Help assistant lab program, HALP

Group 8

By

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

IKT201

Internet Services

Supervised by Christian Auby

Faculty of technologies and science

University of Agder

Grimstad, December 2022

Abstract

Our help assistant lab program, HALP is a webapplication provided to make old queue systems used in lab classes more organized and efficient by creating a better alternative. As there are multiple products providing a similar service to ours, we wanted to stand out by incorporating Time-Edit into our ticket system. This gives our program the ability to fetch the courses, times, and classrooms for a specific lab hour.

This program is a web and mobile-friendly application that can be used for students that are visiting labs, student assistants, and teachers. The web page was developed with the ASP.NET framework, and SignalR. As our project came to an end, our product was not completed. It contains the most important features, but there are a variety of functionalities we would like to incorporate in the future. The main functionality of HALP works as intended, and it performs the action that we wanted, like writing tickets, setting up a queue, and resolving tickets. The report covers the development process, design and choices.

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 signal the students about their responsibility, and the consequences cheating would entail. Lack of such a signal 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 tin 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 collected grade.	No

Publishing agreement

Full power to 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 assignment not publicly accessible.	Yes

Preface

This report documents the work made on our final project in the course IKT201-G. We decided to make a student assistant lab program because it is a thing we would actually benefit from in our future lab classes. The project also builds on our skills learned from the magnitude of courses we have had so far in our studies. By the end of the project, we have learned a lot about web application development, object-oriented programming, and databases. During the project, we have consistently used Jira for scheduling and logging, Bitbucket for easier development collaboration 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 tutor and counselor Christian Auby for great guidance, availability, and support throughout our project.

Grimstad

14. December 2022

Charlotte Thorjussen, Markus Hagli, Nikolai Eidsheim, Ole-Johan Øvreås, Sivert Espeland Husebø, and Sondre Horpestad

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Product vision . . . 1
1.3 Similar products . . . 1

2 Process **3**
2.1 Planning . . . 3
2.2 Meetings . . . 3
2.3 Scrum . . . 3
2.4 Group roles . . . 3
2.5 Sprints . . . 4
2.6 Version control . . . 4
2.7 Usage of Discord bot . . . 4
2.8 Design applications . . . 4
2.8.1 User Interface . . . 4
2.8.2 Flow . . . 5

3 Requirements **6**
3.1 Functional requirements . . . 6
3.1.1 Must have . . . 6
3.1.2 Should have . . . 7
3.1.3 can have . . . 7
3.1.4 Wont have . . . 7
3.2 Technical requirements . . . 7

4 User interface design **8**
4.1 Registration and login methods . . . 8
4.1.1 The universal login screen . . . 8
4.1.2 Register new user . . . 9
4.2 Student view . . . 10
4.2.1 Create new ticket, user logged in . . . 10
4.2.2 Create a new ticket, anonymous user . . . 10
4.2.3 The user settings page . . . 11
4.2.4 The Ticket queue . . . 12
4.3 The Student Assistants view . . . 12
4.3.1 The Helplist . . . 12
4.3.2 Archive . . . 13
4.4 Admin view . . . 13
4.4.1 The admin settings page . . . 13
4.4.2 Time-Edit links . . . 14
4.4.3 The Roles settings . . . 14
4.4.4 Statistics . . . 15
4.5 Other pages . . . 16
4.5.1 Change password . . . 16
4.5.2 Reset password . . . 16
4.5.3 Generic error . . . 17
4.5.4 Error 401: Unauthorized . . . 17
4.6 Application flows . . . 18

5 Technical background **19**
5.1 Tools and technology used . . . 19

5.1.1 Jira 19
5.1.2 Bitbucket and Git 19
5.2 Frameworks 19
5.2.1 ASP.NET 19
5.2.2 Identity Framework 19
5.2.3 Entity Framework 20
5.2.4 Blazor 20
5.3 Libraries 20
5.3.1 SignalR 20
5.3.2 Bootstrap 20
5.3.3 OAuth Discord 21
5.3.4 QR Codes for JavaScript 21
5.4 Languages 21
5.4.1 C# 21
5.4.2 Javascript 21
5.4.3 HTML 21
5.4.4 CSS 21

6 Product **22**
6.1 User interface design 22
6.2 Registration and login methods 22
6.2.1 Register new user 22
6.2.2 The universal login screen 23
6.3 Student view 24
6.3.1 Create a new ticket, anonymous user 24
6.3.2 Create a new ticket, user logged in 24
6.3.3 Ticket queue 24
6.3.4 Edit ticket 25
6.3.5 The settings page 26
6.4 Student Assistant's view 27
6.4.1 Overview of lab queues 27
6.4.2 Help-list for courses 27
6.4.3 Archive 27
6.5 The Admin view 28
6.5.1 Pages 28
6.5.2 Settings 28
6.6 Other pages 30
6.6.1 Change password 30
6.6.2 Forgotten password 30
6.6.3 External login 31
6.6.4 Two factor authentication 32

7 Implementation **33**
7.1 Database design 33
7.2 Web design 33
7.3 Application flow 33
7.4 ASP.NET scaffolding 34
7.5 Email-verification 34
7.6 Discord-login 34
7.7 Settings 34
7.7.1 Retrieval of TimeEdit data 34
7.8 Lists 35

7.9 Help list 35
7.10 Archive list 35
7.11 Tickets 35
7.11.1 Create ticket 35
7.11.2 Edit ticket 36
7.12 Queue 36
7.13 Application security 36
7.13.1 User Roles 36
7.13.2 Password handling 36
7.13.3 Two-factor authentication 37

