile, or server-rendered applications in combination with frameworks like Next.js. React.js is widely used for building interactive and scalable web applications. [21]

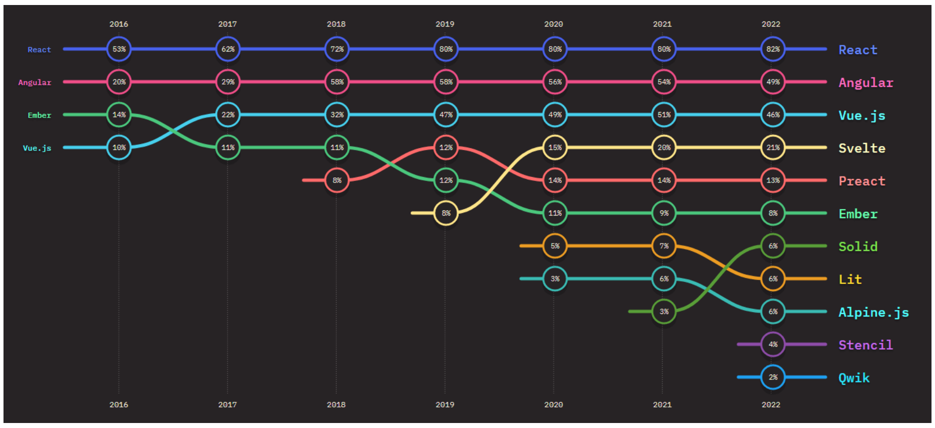


Figure 1: JavaScript frameworks ranking

2.3.4 Nextjs

Next.js is an open-source React framework created by a private company called "**Vercel**" for building server-rendered, static, and hybrid web applications, it provides a powerful set of features that simplify the development process and enhance the performance of web applications. [22]

2.3.4.1 Why Nextjs?

There are several reasons why Next.js is chosen for this project:

- **Server-side rendering (SSR)**: Next.js supports server-side rendering out of the box, allowing pages to be rendered on the server and delivered to the client. This improves performance and SEO.
- **Automatic code splitting**: Next.js automatically splits your JavaScript code into smaller chunks, loading only what is necessary for each page. This speeds up initial page loads and improves performance.
- **API development**: Next.js provides an easy way to create API endpoints within your application. This simplifies backend development and enables seamless communication between the client and server.

- **Developer experience:** Next.js offers features like hot module reloading and automatic code rebuilding, enhancing the development experience. It also provides helpful error reporting and debugging tools.
- **Active community and ecosystem**: Next.js has a vibrant community with extensive resources, tutorials, and plugins available. This fosters collaboration, provides support, and expands the capabilities of the framework.
- **SEO and performance optimization**: With built-in server-side rendering, Next.js improves SEO by delivering pre-rendered HTML to search engines.
- **Scalability**: Next.js is designed to handle large-scale applications and offers features like serverless deployment and support for edge networks. This enables your application to scale effectively and handle high-traffic loads.

With its strong community support and extensive documentation, Next.js is widely adopted for building modern, scalable, high-performance web applications.

2.3.5 Tailwindcss

Tailwindcss is a highly customizable CSS framework allowing developers to build modern, responsive user interfaces. it provides a comprehensive set of pre-defined classes that can be directly applied to HTML elements. These classes enable developers to quickly style components by composing them together, rather than writing custom CSS. It also includes advanced features like responsive design utilities, dark mode support, and a powerful JIT (Just-In-Time) compiler for optimizing the final CSS bundle. Tailwindcss helps developers create unique and efficient UIs while maintaining a consistent and scalable codebase. [23]

2.3.6 Supabase

Supabase is an open-source, cloud-based database platform that offers developers a set of tools and services for building scalable and real-time applications. It combines database functionality with an API backend, providing features like data storage, authentication, and real-time subscriptions. Supabase is built on PostgreSQL and offers a user-friendly interface for managing data. It provides client libraries for various programming languages and integrates well with frontend frameworks like React and Next.js. With Supabase, developers can rapidly develop applications without worrying about complex infrastructure management, enabling them to focus on building core functionality. [24]

2.3.7 Prisma

Prisma is a next-generation ORM (Object-Relational Mapping) for different databases like PostgreSQL, MySQL, and MongoDB, it provides type safety, automated migrations, and an intuitive data model. Developers can define database models using declarative schema, and Prisma will automatically generate the corresponding database schema and the CRUD (Create, Read, Update, Delete) operations [25]. It consists of the following parts

- **Prisma Client**: Auto-generated and type-safe query builder for Node.js & TypeScript.
- **Prisma Migrate**: Migration system.
- **Prisma Studio**: GUI to view and edit data in your database.

Prisma integrates well with popular front-end frameworks such as Nextjs, enabling efficient and type-safe database operations. it helps developers to write cleaner, and more maintainable code while improving productivity and reducing the potential for errors by abstracting away the complexities of database interactions.

2.3.8 TRPC

TRPC (typed Remote Procedure Call) is an open-source library that simplifies communication between a client and server in web applications. what makes TRPC powerful is it uses TypeScript under the hood to expect the data type from the API endpoint, this ensures strong typing and fewer communication errors between the client and the server. It also supports features like server-side caching, authentication, and authorization, allowing developers to build secure and efficient APIs. TRPC helps developers reduce the amount of complex code needed for API communication and focus more on building the actual functionality of their applications. It promotes code reusability, improves developer productivity, and helps maintain a clean and scalable codebase. [26]

2.3.9 Zustand

Zustand is a lightweight state management library for React. It uses React context API and the "useState” hook to create a store-like mechanism for managing states within components. It supports features like derived state, middleware, and selective subscription for efficient component re-rendering. Zustand has a minimalistic and declarative syntax, reducing boilerplate code and improving performance. It integrates well with other React libraries and is suitable for smaller to medium-sized applications as an alternative to more extensive state management solutions like Redux. [27]

2.3.10 JWT

JWT (JSON Web Token) is an open standard that defines a compact and self-contained method for sending securely encoded JSON objects between parties. [28]
JWT consists of three parts: a header, a payload, and a signature:

- **The header:** contains information about the type of token and the signing algorithm used.
- **The payload** carries the claims or attributes about the user or other relevant information.
- **The signature** is created by combining the encoded header, encoded payload, and a secret key. It is used to verify the authenticity and integrity of the token.

When a user successfully authenticates, an authentication server will normally issue a JWT, which can be used to send authentication data securely between a client (such as a web browser) and a server. It is frequently used in web applications for authentication and authorization.

2.4 Designing Tools

2.4.1 Figma

Figma is a cloud-based design and prototyping tool used for creating user interfaces and interactive prototypes. It offers real-time collaboration, vector editing tools, component libraries, and design system management. Figma supports remote teamwork, version control, and design handoff for developers. Figma has gained popularity among designers and developers for its ability to streamline the design process and improve collaboration between designers and stakeholders. [29]

2.4.2 Draw.io

Draw.io, now known as diagrams.net, is a lightweight diagramming software for creating diagrams and flowcharts. It has a user-friendly interface and a set of tools for designing software architecture diagrams, network diagrams, UML diagrams, and more. Draw.io supports exporting diagrams in multiple formats and is widely used by professionals and teams to communicate ideas and concepts effectively. [30]

2.5 Conclusion

This part defines the tools and technologies used to design and implement the project and how they impact productivity and code quality. It highlights the importance of selecting each tool and how they optimize the development workflow.

Chapter Three:

System Design

3.1 Introduction

This chapter explores the use of UML and Entity Relationship Diagrams (ERDs) in system design. UML provides a standardized visual language for modeling system architectures. We will discuss the principles, techniques, and practical applications of using UML in particular we choose three diagrams to model these roles; use case, sequence and class diagrams. We end the chapter by introducing ERDs and how we use them to depict relationships in a database

3.2 Unified Modeling Language

Unified Modeling Language (UML) is a standardized modeling language in software engineering. It offers a set of graphical notations and symbols to represent various aspects of a software system. UML diagrams serve as visual representations that help in documenting, analyzing, and designing software systems. UML enables a clear understanding and visualization of complex systems, making it a perfect tool for software development projects [31]. The UML can be in two types:

- **Structural Diagrams:** These diagrams focus on the static structure of a system and represent the elements and relationships within it. Some of these diagrams are: class, object, component, and deployment diagrams.
- **Behavioral Diagrams:** These diagrams emphasize the dynamic behavior and interactions between different elements in a system. They depict how the system functions and responds to events. Some of these diagrams are: use case, sequence, activity, and state machine diagrams.

This project will be analyzed and designed using class, sequence, and use case diagrams.

3.3 Use case diagram

3.3.1 Definition

A use case diagram is a UML diagram that depicts the functionalities of a system from the perspective of its users, known as actors. It shows how users interact with the system and the specific actions it offers. Actors represent entities interacting with the system, while use cases represent the system's functionalities. The relationships between actors and use cases are shown using associations and dependencies [31]. We can summarize the main components of the use case diagram as follows:

- **Actors**: Actors represent the external entities that interact with the system. They can be users, external systems, or other software components.
- **System Boundary:** The system boundary is a box or boundary that encloses all the actors and use cases within the system. It represents the scope of the system being modeled.

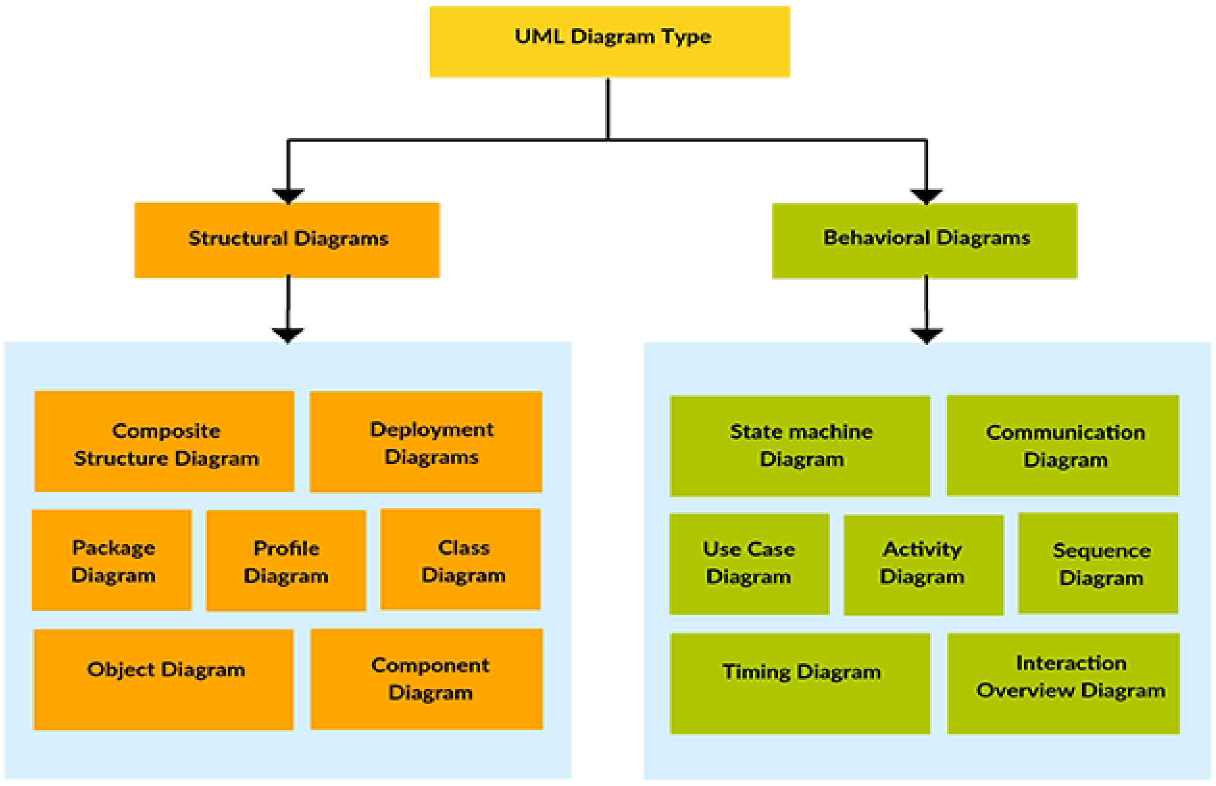


Figure 2: Types of UML diagrams

- **Relationships:** They show how actors interact with specific use cases. The main types of relationships are summarized in the table below:

Symbol	Description
association ————————	**Association:** It indicates that an actor participates in a particular use case. It's shown as a solid line connecting the two ends.
include - - - - - - - - - - - - - - ->	**Include:** It indicates that the base use case relies on the behavior defined in the included use case. It's shown as a dashed arrow pointing from the base use case to the included use case.
extend <- - - - - - - - - - - - - - -	**Extend:** It indicates that the extending use case can be optionally invoked during the execution of the base use case. It's shown as a

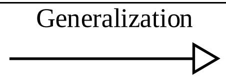
	dashed arrow pointing from the extending use case to the extended use case.
	**Generalization:** It represents a parent-child relationship between use cases, it allows for the classification of use cases into more general and specialized categories. It's shown as an arrow pointing from the specialized use case to the general use case.

Table 2: Summary of use case diagram arrows

- **Use Cases:** Use cases represent the specific actions or functionalities provided by the system. They describe a goal or task that the system performs for the actors.

  The following table shows the Actors and their corresponding use cases used in this project:

Actor	**Use case**
Student	- Sign in - Sign up - Posts page (General/Year/IGEE/Club) - Chat page - Profile page - Courses page - Project proposals page - Notifications page
Teacher	- Sign in - Sign up - Posts page (General/IGEE/Club) - Email page - Profile page - My project proposals page - Notifications page
IGEE Admin	- Sign in - Sign up - Posts page (General/IGEE/Club) - Profile page - Courses page - Registrations page - Reported posts page - Notifications page

Table 3: Actors and their use cases

3.3.2 Student use case diagram

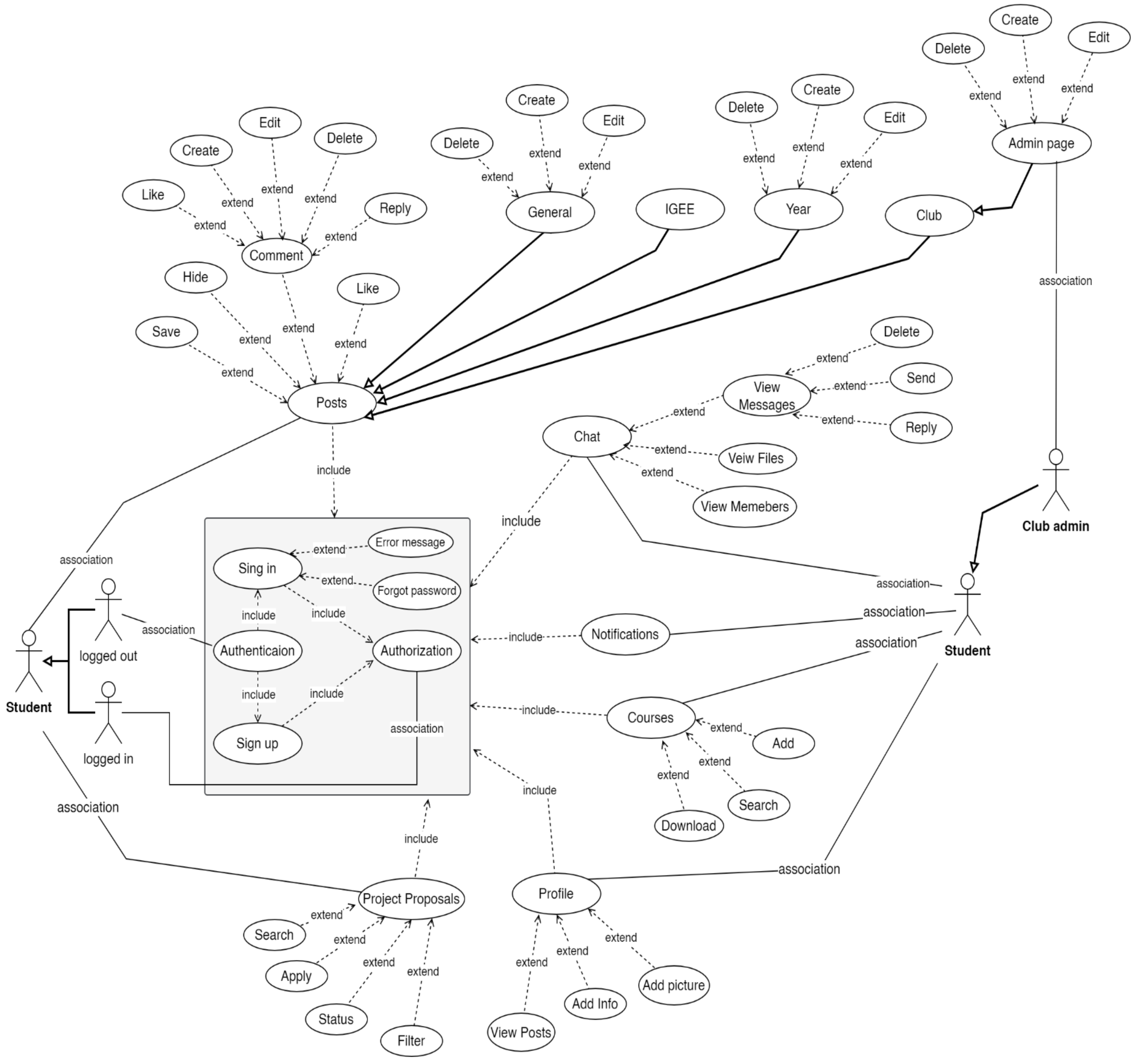
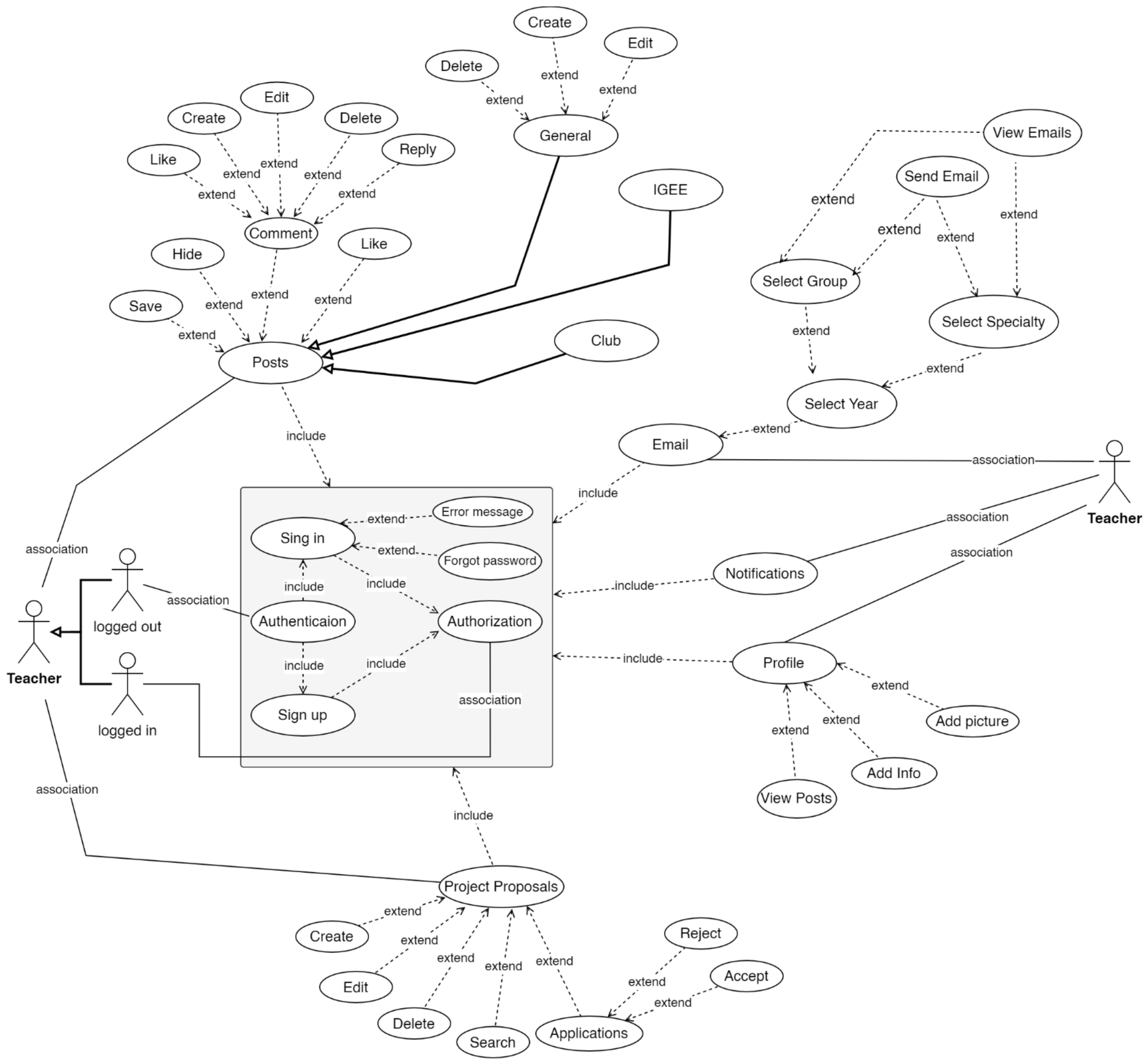


