

Bachelor's Thesis



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Computer Science**

System for Communication Between Teachers and Students

Nikita Lepikhin

Supervisor: Ing. Božena Mannová, Ph.D.

Field of study: Open Informatics

Subfield: Software

January 2023

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lepikhin** Jméno: **Nikita** Osobní číslo: **491851**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra počítačů**
Studijní program: **Otevřená informatika**
Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro komunikaci mezi učiteli a studenty

Název bakalářské práce anglicky:

System for communication between teachers and students

Pokyny pro vypracování:

Analyzujte způsoby komunikace mezi studenty a učiteli. Vyberte některá aktuálně existující řešení a tato řešení analyzujte a porovnejte. Vyhodnoťte i roli anonymity v takové komunikaci. Na základě provedené analýzy navrhnete vlastní řešení, které by eliminovalo nevýhody posuzovaných aplikací a umožnilo anonymní komunikaci. Zvolte architekturu aplikace a vyberte vhodné technologie pro implementaci. Implementujte navržené řešení a rozsáhle jej otestujte. Využijte vhodných prostředků softwarového inženýrství.

Seznam doporučené literatury:

- [1] Roger S. Pressmann Bruce Maxim: Software Engineering: A Practitioner's Approach , ISBN-10: 9780078022128
- [2] <https://blog.genial.ly/en/techniques-online-communication-students-and-teachers/>

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Božena Mannová, Ph.D. kabinet výuky informatiky FEL

Jméno a pracoviště druhého(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.02.2022**

Termín odevzdání bakalářské práce: **10.01.2023**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Božena Mannová, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta

Acknowledgements

The completion of this thesis would not have been possible without the guidance, and invaluable feedback of Ing. Božena Mannová, Ph.D., my advisor. I would also like to extend my thanks to my family for their encouragement and unwavering support throughout my studies.

Declaration

I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

Prague, December 20, 2022

Prohlašuji, že jsem bakalářskou práci na téma Systém pro komunikaci mezi učiteli a studenty zpracoval sám. Veškeré prameny a zdroje informací, které jsem použil k sepsání této práce, byly citovány a jsou uvedeny v seznamu použitých pramenů a literatury.

V Praze, 20. prosince 2022

Abstract

This bachelor's thesis focuses on the development of a web application for anonymous communication between teachers and students. The application allows teachers and students to communicate via posts and comments created in different channels. The work begins by analyzing the problem of such communication and the existing solutions. The analysis later serves as a foundation for system design. Following is a thorough description of implementation details. The end result is a tested and deployed web application for anonymous communication in the academic setting.

Keywords: web application, anonymous communication, communication between teachers and students, React, Single Page App, REST, NestJS, Prisma

Supervisor: Ing. Božena Mannová,
Ph.D.
Kabinet výuky informatiky ČVUT FEL,
Praha, Resslova 9, E-430

Abstrakt

Tato bakalářská práce se zaměřuje na vývoj webové aplikace pro anonymní komunikaci mezi učiteli a studenty. Aplikace umožňuje učitelům a studentům komunikovat prostřednictvím příspěvků a komentářů vytvořených v různých kanálech. Práce začíná analýzou anonymní komunikace a existujících řešení. Analýza později slouží jako základ pro návrh systému. Následuje podrobný popis implementačních detailů. Výsledkem práce je otestovaná a nasazená webová aplikace pro anonymní komunikaci v akademickém prostředí.

Klíčová slova: webová aplikace, anonymní komunikace, komunikace mezi učiteli a studenty, React, jednostránková webová aplikace, REST, NestJS, Prisma

Překlad názvu: Systém pro komunikaci mezi učiteli a studenty

Contents

1 Introduction	1	4.4.6 UI Components	44
1.1 Motivation	1	4.4.7 Switching Color Scheme	45
1.2 Goals	1	4.5 Deployment	45
2 Analysis	3	5 Testing	47
2.1 Problem Analysis	3	5.1 Automatic Testing	47
2.2 Analysis of Existing Solutions . . .	4	5.2 User Testing	47
2.3 Requirements Specification	5	5.2.1 Student A	48
2.3.1 Domain Concept Model	5	5.2.2 Student B	48
2.3.2 Functional Requirements	7	5.2.3 Student C	49
2.3.3 Non-functional Requirements . . .	10	5.2.4 Teacher A	49
2.3.4 Use Case Analysis	11	5.2.5 Teacher B	50
3 System Design	19	5.2.6 Teacher C	50
3.1 Application Architecture	19	5.2.7 Results of User Testing	51
3.2 Database Design	20	6 Conclusion	53
3.3 Back End Design	21	A Database Structure	55
3.3.1 Back End Structure	21	B Example of End-To-End Test	57
3.3.2 REST API	21	C User Test Scenarios	59
3.3.3 Authentication	22	C.1 Teacher Test	59
3.3.4 Authorization	26	C.1.1 Part 1. Signing up as a	
3.3.5 Voting System	27	teacher	59
3.3.6 Anonymity	28	C.1.2 Part 2. Using the application	
3.4 Front End Design	28	as a teacher	59
3.4.1 Wireframes	29	C.1.3 Part 3. Testing Feedback . . .	61
4 Implementation	33	C.2 Student Test	61
4.1 TypeScript	33	C.2.1 Part 1. Signing up as a	
4.2 Persistence	34	student	61
4.2.1 Database Engine	34	C.2.2 Part 2. Using the application	
4.2.2 Object-Relational Mapping . . .	34	as a student	61
4.2.3 Database Migrations	34	C.2.3 Part 3. Testing Feedback . . .	62
4.2.4 Prisma ORM	35	D Front End Wireframes	63
4.2.5 Custom Queries	35	E Switching Color Scheme	69
4.3 Back End	36	F Application Demonstration	71
4.3.1 Back-End Framework	36	G Bibliography	77
4.3.2 API Documentation	36		
4.3.3 Using HTTPS in Development . .	37		
4.3.4 Authentication	38		
4.3.5 Authorization	40		
4.3.6 Modular Structure	41		
4.4 Front End	42		
4.4.1 Front-End Library	42		
4.4.2 Using HTTPS in Development . .	42		
4.4.3 State Management and Data			
Fetching	42		
4.4.4 Navigation	43		
4.4.5 Handling Forms	43		

Figures

2.1 Domain Concept Model	6	F.4 Requests Page (top) - Schools Page (bottom)	74
2.2 Use Case Actors Diagram	11	F.5 Student Account Page (top) - Teacher Public Profile Page (bottom)	75
2.3 System Management-Related Use Cases	12		
2.4 Account-Related Use Cases	13		
2.5 Channels-Related Use Cases	15		
2.6 Posts-Related Use Cases	16		
2.7 Comments-Related Use Cases	17		
3.1 Data Flow in an MVC-Based Application	19		
3.2 Components Comprising the Application	20		
3.3 Back End Layered Architecture	21		
3.4 Handling a JWT Token	25		
3.5 Generic Authenticated Request Sequence	26		
3.6 Authentication Flow Sequence	27		
3.7 Refreshing the Access Token Sequence	28		
3.8 Logging Out Sequence	28		
3.9 Verification State	29		
3.10 Vote State	30		
A.1 Database Structure	56		
D.1 Login Page Wireframe (top) - Signup Page Wireframe (bottom)	63		
D.2 Schools Page Wireframe (top) - Faculties Page Wireframe (bottom)	64		
D.3 Requests Page Wireframe (top) - Account Page Wireframe (bottom)	65		
D.4 Feed Page Wireframe (top) - Channel Page Wireframe (bottom)	66		
D.5 Post Page Wireframe (top) - Comments Page Wireframe (bottom)	67		
E.1 Light Mode (top) - Dark Mode (bottom)	69		
F.1 Login Page (top) - Signup Page (bottom)	71		
F.2 Feed Page (top) - Channel Page (bottom)	72		
F.3 Post Page (top) - Comment Page (bottom)	73		

Chapter 1

Introduction

1.1 Motivation

Remote studying has increased in popularity over recent years. Since communication with teachers is a vital part of education, it is necessary to optimize it so that students are more engaged in studying. This would help them achieve higher results and get a deeper understanding of the material. It is believed that anonymity in communication with teachers has a positive impact on absorbing knowledge. I aim to create a modern and reliable solution that would respect students' anonymity when communicating with teachers.

1.2 Goals

Following are the goals of this bachelor thesis.

- Analyze the means of communication between students and teachers, the role of anonymity in it, and the currently existing solutions for such communication.
- Conduct software analysis and design a new application that would eliminate most downsides of the existing ones.
- Implement the designed solution and test it.
- Deploy the application.

Chapter 2

Analysis

2.1 Problem Analysis

The important role that teachers play in the educational process cannot be underestimated. With the emerging need to perform the required activities in an online space, students may find it difficult to communicate with teachers. According to the examined research [1], the following issues arise.

- Students may feel that they are being judged or unnecessarily assessed by other participants. This puts extra pressure on them and mentally restrains them.
- Students may feel that they must perform at a certain level because of their social status, academic achievements, or knowledge proficiency.
- Students may be more reluctant to actively participate in class since they are afraid to make mistakes that can be associated with them for longer periods.

The following research [1][2] suggests that peer pressure may also be an influential factor in students' decreased will to participate. Most students have reported the following things affecting them:

- unwillingness to display the discrepancy gap in their knowledge and that of their peers as that could harm their relationships
- being concerned about hurting someone by making mistakes
- feeling anxious from receiving potentially low evaluations from their peers
- being distracted and feeling less confident when their real name is on display
- condemning oneself for making mistakes publicly
- refraining from accepting certain educational opportunities, depending on the school culture.

The main downside, in my opinion, is that all communication is centered around presentations. Slido can be integrated with other communication tools, such as Microsoft Teams, however, there are no discussion boards, nor private chats available.

Mentimeter [9] is similar to the previous application. It shares a comparable set of features by allowing users to connect to presentations using their phones. Participants can then engage in polls with their responses anonymously shared and visualized. However, it also does not offer forums nor chats for communication.

Having analyzed multiple solutions, I have concluded that most students would prefer staying anonymous and having a platform to communicate freely without any restrictions imposed on them. Users do need to have their questions separated by subjects and/or topics. Since there already exist several applications for voting purposes, I have decided to implement a forum-based website where participants would discuss education-related subjects while having their identity protected at all times.

■ 2.3 Requirements Specification

■ 2.3.1 Domain Concept Model

Domain concept model (Fig. 2.1) is a high-level model that defines entities and the relationships between them. I have identified the following entities.

- User - a generic user entity that contains basic data associated with that user. Each user is assigned a maximum of one role (Role enumeration). User accounts can be active or deleted (Status enumeration).
- Teacher - a user entity that holds data specific to all teachers. Each teacher is connected to one instance of the user entity.
- Representative - a user entity that holds data specific to all representatives. Each representative is connected to one instance of the user entity.
- School - an entity representing a school or university registered in the system. All representatives are related to some school that they manage. All teachers are also related to one school that they work at.
- Faculty - an entity representing a faculty or some other unit of a school. All schools have at least one faculty associated with them. Each teacher works at some faculty.
- Channel - an entity that represents a communication channel where users would be able to create posts.
- Post - an entity that represents a post created in a channel. Each post is located in some channel and cannot exist by itself. In addition to being

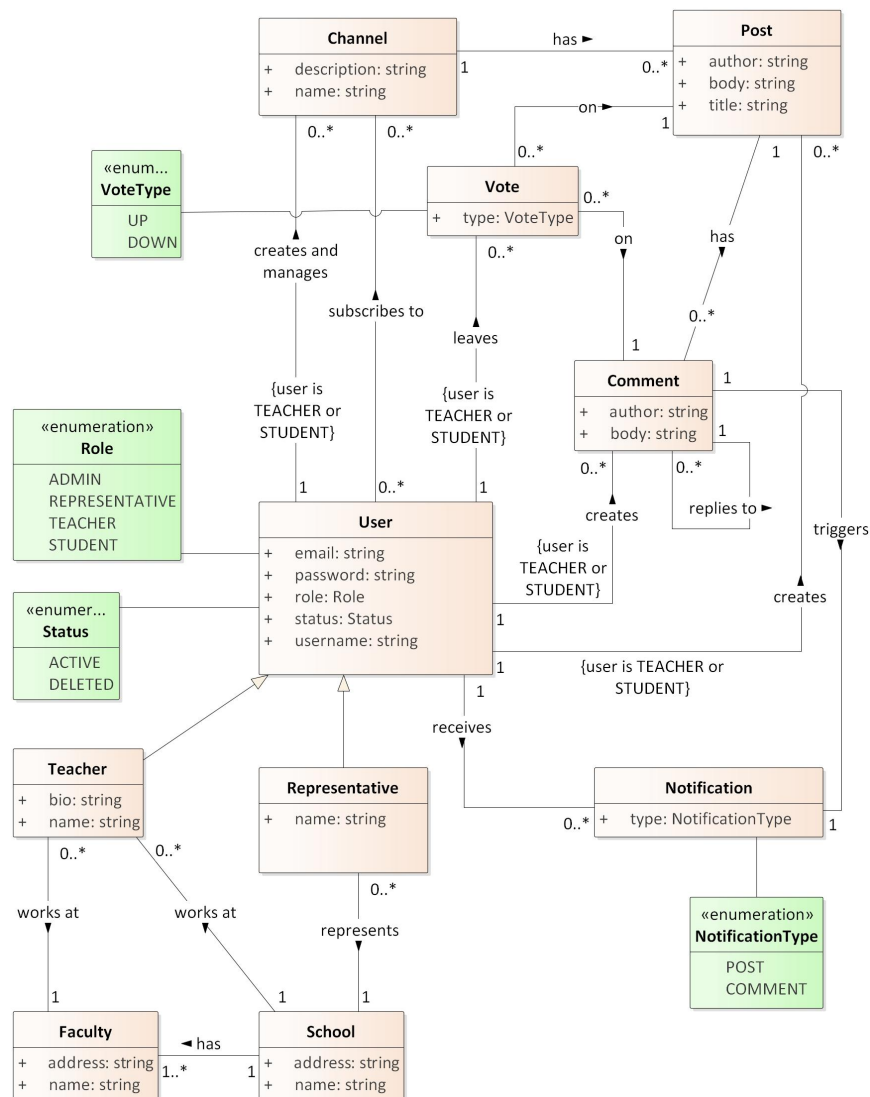


Figure 2.1: Domain Concept Model

linked to some users, post stores some user identifiers that are never updated.

- **Comment** - an entity that represents a comment created either in response to a post or another comment. Comments cannot exist by themselves. Similar to posts, comments store a user identifier that is never updated.
- **Vote** - an entity representing a user vote on a post or a comment. Votes can be either positive or negative (VoteType enumeration).
- **Notification** - an entity representing a notification sent to the author of the post or comment (NotificationType enumeration) when their post or comment receives a reply. Notification is linked to the author and comment that has triggered it.

■ 2.3.2 Functional Requirements

According to the definition [10], functional requirements form the foundation for tasks that have to be implemented to achieve the desired functionality. Following is a list of such requirements divided by user types. The requirements will be specified for admins, representatives, teachers, students, and guests. In addition to the user types listed in the domain concept model, it is necessary to also consider guest users. Such users are unauthenticated persons, or the ones who have never registered in the application.

■ Functional Requirements for Guests

■ **FR001 Creating an account**

As a guest, I need to be able to sign up, so that I can use the service.

■ **FR002 Logging into the account**

As a guest, I need to be able to log into my account created beforehand, so that I can continue using the service.

■ Functional Requirement Common for Registered Users

■ **FR003 Deleting an account**

As a registered user, I need to be able to delete my account, so that my information does not remain on the server.

■ **FR004 Updating the email**

As a registered user, I need to be able to update my email address, so that I can log back in with the new email.

■ **FR005 Changing the password**

As a registered user, I need to be able to change my password, so that my account data is secure.

■ **FR006 Searching for channels**

As a registered user, I need to be able to search for channels, so that I can see which channels are available in the system.

■ **FR007 Browsing channels**

As a registered user, I need to be able to browse channels, so that I can see the posts created in them.

■ **FR008 Browsing posts**

As a registered user, I need to be able to browse posts, so that I can see what other users have shared.

■ **FR009 Browsing comments**

As a registered user, I need to be able to browse comments, so that I can see what other users have said.

■ **Functional Requirements for Admins**

■ **FR010 Creating a school**

As a an admin, I need to be able to create a new school, so that representatives and teachers working there can register in the system.

■ **FR011 Browsing schools**

As an admin, I need to be able to browse the list of schools, so that I can see which schools are available in the system.

■ **FR012 Updating a school**

As an admin, I need to be able to update a school, so that the information about that schools remains up-to-date.

■ **FR013 Deleting a school**

As an admin, I need to be able to delete a school, so that there are no unused school entries in the system.

■ **FR014 Verifying representatives**

As an admin, I need to be able to verify new accounts created by representatives, so that they can start using the application.

■ **FR015 Verifying teachers**

As an admin, I need to be able to verify new accounts created by teachers, so that they can start using the application.

■ **Functional Requirements for Representatives**

■ **FR017 Verifying teachers from the same school**

As a representative, I need to be able to verify new accounts created by teachers from the school that I work at, so that they can start using the application.

■ **FR018 Changing the name**

As a representative, I need to be able to change my profile name, so that the information about me remains up-to-date.

■ **Functional Requirements Common for Admins and Representatives**

■ **FR019 Creating a faculty**

As an admin or a representative, I need to be able to create a new faculty assigned to a school, so that teachers working there can register in the system.

■ **FR020 Browsing school faculties**

As an admin or a representative, I need to be able to browse the list of faculties assigned to a school, so that I can see which faculties are available in the system.

- **FR021 Updating a faculty**

As an admin or a representative, I need to be able to update a faculty, so that the information about that faculty remains up-to-date.

- **FR022 Deleting a faculty**

As an admin or a representative, I need to be able to delete a faculty, so that there are no unused faculty entries in the system.

- **Functional Requirements for Teachers**

- **FR023 Changing the bio**

As a teacher, I need to be able to update my bio, so that the information about me remains up-to-date.

- **FR024 Changing the name**

As a teacher, I need to be able to update my name, so that the information about me remains up-to-date.

- **Functional Requirements Common for Teachers and Students**

- **FR025 Creating a channel**

As a verified teacher or a student, I need to be able to create a new channel so that other users have a place where to write posts.

- **FR026 Updating a channel**

As a verified teacher or a student that owns a channel, I need to be able to update its name, description and ID so that the information remains up-to-date.

- **FR027 Deleting a channel**

As a verified teacher or a student that owns a channel, I need to be able to delete such channels so that the information does not remain on the server.

- **FR028 Joining channels**

As a verified teacher or a student, I need to be able to join channels so that posts created in them are shown to me in the feed.

- **FR029 Sharing channels**

As a teacher or a student, I need to be able to share channels so that other users can quickly access them.

- **FR030 Creating a post**

As a verified teacher or a student, I need to be able to create a post so that I can share something with other users.

