HALP

Group 12

By

Charlotte Celine Thorjussen, Nikolai Bunch Eidsheim, Ole-Johan Øvreås, Sivert Espeland Husebø, Sondre Horpestad & Markus Hwan Tørå Hagli

IKT205

Application Development

Supervised by Assistant Professor Christian Auby

Faculty of Technologies and Science

University of Agder

Grimstad, spring 2023

Abstract

HALP (Help Assistant Lab Program) is created and aimed to be a mobile application for getting help during our school's lab hours that is mobile-friendly and easy to use. The HALP application is an addition to the web application that we made last semester but with some tweaks. The users of the app are the students that are visiting labs, student assistants, and teachers [21]. The old system was just an Excel sheet, that we wish to replace with our app, which has a more up-to-date feeling, and still aims to be as easy to use as the old system. The mobile application is connected to Timeedit so it can fetch the schedule of the classes at the University, and works for those classes that have "lab" or "øving" in their schedule. The application is built with React Native, and since it is based on our last project we decided that we wanted to use the same backend. Our product was not entirely finished, but we worked to get most core features of the application to work properly. In the end, we realized we did not have the time to get all of the quality-of-life features for the users that create an account, features that could be implemented in future work. This report is documenting our process, methods, and tools and ends with a discussion of what could be done better and what we were happy about.

Mandatory group declaration

The individual student is himself responsible to know which aids are allowed, the guidelines for using them, and the rules related to the use of sources. The declaration must inform the students about their responsibility, and the consequences cheating would entail. Lack of such a declaration does not release the student from their responsibility.

1.	We hereby declare that our answer is our own work, and that we have not used other peoples sources or received other help than that which has been declared.	Yes
2.	**We further declare that this answer:** • Has not been used on another exam by another department/university/college domestic or foreign. • Does not reference any other work without being stated. • Does not reference the individual students own earlier work without being stated. • Has all references stated in the bibliography. • Is not a copy, duplicate or transcript of other peoples work or answer.	Yes
3.	We are familiar with that violations of the rules mentioned above is considered cheating and may entail cancellation of the exam, and exclusion from universities and colleges in Norway, cf. Universities and Colleges Act §§4-7 and 4-8 and Examination Regulations §§ 31.	Yes
4.	We are familiar with the fact that submitted assignments may be checked for plagiarism.	Yes
5.	We are familiar with the fact that the University of Agder will treat all cases where there is suspicion of cheating by the university's guidelines for processing cases of cheating.	Yes
6.	We have looked into the rules and guidelines of using sources and references on the library's own website.	Yes
7.	We have by majority come to an understanding if the individual effort within the group is noticeably different we could choose to be graded individually. Ordinarily the group gets a collective grade.	No

Publishing agreement

Power of attorney to the electronic publication of the assignment. The publishers have a copyright to the assignment. This entails that the publishers have exclusive rights to make the assignments available to the general public. (The Copyright Act. §2).
Assignments that are not public, or are confidential will not be published.

We hereby give the University of Agder the ability to make the assignment available for electronic publication.	Yes
Is the assignment confidential?	No
Is the task exempt from public disclosure?	Yes

Preface

This report documents the work made on our final project in the course IKT205g-23V. We are a group of 6 computer science students that decided to make a mobile application for a student assistant lab program that is based on our last project in IKT201g-22h [21]. By the end of the project, we have learned a lot about using tools to create our mobile application, development process, methods, design concepts, and working together as a big group. During the project, we have consistently used Jira for scheduling and logging, Gitlab for easier development collaboration, pipelining, testing, and file sharing, and Discord for communication.

The assignment was given by The University of Agder, Institute for Science and Technology.

Furthermore, we would like to thank our Assistant Professor Christian Auby for his great guidance, availability, and support throughout our project.

Contents

List of Figures

1 Introduction

In the IKT205 application development course, we were assigned a project to create a mobile application where we utilized tools used to develop a mobile application and use fundamental technology with our app, such as data storage, authentication, design concepts, and general frontend, and backend implementation. This report will take the reader through the various processes of development to give an insight into how we started with an idea, to the actual application we ended up creating. In the next subsections, we will describe our background for the project, our vision, and how our product stands out from other similar products to make it successful.

1.1 Background

The idea of creating this application stems from the concept of creating a simple queuing system to be used in university lab classes. The lab class queues are currently made using an Excel spreadsheet, which people have to find using a link, and then write their name and a description of the problem onto. The current system can be defined as unorganized, unintuitive, and may for some be difficult to use. Therefore, we wanted to create an application that addresses these issues; it should give the students a clear interface to create tickets and show their place in the queue. It should give the student assistants an overview of who is next in line, and who has been helped by providing both a helplist and an archive. It should look good, and provide the users with a good user experience. These are all things that the simple spreadsheet approach does not provide. We then combine this idea with wanting to make the existing HALP [21] system simpler to use, and even more intuitive, by providing the users with a cross-platform application they can keep on their phone, instead of having to go to a website each time they attend a lab class.

1.2 Product vision

Our vision for the mobile application is to streamline the way students and student assistants interact during labs. By creating an app, we will make HALP accessible anytime and anywhere. Our app will be designed specifically for educational institutions looking for a streamlined ticket management solution. The app will integrate TimeEdit in order to synchronize lab hours.

As mentioned above, the mobile application will be integrated with the existing HALP system, and as such, this is one of our primary visions for the project.

1.3 Similar products

Upon researching online, we discovered several systems resembling our own project, one of them being ManageEngine [26][21]. The predominant ticket systems we came across are categorized as "help desks", which are digital platforms encompassing various features like chat, messages, phone, and email, among others. These features share similarities with our project's vision but are generally too extensive and include unnecessary features that are not applicable to our specific work goals. What sets our product apart from others is its integration with TimeEdit. We have not found any other service that offers this capability. TimeEdit is the primary scheduling tool utilized by most universities in Norway and some international universities [37]. Our product caters to a niche market, which is more suitable since it is tailored to our institution's requirements.

2 Process

2.1 Method

In order to achieve our project vision, we adopted a hybrid development approach comprising both waterfall and agile methodologies. The waterfall methodology is commonly employed in larger projects, where requirements and scopes are well-defined. Conversely, the agile methodology is better suited for smaller projects, where changes can be made during the development process. This approach is more responsive and flexible, as changes are incorporated into the development process and customer feedback is taken into account for scope modifications.

Our decision to initiate the development process using the waterfall methodology was based on our previous experience with the product in the IKT201 project. This allowed us to establish the necessary requirements for the product's completion. Moreover, we were able to reuse the visual and structural aspects of the previous project with some modifications to support mobile-friendliness. Subsequently, we continued with the agile methodology, specifically the scrum approach, to ensure flexibility and efficiency in the development process.

2.2 Planning

As we are the same group as last time, the communication should be easier and hopefully, more efficient working together. As for the requirements for the project, we have to plan for how we want to do the scheduling for the project. We decided that we want to have two physical meetings a week where we worked together. The tools we were going to use used for planning were Discord, Jira, and physical meetings. The plan is to use a combination of Scrum and Waterfall. We want to start off with a Waterfall approach, where we are going to make the requirements and design. After this is done, we want to go for a Scrum working approach where we want to split our tasks into Scrum sprints and divide said tasks among the members.

2.3 Waterfall meetings

The plan is to first have the meetings for planning and decide what requirements we are going to make, and how the design should look like. Our decision is to meet every Monday and Friday from 10 O'clock to work on requirements first, and then do the design.

2.4 Waterfall tasks

The requirements should be based on the requirements we did last semester from our HALP website. These requirements have to be made and written down in our report, and it is important that the discussions about the requirements are going to be documented right away. We decided that the design should be made in Figma and use Material Design as a template. This provided us with fitting colors to use, as well as sizing, positioning, and user-friendliness.

The requirements and the design we decided as a group, but the process of making each individual screen will be divided among the members. The requirements and the design have to be finished in both Figma and in the report before we can move on to the implementation part of the project.

2.5 Transition from Waterfall to Scrum

Our transition from Waterfall to Scrum will be done after we have finished everything we could have done in the requirements and design phase. This is where we are going to go from the foundation that we have made to start the development of the actual mobile application. Our Scrum method will be described in the next sections on how we are going to work on the implementation.

2.6 Scrum Meetings

Similar to the waterfall meetings, we plan to meet every Monday and Friday at 10 o'clock and work for as long as we needed to get the planning, structure, implementation, and coordination for our project sorted out.

Mondays are set as the scrum meeting. The first 15-30 minutes of every Monday shall be the Scrum meetings where we discuss how the sprint has gone, what tasks are done / not done, ask for assistance, and quick recapture of the Sprint. The next meeting agenda is to merge the Sprint into the main branch, and creating a new sprint, assign sprint tasks to the sprint, and split the tasks among the members.

Friday's meeting was the evaluation/update day where we went through the status of the sprint tasks.

2.7 Sprints

The implementation part is where we had planned to start with Scrum. We want to divide the tasks into short cycles, called Scrum sprints. The first step is to write down all the tasks we could get from the user stories/requirements and then evaluate the priorities in order of what we needed to complete for the implementation. We are planning to have one-week sprints, with a new sprint starting each Monday.

2.8 Scrum roles

In this project, we did not assign any specific roles for our team members, but we did decide that Charlotte and Sondre was the one that got the main roles for making the Sprints under the scrum meetings, and Nikolai had the main role for the backend. The rest of the members had other responsibilities like design, logic, and developing the screens. Though we had worked together before, developing mobile applications was a new area for most of us, so we all want to contribute as many aspects of the project to have the best learning outcome. We have no product owner as this is a school project, and we have to fill the requirements ourselves.

2.9 Version control

In the process of making HALP, we will keep track of our Versions with Git. We will make a readme file on how to name, use, and merge the various branches, the file can be found in the appendix folder. We will have three different types of branches, "tasks branches", an active "Sprint branch" and a "main branch". The task branches are the branches that we are going to work on each sprint, and the naming convention of the branches is set to be CHANV2-, the number of the branch, and the name of the Jira task at the end, e.g. "CHANV2-48 Global Stylesheet Implementation", so we know the connection between the task and branch. The sprint branches are the branches that are going to be made on each sprint and are only named "Sprint" with an underscore with its corresponding sprint number, e.g. "Sprint_4". When a task is done, we can pull the current sprint branch into the task branch to handle the merge conflicts, and then we can merge the task branch into the sprint branch. This should be done continuously throughout the week. It's a criteria that the sprint branch must work properly, and that the main branch also has to work properly, in the sense of it can be built and run without problems. We will set up a pipeline that automatically builds branches on commits, in order to make sure that there are no issues. When a task is merged into the Sprint branch, it also should delete branches that had been merged in, so it was easier to keep track of which branches that was in use. Finally, at the end of each scrum sprint, we can merge the sprint branch into the main, so main always was updated for each sprint.

2.10 Usage of Discord bot

To easier keep track of the merge requests incoming and approved, Sondre made a Discord bot that sent everyone in the team a notification when a merge request is sent in GitLab.

2.11 The process of documenting

From what we experienced last semester, we figured that we needed to be better to document while we were implementing and discussing our decisions. This time we will require that everyone should write down at least some notes if it was something worth writing about. The chapters that could be finished early, we decided should be written as fast as possible, to not forget about everything until the end.

2.12 The Process of the Design

As we mentioned earlier in this chapter we wanted to make the design in Figma. We decided that each member should have the responsibility to make the design of at least one screen each, and if some of the screens look alike it was also included. If there are some screens left, these will be divided among those who got the time to complete them.

2.13 The process of the logic implementation

The logic implementation for each screen will be delegated to the person who made the screen design. The rest of the screens and other functions will then be divided into each Scrum Sprint.

2.14 The process of the backend

As we are using the backend of our last project, we want to use this for our mobile application as well. Nikolai was the one who started to work on it and was thus responsible for creating the bridge between the applications.

2.15 The process of testing

Following the Scrum method we wish to test the Scrum tasks that we were finished with. These requirements can be tested both in the backend and manually by the members, especially the ones who implemented them.

3 Requirements

Since this is a student project like last time, we did not have any "product owner" to provide us with any requirements for the product. Normally on agile projects, the customers continuously have an interaction with the developers [6]. Because we had no "product owner" we discussed amongst ourselves, and used our experience with the currently used system for the labs to come up with the requirements for our product.

In this project, we are using a combination of the Waterfall and Scrum method, so we made all our requirements at the start of the project. Because this project is based on our website project, the requirements are not that different from what we had made before. Though last time we did not have that much experience with making requirements, therefore after some evaluation of the requirements, we rearranged some of the requirements into different categories.

3.1 Functional requirements

3.1.1 Must have

Must have requirements	
Requirement-id	**User stories**
[Req-1]	As a user, I must be able to register an account.
[Req-2]	As a user, I must be able to register an account through email.
[Req-3]	As an admin, I must be able to appoint student assistants.
[Req-3]	As a student assistant or admin, I must be able to log in as a student assistant.
[Req-4]	As a student assistant, I must be able to mark tickets as resolved.
[Req-5]	As a student assistant, I must be able to view the tickets for the class.
[Req-6]	As a user, I must have the ability to create a ticket.
[Req-7]	As a user, I must be able to leave the queue.
[Req-8]	As an admin, I must be able to add links from Timeedit.

3.1.2 Should have

Should have requirements	
Requirement-id	**User stories**
[Req-9]	As a user, I should have the ability to view where I'm in the lab queue after writing the ticket
[Req-10]	As a user, I should be able to update my ticket.
[Req-11]	As a user, I should have the ability to log in.
[Req-12]	As a user, I should be able to register an account through Discord.
[Req-13]	As a user, I should be able to confirm my email after registering.

3.1.3 Can have

Can have requirements	
Requirement-id	**User stories**
[Req-14]	As a user, I can view the queue from discord
[Req-15]	As a user, I can rate the student assistant that helped me.
[Req-16]	As a user, I can have the ability to know:
[Req-16-A]	What labs do I have
[Req-16-B]	What rooms the labs are in
[Req-16-C]	What time the labs are
[Req-17]	As a student assistant, or administrator, I should be able to see the following statistics
[Req-17-A]	The most frequently used labs
[Req-17-B]	Which student assistant solves the most tickets

3.2 Technical requirements

Security	
Requirement-id	**User stories**
[Tech-Req-1]	As a user, I must have my data transferred securely to avoid intruders from accessing it
[Tech-Req-2]	As a user, I must be able to delete my personal data from the application
[Tech-Req-3]	As a user, I should have the possibility to have two-factor authentication
[Tech-Req-4]	As a user, I must have my data secure and only authorized users to be able to access it

3.3 Non-functional requirements

Requirement-id	User stories
[Non-func-1]	The application must be user-friendly
[Non-func-2]	The application must handle multiple users using the app at once
[Non-func-3]	The application must update fast live
[Non-func-4]	The application must implement security measures in order to protect user data
[Non-func-6]	The application must be compatible with Android
[Non-func-7]	The application must have a logo
[Non-func-8]	The application must be adequately reliable

3.3.1 Changes of requirements from the last project that this project is based on

These requirements were added:

1. Req-3 As an admin I must be able to appoint student assistants.
   This was a function that we thought about and implemented early in the last project, but forgot to add as a requirement, but we wanted to add this now as this is an important function of an admin.
2. Req-8 As an admin I must be able to add links from Timeedit
3. All the non-functional requirements are added
4. The technical requirements except the two-factor authentication are added

These requirements was moved around:
We realized that not every requirement was placed in the correct category and therefore we decided to move multiple requirements. Especially a lot of the "must have" requirements had too many requirements that were just working as an improvement rather than a vital function of the application.

1. Req-5: As a user, I must have the ability to view where I'm in the lab queue after writing the ticket.
   This is moved from must-have to should-have.

2. Req-6: As a user, I must have the ability to log in if I want to.
   Moved from must have to should have.
3. Req-7 As a user, I must have the ability to know:
   Req-7-A What labs do I have
   Req-7-B What rooms the labs are in
   Req-7-C What time the labs are
   Moved from Should Have to can have
4. Req-8-B As a user, I must be able to register an account through Discord:
   Moved from Must have to should have
5. Req-9 As a user, I must be able to confirm my email after registering
   Moved from Must have to should have
6. Req-10 As a student assistant, or administrator, I should be able to see the following statistics
   Req-10-A The most frequently used labs
   Req-10-B Which student assistant solves the most tickets
   Moved from should have to can have
7. Req-11 As a user, I should be able to leave the queue.
   Moved from should have to must have

These requirements were deleted:
These were deleted because we discussed that these functions had no purpose in this application.

1. Req- 3 As a student assistant I must be able to assign tickets
2. Req-15 The system can create a prioritized queue
3. Req-16 As a student assistant, I can set tickets on hold.

We also needed more non-functional requirements, as we needed more security, and wanted to specify how the application will perform. We then made some new non-functional requirements, so we validate that the application can perform to our requirements.

4 UI/UX Design

4.1 The design process

The overall design of our app is made to match the HALP application that we made last semester. We intend to combine this design with official design patterns made by reputable companies. Our goal is to make a user-friendly design that is easy on the eyes and flows smoothly. One of the many design choices we will make to accommodate these goals is to implement a dark and a light mode.

4.1.1 Design Systems

As the design system, we have chosen to use Material Design. Material Design is the standard design library developed by Google and is used as default in Android [19]. Since our development team had 50/50 Apple and Android devices, we mainly opted to use this design system over Apple's Human Interface Guidelines, because Charlotte already had some experience working with it.

4.1.2 Colors

Colors complement the HALP colors from the previous project. They have been altered to use the Material Design palette derived from the blue color of the HALP system for primary and accent colors. The design implements a dark and light mode, with colors gathered from this palette.

4.1.3 Font

We have opted to use the font Roboto, the standard font used for Android [23]. This font is therefore also used as a standard in the Material Design components we have chosen to use.

4.2 Interface Design Tools

We have opted to use Figma as our design tool of choice. Figma is a powerful design tool developed for creating user interfaces, logos, and other designs. The tool lets you use and create predefined components that can be utilized throughout the design.

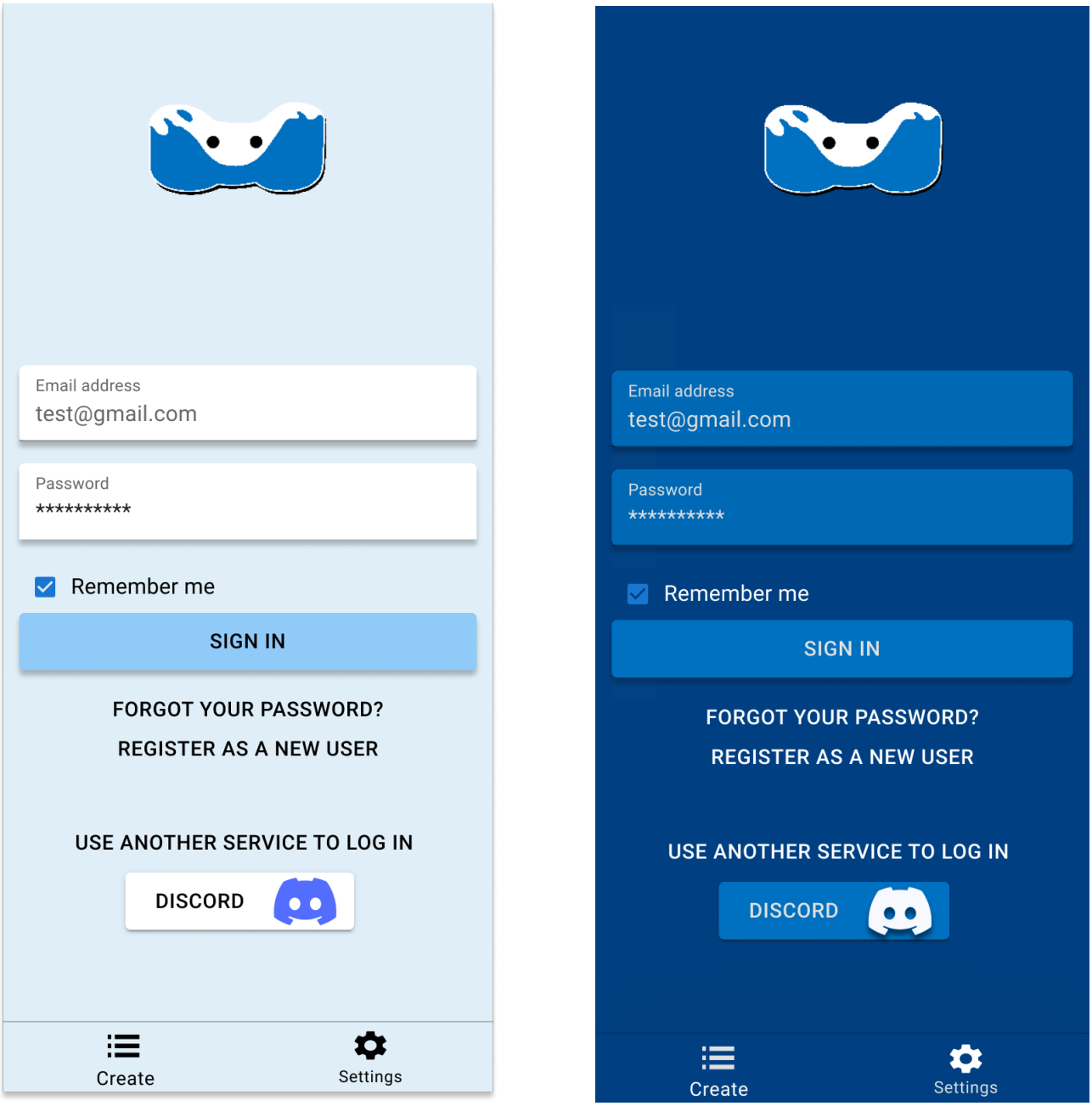
4.3 Application logo

Figure 1: Application logo

An application logo was created to easily identify the application. The logo was created using the Artificial Intelligence logo generator Hotpot.ai [1] and may to some degree share some resemblance to the Discord logo [15]. We have chosen to call it Halpy, which is a small nod to Microsofts older assistant, Clippy.

4.4 Registration and login methods

4.4.1 The universal login screen



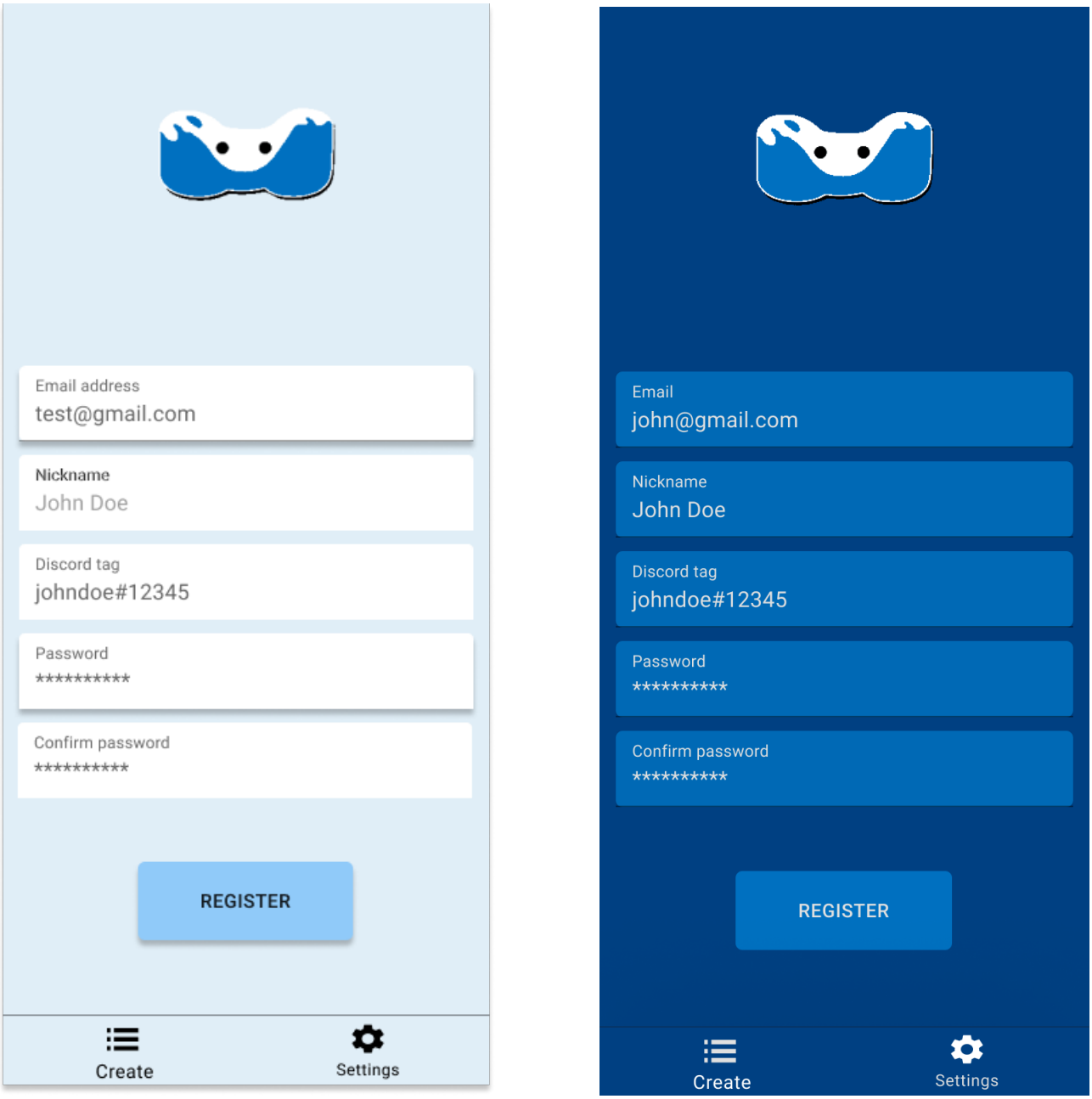
(a) Login screen light mode

(b) Login screen dark mode

Figure 2: Login screen

The main design aspect of the login page is a form where the user can enter their account information, a remember me checkbox, and a button for signing in. This page is meant for users that already have an account in our system. We also designed a button where Discord users can log in with their Discord instead of mail and password. For users that do not have an account in our system or have simply forgotten their password we have designed two text buttons, "FORGOT YOUR PASSWORD?", and "REGISTER AS A NEW USER". These buttons will redirect the users to screens that can assist them with their current problems.

4.4.2 Register new user



(a) Register screen light mode

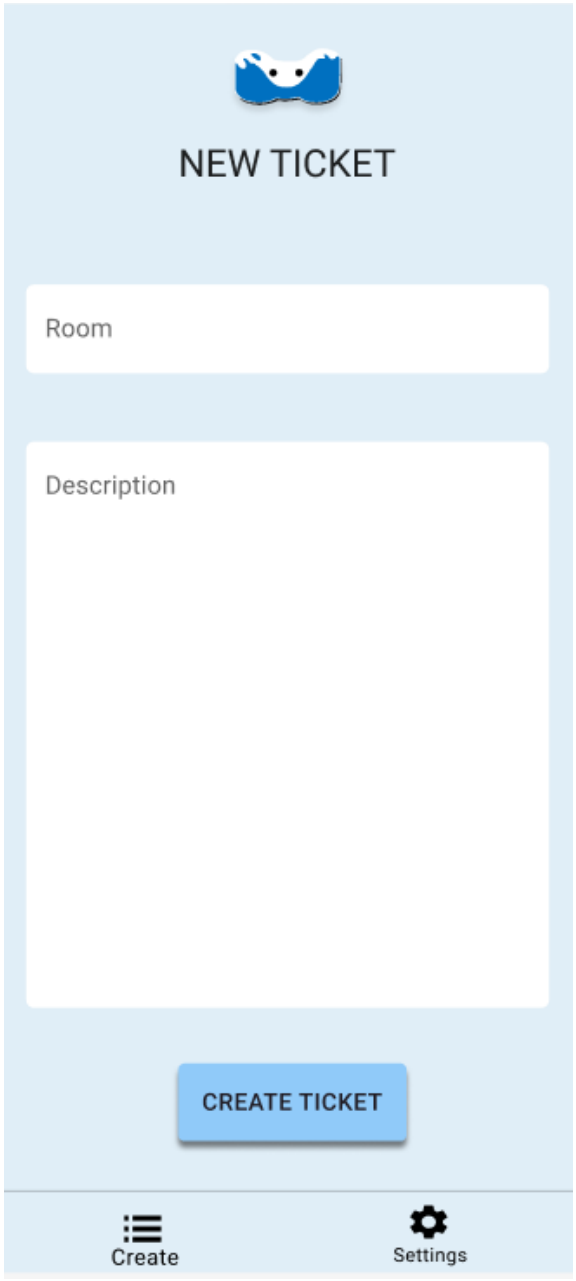
(b) Register screen dark mode

Figure 3: Register screen

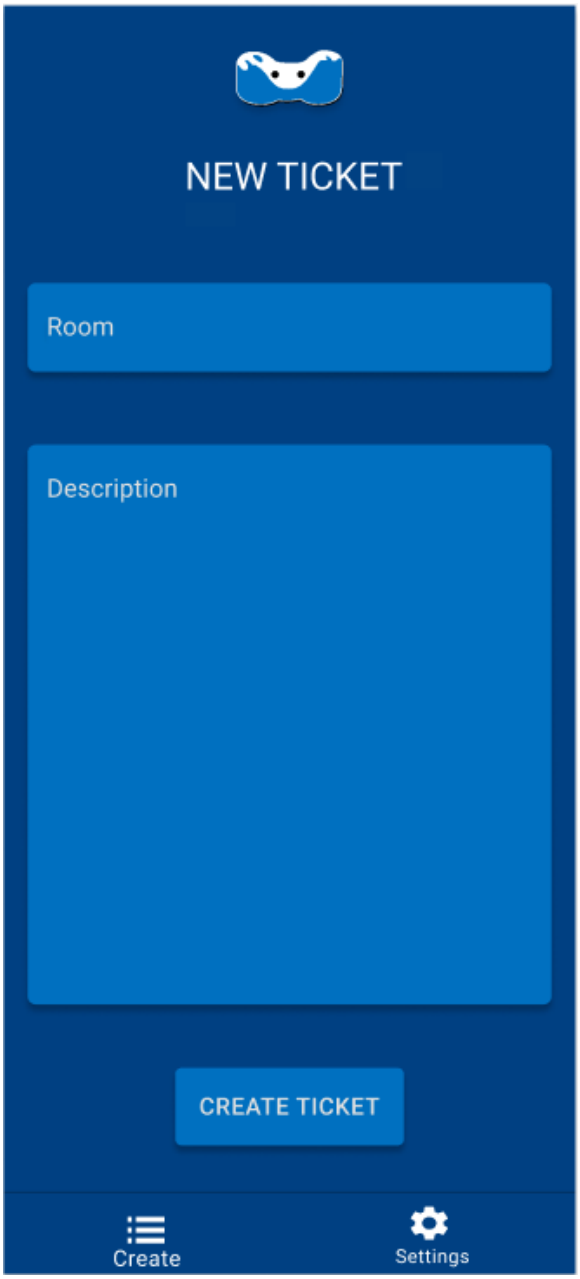
The register screen is designed simply with a screen consisting of a registration form and a register button. For the form, the user must enter their email, a nickname, their discord tag, as well as a password. The discord tag section is planned to be optional. Once a user has filled all the required boxes with the correct information they can press register to get an account in our system.