Figure 3: Student use case diagram

3.3.3 Teacher use case



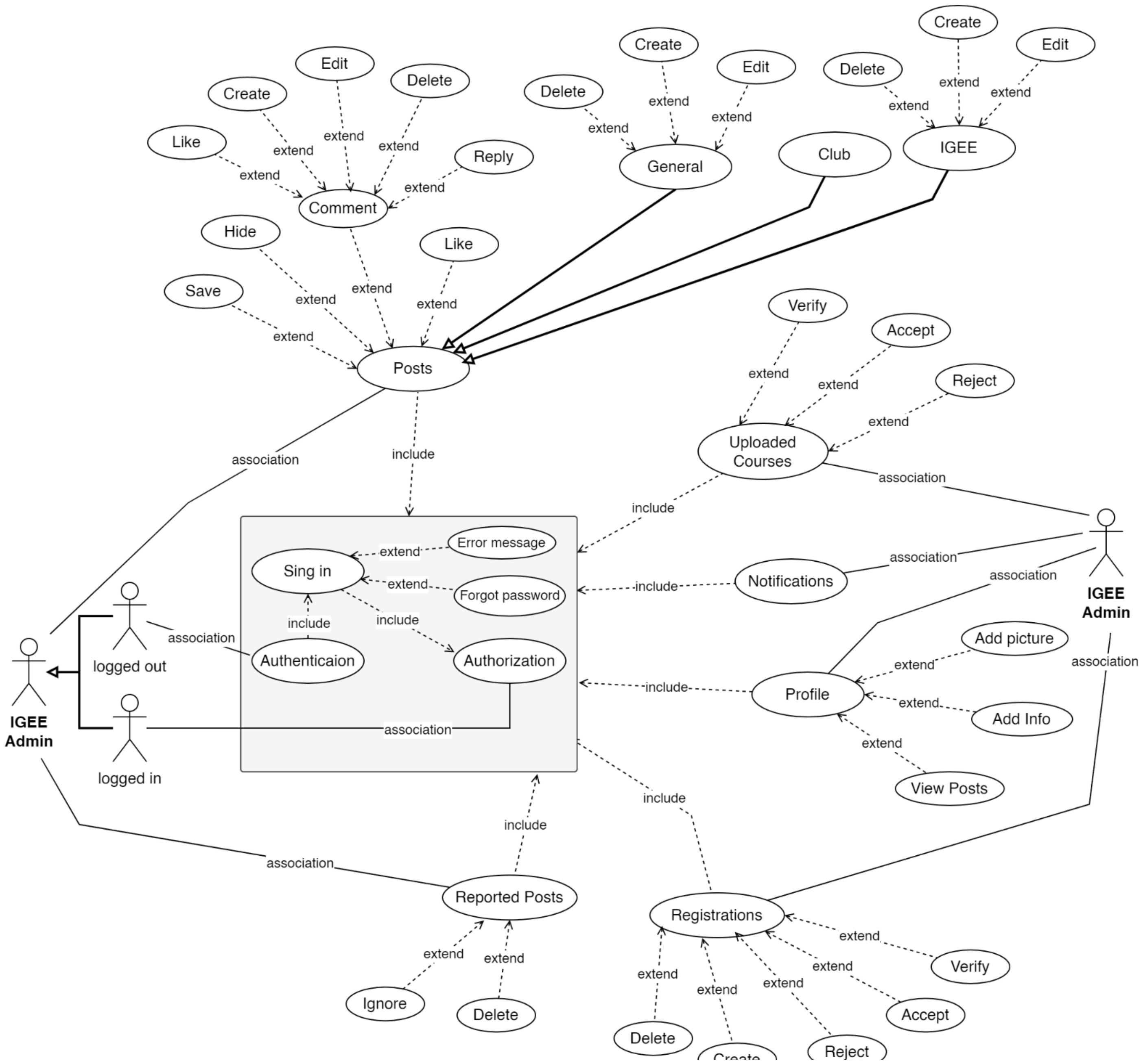


Figure 4: Teacher use case diagram

3.3.4 IGEE admin user case

Figure 5: IGEE admin use case diagram

3.3.5 Textual description of use cases

Description	The Sign-In use case allows a registered user to log in to the platform by providing valid credentials, then verifying them and giving it access.
Actor	Student, Teacher, IGEE admin
Preconditions	The user must have a registered account.
Postconditions	The user is successfully signed in and redirected to the "General posts" page. An access token (JWT) is generated for the user, allowing him to navigate through the platform.
Main Flow	1. The user visits the login page of the web app ("/login"). 2. The web app requests the user to enter his credentials (email and password). 3. The user enters his credentials (email and password). 4. The user clicks the "Login" button. 5. The web app sends a request to the server to verify the submitted credentials against the database. 6. If the credentials are valid, the server responds with an access token (JWT) allowing the user to access the platform. 7. The web app redirects the user to the "General Posts" page.
Alternative Flows	If the user enters invalid credentials (email or password), An error message is displayed indicating that one of the login credentials is incorrect. The user is prompted to re-enter the credentials or recover their account through the "Forgot password" option.

3.3.5.1 Sing in use case

Table 4: Sing in use case

3.3.5.2 Sing up use case

Description	The Sign-Up use case allows a user to create an account by providing the required information then it sends the information to the IGEE administrator to check if they are valid and create a new account.
Actor	Student, Teacher
Preconditions	The user does not have a registered account.

Postconditions	The user is registered and a verified message is sent to his email
Main Flow	1. The user visits the registration page of the web app “/register”. 2. The web app requests the user to enter the required information. 3. The user enters the required information. 4. The user clicks the “Register” button. 5. The web app sends a request to the server to verify if the email is not available. 6. If the email is not available, the server creates a new account in the database as a not verified account. 7. The IGEE admin checks the new account information if they are valid. 8. The IGEE admin sends an email to the user telling him that his account is verified. 9. The user can log in to the platform.
Alternative Flows	If the user enters a used email, an error message is displayed indicating that the email account has already been taken. If the user did enter specific information, an error message is displayed indicating the information that hasn't been entered.

Table 5: Sing up use case

3.3.5.3 General posts use case

Description	The General Posts use case allows a user to view, like, and comment on other users’ posts or create and manage (edit, delete, hide ...etc.) his posts within the communication platform, it’s called “General” because all different years can share posts.
Actor	Student, Teacher, IGEE admin
Preconditions	The user must be logged in to the platform.
Postconditions	The user can interact and use all the functionalities provided by the “General” page
Main Flow	1. The user navigates to the General posts page. 2. The web app displays a list of existing posts (excluding the hidden ones). 3. The user can view the content and details of specific posts. 4. The user can like, comment, or reply to a comment (sub-comment) on a specific post).

	5. The user can scroll down to fetch and view more posts. 6. The user can manage his post using the post options (edit, delete, save, share, disable comments). 7. The user can hide, report, share, or save other users' posts. 8. The user can create a new post by providing content, attachments (image, video, or file), or even creating a poll for voting. 9. A notification is generated depending on the user's action (for example: "student created a new post on the general page", "a teacher liked your post", …etc.) 10. The notification icon in the navbar is highlighted with the number of sent notifications.
Alternative Flows	None

Table 6: General posts use case

3.3.5.4 Year posts use case

The year posts page is the same as the General use case except that on the 2nd step of the main flow, the server checks if the student is on the same year the page requires. For example, L1 students can navigate to the "/l1" path and view only posts posted by other L1 students. If he navigates to the "/l2", "/l3", "/m1", or "/m2" path, the server will check his year and redirect him automatically to "/l1", meaning he cannot view other years' posts. Teachers and IGEE admin can access any year path freely.

3.3.5.5 IGEE posts use case

The IGEE posts page is the same as the General use case except that on the 2nd step of the main flow, the server checks if the user role is "IGEE admin", then it will show the create post option; meaning only the IGEE admin who has the privileges to create posts on this page, the other user roles (Students and Teachers) can only view the content and interact with it by commenting, liking …etc.

3.3.5.6 Club posts use case

The Club posts use case is also the same as the General use case. However, on the 2nd step of the main flow, the server checks if the user role is "Student" and is "Club admin", then it will show the create posts option; meaning the only user who can create posts is the club admin. Teachers, IGEE admin, and other students can just view the content and interact with it.

3.3.5.7 Chat use case

Description	The Chat use case enables students to engage in real-time messaging conversations within the communication platform. It allows them to send and receive text messages or files, and each group or specialty has its student members
Actor	Student
Preconditions	The student must be logged in to the platform. The student must be assigned to a year/group like (L1-Group#02) or a year/specialty like (M2-Computer)
Postconditions	The student successfully interacts and uses all the functionalities provided by the Chat page.
Main Flow	1. The student navigates to the Chat posts page. 2. The server checks if the user role is “Student”. 3. The web app displays the interface of the Chat page, previously exchanged messages, sent Files, and Online/Offline members. 4. The student views messages and scrolls up to fetch a view more messages. 5. The student types and sends a text message or a file. 6. The student replies to a specific message by clicking on the reply icon. 7. The student creates a poll to vote. 8. The student deletes a wrongly sent message. 9. The student calls a specific member by mentioning him on the chat. 10. Each action from sending, replying, voting, or deleting a message is sent as a request to the server. 11. The server broadcasts the message to all chat members and saves it on the database. 12. The other students receive the message in real-time, view it, and reply in their chat interface.
Alternative	If a user receives a new message while not actively viewing the

Flows	conversation, the web app highlights the number of unread messages on the chat icon in the navbar.

3.3.5.8 Notifications use case

Description	The Notifications use case handles the process of notifying users about new activities, messages, or updates within the communication platform. It ensures that users stay informed about important activities and events.
Actor	Student, Teacher, IGEE admin
Preconditions	The user must be logged in to the platform.
Postconditions	The user receives notifications for relevant activities and updates.
Main Flow	1. The web app continuously monitors and tracks activities and actions that are done by any user (Student, Teacher, or Admin). 2. If the user does any action, like creating, commenting, or liking a post, sending a message, or mentioning a user, the web app generates a notification base on that specific action, for example: “Student liked your post”, “Student mentioned you in a post”, …etc. 3. The user navigates to the notification page. 4. The user views the notification content, by who it’s sent, and how long ago it was sent. 5. The user scrolls down to fetch and view more previous notifications. 6. The user clicks on a notification and navigates to the source of that notification, for example, the notification “Student mentioned you in a post” takes to the post when the user is mentioned.
Alternative Flows	None

Table 8: Notifications use case

Table 7: Chat use case

3.3.5.9 Profile use case

Description	The Profile use case allows users to manage and view their personal profile information within the communication platform.
Actor	Student, Teacher, IGEE admin
Preconditions	The user must be logged in to the platform.
Postconditions	The user profile information is successfully updated and maintained.
Main Flow	1. The user navigates to his profile. 2. The user can view his profile information, including first/last name, picture, headline, bio, and any other relevant details. 3. The first name and last name are read-only, meaning the user cannot change them. 4. The user can select an editable option on their profile to update, like uploading an image or changing the username. 5. The user views three types of posts: 1- posts he created. 2- posts he commented on, replied to, or liked. 3- posts he saved.
Alternative Flows	If the user uploaded a picture profile that has a size bigger than the required size, an error message is displayed.

Table 9: Profile use case

3.3.5.10 Courses use case

Description	The Courses use case allows users to access the available courses within the communication platform.
Actor	Student, Teacher, IGEE admin
Preconditions	The user must be logged in to the platform.
Postconditions	The user uploads or downloads courses.
Main Flow	1. The user navigates to the course page. 2. The web app displays a list of all years from L1 to M2. 3. The user selects a specific year page and navigates to it. 4. The user views all the courses available on that page (pdfs, links, ...etc.). 5. The user can search, select, or download a course. 6. The user can upload a course for a specific year.

	7. The user gives a valid title for the course along with an optional description. 8. The uploaded file is checked by the IGEE admin. 9. The file is added to the course collection.
Alternative Flows	If the user uploads an invalid course, the admin rejects the course, and the file is not added.

3.3.5.11 Project proposals use case

Description	The Student Project proposal use case allows students (only L3, and M2) to access the project proposals created by teachers, in order to apply for them.
Actor	Student
Preconditions	The student must be logged in to the platform. The student must be License 03 or Master 02.
Postconditions	The student applies for a project proposal.
Main Flow	1. The student navigates to the project proposals page. **Table 10:** Courses use case 2. The web app displays a list of project proposals created by teachers. 3. The student can view, search, filter, or scroll through the project proposals. 4. The student clicks “apply” for a project proposal if the proposal is still valid (did not reach the deadline). 5. An application window pops up, asking the student to write a letter and click send. 6. The student application is sent to the teacher who created the proposal. 7. The student receives a notification about the teacher's decision (accepted or rejected).
Alternative Flows	The student can cancel the application.

Table 11: Project proposals use case

3.3.5.11 Email use case

Description	The Email use case allows the teacher to send direct emails to students' chat page in real-time, where each email is associated with a specific year/group (for L1, L2, and L3) or year/specialty (for M1 and M2)
Actor	Teacher
Preconditions	The teacher must be logged in to the platform.
Postconditions	The teacher creates an email and sends it.
Main Flow	1. The teacher navigates to the email page. 2. The web app displays all emails sent to each year. 3. The teacher can view the emails and filter them based on year, group, and specialty. 4. The teacher clicks "create new email". 5. A new window pops up asking the teacher to type a title, a content and selects year/group or year/specialty. 6. The teacher fills out the email form and clicks send. 7. The new email is sent instantly to the selected chat room.
Alternative Flows	The teacher can close the "create email" window.

Table 12: Email use case

3.3.5.12 My Project proposals use case

Description	The Teacher Project proposal use case allows teachers to create project proposals for L3 and M2 students, and accept or reject their applications.
Actor	Teacher
Preconditions	The teacher must be logged in to the platform.
Postconditions	The teacher creates a project proposal. The teacher makes a decision about the students' applications.
Main Flow	1. The teacher navigates to the project proposal page. 2. The web app displays only the list of project proposals created by a specific teacher. 3. The teacher can view, search, filter, or scroll through his project

	proposals. 4. The teacher clicks “Create new project proposal”. 5. A form window pops up asking the teacher to type in a title, an abstract, a decision date and which year or specialty is concerned with this project. 6. The teacher fills out the form and clicks “create”. 7. The project proposal is sent to all students of L1 and M2 as “OPEN” so that 8. they can apply for it. 9. The teacher clicks on the project proposal. 10. The web app displays all the applications related to this project. 11. The teacher selects students and clicks accept. 12. The non-selected students are set as rejected and the project is set as “CLOSED” 13. A decision notification is sent to all students who sent an application.
Alternative Flows	The teacher cancels the project proposal.

Table 13: My project proposals use case

3.3.5.13 Registrations use case

Description	The registrations use case allows the IGEE admin to view and verify the students' and teachers requested registrations, then accept or reject them. It also gives him the option to create pre-made accounts so the students or teachers can log in directly. IGEE admin.
Actor	IGEE admin
Preconditions	The IGEE admin must be logged in to the platform.
Postconditions	The IGEE admin can accept or reject the registrations The IGEE admin can create accounts.
Main Flow	1. The IGEE admin navigates to the registrations page through “/registrations” 2. The server checks if the user role is “IGEE admin” 3. The web app displays a list of requested registrations. 4. The IGEE admin can filter them by students or by teachers. 5. The IGEE admin scrolls down to fetch more registrations. 6. The IGEE admin verifies each registration information.

	7. The IGEE admin accepts the registration. 8. An acceptance email is sent to the user’s email address telling him his registration has been accepted.
Alternative Flows	The IGEE admin rejects the registration. A rejection email is sent to the user’s email address telling him his registration has been rejected.

Table 14: Registrations use case

3.3.5.14 Uploaded courses use case

Description	The uploaded courses use case allows the IGEE admin to view and verify the courses uploaded by the students, then accept or reject them. He can delete previously uploaded courses.
Actor	IGEE admin
Preconditions	The IGEE admin must be logged in to the platform.
Postconditions	The IGEE admin can accept or reject the uploaded course. The IGEE admin can delete previously uploaded courses.
Main Flow	1. The IGEE admin navigates to the uploaded course page through “/uploaded-courses”. 2. The server checks if the user role is “IGEE admin” 3. The web app displays a list of newly uploaded courses. 4. The IGEE admin scrolls down to fetch more courses. 5. The IGEE admin verifies each course information. 5. The IGEE admin accepts the course. 6. The course is added to the student's courses page
Alternative Flows	The IGEE admin rejects the course. The IGEE admin deletes a previous course.

Table 15: Uploaded courses use case

3.3.5.15 Reported posts use case

Description	The Reported posts use case allows the IGEE admin to view and verify the posts reported by students, then he can delete the post or ignore the report.
Actor	IGEE admin
Preconditions	The IGEE admin must be logged in to the platform.

Postconditions	The IGEE admin deletes the reported post. The IGEE admin ignores the reported post.
Main Flow	1. The IGEE admin navigates to the reported posts page through “/reported-posts”. 2. The server checks if the user role is “IGEE admin” 3. The web app displays a list of reported posts. 4. The IGEE admin scrolls down to fetch more posts. 5. The IGEE admin verifies each post. 5. The IGEE admin deletes the post. 6. The post is not available for all students.
Alternative Flows	The IGEE admin ignores the course.

Table 16: Reported posts use case

3.4 Class diagram

3.4.1 Definition

A class diagram is a visual representation in UML (Unified Modeling Language) that depicts the structure and relationships among classes in a system. It illustrates the classes, their attributes, methods, and associations with other classes. Class diagrams are commonly used in software development to design and document the static structure of object-oriented systems. [31]

3.4.2 Visibility of attributes and operations

In UML class diagrams, the visibility of attributes (variables) and operations (methods) within a class can be indicated using different symbols. The visibility determines the accessibility and scope of these class members. The following symbols are commonly used to represent visibility:

1. **Public:** Denoted by the '+' symbol. It indicates that the member is accessible and can be accessed by any other class or object.
2. **Private:** Denoted by the '-' symbol. It indicates that the member is only accessible within the class itself and cannot be accessed by other classes or objects.
3. **Protected:** Denoted by the '#' symbol. It indicates that the member is accessible within the class itself and its subclasses (inherited classes), but not from other classes outside the inheritance hierarchy.

4. **Package:** Denoted by the '~' symbol. It indicates that the member is accessible within the same package or module.