- **FR031 Updating a post**

As a verified teacher or a student, I need to be able to update posts created by me so that I do not have to create a new one in case of a mistake.

■ **FR032 Deleting a post**

As a verified teacher or a student, I need to be able to delete posts created by me so that the information does not remain on the server.

■ **FR033 Voting on a post**

As a verified teacher or a student, I need to be able to vote on a post so that other users see how popular it is.

■ **FR034 Sharing a post**

As a teacher or a student, I need to be able to share posts so that other users can quickly access them.

■ **FR035 Creating a comment**

As a verified teacher or a student, I need to be able to create comments under a post, or to reply to other comments so that other users know what I think.

■ **FR036 Editing a comment**

As a verified teacher or a student, I need to be able to edit comments created by me so that I do not have to create a new one in case of a mistake.

■ **FR037 Deleting a comment**

As a verified teacher or a student, I need to be able to delete comments created by me so that the information does not remain on the server.

■ **FR038 Voting on a comment**

As a verified teacher or a student, I need to be able to vote on a comment so that other users see how popular it is.

■ **FR039 Sharing a comment**

As a teacher or a student, I need to be able to share comments so that other users can quickly access them.

■ **FR040 Receiving notification about replies to my comments**

As a verified teacher or a student, I need to be able to receive notifications when other users reply to my comments so that I can react to them.

■ **FR041 Receiving notification about comments left under posts created by me**

As a verified teacher or a student, I need to be able to receive notifications when other users leave comments under posts created by me, so that I can react to them.

■ **2.3.3 Non-functional Requirements**

The definition [10] states that non-functional requirements define a set of attributes that an application should possess. After considering the characteristics applicable to all modern apps, I have come up with the following requirements.

- The application must be secure enough, that user data is protected and guarded against unauthorized access.
- The API should be logically structured and intuitive to use.
- The front end should have a modern, minimalist user interface, be responsive and follow a mobile-friendly approach.
- Deployment should occur automatically and should not result in any downtime for users.

■ 2.3.4 Use Case Analysis

A use case is a written description of how users will perform tasks on the website [11]. Before specifying the use cases, it is necessary to list the actors performing them. The definition of user types in the domain concept modal (Fig. 2.1) has led to the following actors being created (Fig. 2.2).

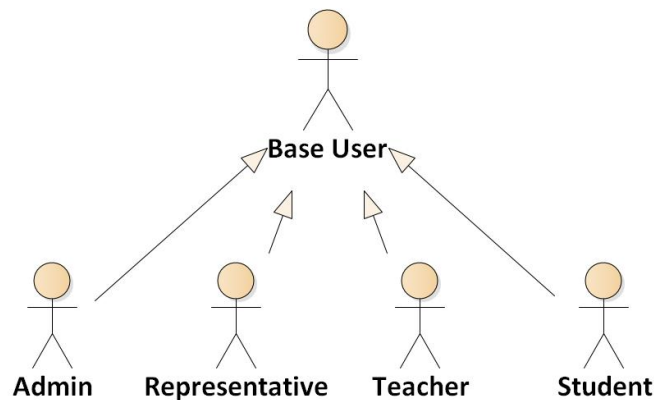


Figure 2.2: Use Case Actors Diagram

Based on the aforementioned list of functional requirements I have created use case diagrams that showcase users' interaction with the application. Use cases are divided into multiple diagrams based on related functionality.

■ System-Management Related Use Cases

Following is a list of use cases that are related to system management (Fig. 2.3).

- **UC001 Create a school**
Admins create a school so that teachers and representatives working there can register.
- **UC002 View all schools**
Admins view all schools so that they know which schools have already been added.

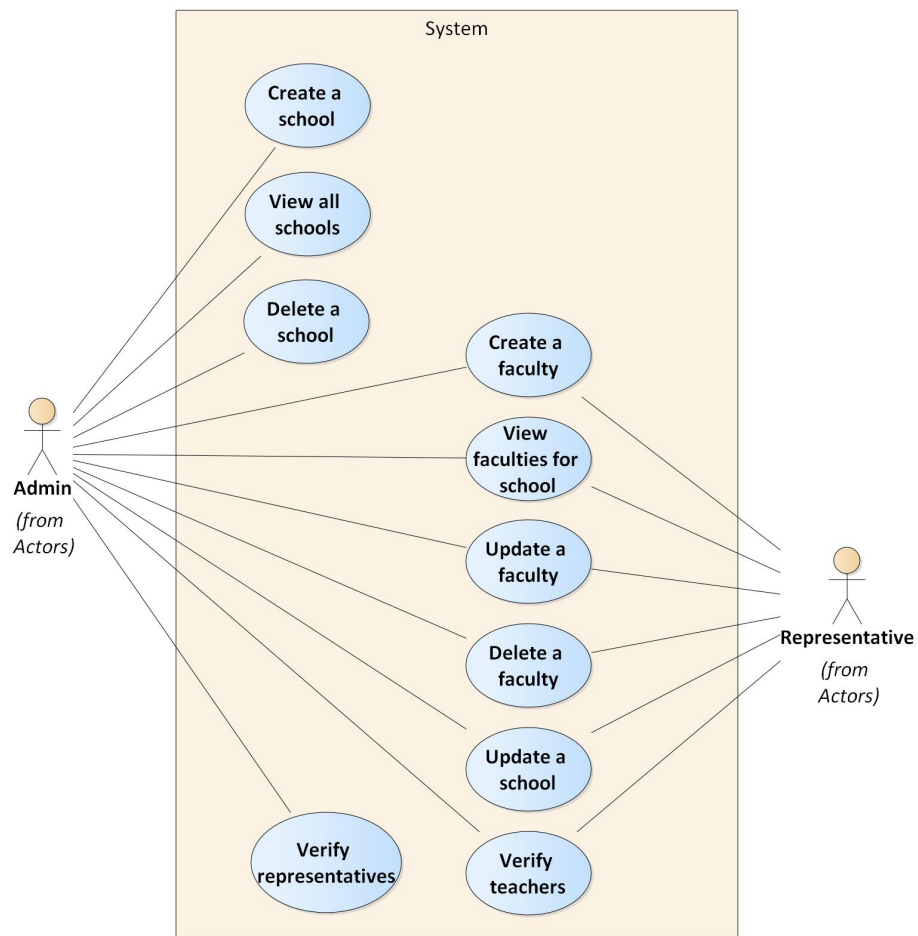


Figure 2.3: System Management-Related Use Cases

■ **UC003 Update a school**

Admins or verified representatives working at a given school update that school so that they do not have to create a new school entry in case of a mistake.

■ **UC004 Delete a school**

Admins delete a school so that there are no outdated entries in the system.

■ **UC005 Create a faculty**

Admins or verified representatives working at a given school create a faculty for that school so that teachers working there can register.

■ **UC006 View faculties for a given school**

Admins or representatives working at a given school view faculties at that school so that they know which faculties have already been added.

■ **UC007 Update a faculty**

Admins or verified representatives working at a given school update a

faculty at that school so that they do not have to create a new faculty entry in case of a mistake.

- **UC008 Delete a faculty**

Admin or verified representatives working at a given school delete a faculty from that school so that there are no outdated entries in the system.

- **UC009 Verify representatives**

Admins verify newly registered representatives so that their identity is confirmed and they can start using the application.

- **UC010 Verify teachers**

Admins or verified representatives working at a given school newly registered teachers working at that school so that their identity is confirmed and they can start using the application.

- **Account-Related Use Cases**

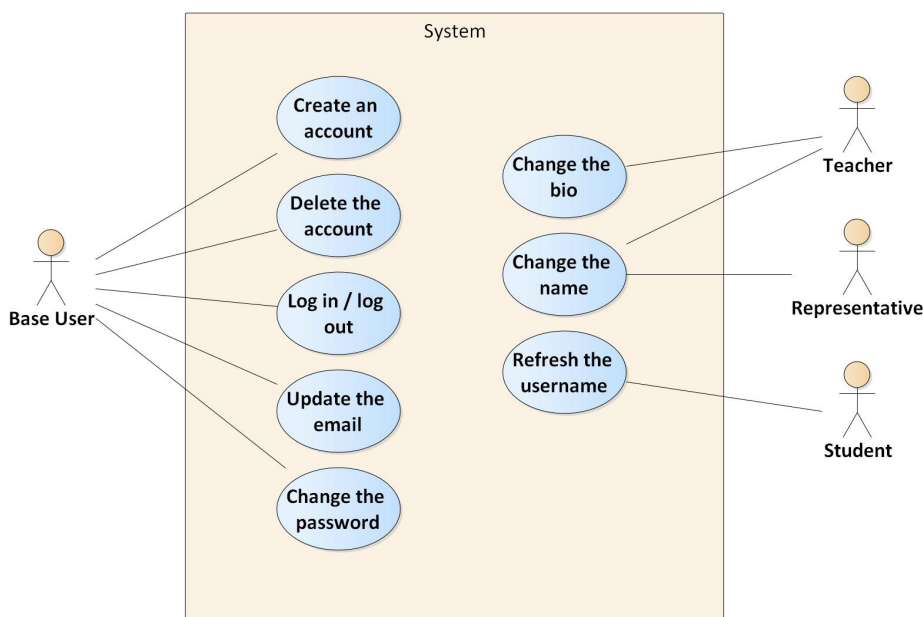


Figure 2.4: Account-Related Use Cases

Following is a list of account-related use cases (Fig. 2.4).

- **UC011 Create an account**

New users create an account so that their data and preferences are saved and they can use the application.

- **UC012 Delete the account**

Users delete an account so that their information is not stored in the system.

- **UC013 Log in / Log out**
Users log in using their credentials so that they can access their data.
Users log out so that unauthorized access to their data is prevented.
- **UC014 Update the email**
Users update the email associated with their account so that they can log in using a new email address.
- **UC015 Change the password**
Users change the password to their account so that their data is kept secure.
- **UC016 Change the name**
Teachers or representatives change their profile name so that the information is up-to-date.
- **UC017 Change the bio**
Teachers change the bio so that the information is up-to-date.
- **UC018 Refresh the username**
Students refresh their username so that their identity is hidden at all times.

■ Channels-Related Use Cases

Following is a list of channels-related use cases (Fig. 2.5).

- **UC019 Search for channels**
Users search for channels so that they can access the ones available in the system.
- **UC020 View channels**
Users view channels so that they can see what posts have been created in them.
- **UC021 Create a channel**
Verified teachers or students create a channel so that they can offer a place for others to create posts on a specified topic.
- **UC022 Update a channel**
Verified teachers or students update the channel that they have created so that the information is up-to-date.
- **UC023 Delete a channel**
Verified teachers or students delete the channel that they have created so that other users are no longer able to access it.
- **UC024 Join channels**
Verified teachers or students join channels so that they can see the posts from them in their feed.

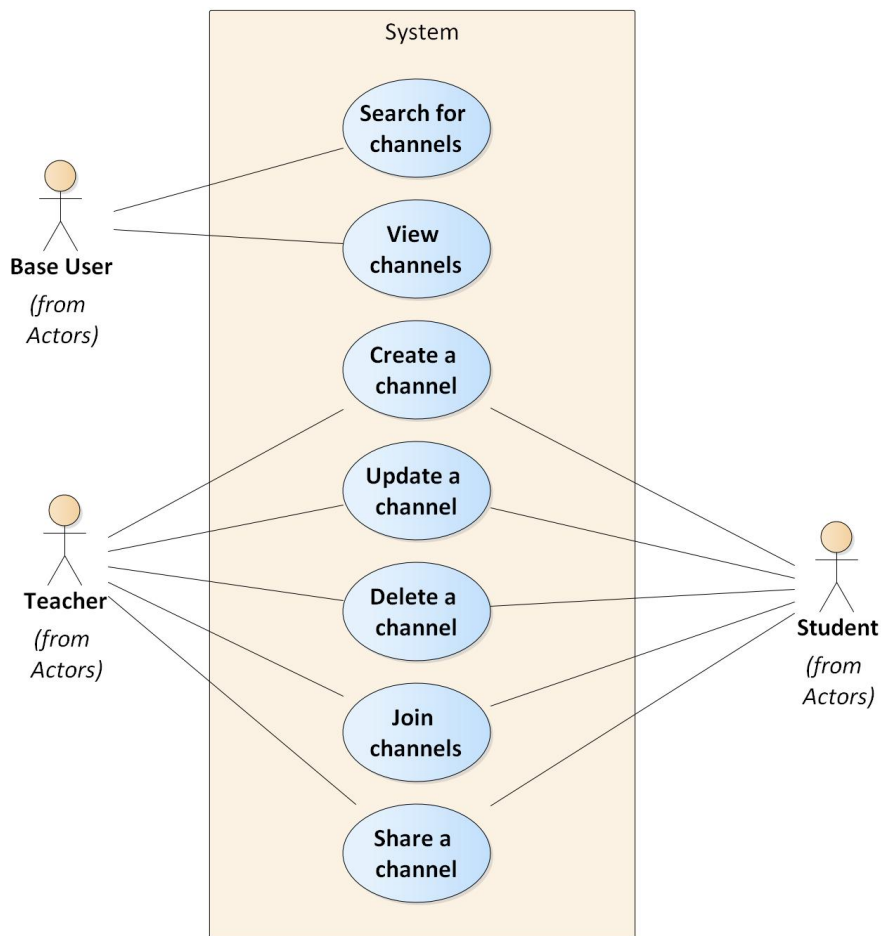


Figure 2.5: Channels-Related Use Cases

- **UC025 Share channel link** Students or teachers can share a link to a channel so that it is easy for others to quickly access that channel.

■ Posts-Related Use Cases

Following is a list of posts-related use cases (Fig. 2.6).

- **UC026 View channel posts**
Users view posts in a channel so that they can see what other users have said.
- **UC027 Create a post**
Students or verified teachers create a post so that they can share information with others.
- **UC028 Update a post body**
Students or verified teachers update the body of a post that they have created so that they do not have to create a new one in case of a mistake.

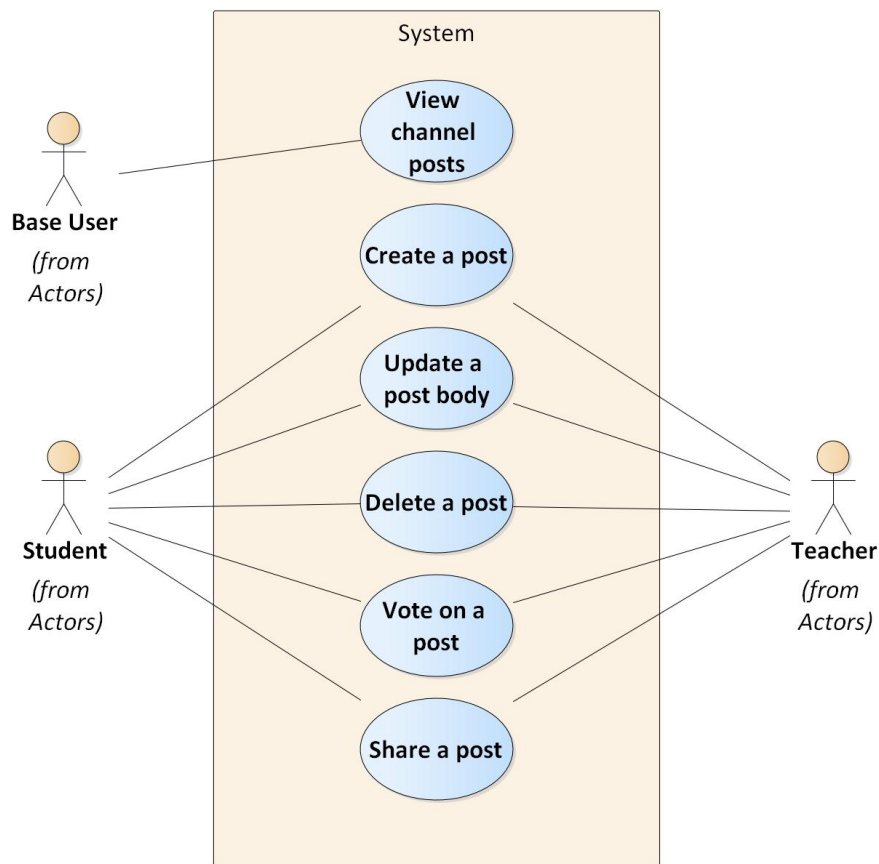


Figure 2.6: Posts-Related Use Cases

- **UC029 Delete a post**

Students or verified teachers delete a post that they have created so that other users can no longer see it.

- **UC030 Vote on a post**

Students or verified teachers vote on a post so that other users know how popular it is and are more likely to find it.

- **UC031 Share a post**

Students or teachers share a link to a post so that it is easy for other users to quickly access that post.

■ Comments-Related Use Cases

Following is a list of comments-related use cases (Fig. 2.7).

- **UC032 View post comments**

Users view comments created under a post so that they can see the opinion of other users on that topic.

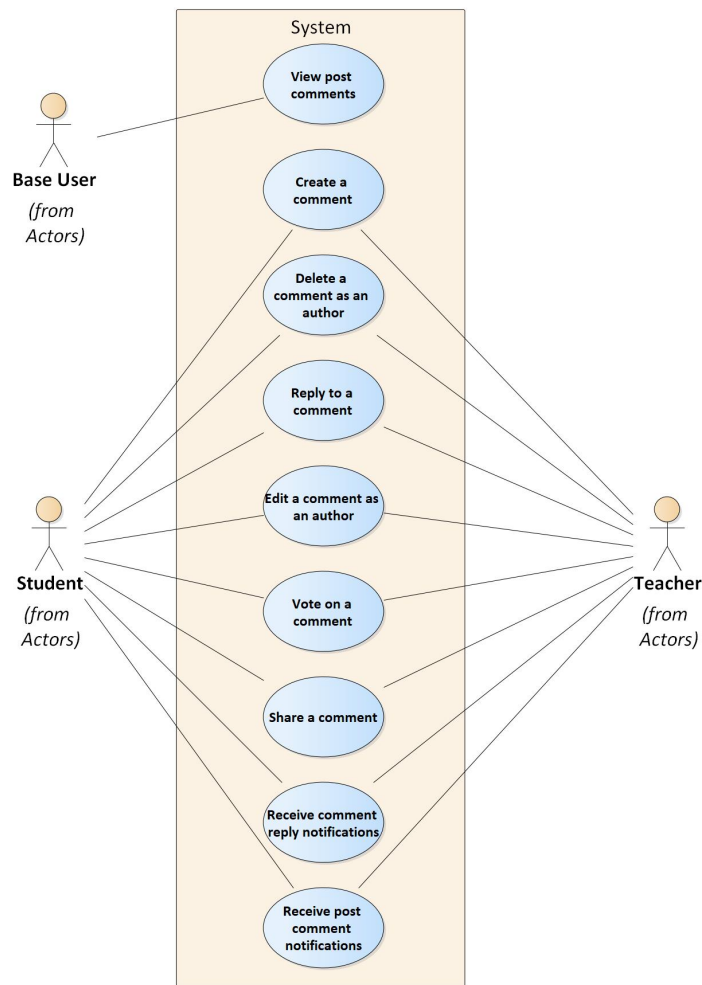


Figure 2.7: Comments-Related Use Cases

- **UC033 Create a comment**
Students or verified teachers create a comment under a post so that they can let the post author know what they think.
- **UC034 Delete a comment**
Students or verified teachers delete a comment that they have created so that it is no longer available for others to see.
- **UC035 Reply to a comment**
Students or verified teachers reply to comments left by other users so that they can engage in a conversation with them.
- **UC036 Edit a comment**
Students or verified teachers update a comment that they have created so that they do not have to create a new one in case of a mistake.
- **UC037 Vote on a comment**

Students or verified teachers vote on a comment so that other users know how popular it is and are more likely to see it.