4.5 Student view

4.5.1 Create new ticket, user logged in



(a) Logged-in user create ticket light mode

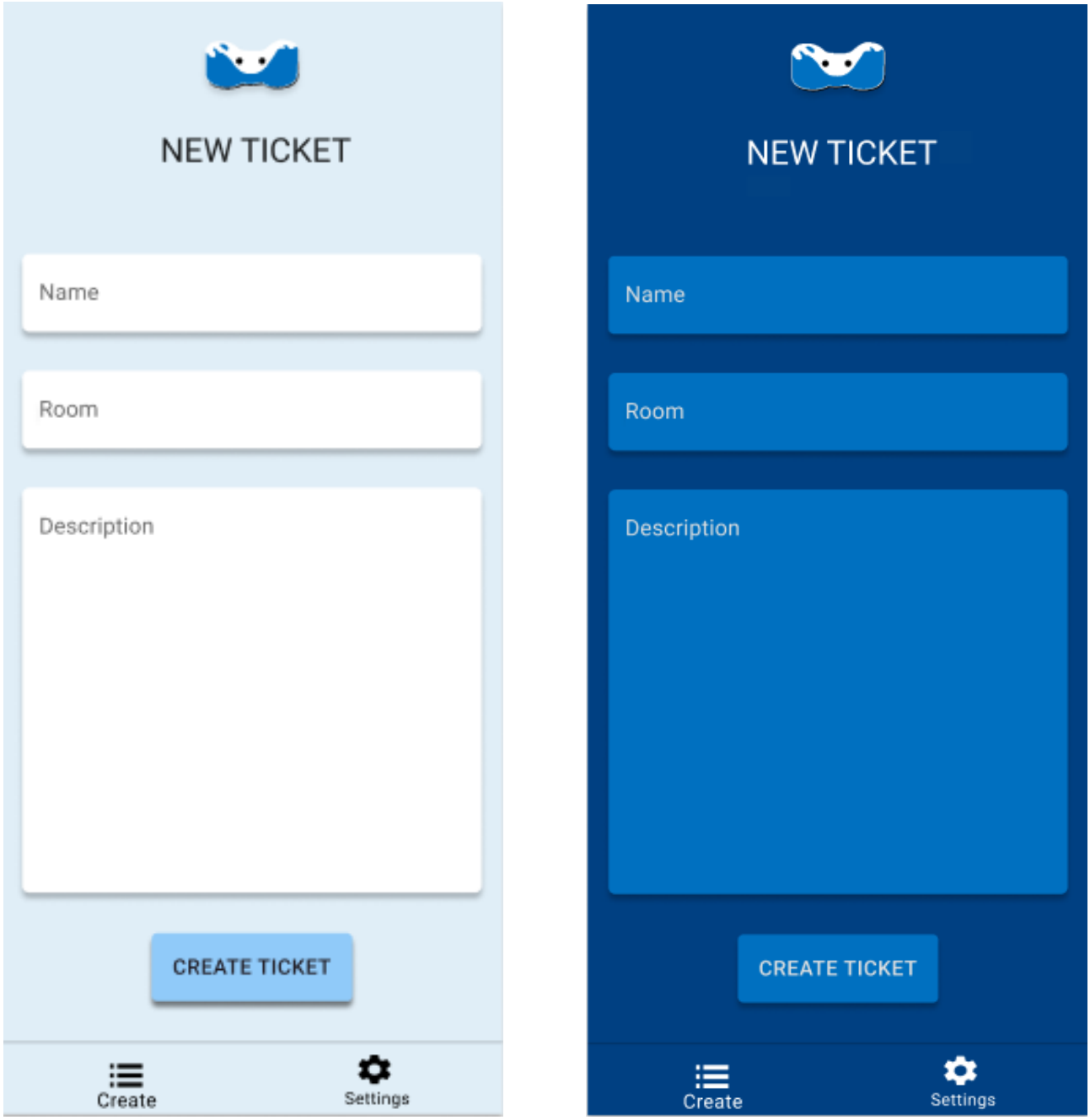


(b) Logged-in user create ticket dark mode

Figure 4: Logged in user create ticket

This is the logged-in create ticket screen and is designed when a logged-in user wants to submit a ticket. The screen is designed with a form consisting of a "Room" box, and a "Description box". For the "Room" box it's going to be a drop-down menu, and in the "Description" box you will enter whatever problem you need assistance with. For logged-in users, there is no need to enter their nickname. After filling out the form the user can press the create ticket button to submit their ticket to the system.

4.5.2 Create a new ticket, anonymous user



(a) Anonymous user create ticket light mode

(b) Anonymous user create ticket dark mode

Figure 5: Anonymous user create ticket

This is the design of the screen for anonymous users. The create a ticket screen for logged-in users and anonymous users are practically identical, with the only difference being that an anonymous user has to enter their name when submitting a ticket. This is done by having one more text box within the form.

4.5.3 The user settings page

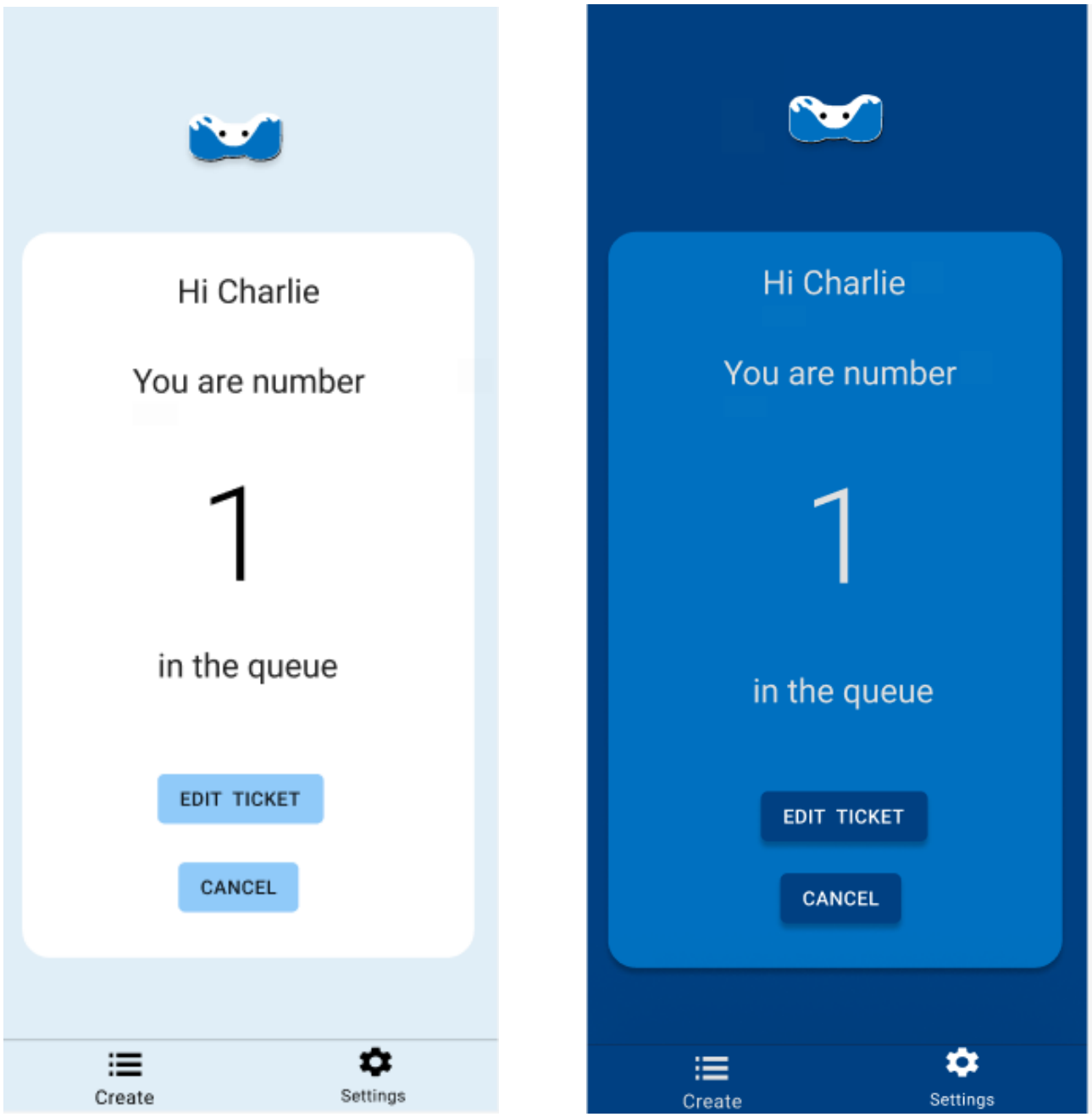
(a) User settings light mode

(b) User settings dark mode

Figure 6: The user settings

This is the design of the settings screen for regular users. This screen will show the user numerous choices with boxes that we also designed with popup boxes, with each leading to their respective screens.

4.5.4 The Ticket queue



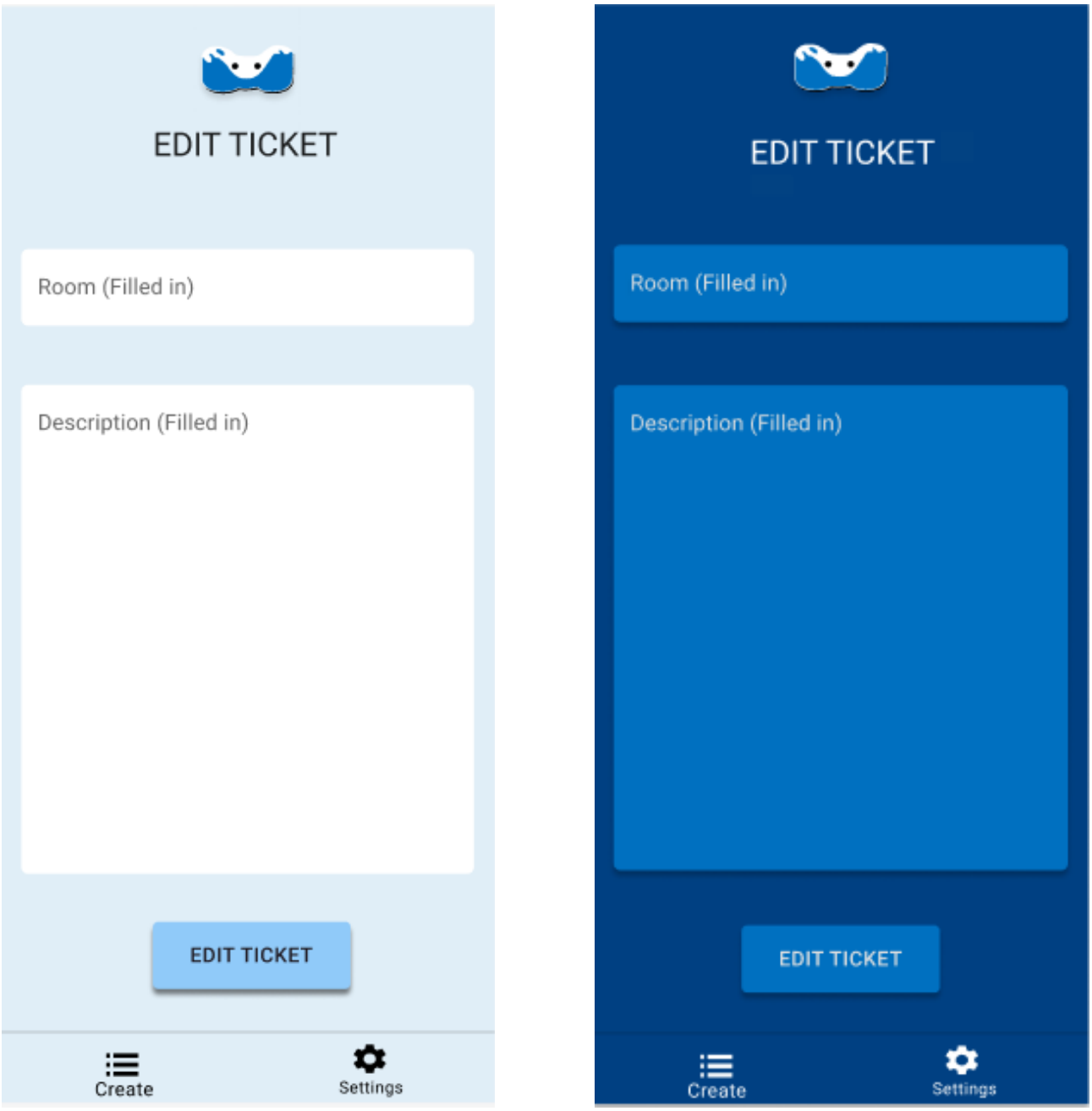
(a) Ticket queue light mode

(b) Ticket queue dark mode

Figure 7: The ticket queue

The ticket queue design is the screen the user will be redirected to after submitting a ticket. The screen is designed to greet the user by their name and tell them their current position in the queue. We also made a cancel button to exit, and a edit ticket button to edit the ticket.

4.5.5 Edit ticket, logged in user



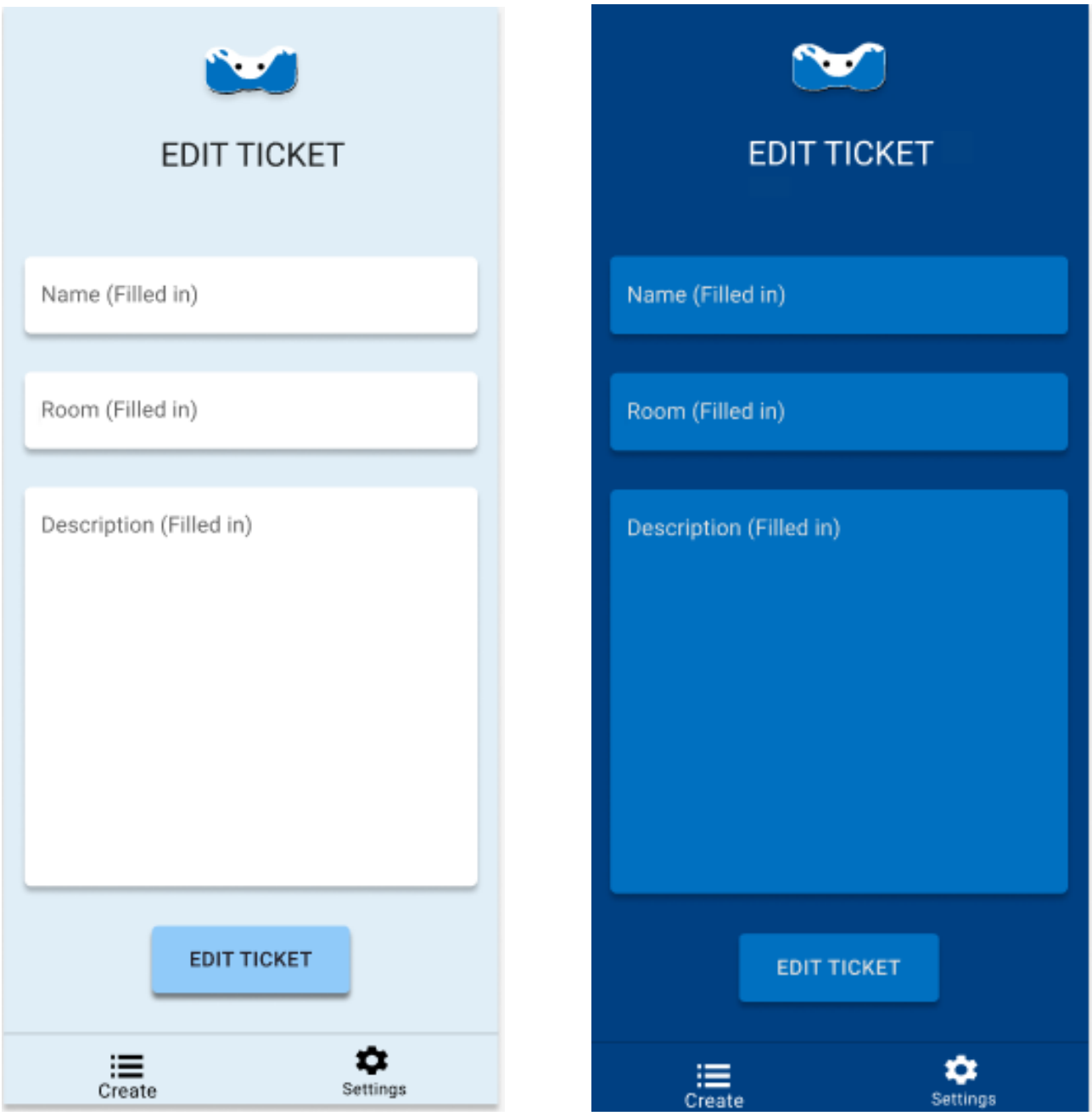
(a) Edit ticket, logged in user light mode

(b) Edit ticket, logged in user dark mode

Figure 8: Edit ticket, logged in user

This is the design of the screen the user will be directed to after pressing the edit ticket button on the queue screen. The edit ticket screen will look almost identical to the create ticket screen but with all the text boxes filled out. This will make it easy for the user to see exactly what they wrote in their original ticket, and which parts they wanna edit. When the user is happy with the changes to their ticket they can press the edit ticket button to return to the ticket queue screen.

4.5.6 Edit ticket, anonymous



(a) Edit ticket, anonymous light mode

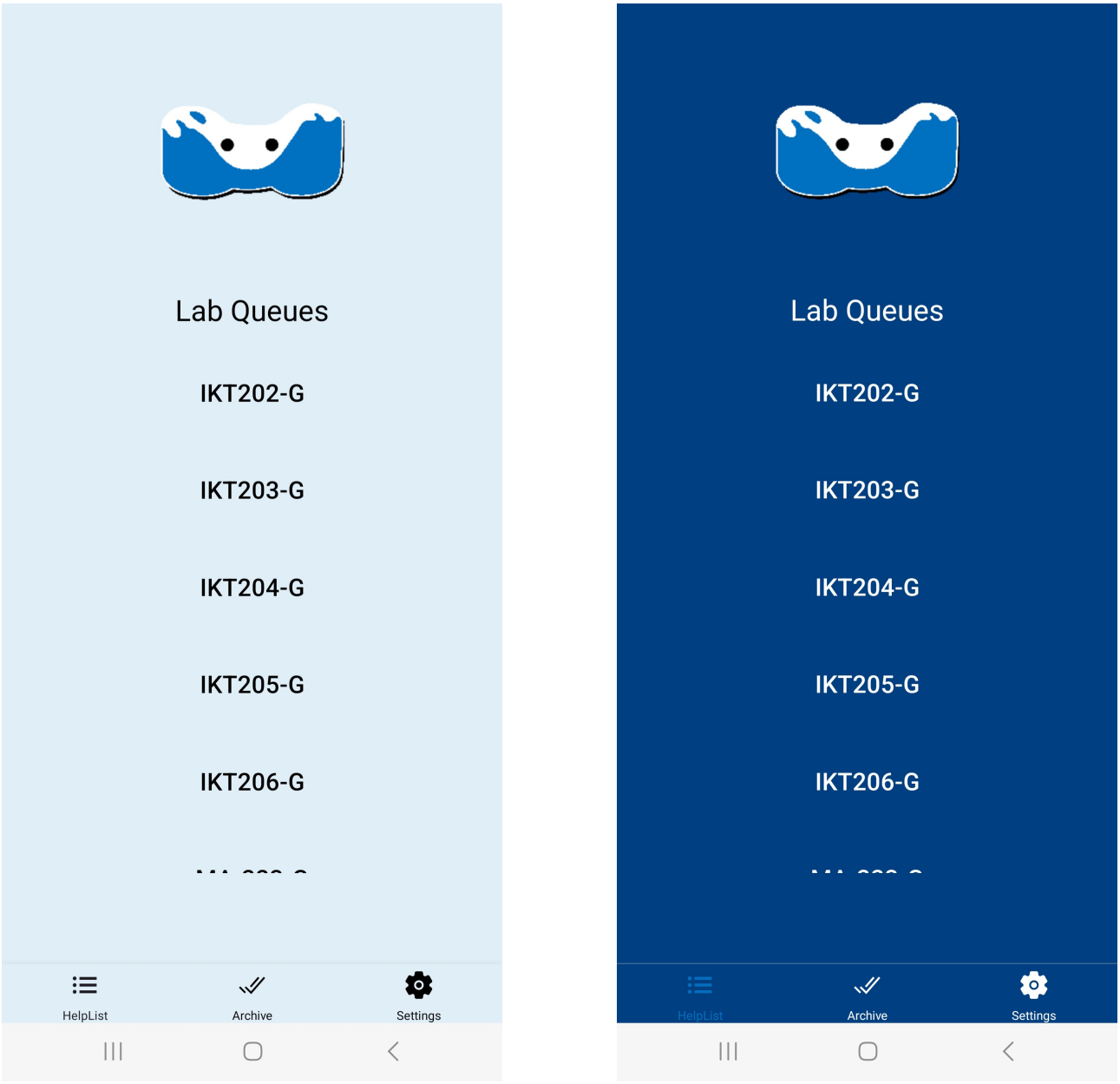
(b) Edit ticket, anonymous user dark mode

Figure 9: Edit ticket, anonymous user

This is the design of the edit ticket screen for anonymous users. The screen will work almost identically as it would for logged-in users with the only difference being the addition of the name text box. Similarly, the text boxes including the new name text box will be filled in with whatever they wrote when creating the ticket. They can then edit the ticket to their liking, and press the edit ticket button to return to the queue.

4.6 The Student Assistants view

4.6.1 The Lab queues



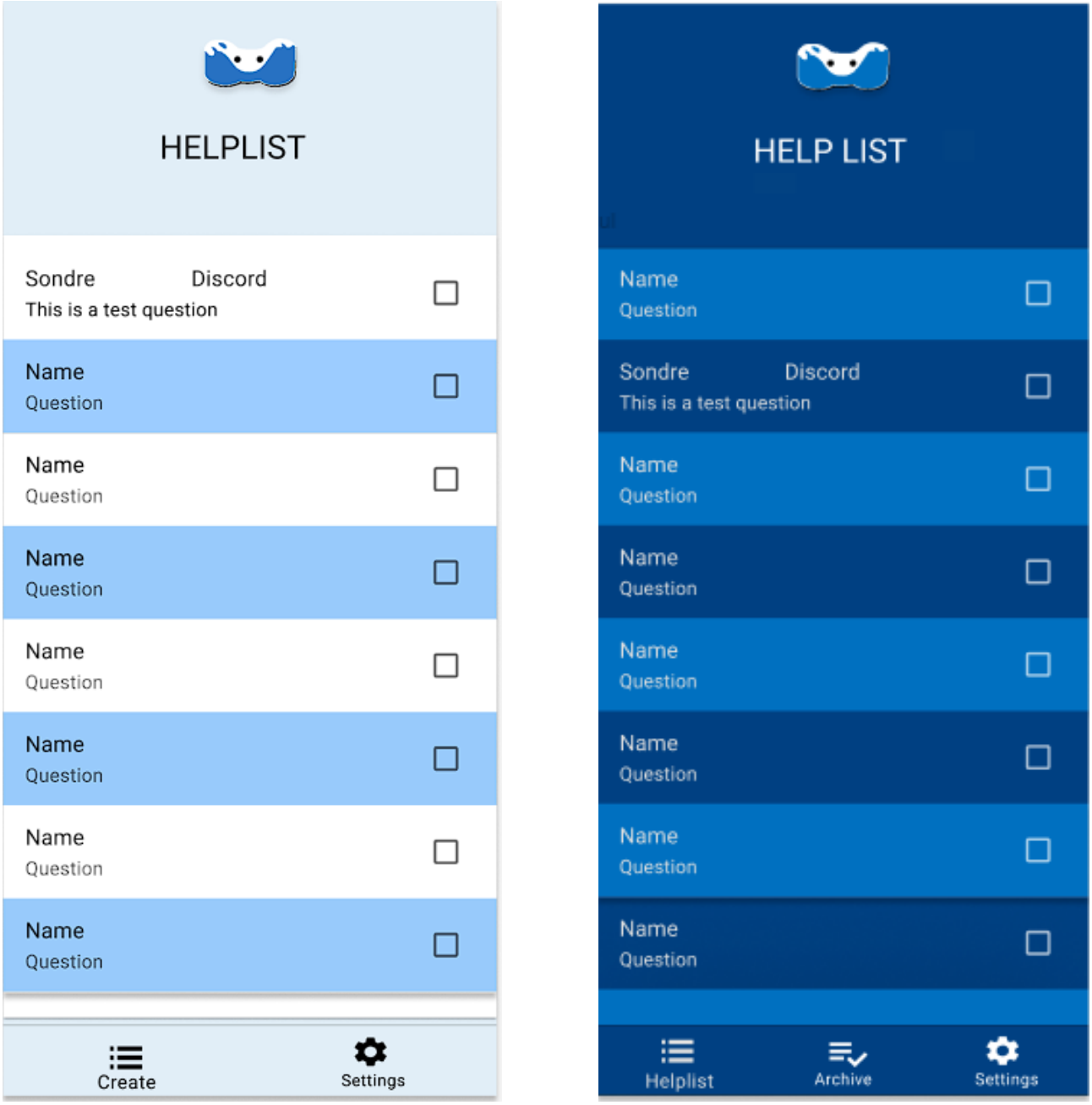
(a) Lab queues Light Mode

(b) Lab queues dark mode

Figure 10: Logged in user create ticket

This is the Lab queues screen that the student assistants will see when they choose the class they are going to be assistants in. The screen consists of a list of buttons with the names of the classes the student assistant can choose from.

4.6.2 The Helplist



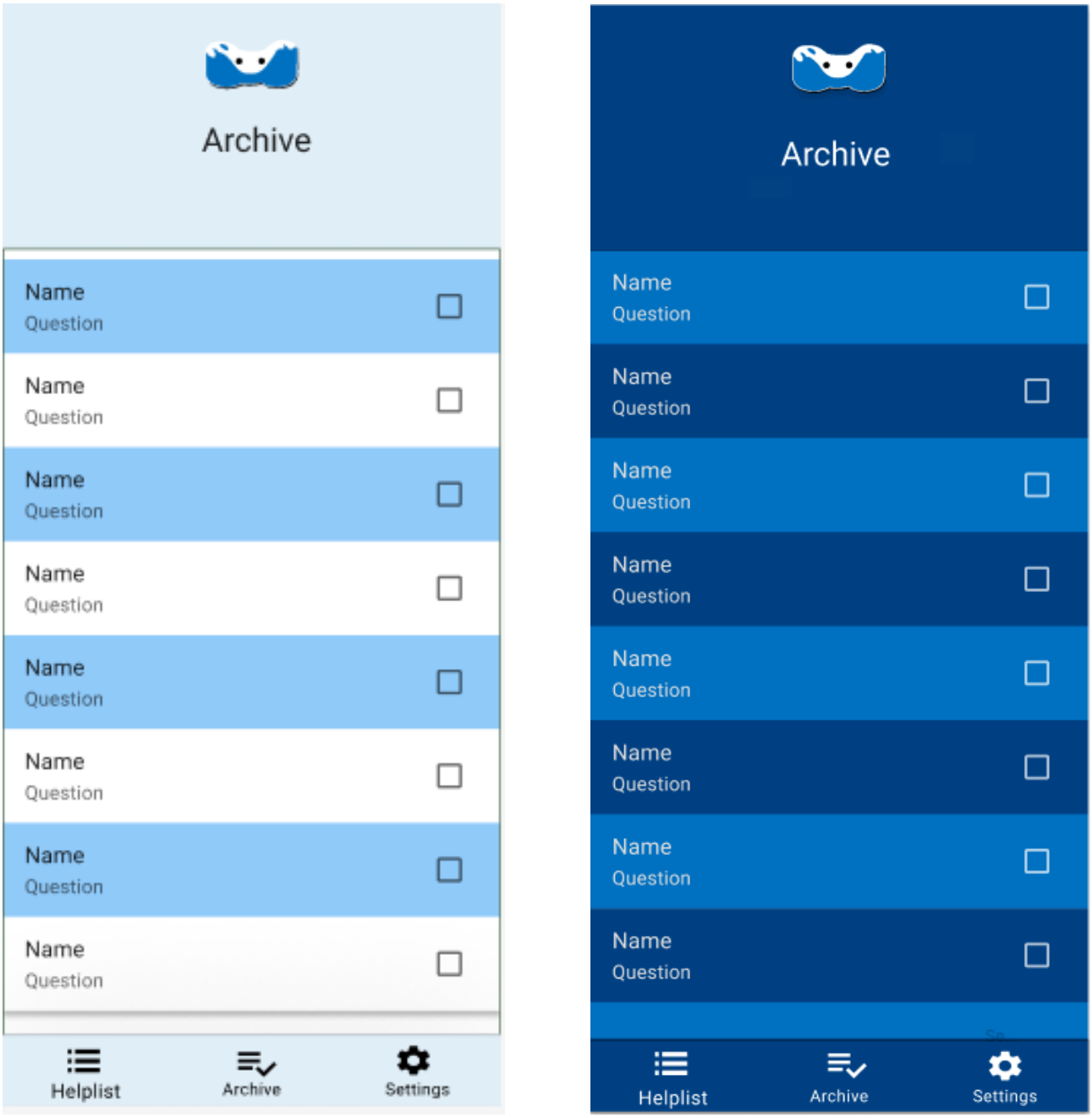
(a) Helplist light mode

(b) Helplist dark mode

Figure 11: Logged in user create ticket

The helplist screen design will be the screen where the student assistants can see all the currently active tickets. These tickets will show as a list in our design. We also figured out that each list item should be designed with an expandable function to read the description if it is longer than one sentence. The list item will show the nickname, discord tag, and description of the ticket, and the helplist will sort of be an overview of the entire ticket queue in each class. We also choose to use two different colors to differentiate the ticket in the list and also made a checkbox next to the ticket to archive the ticket.