3.4.3 Class graphical description

The following figure shows how a class should be represented on a UML class diagram:

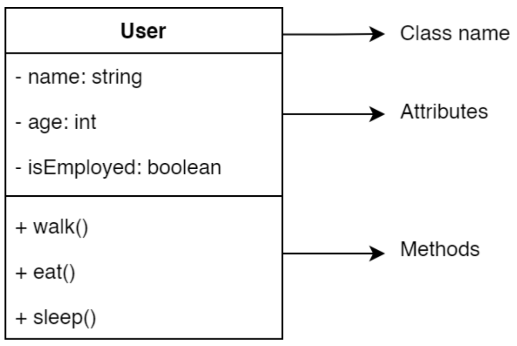


Figure 6: Class notation

3.4.4 Multiplicity

Multiplicity specifies the allowed number of instances of a class that can be associated with another class in a particular relationship. Some commonly used multiplicity notations:

- Single instance: **1**
- Zero or more instances: **0..***
- One or more instances: **1..***
- Zero or one instance: **0..1**
- Fixed number of instances: **1, 2, 3 …etc.**

3.4.5 Relationships between classes

Relationships between classes are used to depict how classes are connected or related to each other. The following table shows the common types of relationships between classes:

Relationship	Description	Graphical representation

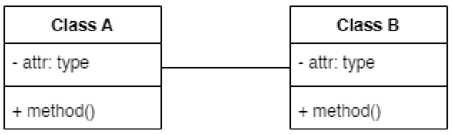
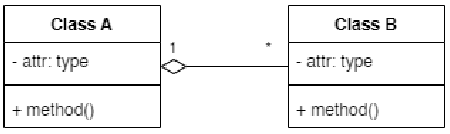
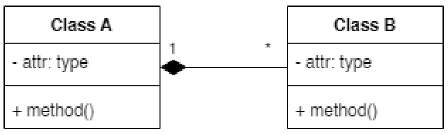
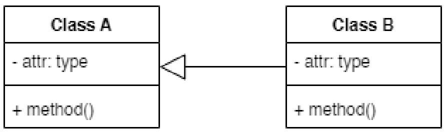
Association	It indicates that objects of one class are connected to objects of another class.	 Class A - attr: type + method() Class B - attr: type + method()
Aggregation	It indicates that a class contains or is composed of other classes.	 Class A - attr: type + method() 1 * Class B - attr: type + method()
Composition	It indicates that a class exists only if another class exists.	 Class A - attr: type + method() 1 * Class B - attr: type + method()
Inheritance	It indicates that a child class inherits all attributes and methods of its parent class.	 Class A - attr: type + method() Class B - attr: type + method()

Table 17: Class diagram relationships

3.4.6 Class diagram

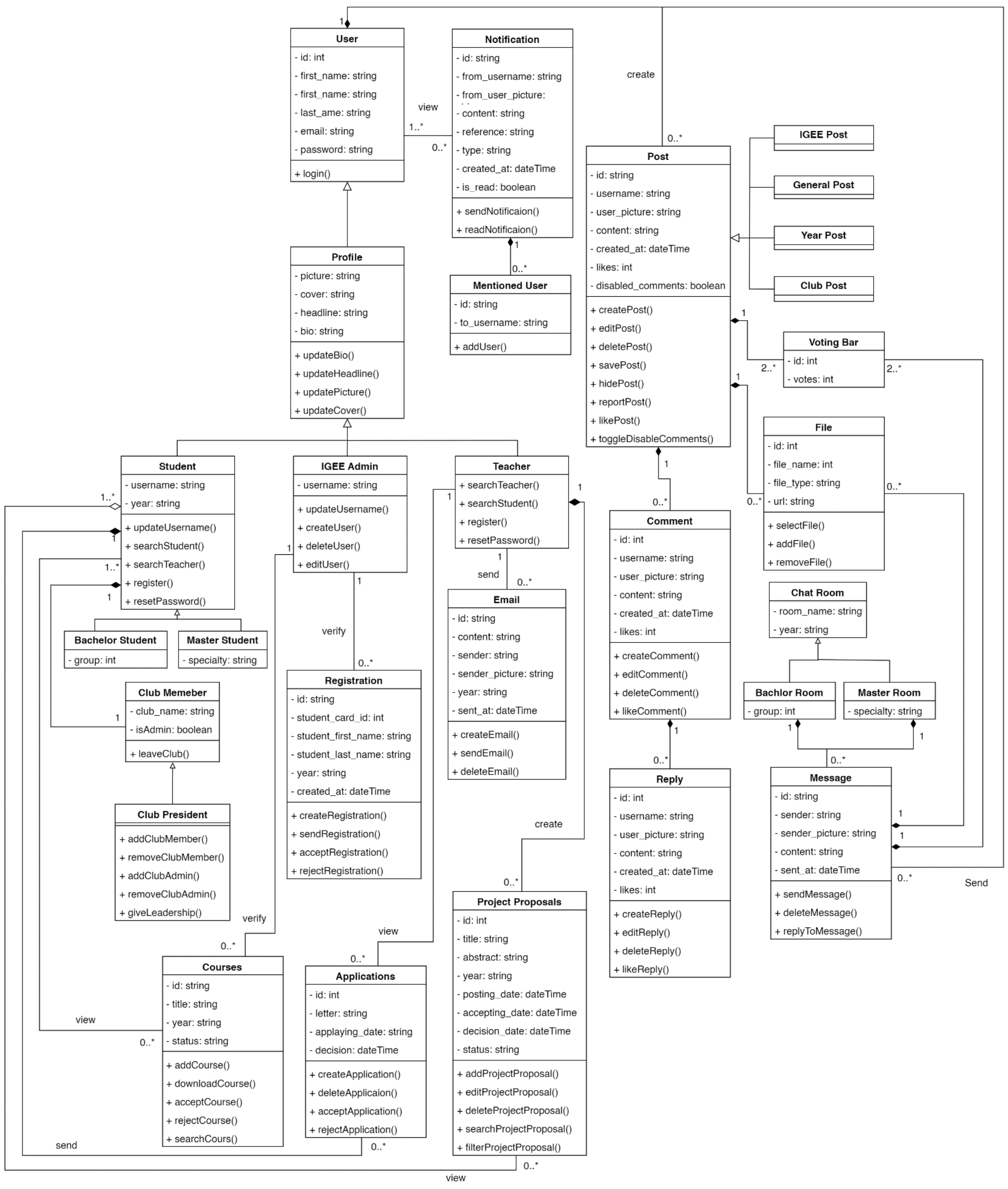


Figure 7: Platform class diagram

3.5 Sequence diagram

3.5.1 Definition

Sequence diagram illustrates the interactions and flow of messages between objects or actors over a specific period of time. It represents the dynamic behavior of a system, showcasing the sequence of actions and the order in which they occur. Sequence diagrams are commonly used to visualize the interactions between various components, classes, or objects within a system [31].

3.5.2 Sequence diagram notations

Sequence diagram uses several notations to represent the different elements and interactions. Here are the commonly used notations in sequence diagrams:

- **Actor:** An actor represents a role played by an external entity (such as a user, system, or another component) that interacts with a system.
- **Lifeline:** A lifeline represents an object or actor participating in the sequence diagram.
- **Activation bar:** Activation bars represent the period of time during which an object or actor is actively processing a message.
- **Messages:** Messages depict the interactions and communications between objects or actors. They represent the exchange of information, requests, or responses. Messages can be of different types:
    - **Synchronous Message:** It indicates a synchronous call, where the sender waits for the response from the receiver before proceeding.
    - **Return Message:** It represents the return of a response from the receiver to the sender.
    - **Self-Message:** It indicates an interaction within the same object or actor.

- **Loop Fragment (loop):** The loop fragment is used to depict a repetitive sequence of interactions.
- **Alternative Fragment** (**alt**): The alternative fragment is used to represent alternative paths or conditional behavior.
- **Reference Fragment (ref)**: The reference fragment is used to reference another sequence diagram within the current sequence diagram.

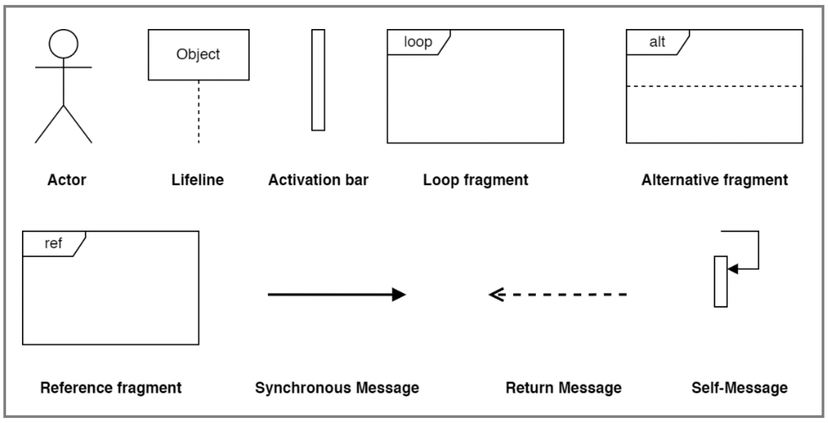


Figure 8: Sequence diagram notations

3.5.3 Sequence diagrams

The following sequence diagrams show the most important scenarios that occur in the presented project:

3.5.3.1 Authentication and authorization sequence diagram

After the user visits the platform, the web app checks if the user is already logged in, meaning he has an access token, and he will access the platform directly. If not, a login page would be displayed asking him to enter his credentials. If the user enters a valid email and password, he will have access to the platform; otherwise, an error message will appear.

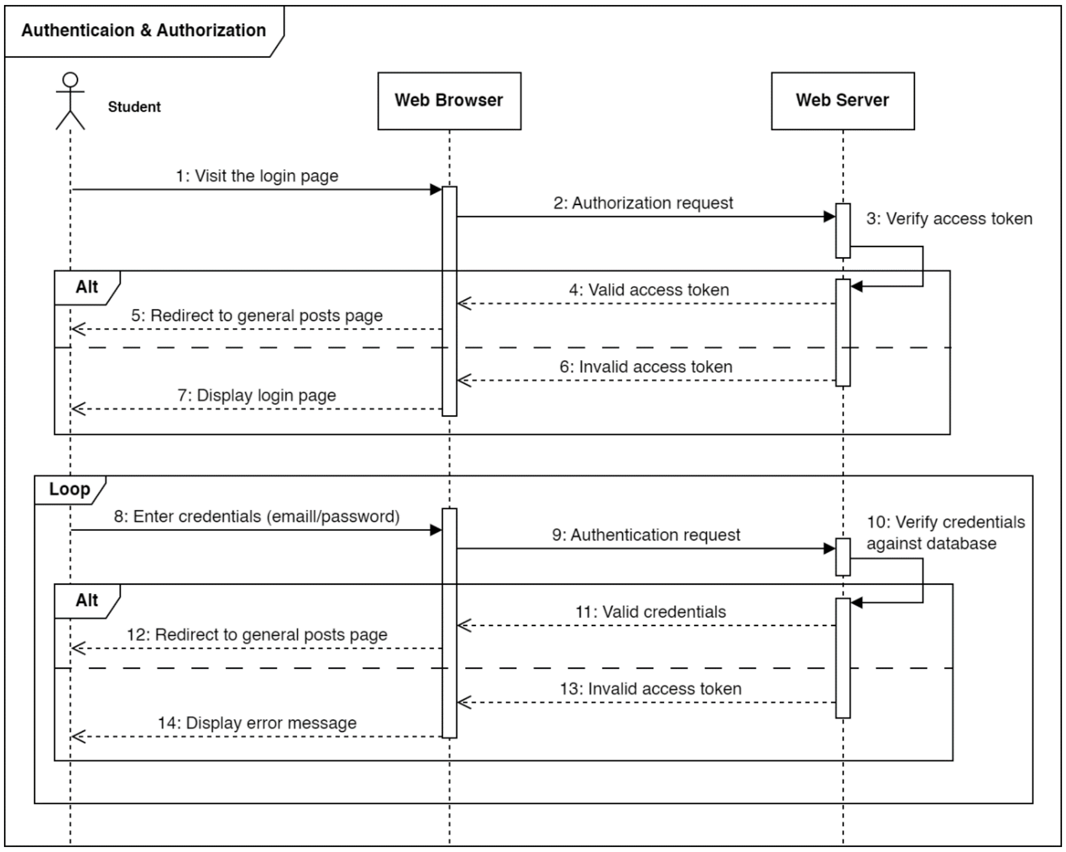

Authenticaion & Authorization
Student
Web Browser
Web Server
1: Visit the login page
2: Authorization request
3: Verify access token
Alt
4: Valid access token
5: Redirect to general posts page
6: Invalid access token
7: Display login page
Loop
8: Enter credentials (emaill/password)
9: Authentication request
10: Verify credentials against database
Alt
11: Valid credentials
12: Redirect to general posts page
13: Invalid access token
14: Display error message

Figure 9: Authentication's sequence diagram

3.5.3.2 General posts sequence diagram

Once the user is logged in, he can now visit one of the posts pages provided by the platform; in this case, it's the general posts page. The student can create a new post or view and interact with all posts created by other users.

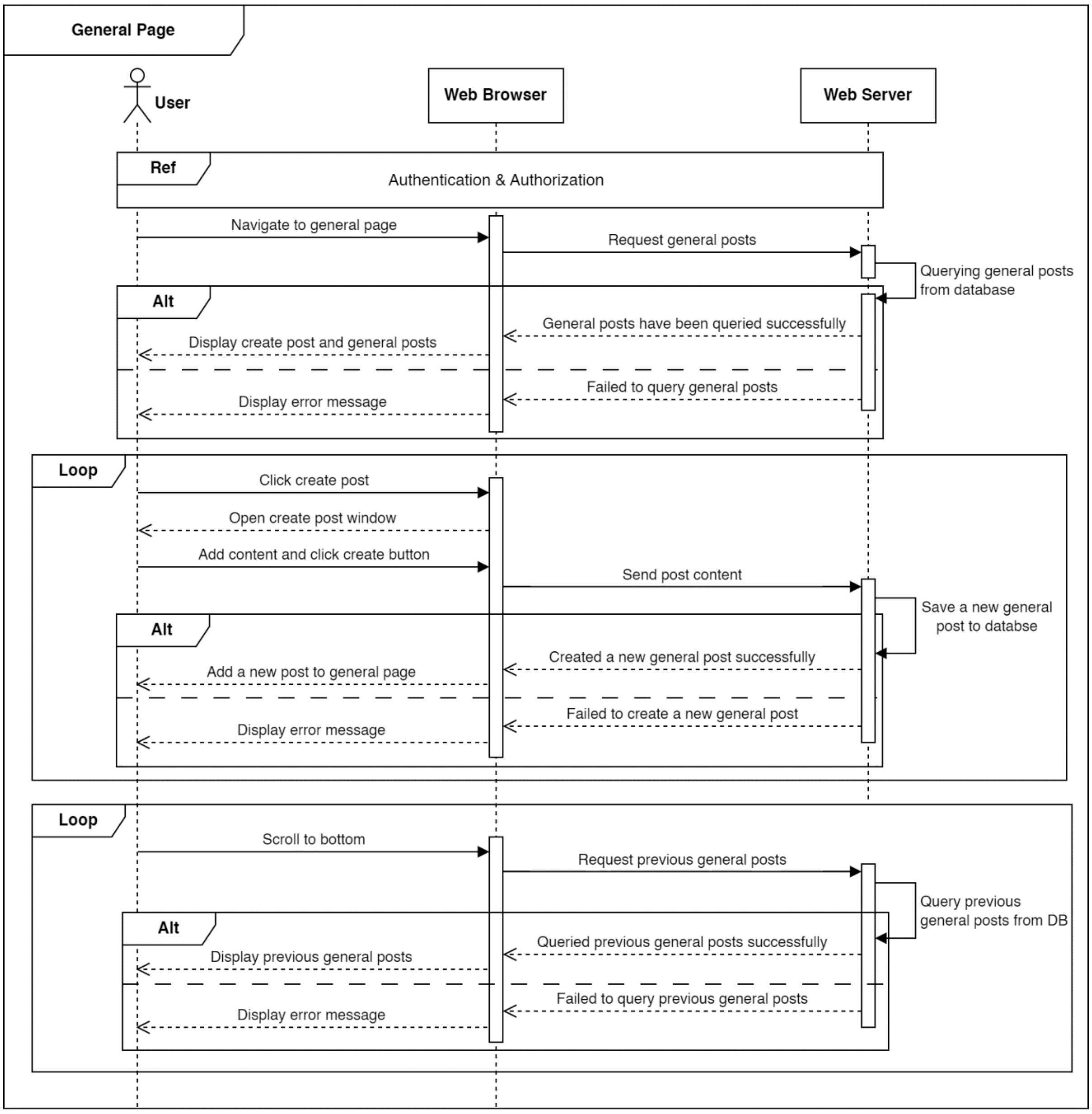
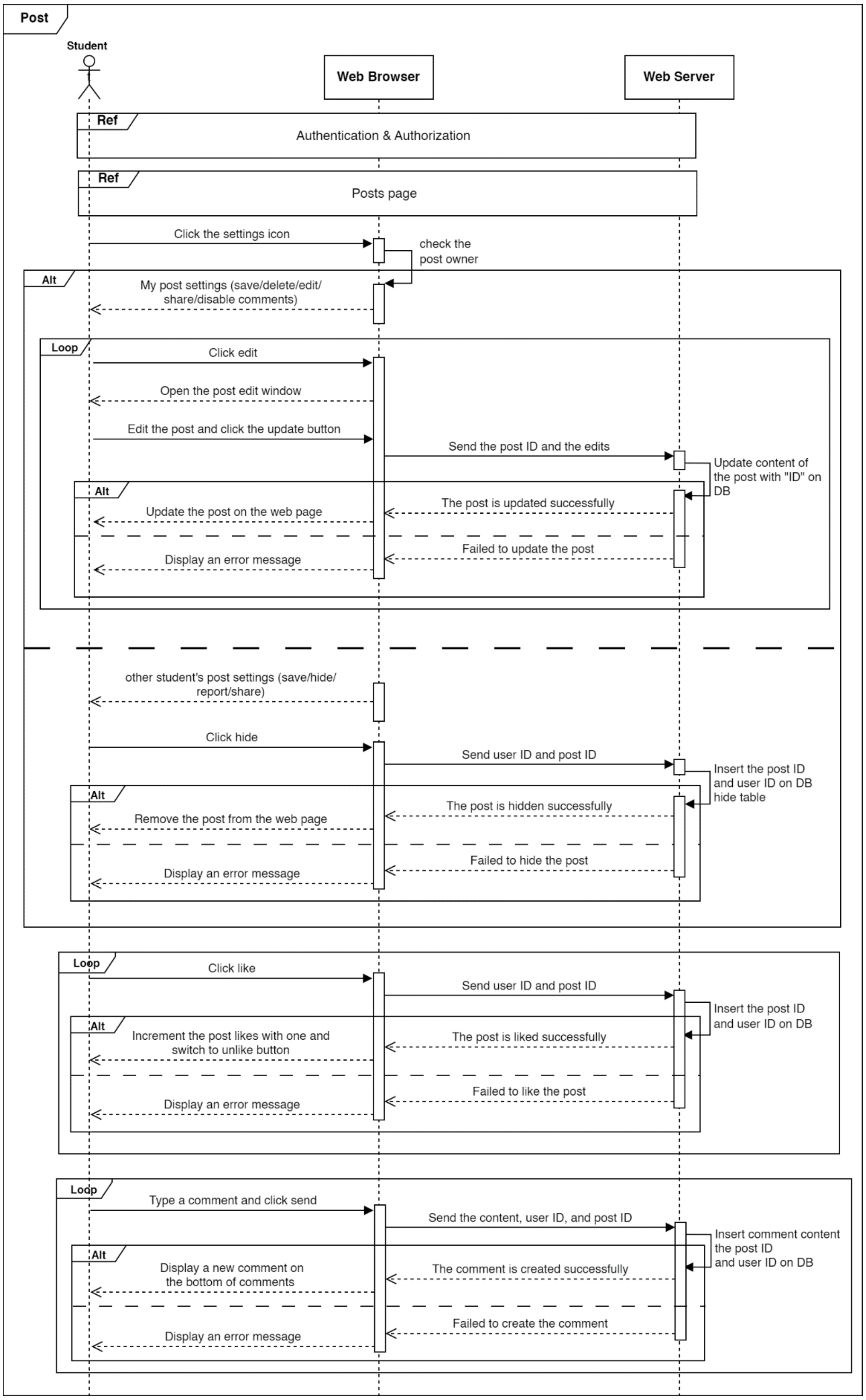

General Page
User
Web Browser
Web Server
Ref
Authentication & Authorization
Navigate to general page
Request general posts
Querying general posts from database
Alt
General posts have been queried successfully
Display create post and general posts
Failed to query general posts
Display error message
Loop
Click create post
Open create post window
Add content and click create button
Send post content
Save a new general post to databse
Alt
Created a new general post successfully
Add a new post to general page
Failed to create a new general post
Display error message
Loop
Scroll to bottom
Request previous general posts
Query previous general posts from DB
Alt
Queried previous general posts successfully
Display previous general posts
Failed to query previous general posts
Display error message

Figure 10: General posts sequence diagram

3.5.3.3 Post sequence diagram

While the student is on the general posts page, he can interact with a post using the post options. In this case, if the post is created by the student, he can save, delete, edit, share, or disable the comments. If the post is created by another student, he can save, hide, report or share it. The common functionality between both cases is liking and commenting.