■ **UC038 Share a comment**

Students or teachers share a link to a comment so that it is easy for other users to quickly access that comment.

■ **UC039 Receive comment reply notifications**

Students or verified teachers are notified when someone replies to their comment so that they can react to that comment.

■ **UC040 Receive post comment notifications**

Students or verified teachers are notified when someone leaves a comment under a post created by them so that they can reply.

Chapter 3

System Design

3.1 Application Architecture

Following the principle called "separation of concerns" is crucial for the app to remain scalable and maintainable over time. Thus implementing some sort of architectural pattern is necessary. I have decided to build the app using the Model-View-Controller pattern (Fig. 3.1). This pattern has proven itself to be a reliable way of organizing the app logic. The main idea is that the app logic is split into three components, according to the name. The model holds raw application data and defines the core modules of the app. The view is responsible for rendering the application state. It visually reflects the data that the model holds. By itself the view does not operate on data, nor does it know anything about the underlying logic of the app. The view communicates with the model via a controller. It is a middleman responsible for defining the communication standard between the two and protecting the underlying model from unwanted interaction.

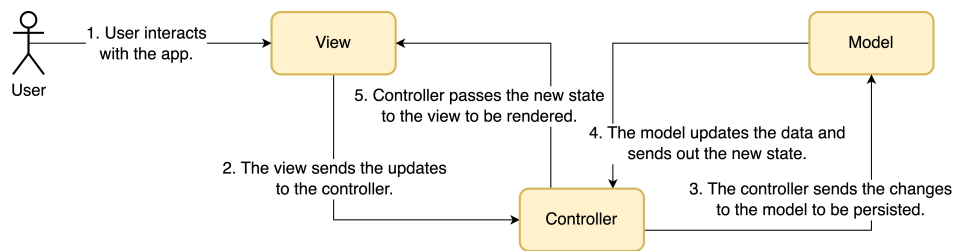


Figure 3.1: Data Flow in an MVC-Based Application

Following these principles, the app is separated into three core components: the front end (View), the back end (Controller), and the persistence layer, or database (Model). The component diagram (Fig. 3.2) shows how the components are organized and how they interact with each other.

- The database is responsible for persisting the application data and state.
- The back end is responsible for handling client requests, processing them, and interacting with the database. Communication with the persistence

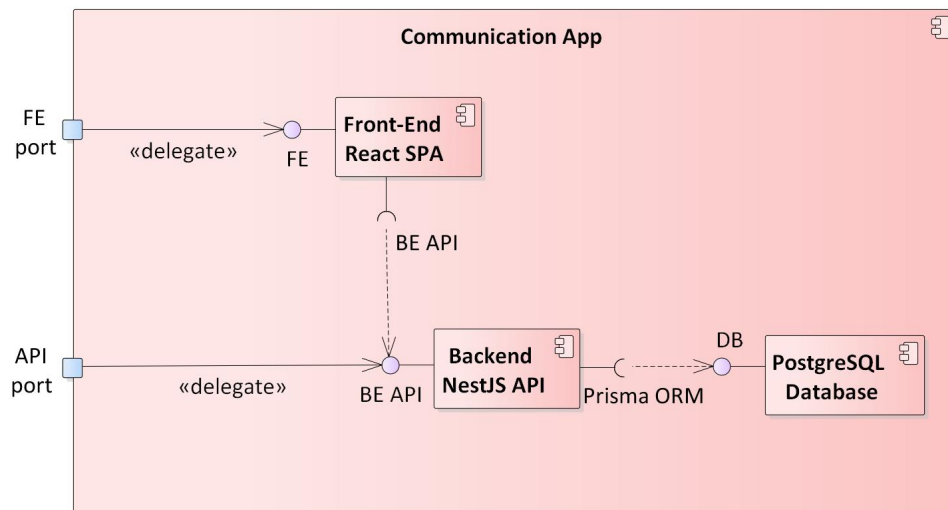


Figure 3.2: Components Comprising the Application

layer is done primarily, but not exclusively, with the help of an ORM (Object-Relational Mapper). The back end exposes a REST API for communication.

- The front end application acts as a presentation layer and allows users to interact with the app. Although the API is accessible without the front end, it contains some additional logic that helps users work with the app more efficiently.

3.2 Database Design

The conceptual model mentioned in the previous chapter has laid the foundation for the domain model of a relational database. The diagram (Fig. A.1) shows the structure of such a database.

It is necessary to make some key remarks regarding the database structure.

- There are four roles (types of user accounts): admin, representative, teacher, and student. Core data for each user is stored in the User table. The admin and student roles are regarded as a “base user” and so the data is only stored in the User table. The teacher and representative roles are special in the sense that they have some additional fields. The data for those fields are stored in two respective tables called Teacher and Representative.
- The Authority table contains all permissions. Each permission has a list of roles assigned to it. Permissions are required to perform some operation or access a specific API resource.
- The RefreshToken table stores all refresh tokens that have been issued so far. You can read how the tokens work further in this chapter.

Implementation details will be provided in the next chapter.

- The UserPostVotes table contains votes that users leave on specific posts.
- The UserCommentVotes table contains votes that users leave on specific comments.

The remaining tables follow the structure of the conceptual model, including the relationships between the entities.

■ 3.3 Back End Design

■ 3.3.1 Back End Structure

The back end follows the same principle of "separation of concerns" and therefore is implemented using the layered architecture (Fig. 3.3).

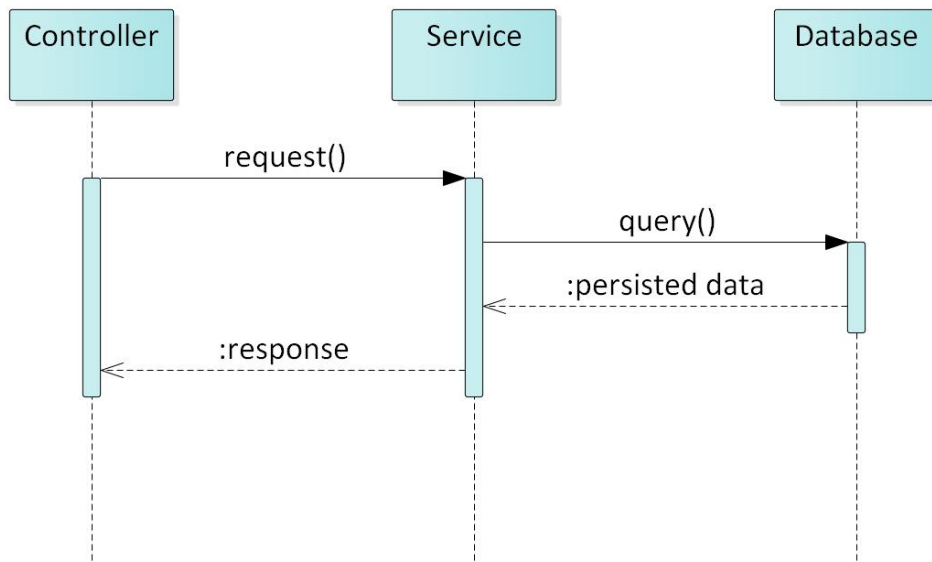


Figure 3.3: Back End Layered Architecture

When a request is received by a controller, it validates the received data. If the data meets the required criteria, it is passed along to the respective service. The service performs some operations on that data and interacts with other services and the database. Then it returns the result to the controller. The controller then sends a response to the client.

■ 3.3.2 REST API

A REST API (also known as a RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.

REST stands for representational state transfer and was created by computer scientist Roy Fielding [12].

For an API to be considered RESTful it has to conform to several criteria. The main ones are

- The app must implement a client-server architecture over the HTTP protocol.
- Communication must be stateless. This means that GET requests are independent of one another and are not connected. Client information is not stored between the requests.
- A uniform communication interface must be established between the client and the server.

The back end of my application exposes a RESTful API with 39 endpoints. Some endpoints may accept multiple types of requests based on the HTTP request method specified in the header. Depending on the endpoint, request data may be transferred in a request body in JSON format, path parameters, or query parameters.

■ 3.3.3 Authentication

To begin with, let me define what a cookie is. A cookie is a key-value pair that contains some data stored on the client side. The server sets a name for a cookie (the key) and some value stored as a string. The server then attaches the cookie to an HTTP response using the set-cookie header. Cookies issued by a server are always sent back to that specific server in all outgoing HTTP requests. Some cookies can also be sent to other servers as part of all outgoing HTTP requests. This behavior, however, can be altered by setting some options on the cookie.

There are multiple ways to protect an API from being accessed by unauthenticated users. One way is to use sessions. Sessions are cookies that contain a session ID assigned to the client. Session objects are stored in the database. The user only gets a cookie with the ID and so user credentials never leave the server. However, this approach has two main downsides.

- The cookie is never secure on the client side. It can get stolen through various types of attacks and misused by a third party.
- The server has to look the user up on every request. This reduces performance and delays the response.

Another option is to issue access tokens that would store hashed user credentials. This approach eliminates the need to look the user up on every request. However, contrary to session cookies, access tokens cannot be invalidated by the server. Once the token is issued, it will remain valid until its expiration time which means that anyone who can obtain the token, can use it to impersonate the original owner. Storing tokens in memory (as a

variable) should be sufficient to prevent unwanted access. Making access tokens expire swiftly after being issued (after only five minutes) would lower the likelihood of their misuse.

To prevent users from having to log back in every time they get an unauthorized exception or open the app after a period of inactivity, we must introduce refresh tokens. These tokens have a single purpose of validating user credentials after an access token expires. Refresh tokens are only used for issuing a new access token. It is important to stress that these tokens cannot be used to access any endpoints. Rather than expiring quickly, refresh tokens are long-lived (one week). They are stored in a secure, HTTP-only, same-site authentication cookie and can be revoked at any time by the server.

The last issue with the approach introduced above is that storing refresh tokens in a cookie poses the same risk as storing a session cookie. It is possible that the cookie gets stolen and then exploited to generate an unlimited amount of access tokens. This would lead to a stolen client identity and since access tokens cannot be revoked, nor invalidated, this becomes a tremendous security risk.

To overcome it, it is essential to prevent refresh tokens from being reused. To achieve this, refresh tokens are assigned a family ID and stored in a database table. Once a refresh token is used, it would be marked as used and a new refresh token with the same family ID would be issued. If a refresh token is attempted to be reused, then this is immediately detected and the whole family of tokens gets invalidated. This means that if a refresh token is stolen, it can only be used once to generate a short-lived access token. Then upon the real owner trying to use their refresh token, the theft is noticed and access is denied for everyone. The user would then have to log in again.

This system appears to be secure enough for the following reasons.

- The access token has a very short lifespan of only five minutes and so the likelihood of it being used for malicious intents is very low.
- The access token is stored in memory and cannot be stolen by a malicious script.
- The cookie containing the refresh token is secure which means it can only be used over HTTPS secure connections.
- The cookie containing the refresh token is HTTP-only which prevents it from being accessed by client-side APIs. This eliminates the threat of cookie theft via cross-site scripting (XSS).
- The cookie containing the refresh token has the same-site attribute set to strict which means it can only be sent to a target domain that is the same as the origin domain.

Both types of tokens will follow the JSON Web Token standard.

■ JSON Web Tokens (JWTs)

JSON Web Tokens are an open, industry-standard RFC 7519 method for representing claims securely between two parties [13]. JWT consists of three distinct parts separated by a period:

- Header: stores base64 encoded information about the algorithm used to encode and decode the token.
- Payload: information that is stored in the token
 - sub (subject): ID of the user being authenticated
 - iat (issued at): when the token was issued
 - exp (expiration): when the token expires
 - other fields unrelated to the token structure
- Signature: used to verify that the token has not been tampered with before being sent to the server.

Assuming the hashing function is called `bcrypt` and the secret is a 256-bit secret, the signature is calculated using the following formula.

$$\text{signature} := \text{bcrypt}(\text{base64}(\text{header}) + "." + \text{base64}(\text{payload}), \text{secret})$$

When the server receives a token (Fig. 3.4), its base64-encoded header and payload are extracted. Then the two parts are combined using a period and hashed together with a 256-bit secret using the algorithm specified in the header. If the resulting signature matches the one in the received token and the token has not expired yet, then the JWT is valid. If it does not or the token is past the expiration date, it is invalid.

■ JWT Access Tokens

Access tokens store user credentials. When the request is made to the server, the data is extracted from the token to verify that the user has the right to access the resource (Fig. 3.5). To obtain an access token, the user must first authenticate using an email and a password. The server then verifies the credentials, and if successful, a JWT access token is issued. The token contains:

- user's UUID
- user's role
- user's email
- token expiration time
- some extra data that is not relevant in this chapter

The access token is sent to the user in the HTTP response body.

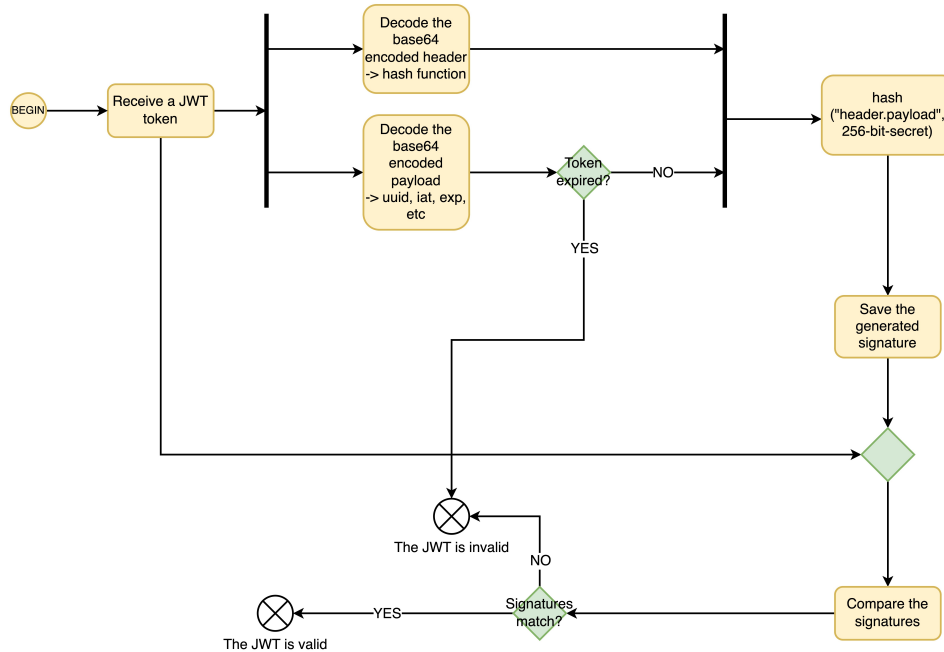


Figure 3.4: Handling a JWT Token

JWT Refresh Tokens

Refresh tokens store the same set of fields as access tokens and, in addition, a token family UUID. Refresh tokens are issued together with access tokens but they are stored in an authentication cookie to persist them on the client's side after the web page is closed. When the user logs out, the whole refresh token family gets invalidated and the user receives a new authentication cookie with zero lifetime to trigger the deletion of the cookie.

Authentication Flow

The authentication flow is depicted in the following sequence diagram (Fig. 3.6).

The user logs in using an email and a password. The server then looks the user up in the database. After that password hash is verified, an access token is generated, and a cookie with the newly issued refresh token is attached to the HTTP response. The user then receives all the necessary data, the access token, and the authentication cookie. If at any point the server encounters an error, the server responds with an unauthorized exception.

Refreshing the Access Token

Once the front end encounters an unauthorized exception, a request is made to refresh the access token (Fig. 3.7). The server uses the refresh token in the authentication cookie to issue a new pair of tokens. The old refresh token

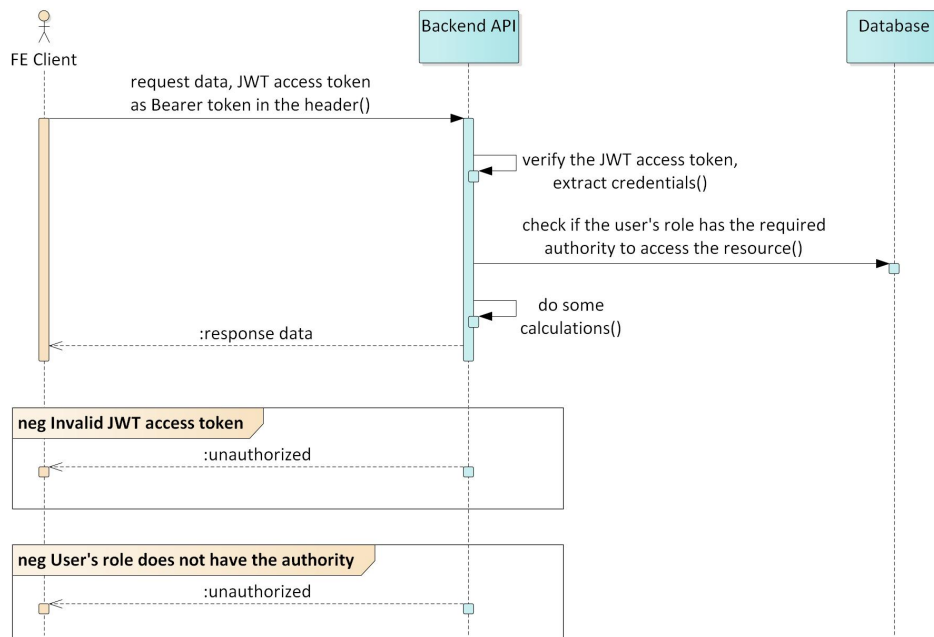


Figure 3.5: Generic Authenticated Request Sequence

is marked as used in the database. The following diagram shows the whole procedure.

■ Logging Out

When the user sends a logout request, the server erases the family of refresh tokens from the database and sends a zero lifetime authentication cookie to trigger its deletion on the client's side (Fig. 3.8).

■ 3.3.4 Authorization

■ Role-Based Authorization

To prevent unauthorized access a system based on roles and authorities is introduced. Following the analysis in chapter two, there are four types of users: admins, representatives, teachers, and students (Fig. 2.2). Each user type is mapped to a role in the system. Roles will have a list of authorities assigned to them. Authorities are permissions that are required to access specific resources. For instance, an admin user can only read posts in a channel, but a teacher can also create posts. Authorities are stored in the respective database table.