4.6.3 The Archive



(a) Archive light mode

(b) Archive dark mode

Figure 12: The Archive

The archive screen is designed almost identically to the helplist screen, but instead of showing currently active tickets, it shows inactive/resolved tickets. This screen will hold all the resolved tickets for each specific lab. The student assistant can see the creator's name and the question they asked.

4.7 Admin view

4.7.1 The admin settings page

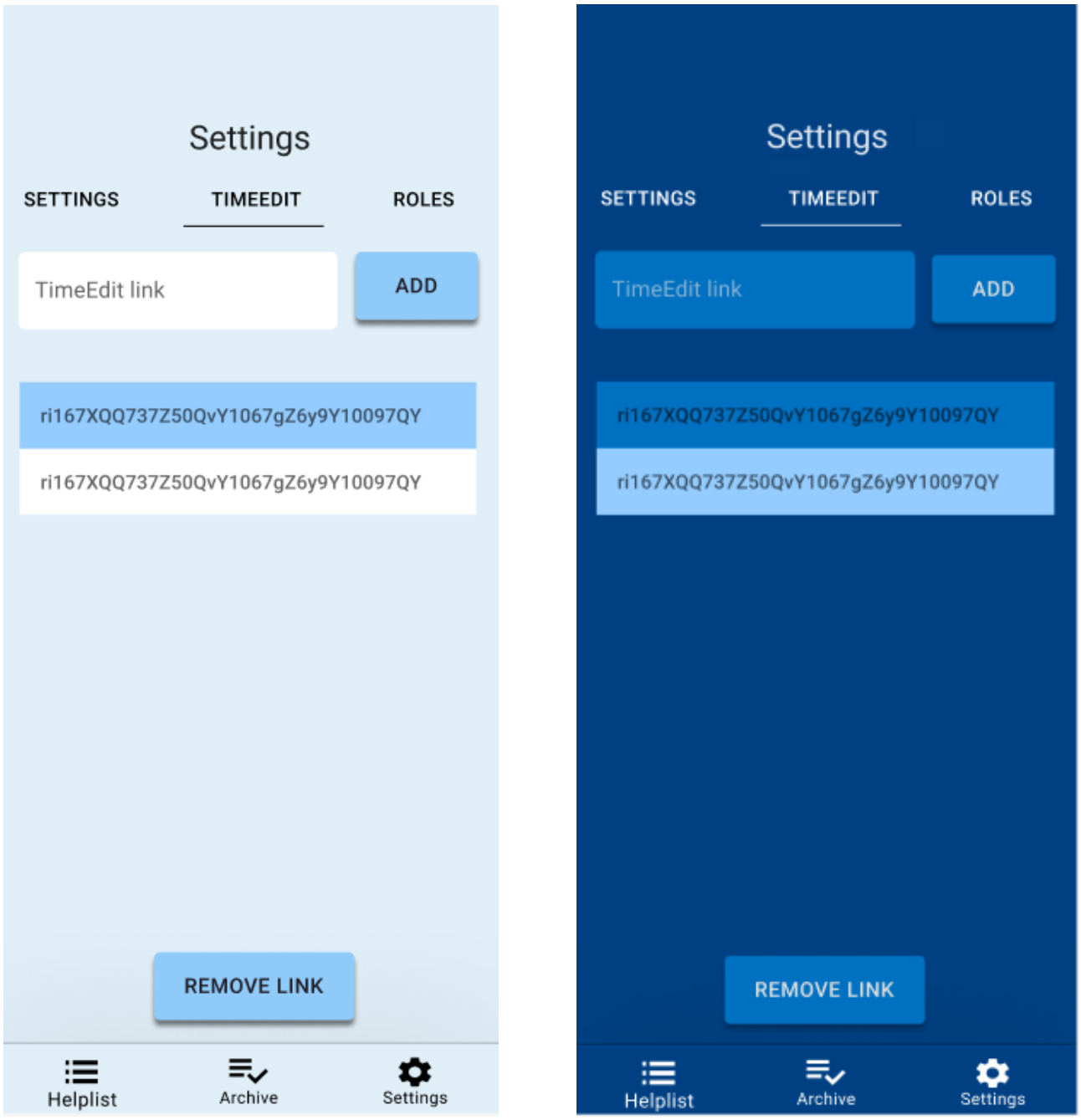
(a) General settings light mode

(b) General settings light mode

Figure 13: General settings

This screen is the general setting for admins. It will likely be the same as the general settings for student assistants. We decided to design this screen with tabs to choose between Timeedit, settings, and Roles to easier browse through the settings. The logic of this screen has not been 100% figured out, therefore the buttons might change, but the styling and layout will remain the same.

4.7.2 Time-Edit links



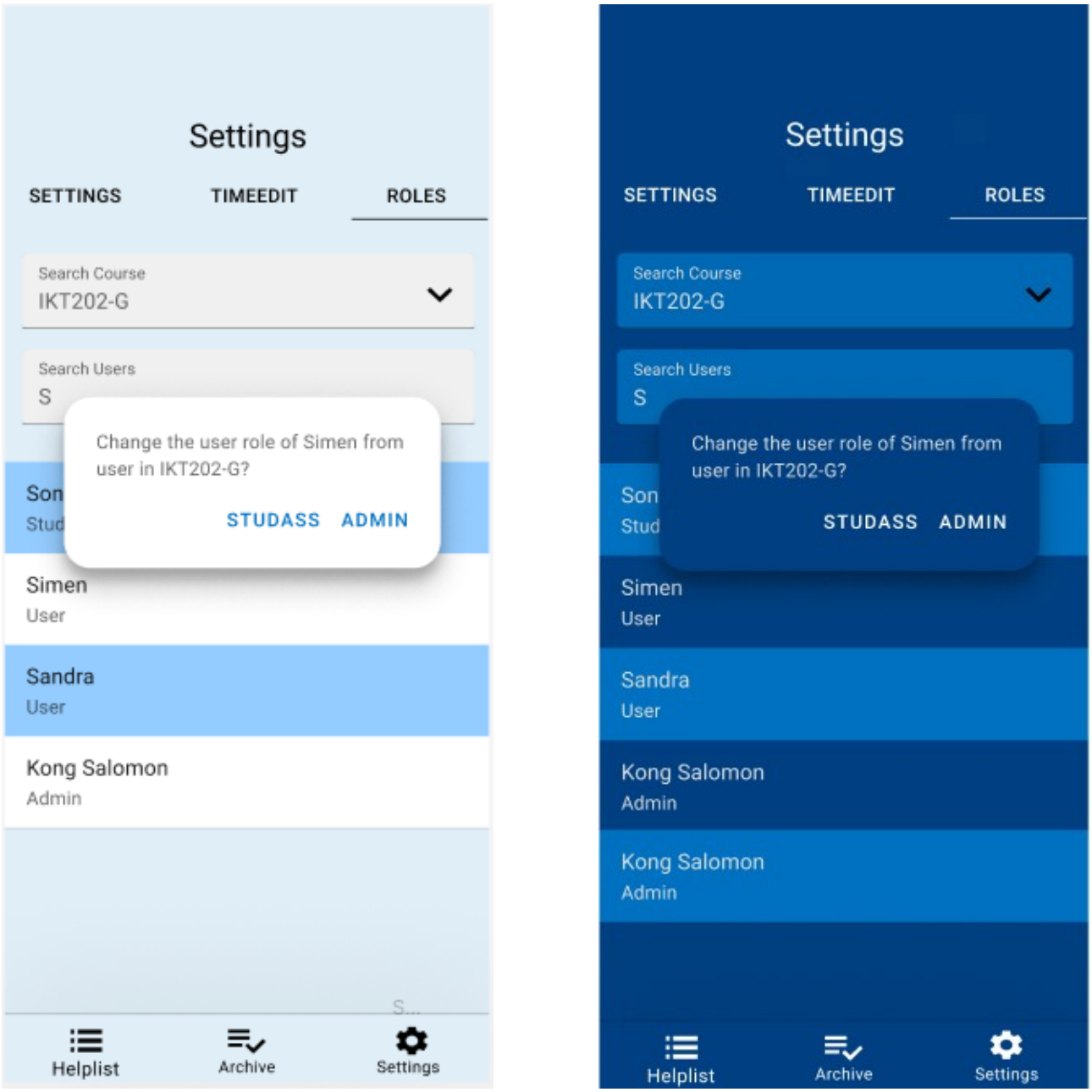
(a) TimeEdit settings light mode

(b) TimeEdit settings light mode

Figure 14: TimeEdit settings

The TimeEdit part of the admin's settings is where an admin can add or remove TimeEdit links from the system. Therefore we made a textbox that takes input and a "ADD" button to add a TimeEdit link. When a TimeEdit link is added it will also be added to a list of links below. We also made a "remove link" button, to remove marked links if needed.

4.7.3 The Roles settings



(a) Roles settings light mode

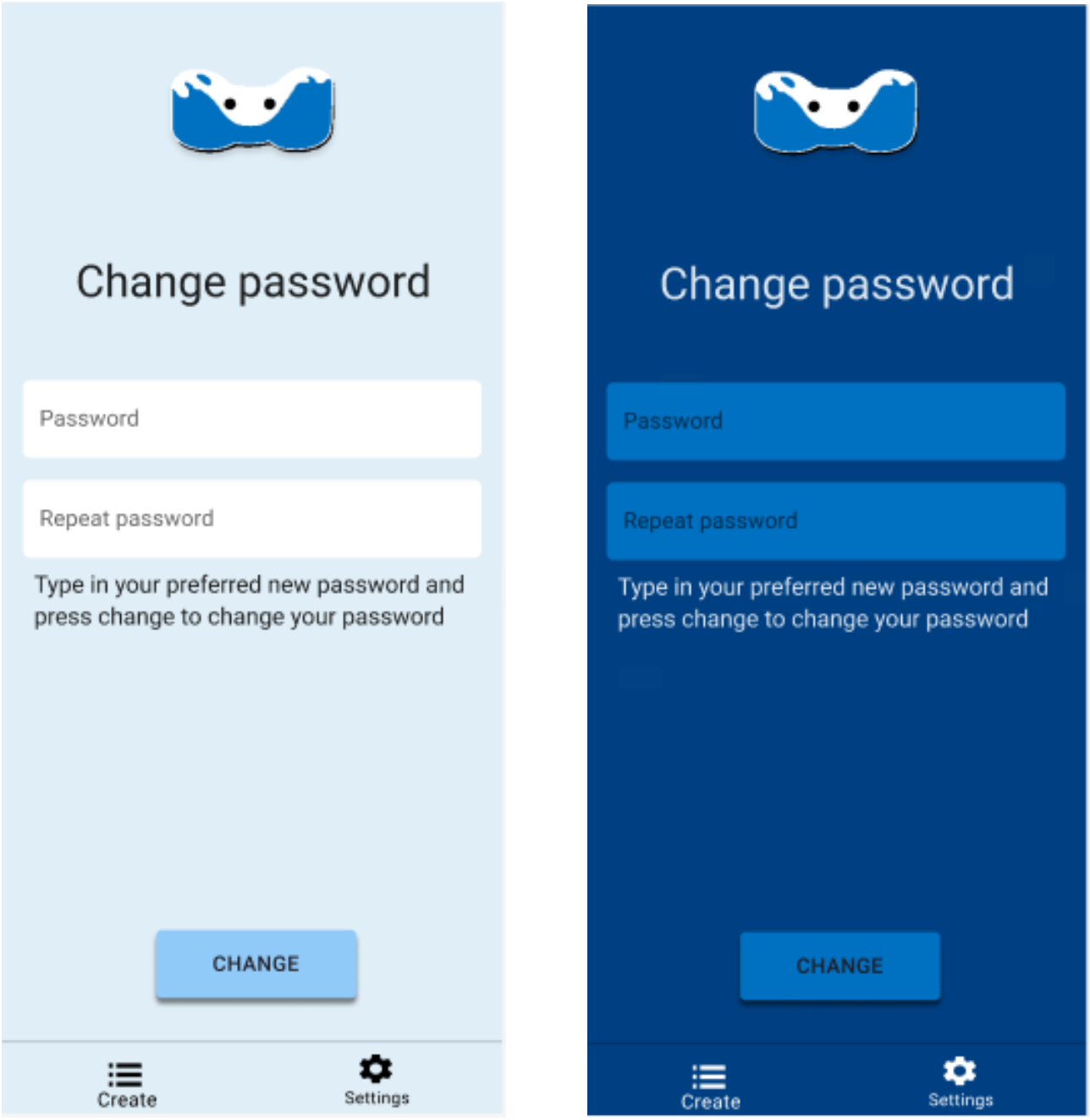
(b) Roles settings dark mode

Figure 15: Roles settings

The roles part of the admins setting page is the screen where the admin can give out the different roles of the system. This screen consists of one drop box where you can select a course and one text box where you can search for a user. You then get a list of users fitting these criteria. If the admin taps any of the users in this list he will get a pop-up that asks for what role he wanna give the selected user.

4.8 Other pages

4.8.1 Change password



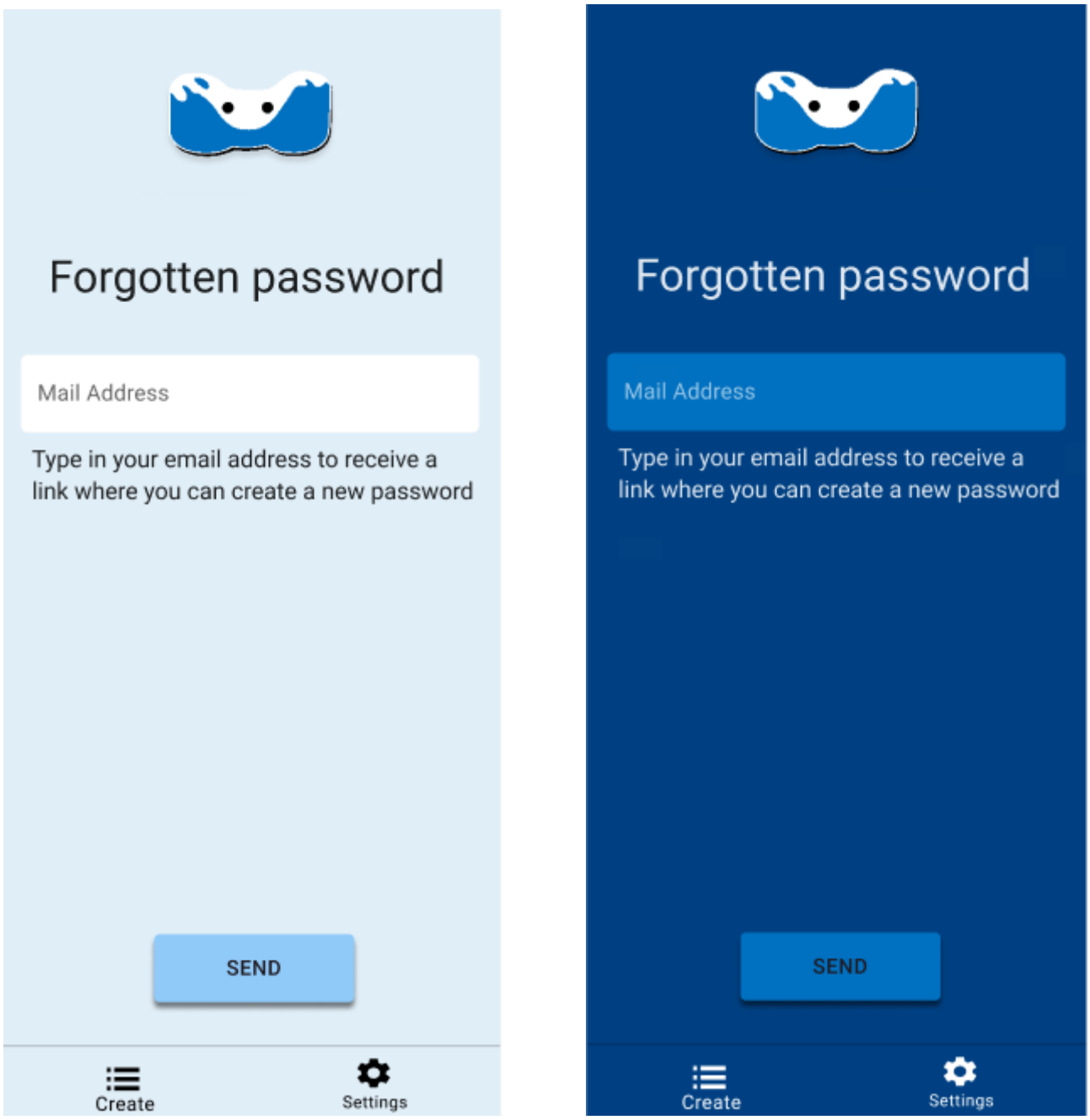
(a) Change password light mode

(b) Change password dark mode

Figure 16: Change password

The change password screen makes it possible for a user to change their password. A user needs to already be logged into their account, and they will find a button redirecting them to the change password screen under their settings. This screen consists of two text boxes where the user can enter their preferred password, as well as a change button. If the two passwords entered match each other and the user presses the change button their password will be updated.

4.8.2 Reset password



(a) Reset password light mode

(b) Reset password dark mode

Figure 17: Reset password

The reset password screen makes it possible for a user who has forgotten their password to reclaim their account. The screen consists of a text box where they can enter their email, as well as instructions on how resetting their password works. If the user enters their email and presses the send button at the bottom of the screen they will receive an email where they can change their password.

4.8.3 Navigation Bar

(a) Navigation bar light mode

(b) Navigation bar dark mode

Figure 18: Navigation bar

This is our design for the navigation bar. The also follows the dark and light mode, and the choices that the user can see, is based on the role and if the user is logged in. We also want to change the color of the symbol of which screen that the user is currently on.

4.9 Application flows

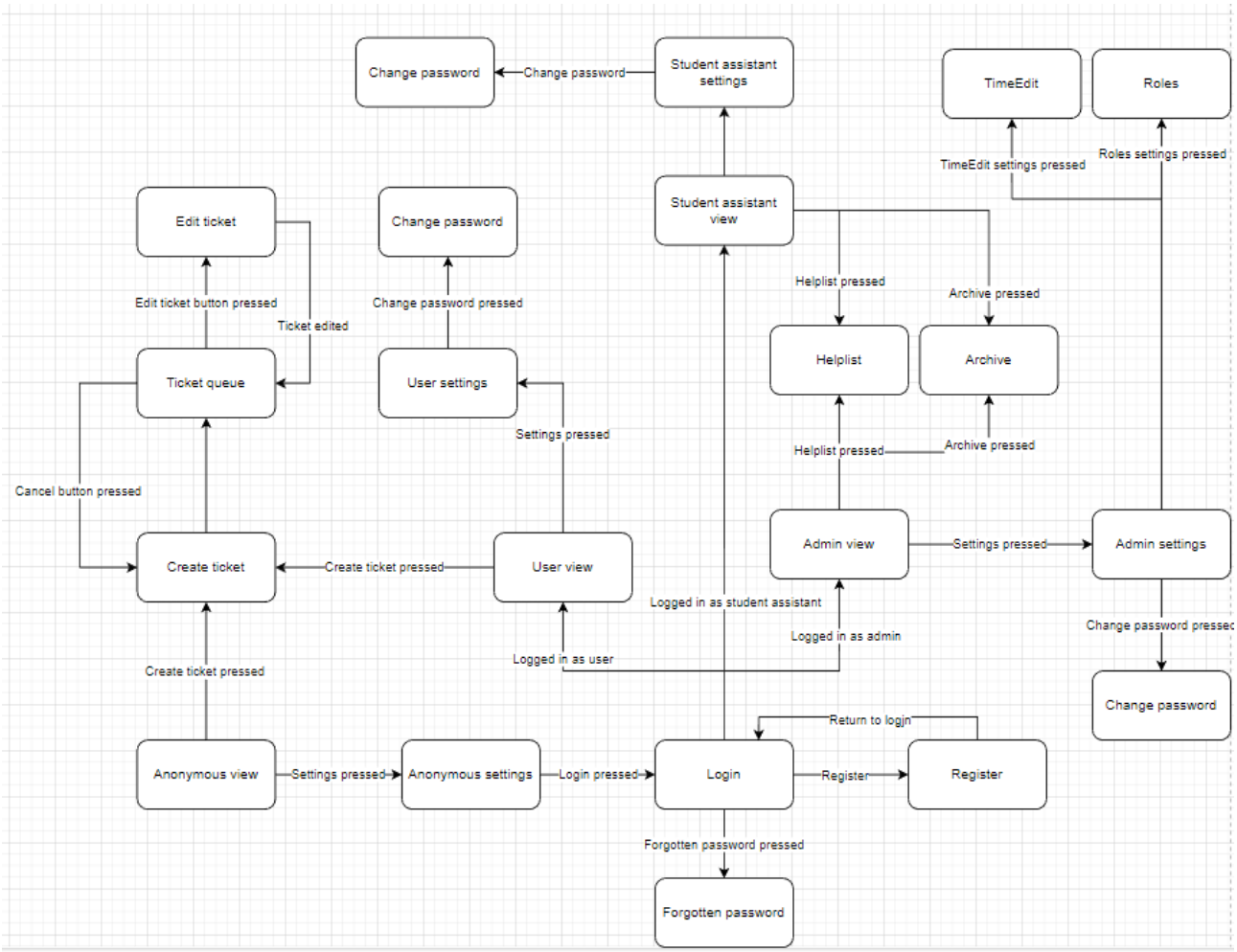


Figure 19: The initial flow design of the system

This figure is the initial application flow of our application. The flowchart is designed before we started the implementation so we had a flow template we could follow. Each box in the flowchart represents a screen that can be accessed in our planned application. The arrows are the paths you can travel in our application to get to the various screens.

5 Technical background

5.1 Technologies

5.1.1 REST API

An API, or Application Programming Interface [22], is a mechanism that allows systems to communicate with each other. A REST API is an API that creates a simple interface for CRUD (Create, Read, Update, Delete) operations. The REST API systems communicate with each other through HTTP and utilize POST, GET, PUT, and DELETE requests respectively that we used extensively throughout our project. This project includes the creation of a REST API to connect the application to the existing backend system. The API receives requests and responds with the appropriate data based on multiple factors. The entire application is entirely dependent on this API, and it is thus an extremely vital part when sending our tickets through the system.

5.1.2 Figma

Figma is a web-based design and prototyping tool used to create user interfaces [17], web pages, and mobile applications. It allows us to collaborate in real-time and work on the same project simultaneously. With Figma, designers we could create vector graphics, import images and assets, and create interactive prototypes that simulate user interaction with the final product. Figma's cloud-based platform made it easy to access and share design files with the other team members. Figma also offers a wide range of plugins and integrations, allowing us to customize the workflow and integrate with other tools. We used Figma to create our design based on the previous HALP design [21] This tool played a large and essential part in planning the initial design of the application and provided a nice and intuitive way of doing this. This meant that we could spend more time on implementation instead of using less dedicated tools, such as Microsoft Powerpoint, to design the application, something we did last year.

5.1.3 Git

Git is a fundamental tool used in software development that allows developers to effectively track and manage changes to their code [7]. The main purpose of Git is to keep a history of changes made within a project. This helps in identifying when and where changes in the project were made, as well as by whom.

We used Git religiously throughout our project which made it easy to track our changes and it gave us a good overview of what had been done, and what needed to get done [7]. The tool has been very extensively used in both the front and the backend and has allowed us to work independently on our own branch before merging the code into a combined branch.

5.1.4 DuckDNS

DuckDNS is a free dynamic DNS service [16] which can be used to connect a server to the internet. The service provides a subdomain of duckdns.org and is run over HTTPS utilizing a signed SSL certificate. DuckDNS works by adding the IP address of the server to it and assigning it a subdomain. This will then let the user connect to the server via the subdomain of the DuckDNS.

DuckDNS connects our application to the backend server by providing a public route for the data to pass through. When setting up an HTTPS connection, it is required to register a domain, which is also provided by DuckDNS. This adds DuckDNS to the list vital components needed to make the application function correctly.

5.1.5 Material Design

Material Design is a design language developed by Google in 2014 [19]. The design language is made to provide a consistent user experience across all platforms and device sizes. Its most fundamental principles are based on the physical world and its textures, including its lighting, surfaces, and movement. The main goal of Material Design is to have a design language that mixes classical design principles with modern technology and science.

We chose to use Material Design because we wanted to design our application in a way that provided the best user experience possible. With this design language, we managed to create an application that is both visually appealing as well as functional and intuitive [19], as the components have already been designed and perfected for us. Most of the applications, except for some buttons and modals use the Material Design components provided.

5.2 Languages

5.2.1 Typescript

Typescript is a JavaScript-based programming language with the ability to type-define functions, objects, variables, and other constructs [42]. The language aids us in making the code more readable and facilitates debugging, testing, and prototyping. The object-oriented nature of the language makes functional programming much more optimal, as all the functionality can be type-defined. Typescript is the language we used to create the HALP application. The language aids in creating the components shown in the application and performs the logic behind them.

5.2.2 C#

C# is a programming language created by Microsoft [9], with deep roots connecting it to C and C++. It is by Microsoft defined as being a modern, object-oriented, and type-safe programming language, and is filled with multiple technologies to create robust and durable applications. C# is the programming language used to create the backend server. It is what builds the web server for the HALP web server, as well as the API with endpoints to which the application connects.

5.3 Frameworks

5.3.1 React Native

React Native is a framework for building mobile applications on both iOS and Android devices, using both React and JavaScript[29]. The advantages of React Native include ease of code generalization and reuse, as well as the efficient rendering of only necessary components in response to data changes [28]. React enables the combination of HTML, CSS, and JavaScript/TypeScript in each component, with the ability for components to handle their own state through hooks provided by the framework. These features provide advantages in both performance and code reuse. React Native is the framework used to create the application. By using the built-in components or by creating our own from said components, we ensure that our application stays compatible with both Android and iOS. We used hooks such as useEffect and useState in order to compact our code and avoid the use of classes. We generalize code by creating and exporting functions to be used in other parts of the system.

5.3.2 .NET

.NET is a software development platform created by Microsoft to enable developers to build multiple types of applications [38]. It supports multiple languages, such as C#, F#, and Visual Basic, and the build system is fully cross-platform. .NET lets developers create applications for different types of systems, such as web, mobile, and desktop applications, all in one package.

It is also completely free, and open source.

The backend system is built upon the .NET platform and is the important building block that enables the existing HALP system to be expanded to include an API as well. The .NET framework also enables the backend server to be both developed and run on both Linux and Windows machines, making collaboration and running on multiple machines a walk in the park.

5.4 Packages

5.4.1 React Navigation

React Navigation is a library used for routing and navigation within a React native application [39]. We used this library to manage transitions between the various screens in our application. It also provided the functionality to share data between these screens. The React Navigation library provided us with our fundamental navigation structure. We also used the library to manage the flow of data between the different screens in our application [39].

5.4.2 React Native Redux

Redux is a library used to make state containers for JS apps. It helps us to write the application so it behaves consistently in different environments [5]. We used this to keep track of all our states that needed to get updated. By using the redux store and reducers we were able to update some of the functions like the helplist, archive, and darkmode functions.

5.4.3 Swagger

Swagger is a REST API documentation tool that makes testing and documenting the API very simple [33]. The tool was added as a package to the ASP.NET project and provided an extensive API test page on the "/swagger" route. Swagger uses JSON to document the API and visualizes it in a nice and well-organized way. The information displayed includes the route, the method, the parameters needed, and the response code and body. Swagger has in this project been used extensively to test and verify the responses of the API endpoints. The tool has been used by all team members to check how to connect the application to the server using the endpoints.