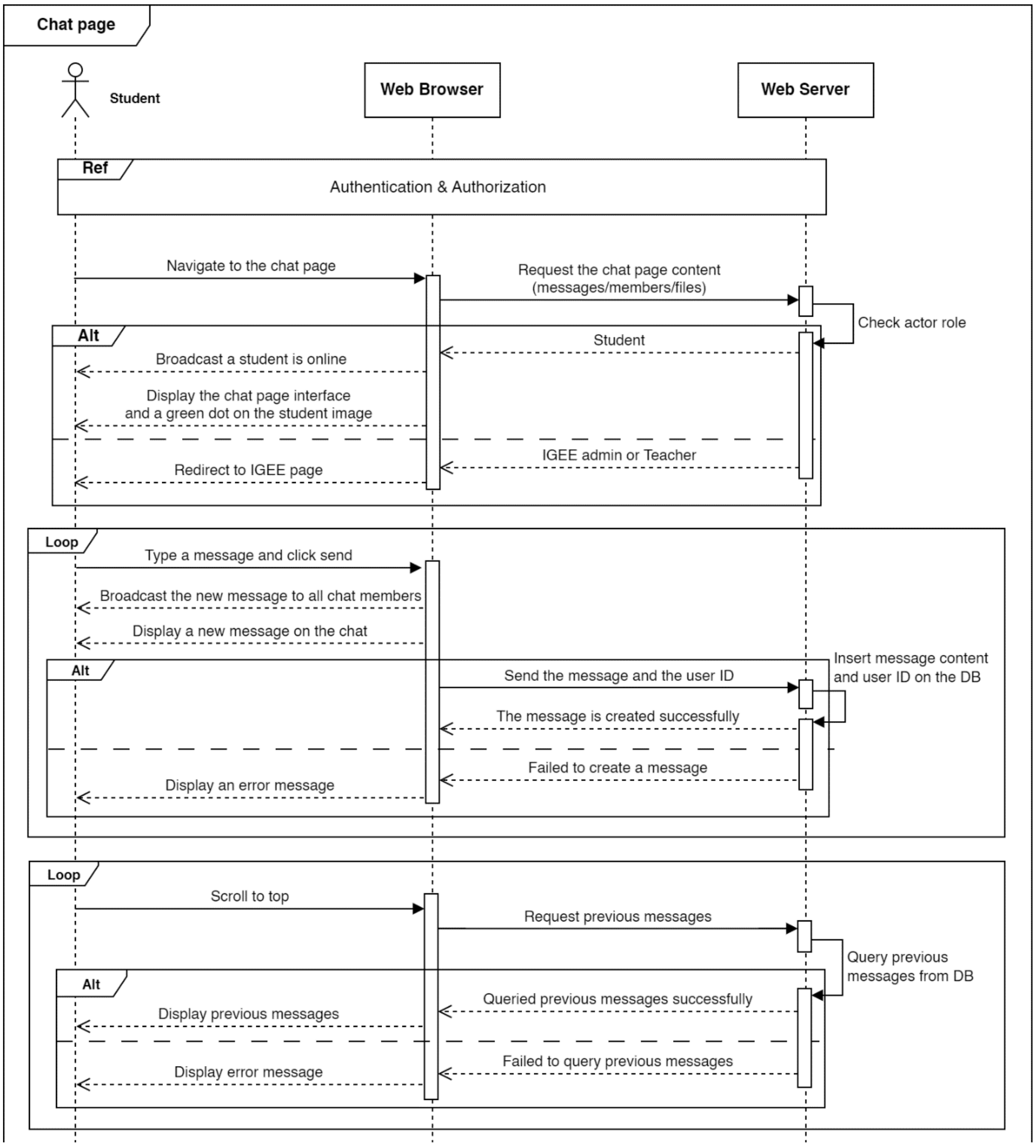


Post
Student
Web Browser
Web Server
Ref
Authentication & Authorization
Ref
Posts page
Click the settings icon
check the post owner
Alt
My post settings (save/delete/edit/share/disable comments)
Loop
Click edit
Open the post edit window
Edit the post and click the update button
Send the post ID and the edits
Update content of the post with "ID" on DB
Alt
The post is updated successfully
Update the post on the web page
Failed to update the post
Display an error message
other student's post settings (save/hide/report/share)
Click hide
Send user ID and post ID
Insert the post ID and user ID on DB hide table
Alt
The post is hidden successfully
Remove the post from the web page
Failed to hide the post
Display an error message
Loop
Click like
Send user ID and post ID
Insert the post ID and user ID on DB
Alt
Increment the post likes with one and switch to unlike button
The post is liked successfully
Failed to like the post
Display an error message
Loop
Type a comment and click send
Send the content, user ID, and post ID
Insert comment content the post ID and user ID on DB
Alt
Display a new comment on the bottom of comments
The comment is created successfully
Failed to create the comment
Display an error message

Figure 11: Post’s sequence diagram

3.5.3.4 Chat sequence diagram

The chat page allows the student to view the group's messages, online/offline members, and all the sent files. The student can send a message or files in real-time; meaning the message will reach all the online members instantly. The student can delete his messages or reply to another student's message.



Chat page
Student
Web Browser
Web Server
Ref
Authentication & Authorization
Navigate to the chat page
Request the chat page content (messages/members/files)
Check actor role
Alt
Student
Broadcast a student is online
Display the chat page interface and a green dot on the student image
IGEE admin or Teacher
Redirect to IGEE page
Loop
Type a message and click send
Broadcast the new message to all chat members
Display a new message on the chat
Alt
Send the message and the user ID
Insert message content and user ID on the DB
The message is created successfully
Failed to create a message
Display an error message
Loop
Scroll to top
Request previous messages
Query previous messages from DB
Alt
Queried previous messages successfully
Display previous messages
Failed to query previous messages
Display error message

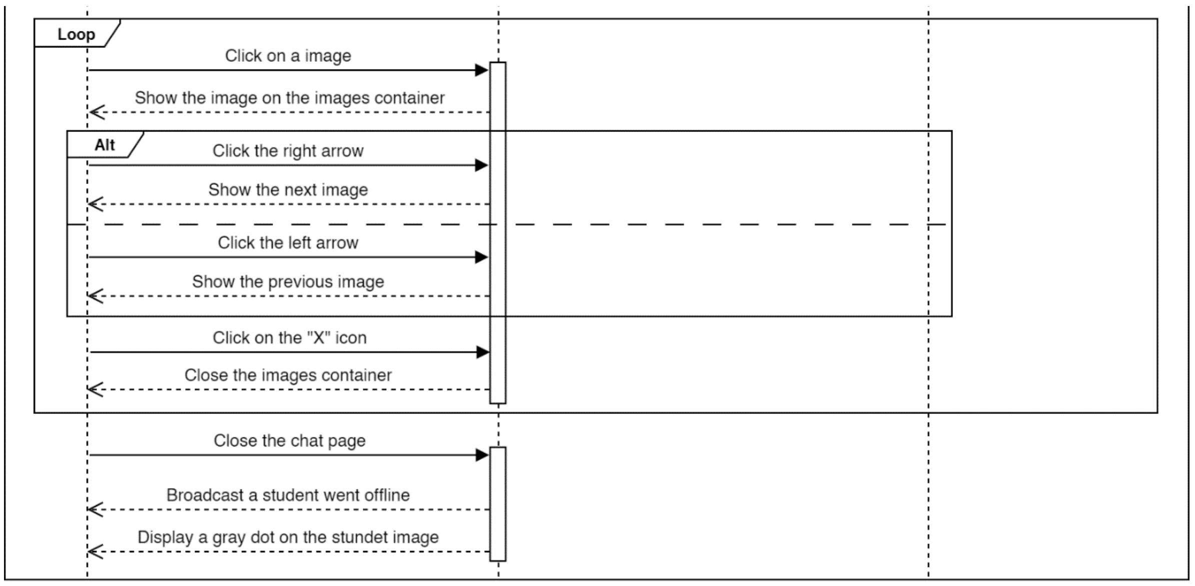
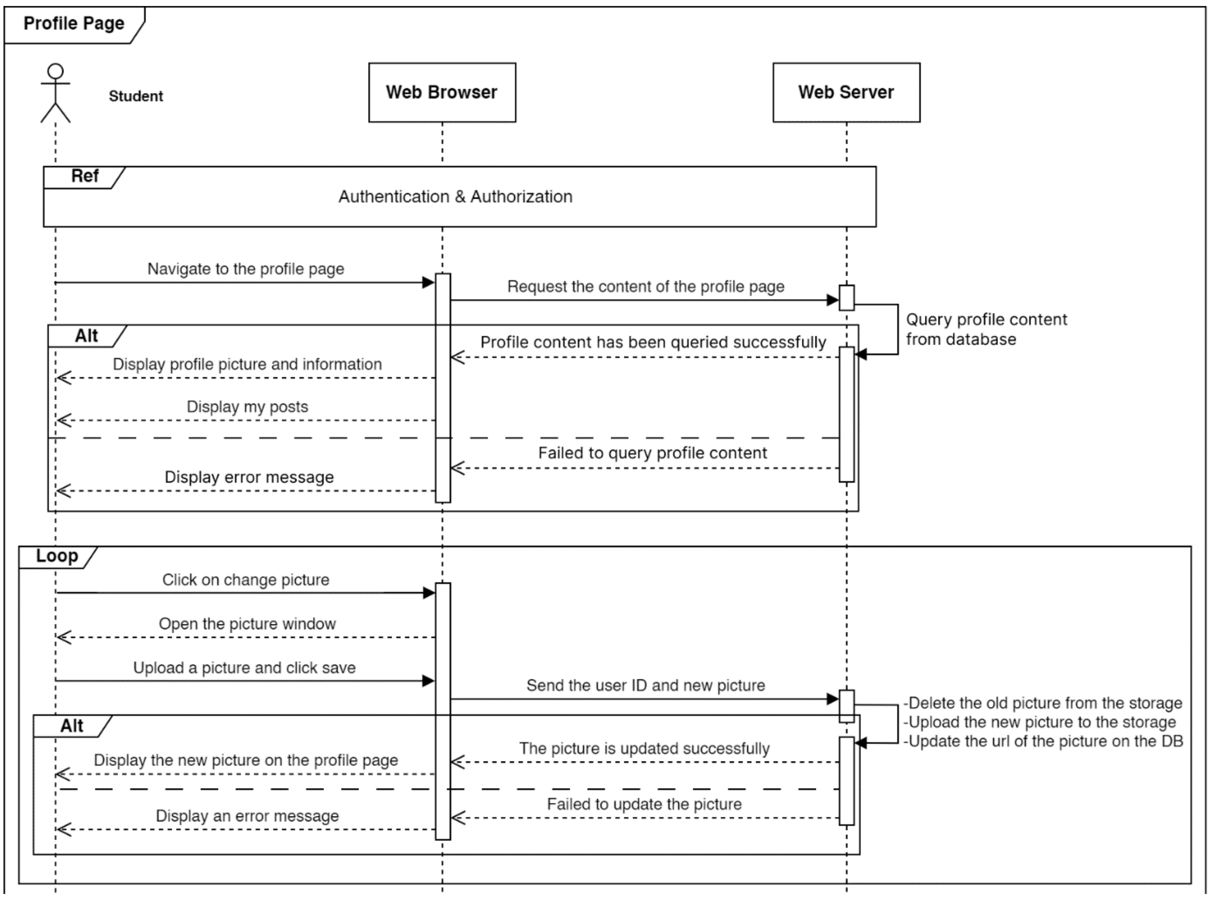


Figure 12: Chat's sequence diagram

3.5.3.5 Profile sequence diagram

When the student visits his profile page, the web app will display his profile picture and information (first name, last name, bio, …etc.). The student can then change the profile picture by uploading a new one, or updating his information; the first and last names are read-only, meaning he cannot change them.



Profile Page
Student
Web Browser
Web Server
Ref
Authentication & Authorization
Navigate to the profile page
Request the content of the profile page
Query profile content from database
Alt
Profile content has been queried successfully
Display profile picture and information
Display my posts
Failed to query profile content
Display error message
Loop
Click on change picture
Open the picture window
Upload a picture and click save
Send the user ID and new picture
-Delete the old picture from the storage
-Upload the new picture to the storage
-Update the url of the picture on the DB
Alt
The picture is updated successfully
Display the new picture on the profile page
Failed to update the picture
Display an error message

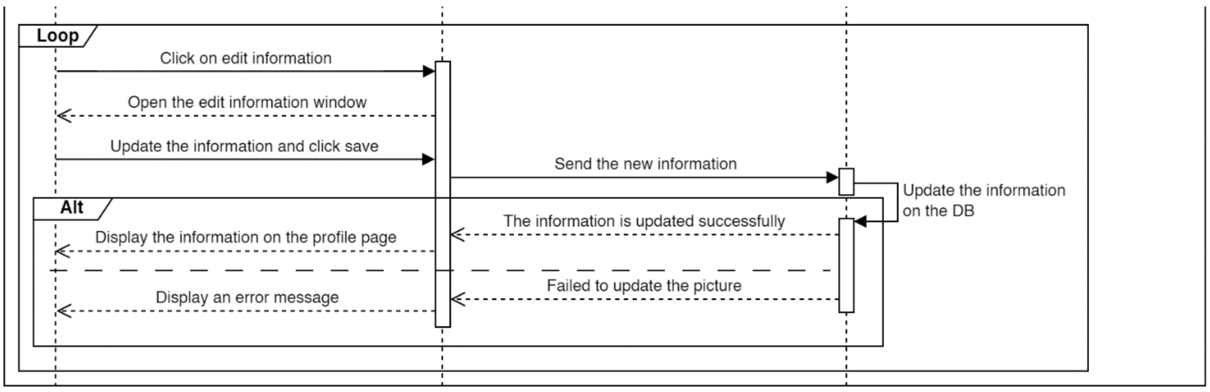
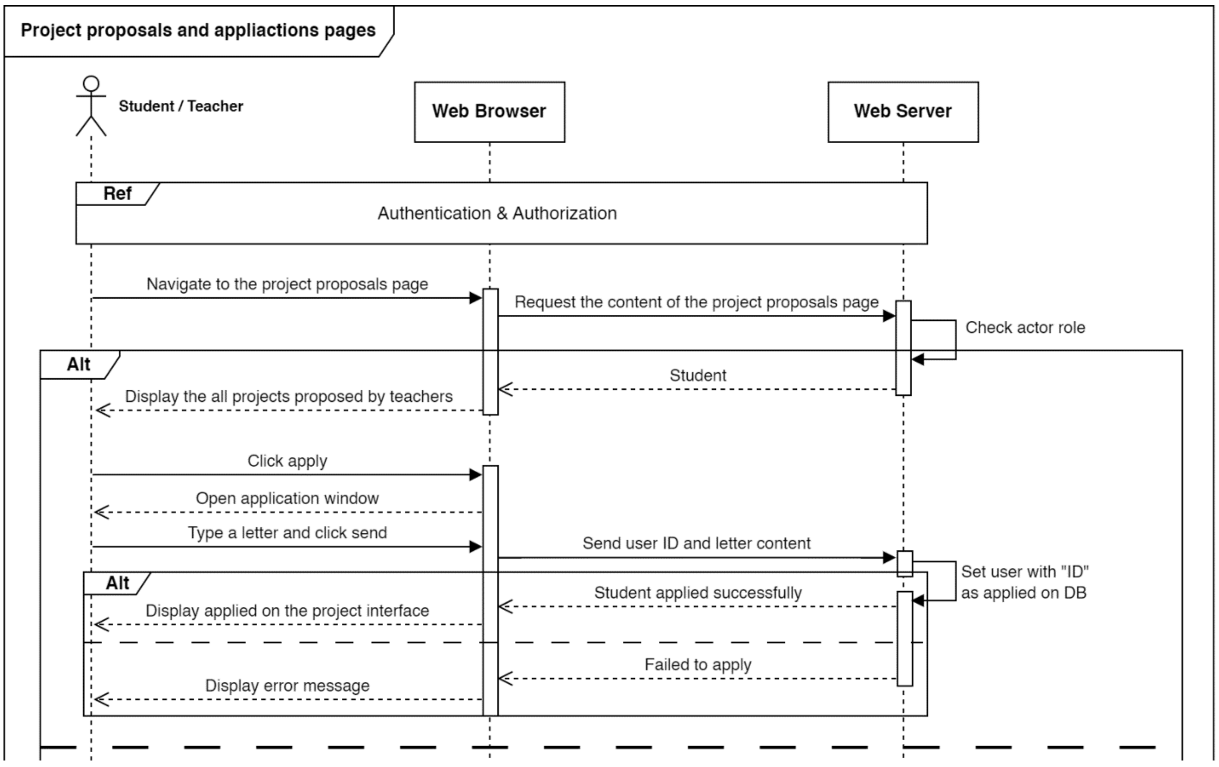
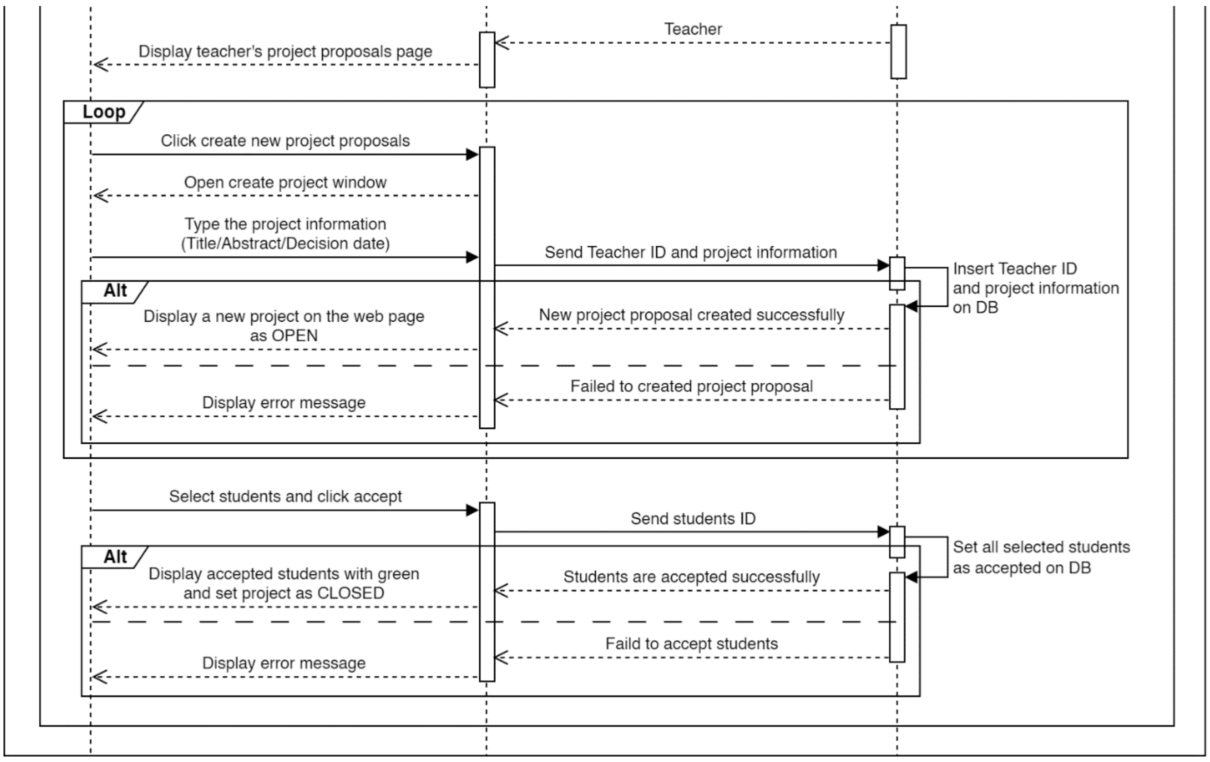

Loop
Click on edit information
Open the edit information window
Update the information and click save
Send the new information
Update the information on the DB
Alt
The information is updated successfully
Display the information on the profile page
Failed to update the picture
Display an error message

Figure 13: Profile's sequence diagram

3.5.3.6 Project proposals and project applications sequence diagram

When the teacher opens the project proposals page, he will view all of his created proposals along with the students' applications for each proposal; then he will accept or reject students based on their applications letter and their profiles. The teacher can also create a new project proposal or delete old ones. On the other hand, when the student visits the project applications page, he will see all proposals created by all teachers; and then he can apply for each project proposal.