■ User Verification

According to the analysis, teachers, and representatives should maintain their identity and not be able to act anonymously. Therefore, verifying users with

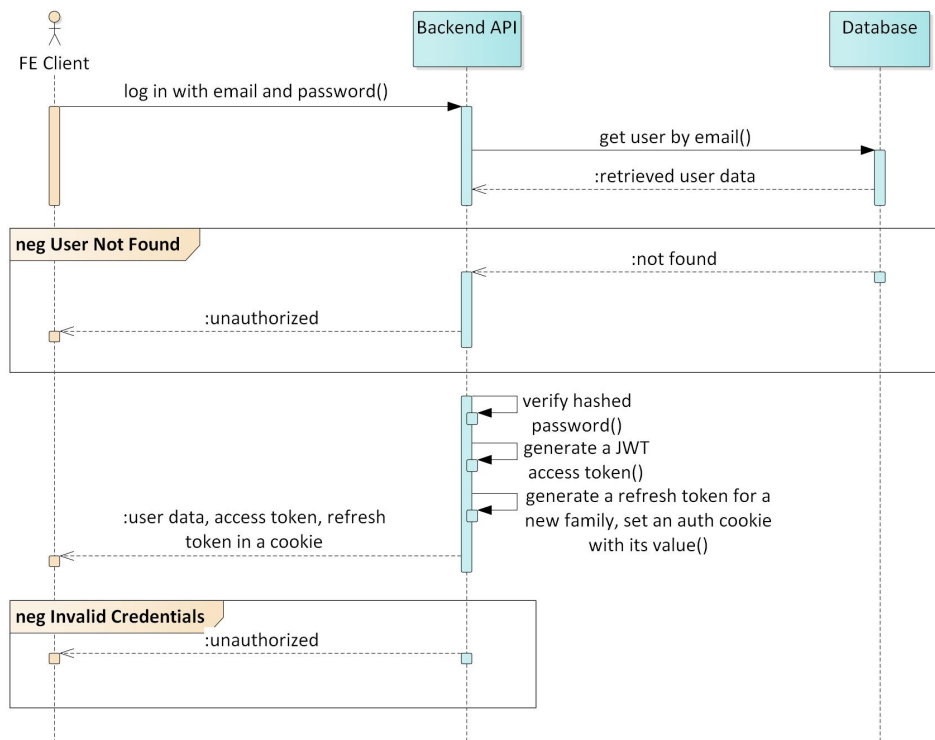


Figure 3.6: Authentication Flow Sequence

these roles is required.

Upon the first login, teachers and representatives are prompted with a notification asking them to wait for their account to be verified. In the meantime, they are not able to perform any meaningful actions apart from browsing the content or editing their profile. This restriction applies both to the front end and the back end parts of the application.

The users can be verified, or their verification may be declined for some reason. If the user is verified, their account is given full functionality within the scope of available permissions for their role. If the user's verification is declined, they are prompted with the reason why that is the case. Once the user is ready to request to be verified again, they can do so. The state diagram (Fig. 3.9) shows the structure of the verification process.

3.3.5 Voting System

Users can vote on posts and comments to raise a post or comment in popularity. They can either leave a positive vote (an upvote) or a negative vote (a downvote). A positive vote increases the resulting vote, while a negative one decreases it. Users can leave a maximum of one vote per post or comment. The resulting vote is simply a difference between positive and negative votes. The state diagram (Fig. 3.10) shows how the voting system works. The *dir* parameter (stands for direction) indicates whether the user has upvoted or downvoted the post, or has removed their vote completely.

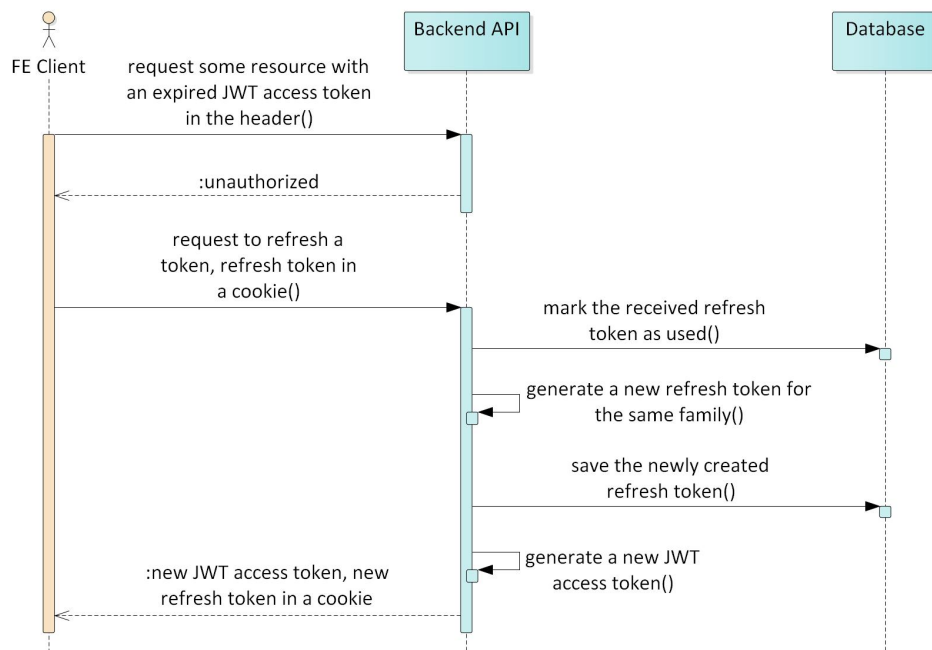


Figure 3.7: Refreshing the Access Token Sequence

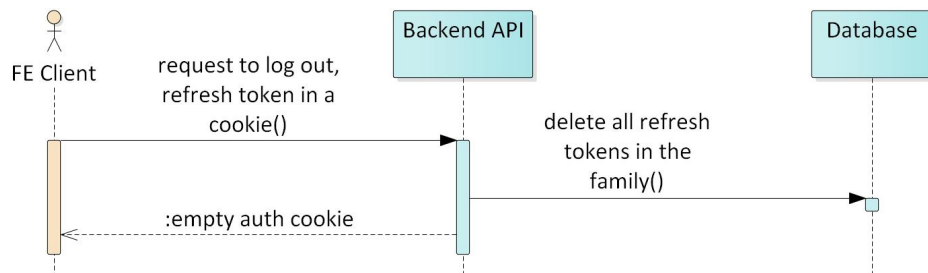


Figure 3.8: Logging Out Sequence

3.3.6 Anonymity

Anonymity lies in the foundation of the app. To implement the central requirement of protecting students' identity, they can request a new username at any time. Once a student renews the username in their user profile, all posts and comments created earlier maintain the old username. This is accomplished by storing a copy of the username on the post or comment entities.

3.4 Front End Design

The primary goal of the frontend application is to enable users to interact with the API in an efficient and user-friendly manner. In addition, the front end should be able to prevent users from performing unwanted actions. Such

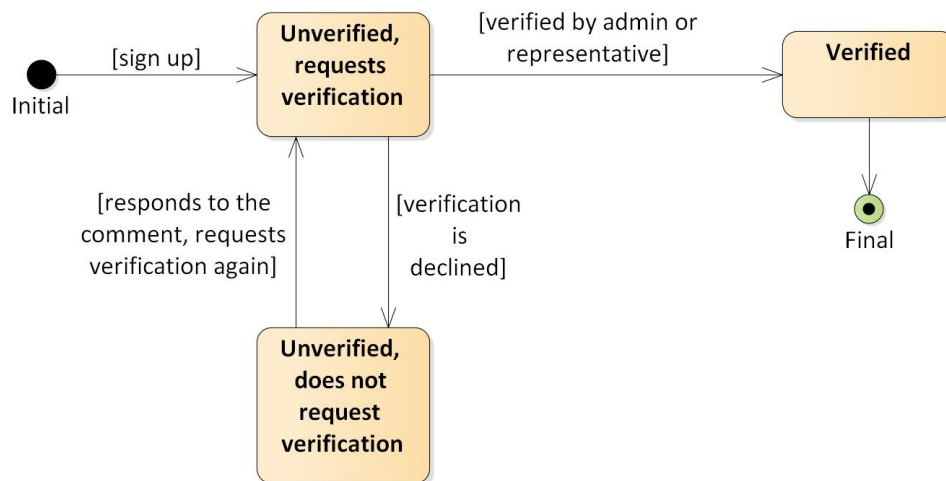


Figure 3.9: Verification State

actions may include accessing resources that are not meant to be accessed by their specific user role (e.g. editing schools with a student role) or performing actions that are not available to them at the given time (e.g. creating posts before being verified).

■ 3.4.1 Wireframes

Wireframes provide an overview of the UI structure, layout, and intended user behaviors. As the wireframes are usually created to display future functionality, they lack in design. Depending on the application, wireframes may not reflect the whole UI structure, or all user flows [14][15]. Since this project is a solo undertaking, I have decided to omit modal windows, some navigation elements, and some conditionally displayed elements when creating the wireframes. The ones I have created show the interface of the key pages.

■ Authentication Screen

Authentication screen (Fig. D.1) enables users to create an account or to log in using their credentials. Upon opening this screen, users are prompted with a login form. This ensures that existing users can instantly authenticate without any additional steps. New users can create an account by toggling the form to a signup mode. The form changes accordingly based on the desired user role.

■ Managing Schools and Faculties

According to the use case diagram (Fig. 2.3), admin users should be able to manage the list of available schools (Fig. D.2). The table shows the list of schools available in the system. Users can edit schools, delete schools, or navigate to the list of faculties for a given school.

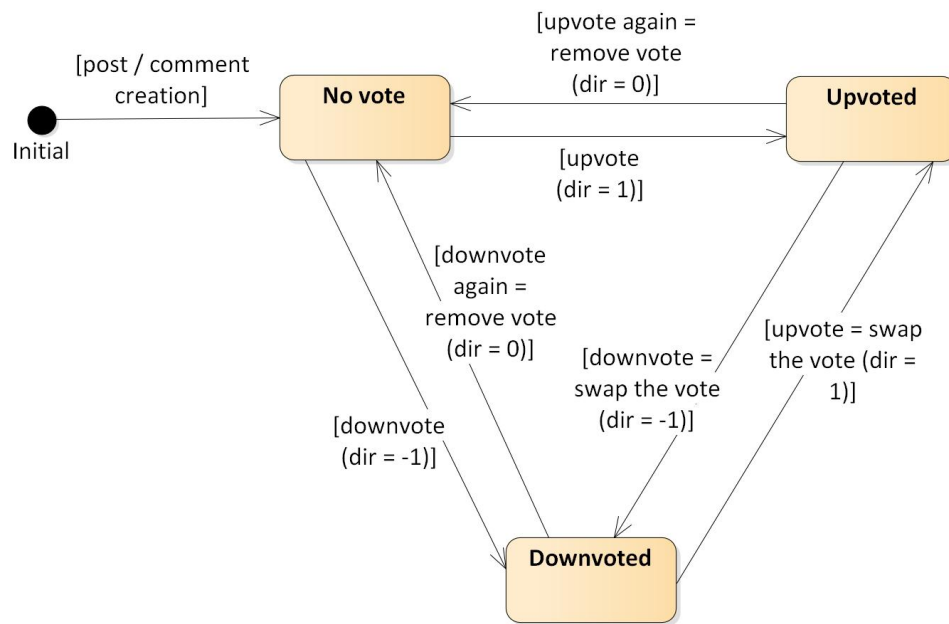


Figure 3.10: Vote State

Managing faculties is performed on the faculties page (Fig. D.2). This page should be accessible by admin users, and representatives. Here users can edit faculties, or delete certain faculties. Representative users should only have access to the list of faculties of the school that they are associated with.

■ User Verification

According to the use case diagram (Fig. 2.3), admin users should be able to verify representatives and teachers. Representative users, however, should only be able to verify teachers associated with the same school at which they work. This is done on the requests page (Fig. D.3). The tab with requests from representatives is only to be accessible by admins.

■ Channels

Channel page (Fig. D.4) should enable users to see posts created in a given channel. Users should be able to sort posts based on the time they were created, or on their popularity. Users should be able to join a channel so that the posts from the channel appear in their feed (Fig. D.4). The owner of the channel should be able to edit the channel ID, its name, or description, or delete the channel with all its content entirely. In addition, the UI should also clearly show the number of members, the creation date, and the owner of the channel. Users should be able to create posts in channels regardless of their membership.

■ The Feed

The feed (Fig. D.4) shows user posts from the channels that they have decided to join. The posts in this view should have a link to the channel they originate from.

■ Posts

Posts in the feed (Fig. D.4) and on the channel page (Fig. D.4) only display a limited amount of content. Post page (Fig. D.5) allows users to access the whole post. Here users should be able to access comments under a given post or leave their comments.

■ Comments

For the reason the horizontal space on the screen is limited, the post page (Fig. D.5) only displays six nested levels of comments. To access comments nested deeper in the comments tree users should be able to navigate to the comments page (Fig. D.5). Each comment, regardless of the page, enables users to read it, or reply to one. Comments that are created by a given user can also be edited or deleted.

■ User Profile

User profile page (Fig. D.3) has four sections is available only for certain users. Refreshing the username is only available to students. On the other hand, editing the profile is only available to teachers and representatives. Though representatives are only able to change their names. The remaining two sections are available to all users.

Chapter 4

Implementation

In this chapter I will elaborate on implementation details of the application. I will also cover the technological stack used, and briefly provide core concepts for each technology. However, first I would like to mention the language in which the application is written and the reasoning behind my choice.

4.1 TypeScript

The whole application is written in TypeScript.

TypeScript is a free and open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language [16]. Static type checking is the process of analyzing the source code of the program. If a program passes the type checking, it is guaranteed to satisfy some set of type safety properties for all possible inputs [17].

TypeScript has a built-in compiler that is itself written in TypeScript. Code written in TypeScript is transpiled into the specified version of JavaScript. Before the code is transpiled into JavaScript, the compiler validates the types used. This is the main benefit of using TypeScript since it helps eliminate errors that would otherwise appear at runtime. Another substantial advantage is that it speeds up the development process by giving autocomplete suggestions based on the type of a variable. Although being a standard feature in all strongly-typed languages, when programming in the JavaScript ecosystem, this is only available for TypeScript-based development.

Now that I have provided a brief overview of TypeScript and how it works, I would like to give the reasoning behind my choice:

- TypeScript is the language I am very familiar with since for over a year I have been using it for both personal and work-related development projects.
- The front end of the web application can only be written in JavaScript since that is the only language that browsers use to run client-side code. To avoid switching contexts and have as pleasant of a developer experience as possible, I have decided to write the whole application in TypeScript.

■ 4.2.4 Prisma ORM

Choosing an ORM for this project has been a rather trivial task. I have selected Prisma as an ORM for several reasons which I would discuss further in this section. However, first, let me explain how Prisma works.

Prisma ORM consists of three parts:

- Prisma Client - a type-safe database query builder
- Prisma Migrate - a set of database migration tools
- Prisma Studio - a database GUI inspector, which I have not used

In the beginning, a Prisma schema file is created using an intuitive, yet proprietary data modeling language. This file defines the structure of a database (the entity models and the relations between them) as well as a database connection string (URL). Then you run a command to create a migration file. The Prisma migration tool detects changes in the schema and automatically generates the necessary migrations. The migrations are then applied to the database. The schema file also specifies a generator that generates the Prisma client. The Prisma client is a query builder - a set of typed objects that are used to run database queries. The reasons for my choice of Prisma have been the following:

- Prisma is an ORM for TypeScript-based projects, therefore it offers code autocompletion that simplifies the development process.
- The code that is generated to communicate with the database (Prisma client) is type-safe and is automatically generated from the underlying schema. It is tailored to the specific database structure and is updated when the schema changes.
- Prisma migrate automatically detects changes in the schema and generates appropriate migrations.
- Prisma ORM also offers an option to run raw SQL queries if needed.
- Prisma does not impose its principles on developers. Rather it allows developers to intervene: it offers an option to alter the migrations before applying them, or it provides an ability to run raw SQL queries if needed.

■ 4.2.5 Custom Queries

Although Prisma allows running almost all types of queries, it still lacks a few key components of the required functionality. The missing feature that I encountered and had to substitute was the ability to run recursive queries. PostgreSQL allows running recursive queries with the help of CTEs (common table expressions). CTE is a temporary result set that can be referenced in another SQL statement [19]. To create a recursive query one would simply reference a given CTE from within this CTE. One caveat is that such a CTE has to be marked with the *recursive* keyword.

a communication layer. However, I will provide the details of this in the following sections.

■ 4.3.3 Using HTTPS in Development

When the app is deployed into a production environment, it will be served using the encrypted HTTPS protocol. To simulate this behavior when developing the application locally, it is necessary to generate a locally signed SSL certificate.

An SSL certificate is a digital certificate that authenticates a website's identity and enables an encrypted connection. SSL stands for Secure Sockets Layer, a security protocol that creates an encrypted link between a web server and a web browser [22]. Using SSL certificates keeps the information transferred between the server and the client private and encrypted.

First, the reader must assume the following three cryptographic concepts hold.

- Asymmetric encryption (used in HTTPS communication) relies on two keys: a public one, and a private one.
- A public key is used to encrypt messages, however, they can only be decrypted using a private key. A public key is shared with other parties. A private key, in line with the name, is always kept with the original owner.
- A public key can be used to verify that a message has been encrypted using the corresponding private key.

Second, I will summarize how communication over HTTPS works. Note that this is a simplified version of the process and so it omits various other requests that are made to establish a secure communication channel (e.g. requests to DNS servers).

- The browser (the client) sends a request to an SSL-protected web server stating that it wants to access some resource.
- The web server responds with its SSL certificate containing its public key signed by a certification authority (CA).
- The browser then checks if it knows such a CA and whether it is trusted. Since the browser has access to the CA's public key, it is easy to verify the validity of the certificate sent by the server.
- If the certificate is valid, the browser then creates a secret, encrypts it using the server's public key, and sends it back to the server.
- Since the server is the only party that can decrypt this secret, a secure, symmetrically-encrypted communication channel using the generated secret is now established.

- JWT strategy - this strategy verifies a user by decoding a JWT access token received in the request header.
- Basic strategy - this strategy verifies a user by their username and password stored in the basic authorization header of the request object.

The local strategy works by looking up a user using their email in the database. If successful, the password is compared. The user is then either granted or denied access to the resource. I have used the local strategy when authenticating login requests.

The basic strategy works by comparing the username and password transferred in the basic authorization header of the request:

Authorization : Basic < credentials > .

The credentials are a base64 encoded string in the form of *username : password*. This authentication strategy is used to create admin users and is only available through the API. It is not implemented in the front-end part of the app.

The JWT strategy works by decoding a JWT authentication token stored in the bearer authorization header of the request object:

Authorization : Bearer < token > .