5.4.4 SignalR

SignalR is an open-source communication framework created by Microsoft to allow two-way communication between hosts [30]. It supports multiple technologies but will use WebSockets when available. The framework works by creating a hub on the back-end system and connecting to this hub from the client. SignalR is built to support multiple languages for the client, such as JavaScript, .NET, and Java, and was built with high performance in mind.

SignalR is the technology that drives the live update aspect of the application. When an update arrives on the server, the server will send the relevant updated data to the clients connected using SignalR. This requires the clients to create a persistent WebSocket connection to the server and listen to specific events. Live updates both in the helplist and in the queue are an important aspect of our application, and SignalR is thus a crucial component to make it work optimally.

5.5 Development Operations Platforms

5.5.1 Gitlab

GitLab is a web-based Git repository manager that provides version control for source code management (SCM) [35], Continuous Integration and Deployment (CI/CD) pipelines, issue tracking, wiki creation, and more. It allowed us to collaborate on the project, track changes

to our code, and manage code releases. It supports multiple programming languages and integrates with a variety of third-party tools, making it a popular choice for software development teams. We used GitLab as a way to provide version control for our code. We also used it for automatically testing the new code in a pipeline. We connected GitLab to Jira to track the code according to issues.

5.5.2 Github

Github is similar to Gitlab a hosting platform to store, manage and collaborate on coding projects [36]. The service lets users collaborate and work together using the version control system Git. GitHub lets the user push and pull code from the server, keeping track of versions and edits being made to the project [36]. In our project, we used both Gitlab and Github. Because the Gitlab instance was hosted on the internal network at the University of Agder in Grimstad, connecting to the tool required either being on location connected to their network or utilizing a VPN. To avoid problems with this on the server, we chose to store the system backend on Github instead of the university Gitlab instance. This allowed the server to be easily updated by pushing and pulling to and from Github.

5.5.3 Jira

Jira is a project management tool that is commonly used by software development teams to organize and plan their work [21]. We used Jira's features that included the ability to add user requirements as tasks, create weekly sprints with a list of tasks to be completed, and track time spent working on specific tasks. During weekly meetings, the team reviewed and updated the current sprint using Jira. Jira's task management and time tracking capabilities made it an easy choice for us to improve the workflow and productivity.

5.6 Deployment

5.6.1 Google Play Store

The Google Play Store serves as the official app store for the Android operating system [20]. It allows users to browse and download applications developed for Android phones and published through Google. For mobile application developers, especially those developing on the Android platform, the Google Play Store is a crucial platform for app distribution. The Google Play Store also provides tools for managing app releases, accessing performance metrics, and user behavior through various reports. We used the Google Play Store to upload our application and make it available to the public [20], and the reason for this is simply that it was simpler to compile to Android than to iOS. Compiling to iOS requires a device running MacOS or a CI/CD service capable of compiling React Native to iOS, so we decided the Android route was the way to go.

5.7 Security

5.7.1 TLS

TLS (Transport Layer Security is a standardized protocol used to communicate over encrypted HTTP connections [41]. The connection is secured using a three-way handshake where the client and server exchange encryption keys and protocols. TLS is further developed from the previously widely used security protocol SSL and provides the means to enable encryption, authentication, and integrity. To validate that the connection is secure, the used certificate will have to be verified by a certificate authority (CA), such as Let's Encrypt. TLS is a very fast protocol, and there is virtually no performance issue compared to inducing HTTP connections.

All connections between the application and server are secured with TLS, which makes the data transfer significantly harder to decrypt if intercepted. This is an important security aspect of the application and makes sure that no data ends up astray. This applies both to connections being made with normal HTTP requests and responses, as well as WebSocket SignalR connections.

5.7.2 Let's Encrypt

Let's Encrypt is a certificate authority (CA), and can provide TLS certificate authorization for HTTPS [18]. This allows a client and server to encrypt a connection and communicate securely over it. The service is provided by the Internet Security Research Group (ISRG), which exists to reduce barriers to enable easier secure communication over the internet [24].

Let's Encrypt is the certificate authority responsible for signing the TLS certificate to allow for encrypted communication between the server and clients. The service was chosen because it is free, and well known in the open source community [18].

5.7.3 Certbot

Certbot is an open-source software tool that uses the Let's Encrypt certificates to let manually-administrated websites enable HTTPS [4]. It is part of the Let's Encrypt project which aims to make the process of getting SSL/TLS certificates as simple as possible. We used Certbot in our project to secure web communication. By using Certbot to enable HTTPS we get a better way to secure sensitive data, and authenticate servers to clients [4]. Certbot was used to create the TLS certificate for use with encryption. The tool was used in collaboration with Let's Encrypt to generate and sign the certificate used on the server, to allow for secure HTTP (HTTPS) connections to be established between the server and clients.

5.7.4 JSON Web Tokens

Authorization is provided using JSON Web Tokens (JWT) [8]. Once a user enters valid credentials, the server generates a digitally signed JWT token, which is returned to the user. This token will then have to be passed to the server in the "Authorization: Bearer" header when requesting communication with endpoints that require authorization. JWT tokens can use a public/private key pair with the X.509 certificate for signing, which is more secure than using the shared secret with SWT (Simple Web Token) tokens.

JSON Web Tokens are used to authenticate the user sending requests to the server. This enables the server to make sure that the user is logged in and has the correct authorizations. If the user tries to access functionality that they do not have access to using this token, the server will return a 401 Unauthorized.

5.8 Applications

5.8.1 TeamViewer

TeamViewer is a software solution that allows the user to control devices remotely [34]. The service allows the remote connection to be opened without the use of any VPN service or the opening of ports in the router. TeamViewer gives access to the whole screen of the remote host and is thus widely used as an IT support platform. In 2021 TeamViewer was given the Best User Experience and Employee Experience at European Customer Centricity Award 202

TeamViewer has been extensively used to update and keep track of what has been happening on the server. The service has allowed us to both control the console window of the running application, as well as access other tools such as DB Browser for SQLite to troubleshoot the database. This would not have been as simple had we opted for a solution such as plain SSH.

5.8.2 DB Browser for SQLite

DB Browser for SQLite, also known as DB4S, is an open-source tool to create, design and edit databases that run SQLite [13]. The browser provides a graphical user interface and makes it possible to edit the database both using visual components, as well as executing SQL queries.

The tool has been used on the server side to keep track of, edit, and visualize the database when needed. This has enabled us to troubleshoot and make changes in the testing environment to get ready for publishing to production. After spinning up the backend server application, the DB Browser has played an important part in verifying that the system works correctly.

5.9 Communication Technologies

5.9.1 WebSockets

Websockets is a technology that can be used to create a two-way connection between a client and a server [3]. This lets the client and server communicate freely without the client having to poll the server in any way to get information immediately. This makes the technology superior to polling-based technologies, as well as server-sent events. The WebSocket protocol is designed to use HTTP over port 80, and HTTPS over port 443, although it is not limited to these application protocols. Just like with TCP, the connection is established using a handshake, and data transferring is possible immediately after the connection is established.

5.9.2 Server-sent events

Server-sent events is a technology that allows an application to listen for messages sent from the server [11]. SSE works by adding specific headers to the connection after it is established, to keep it open so that the application can receive the messages from the server. SSE is a one-way service, in the sense that only the server can send data after it has been established. In contrast to long polling, SSE connections are kept alive after the client has received data from the server.

5.9.3 Long Polling

Long polling is a slightly more advanced form of polling and can be classified as something in the middle of polling and server-sent events [25]. Long polling consists of sending the server a request, adding the correct headers to keep the connection open, and then closing the connection after the server has sent a response. This response will not be sent immediately, but rather when the server is ready. After the connection has been closed, the client will immediately re-open it, and wait for new updates.

5.9.4 Polling

Polling is the easiest way to get updates from the server and is the process of asking the server if it has any new updates [25]. There are two downsides to this approach, however: It is both resource intensive and slow. The server will receive a lot of potentially unnecessary requests it will have to process, and the client will have to send unnecessary requests, and handle the response. The information updates will also not be instant on the client side, as the more often the client checks for updates, the more flooded the server will be with requests. This leaves the developer with having to find a "sweet spot" between server request handling capacity and application "live update" speed.

6 Product

In this section, we will explain how our product will work and look for a person using it. We will show the different screens of our application, and explain how they are used and connected together. Furthermore, we will explain how the application differentiates for users, student assistants, and admins.

6.1 User interface design

When creating the user interface design, we followed the design we created as much as possible but also made some tweaks where it needed to be different to make it more intuitive. We wanted to make the design resemble that of our HALP website, while still keeping our main focus on making the design as user-friendly as possible. The design we ended up implementing is shown in the chapters below.

6.2 Application logo

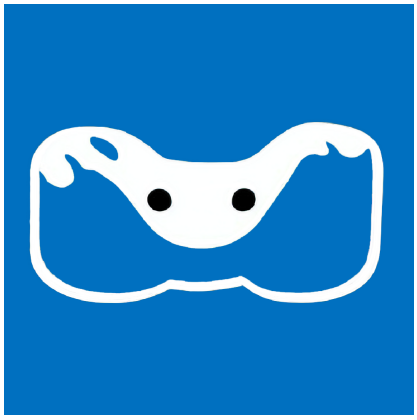
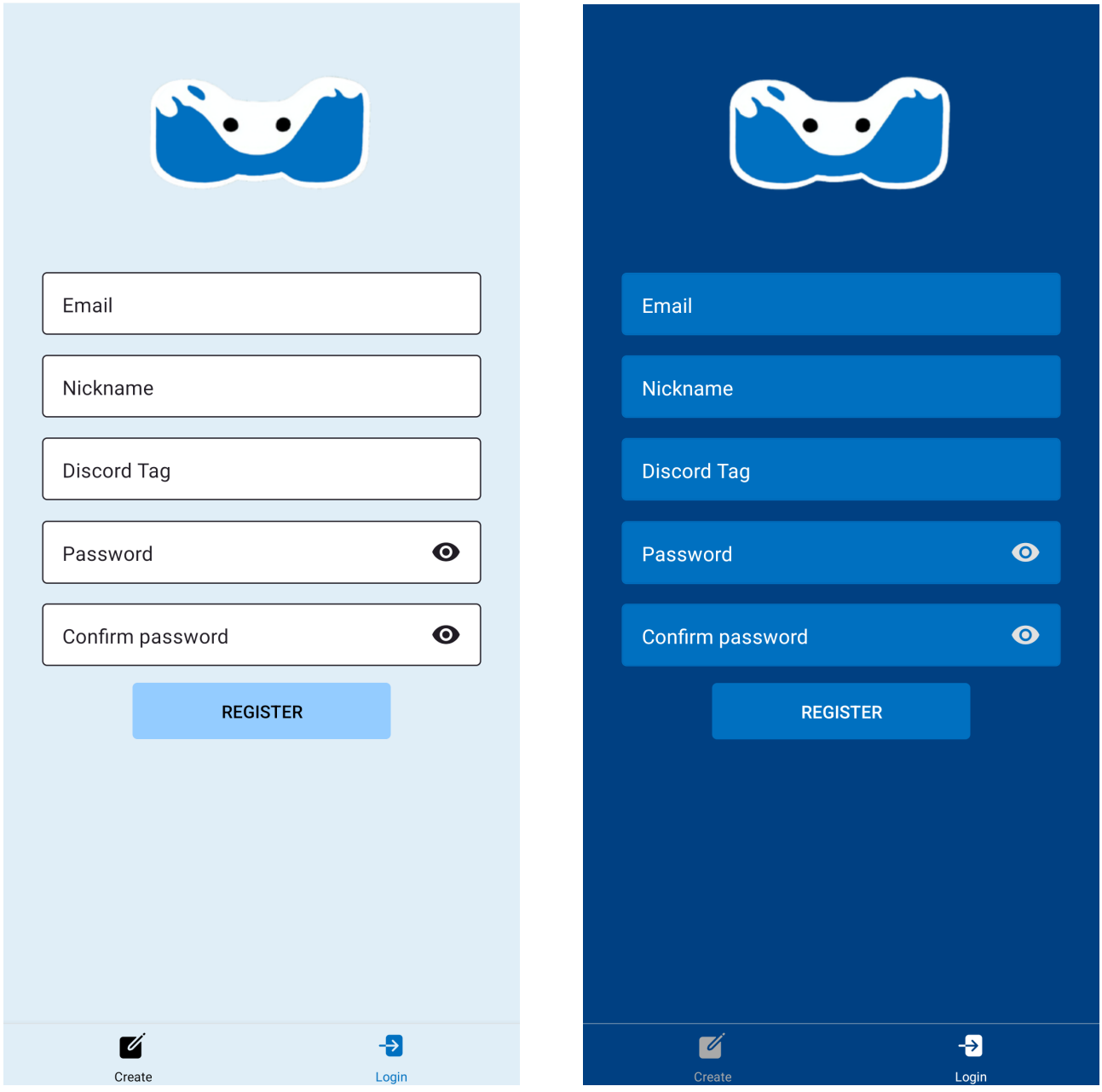


Figure 20: Final application logo design

The figure above shows the final design of the logo for our application. This logo is a cleaned-up version of our original logo design. The main difference between the two logos is the removal of the apparent black outline surrounding the old logo. This made our final design look more clean and professional. Furthermore, we put the logo on a blue background to better fit the size of a standard mobile application logo. This logo is used for our application logo, and the figure within the logo is featured at the top of every screen in the application.

6.3 Registration and login methods

6.3.1 Registration screen



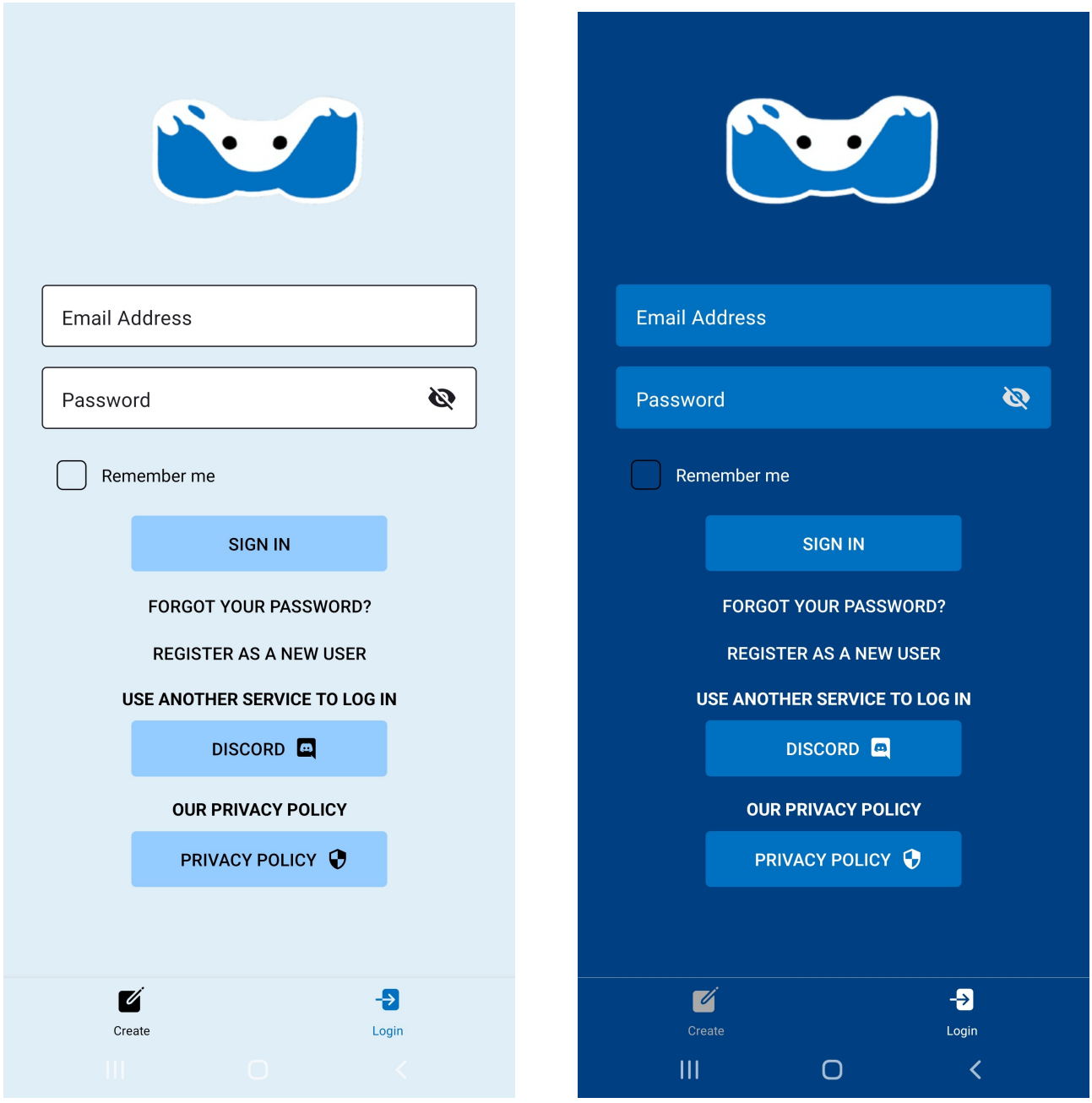
(a) Register light mode

(b) Register dark mode

Figure 21: Finished product screen for registering

This is the screen that lets a user register an account for our application. The register screen consists of a logo, five text-input boxes, and a register button. This is essentially a form where the user can enter their email, nickname, discord tag, and password twice to create their account. After registering their account they will receive an email where they will receive a link to confirm their email.

6.3.2 Login screen



(a) Login light mode

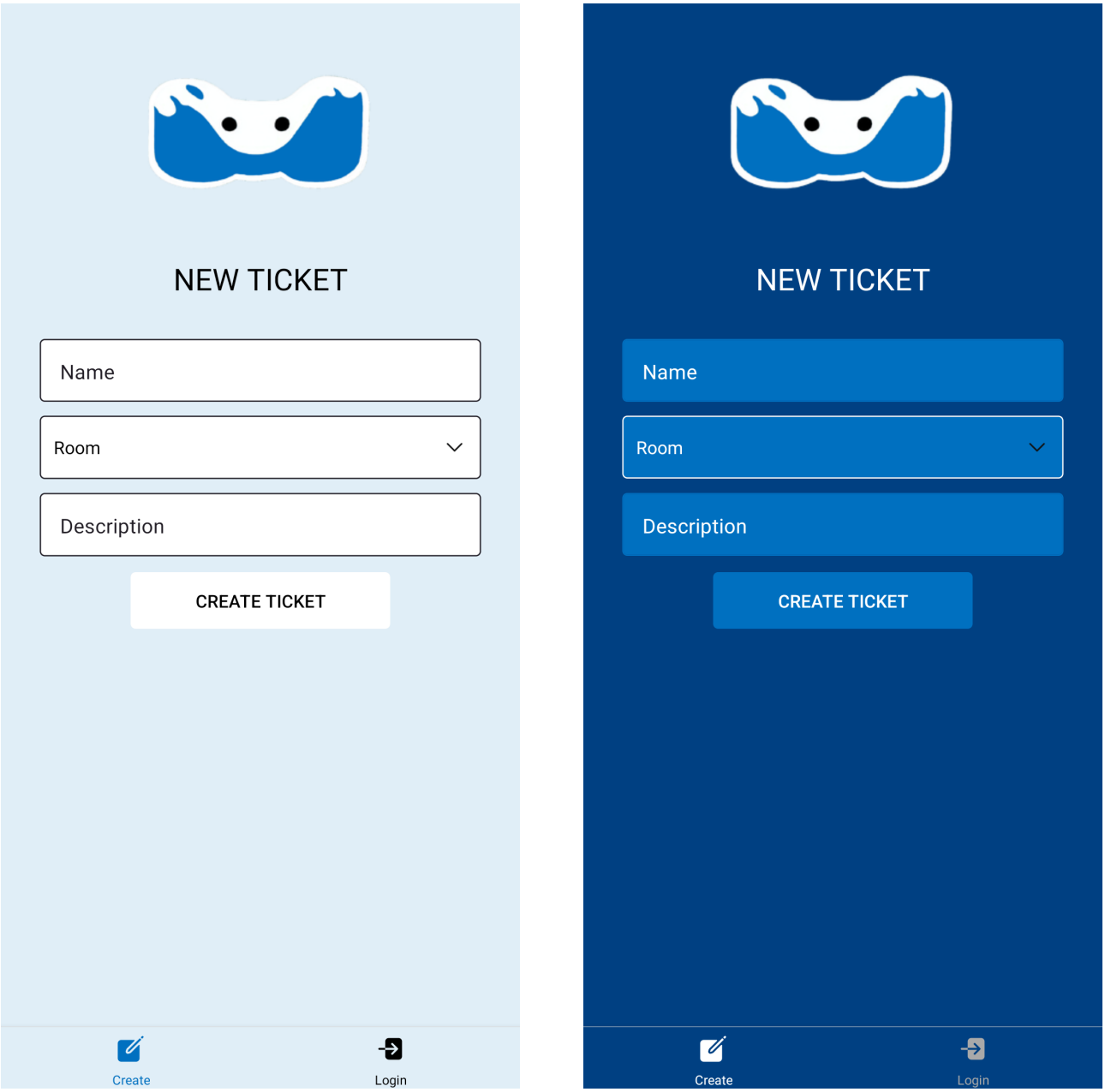
(b) Login dark mode

Figure 22: Finished product screen for logging in

The login screen is the screen where a user, student assistant, or admin can log into our application. This screen consists of the applications logo, a login form, a remember me checkbox, buttons for redirecting to the register screen and forgotten password screen, as well as a button for logging in with Discord. Once a user has logged in to our application they will be redirected to the create ticket screen. The first time a user tries to log in with Discord they will be prompted with a screen to authenticate their account. This will only happen the first time a user logs in with Discord.

6.4 Student view

6.4.1 Create a new ticket, anonymous user



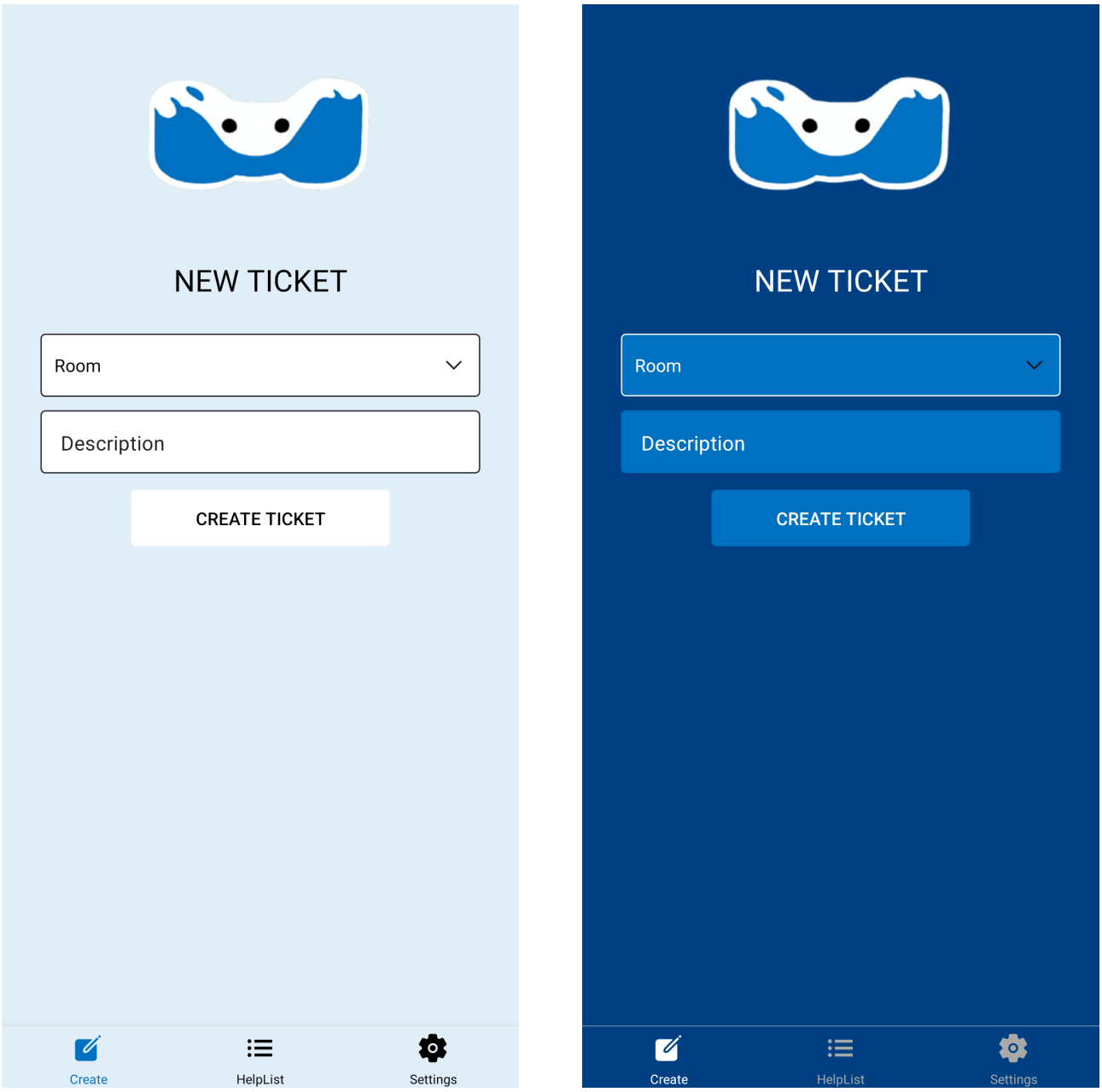
(a) Anonymous create ticket light mode

(b) Anonymous create ticket dark mode

Figure 23: Finished product screen for anonymous users creating tickets

The create ticket screen for anonymous users is the first screen a user encounters when they open our application. This screen consists of the application's logo as well as a form for submitting a ticket. Here the user can enter their nickname, choose the room they're in from a drop-down menu, and write a description of their problem. Then they can submit their ticket to the helplist, so they can get in the queue and eventually get assistance from a student assistant. This screen got changed a little from the original Figma design. The main change was making the description a text-input box that expands as you put more text in it. This was done because we did not want to take up unnecessary space on the screen, when creating for a mobile application.

6.4.2 Create a new ticket, user logged in



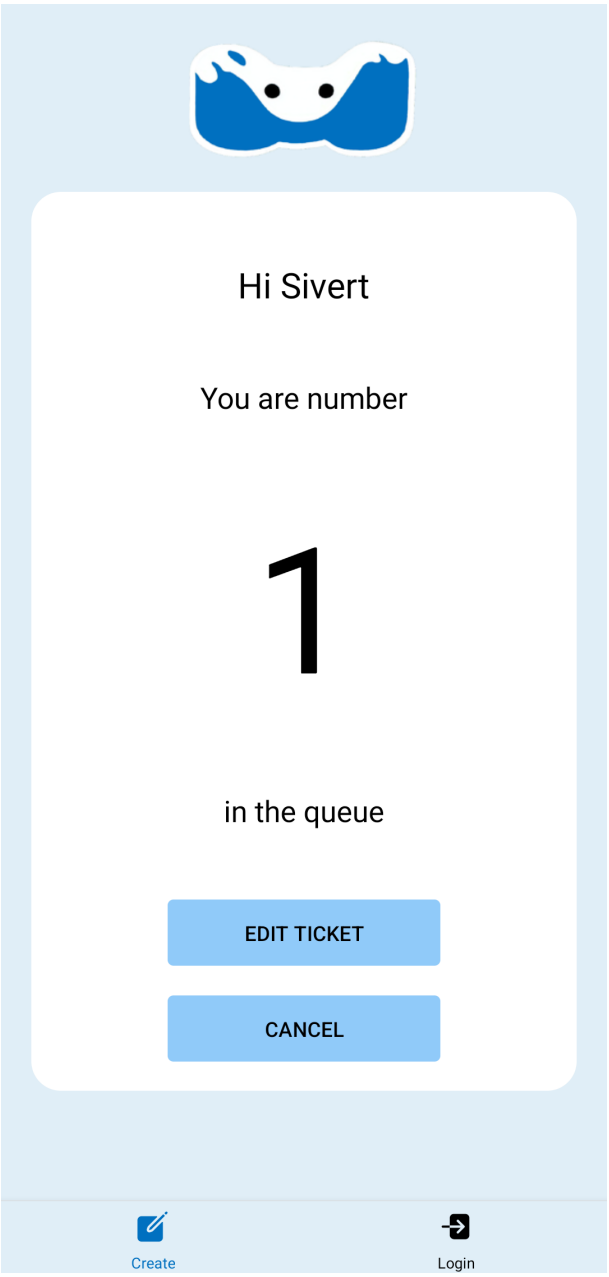
(a) Logged in create ticket light mode

(b) Logged in create ticket dark mode

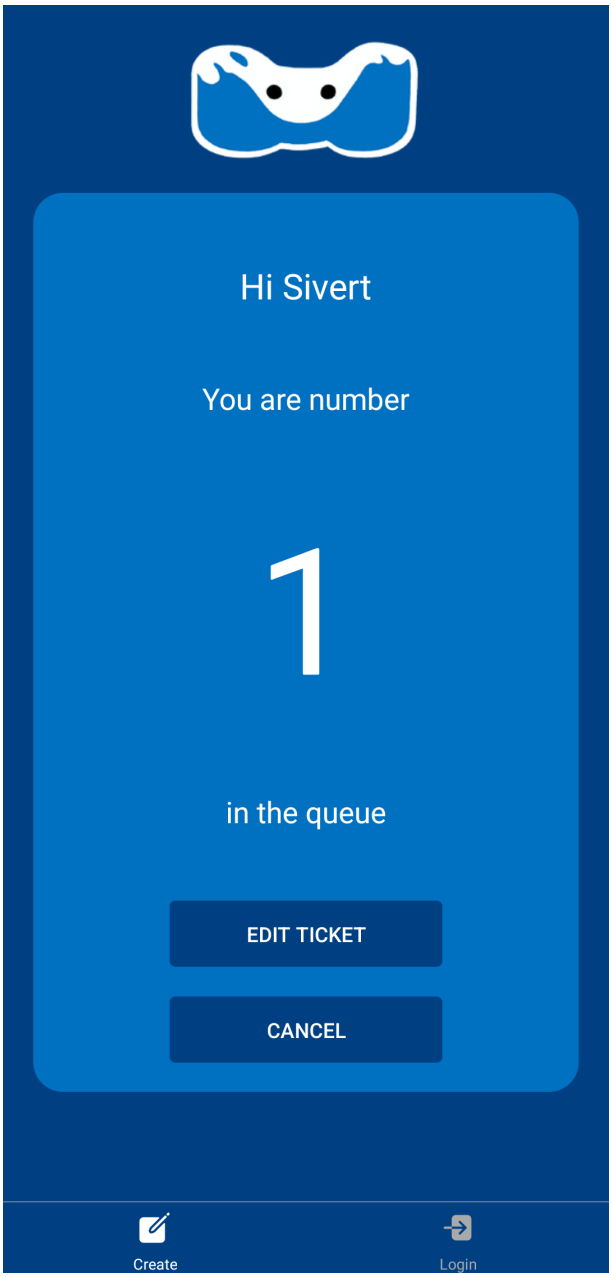
Figure 24: Finished product screen for logged in users creating a ticket

When a user logs in to our application they will be redirected to the create a ticket screen for logged-in users. This screen is almost identical to the create ticket screen for anonymous users. The only difference is that when a user is logged in their nickname will already be filled in. In other words, this create ticket page consists of the same form but without the name text input as for the anonymous users.