Project proposals and appliactions pages
Student / Teacher
Web Browser
Web Server
Ref
Authentication & Authorization
Navigate to the project proposals page
Request the content of the project proposals page
Check actor role
Alt
Student
Display the all projects proposed by teachers
Click apply
Open application window
Type a letter and click send
Send user ID and letter content
Set user with "ID" as applied on DB
Alt
Student applied successfully
Display applied on the project interface
Failed to apply
Display error message



Teacher
Display teacher's project proposals page
Loop
Click create new project proposals
Open create project window
Type the project information (Title/Abstract/Decision date)
Send Teacher ID and project information
Insert Teacher ID and project information on DB
Alt
Display a new project on the web page as OPEN
New project proposal created successfully
Failed to created project proposal
Display error message
Select students and click accept
Send students ID
Set all selected students as accepted on DB
Alt
Display accepted students with green and set project as CLOSED
Students are accepted successfully
Faild to accept students
Display error message

Figure 14: Project proposals and project applications sequence diagram

3.5.3.7 Courses sequence diagram

When the student visits the courses page, a list of all years will be displayed from L1 to M2; then when he selects a specific year, the web app will display all the courses related to that year. The user can then search, download, or upload courses. When the courses are uploaded for the first time, they are set as not verified until the IGEE admin verifies and accepts them; the courses will be accessible to all students.

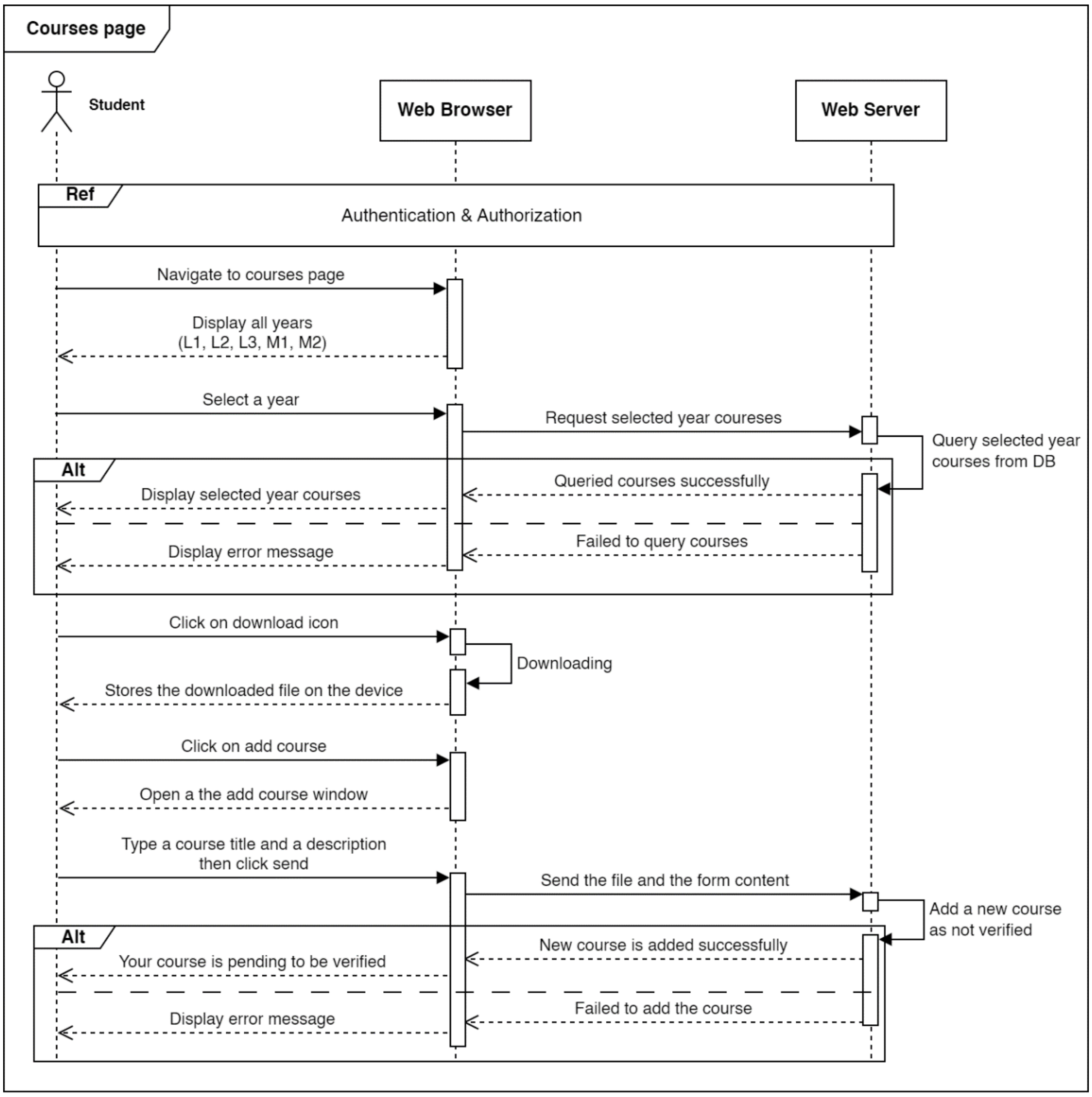

Courses page
Student
Web Browser
Web Server
Ref
Authentication & Authorization
Navigate to courses page
Display all years (L1, L2, L3, M1, M2)
Select a year
Request selected year coureses
Query selected year courses from DB
Alt
Queried courses successfully
Display selected year courses
Failed to query courses
Display error message
Click on download icon
Downloading
Stores the downloaded file on the device
Click on add course
Open a the add course window
Type a course title and a description then click send
Send the file and the form content
Add a new course as not verified
Alt
New course is added successfully
Your course is pending to be verified
Failed to add the course
Display error message

Figure 15: Courses' sequence diagram

3.6 Entity Relationship Diagram

3.6.1 Definition.

An Entity Relationship Diagram (ERD) is a visual representation of the relationships between entities (or objects) in a database. It is a widely used modeling tool in database design and depicts the logical structure of a database system [32]. Here are the main components and symbols used in an ERD:

1. **Entities:** Entities represent the objects or concepts in the database. They are typically depicted as rectangles with their names inside. For example, in a university database, entities could be Student, Course, and Faculty.

2. **Attributes:** Attributes are the characteristics or properties of an entity. They provide further details about the entity. Attributes are shown as ovals connected to their respective entities. For instance, a Student entity may have attributes such as Student ID, Name, and Age.
3. **Relationships:** Relationships illustrate the associations between entities. They describe how entities are connected to each other. Relationships are represented by diamond shapes connecting related entities. Examples of relationships include:
   - One-to-Many.
   - Many-to-Many.
   - One-to-One.
4. **Cardinality:** Cardinality represents the number of instances of one entity that can be associated with another entity in a relationship. It is indicated using numbers or symbols near the relationship lines. Common cardinality notations include:
   - One instance.
   - Zero or One instance.
   - Zero to Multiple instances.
   - One to Multiple instances.
5. **Primary Key:** A primary key uniquely identifies each instance of an entity. It is denoted by underlining the attribute in the entity. For example, in a Student entity, the Student ID attribute may serve as the primary key.

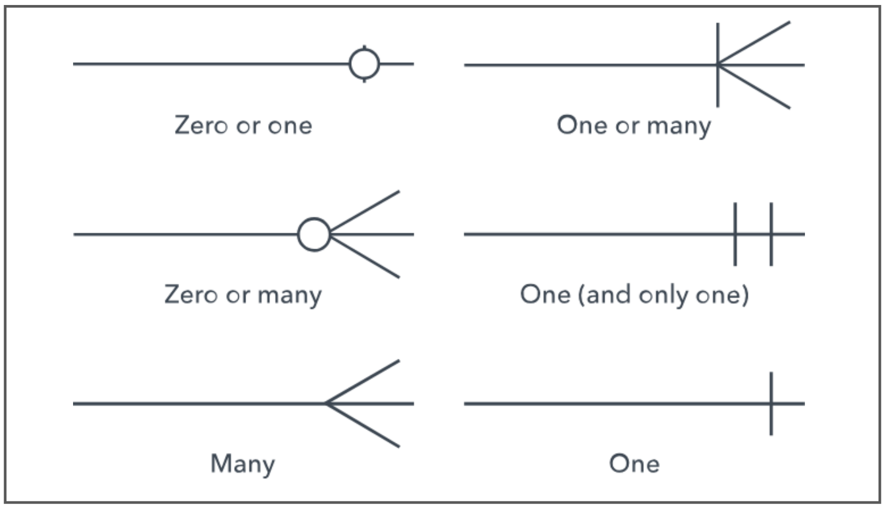


Figure 16: Entity relationship diagram symbols

3.6.2 Database schema tables

The following table demonstrates all the database tables used in this project. Each table has a primary key "PK" that uniquely identifies each row within that table; whereas relationships between tables are established using foreign keys "FK(column, table)".

Table	**Column**
User	id PK first_name last_name email password notification_seen_at role
Profile	user_id PK FK(id, User) username headline year group specialty bio picture cover club club_role
Post	id PK user_id FK(id,User) post_content created_at comments_disabled type page
PostLike	user_id PK FK(id, User) post_id PK FK(id, Post)

SavedPost	user_id PK FK(id, User) post_id PK FK(id, Post)
HiddenPost	user_id PK FK(id, User) post_id PK FK(id, Post)
ReportedPost	user_id PK FK(id, User) post_id PK FK(id, Post)
Comment	id PK user_id FK(id, User) post_id FK(id, Post) comment_content created_at
CommentLike	user_id PK FK(id, User) comment_id PK FK(id, Comment)
SubComment	id PK user_id FK(id, User) post_id FK(id, Post) comment_id FK(id, Comment) sub_comment_content created_at
SubCommentLike	user_id PK FK(id, User) sub_comment_id PK FK(id, SubComment)
File	id PK post_id FK(id, Post) name url type
Poll	id PK post_id FK(id, Post) poll_content
Voter	id PK poll_id FK(id, Poll) voter_id

Message	id PK sender_id FK(id, User) content year group specialty sent_at
MessageFile	id PK message_id FK(id, Message) name url type year group specialty
MessagePoll	id PK message_id FK(id, Message) poll_content
MessageVoter	id PK message_poll_id FK(id, MessagePoll) voter_id
Notification	id PK user_id FK(id, User) subject_id reference type year group specialty created_at
MentiondUser	user_id PK FK(id, User) notification_id PK FK(id, Notification)

ProjectProposal	id PK user_id FK(id, User) title abstract posting_date accepting_date decision_date year status
Application	id PK user_id FK(id, User) project_id FK(id, ProjectPorposal) letter applaying_date decision
Course	id PK title description year status url type

Table 18: Database tables

3.6.3 Database schema diagram

In order to depict the database schema tables in a more readable diagram, EDR methods are utilized. The following figure shows the database schema diagram of the platform.

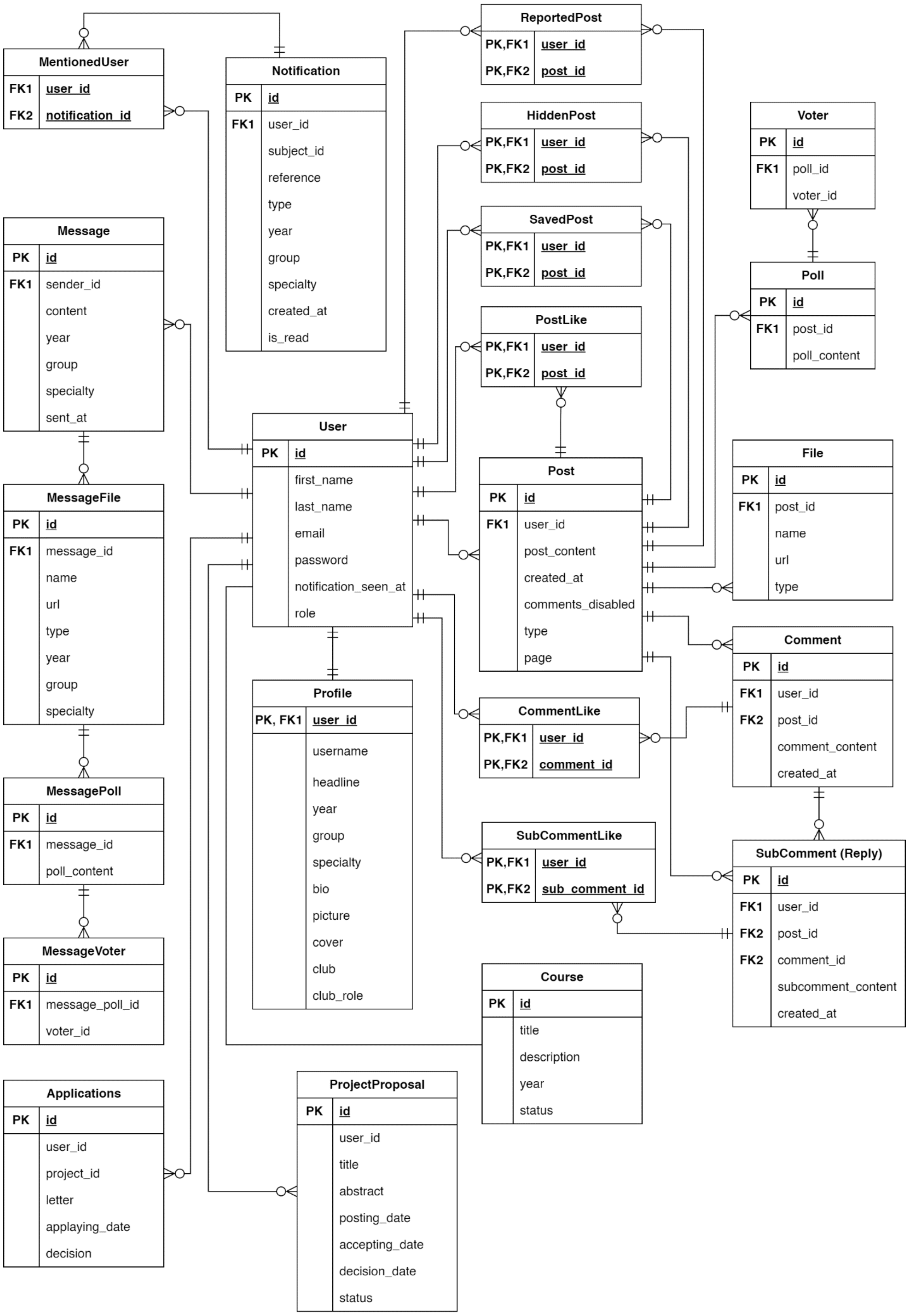


Figure 17: Database schema diagram

3.7 Conclusion

In this chapter, we utilized UML diagrams to design the platform system. We defined the platform needs through the use case diagram, followed by sequence diagrams to illustrate component interactions. Then we designed our application using a class diagram, defining class relationships. Finally, we introduced Entity Relationship Diagram (ERD) and used its methods to depict the database schema.

Chapter Four:

System Implementation

4.1 Introduction

In this final chapter, we shift our focus to system implementation after completing the application design and defining user roles. Here, we present an overview of the application and its functionalities along with the user interfaces and some main coding parts to provide a solid representation of the system.

4.2 Platform User Interface

4.2.1 Interface of Landing page

The landing page is the entrance of the platform; it's the first page the user faces when he visits the platform. The landing page has a header that shows the platform name "IGEE SPACE", a sub-header that gives a brief explanation of the platform, and two buttons for registration.

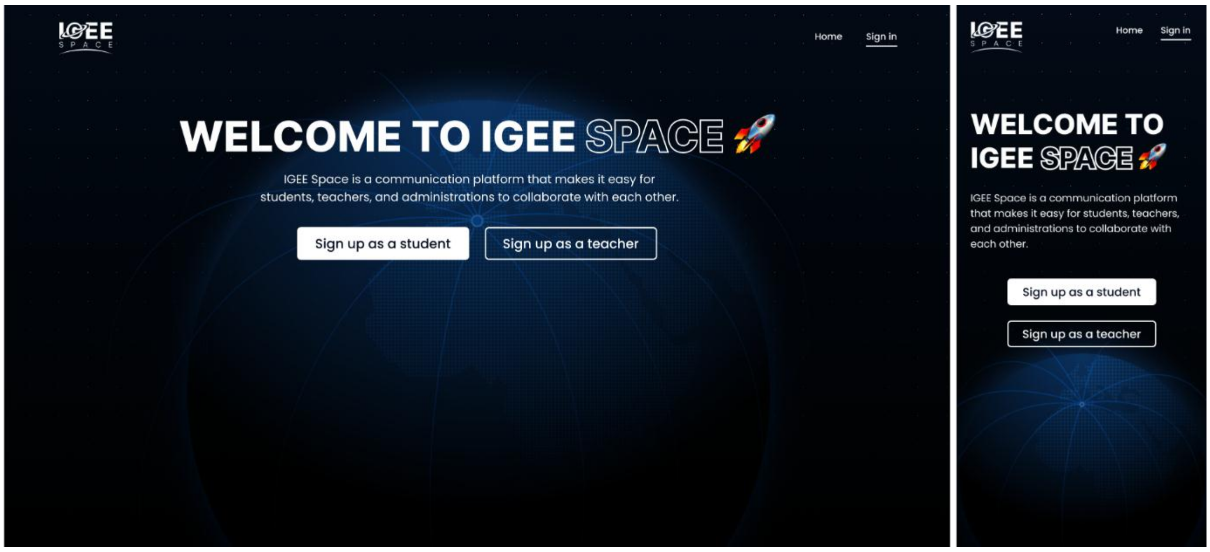

IGEE SPACE
Home
Sign in
WELCOME TO IGEE SPACE
IGEE Space is a communication platform that makes it easy for students, teachers, and administrations to collaborate with each other.
Sign up as a student
Sign up as a teacher

Figure 18: Landing page

4.2.2 Interface of Authentication page

The authentication page has three main pages, two for the registration part and one for the login part. The registration part includes both student and teacher registration pages where each one has to fill up the form and then click register, the login part includes entering valid credentials to gain access to the platform; whether it is a student, teacher, or an admin.