The process of decoding a JWT token has been described in the previous chapter. I have adopted this strategy to authenticate the remaining requests, including requests to refresh an access token. However, in that case, the JWT token is a refresh token stored in an authentication cookie.

■ Nest Guards

Nest uses guards in the form of an annotation to determine whether an incoming request should be handled depending on certain conditions available at runtime. Guards are used mainly for authentication and authorization purposes. Contrary to middleware, guards are aware of what is going to be executed next in the request-handling pipeline. Therefore they can stop an incoming request at exactly the right moment. In essence, guards are simply functions that return a boolean value indicating whether the request should be handled by the controller. To protect a resource one would simply annotate it with the required guard. For each Passport strategy used I have implemented a corresponding guard:

- Local authentication guard - for the local authentication strategy
- JWT authentication guard - for the JWT authentication strategy
- JWT refresh authentication guard - for the JWT authentication strategy when refreshing an access token
- Basic authentication guard - for the basic authentication strategy

■ 4.3.6 Modular Structure

The back-end part of the application consists of separate modules. The modules are responsible for handling a certain part of the functionality. I have divided the app into the following modules:

- Authentication - logging in, logging out, creating users, refreshing access tokens, modifying user profile data Authorities - getting authorities
- Channels - channels CRUD, toggling channel membership, looking up channel ID availability
- Comments - comments CRUD, voting on comments
- Cookies - generating authentication cookies
- Faculties - faculties CRUD
- Feed - getting posts for the feed view
- Notifications - getting user notifications and dismissing them
- Posts - posts CRUD and voting on posts
- Prisma - interacting with the database through an ORM
- Refresh Tokens - handling refresh tokens
- Representatives - getting representatives' requests
- Schools - schools CRUD
- Teachers - getting teachers' requests
- Users - verifying users and getting user profiles

In general, modules are based on the corresponding entity in the database. Each module consists of a controller, a service, and a set of request parameter objects. A controller is responsible for handling client HTTP requests. A service is in charge of the core application logic: working with DTOs (data transfer objects) and communicating with the database.

■ Request Parameters

The data contained in HTTP requests can be stored in the request body (in JSON format), in path parameters, or query parameters. Path parameters are a part of the endpoint route and are always required. Query parameters are optional and are found in the query string that starts with a question mark, at the end of a URL. Usage of all types of parameters can be found in the application.

To be more precise, I have used Redux Toolkit which is an opinionated toolset for Redux development [27]. Redux toolkit allows you to create so-called “slices” of data and define dispatcher functions that would be used to modify the store. Usually a slice is defined for each separate domain of data used in the application. I have followed this approach and divided the store into the following slices:

- Authentication slice
- Comments slice
- Feed slice
- Posts slice

The authentication slice holds user credentials and some additional user specific data. The latter three slices only serve a complementary function. The reason for this is that I have also decided to use another tool called RTK Query (Redux Toolkit Query). RTK Query creates a separate API slice for all the data that it manages.

RTK Query is an optional addon in the Redux Toolkit package and it serves as a data fetching and caching tool [28]. RTK Query makes use of the OpenAPI specification, that is generated on the back end, by defining endpoints and exporting functions that encapsulate data loading. In addition, it creates a caching layer that holds the data obtained from HTTP responses. Whenever the application sends a request, RTK Query first checks if there is already some data in the store. If the data exists and is not stale, no request is made and the response is created from the data found in the cache. Otherwise, a request is made and the response is cached for some time. In order to avoid receiving outdated cached data, RTK Query introduces a tag based system. Endpoints that send GET requests (queries) provide a list of tags. On the other hand, endpoints that send other types of HTTP requests (mutations) invalidate a list of tags. Whenever a mutation is called, data with the corresponding provided tag is immediately invalidated and later refetched.

■ 4.4.4 Navigation

I have chosen to implement client side navigation using the React Router library [29]. The idea behind this concept is that initially the whole application bundle is sent to the user. The router library then observes the history stack and renders required content based on the URL. React Router is also capable of rendering nested layouts, where parts of the page switch dynamically based on path parameters present in the URL. For instance, such layouts form the channels, posts, or comments pages.

■ 4.4.5 Handling Forms

Single-page applications introduce a problem with handling forms. Managing the form state, keeping track of values and errors, validating fields, and

submitting the form becomes a non-trivial task. I have used a library called Formik [30] that solves all the aforementioned issues. Since the form state is rather ephemeral and the form data is usually accessed from a single component only, Formik stores the state of the form using built-in storage. The whole form is wrapped in a single Formik component that provides field values, errors, and a submission callback. On top of that, Formik also allows fields to be validated whenever the value changes or the field loses focus. Validation is done using a library called Yup. Yup offers a way to define a schema and then validate objects against it. Finally, Formik prevents form submission before all form fields are considered valid.

■ 4.4.6 UI Components

The main reason I have decided to create my own UI components is that most component libraries force you to adopt predefined styles. However, I have opted to have an application with a unique look. Moreover, this decision has been influenced by the effort to satisfy one of the non-functional requirements which states that the application UI has to dynamically adapt to all screen sizes. It is known that writing pure CSS can be rather cumbersome. Therefore, for styling purposes, I have chosen Tailwind CSS. Tailwind is a CSS framework that offers hundreds of utility CSS classes. These classes are applied inline, directly on every HTML element. The advantage of using a styling framework rather than creating the styles by hand is that such frameworks offer a system of constraints that make it easy to maintain consistency with colors, spacing, etc. I have created most components used in the app completely from scratch:

- two types of modal windows
- four types of alerts
- badge
- five types of buttons
- icon button
- input textbox
- styled link
- loading spinner
- radio group
- selector
- table
- text area

However, several components require implementing rather complex logic associated with them. Headless UI is a library from the makers of Tailwind that offers unstyled components. The following components have been adopted from the library, styled, and integrated:

- popover
- dropdown
- autocomplete
- tabs

Popover component adds a button that upon clicking shows some content stacked on top of the page UI. Dropdown enables building menus that are revealed when clicking on a button. Autocomplete is a textbox augmented by a dynamically formed list of options. Tabs offer a way to switch between multiple views by clicking on the corresponding tab button.

■ 4.4.7 Switching Color Scheme

The dark mode feature enables users to switch the color scheme of the UI to a dark variant. I have implemented three modes (Fig. E.1): a light mode, a dark mode, and a system mode. System mode is enabled by default, and it matches the *prefers – color – scheme* media query. This media query updates whenever the system preference changes. The light mode allows users to permanently keep the light variant of the UI. The dark mode does the same, but for the dark variant of the UI. If the user selects either of the non-default modes (light or dark), their preference is saved in the browser’s local storage. If the user switches to the system mode, their preference is erased and the system-defined value is applied.

■ 4.5 Deployment

The application is deployed onto Digital Ocean App Platform [31]. The app platform is an all-in-one PaaS (platform as a service) solution for deploying applications without having to configure the environment. I have created an app and attached a database to it. Prisma ORM allows me to deploy migrations directly to the production database using a URL. The front-end part is deployed to a CDN within that app. The back end is deployed as a constantly running web service.

Another useful feature of the app platform is that it allows triggering automatic redeploys when a certain Git branch is updated. Therefore I have connected my GitHub and Digital Ocean accounts and have configured the master branch as the target branch for automatic deployment jobs. The production version of the application is available at

<https://comm-app-2gvj8.ondigitalocean.app>.

Chapter 5

Testing

5.1 Automatic Testing

As part of application development, I have written several suites of automatic tests. Specifically, I have opted to write end-to-end (integration) tests. Such tests check how different app components integrate. Moreover, integration tests are designed to follow various interaction scenarios and thus resemble actual users' behavior. I have set up an isolated testing environment with its own environment variables and a separate database. Before each test suite is run, the database is reset and the migrations are applied for the tests to yield consistent results.

Nest provides a set of preconfigured tools for all types of testing. It uses a framework called Jest to run tests and assert the correctness of tested data. Another library called Supertest helps emulate HTTP requests being sent to controllers. All tests that I have created follow the AAA principle: arrange-act-assert. First, I arrange something before the test runs. For instance, I could send a request to log in and obtain the access token. Second, I act, meaning the actual test is run. Finally, the results of the test are asserted to be correct against expected values.

An example of an end-to-end test (Appx. B) imitates a user creating a post and the post data is later asserted.

5.2 User Testing

After deploying the application, I set up two testing scenarios: one for teachers, and one for students (Appx. C). Each testing scenario consists of three parts. In the first part, users are asked to create an account and log in using their credentials. In the second part, users are asked to perform some actions by interacting with the application. Some of the actions are: creating a channel, joining a channel, creating a post, leaving a comment, etc. The last part of the scenarios asks users the following three questions.

- Please specify if you have encountered any errors. Did you see any error alerts? If so, where?

- Did you find any task complicated, confusing, or unintuitive to complete?
- Do you have any suggestions on how the user experience could be improved?

I have asked three students and three teachers to test the production version of the application. Following is their feedback.

■ 5.2.1 Student A

Student A has said that they liked the overall feel of the application. They have mentioned that they had tried to hack the security and get access to parts of the application to which they should not have access but to no avail. Regarding the questions, they provided the following answers.

1. **Please specify if you have encountered any errors. Did you see any error alerts? If so, where?**
 - The search was not able to find the ID of my channel. I had to refresh the page.
2. **Did you find any task complicated, confusing, or unintuitive to complete?**
 - At first, I did not know how to change the color scheme. I was confused as the button for doing so had an icon with a computer monitor.
 - I thought that the channel ID should be a number and was perplexed why it could not be generated automatically. It turned out that the channel ID is similar to usernames, but for channels.
 - The button for creating a channel should not have been placed in the profile menu. It has nothing to do with the user profile.
 - It would be great to see a popup saying "The link has been copied" when the share button is clicked.
 - I was also not sure why teachers should have random usernames assigned to them if they are not anonymized.
3. **Do you have any suggestions on how the user experience could be improved?**
 - I would prefer to see a filter for channels. Also it is not possible to access a list of created channels.

■ 5.2.2 Student B

Student B has provided the following answers.

1. **Please specify if you have encountered any errors. Did you see any error alerts? If so, where?**

- I have not found any errors.
2. **Did you find any task complicated, confusing, or unintuitive to complete?**
 - When creating a channel I was confused by error messages in the channel ID field. Upon entering only one symbol, the error said, "Must only contain numbers and letters." It should have said, "Must be between 2 and 20 characters."
 3. **Do you have any suggestions on how the user experience could be improved?**
 - It would be helpful to access the list of created channels and also the list of channels that I have joined.

■ 5.2.3 Student C

Student C has complimented the design and has praised the mobile-friendly UI. They have sent the following answers.

1. **Please specify if you have encountered any errors. Did you see any error alerts? If so, where?**
 - I have not encountered any errors.
2. **Did you find any task complicated, confusing, or unintuitive to complete?**
 - The channel ID field said the input must only contain numbers and letters. It turned out that no capital letters are allowed. I would suggest switching the error text to a more comprehensive one.
 - When creating a channel, the field "Name" should be instead labeled "Channel Name". I was confused about which name I was supposed to enter.
 - The button for creating a channel should be placed elsewhere. It does not relate to the user profile section.
3. **Do you have any suggestions on how the user experience could be improved?**
 - I would suggest adding an option to make the channels private so that the content is only available to some users.

■ 5.2.4 Teacher A

Teacher A has said that they use similar applications (Reddit, Discord, Slack) on a regular basis and so the UI seemed very intuitive to use. They have provided the following feedback.

1. **Please specify if you have encountered any errors. Did you see any error alerts? If so, where?**
 - I have had an error when creating a channel. Specifically I was not able to use letters in the channel ID, even though the input error said "Must only contain letters and numbers".
2. **Did you find any task complicated, confusing, or unintuitive to complete?**
 - The only thing I have found difficult is choosing the ID for my channel.
3. **Do you have any suggestions on how the user experience could be improved?**
 - I do not have any suggestions.

■ 5.2.5 Teacher B

Teacher B has sent the following answers.

1. **Please specify if you have encountered any errors. Did you see any error alerts? If so, where?**
 - I have had issues creating a channel. I could not overcome the channel ID input error that said "Must only contain numbers and letters".
 - The search bar did not work as expected. Sometimes no results were shown with identical input.
2. **Did you find any task complicated, confusing, or unintuitive to complete?**
 - Upon signing up I entered the school name and then did not choose any of the suggested results. I was confused as to why the input field became blank.
3. **Do you have any suggestions on how the user experience could be improved?**
 - I would suggest adding tooltips that are shown when hovering over buttons.

■ 5.2.6 Teacher C

Teacher C has provided the following feedback.

1. **Please specify if you have encountered any errors. Did you see any error alerts? If so, where?**

- I have not found any errors.

2. Did you find any task complicated, confusing, or unintuitive to complete?

- I was slightly confused why the names of schools were listed in Czech but the form was in English.
- I could not find the button for creating a channel. I believe that it should not have been placed in the profile menu.
- I did not understand the purpose of the "Join / Leave" button.

3. Do you have any suggestions on how the user experience could be improved?

- I do not have any suggestions.

5.2.7 Results of User Testing

After receiving the feedback I implemented a fix for the search bar. I could not reproduce the channel ID input error that most people had seen. It seemed as they had been trying to use capital letters and so the validation error was not clear. I changed the error to "Must only contain small letters and numbers".



Chapter 6

Conclusion

All goals stated in the beginning of the thesis have been realized.

First, I analyzed online communication between teachers and students and studied the existing solutions. I identified the advantages and disadvantages of those solutions and introduced a list of functional and non-functional requirements for the application.


Second, I designed the application with the specified requirements in mind. I chose a stack of technologies and implemented the application.

In the last part of the thesis, I conducted automatic end-to-end tests, and later asked a group of teachers and students to test the application. End-to-end tests allowed me to maintain the app functionality after code changes. User tests have provided me with feedback and insights on how to improve the application.

In addition to suggestions received from people who have tested the app, I would like to point out further improvements that I have come up with after finishing the development.

- Upgrade posts to a rich-text format for improved readability.
- Add an ability to upload images and attach other documents to posts.
- Introduce private one-on-one chats with teachers.
- Make posts searchable.

My work has resulted in the creation of a tested and deployed web application. Throughout this project, I have studied and practically applied several new technologies, for instance, RTK Query on the front end, and NestJS and Prisma on the back end. The invaluable feedback I have received from users has provided me with the direction for future development. Nevertheless, the application is already a fully-functioning product, and people from educational institutions around the globe can benefit from it today.



Appendix A

Database Structure

The domain model (Fig. A.1) is shown on the following page.

A. Database Structure

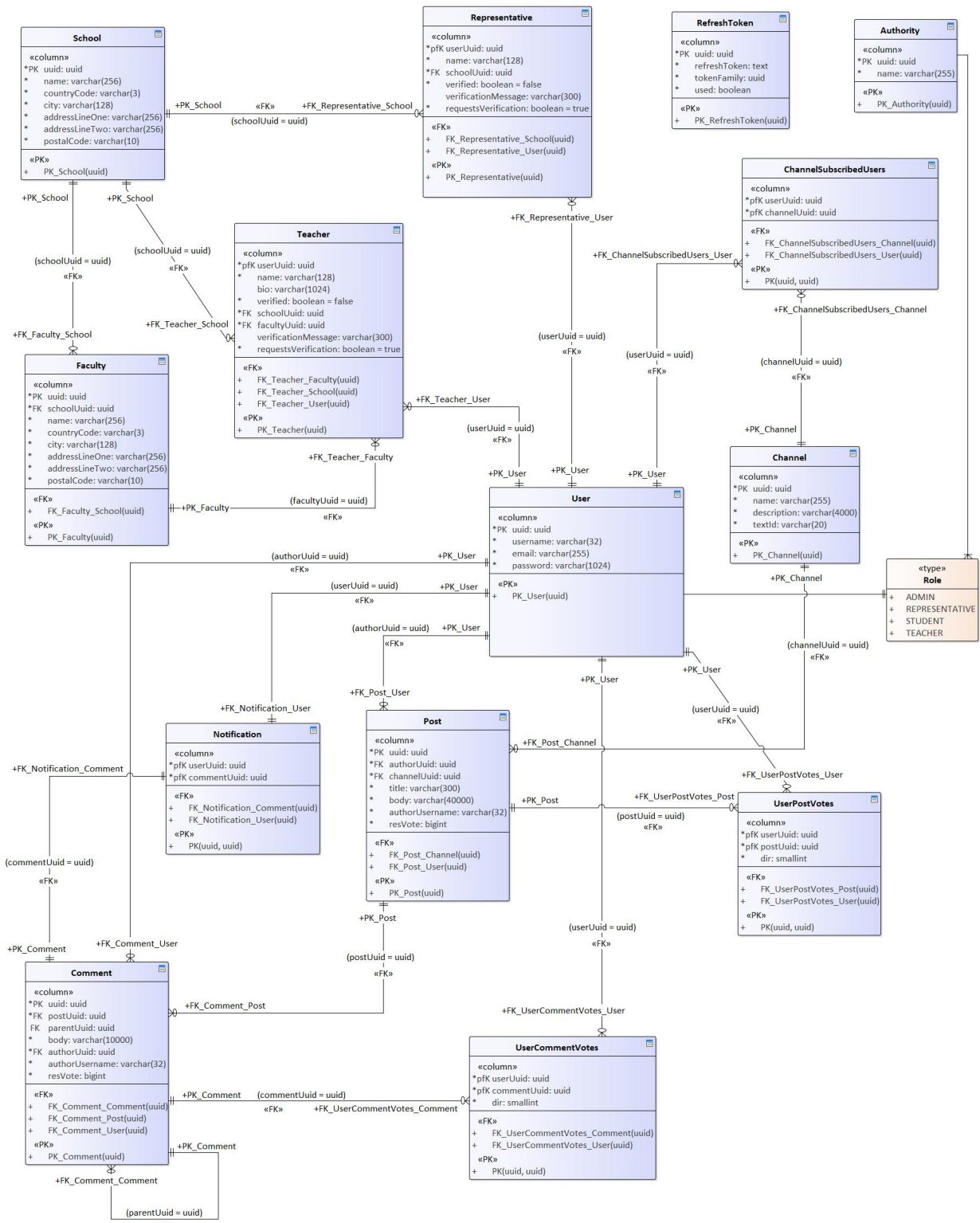


Figure A.1: Database Structure

Appendix B

Example of End-To-End Test

```
describe("Posts Module Tests", () => {
  let app: INestApplication;
  let accessToken = "";

  // 1. arrange
  beforeAll(async () => {
    // initializing the application
    app = await TestUtils.initApp();
    await app.init();

    // logging in and obtaining the access token
    const payload: LogInUserRequestDto = {
      email: "student@email.com",
      password: "pass123"
    };
    await request(app.getHttpServer())
      .post("/auth/login")
      .send(payload)
      .expect((res) => {
        accessToken = res.body.accessToken;
      });
  });

  it("should create a post in the demo channel", async () => {
    // 2. act
    const payload: CreatePostRequestDto = {
      channelUuid: "ff5958fb-d1be-426b-ab5b-a3302ec803f0",
      title: "post title",
      body: "post body",
    };
    const response = await request(app.getHttpServer())
      .post("/posts")
      .set("Authorization", TestUtils.bearer(accessToken))
      .send(payload)
      .expect(201);

    continue on the following page ...
  });
});
```

B. Example of End-To-End Test

```
// 3. assert
expect(response.body.uuid).toBeDefined();
expect(response.body.uuid).not.toEqual("");
});

// other tests of the posts module
})
```

Appendix C

User Test Scenarios

C.1 Teacher Test

Thank you for agreeing to test my application. This is an application used for anonymous communication between teachers and students. Only students get to communicate anonymously. The application allows creating channels, creating posts in them, and commenting under posts.