6.4.3 Ticket queue



(a) Ticket queue screen light mode

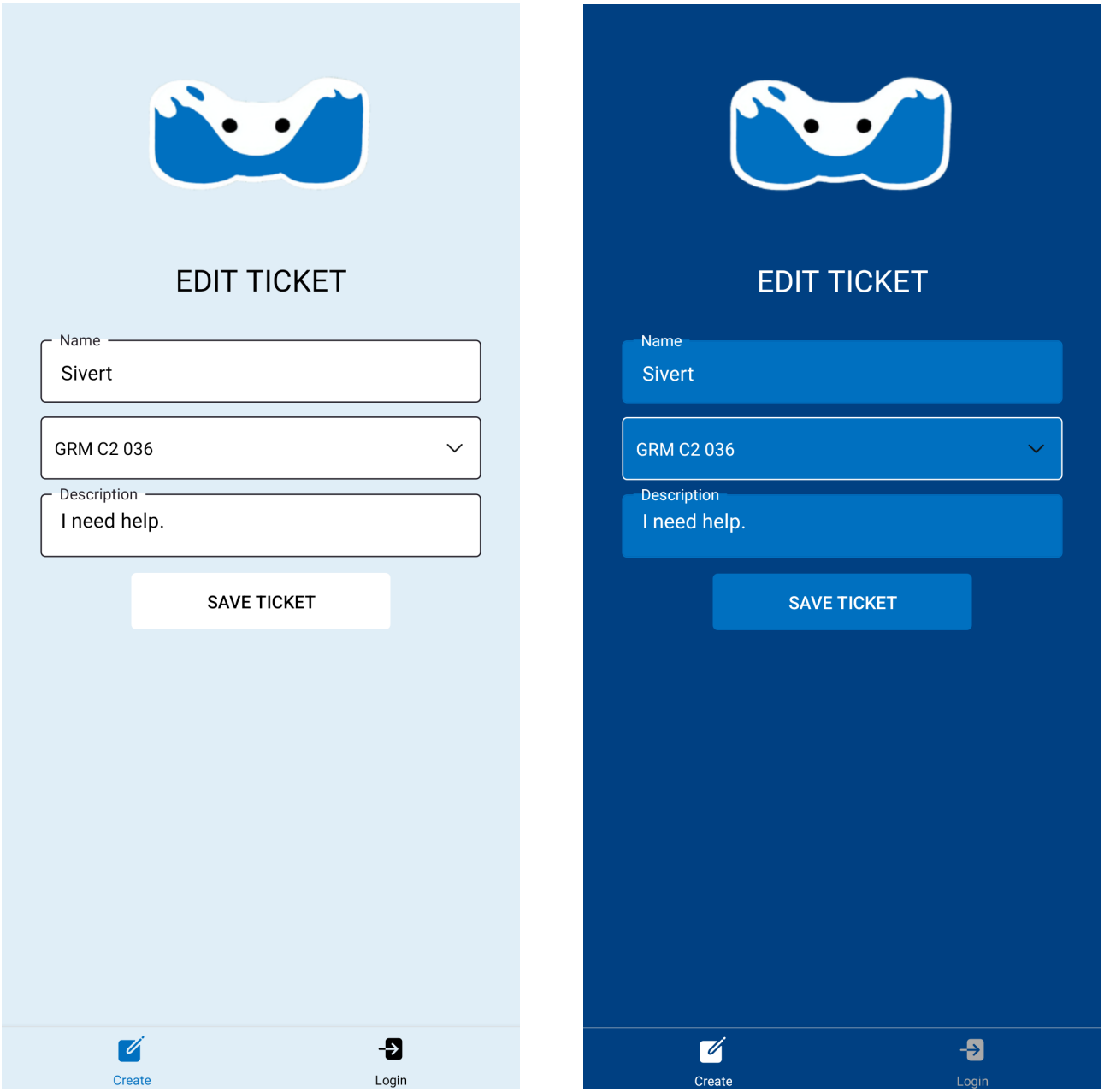


(b) Ticket queue screen dark mode

Figure 25: The finished product ticket queue screen

The ticket queue screen is the screen the user gets redirected to after submitting a ticket to the helplist. This screen consists of a box that greets the user by their nickname and shows them their current position in the queue. Their position in the queue will be updated live. This lets the user know how many people will be helped before it is their turn. At the bottom of the screen, there are two buttons "Edit ticket" and "Cancel". The "Edit ticket" button redirects the user to a screen where they can edit their ticket while the "Cancel" button cancels their current ticket and redirects them back to the Create ticket screen.

6.4.4 Edit ticket



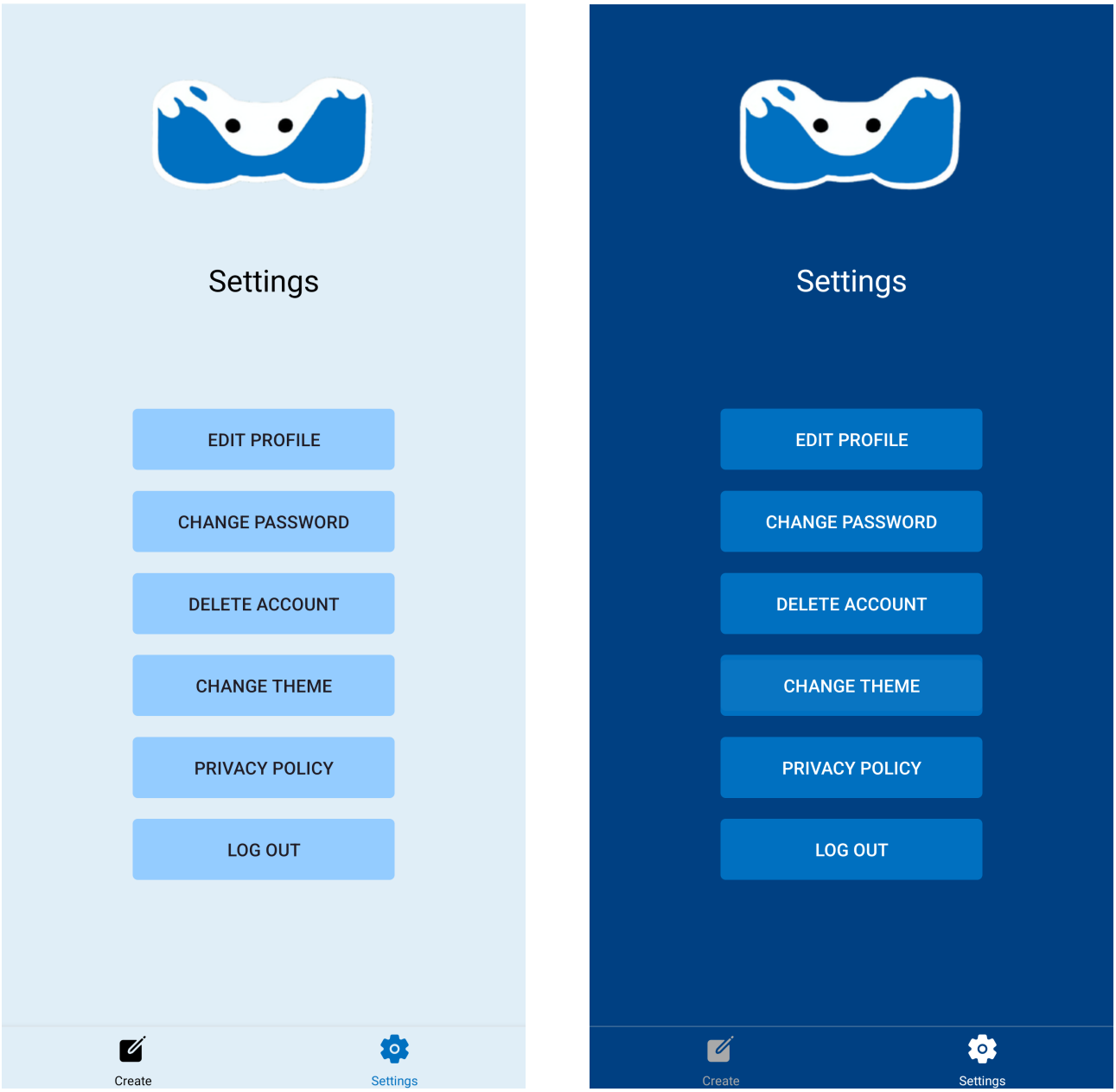
(a) Edit ticket screen light mode

(b) Edit ticket screen dark mode

Figure 26: The finished product screen for editing tickets

The edit ticket screen is the screen you get redirected to after pressing the "Edit ticket" button in the queue. This is a screen that is similar to the create a ticket screen. It has the same exact form but with the title "Edit ticket" instead of "Create ticket". The major difference in the edit ticket screen is that the text input boxes will be pre-filled with the information the user entered when they first created the ticket. This is to make it user-friendly for the user to edit their ticket. Once the user has edited the ticket to their liking they can press the edit button to edit the ticket and return to the queue. The user's position in the queue will remain the same even if they have edited their ticket.

6.4.5 Student settings screen



(a) Student settings screen light mode

(b) Student settings screen dark mode

Figure 27: The settings page for logged-in students

The student setting screen is a screen the user can reach from the navigation bar at the bottom of every screen. This screen presents the user with five buttons; edit profile, change password, delete account, change theme and log out. The profile buttons open a modal where they can change their name, email, and discord tag. If the user presses the change password button they get redirected to a screen where they can change their password. The delete account button lets a user delete their account and get logged out at the same time. The change theme button is a toggle between light and dark mode and lastly is the logout button that let the user log out of their profile.

6.4.6 Navigation bar for anonymous students

(a) Anonymous student navigation bar light mode

(b) Anonymous student navigation bar dark mode

Figure 28: Final product navigation bars for anonymous students

The navigation bar for anonymous students has two pages "Create" and "Login". The "Create" page redirects them to the screen where an anonymous user can create tickets if they need help. The "Login" page is the page where they can log in, and register for our application.

6.4.7 Navigation bar for logged-in students

(a) Logged-in student navigation bar light mode

(b) Logged-in student navigation bar dark mode

Figure 29: Final product navigation bars for logged-in students

The navigation bar for logged-in students is similar to the navigation bar for anonymous students. The difference is that for logged-in students the "Login" page is swapped out with a "Settings" page. As a logged-in user is logged in to our application they have no need for a "Login" page. Instead, we give them the "Settings" page where they can change the settings for their account.

6.5 Student Assistant's view

6.5.1 Overview of lab queues

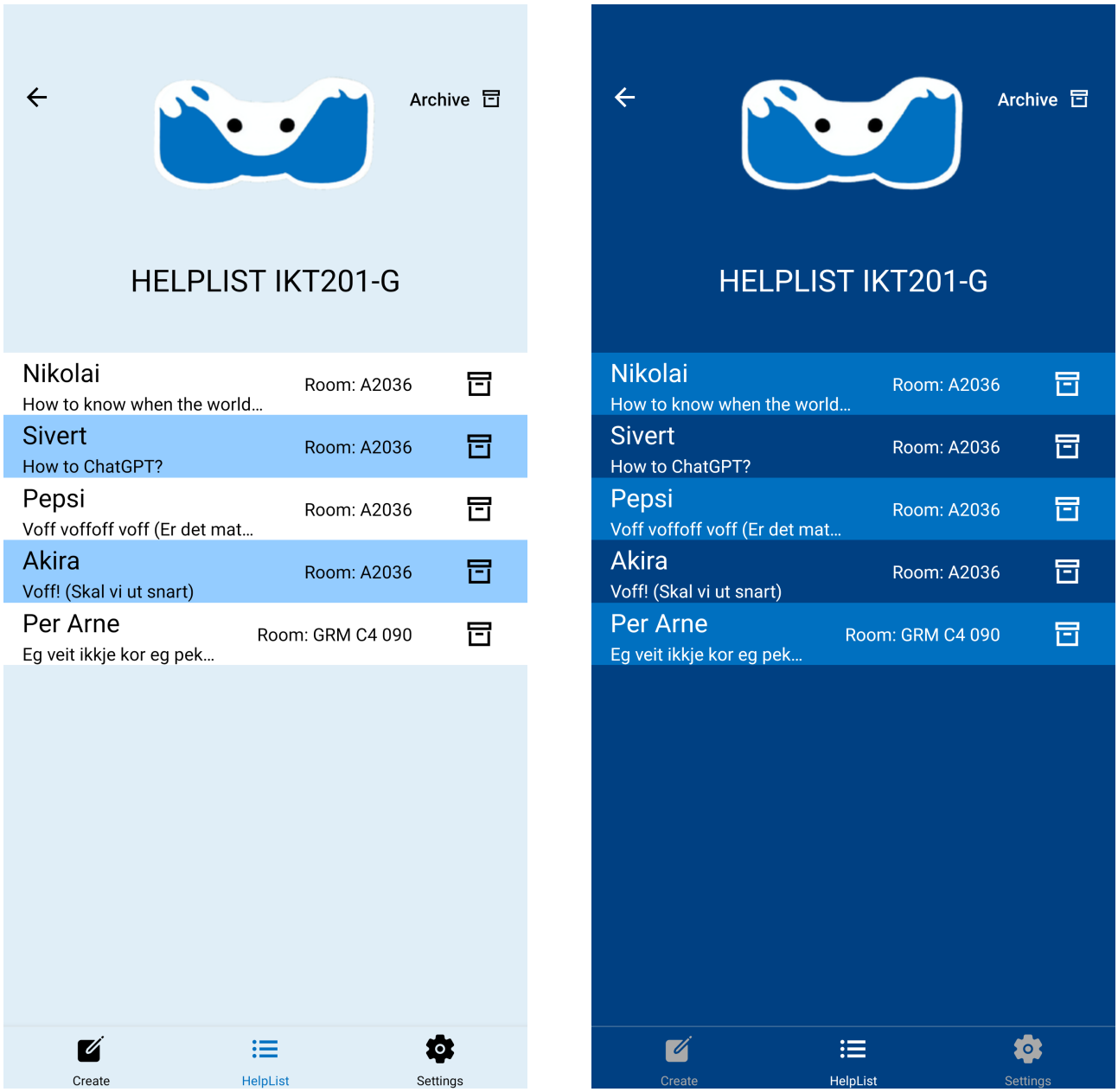
(a) Lab queues light mode

(b) Lab queues dark mode

Figure 30: Overview of lab queues screen

The first screen a student assistant sees after they log in is the overview of lab queues screen. This screen simply consists of a list of all the courses the logged-in student assistant is assigned to. From here the student assistant can click on any of the courses in the list to be redirected to their corresponding help list.

6.5.2 Helplist for courses



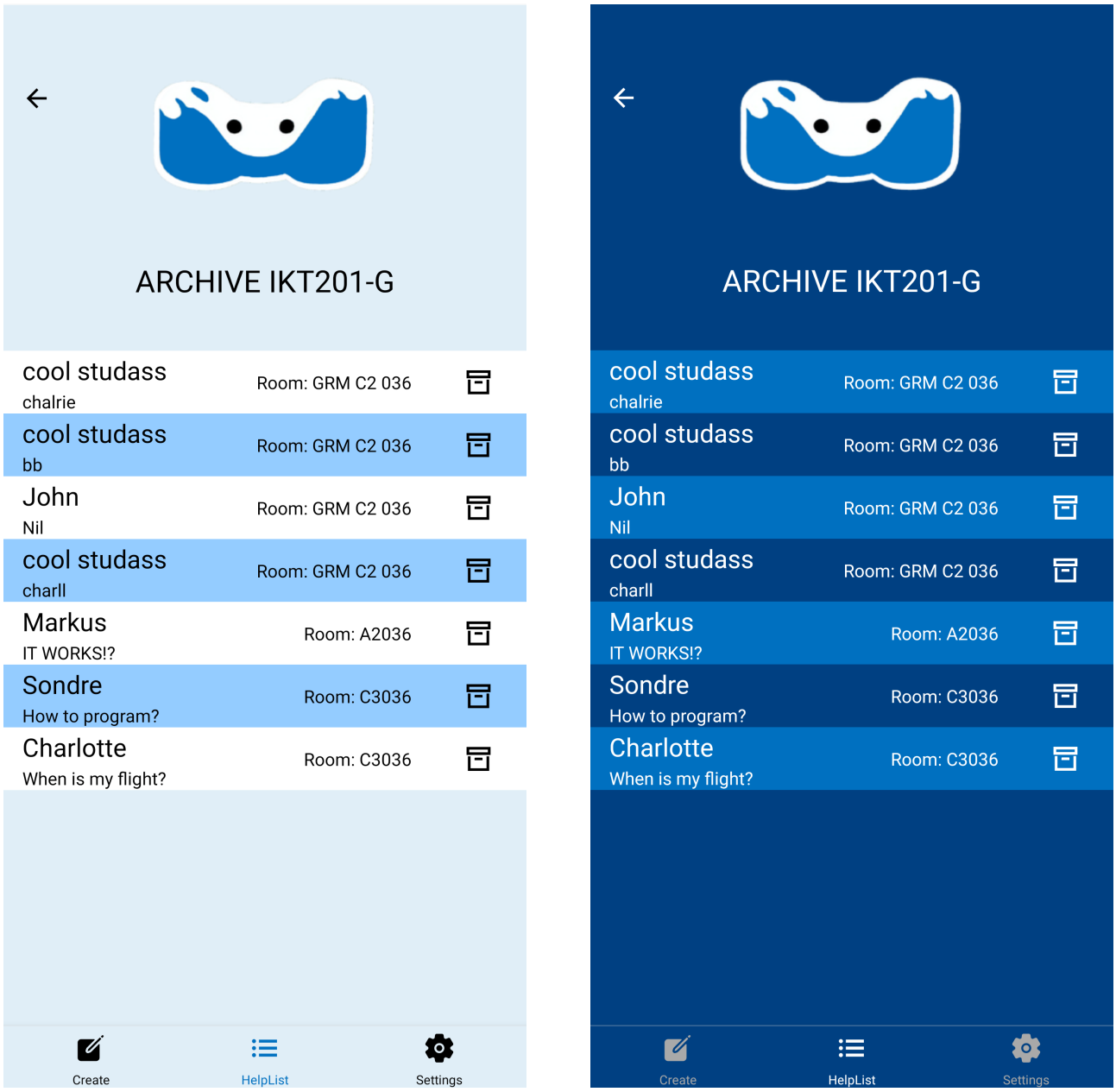
(a) Helplist light mode

(b) Helplist dark mode

Figure 31: Helplist for courses

The helplist screen is the screen the student assistant gets redirected to after choosing one of their courses from the lab queues overview. This screen contains a list of all the tickets that have been submitted to the lab queue for the chosen course. All the tickets will be sorted with the oldest tickets being on the top of the helplist, and the newest being at the bottom. Essentially the tickets at the top of the screen are the ones that are first in the queue. Every ticket in the helplist will show the student's nickname, the room they are in, and the description of their problem. Once a student assistant has successfully helped a student with their ticket they can press the archive icon next to their ticket to complete their ticket and send it to the archive. This screen was changed somewhat from our original Figma design. We switched from discord to room number on the right of every ticket because we needed to know which room the ticket came from. We added a better archive icon button for completing tickets and also an archive icon on the top-right to the screen to go to the archive.

6.5.3 Archive



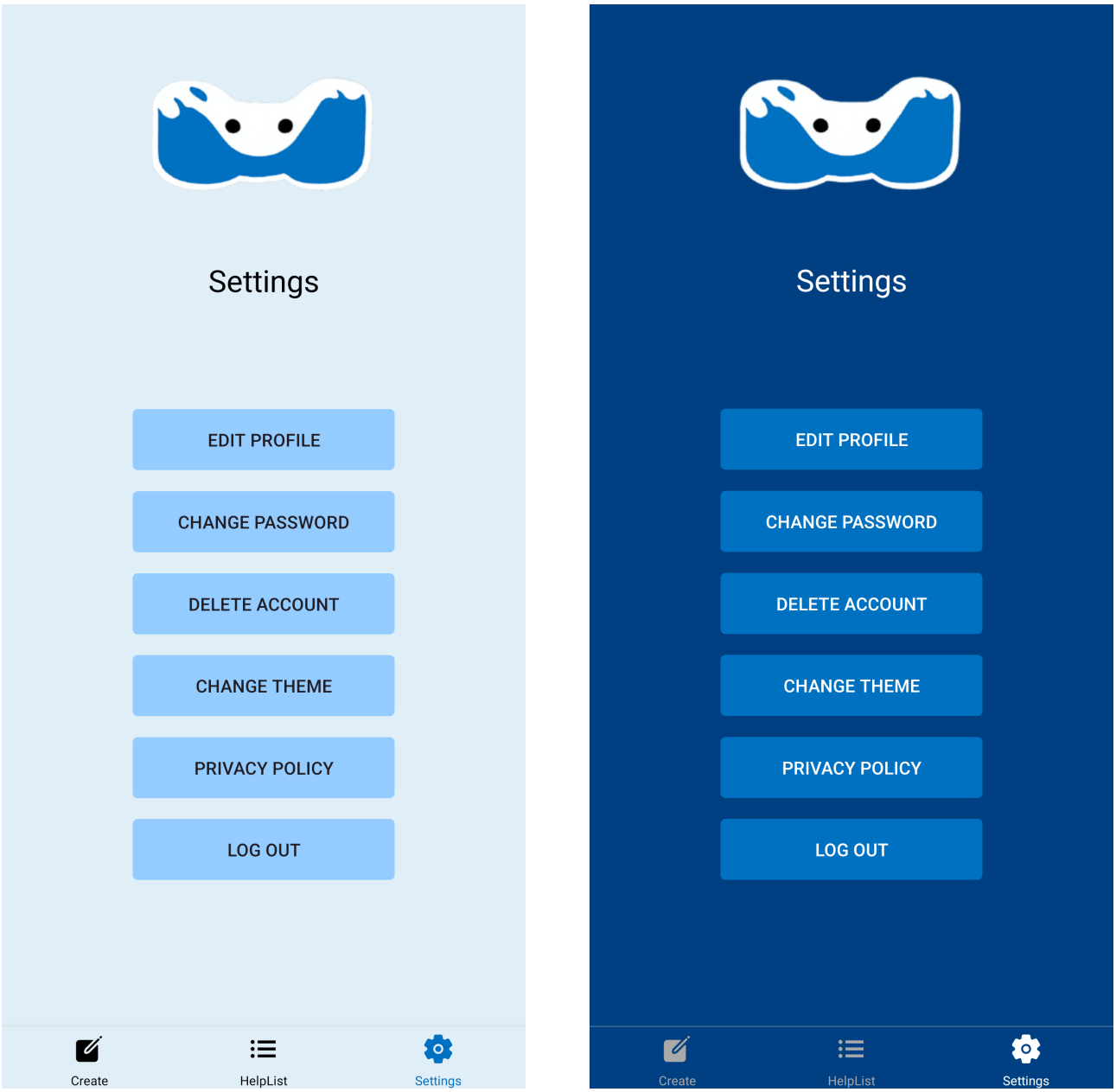
(a) Archive light mode

(b) Archive dark mode

Figure 32: Archive for courses

The archive screen is where tickets end up after they have been marked as completed by a student assistant. This screen consists of a list of all the completed tickets for a specific course. Similarly to the helplist each ticket in the archive list contains the nickname of the student, the room they're in, and the description of their problem. In the archive, the student assistant can go through old tickets if needed, and they can even return tickets from the archive to the helplist if they were marked as completed by mistake. We added a arrow button on the top-left to navigate back to the helplist.

6.5.4 Student assistant settings



(a) Student settings screen light mode

(b) Student settings screen dark mode

Figure 33: The settings page for logged-in students

The student assistant settings screen is identical to the settings screen of a normal user. This is because a student assistant account is just a slightly upgraded user account. They can do everything a user can, but also keep track of the courses they are assistants in. They have no need for any additional setting functionality for this job.

6.5.5 Navigation bar for student assistants

(a) Student assistant navigation bar light mode

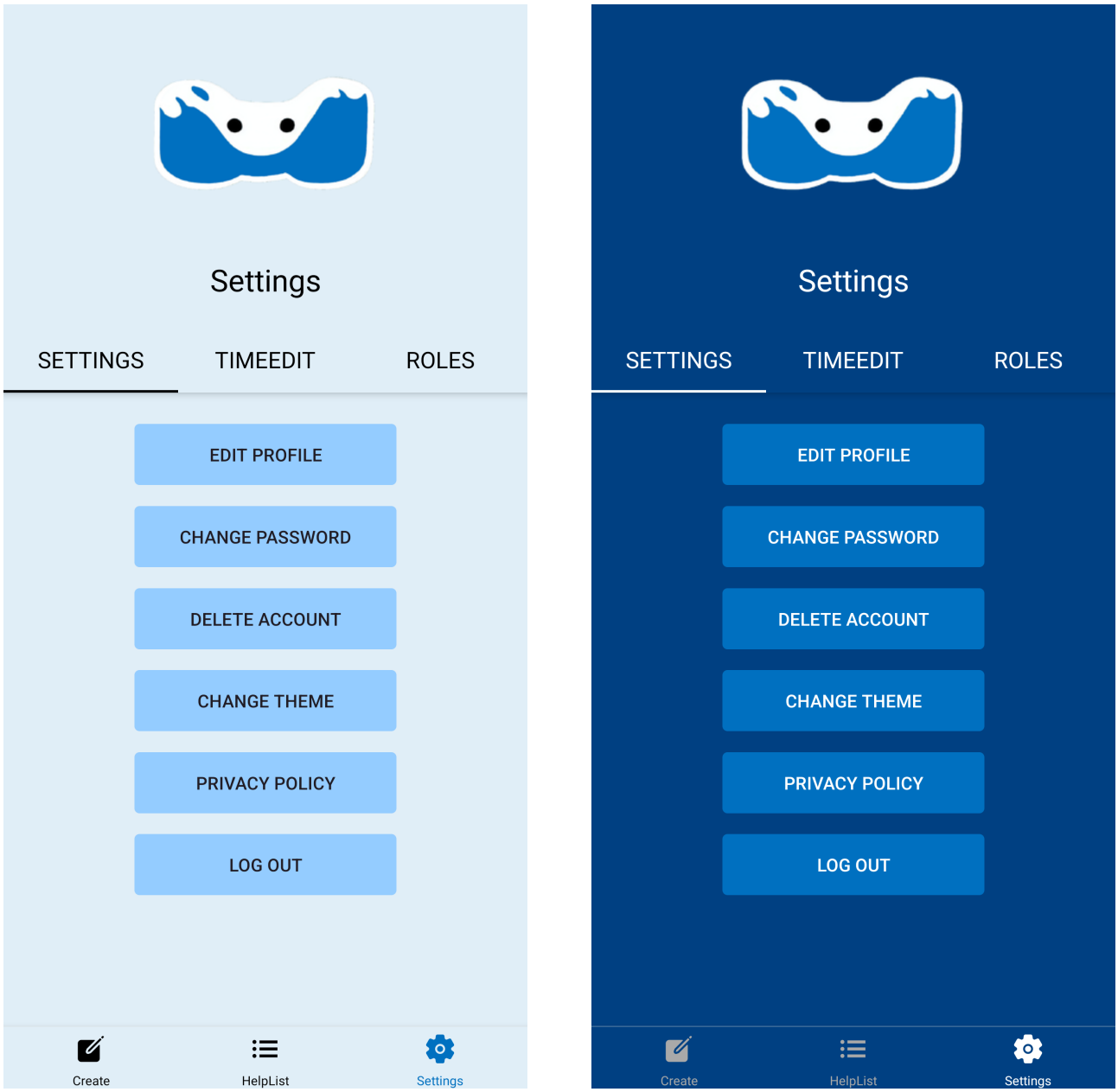
(b) Student assistant navigation bar dark mode

Figure 34: Final product navigation bars for student assistants

The navigation bar for student assistants consists of three pages: "Create", "Helplist", and "Settings". The "Create" page directs the student assistant to the Create a Ticket page. Student assistants are still students so they still need to be able to attend labs for their own courses. The "Helplist" page directs them to the lab queues screen where they have an overview of the courses they are a student assistant in. This is also the page where they can manage tickets for a course they work in. Lastly, the "Settings" page directs them to their account settings.