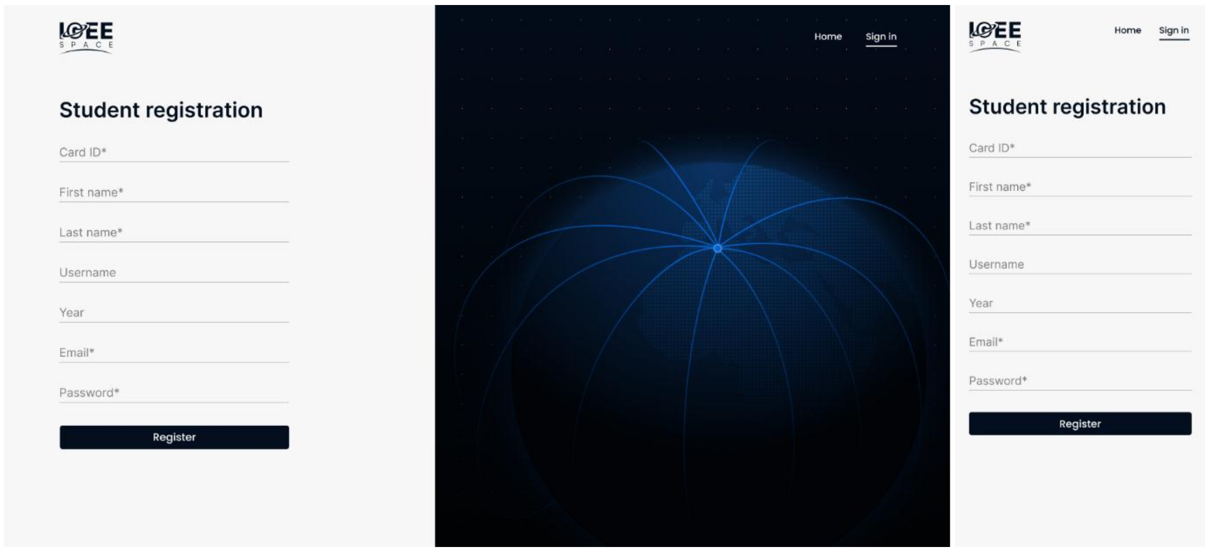


Figure 19: Student registration page

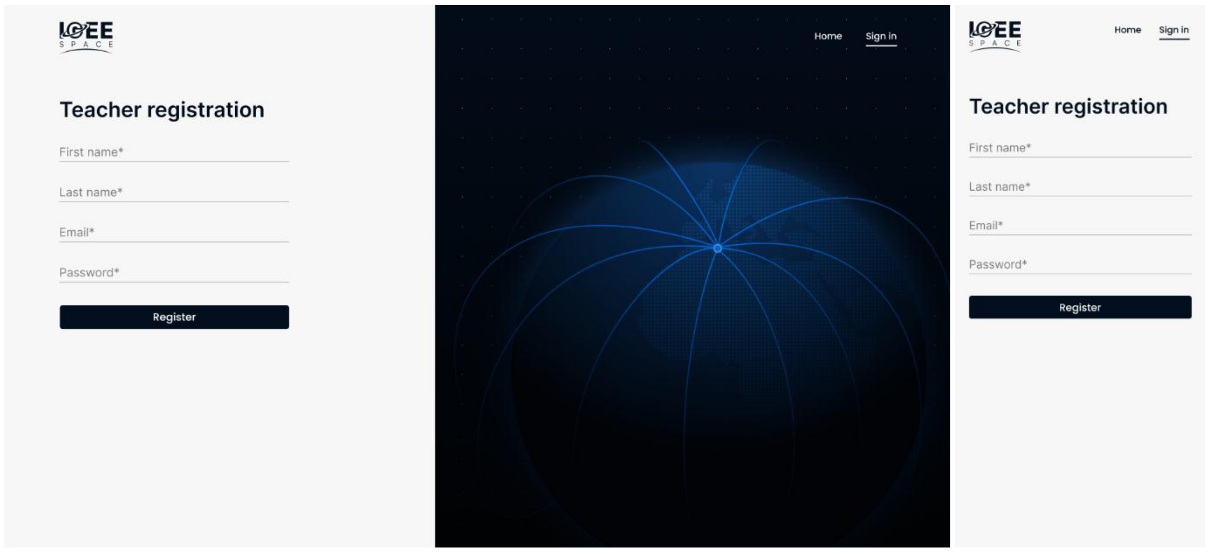


Figure 20: Teacher registration page

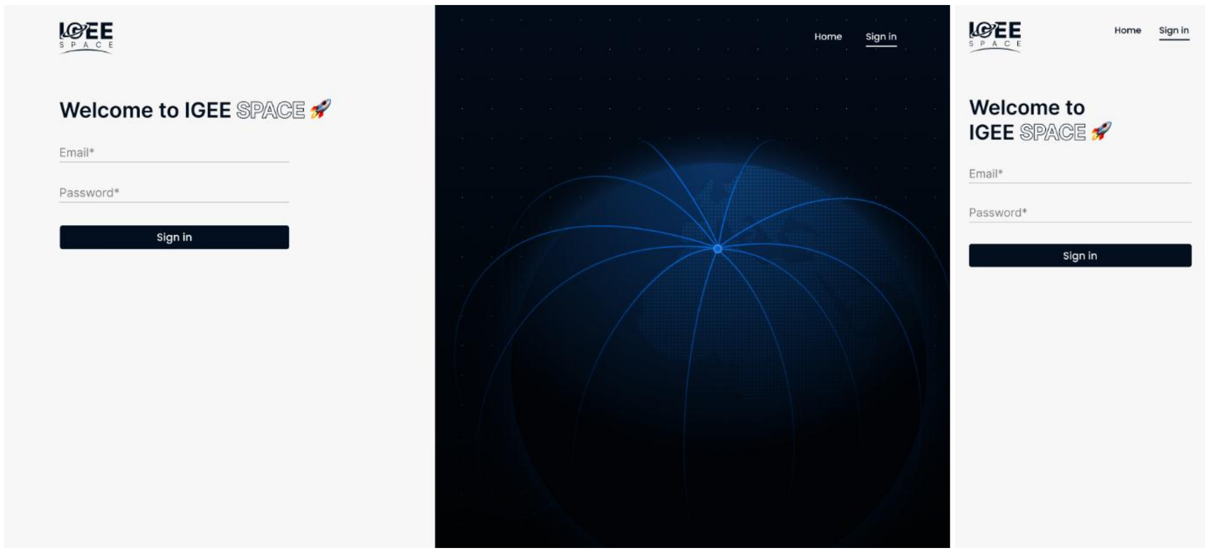
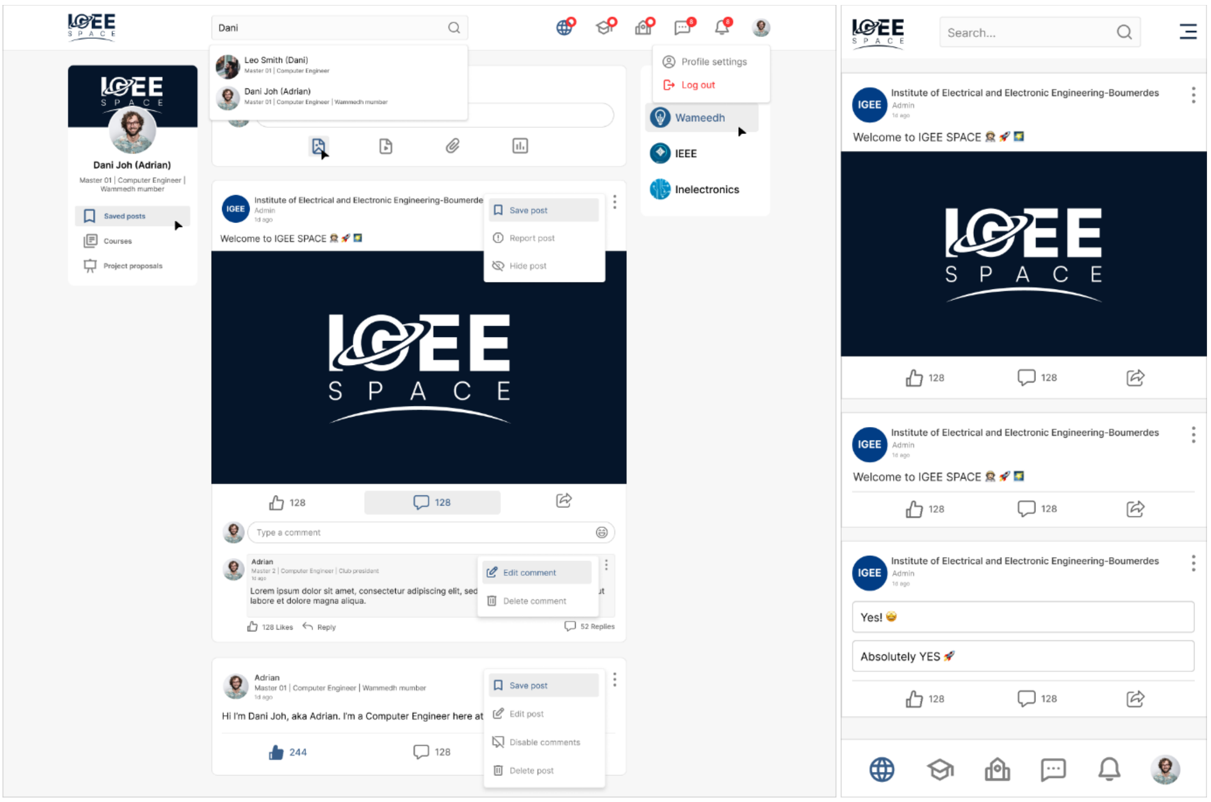


Figure 21: Login page

4.2.3 Interface of Posts page

There are four types of posting pages, General, Year, IGEE, and Club pages, each one serves a different purpose. However, they all share a common functionality, they allow the user to share posts, managing, commenting, or liking them.



Welcome to IGEE SPACE

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed ... labore et dolore magna aliqua.

Hi I'm Dani Joh, aka Adrian. I'm a Computer Engineer here at

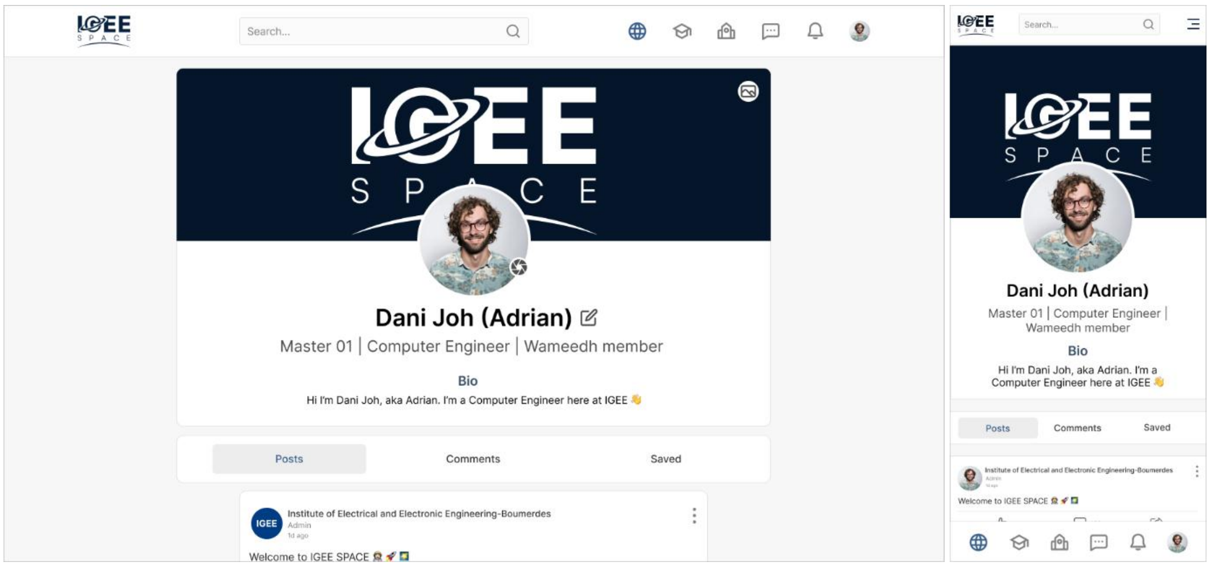
Yes!

Absolutely YES

Figure 22: Posts page

4.2.4 Interface of Profile page

The profile page is like a resume paper because students or teachers can share their information, pictures, bio, skills …etc. Teachers can view students' profiles when it comes to choosing project proposals. There the profile should be clean, readable, and looks more professional. It also shows the posts the user created, interacted with, or saved..


Dani Joh (Adrian)

Master 01 | Computer Engineer | Wameedh member

Bio

Hi I'm Dani Joh, aka Adrian. I'm a Computer Engineer here at IGEE 👋

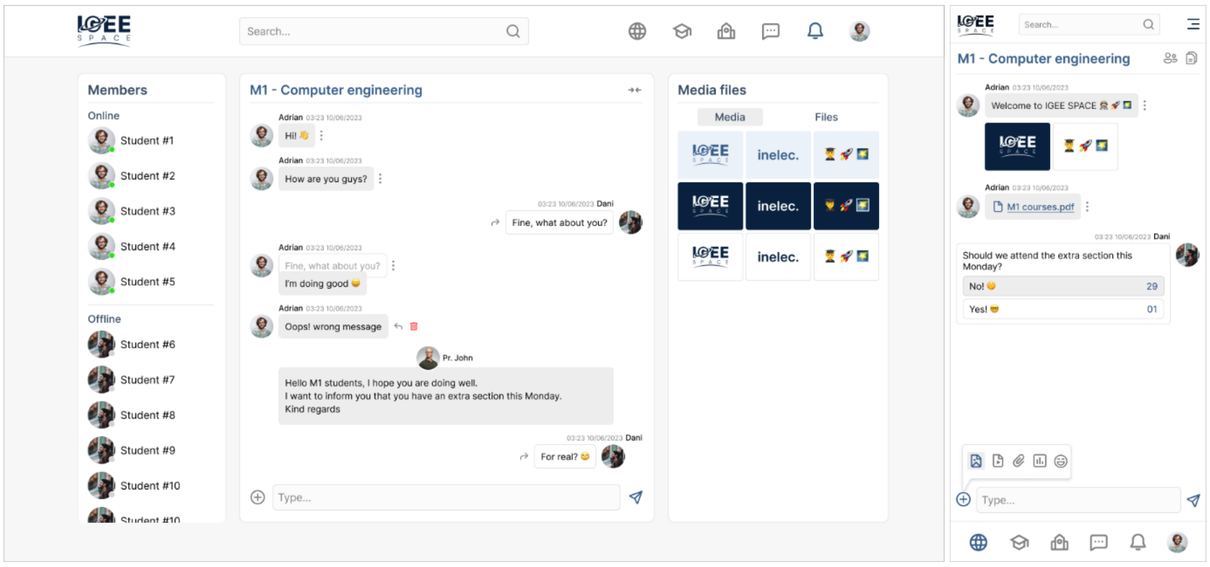
Institute of Electrical and Electronic Engineering-Boumerdes
Admin
1d ago

Welcome to IGEE SPACE 👨‍🎓🚀🖼

Figure 23: Profile page

4.2.5 Interface of Chat page

The chat page is where students can send messages, upload files, or create polls. It has all the necessary features that make collaborations and interaction between students much easier. When teachers send emails, it goes directly to the chat page which makes sure that all chat members read the email.


Adrian 03:23 10/06/2023
Hi! 👋

Adrian 03:23 10/06/2023
How are you guys?

03:23 10/06/2023 Dani
Fine, what about you?

Adrian 03:23 10/06/2023
Fine, what about you?
I'm doing good 😊

Adrian 03:23 10/06/2023
Oops! wrong message

Pr. John
Hello M1 students, I hope you are doing well.
I want to inform you that you have an extra section this Monday.
Kind regards

03:23 10/06/2023 Dani
For real? 😂

Adrian 03:23 10/06/2023
Welcome to IGEE SPACE 👨‍🎓🚀🖼

Adrian 03:23 10/06/2023
M1 courses.pdf

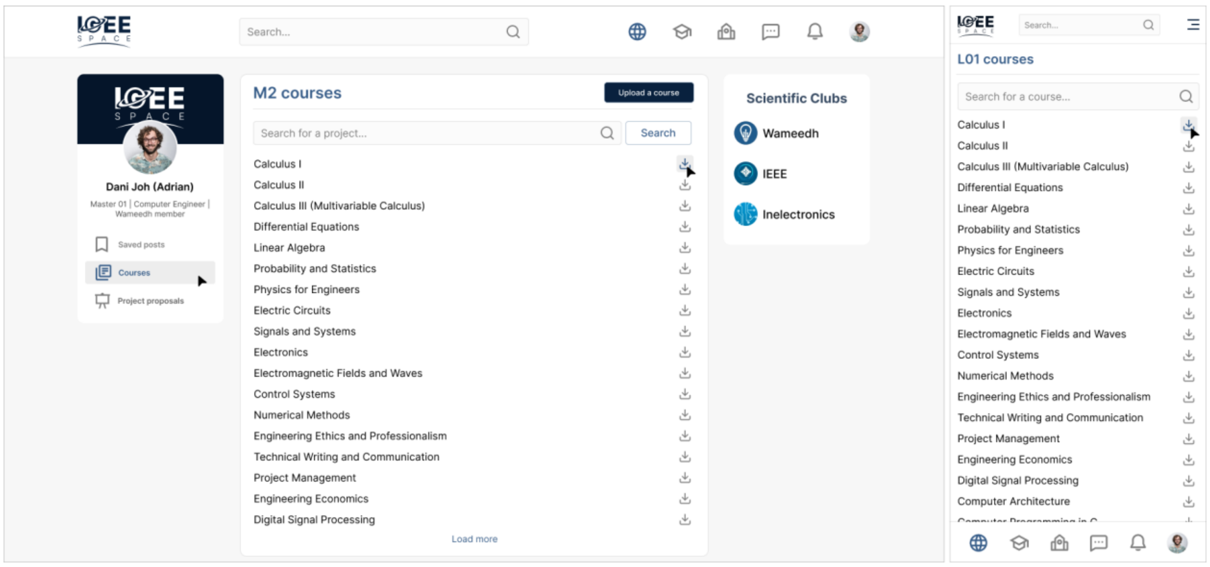
03:23 10/06/2023 Dani
Should we attend the extra section this Monday?

No! 😊	29
Yes! 😎	01

Figure 24: Chat page

4.2.6 Interface of Courses page

The course page provides the user with all the uploaded courses and resources along with a simple and clean UI. Students can view a specific year course like the M2 courses, search for a course title, download or upload a new course.