Please test the application using the instructions below.

C.1.1 Part 1. Signing up as a teacher

1. Open the application at <https://comm-app-2gvj8.ondigitalocean.app>.
2. Sign up as a teacher:
3. For the school choose any school from the following (because only these three have been registered in the system so far):
 - České vysoké učení technické v Praze
 - Vysoká škola ekonomická v Praze
 - Univerzita Karlova
4. For the faculty choose any faculty from the list of available ones. Try searching for the faculty by typing “fakulta”.
5. After signing up successfully, log in using your credentials.
6. Make sure that you see a notification at the top of the page saying that your account needs to be verified first.

C.1.2 Part 2. Using the application as a teacher

1. Since the account you just created must be verified first, please use the following credentials in this part of the test.
 - Email: teachertest@gmail.com
 - Password: teacher2022

C. User Test Scenarios

2. Change the color scheme, if you prefer. By default, the system mode will follow your system preferences. Alternatively, you can set the light, or the dark mode permanently.
3. Create a channel using a button in the navigation bar at the top of the page.
4. Navigate to the newly created channel using one of the following ways:
 - after creating a channel you should be redirected to it automatically
 - search for it by typing the name or the channel ID in the search bar at the top of the page
 - go to the URL directly:
`https://comm-app-2gvj8.ondigitalocean.app/channels/<your channel ID>`
5. Join your channel.
6. Create a post in your channel.
7. Leave a comment under this post.
8. Vote on your post and your comment.
9. Edit your channel name and/or channel ID and/or description.
10. Edit your comment.
11. Delete your comment.
12. Delete your post.
13. Join some more channels. The following channels are available:
 - demo
 - forstudents
 - test
 - or feel free to create more channels yourself and join them
14. Return to the home view (the feed).
15. Make sure you can see the posts from the channels that you have joined.
16. Navigate to your profile.
17. Change your name and bio.
18. Log out.

■ C.1.3 Part 3. Testing Feedback

1. Please specify if you have encountered any errors. Did you see any error alerts? If so, where?
2. Did you find any task complicated, confusing, or unintuitive to complete?
3. Do you have any suggestions on how the user experience could be improved?

This is the end of the test.

Thank you for helping me test the application!

If you have any more suggestions, I will be glad to hear them.

■ C.2 Student Test

Thank you for agreeing to test my application. This is an application used for anonymous communication between teachers and students. Only students get to communicate anonymously. The application allows creating channels, creating posts in them, and commenting under posts.

Please test the application using the instructions below.

■ C.2.1 Part 1. Signing up as a student

1. Open the application at <https://comm-app-2gvj8.ondigitalocean.app>.
2. Sign up as a student.
3. After signing up successfully, log in using your credentials.

■ C.2.2 Part 2. Using the application as a student

1. Make sure you are still logged in after finishing the first part of the test.
2. Change the color scheme, if you prefer. By default, the system mode will follow your system preferences. Alternatively, you can set the light, or the dark mode permanently.
3. Create a channel using a button in the navigation bar at the top of the page.
4. Navigate to the newly created channel using one of the following ways:
 - after creating a channel you should be redirected to it automatically
 - search for it by typing the name or the channel ID in the search bar at the top of the page
 - go to the URL directly:
<https://comm-app-2gvj8.ondigitalocean.app/channels/<your channel ID>>

5. Join your channel.
6. Create a post in your channel.
7. Leave a comment under this post.
8. Vote on your post and your comment.
9. Edit your channel name and/or channel ID and/or description.
10. Edit your comment.
11. Delete your comment.
12. Delete your post.
13. Join some more channels. The following channels are available:
 - demo
 - forstudents
 - test
 - or feel free to create more channels yourself and join them
14. Return to the home view (the feed).
15. Make sure you can see the posts from the channels that you have joined.
16. Navigate to your profile.
17. Refresh the username. Make sure you see the prompt with your new username.
18. Change the email.
19. Change the password.
20. Log out.
21. Log back in using your new credentials.
22. Log out again.

■ C.2.3 Part 3. Testing Feedback

1. Please specify if you have encountered any errors. Did you see any error alerts? If so, where?
2. Did you find any task complicated, confusing, or unintuitive to complete?
3. Do you have any suggestions on how the user experience could be improved?

This is the end of the test.

Thank you for helping me test the application!

If you have any more suggestions, I will be glad to hear them.

Appendix D

Front End Wireframes

The image displays two wireframes for a 'Communication App' interface, each enclosed in a light gray rounded rectangle. The top wireframe is for the login page, and the bottom wireframe is for the signup page.

Top Wireframe (Login Page):

- Title: Communication App
- Buttons: Log In (left), Sign Up (right)
- Form fields: Email, Password
- Submit button: Log In

Bottom Wireframe (Signup Page):

- Title: Communication App
- Buttons: Log In (left), Sign Up (right)
- Form fields: Role (dropdown menu with 'Teacher' selected), Name, School (dropdown menu with 'České vysoké učení technické v Praze' selected), Faculty (dropdown menu with 'Fakulta elektrotechnická' selected), Email, Password
- Submit button: Sign Up

Figure D.1: Login Page Wireframe (top) - Signup Page Wireframe (bottom)

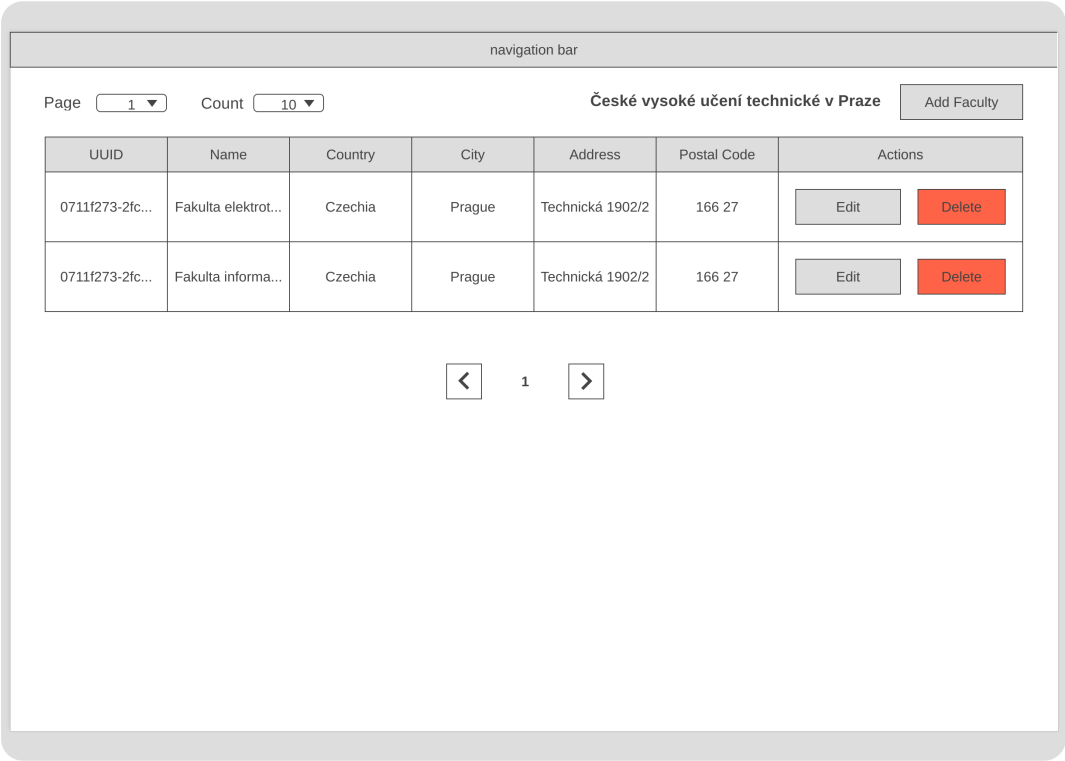
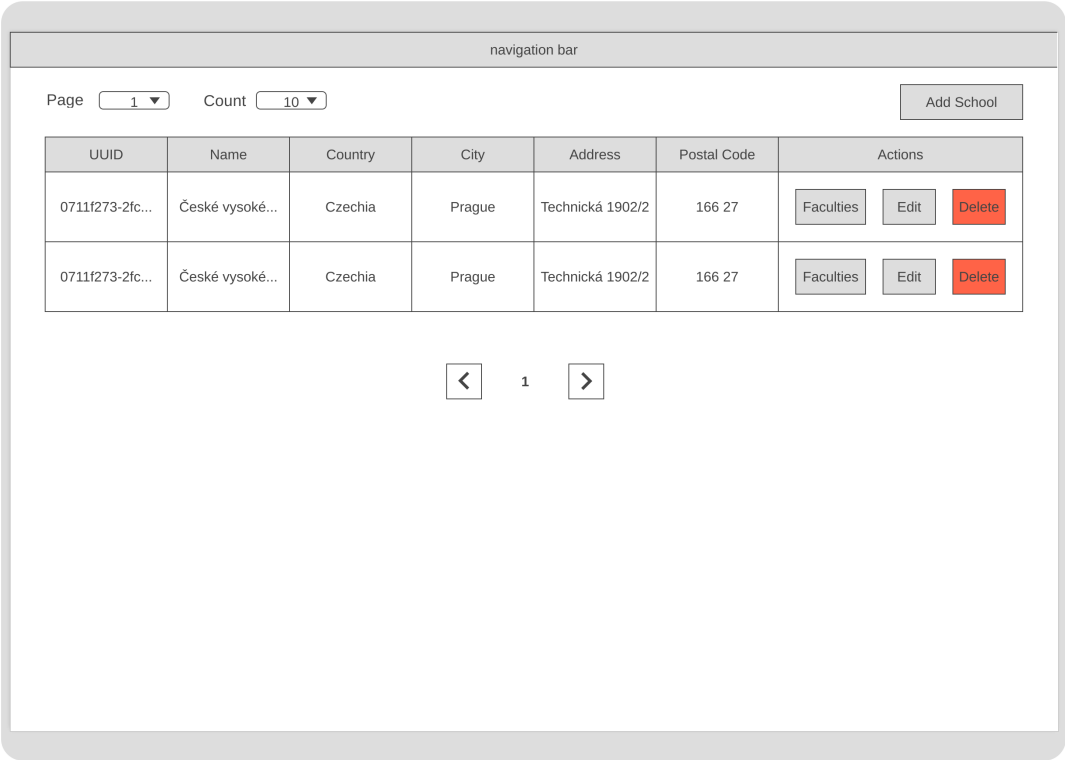


Figure D.2: Schools Page Wireframe (top) - Faculties Page Wireframe (bottom)

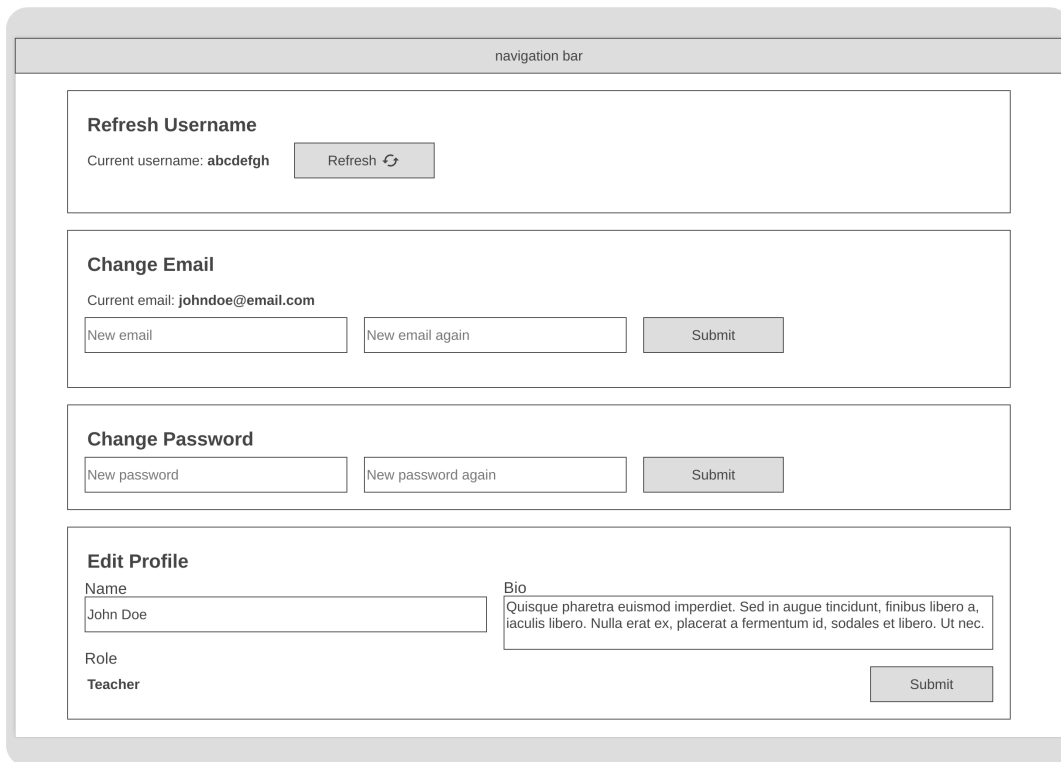
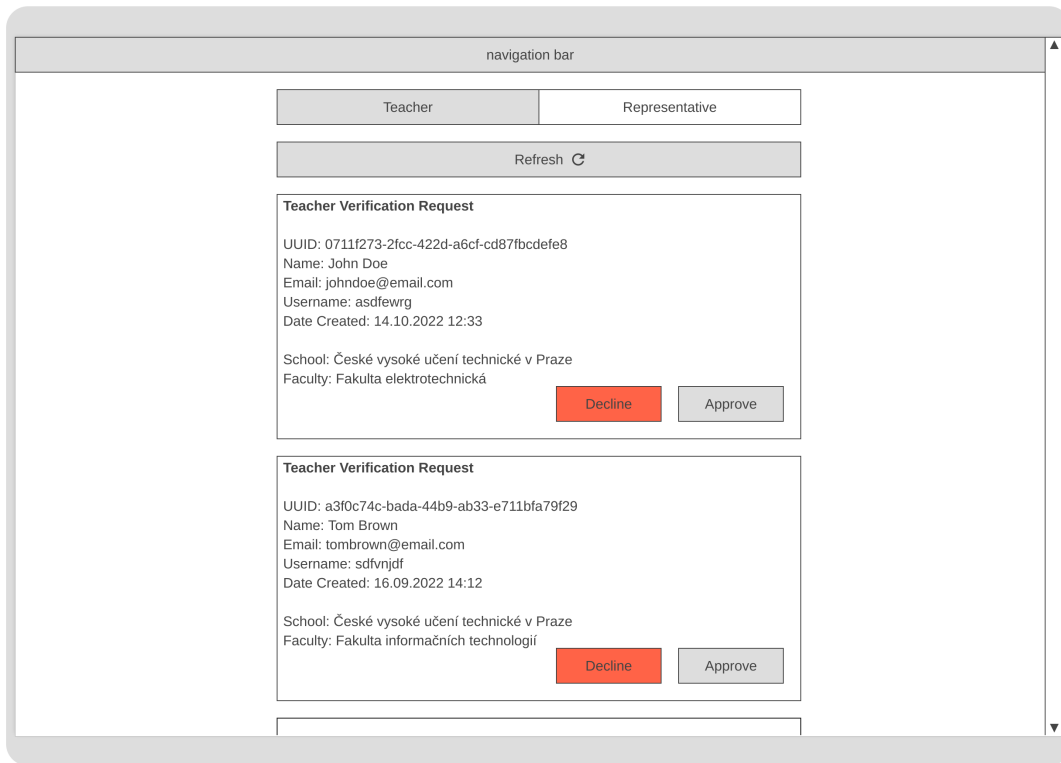


Figure D.3: Requests Page Wireframe (top) - Account Page Wireframe (bottom)