6.6 The Admin view

6.6.1 Settings

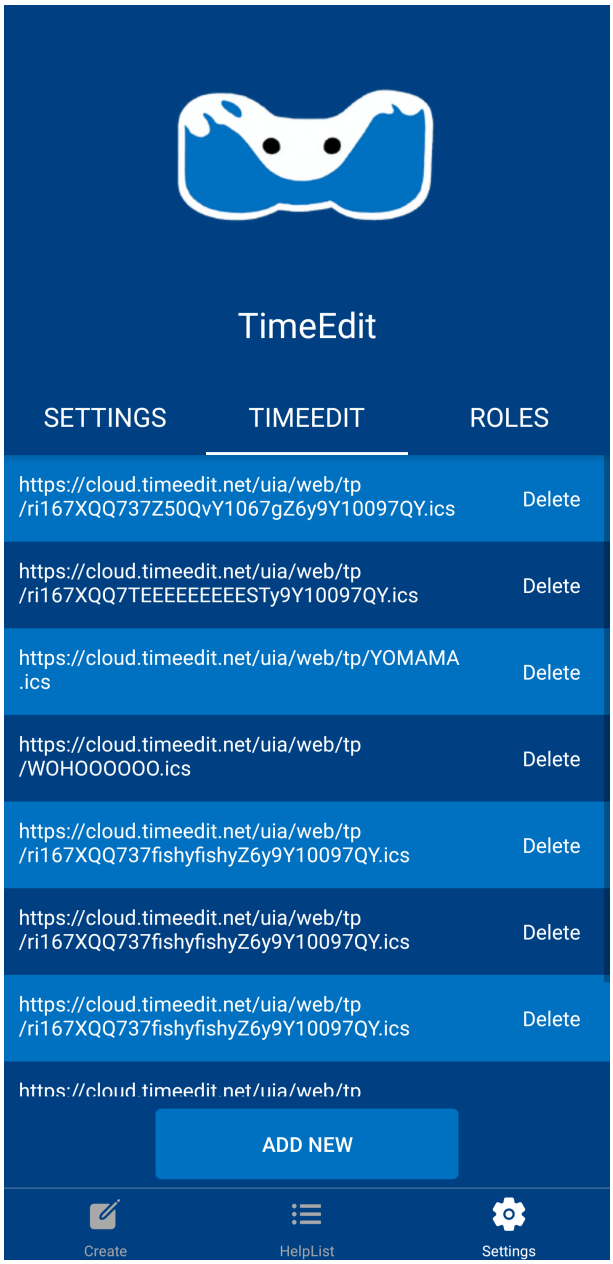


(a) Admin general settings light mode

(b) Admin general settings dark mode

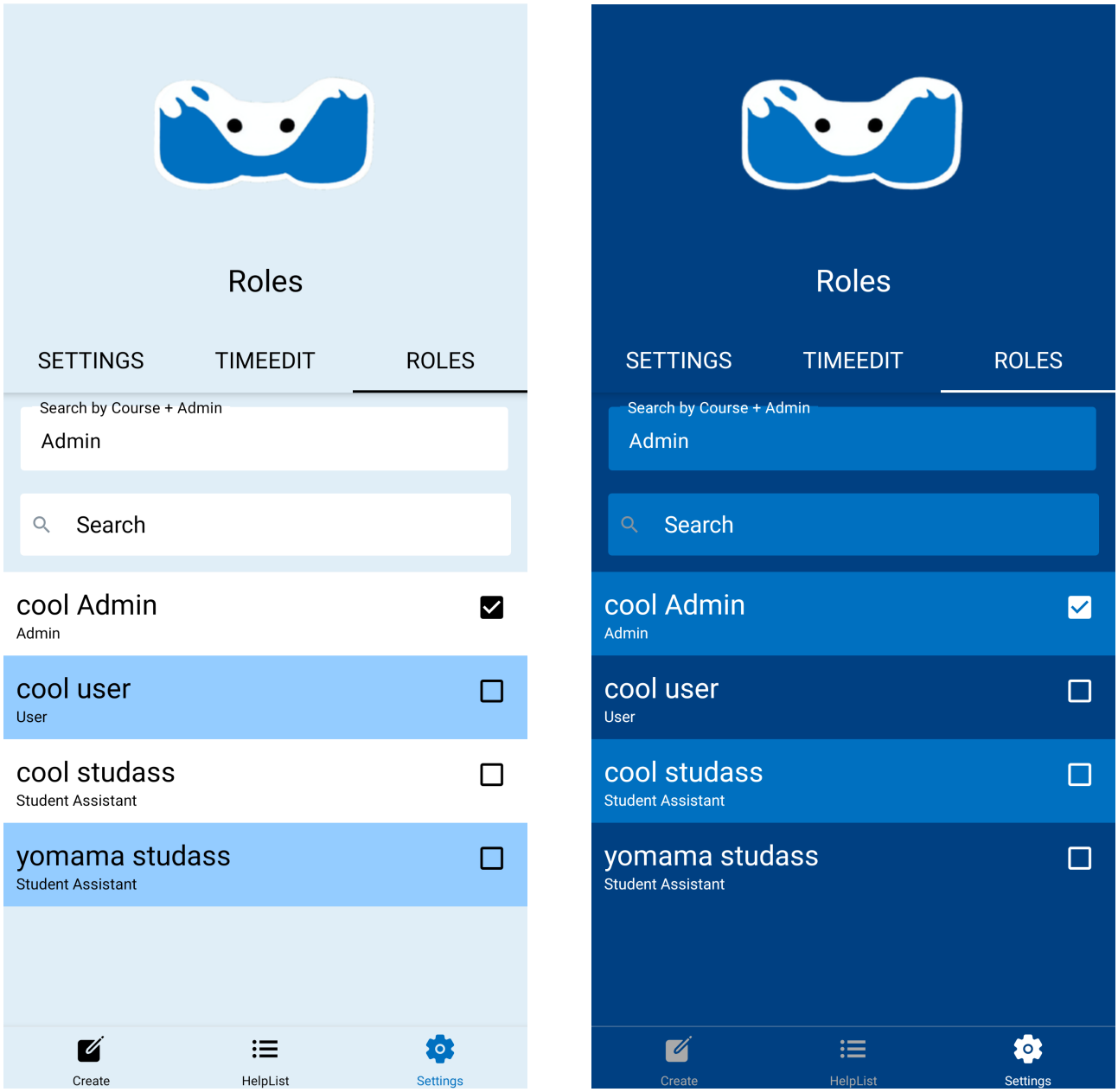
Figure 35: Admin general settings

(a) Admin TimeEdit settings light mode



(b) Admin TimeEdit settings dark mode

Figure 36: Admin TimeEdit settings



(a) Admin roles settings light mode

(b) Admin roles settings dark mode

Figure 37: Admin roles settings

The admin settings screen has three different settings screens the admin can navigate. These are the normal settings, TimeEdit settings, and roles settings. The normal settings screen will be the exact same setting screen as the students have with five buttons; profile, change password, delete account, change theme, and log out. The TimeEdit settings screen lets an admin view the current TimeEdit links in the database and remove/add TimeEdit links to the database. This essentially lets the admin add additional courses to the database. The roles settings screen is where the admin can give and take away roles from other users and student assistants. Admins can turn any user into a student assistant or admin, and they can also turn other admins and student assistants back into normal students.

6.6.2 Navigation bar for admins

(a) Admin navigation bar light mode

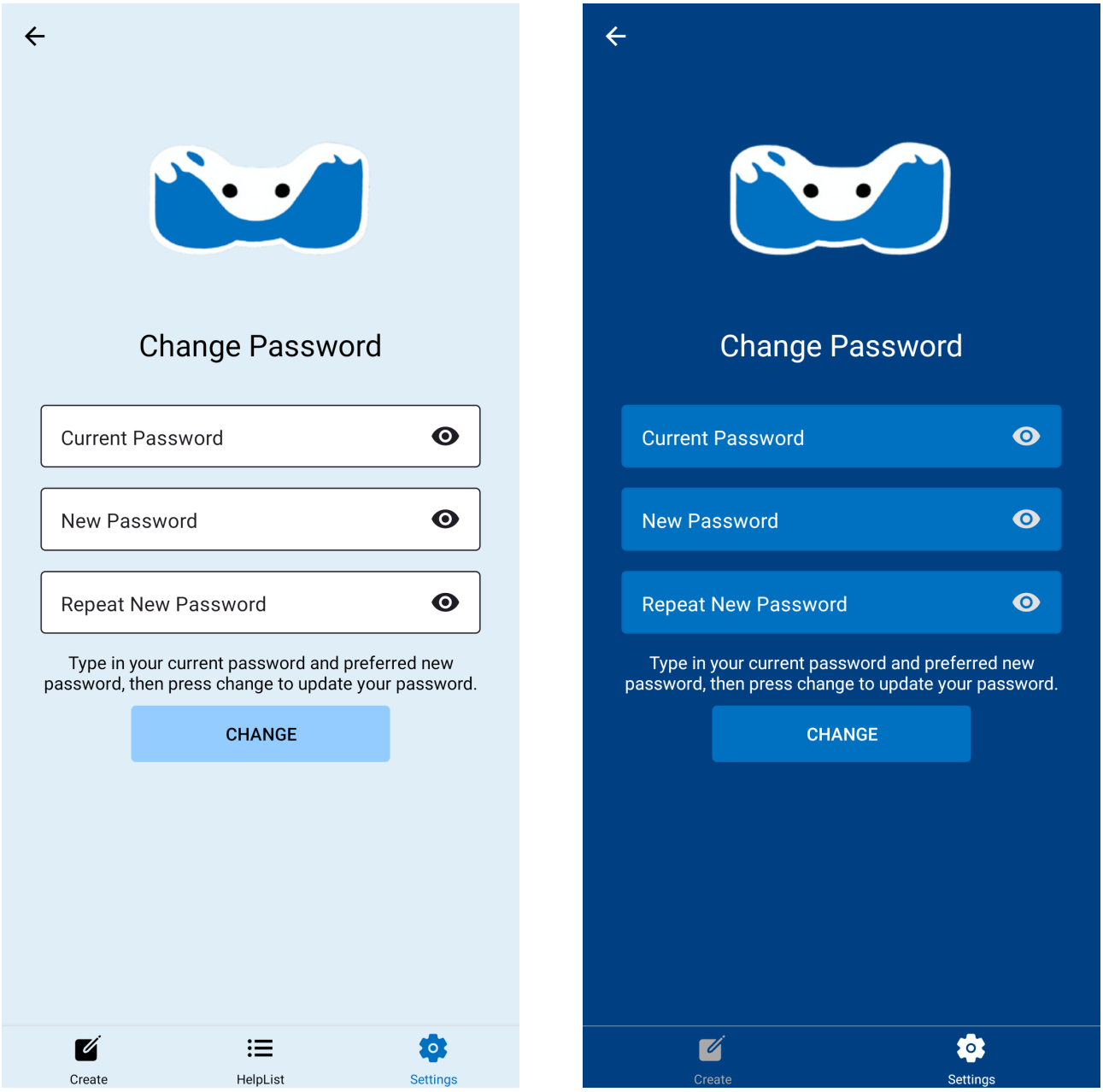
(b) Admin navigation bar dark mode

Figure 38: Final product navigation bar for admins

The navigation bar for admins is identical to the navigation bar for student assistants with the "Create", "Helplist", and "Settings" pages. An admin will probably never use the "Create" page but it is there so they do have access to everything the application contains. Admin/teachers are sometimes student assistants in their own lab hours so they have access to the "Helplist" page. Lastly, the admins have access to the "Settings" page which is where they can manage their own settings, TimeEdit links for the application, and the roles of other users.

6.7 Other pages

6.7.1 Change password



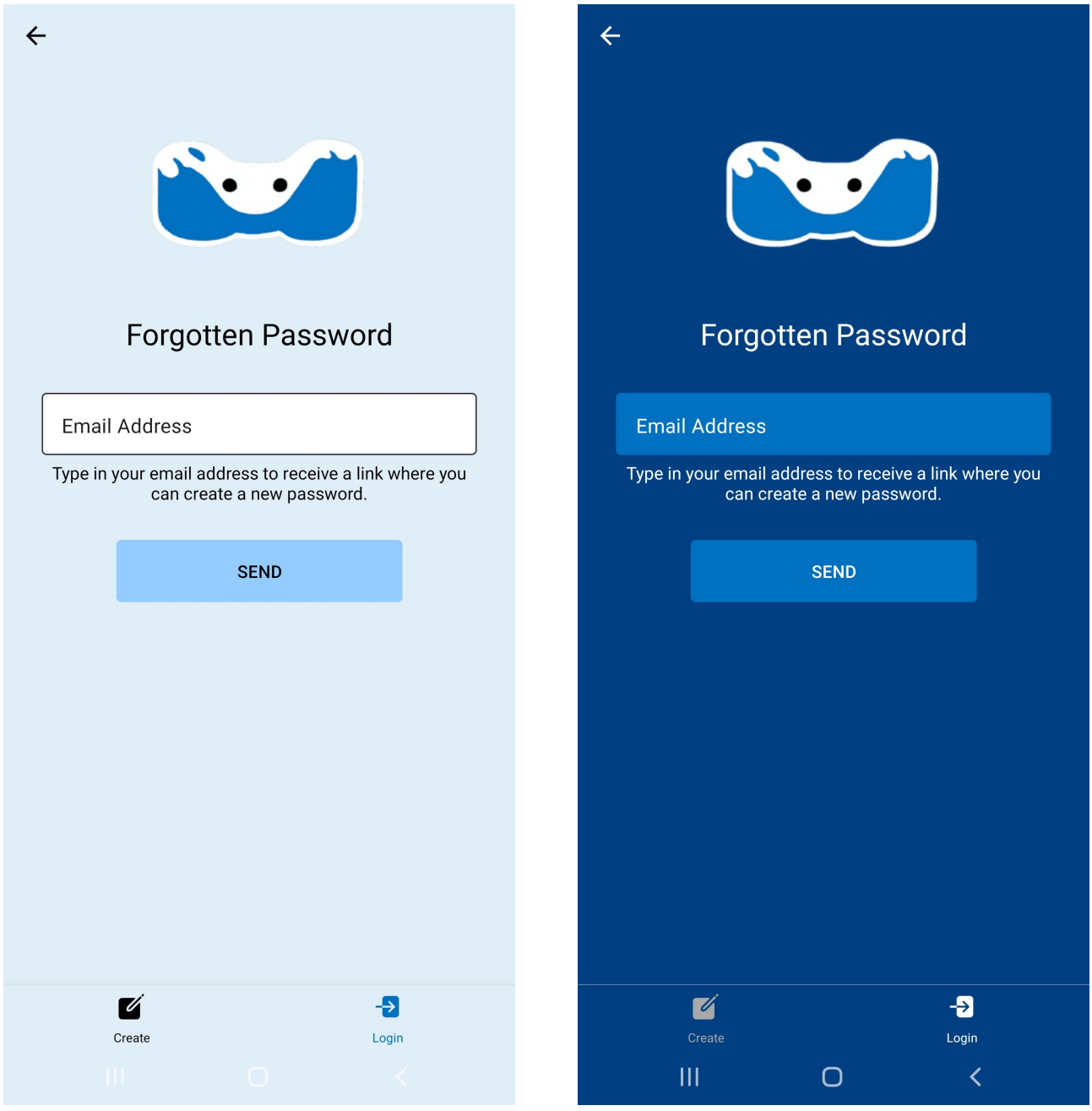
(a) Change password light mode

(b) Change password dark mode

Figure 39: Finished product change password screen

The change password screen can be reached by any type of user account from the settings page. This screen consists of a headline "Change password" and a form where the user can enter their preferred new password twice to change it. The two passwords entered have to match for the user to be able to successfully change their password. If the password change is a success the user will be redirected back to the settings page.

6.7.2 Forgotten password



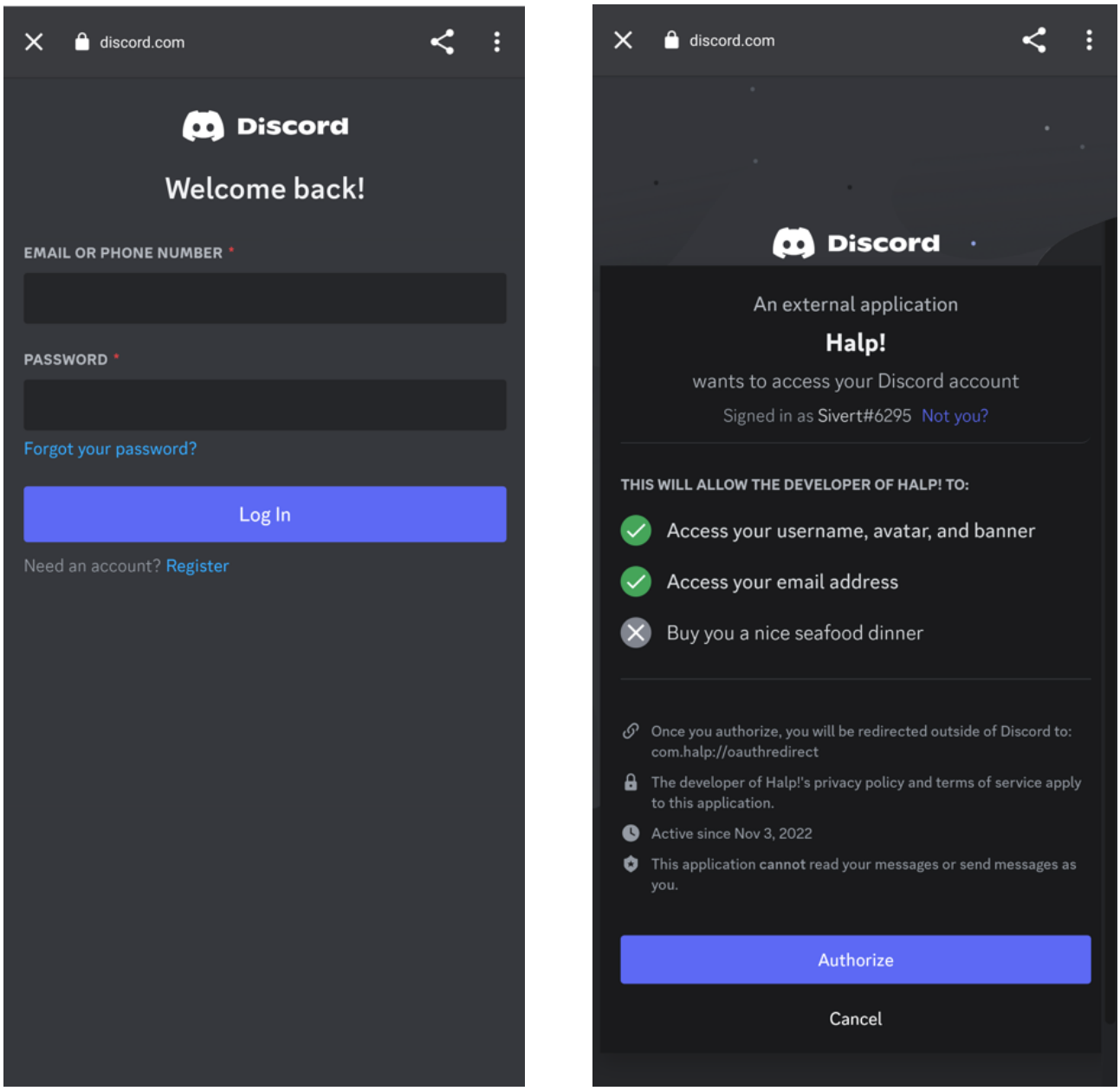
(a) Forgotten password light mode

(b) Forgotten password dark mode

Figure 40: Finished product forgotten password screen

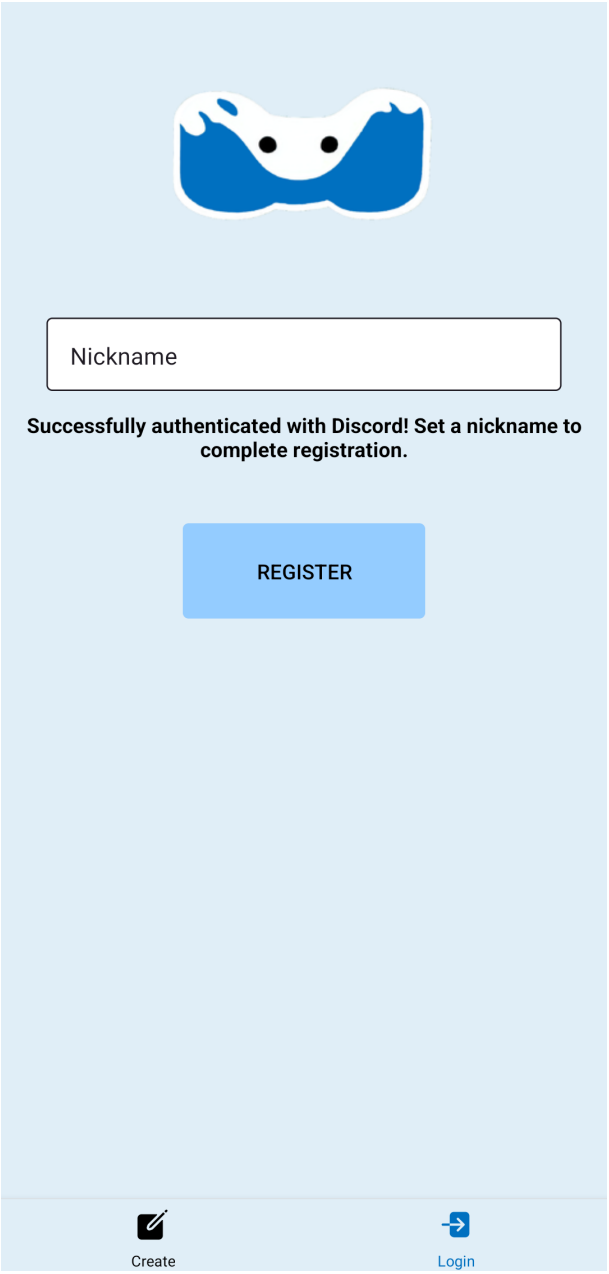
The forgotten password page can be reached from the "Forgotten password" button on the login page. This screen consists of a headline "Forgotten Password" and a form where the user can enter their email address. Once the user has entered their email and pressed the send button they will be redirected back to the login page. If the user entered an email that is registered in our application they will receive an email with a link to a page where they can change their password.

6.7.3 External login

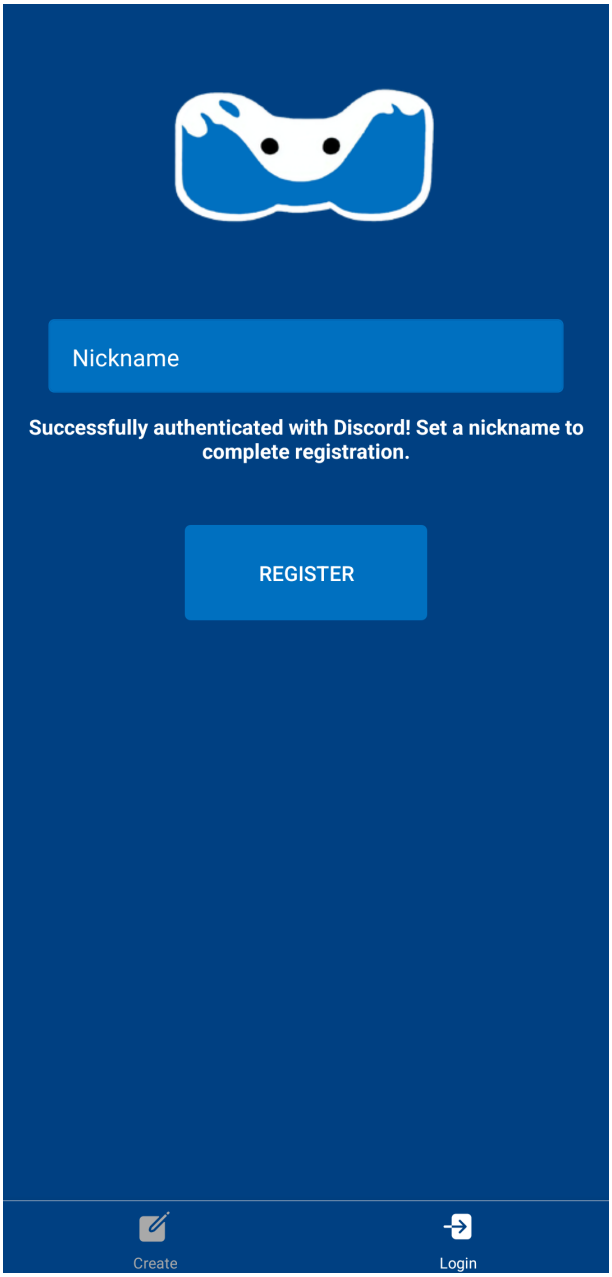


(a) Discord external login **(b)** Discord authentication

Figure 41: Finished product logging in and authenticating through Discord



(a) Setting Discord nickname light mode

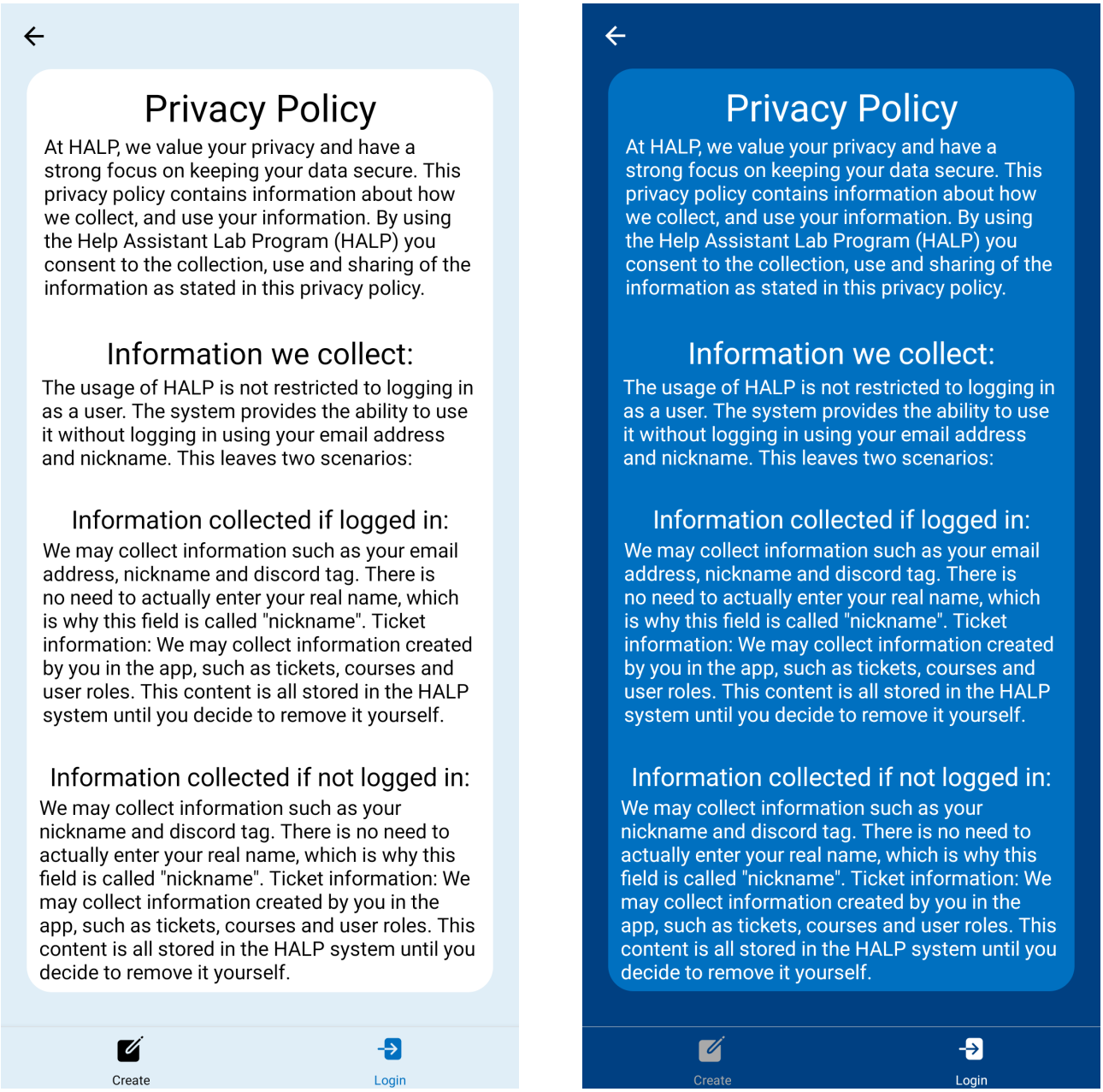


(b) Setting Discord nickname dark mode

Figure 42: Finished product setting nickname after logging in through discord

Our application gives users the option of logging in with Discord if they prefer. When a user clicks the "Discord" button on the login screen they will be redirected to a screen where they can log in and then authorize with Discord. After logging in using Discord the user is redirected to another screen where they can set the nickname for their account. A user only has to set their nickname once upon their first login.