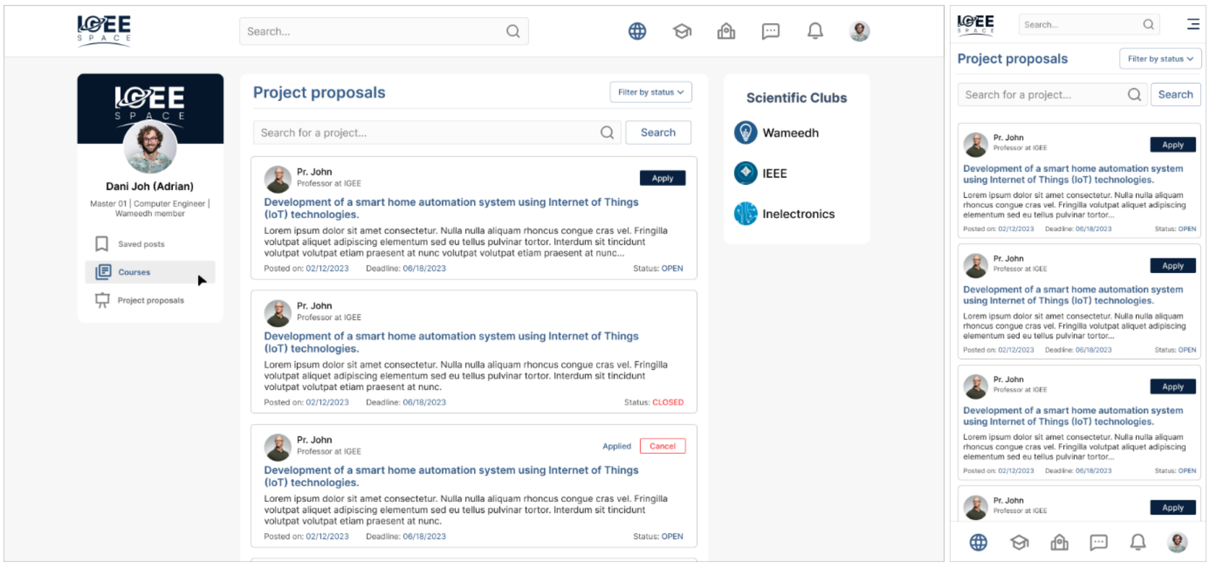
Calculus I
Calculus II
Calculus III (Multivariable Calculus)
Differential Equations
Linear Algebra
Probability and Statistics
Physics for Engineers
Electric Circuits
Signals and Systems
Electronics
Electromagnetic Fields and Waves
Control Systems
Numerical Methods
Engineering Ethics and Professionalism
Technical Writing and Communication
Project Management
Engineering Economics
Digital Signal Processing

Calculus I
Calculus II
Calculus III (Multivariable Calculus)
Differential Equations
Linear Algebra
Probability and Statistics
Physics for Engineers
Electric Circuits
Signals and Systems
Electronics
Electromagnetic Fields and Waves
Control Systems
Numerical Methods
Engineering Ethics and Professionalism
Technical Writing and Communication
Project Management
Engineering Economics
Digital Signal Processing
Computer Architecture

Figure 25: Courses page

4.2.7 Interface of Project proposals page

Teachers can create project proposals for M2 and L3 students and students can view these proposals through the project proposals page, this makes it much easier than sending emails to teachers and waiting for responses. This page shows the title, the abstract, the related dates, the status (OPEN or CLOSED) of each project, filtering, or searching for a project title. After students apply, teachers start selecting the qualified students based on their letters and even the profiles.



Dani Joh (Adrian)
Master 01 | Computer Engineer | Wameedh member

Project proposals

Pr. John
Professor at IGEE
Development of a smart home automation system using Internet of Things (IoT) technologies.
Lorem ipsum dolor sit amet consectetur. Nulla nulla aliquam rhoncus congue cras vel. Fringilla volutpat aliquet adipiscing elementum sed eu tellus pulvinar tortor. Interdum sit tincidunt volutpat volutpat etiam praesent at nunc volutpat volutpat etiam praesent at nunc...
Posted on: 02/12/2023 Deadline: 06/18/2023 Status: OPEN

Pr. John
Professor at IGEE
Development of a smart home automation system using Internet of Things (IoT) technologies.
Lorem ipsum dolor sit amet consectetur. Nulla nulla aliquam rhoncus congue cras vel. Fringilla volutpat aliquet adipiscing elementum sed eu tellus pulvinar tortor. Interdum sit tincidunt volutpat volutpat etiam praesent at nunc.
Posted on: 02/12/2023 Deadline: 06/18/2023 Status: CLOSED

Pr. John
Professor at IGEE
Development of a smart home automation system using Internet of Things (IoT) technologies.
Lorem ipsum dolor sit amet consectetur. Nulla nulla aliquam rhoncus congue cras vel. Fringilla volutpat aliquet adipiscing elementum sed eu tellus pulvinar tortor. Interdum sit tincidunt volutpat volutpat etiam praesent at nunc.
Posted on: 02/12/2023 Deadline: 06/18/2023 Status: OPEN

Project proposals

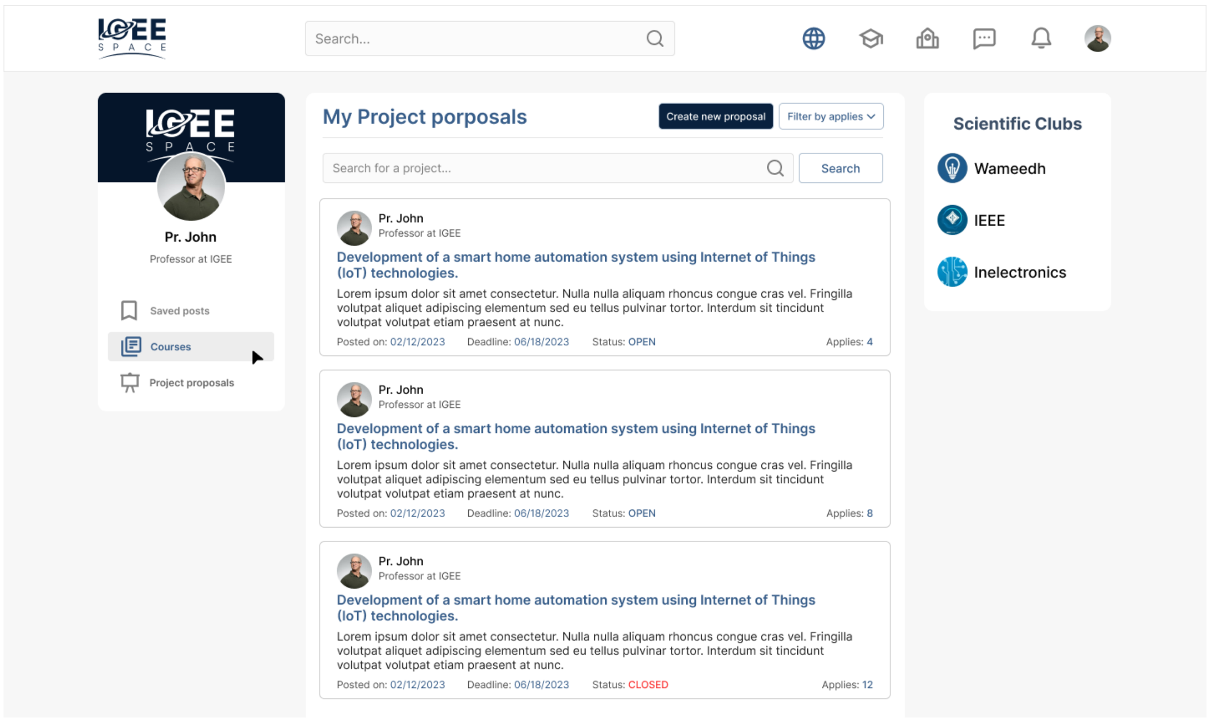
Pr. John
Professor at IGEE
Development of a smart home automation system using Internet of Things (IoT) technologies.
Lorem ipsum dolor sit amet consectetur. Nulla nulla aliquam rhoncus congue cras vel. Fringilla volutpat aliquet adipiscing elementum sed eu tellus pulvinar tortor...
Posted on: 02/12/2023 Deadline: 06/18/2023 Status: OPEN

Pr. John
Professor at IGEE
Development of a smart home automation system using Internet of Things (IoT) technologies.
Lorem ipsum dolor sit amet consectetur. Nulla nulla aliquam rhoncus congue cras vel. Fringilla volutpat aliquet adipiscing elementum sed eu tellus pulvinar tortor...
Posted on: 02/12/2023 Deadline: 06/18/2023 Status: OPEN

Pr. John
Professor at IGEE
Development of a smart home automation system using Internet of Things (IoT) technologies.
Lorem ipsum dolor sit amet consectetur. Nulla nulla aliquam rhoncus congue cras vel. Fringilla volutpat aliquet adipiscing elementum sed eu tellus pulvinar tortor...
Posted on: 02/12/2023 Deadline: 06/18/2023 Status: OPEN

Pr. John
Professor at IGEE

Figure 26: Student project proposals page



Pr. John
Professor at IGEE

My Project porposals

Pr. John
Professor at IGEE
Development of a smart home automation system using Internet of Things (IoT) technologies.
Lorem ipsum dolor sit amet consectetur. Nulla nulla aliquam rhoncus congue cras vel. Fringilla volutpat aliquet adipiscing elementum sed eu tellus pulvinar tortor. Interdum sit tincidunt volutpat volutpat etiam praesent at nunc.
Posted on: 02/12/2023 Deadline: 06/18/2023 Status: OPEN Applies: 4

Pr. John
Professor at IGEE
Development of a smart home automation system using Internet of Things (IoT) technologies.
Lorem ipsum dolor sit amet consectetur. Nulla nulla aliquam rhoncus congue cras vel. Fringilla volutpat aliquet adipiscing elementum sed eu tellus pulvinar tortor. Interdum sit tincidunt volutpat volutpat etiam praesent at nunc.
Posted on: 02/12/2023 Deadline: 06/18/2023 Status: OPEN Applies: 8

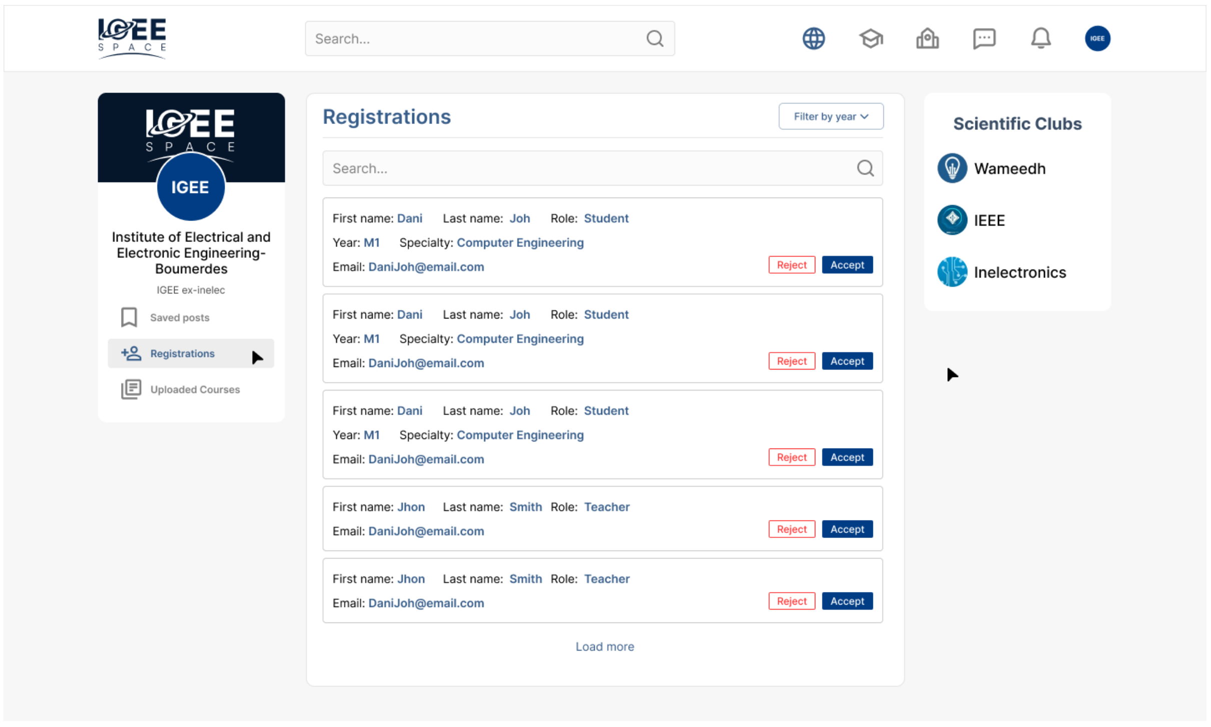
Pr. John
Professor at IGEE
Development of a smart home automation system using Internet of Things (IoT) technologies.
Lorem ipsum dolor sit amet consectetur. Nulla nulla aliquam rhoncus congue cras vel. Fringilla volutpat aliquet adipiscing elementum sed eu tellus pulvinar tortor. Interdum sit tincidunt volutpat volutpat etiam praesent at nunc.
Posted on: 02/12/2023 Deadline: 06/18/2023 Status: CLOSED Applies: 12

Figure 27: Teachers project proposals page

4.2.8 Interface of Registrations page

When teachers and students complete the registration process, the admin checks the form information of each user on the registrations page, then he accepts or rejects based on the validity of the information. The admin can register a group of students by uploading a

.csv file this will speed up the process of registration because students will gain access directly without registering manually.



Registrations

First name: Dani Last name: Joh Role: Student
Year: M1 Specialty: Computer Engineering
Email: DaniJoh@email.com Reject Accept

First name: Dani Last name: Joh Role: Student
Year: M1 Specialty: Computer Engineering
Email: DaniJoh@email.com Reject Accept

First name: Dani Last name: Joh Role: Student
Year: M1 Specialty: Computer Engineering
Email: DaniJoh@email.com Reject Accept

First name: Jhon Last name: Smith Role: Teacher
Email: DaniJoh@email.com Reject Accept

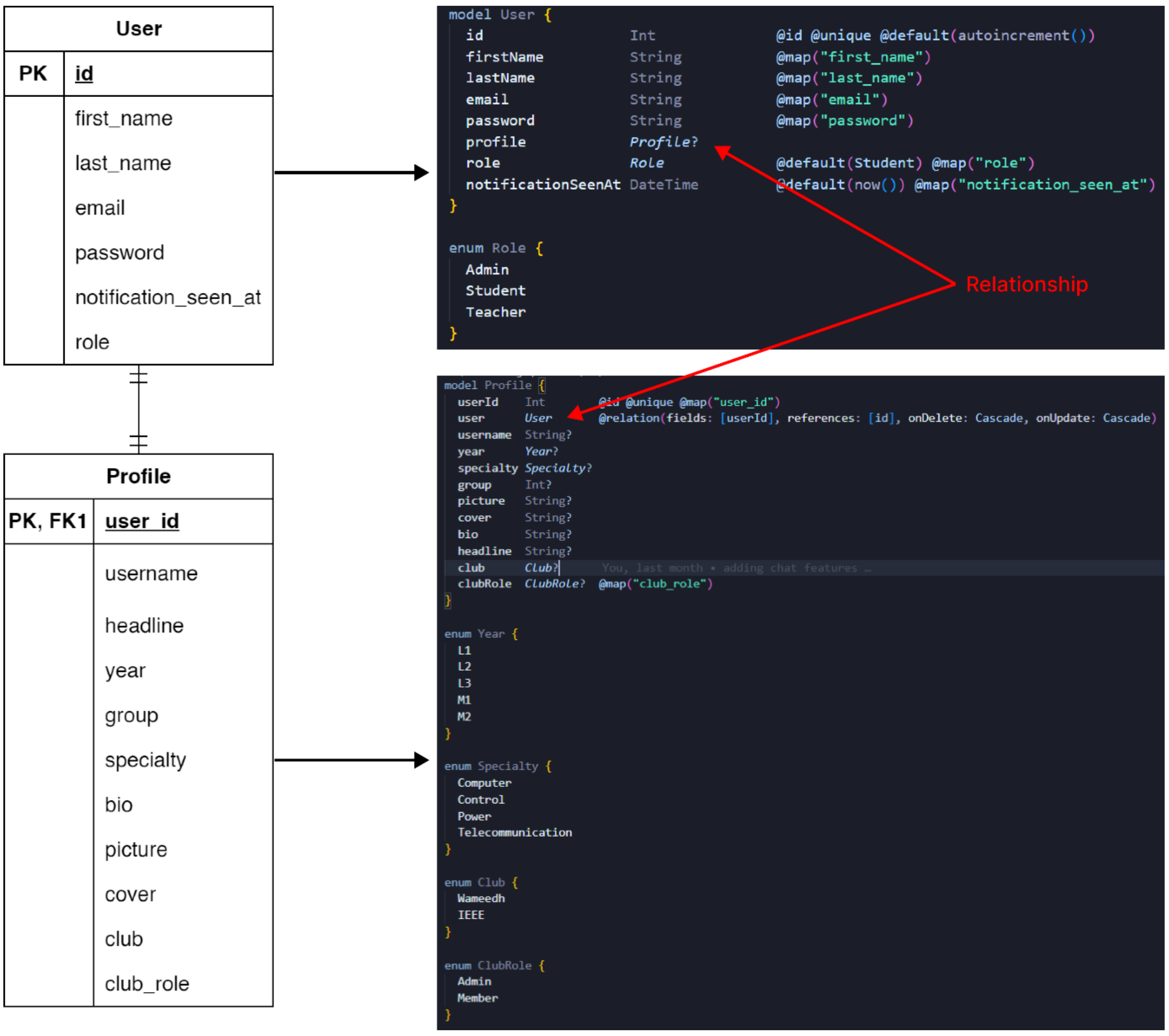
First name: Jhon Last name: Smith Role: Teacher
Email: DaniJoh@email.com Reject Accept

Figure 28: Registrations page

4.3 Code Implementation

4.3.1 Converting Database tables into Prisma code:

Prisma uses a term called model to describe a table. If we take the User and Profile tables, we can convert them into models as shown below.



```
model User {
  id                 Int          @id @unique @default(autoincrement())
  firstName          String       @map("first_name")
  lastName           String       @map("last_name")
  email              String       @map("email")
  password           String       @map("password")
  profile            Profile?
  role               Role         @default(Student) @map("role")
  notificationSeenAt DateTime     @default(now()) @map("notification_seen_at")
}

enum Role {
  Admin
  Student
  Teacher
}
```

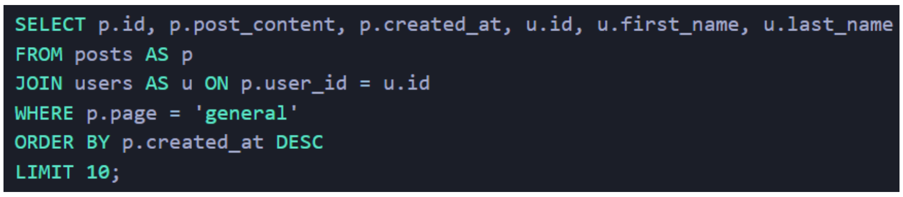
```
model Profile {
  userId     Int         @id @unique @map("user_id")
  user       User        @relation(fields: [userId], references: [id], onDelete: Cascade, onUpdate: Cascade)
  username   String?
  year       Year?
  specialty  Specialty?
  group      Int?
  picture    String?
  cover      String?
  bio        String?
  headline   String?
  club       Club?         You, last month • adding chat features …
  clubRole   ClubRole?   @map("club_role")
}

enum Year {
  L1
  L2
  L3
  M1
  M2
}

enum Specialty {
  Computer
  Control
  Power
  Telecommunication
}

enum Club {
  Wameedh
  IEEE
}

enum ClubRole {
  Admin
  Member
}
```

Figure 29: From SQL table to Prisma

When this mode gets executed, it will be converted into SQL syntax. Note that the foreign key is indicated using the “@relation” keyword. The “?” means that the property (column) can be null.