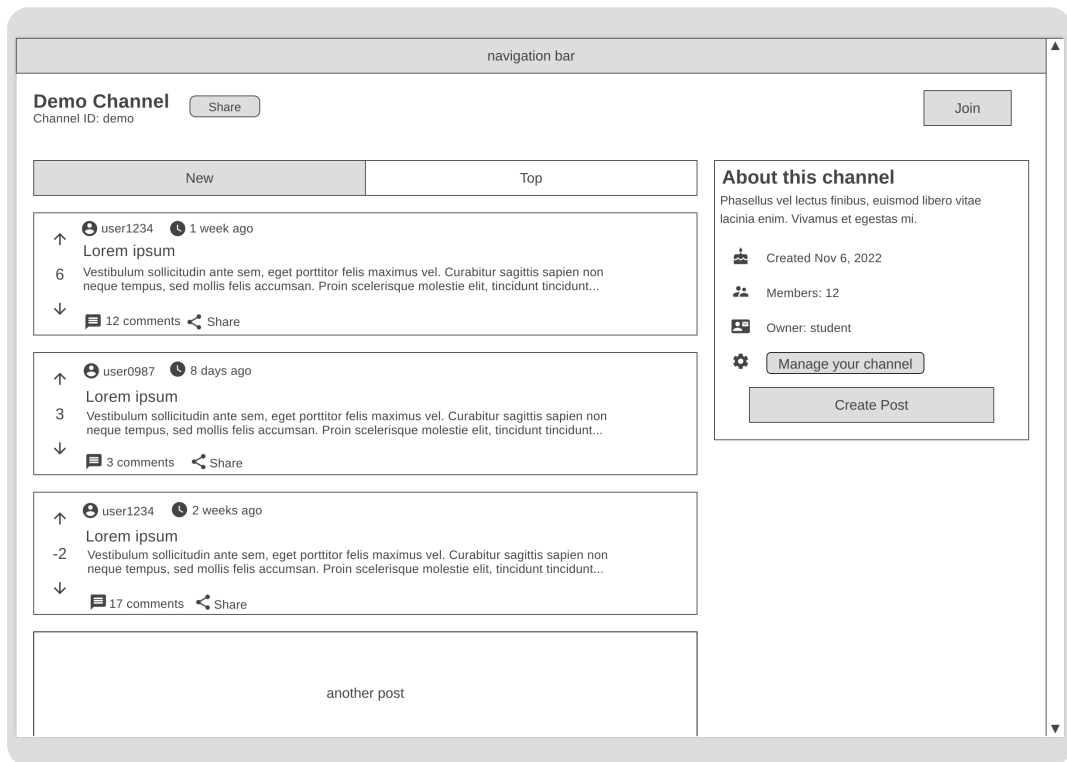
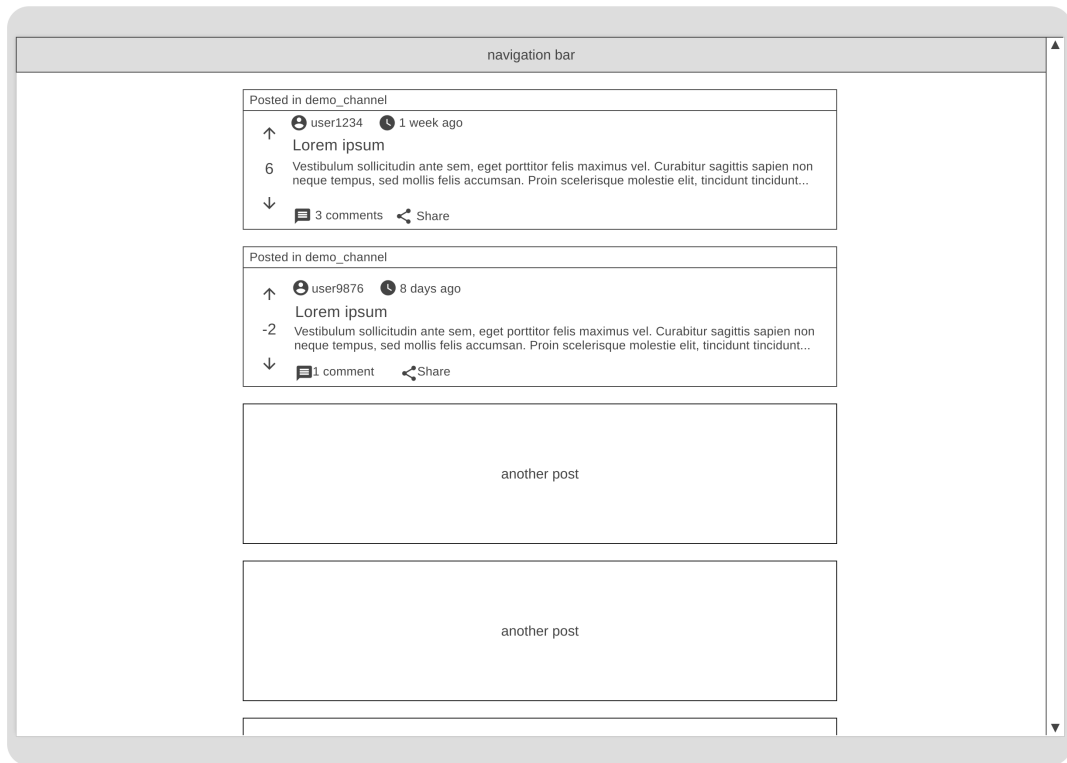


Figure D.4: Feed Page Wireframe (top) - Channel Page Wireframe (bottom)

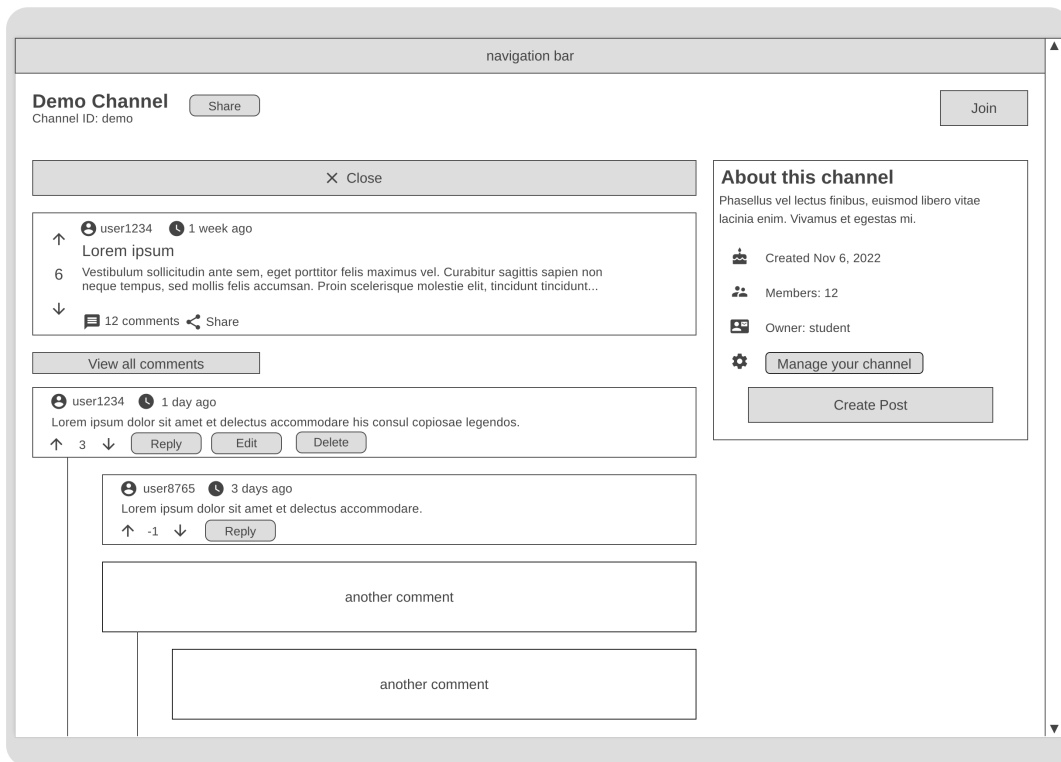
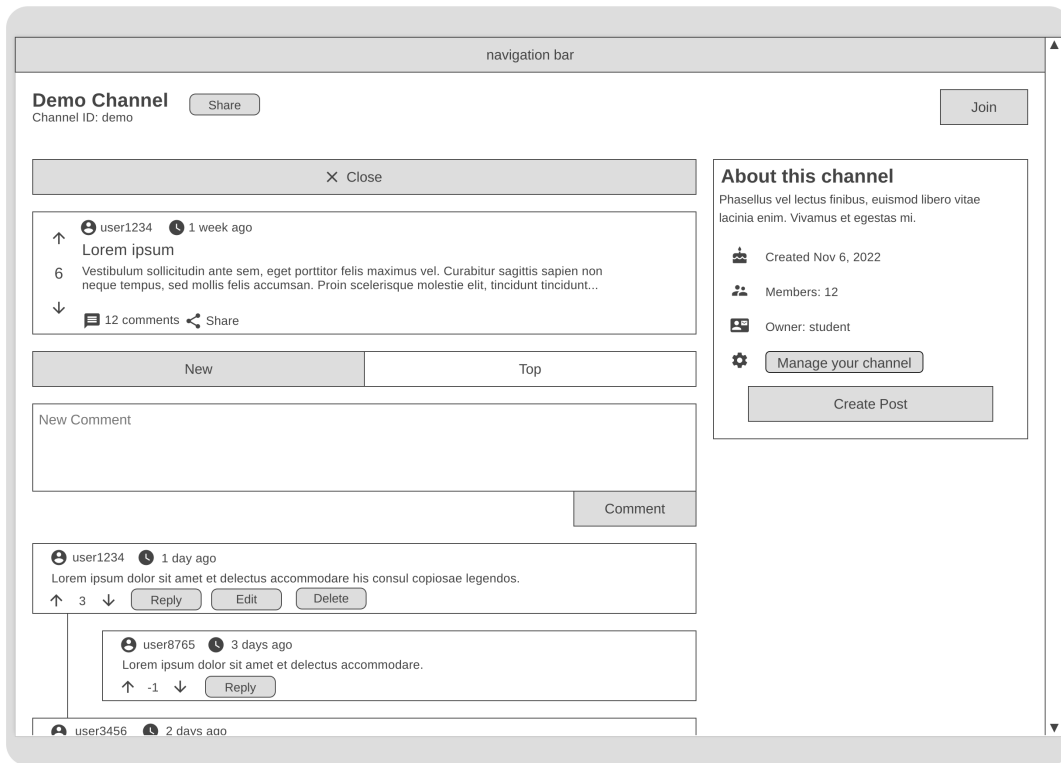


Figure D.5: Post Page Wireframe (top) - Comments Page Wireframe (bottom)

Appendix E

Switching Color Scheme

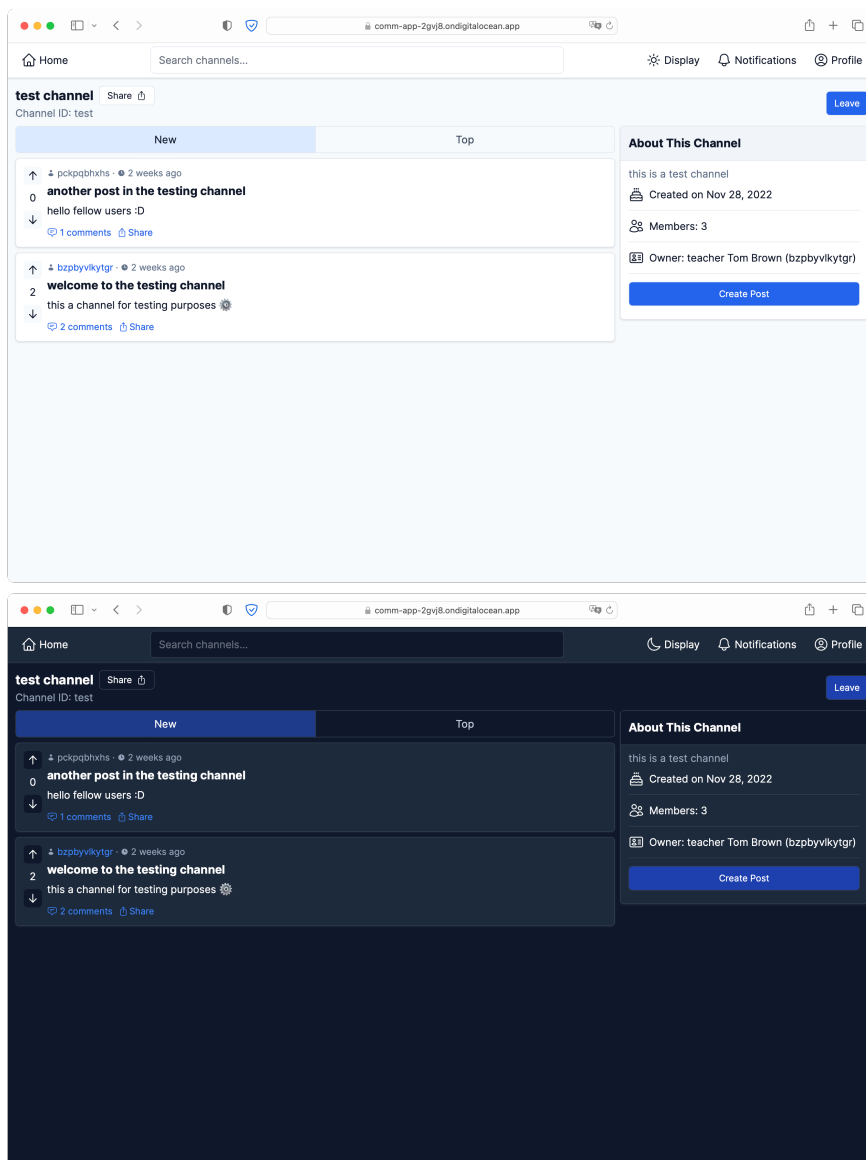


Figure E.1: Light Mode (top) - Dark Mode (bottom)

Appendix F

Application Demonstration

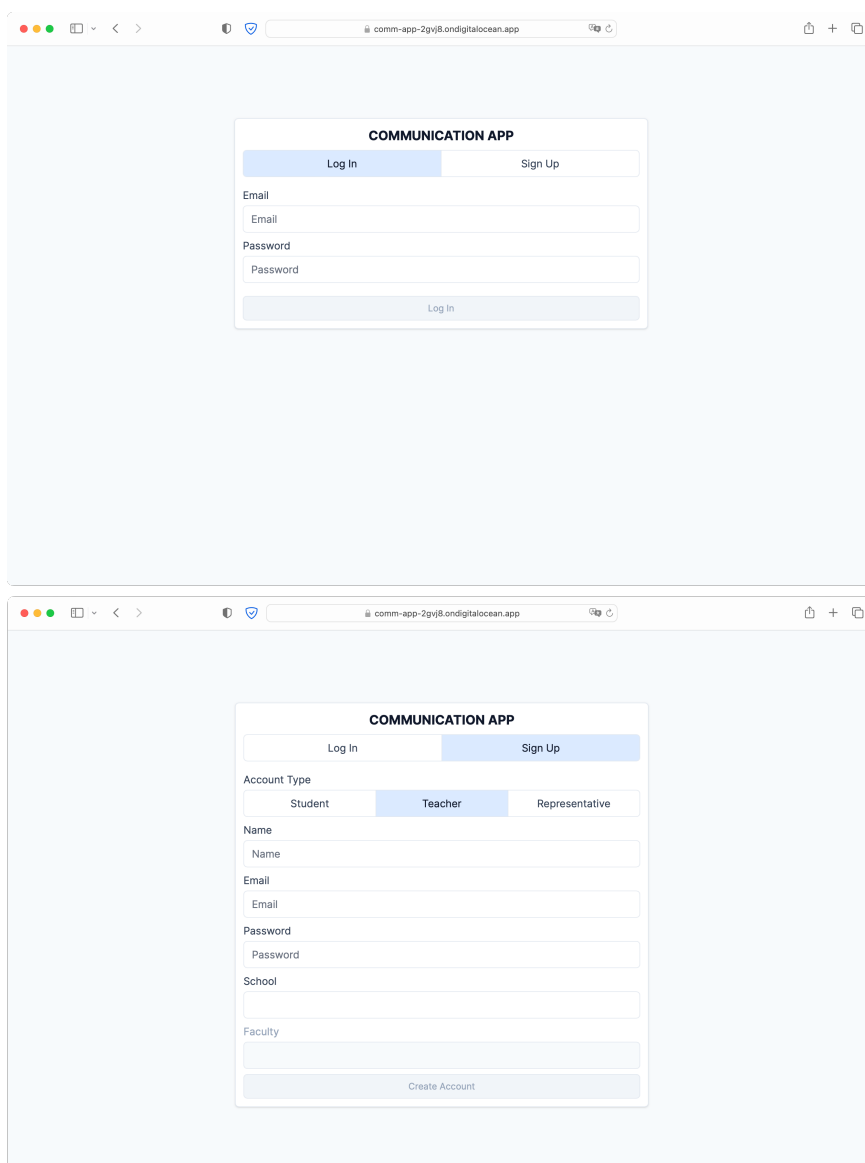


Figure F.1: Login Page (top) - Signup Page (bottom)

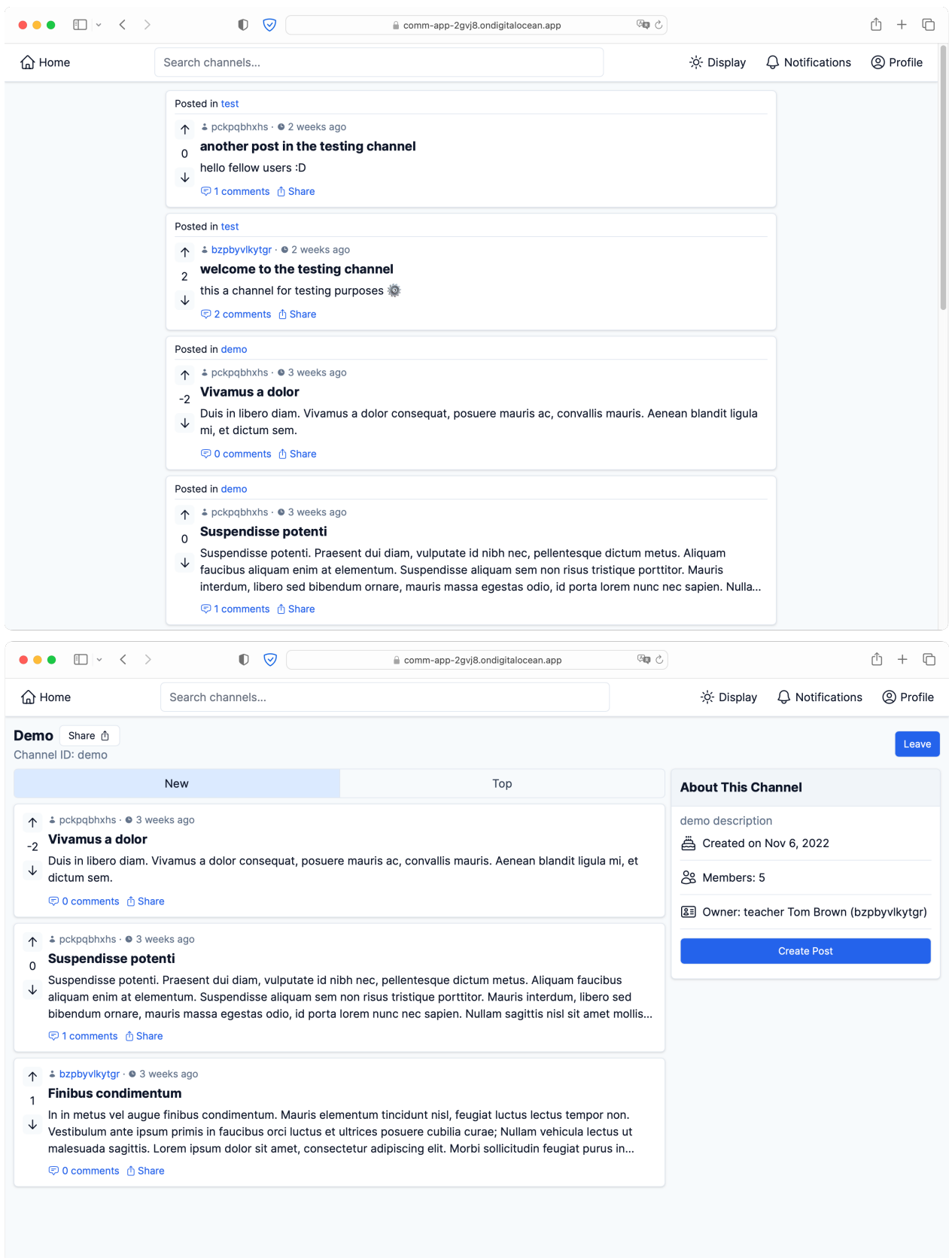


Figure F.2: Feed Page (top) - Channel Page (bottom)