6.7.4 Privacy policy



(a) Privacy policy light mode

(b) Privacy policy dark mode

Figure 43: Finished product privacy policy

The privacy policy was not in our initial design, but rather a screen we thought of later as we wrote our own custom privacy policy for the application. This screen contains information about the type of information we collect from our users and how we use it. The privacy policy screen consists of a box where our privacy policy is written. This box can be scrolled in to read the entire document. An anonymous user can find the privacy policy at the bottom of the login page while a logged-in user can find it in their settings.

7 Implementation

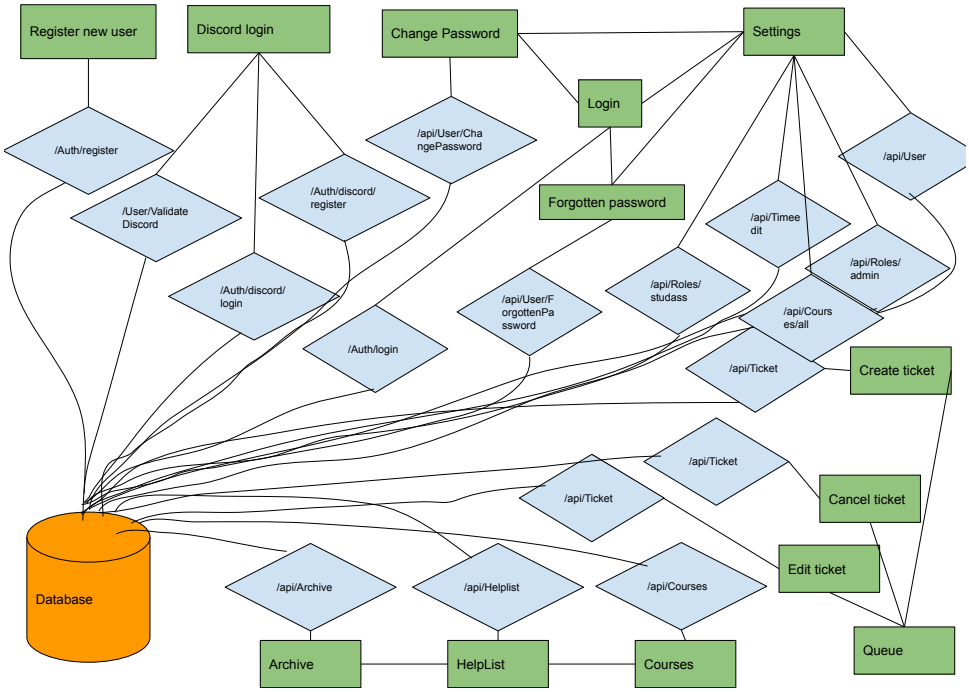


Figure 44: Architecture

7.1 Database

To further unify the HALP system [21], this application utilizes the same database as the .NET application. The database is illustrated and gathered from the .NET project we made:

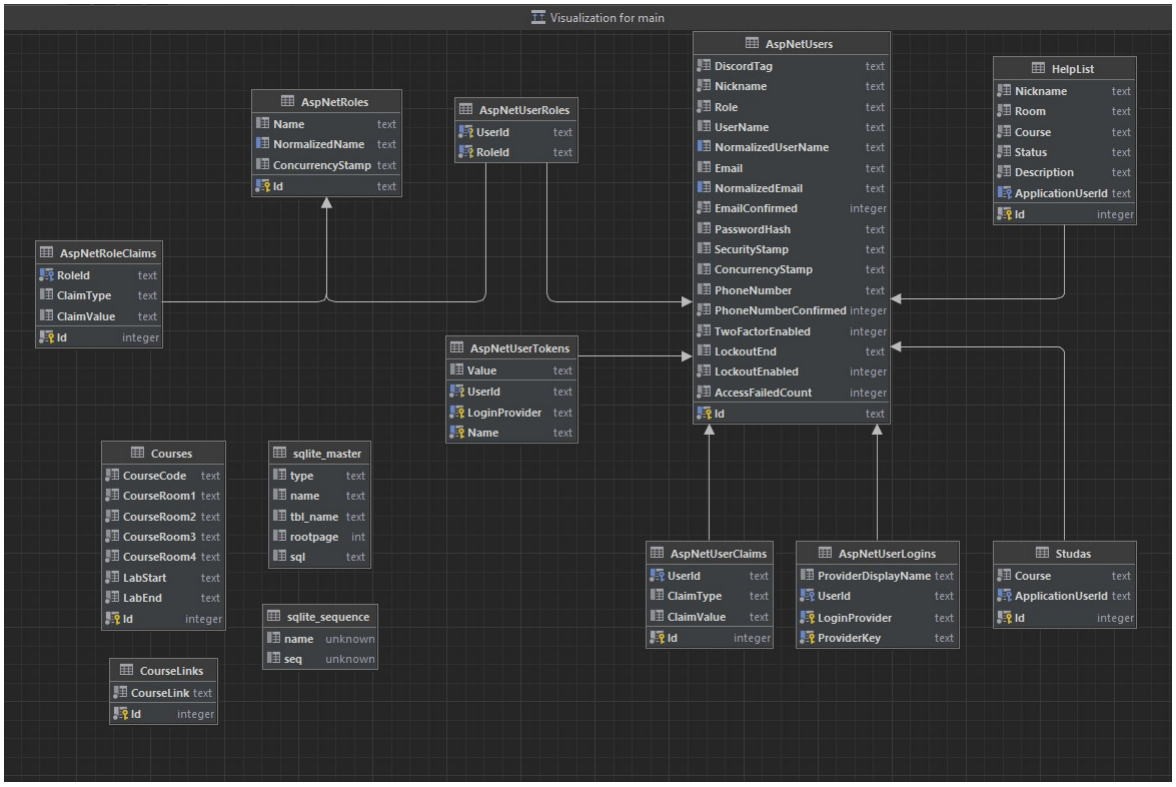


Figure 45: Backend - database

7.2 Testdata

In our application, we needed to hardcode some test data to make this possible to test the application after the semester is over. This is done because the calendar in Timeedit will be wiped when the semester is done. We also made some users that can be used to log in. All of this information will be described in a readme file.

7.3 Navigation

The navigation bar is a predefined component from React Navigation with some custom styling. It has two different views, one for student assistants and one for users without a specific role. The component receives two boolean parameters, "isStudass" and "isLoggedIn", which are used to determine which tab screens should be displayed in the bar. Each tab has its "defined" screens that are set to allow passing values through routing.

7.4 Theme

The theme has been implemented using react-redux and context logic from React. The context provider has been placed in the App file, which is the root file of the application. The context logic prevents the need to pass down values to child components but rather retrieves values using a useContext hook from Redux. The advantage of this is that there is less chance of errors and re-rendering issues. The theme, which is stored in the context provider, is retrieved from a custom hook called themeHook. This hook returns the theme and a function to change the theme. The theme is stored in a global state to be accessed at any time.

7.5 Login

The login page functions similarly to the IKT 201 login page, with the exception that this time there is front-end validation. Validation occurs both in the front end and back end. The front end uses regex to validate if the email contains an "@" symbol, characters, and two or three characters after the period. Similarly, regex is used to validate the password, which tests whether the password contains a minimum of eight characters, one special character, one uppercase letter, one lowercase letter, and at least one number. When the validation passes, a POST request is made to the API with the data. The back end then checks if the email exists in the database and if the password matches. If the request is successful, user information is returned, such as the nickname, id, email, role, token, and the Discord tag if present. This information is stored in a global state called 'user'. Afterward, the user is redirected to the settings page. If there is an error from the API, the user will remain on the login page.

7.5.1 API Endpoints for the "Login" screen

Endpoint: /auth/login

Type: POST

Requires auth: No

Description: The login endpoint takes a json object from the user containing an email and password. These credentials are then validated and the API returns an object containing all the information the application needs.

Request Body	Success Response (200)	Error Response (401)

<pre>{
  "email": "string",
  "password": "string"
}</pre>
	<pre>{
	  "Id": "string",
	  "Nickname": "string",
	  "Email": "string",
	  "Token": "string",
	  "Role": "string",
	  "DiscordTag": "string"
	}</pre>
		Unauthorized

7.6 Discord-login

Logging in with Discord was implemented using the authorize function from the react-native-app-auth library. The first step was to retrieve the client and secret IDs and use them in combination with other parameters to create a discordConfig object which would be passed to the discord OAuth API. Passing this config gave us permission to retrieve data about the user required to either login or register the user locally in our own backend. In order to figure out if the user was already registered in our DB, we checked if either the user's e-mail or the unique discord ID matched any existing records using the */api/User/validateDiscord* endpoint. If they did, we simply log the user in. Otherwise, the user is redirected to a registration screen and prompted to set a nickname and then register the user.

7.6.1 API Endpoints for the "Discord Login" screen

Endpoint: /auth/discord/login

Type: POST

Requires auth: No

Description: The server takes in the discordId parameter, fetched through discords OAuth API and returns the user object that matches the discordId.

Request Body	Success Response (200)	Error Response (401)

<pre>{
  "discordId": "string"
}</pre>
	<pre>{
	  "id": "string",
	  "nickname": "string",
	  "email": "string",
	  "token": "string",
	  "role": "string",
	  "discordTag": "string"
	}</pre>
		Unauthorized

Endpoint: /api/User/validateDiscord

Type: PUT

Requires auth: No

Description: The server takes in the discordId and email parameters retrieved from discords OAuth API and checks if either parameter exists in the DB. This endpoint is designed to distinguish between the user being logged in or registered when pressing the Discord button.

Request Body	Success Response (204)	Error Response (404)
{ "email": "string", "discordId": "string" }	No Body	No Body

7.7 Register

Similarly to our login screen, the register screen uses API calls to the HALP database as its backend. This API call takes the parameters "email", "username", "password", and "discordTag". The API call first checks if there already is an account in the system with the entered email. It also checks that all the parameters are filled and not empty. If one of these clauses is met the response will be unsuccessful. If this is not the case the entered information will be inserted into the database which successfully registers an account. There is some logic in the actual register.tsx file. This logic checks that the two entered password match, and follows the guidelines for a correct password. The logic also makes sure that ass the text boxes are filled.

7.7.1 API Endpoints for the "Register" screen

Endpoint: /auth/register

Type: POST

Requires auth: No

Description: The server validates the password and returns a 201 Created response if the register was successful. If the user creation was unsuccessful in any way, such as a user already existing with that username or email, or invalid password, the server will return a 400 Bad Request with the error message.

Request Body	Success Response (201)	Error Response (400)
{ "email": "string", "nickname": "string", "password": "string", "discordTag": "string" }	{ "email": "string", "nickname": "string", "discordTag": "string" }	{ "errorID": ["errorDescription"] }

Endpoint: /auth/discord/register

Type: POST

Requires auth: No

Description: The server recieves an email, nickname, discordTag and discordId, which it validates based on certain rules such as emails needing to be unique, etc. and, if successful, returns the user object which was just passed to it. If unsuccessful, it returns 400 Bad Request with a body containing the error.

Request Body	Success Response (200)	Error Response (400)

{
"email": "string",
"username": "string",
"discordTag": "string"
"discordId": "string"
}
	{
	"email": "string",
	"username": "string",
	"discordTag": "string",
	"discordId": "string"
	}
		{
		"errorID": [
		"errorDescription"
		]
		}

7.8 Settings

The code starts by importing the necessary libraries and modules for building the user interface and managing the state.

The Tabs component displays different tabs based on the user's role. Administrators see additional tabs for TimeEdit and Roles, while non-administrators only see the Settings tab.

The main component, Settings, handles the user's settings, including profile editing and account deletion. It uses models for these actions and performs API calls to update the server accordingly.

Next, there is a component called TimeEdit, which fetches and displays TimeEdit data from a server. It allows admins to add new links and delete existing ones.

Another component, Roles, manages user roles it is where users get assigned to either student assistant in a course or admin. Administrators can assign roles to users, and the component displays a list of users with their respective roles.

These components provide users with an interface for managing their settings, handling time edits, and managing user roles.

Overall, the code imports necessary libraries defines components for different functionalities, and ensures smooth transitions between the sections.

7.8.1 API Endpoints for the "Settings" screen

Endpoint: /api/User

Type: PUT

Requires auth: Yes

Description: The server recieves an ID and a nickname, email and discordTag. The server then modifies the user with the corresponding ID and changes its value to the provided parameters. If the user is not found then the server responds with 404 Not Found.

Request Body	Success Response (204)	Error Response (404)

{
"id": "string",
"nickname": "string",
"email": "string",
"discordTag": "string"
}
	No content	Not found

Endpoint: /api/User/get

Type: PUT

Requires auth: Yes

Description: The server takes in an email and returns the user object with that email. GET does not recieve an object, PUT does. This is why we use PUT instead of GET to retrieve users. If the email does not match any user, the server responds with 404 Not Found.

Request Body	Success Response (200)	Error Response (404)

<pre>{
  "email": "string"
}</pre>
	<pre>{
	  "id": "string",
	  "nickname": "string",
	  "email": "string",
	  "discordTag": "string",
	  "role": "string"
	}</pre>
		Not found

Endpoint: /api/User

Type: DELETE

Requires auth: Yes

Description: The server takes in a userID and deletes the user with the matching ID from the DB.

Request Body	Success Response (204)	Error Response (404)

<pre>{
  "userID": "string"
}</pre>
	No content	Not found

Endpoint: /api/Timeedit

Type: GET

Requires auth: Yes

Description: The server returns all timeedit links and their IDs

Request Body	Success Response (200)	Error Response (404)

No body	<pre>[
	  {
	  "id": 0,
	  "courseLink": "string"
	  }
	]</pre>
		Not found

Endpoint: /api/Timeedit

Type: POST

Requires auth: Yes

Description: The server takes in a string which is the timeedit link and returns the ID it gave to that link. The link is validated server-side to check if it contains the prefix "https://" as well as the suffix ".html". If the link is invalid, the server will return a 400 Bad Request.

Request Body	Success Response (201)	Error Response (400)

<pre>{
  "link": "string"
}</pre>
	<pre>{
	  "id": 0
	}</pre>
		Bad request

Endpoint: /api/Timeedit

Type: DELETE

Requires auth: Yes

Description: The server takes in an ID and removes the corresponding timeedit link. If there is no timeedit link with the ID, the server responds with 404 Not Found.

Request Body	Success Response (204)	Error Response (404)

<pre>{
  "id": "integer"
}</pre>
	No content	Not found

Endpoint: /api/Courses/all

Type: GET

Requires auth: Yes

Description: The server returns an array containing each course code, given that the person performing the GET is authorized.

Request Body	Success Response (200)	Error Response (404)

No body	<pre>[
	  "string"
	]</pre>
		Not found

Endpoint: /api/User/all

Type: GET

Requires auth: Yes

Description: The server responds with an array containing all users.

Request Body	Success Response (200)	Error Response (404)

No body	<pre>[
	  {
	  "id": "string",
	  "nickname": "string",
	  "email": "string",
	  "isAdmin": true,
	  "discordTag": "string",
	  "courses": [
	  "string"
	  ]
	  }
	]</pre>
		Not found

Endpoint: /api/Roles/admin

Type: PUT

Requires auth: Yes

Description: The server receives a userID and sets the user with that ID to be an administrator.

Request Body	Success Response (204)	Error Response (404)

<pre>{
  "userID": "string",
  "set": true
}</pre>
	No content	Not found

Endpoint: /api/Roles/studass

Type: PUT

Requires auth: Yes

Description: The server receives a userID and a course and sets that combination as true, meaning that the user with the ID will be a student assistant in the given course.

Request Body	Success Response (204)	Error Response (404)

<pre>{
  "userID": "string",
  "course": "string",
  "set": true
}</pre>
	No content	Not found

7.9 Help- and archive- list

7.9.1 Helplist

Both the helplist and the archive list utilize a shared list component. This component is generic and contains minimal logic. Its only function is to handle expand capability and maintain the state of the tickets being 'checked'.

The data for the list is initially retrieved through a GET request and then stored in Redux. In Redux, we store the data per course to avoid mixing the data from different courses. The GET request is only executed if the course has not loaded any data yet. The course code required to execute the GET request is obtained from the routing. To achieve live updates, we use SignalR. Since the same backend as IKT201 is being used, this functionality was already defined. We listen for new tickets, updated tickets, and deleted or deactivated items through SignalR. The response from SignalR is then stored in Redux.

The update of the list is performed using a PUT request. The PUT request invokes certain SignalR tasks that modify the helplist and update the archive list.

7.9.2 API Endpoints for the "Helplist" screen

Endpoint: /api/Helplist?course=course

Type: GET

Requires auth: Yes

Description: The server takes in a course code and returns all tickets related to that course.

Request Body	Success Response (200)	Error Response (404)

No body	<pre>[{
	"id": 0,
	"nickname": "string",
	"description": "string",
	"room": "string"
	}]</pre>
		Not found

Endpoint: /api/Helplist?id=ticketId

Type: PUT

Requires auth: Yes

Description: The ticket with the provided ID is archived and removed from the helplist.

Request Body	Success Response (204)	Error Response (404)
No body	No content	Not found

7.9.3 Archive

As mentioned earlier, the archive list also utilizes a shared list component. The data is retrieved in the same way as the helplist, using a GET request where the course code is sent as the body. The course code is obtained from the routing The response is stored in Redux as a key-value object, where the key represents the course code. Additionally, there are listeners on SingalR for when tickets are added to the archive and where they are moved.

When a PUT request is made, will the backend updates the SingalR tasks that remove the ticket from the archive and ass the ticket to the helplist.

7.9.4 API Endpoints for the "Archive" screen

Endpoint: /api/Archive?course=course

Type: GET

Requires auth: Yes

Description: The server receives a request containing the course, and returns the tickets if it is found.

Request Body	Success Response (200)	Error Response (404)

No body	<pre>[{
	"id": 0,
	"nickname": "string",
	"description": "string",
	"room": "string"
	}]</pre>
		Not found

Endpoint: /api/Archive?id=ticketId

Type: PUT

Requires auth: Yes

Description: The server receives a request containing the id of the ticket, and moves it into the helplist if it is found.

Request Body	Success Response (204)	Error Response (404)
No body	No content	Not found

7.10 Create and edit ticket

The editing and creation page in our app is dependent on a shared component named "Ticket". This component receives "onSubmit" and "ticket" as props, where "ticket" is an optional ticket object and "onSubmit" is a void function that takes "ticket" as an argument. The ticket object contains the attributes "name", "room", and "description", which are also text fields in the form and validated for content. If these fields are empty, the input field is highlighted with a red border to indicate an error. The "room" input field has the most logic and retrieves data from the API from a GET call, which returns all rooms that have a lab time at the moment the user accesses the editing or creation of the ticket. The data is also sorted alphabetically. The purpose of creating a shared "Ticket" component is to minimize the likelihood of errors, simplify readability, and save time in case of any future changes.

The logic for ticket creation occurs within the "CreateTicket" component. This component contains a function called "handleSubmit", which is executed when the user clicks the "create ticket" button. The user cannot submit a ticket unless all the fields in the form are fully completed. The "handleSubmit" function sends a POST request to the API with the ticket object obtained from the "onSubmit" function in the "Ticket" component. If the request is successful, the user is navigated to the "Queue" page with the ticket data, otherwise, the user remains on the creation page.

Similar to the "CreateTicket" component, the "EditTicket" component also handles saving. The difference is that the "handleSubmit" function in "EditTicket" makes a PUT request instead of a POST request. "EditTicket" also passes the existing ticket object to the "Ticket" component to obtain default values in the input fields. The ticket object is obtained from the routing that comes from CreateTicket > Queue > EditTicket.

7.10.1 API Endpoints for the "New Ticket" screen

Endpoint: /api/ticket

Type: POST

Description: Server takes in the ticket parameters, adds it to the DB and returns the ticket object with an ID and a placement parameter added.

Requires auth: No

Description: The server takes in a course code and returns all tickets related to that course.

Request Body	Success Response (201)	Error Response (404)

{
"name": "string",
"room": "string",
"description": "string"
}
	{
	"Name": "string",
	"Room": "string",
	"Description": "string",
	"Course": "string",
	"Id": 0,
	"Placement": 0
	}
		Not found

Endpoint: /api/Rooms

Type: GET

Requires auth: No

Description: The server returns an array containing all rooms currently in the DB.

Request Body	Success Response (200)	Error Response (404)
No body	["string"]	Not found

7.10.2 API Endpoints for the "Edit Ticket" screen

Endpoint: /api/Ticket

Type: PUT

Requires auth: No

Description: The server receives the edited ticket, as well as the ticket id. The ticket is then updated in the database.

Request Body	Success Response (200)	Error Response (404)

{
"nickname": "string",
"description": "string",
"room": "string"
}
	{
	"Nickname": "string",
	"Room": "string",
	"Course": "string",
	"Description": "string",
	"Id": 0
	}
		Not found

7.11 Queue

After a user creates a ticket, the data is forwarded via routing. On the Queue page, this information is used to determine the user's position in the queue by counting the number of tickets above them in the helplist. This counter is updated in real-time using SignalR, just like in IKT 201. The Edit button on the page redirects the user to the Edit page with the ticket object. The Cancel button deletes the ticket with a DELETE request to the API and redirects the user to the Create ticket page if the API call is successful.

7.11.1 API Endpoints for the "Queue" screen

Endpoint: /api/Ticket

Type: DELETE

Requires auth: No

Description: The server takes in an ID and deletes the ticket which has that ID. If there is no ticket with the ID, the server responds with 404 Not Found.

Request Body	Success Response (204)	Error Response (404)

<pre>{
  "id": "string"
}</pre>
	No content	Not found

Endpoint: /api/Ticket?id=ticketId

Type: GET

Requires auth: No

Description: The server gets an ID and returns the matching ticket if found. Else it returns 404 Not Found.

Request Body	Success Response (204)	Error Response (404)

No body	<pre>{
	  "nickname": "string",
	  "room": "string",
	  "description": "string",
	  "course": "string",
	  "id": "number",
	  "placement": "number"
	}</pre>
		Not found

7.12 Lab queues

The Lab Queue page is essentially just a list of all the courses from TimeEdit. The courses are retrieved when the user enters the page. A GET request is then made to the API, which returns all the courses from TimeEdit with their course codes. The courses are listed using a flatlist component from React Native. Each item in the list is clickable, and if a user clicks on one, they are navigated to the help list for that course. The course code is sent along in the routing.

7.12.1 API Endpoints for the "Queues" screen

Endpoint: /api/User/Courses

Type: GET

Requires auth: Yes

Description: The server takes in an email and responds with an array containing the courses in which the user with the given email is a student assistant in.

Request Body	Success Response (200)	Error Response (404)
`{ "email": "string" }`	`[ "string" ]`	Not found

Endpoint: /api/Courses/all

Type: GET

Requires auth: Yes

Description: The server receives a request, and returns all courses.

Request Body	Success Response (200)	Error Response (401)
No Body	`[ "string" ]`	Not found

7.12.2 API Endpoints for the "Change Password" screen

Endpoint: /api/User/ChangePassword

Type: PUT

Requires auth: Yes

Description: The server receives a request containing the parameters required to change the password. If they are valid, the password is changed.

Request Body	Success Response (204)	Error Response (401)
`{ "email": "string", "oldPassword": "string", "newPassword": "string" }`	No content	Unauthorized

7.12.3 API Endpoints for the "Forgotten Password" screen

Endpoint: /api/User/ForgottenPassword

Type: POST

Requires auth: No

Description: The server receives the required parameters to send out a new password to the user. The server will then create a link where the user can reset their password, and send it to the requested email address if the user exists in the database. The endpoint returns a 204 No Content response regardless of the outcome, so that intruders cannot check what accounts are available.

Request Body	Success Response (204)	Error Response (404)
`{ "email": "string" }`	No content	Not found

7.13 Pipeline

We started by implementing a pipeline that sets up an environment to drive an emulator in GitLab since we chose to use GitLab as our version controller. We started by setting up predetermined variables that we knew when we started to code such as what version of the android compiler, build tool and SDK tools it can be seen in the pipeline in the appendix. After the predetermined values were set up we chose 11-jdk as the image to use, after some testing and looking through some documentation[31]. After that was working we updated the packages and install others that are needed, sets up the environment accordingly, and tried to start the app in an emulator in the pipeline. It took a few tries before we realized that we needed to set up our own runners since privileged modes were needed to use emulation in the pipeline. After we set up our own runners we still had problems running the app. So we settled for only building the app in the pipeline and letting it pass if it could build without errors.

7.14 Security

7.14.1 Privacy Policy

Publishing the application to Google Play Store requires the existence of a privacy policy. Our privacy policy is very basic, and loosely based on the privacy policy created by Slack [32].

7.14.2 Password Security

Password security is done on two main levels: hard coded within the application itself, and coded into our backend. For the application itself, we have constraints set in place so a user cannot choose a password that does not fit the rules we have set for correct passwords. Furthermore, we have checks done in every API call that makes sure a password is correct and corresponds to the correct account. When we perform an API call we make sure to send all data as an encrypted JSOn object. This adds another layer of password security.

7.14.3 User validation

The validation of the email syntax is handled by the application, to allow for a faster response if the email is written in an incorrect manner. User credentials are validated on the backend, and the password is validated using the built-in validator of ASP.NET.

7.14.4 User Authorization

To authorize a user we use a combination of normal passwords along with bearer tokens. Bearer tokens help identify each user and only last a set period of time. Password checks are in place to make sure only the actual users of an account have access to it.

7.14.5 SQL injection prevention

SQL Injection is an increasing threat, and one of the most common security threats [2]. SQL injection is the process of adding information to the request sent to the server, which may alter or get classified information from the server. The two actions taken on the server side to prevent SQL injections are listed below:

- Parameterized SQL queries: A tool called LINQ to Entities built into Entity Framework handles the communication with the database, and dynamically creates the database queries using parameterized SQL queries [27]. This means that the user input is treated as data and not as executable SQL code.
- Type validation: The server validates all incoming requests and the requested functions will not be run if the incoming object contains the wrong types. An integer will for example not accept string values.

7.14.6 Data transfer security

The server is configured to only return the required data, and nothing else. This means that all data collected from the database is only passed on to the client if it absolutely needs it. It is also possible to add parameters to the request URL, but this is only done if necessary, and if the data is not confidential. Data passing in the link occurs on sending non-sensitive information such as getting data by a specific ID or course. Any sensitive data or larger amounts of information are passed in a JSON object to the server. This object is encrypted as part of the HTTPS connection.

7.15 Backend

7.15.1 Introduction

The application is set up to use the backend system of the HALP .NET web application [21]. The backend system has been modified to implement a REST API, to be able to communicate with the smartphone application. The functionality of the backend system as a whole has not been modified, other than to let the API use its functionality. The backend server is written in C# and uses the built-in ApiController [12]. This controller is incorporated into the ASP.NET framework and thus requires no additional packages to be installed. The main connection between the API and the web server is the database, which holds all information about users, tickets, and courses. The API mostly uses its own methods to communicate with the database, but some of the controllers used for the web server, such as handling the admins and student assistants, have been rewritten as multi-purpose controllers and can be used by both the API and web application.

7.15.2 Implementation

The API is implemented into the existing HALP system with the use of additional controllers and models. The communication endpoints are implemented in their own file, while the authorization endpoints are in another. To make the endpoints capable of receiving JSON objects from the client, a new file containing the models for these objects was created. Each endpoint is placed as a function inside a respective overarching class, to make both the code and Swagger more systematic.

7.15.3 Authorization

Most of the endpoints require authorization to provide the data, but only the endpoints that are not accessible to non-logged-in users. This authorization is done using JWT tokens and is handled internally on the backend, and these tokens last for 30 minutes. This is done to make sure that intruders will lose access after some time, and to handle users that lose their

authorization level. The requests sent from the client to the server to the "/login" endpoints will have to contain the correct credentials, which will then be verified on the back-end system. The server will then return the valid JWT token, which will have to be sent as the "Bearer" token in the header to the endpoints that require authorization. If the user does not exist, or the password is invalid, the server will return a response 401 Unauthorized response. This is returned in both cases so that intruders cannot check if users exist.

7.15.4 Server Environment

The backend system is set up on a private Linux server with a connection to the internet. This connection is provided through port forwarding, and the DNS service DuckDNS [16]. The subdomain added into is required by the certificate authority, to allow the TLS certificate to be signed. The server is set up to run permanently so that the application may be developed whenever the respective team members are available, and the backend will be publicly available for as long as necessary. The TLS certificate will expire on the 16th of August, but will be renewed as needed.

7.15.5 Standard HTTP responses

The ASP.NET Core framework middleware handles the routing of incoming HTTP requests. This middleware also takes care of responding with generic error responses such as 401 Unauthorized and 500 Internal Server Error. Most of the endpoints have a 404 Not Found response, indicating that the element requested does not exist. Because these errors are fairly generic and do not carry an error response body, they are not included in the tables describing the API endpoints.

7.16 Api testing

The testing was done with the python library pytest. The way we handled the testing was by sending in the values that are required and asserting the response code against the expected ones. We made sure to send both correct and incorrect values and asserted that we got the expected error code back. This means that the API is tested to make sure it returns the correct success and error response values.

7.17 Testing the app

The app is manually tested. We have tried the different functionality manually to ensure that it works as it is supposed to. The testing was done by clicking around in the application, and verifying that everything worked. If something did not work as expected, the problem would be analyzed to check if it was a backend issue, an application issue, or both. The problem would then be checked out and fixed by the appropriate team members.

7.18 Google Play Console

The release of the application is done on the Google Play Store, by using the Google Play Console [10]. After setting up an account it was necessary to go through the process of adding the required information necessary for Google to allow for publishing. This process included adding information about what data is stored, an icon, Play Store images, setting up a test group, and lastly uploading a bundled AAB file for Android.