4.3.2 Querying from database using Prisma

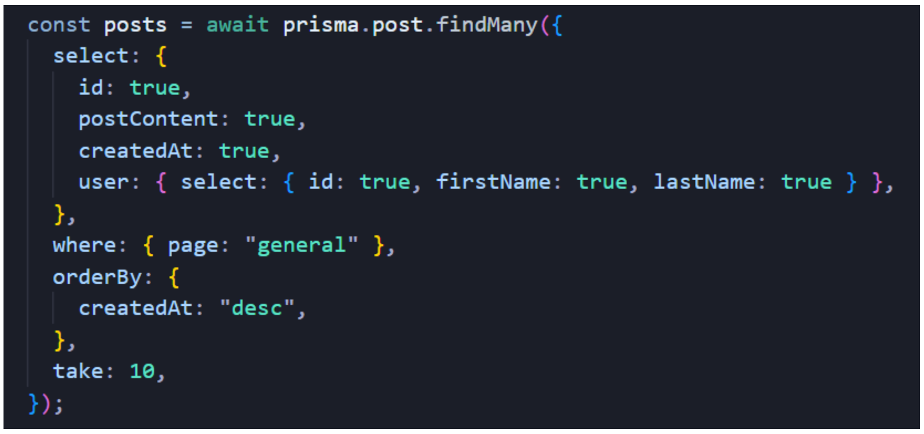
Prisma makes it easy to query from a database. Let’s say we want to query the first 10 posts of the general page ordered based on the created time; in SQL we would type the code like this:



```sql
SELECT p.id, p.post_content, p.created_at, u.id, u.first_name, u.last_name
FROM posts AS p
JOIN users AS u ON p.user_id = u.id
WHERE p.page = 'general'
ORDER BY p.created_at DESC
LIMIT 10;
```

Figure 30: Querying posts using SQL

In Prisma, we would write the code the like this:

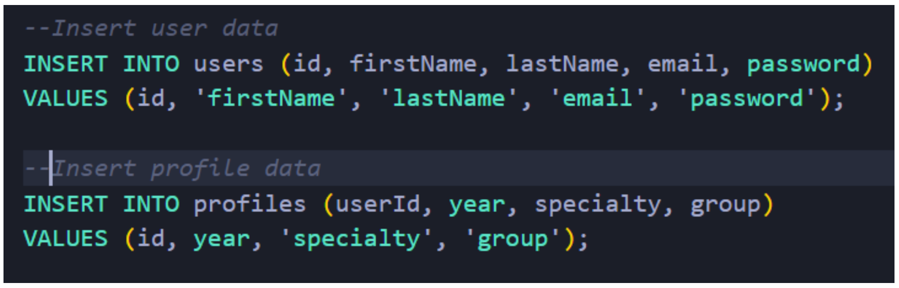


```js
const posts = await prisma.post.findMany({
  select: {
    id: true,
    postContent: true,
    createdAt: true,
    user: { select: { id: true, firstName: true, lastName: true } },
  },
  where: { page: "general" },
  orderBy: {
    createdAt: "desc",
  },
  take: 10,
});
```

Figure 31: Querying posts using Prisma

4.3.3 Inserting into database using Prisma

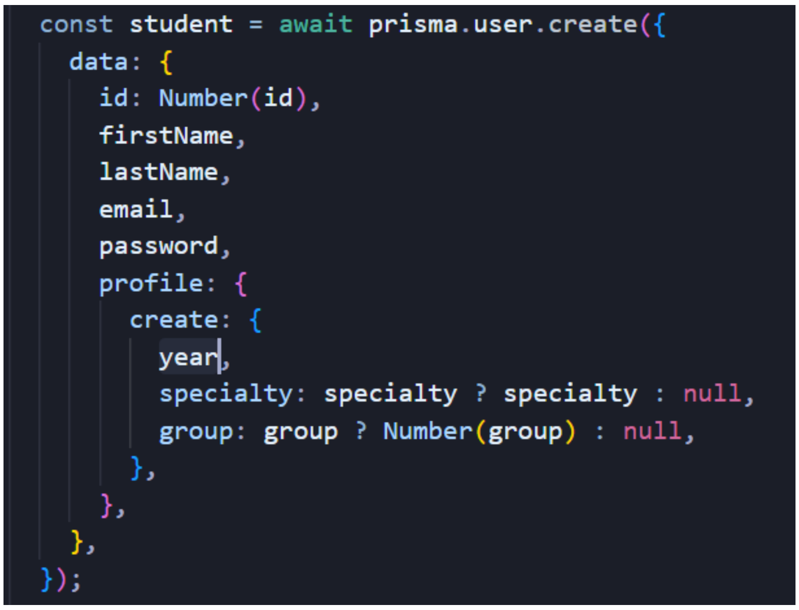
Let’s say we want to create a new student account when the user fills out the registration form and clicks register. The SQL code would look like this:



```sql
--Insert user data
INSERT INTO users (id, firstName, lastName, email, password)
VALUES (id, 'firstName', 'lastName', 'email', 'password');

--Insert profile data
INSERT INTO profiles (userId, year, specialty, group)
VALUES (id, year, 'specialty', 'group');
```

Figure 32: Registering a new student using SQL

However, using Prisma we would still work with our programming language, in this case it is TypeScript, and we would write the code like this:

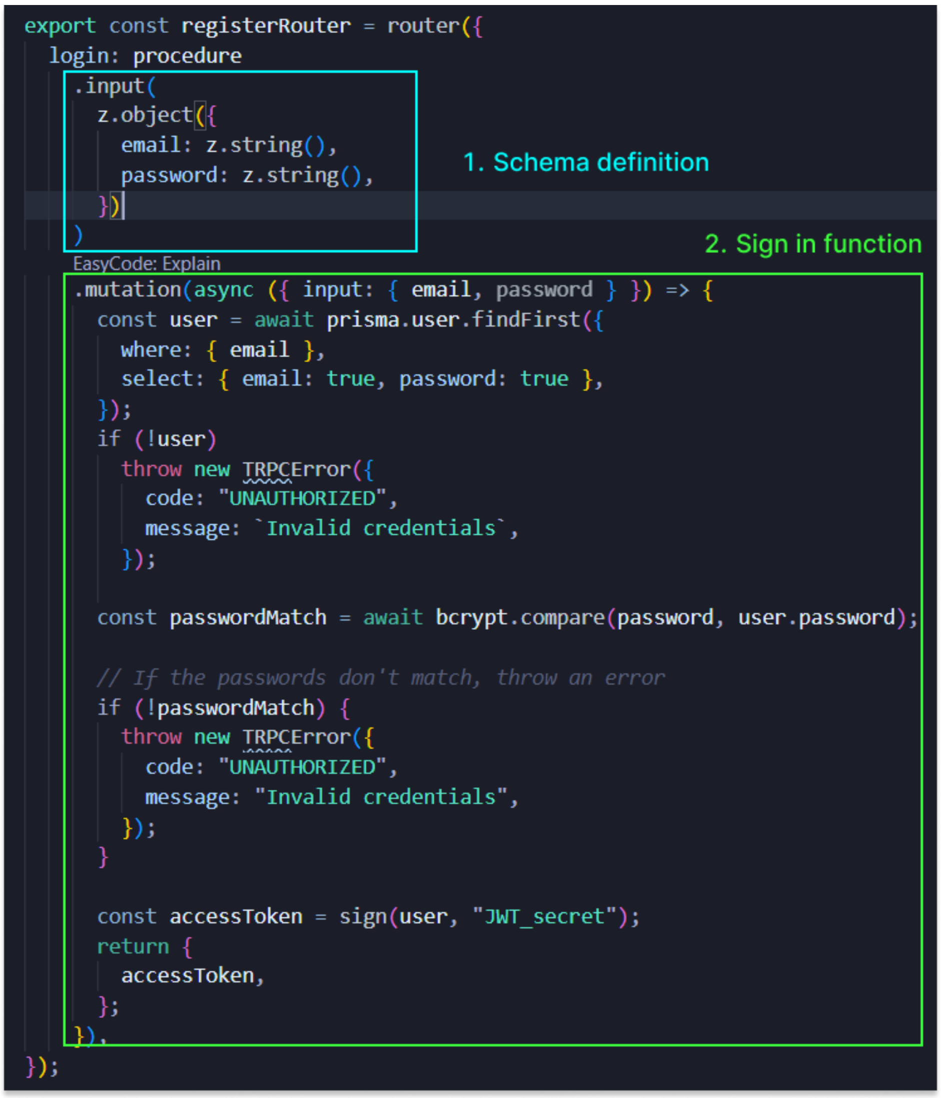


```
const student = await prisma.user.create({
  data: {
    id: Number(id),
    firstName,
    lastName,
    email,
    password,
    profile: {
      create: {
        year,
        specialty: specialty ? specialty : null,
        group: group ? Number(group) : null,
      },
    },
  },
});
```

Figure 33: Registering a new student using Prisma

4.3.4 Implementing the Sign in function using TRPC

The TRPC library helps make API calls easy and safe, in this case, we will see how we make an API request from the front-end to the back-end in order to sign in the user. First, we define back-end TRPC schema along with the sign-in function as shown in the following figure:

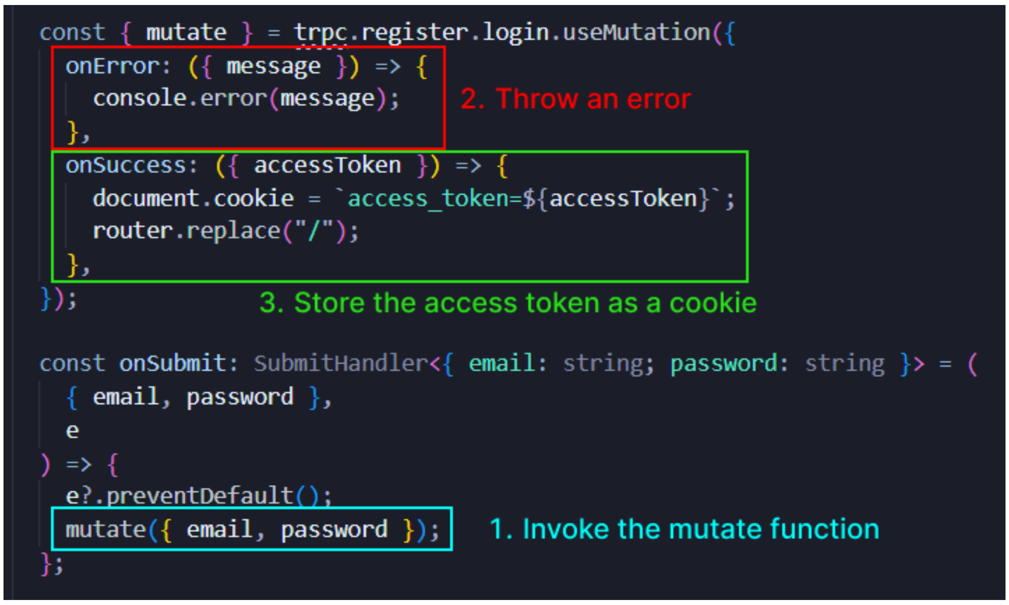


```
export const registerRouter = router({
  login: procedure
    .input(
      z.object({
        email: z.string(),
        password: z.string(),
      })
    )
    .mutation(async ({ input: { email, password } }) => {
      const user = await prisma.user.findFirst({
        where: { email },
        select: { email: true, password: true },
      });
      if (!user)
        throw new TRPCError({
          code: "UNAUTHORIZED",
          message: `Invalid credentials`,
        });

      const passwordMatch = await bcrypt.compare(password, user.password);

      // If the passwords don't match, throw an error
      if (!passwordMatch) {
        throw new TRPCError({
          code: "UNAUTHORIZED",
          message: "Invalid credentials",
        });
      }

      const accessToken = sign(user, "JWT_secret");
      return {
        accessToken,
      };
    }),
});
```

Figure 34: The back-end side of TRPC

The schema definition forces the request coming from the front-end to include both email and password, if one of them is missing an error would be thrown. The mutation function is where we put our logic function, and in this case, it is the sign-in function. After we verify that the credentials are valid, we respond with an access token that will be used in the authorization part.

Moving on to the front-end part, the following figure showcases how it's implemented:



```
const { mutate } = trpc.register.login.useMutation({
  onError: ({ message }) => {
    console.error(message);
  },
  onSuccess: ({ accessToken }) => {
    document.cookie = `access_token=${accessToken}`;
    router.replace("/");
  },
});

const onSubmit: SubmitHandler<{ email: string; password: string }> = (
  { email, password },
  e
) => {
  e?.preventDefault();
  mutate({ email, password });
};
```

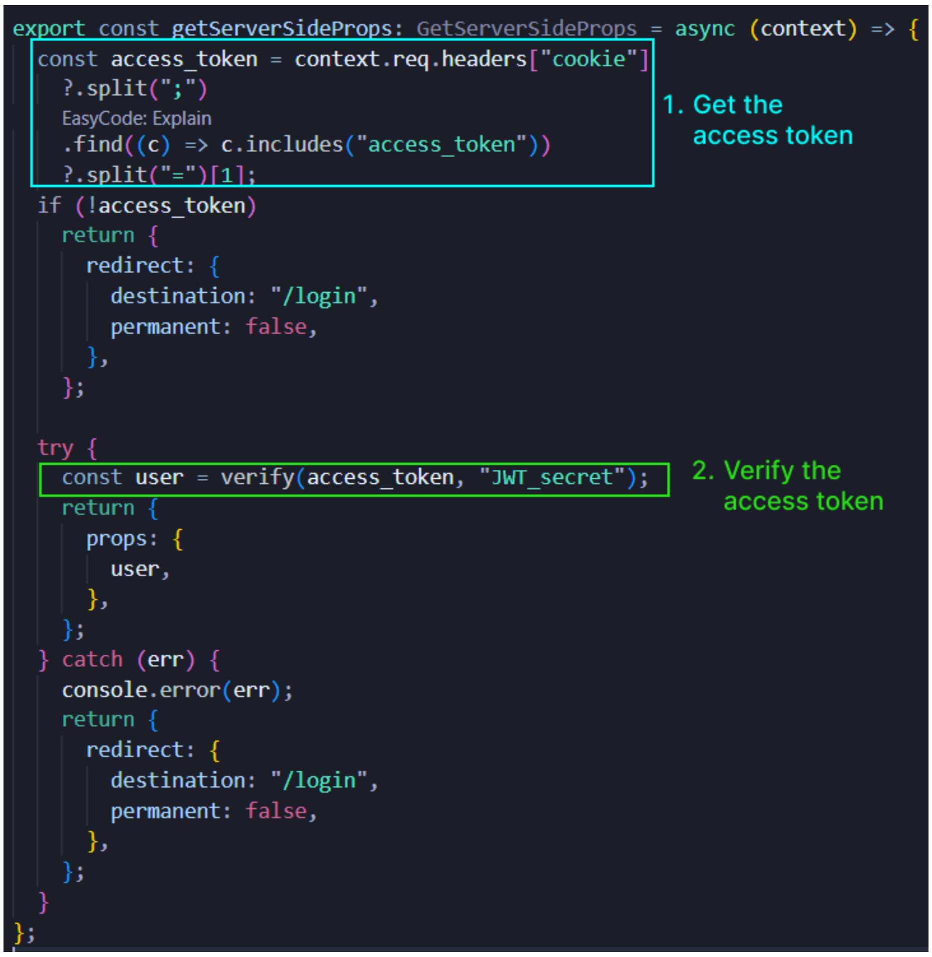
2. Throw an error
3. Store the access token as a cookie
1. Invoke the mutate function

Figure 35: The front-end side of TRPC

When the user enters his email and password and then clicks the "sign-in" button, the mutate function gets invoked. This sends an API request to the back-end and waits for the response. If the server responds with "UNAUTHRIZED", the "onError" method gets called and it prints out the error. However, If the server responds with the access token, the "onSuccess" method gets called and here we save the access token in the browser as a cookie in order to use it in the authorization part.

4.3.5 Handling the Authorization

To handle the authorization, we use a function provided by Nextjs called "getServerSideProps"; when this function is imported, the page turns into an SSR mode, meaning we get to use the server properties.



```
export const getServerSideProps: GetServerSideProps = async (context) => {
  const access_token = context.req.headers["cookie"]
    ?.split(";")
    EasyCode: Explain
    .find((c) => c.includes("access_token"))
    ?.split("=")[1];
  if (!access_token)
    return {
      redirect: {
        destination: "/login",
        permanent: false,
      },
    };

  try {
    const user = verify(access_token, "JWT_secret");
    return {
      props: {
        user,
      },
    };
  } catch (err) {
    console.error(err);
    return {
      redirect: {
        destination: "/login",
        permanent: false,
      },
    };
  }
};
```

Figure 36: Handling the authorization using Nextjs SSR

Every time the user requests the page, this function gets executed. First, we grab the access token out of the request header then we check if the token does not exist, we redirect the user to the login page. If the token exits, we check if it’s valid (not expired or modified) and then redirect the user to the requested page, else redirect to the login page.

4.4 Conclusion

In conclusion, this chapter has covered the implementation phase of the platform. We provided a comprehensive presentation of the application, highlighting its properties, offering user interfaces for visualization, and explaining some of the important coding functions. By transitioning from design to implementation, we have taken a significant step towards bringing the software system to life, ready for deployment and actual usage.

General Conclusion

This report outlines the process of designing and developing a web application that facilitates student communication and collaboration. The project idea came out of the issues that students face when using social media for educational purposes.

The web-based Student Communication Platform presented in this project is an all-in-one solution tailored specifically for IGEE educational environments. It aims to create an optimal space for students to focus on their studies, collaborate effectively, and cultivate a rich learning experience through meaningful interactions and shared knowledge.

Designing and implementing this project was an incredible experience that provided me with an opportunity to explore new concepts and enhance my knowledge and skills in the field of programming.

Future works

After successfully completing the initial version of the Student Communication Platform (SCP), the next version aims to incorporate video meetings directly within the platform, eliminating the need for external meeting applications. Additionally, the focus will expand to include the administration ecosystem, streamlining and automating various paperwork processes directly through the platform. These enhancements will significantly reduce time wastage and improve overall efficiency.

References

[1] https://www.britannica.com/topic/Web-application. Consulted (14-06-2023)

[2] https://geekflare.com/single-page-applications. Consulted (14-06-2023)

[3] https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. Consulted (14-06-2023)

[4] https://www.geeksforgeeks.org/frontend-vs-backend. Consulted (14-06-2023)

[5] https://www.techtarget.com/searchapparchitecture/definition/state-management. (18-06-2023)

[6] https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless. Consulted (14-06-2023)

[7] https://en.wikipedia.org/wiki/Software_framework. Consulted (14-06-2023)

[8] https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. Consulted (14-06-2023)

[9] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. Consulted (14-06-2023)

[10] https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL. Consulted (14-06-2023)

[11] https://www.techtarget.com/searchdatamanagement/definition/database. Consulted (14-06-2023)

[12] https://www.geeksforgeeks.org/dbms. Consulted (14-06-2023)

[13] https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools. Consulted (15-06-2023)

[14] https://aws.amazon.com/what-is/api. Consulted (15-06-2023)

[15] https://www.okta.com/blog/2019/03/what-are-salted-passwords-and-password-hashing. Consulted (15-06-2023)

[16] https://en.wikipedia.org/wiki/HTML. Consulted (15-06-2023)

[17] https://en.wikipedia.org/wiki/CSS. Consulted (15-06-2023)

[18] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. Consulted (15-06-2023)

[19] N. Black, "Boris Cherny on TypeScript," in IEEE Software, vol. 37, no. 2, pp. 98-100, March-April 2020, doi: 10.1109/MS.2019.2958155.

[20] https://en.wikipedia.org/wiki/Visual_Studio_Code. Consulted (15-06-2023)

[21] Sanchit Aggarwal et al. International Journal of Recent Research Aspects ISSN: 2349-7688, Vol. 5, Issue 1, March 2018, pp. 133-137.

[22] https://nextjs.org/docs. Consulted (15-06-2023)

[23] https://blog.hubspot.com/website/what-is-tailwind-css. Consulted (15-06-2023)

[24] https://blog.logrocket.com/firebase-vs-supabase-choosing-right-tool-project. Consulted (15-06-2023)

[25] https://www.prisma.io/docs/concepts/overview/what-is-prisma. Consulted (16-06-2023)

[26] https://betterprogramming.pub/build-full-stack-typescript-applications-using-trpc-eef8588979d8?gi=ac57e1660cad. Consulted (16-06-2023)

[27] https://www.frontendmag.com/insights/zustand-vs-redux-comparison/#What_is_Zustand. Consulted (16-06-2023)

[28] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[29] https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma. Consulted (16-06-2023)

[30] https://www.computerhope.com/jargon/d/drawio.htm. Consulted (16-06-2023)

[31] https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language. Consulted (16-06-2023)

[32] https://www.lucidchart.com/pages/er-diagrams. Consulted (18-06-2023)