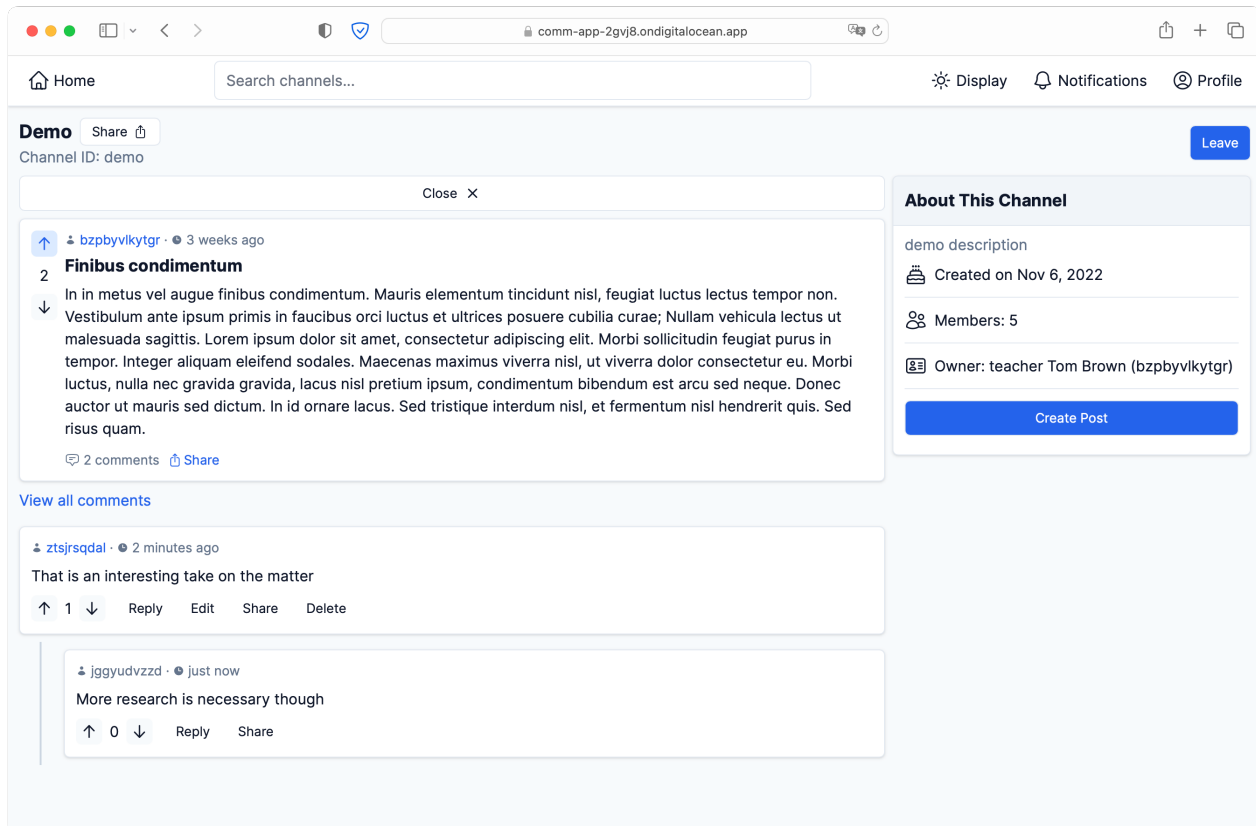
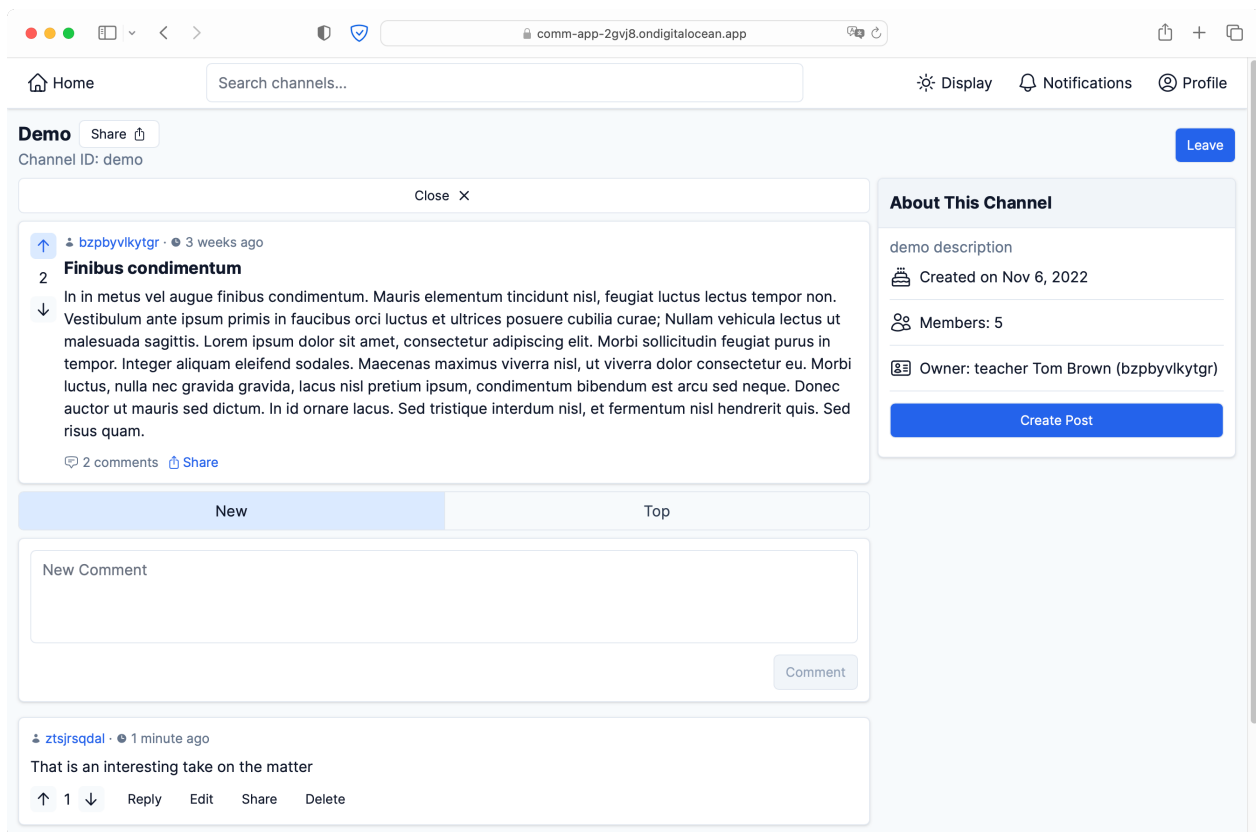


Figure F.3: Post Page (top) - Comment Page (bottom)

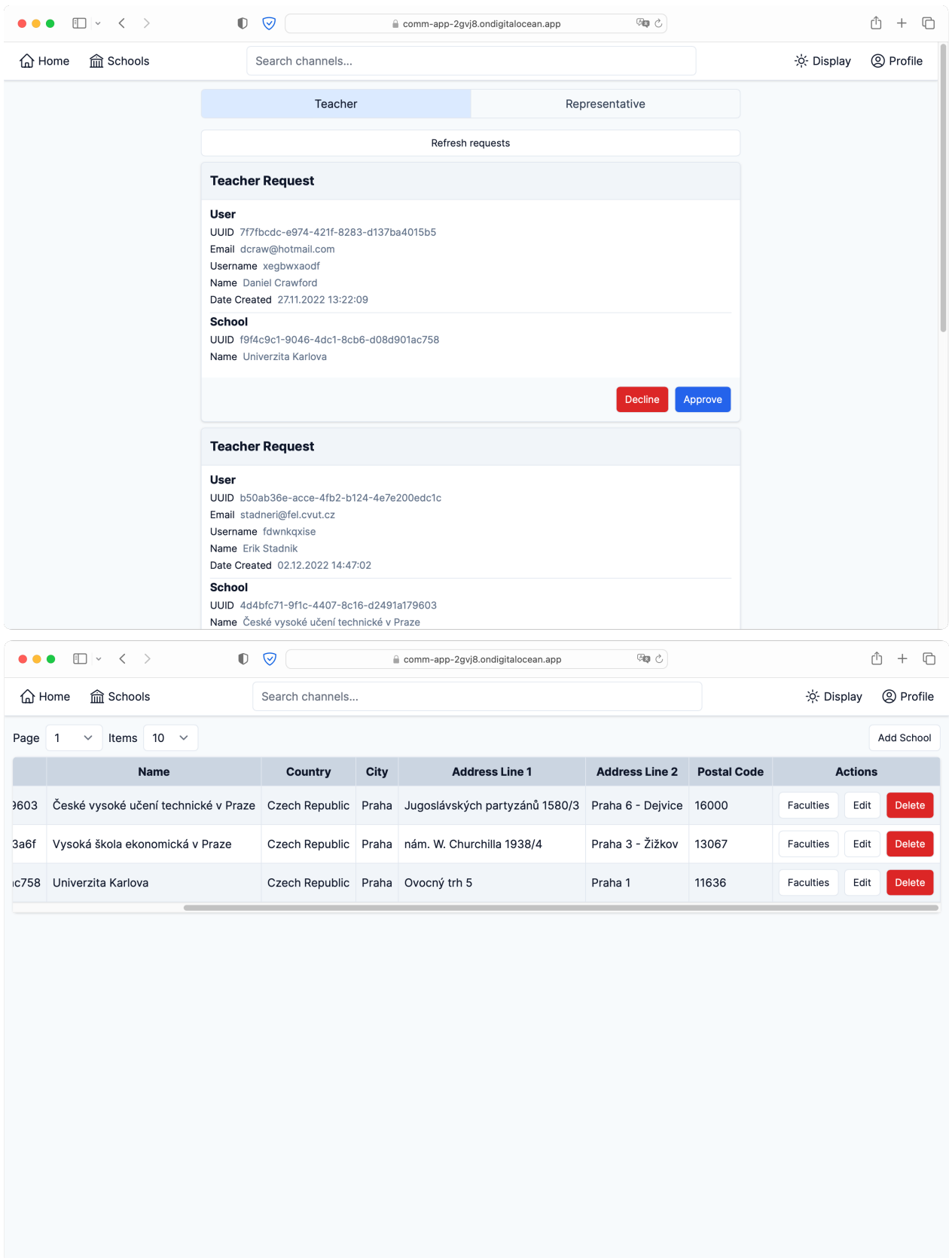


Figure F.4: Requests Page (top) - Schools Page (bottom)

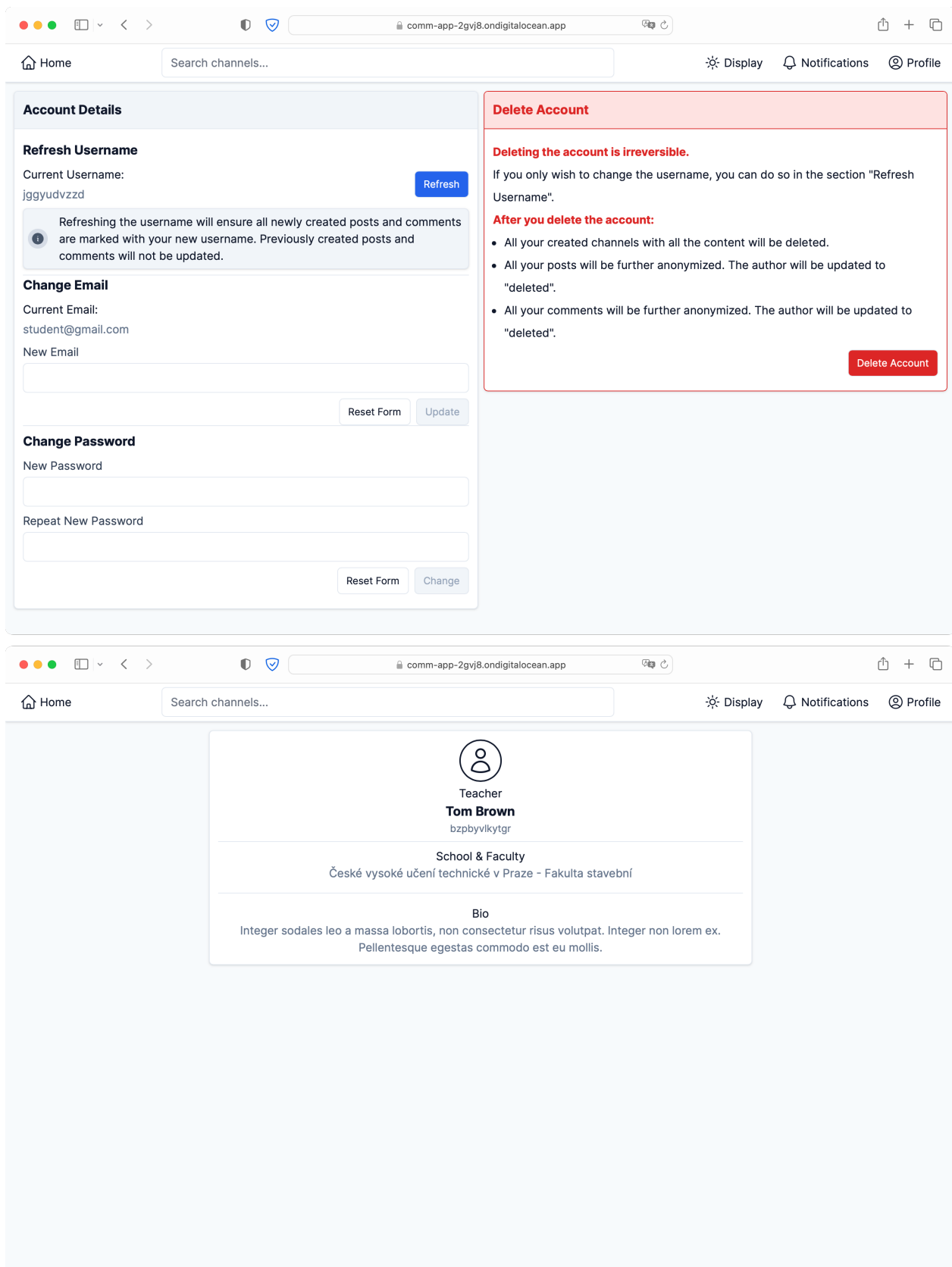


Figure F.5: Student Account Page (top) - Teacher Public Profile Page (bottom)

Appendix G

Bibliography

- [1] Using Anonymity in Online Interactive EFL Learning: International Students' Perceptions and Practices, Chen Chen, The University of Sydney, Australia, 2019. Accessed: 04.07.2022.
<https://files.eric.ed.gov/fulltext/EJ1214276.pdf>
- [2] How Peer Pressure Can Lead Teens to Underachieve—Even in Schools Where It's “Cool to Be Smart”, Leonardo Bursztyn, Georgy Egorov, Robert Jensen, 02.10.2018. Accessed: 04.07.2022.
<https://insight.kellogg.northwestern.edu/article/peer-pressure-can-lead-teens-underachieve-schools-cool-to-be-smart>
- [3] Anonymity and in Class Learning: The Case for Electronic Response Systems, Mark Alistair Freeman, Paul Blayney, Paul Ginns, November 2006. Accessed: 04.07.2022.
<https://www.researchgate.net/publication/259695045>
- [4] Online Teaching: Encouraging Collaboration Through Anonymity, Andrea Chester, Gillian Gwynne, 26.06.2006. Accessed: 04.07.2022.
<https://onlinelibrary.wiley.com/doi/full/10.1111/j.1083-6101.1998.tb00096.x>
- [5] Microsoft Teams - Video Conferencing, Meetings, Calling. Accessed: 15.11.2022.
<https://www.microsoft.com/en-us/microsoft-teams/group-chat-software>
- [6] Zoom - One Platform to Connect. Accessed: 15.11.2022.
<https://zoom.us>
- [7] Discord - Your Place to Talk and Hang Out. Accessed: 15.11.2022.
<https://discord.com>
- [8] Slido - Audience Interaction Made Easy. Accessed: 15.11.2022.
<https://www.slido.com>
- [9] Mentimeter - Interactive Presentation Software. Accessed: 15.11.2022.
<https://www.mentimeter.com>

- [10] Functional and Nonfunctional Requirements: Specification and Types, 23.07.2021. Accessed: 22.11.2022.
<https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/>
- [11] Use Cases. Accessed: 22.11.2022.
<https://www.usability.gov/how-to-and-tools/methods/use-cases.html>
- [12] What Is a REST API, 08.05.2020. Accessed: 24.11.2022.
<https://www.redhat.com/en/topics/api/what-is-a-rest-api>
- [13] JSON Web Tokens. Accessed: 24.11.2022.
<https://jwt.io/>
- [14] What Is a Wireframe? Accessed: 25.11.2022.
<https://careerfoundry.com/en/blog/ux-design/what-is-a-wireframe-guide/>
- [15] What Is Wireframing? Accessed: 25.11.2022.
<https://www.experienceux.co.uk/faqs/what-is-wireframing/>
- [16] What Is Typescript? Javascript Typescript Brief History. Accessed: 26.11.2022.
<https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.htmlwhat-is-javascript-a-brief-history>
- [17] Type Systems in Software Explained With Examples, 27.09.2020. Accessed: 26.11.2022.
<https://thevaluable.dev/type-system-software-explained-example/>
- [18] What Are Database Migrations? Advantages and Disadvantages of Migration Tools. Accessed: 26.11.2022.
<https://www.prisma.io/dataguide/types/relational/what-are-database-migrationswhat-are-database-migrations>
- [19] Common Table Expressions in PostgreSQL. Accessed: 25.07.2022.
<https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-cte/>
- [20] Nestjs: A Progressive node.JS Framework. Accessed: 01.08.2022.
<https://docs.nestjs.com/>
- [21] Swagger OpenAPI Specification. Accessed: 27.11.2022.
<https://swagger.io/specification/>
- [22] What Is an SSL Certificate – Definition and Explanation. Accessed 27.11.2022.
<https://www.kaspersky.com/resource-center/definitions/what-is-a-ssl-certificate>
- [23] Passport Concepts. Authentication. Accessed: 03.08.2022.
<https://www.passportjs.org/concepts/authentication/>

- [24] Class-Validator: Decorator-Based Validation. Accessed: 05.08.2022.
<https://github.com/typestack/class-validator>
- [25] React: A Javascript Library for Building User Interfaces. Accessed: 23.11.2022.
<https://reactjs.org>
- [26] Redux: A Predictable State Container for JS Apps. Accessed: 23.11.2022.
<https://redux.js.org/>
- [27] Redux Toolkit: The Official, Opinionated, Batteries-Included Toolset for Efficient Redux Development. Accessed: 23.11.2022.
<https://redux-toolkit.js.org/>
- [28] Redux Toolkit Query Overview. Accessed: 10.08.2022.
<https://redux-toolkit.js.org/rtk-query/overview>
- [29] React Router Overview. Accessed: 11.08.2022.
<https://reactrouter.com/en/main/start/overview>
- [30] Formik: Build Forms in React, Without the Tears. Accessed: 11.08.2022.
<https://formik.org>
- [31] Digital Ocean App Platform: Fully-Managed Infrastructure. Accessed: 15.11.2022.
<https://www.digitalocean.com/products/app-platform>