8 Discussion

8.1 Evaluation of the Requirements

When we made the requirements for our application, we had to review the old requirements of what we had made last time. Since the application had the same issues that had to be resolved as last time, we wanted to use our previous requirements. Most of our preview requirements were necessary, but we figured that some requirements were placed in the wrong section of priority of needs and had to be replaced, and we also added some and deleted some as mentioned in section 3.3.1.

Keen eyed readers might notice that it is a must have requirement to register as a user, but logging in as a user is a should have requirement. The logic behind this was that student assistants and admins need to log in, not users. As student assistants and admins technically are users when registering, we left the "register an account" requirement as a must have for users.

When looking back at the requirements, we realize that there are some hick-ups. Must-have requirements 1 and 2 should have been 1 and 1-A. Requirement 3 is repeated two times when they should have been two separate requirements. This almost leads us to skip out on testing one of them. Luckily we picked up on the fact that there were two requirement 3's and set up manual tests for both of them. Only the first requirement 3 has an automated API test, however. As the requirement part was done during the waterfall phase, we decided not to fix the hick-ups and instead discuss them here.

8.1.1 Incompletions of requirements

From the should-have requirements, as seen here: 3.1.2, there was one that we did not fulfill; email verification. The feature was implemented and an email verification link was sent, but it is not functional. We did not have time to diagnose why the link did not work. When we pressed the link, it navigated us to a black screen without any error messages.

From the "can-have" requirements, found here: 3.1.3, there were multiple that we did not implement as they did not contribute to the core functionality of the app. The requirements we skipped were number 14, 15, 16 (A, B, and C), and 17 (A and B). The requirements revolve around implementing useful/interesting features, but not features that are required in order to make the app fully functional. Hence why they are not implemented.

Under subsection 3.2 containing the technical requirements, we specified that the user should have the ability to enable two-factor authentication. We initially intended on implementing this feature, as it is present in the web-based version of the application and is generally a good thing to have - but as time passed, we realized that we did not have enough capacity to implement this due to focusing on the implementation of other more important requirements.

8.2 Features added that had no requirements

We added some features that were not included in the requirements. The added features were added to the setting page, where the users can change the personalia in the profile, and the added darkmode option. The personalia changes were mostly done due to the fact that it felt important to include this, more than some of the "can have" requirements. The darkmode was included due to the user-friendliness requirement.

8.3 Evaluation of the Process

8.3.1 Planning

The main purpose of the planning phase was to figure out which tools we would be using for our project, meeting frequency, and method to use. We discussed the various tools available to us and which of these we wanted to incorporate into our development. After some discussion, we landed on using GitLab for our git repository, Figma for designing our application, Jira for sprints and time logging, and Overleaf for writing our report. We are overall very happy about the tools we decided to use, because we were already familiar with most of the tools, but also got to learn new ones. But it was also hard to plan every tool we needed as the group's experience to make an app was limited. So therefore some of the tools were added as the implementation went on.

8.3.2 Meetings

This time when we worked together, we found that it was easier to communicate than the first time since it was a big group and everyone had worked together before. This time we face the challenge that the meetings became sloppy. This includes both the Waterfall and Scrum meetings, though the Scrum meetings were the part where we can improve the most which we will discuss further down.

After some evaluation, we now have some experience of how important it is to have a clear meeting leader who can guide the group through the agenda of the day, and have a more structured meeting. Though the meetings were not as efficient and structured as they could have been, we benefited from having physical meetings, and the communication was good.

8.3.3 Waterfall

Our waterfall process went overall well. We were happy about how we agreed upon and made our requirements and our design, and the decision went quite fast. As we had our base requirements and some base thoughts on the design, plus some advice from our tutor on the design framework, our Waterfall phase became very efficient. And the transition between the Waterfall and Scrum had no big issues.

8.3.4 Evaluation of the Scrum meetings

When we evaluated the Scrum part of the development, we concluded that we did not follow good Scrum practice. After some discussion and reflection over how well we did use Scrum, we concluded that as the group had worked together before, and with no Scrum Master we became a little too comfortable with each other, and therefore too sloppy to have a well-defined structure for the meetings. Because we stressed about actually working on the application during our physical meetings, the Scrum part got less priority and, as such, was worse.

Improvements that we could have done were dedicating a meeting leader, making a good agenda before every meeting, and always dedicating the first 15-20 minutes to a good Scrum meeting. The part where we did well was that we always kept the meetings we planned, and despite that the meetings were not that great, we always were productive when we were gathered physically.

8.3.5 Evaluation of the Scrum Sprints/tasks

Along the way we figured that we made our Scrum tasks too big, so quite often tasks in one sprint were just transferred to the next. It was not possible to finish the tasks on each sprint, just the subtasks that were made of the bigger tasks were done in the sprints. Following the Scrum method, each task should be so small that each task should be finished at each sprint. Especially since we had 1-week sprints, we did not have enough time to finish them. We

concluded that we either had to have longer sprints next time or, even better; make shorter tasks that are more suitable for the length of the sprints. Also, the last sprint, Sprint_6 went for a longer time, as we came close to delivery, we just used the last sprint to be the final sprint that everything got merged into. The better alternative practice would have been to merge sprint 6 into main, and then make a final sprint with just bug fixing.

8.3.6 Scrum roles

As we experienced that roles were not as useful as we initially thought during the last project, we decided not to have a Scrum master this time. In hindsight, this was probably a mistake as the quality of our scrum meeting obviously worsened. If we had had a dedicated member to oversee the meetings, they would probably have improved quite dramatically.

8.3.7 Version control

Our branch setup worked mostly well, but in some cases, we made some test branches that were not included in the Jira tasks, and these were sometimes merged into the Sprint. This is especially true at the end of the development phase, as we made temporary branches in orde to fix small details that did not require their own tasks in Jira. This can be seen by looking at the git log.

8.3.8 Task delegation

Our division of tasks was overall good, as the members picked the tasks they wanted to implement. We all have a positive learning outcome from the application, as most of us were new to react native. But our biggest mistake was not to delegate more people in the start to do the backend, but we will discuss this under the evaluation of implementation.

8.3.9 Report writing

Our process of writing the report was much better than last time. Under the development of requirements and design, we finished the part in the report before going to the implementation of the application. We were also a lot better to write down notes and important subjects that we discussed about changes, usage of development tools, and why we made our choices. Last time a big part of the report was done during the last couple of days, but this time we were a lot better at filling in the sections as we finished the tasks. Sivert and Markus wrote a lot on the report, but we also divided the more specific parts of the implementations part to the person who was the developer.

8.4 Evaluation of our technologies used

8.4.1 MUI vs react-native-paper

Upon commencing development, we quickly realized that React Native did not support the MUI 5 library. Consequently, we switched to using React Native Paper and the built-in libraries provided by React Native. The main reason for React Native's lack of support for MUI 5 is primarily due to the fundamental differences between web and mobile platforms. MUI 5 is designed for web platforms and relies heavily on technologies such as CSS and HTML, while React Native is a mobile app development framework that includes its own optimized set of UI components tailored for mobile devices.

Although both MUI 5 and React Native share the same underlying technology (React), they are built for different platforms and have distinct design goals. This distinction is not to be taken for granted, and we quickly adapted to using native components provided by React Native Paper instead.

In conclusion, the decision to switch to React Native Paper and React Native's native components was necessary to ensure optimal performance and consistency in our mobile application development. While MUI 5 is an excellent design system and React component library for web platforms, its integration into a mobile app development framework such as React Native may require significant customization and adaptation to ensure a seamless user experience.

8.4.2 Gitlab vs Bitbucket

We had a lot of discussions about what service we should use for our git repository. In our last project, we used Bitbucket as the tool for our git repository. For this project, we discussed whether we should use Bitbucket which we were comfortable with, or try something new. Due to the fact that we wanted to try something new, we discussed using Gitlab as the host for our repository. We had already been introduced to Gitlab in another subject and saw this as an opportunity to dive deeper into it. After some debate between Bitbucket and GitLab, we eventually decided on using GitLab for this project. The pros for this were that it was somewhat familiar, but also an application that we still could learn about. Gitlab also supports a pipeline that could be run to test if the mobile application could be built.

8.5 Evaluation of our Design

In our process of developing the design, we wanted to use the same color scheme as what we had chosen for our website. After some discussion and some guidance from our teacher, we came to an agreement that some of the colors weren't that well thought through and decided to follow a color system that is recommended by Figma and material.io [19].

For the design of the rest of the application, we also decided to follow the guidance of Material design, of where we wanted to implement our buttons, menu, sizes, borders, etc. This is also mainly due to our non-functional requirement(see subsection 3.3) that we wanted that our application to be user-friendly.

After we were finished with our design sprint, we figured out that we missed the course screen that we needed to have. We quickly made the design and the screen at the beginning of sprint 3.

We had to discuss what the helplist would look like. First, we discussed if we wanted an expandable text box vs a popup box, and finally decided that it should be an expandable textbox, because of its easy view and ease to use. After we had to decide the implementation of the archiving actions and how it should work. It was either a swipe or pressing a button, and we should have a confirmation action happen when it was pressed. We decided on a button with no confirmation, because a button is visible and easy to use, and the confirmation action was dropped due to the function in the archive where you can easily unarchive the tasks that have been finished.

8.5.1 Finished Design

Our finished design ended up being almost identical to our original Figma design with just a few tweaks. After some discussions, we ended up changing the various logo designs of our navigation bar to fit better with the rest of the design. We also ended up changing the size of the description text input bar in the create and edit ticket screens from a big box to a dynamically expanding one. This is because a big text box was not practical nor did it look good on a mobile application. Lastly, we changed the navigation bar for a student assistant so they also have the ability to get to the create a ticket screen if they want. This was done by replacing the archive button with the create a ticket button in the navigation bar and adding the button for archive within the helplist screen.

8.6 Design differences

8.6.1 Login

The login has an extra text field and button. The reason for this is that a privacy policy was required by the google play console to upload the phone app.

8.6.2 Create new ticket/edit ticket

There is a difference in the description text input, where we made the description box smaller, and expands when typing. This was done because it looked cleaner and we did not need to fill the whole screen for the description.

8.6.3 Labqueues for admins and student assistants

We made a box around the current classes that are available and button colors around the classes, so it is easier to separate and choose the right class.

8.6.4 Helplist / archive

We added the current course as text to the top of the helplist and archive screens to make it more apparent which courses helplist you are looking at. We also added rooms to the list because a single course can have labs in multiple rooms. The checkboxes were changed to the archive icon due to the fact that the need to check a box is not necessary. We added a little archive icon at the top right side of the screen which redirects to the archive.

Some further work that could be done here is implementing some sort of filtering option for the rooms.

8.6.5 Settings

We added our logo in Settings screen because every other screen had the logo. We also added a button which lead to the privacy policy page.

8.6.6 Setting for admins

As we mentioned in the design part, we did not know exactly what buttons we wanted to have here, so some of the buttons differs from the design part.

8.6.7 Settings: Timeedit

For the Add TimeEdit link, we found it much cleaner to have a popup window for the link add, and therefore added it to screen, and not just a text-input box. The darkmode colors of the list we have changed to match the other lists in the application, so the dark color matches the background, and then another lighter blue color on the light list item.

8.6.8 Settings: Roles

When assigning roles to other users as an Admin, we switched from the modal design to a checkbox approach in order to have more nesting - so that the admin can choose between admin or student assistants in a course and then click their respective checkboxes.

8.6.9 Settings: Change password

We changed the design of the screen and added a textbox where the user also has to type in their current password because we needed the added security here to be able to change the password.

8.7 Google universal design review

Google universal design review which is a tool built into the Google Play Console, gives feedback about errors, small errors, and warnings. According to their review of our app, we have some fields and buttons too small and they are not happy with the color contrast on multiple screens. We have seen over and tried to correct most of those. They also complain that we do not have labels that can be read by the screen on our items. We have evaluated that this could be implemented as future work, to allow for better accessibility.

8.8 Evaluation of the implementation

8.8.1 Structure of the code

We were happy about how the implementation of the structure was done. We got pretty good unified code, that was much alike in almost every part even though we were six people working on the codebase. We felt that it was well organized and easy to navigate through. Some of our files were a little long and could have been split up more, but in the grand scheme of things, we were satisfied with how the result became.

8.8.2 Backend implementation

The work on our backend was a work task that we could have done much better. The backend was implemented almost only by Nikolai and was done continuously throughout the development of our application. The result of implementing the backend on the go by one person, instead of everyone participating and finishing the backend at the start, resulted in a bottleneck at times. When looking back at this approach, we see room for improvement. Since we already had the backend, and at least to a certain point could have known what was needed in our mobile application, we could have finished the backend before beginning on the logic of the application. We also experienced that a single person doing the backend was a single point of failure if Nikolai was not attending, and no one knew the code to fix the problems by themselves. In addition, sometimes it is good to have more people to think of solutions.

8.8.3 Design implementation

When we started to implement the design of our app, we all did different things from the start. Especially the implementation of dark mode. We started off with just passing the boolean state of dark and light mode, but Charlotte decided we had to go for a cleaner method, and we created color themes and global hooks with react-redux to keep track of the state. This was a really clean way to do it, but as we had already implemented 5 screens, it took a while to implement it to every screen and to keep up with the other updates that kept coming. The final result was really good, but we learned that we should agree on the design implementation method before everyone starts coding next time.

8.9 Accessibility

We are aware that users that are visually impaired can have problems viewing our application because a lot of the colors in our application are blue, and also we know that the Google Play Console complained that our app doesn't support the Talkback function. This was a priority that was not on our agenda but would be a nice future work concept to implement.

8.10 Evaluation of our Product

8.10.1 Queue bugs

We know for a certainty that our queue doesn't work as properly as we wanted. The queue remembers the queue state when changing users. This was tested a little late, and we did not have the time for fixing it.

8.10.2 Forgotten password bug

We also know that the forgotten password doesn't work, because its something wrong in the backend from the last project, that we did not figure out.

8.10.3 Backend

We were very satisfied with how we managed to connect to our backend from the dotnet project from last year. The backend is constantly running which was necessary in order to implement all the features we wanted to, whenever we wanted to. One of the most challenging features was to make the app update live. We used SignalR in order to solve this. The live update part was a struggle and we discuss it further in section 8.16.3.

8.10.4 Design

We were very satisfied with our design. We made a lot of work and effort into the design of the app. It was a success using Figma as a design creator to make our template. The only thing we did not know until it was too late was that we could have converted the design from Figma to our app. The colors were well thought through, following the color scheme of the material design, and also the universal sizing was used by following material design templates.

8.10.5 Student assistant navigation bar

In our initial student assistant navigation bar design, we included three items/buttons: helplist, archive, and settings. However, after discussing the student assistant role we came to the conclusion that student assistants should also be able to create tickets as they are still students with their own courses. We discussed whether or not we should add more items to the navigation bar or change the ones we already have. We ended up agreeing to change the student assistant navigation bar to include: create ticket, helplist, and settings. The archive could not be accessed from the helplist screen. This design change gave student assistants the added feature of creating tickets without removing any of the current features.

8.11 Evaluation of Security

8.11.1 Privacy

Privacy is an important aspect of our application and we take the protection of our user's information seriously. We have looked around at laws, especially Google's requirements for having an app in the play store. From this information, we have implemented measures to ensure data privacy and security. We only collect the minimum necessary information that we require from our users. This information includes their email, a nickname (which might contain their name if they choose to enter it), a password, and the questions they ask when they submit tickets. Furthermore, we are committed to using this information solely for the purpose of providing a seamless and efficient ticketing experience. We do not disclose or share our user's personal data with any third parties, except as required by law.

8.11.2 Security in the application

We have implemented most of our application functionality with security in mind. There might be holes in the security we have overlooked but this is problems that can be dealt with in the future. For our API calls, we use HTTPS and bearer tokens for added security. As most of our backend is done using API calls this is a big part of the security in our application. We have also implemented security measures for each individual screen. For example, in screens like register, there are hard-coded constraints on passwords, empty fields, and wrong formatted emails.

8.11.3 SQL Injection

The backend API has no implemented security measures regarding SQL injection, other than what is supplied in ASP.NET. This means that the server to some degree is prone to SQL injection attacks. We have assessed the situation, and come to the conclusion that this aspect of the project should be perfected in the future, as LINQ to Entities provided by Entity Framework handles this adequately. The prevention of SQL injection is an important aspect of security prevention, but with other pressing matters, we have had to down-prioritize this feature.

8.12 Language

We also thought implementing language choices for more user-friendliness could be a nice idea. This would mainly be a way for users to swap the language of the applications from English to for example Norwegian. As the application is mainly focused on students attending the University of Agder we would first implement support for the Norwegian language. Support for multiple languages is something we could implement in the future.

8.13 Methods of authentication

oAuth2 is a standard designed used for authorization for accessing resources hosted by a server [40]. The HALP system handles authorization in a very similar manner to oAuth2, but does not directly implement it. The introduction of oAuth to our system could have improved our security and ease of implementation of secure authorization. We did not realize that the standard existed until late in the project, and as our approach with JWT tokens work in the same manner. This led us to keep our existing approach, and not spend time on changing the authorization implementation. The application does not, however, handle refreshing of the JWT token, something that could be implemented in future work with the use of refresh tokens.

8.14 Evaluation of testing

After some discussion at the start, we did not all agree on how we wanted to test our application. The two different choices were to either test the application or the backend. The optimal solution would of course be to test both.

We chose to make automatic tests for the API since we concluded that ensuring the security of that was the most important to keep the data safe. Compared to making an automatic test that checks the size and color of the application. The testing of the application was done manually, and we wrote down tests for the "must-have" requirements.

Following our Scrum method, we also were too slow with our testing. The only testing that was done after each Scrum task was just manual testing, with no documentation of the test. It became easier to test when the automatic testing was in place, but overall we were too slow with both documentation and the tests.

8.15 Evaluation of our Backend work

As discussed earlier, work on the backend was done mainly by one person. However, as all the team members discussed their backend needs and it was continously implemented, the backend has, at least conceptually, been integrated as a team effort. Towards the end of the development process, we had a meeting where Nikolai went over the implementation of the API endpoints on the server-side so each member would have a proper understanding of its functionality and could contribute more in the technical discussions. This was crucial in the integration of SignalR which required significans modifications to the server.

8.16 API Problems

8.16.1 Implementation of the API

The basic implementation of the API was pretty straightforward and consisted of including support for it in the Program.cs file, as well as creating the controllers and models subsubsection 7.15.2. However, adding technologies to the Program.cs file is not always as easy as it sounds. When adding to the file, it is very important that some lines come in the correct order, it is for example very important that the command "app.UseAuthentication();" comes before the line "app.UseAuthorization();", or else the program will simply not start. The addition of the API has therefore led to the original HALP application not being fully operational anymore. It still displays and acts as normal, but it is not possible to log in anymore, and thus the system has lost the admin and student assistant capabilities, as well as the user losing the ability to log into their account. But as the goal of this project was to create an application and connecting it to the old backend using an API, and as this was accomplishable without the old system being fully operational, fixing this error was not prioritized. However, we do have some idea of how to fix this error in an elegant way, see subsubsection 8.16.2.

8.16.2 The ideal server application

Today, both the web application and API is run by the same application, which implies a lot of stress on a single point of failure. We would argue that splitting the application into two systems with connecting dll files would be a more optimal approach. These dll files are libraries containing code that can be utilized by multiple applications running on the .NET framework [14]. This would allow the applications to use the same functions while still being separate systems, and would solve multiple problems:

- Client handling: A singular system would not be responsible for handling clients in both the web application and the API.
- Downtime: If one of the systems goes down either unexpectedly or for maintenance, this would not affect the other system.
- Updates and development: It would be a lot easier to develop and update one of the systems, without having to worry about accidentally breaking the other.
- Scaling: If one system is more used than the other, it would be possible to scale this system independently of the other.

This approach would however take a lot more time, and as all backend functionality is in place for this project, the split approach will instead be implemented in future work.

8.16.3 Problems with live system updates

Live updates are an important aspect of the HALP application, as it would allow the student assistant to get updates in the helplist on the fly, and the user to get immediate updates on their position in the queue. This, however, was not an easy task. We started early with researching the possibilities, and chose to try and implement the SignalR framework, as this was the system we had previously used for live updates in the web application [21].

We tried long and hard to get SignalR to function, but to no avail. The application refused to connect to the backend when testing locally. We then went back to our drawing board and started researching server-sent events (SSE) subsubsection 5.9.2. After performing new, time-consuming tests using SSE, and having the application fail on receiving any updates while a basic JavaScript connection had no problems, we realized that there are problems in the core of React Native regarding how HTTP is buffered. This meant that the data would not arrive in our application until the connection was closed.

Closing an SSE connection to receive data is in short terms the same as utilizing long polling subsubsection 5.9.3. This meant that we proceeded to test with this technology instead. Long polling needed extensive client handling functionalities on the server side, to make sure the correct clients got the update, and they also had to get the update even if it happened while they were reconnecting. After creating these extensive functions and not having them work correctly, we started looking at SignalR again.

We eventually got SignalR working when trying to connect to the remote HALP server, instead of locally. This meant that the ideal method of getting live updates was there, but everything had to be tested "in-production" (while developing the system we still considered the remote server part of the development environment), and this was not an ideal approach. It was, however, our best bet to get the live updates going. In retrospect, we have a theory that the SignalR connection would not connect because of either network issues or TLS certificate issues.

8.17 Future work

8.17.1 Implementation for IOS

As our exam project was only required to be distributed on the Google Play store that was our main focus. The application was developed on a cross-platform, but as we needed a Macintosh to compile our application, and the fee is pretty expensive, we decided that this was not a realistic option for our program at this time but rather a possibility that could be implemented in the future.

8.17.2 More functionality for registered users

In the future, we would like to implement more functionality for users who have made an account and logged in to our application. At the moment the only functionality a logged-in user gets over an anonymous user is that their name will be automatically assigned to their ticket so they don't have to manually enter it. Examples of functionalities we could implement are having a history of their previously submitted tickets, saving their favorite courses, and maybe being able to access statistics.

8.17.3 Submitting to Google Play

The application is currently only closed and internal testing, and is thus not publicly available. Ideally, we would want to publish the application to the Play Store for everyone to download. This would require more work in the publishing field, we have used the Google Play Console to make sure that the application complies with the guidelines thus far.

8.17.4 Digital lab help

A lot of students choose to meet up for lab hours digitally through external services like Discord. In the future, we would like to make it possible for students to use our application digitally. This would link our application through a service like Discord so when it is a digital student's time in the queue they will get helped digitally. We would like the digital and physical students to be placed in the same queue, and be helped accordingly. This functionality would make our application useful for students who need to be home because of for example sickness.

8.17.5 API

We talked about changing from jwt token to a role-based jwt token to heighten the security. Currently, it is either open for those that should be open or locked behind a jwt token, but there is no difference between a user's or an admin's jwt token. We would like to add in so the jwt tokens are different, so we could look at the API after what they should have access to instead of just if their logged in or not.

8.17.6 Testing

As of now, our automatic testing is limited to the backend functionality. We would have liked to implement frontend testing in order to check for graphical consistency in sizes, colors, etc. We also discussed end-to-end testning, but we ended up not prioritizing it.

8.17.7 Pipeline

We have talked about the possibility of automatically exporting a new version of the app to the play store on the finished building in the pipeline, but we have not looked into it since it was not a priority.

8.18 Security

In the future, the HALP system should be upgraded in the security aspect with the features mentioned earlier in the report. This includes security upgrades on both the frontent and backend side of the system. The features should include upgrades such as proper validation on the backend to avoid SQL injection, and the removal of the "discordValidation" endpoint to make it impossible to check which users have an account. To make sure absolutely all holes are filled, it is also recommended that this update includes a new security assessment of the system, to find all these missing holes, and use the lates new security technology.

9 Conclusion

The objective of our project, named HALP, was to develop an enhanced system for managing the lab queue at our university. To accomplish this, we integrated the backend developed in our previous project with the mobile application and implemented the essential features. Our foremost aim with the HALP mobile application was to ensure it was user-friendly and well-suited for its intended purpose. We are proud of the final product we have produced, as it has a clean and intuitive interface, imposes no specific requirements on users, and should prove easily operable by both teachers and student assistants.

However, we acknowledge that HALP is not yet ready for production due to certain known bugs. Despite being aware of these imperfections, we are pleased with the outcome achieved within the constraints of the given time frame.

In summary, the HALP project has successfully achieved its primary objectives, and while some aspects of the application may not function flawlessly, we remain content with the overall result.

References

[1] URL: https://hotpot.ai/art-generator.

[2] *.NET SQL Injection Guide: Examples and Prevention*. [Online; accessed 18. May 2023]. May 2023. URL: https://www.stackhawk.com/blog/net-sql-injection-guide-examples-and-prevention.

[3] Internet Engineering Task Force (IETF). *The WebSocket Protocol*. URL: https://datatracker.ietf.org/doc/html/rfc6455.

[4] *About Certbot*. [Online; accessed 11. May 2023]. May 2023. URL: https://certbot.eff.org/pages/about.

[5] Dan Abramov. *Getting Started with Redux*. 2015-2023. URL: https://redux.js.org/introduction/getting-started.

[6] Karl Wiegers et. al. *Software requirements Third edition*. 2013.

[7] Atlassian. *What is Git | Atlassian Git Tutorial*. [Online; accessed 11. May 2023]. May 2023. URL: https://www.atlassian.com/git/tutorials/what-is-git.

[8] auth0. *Introduction to JSON Web Tokens*. URL: https://jwt.io/introduction.

[9] BillWagner. *A tour of C# - Overview*. [Online; accessed 15. May 2023]. May 2023. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp.

[10] Google Play Console. *Google Play Console*. [Online; accessed 16. May 2023]. May 2023. URL: https://play.google.com/console/about.

[11] Mozilla Corporation. *Using server-sent events*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events.

[12] *Create web APIs with ASP.NET Core*. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/?WT.mc_id=dotnet-35129-website&view=aspnetcore-7.0.

[13] *DB Browser for SQLite*. [Online; accessed 11. May 2023]. May 2023. URL: https://sqlitebrowser.org.

[14] Deland-Han. *Dynamic link library (DLL) - Windows Client*. [Online; accessed 16. May 2023]. May 2023. URL: https://learn.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library.

[15] *Discord | Your Place to Talk and Hang Out*. [Online; accessed 18. May 2023]. May 2023. URL: https://discord.com.

[16] *Duck DNS*. URL: https://www.duckdns.org/about.jsp.

[17] *Free Design Tool for Websites, Graphic Design and More | Figma*. [Online; accessed 13. Mar. 2023]. Mar. 2023. URL: https://www.figma.com/design.

[18] *Getting Started - Let's Encrypt*. URL: https://letsencrypt.org/getting-started/.

[19] Google. *Material Design*. URL: https://m3.material.io/.

[20] *Google Play | Google for Developers - Software Development Guides, Tools & More | Google Developers*. [Online; accessed 11. May 2023]. Apr. 2023. URL: https://developers.google.com/learn/topics/google-play.

[21] *Help assistant lab program, HALP*. [Article; accessed 13. Mar. 2023]. May 2022.

[22] IBM. *What is a REST API?*

[23] Mike Isaac. *Google Unwraps Ice Cream Sandwich, the Next-Generation Android OS*. URL: https://www.wired.com/2011/10/android-ice-cream-sandwich-3/.

[24] ISRG. *About Internet Security Research Group*. URL: https://www.abetterinternet.org/about/.

[25] *Long polling*. URL: https://javascript.info/long-polling.

[26] ManageEngine. *Help desk software | ManageEngine ServiceDesk Plus*. [Online; accessed 17. Mar. 2023]. Nov. 2022. URL: https://www.manageengine.com/products/service-desk.

[27] mcleblanc. *LINQ to Entities - ADO.NET*. [Online; accessed 18. May 2023]. May 2023. URL: https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/ef/language-reference/linq-to-entities.

[28] *React*. URL: https://reactjs.org/.

[29] *React Native*. URL: https://reactnative.dev/.

[30] *Real-time ASP.NET with SignalR*. [Online; accessed 10. Mar. 2023]. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/signalr.

[31] *Setting up the development environment · React Native*. [Online; accessed 15. May 2023]. May 2023. URL: https://reactnative.dev/docs/environment-setup?os=linux.

[32] Slack. "Privacy policy | Legal". In: *Slack* (Jan. 2023). URL: https://slack.com/intl/en-gb/trust/privacy/privacy-policy#collect.

[33] *Swagger*. URL: https://swagger.io/solutions/api-documentation/.

[34] *TeamViewer – The Remote Connectivity Software*. [Online; accessed 11. May 2023]. May 2023. URL: https://www.teamviewer.com/en.

[35] *The DevSecOps Platform*. [Online; accessed 13. Mar. 2023]. Mar. 2023. URL: https://about.gitlab.com.

[36] *The DevSecOps Platform*. [Online; accessed 13. Mar. 2023]. Mar. 2023. URL: The%20DevSecOps%20Platform.

[37] Timeedit. *Modern scheduling for higher education*. URL: https://www.timeedit.com/higher-ed/higher-education.

[38] *What is .NET? An open-source developer platform*. [Online; accessed 15. May 2023]. May 2023. URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet.

[39] *What Is GitHub? A Beginner's Introduction to GitHub*. [Online; accessed 11. May 2023]. Dec. 2022. URL: https://kinsta.com/knowledgebase/what-is-github.

[40] *What is OAuth 2.0 and what does it do for you? - Auth0*. [Online; accessed 19. May 2023]. May 2023. URL: https://auth0.com/intro-to-iam/what-is-oauth-2.

[41] *What is TLS (Transport Layer Security)?* URL: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/.

[42] *What is TypeScript?* URL: https://www.typescriptlang.org/.