8 Testing and validation **38**
8.1 Tests regarding "must have" requirements 38

9 Discussion **42**
9.1 Process 42
9.1.1 Planning 42
9.1.2 Git 42
9.1.3 Challenges in the process 42
9.2 Product 42
9.2.1 Achievements 42
9.2.2 Improvements 42
9.2.3 Incompletions of requirements 43
9.2.4 Finished Design 43
9.2.5 Design Incompletions 44
9.3 Design anomalies 45
9.4 The mobile application 45
9.5 Application Integration 45
9.6 Implementation 45
9.6.1 Tickets 45
9.6.2 Queue 46
9.7 Satisfactions 46
9.7.1 Report writing 46
9.8 Challenges 46
9.8.1 Approach for retrieval of TimeEdit data 46
9.8.2 Problems with git rebase 47
9.8.3 Database design 47
9.8.4 HTTP Requests vs SignalR 47
9.8.5 External logins 47
9.9 Future work 47
9.9.1 Docker support 47
9.9.2 More functionality for registered users 47
9.9.3 Mobile application 48
9.9.4 Digital lab help 48

10 Conclusion **49**

List of Figures

1 The universal login screen 8
2 Page for creating new user 9
3 Page for creating a new ticket, if you are logged in 10
4 Page for an anonymous user to create a new ticket 10
5 The user settings 11
6 the page for showing the user what number he / she is in the queue 12
7 UI10: The lab queue showing the tickets 12
8 UI14: The list of archived tickets 13
9 The admin settings view 13
10 The view where the admin can add or remove courses 14
11 The view where the admin can manage user roles 14
12 The admin statistics view 15
13 Page for changing the password 16
14 UI4: Page for getting a new password if you have forgotten it 16
15 UI13.1: The generic error page 17
16 UI13.2: The error page of error code 401: Unauthorized 17
17 Flowchart showing the initial flow design of the system 18
18 Register user 22
19 Mail Confirmation 22
20 Login 23
21 Create ticket 24
22 Ticket queue 25
23 Edit ticket 25
24 Settings view for users 26
25 Overview labqueue for a course 27
26 List over helplist courses 27
27 Page for the archived tickets 28
28 Settings view for Timeedit links 29
29 Role settings view admin 29
30 Page for letting the user change their password 30
31 Page for users having forgotten their password 30
32 Page for users to reset their password 31
33 Page for external login with discord 31
34 Page for set up the two factor authentication 32
35 Database design 33
36 Flowchart showing the current flow design of the system 34

1 Introduction

In this section we talk about the background why we decided to make this product, our vision for the product, and discuss similar products that already exist.

1.1 Background

The foundation behind our idea for creating a ticket system for lab hours is based on the fact that the currently used system made with simple spreadsheets, is unorganized, unintuitive, and generally difficult to use. Making a program that addresses these issues would generally help us in our day to day lives as students, and therefore actually benefit us. The idea of making a program that is as simple as using spreadsheets but also incorporates more structure, and features resonated with us. Our program would also give the student assistants a good overview of the tasks that they should complete. As we searched for apps and similar products we found a lot of ticket-based programs, but those products are mainly focused on digital help and emails. These services are also too big for what the intent of what our system needs.

1.2 Product vision

Our vision is to create an actual program dedicated to lab hours, rather than just relying on the use of excel sheets. We want to give a structured overview over the students that require assistance. Also, we want to create a platform that has more features incorporated rather than just questions. We want to implement multiple quality-of-life features like allowing for both anonymous and logged-in users, having a priority queuing system, and having an easy way to keep track of who needs help, and when. Our product will be aimed at educational institutions with needs for an organized questionnaire system, that is more user-friendly, organized, and all-around more efficient.

- Primary users: Educational institutions have teachers and teacher assistants who help students
- Secondary users: Students of the primary users.

We want to make a website that works both for primary and secondary users, but with different views. The student assistants will have access to the website for setting up the lab hours, seeing the incoming tickets in the queue system, tagging tickets as completed, and go to the next ticket in line. The students/users can use the the website to create new tickets. After creating a ticket they will be put in a queue, and will be provided with an overview of their position in the queue. Students will have the ability to create an account on the website. Logged in students can choose favorite subjects, save a discord username to easier connect with discord, and have access to their help history. Users who are not logged in still have the possibility to create/edit/delete a ticket.

1.3 Similar products

After looking around on the internet, we have found multiple systems that are similar to what we are trying to produce, one of them being Zendesk [19]. The most common ticket systems we have found are different kinds of "help desks" that are digital platforms with multiple features like chat, messages, phone, mail, etc. These features have similarities to what we are envisioning, but they are generally too big, and contain unnecessary functionality for our intent of work. The biggest difference between our product and other services, is that we are going to synchronize our program with Time-Edit. We have not found any other services that has this functionality. Time-edit is the service that most of the Universities in Norway are using to schedule their courses and rooms, and it is also used in some international universities[18]. This makes our product more of a niche product, which fits better as it is more tailored for what our institution needs.

2 Process

In this chapter we will dive deeper into the process and the methods we used to develop our product. We will take a look at every step of the process from brainstorming to planning and how we as a group managed to overcome the challenges we encountered.

2.1 Planning

Since almost everyone except Charlotte in the group were relatively new to working on a larger project and developing a product in a bigger team, we knew there would be some challenges with planning, and communication. With that in mind we also knew that we had to come to an agreement on which parts were the most important for our product, how we should design it and how we should split the tasks evenly. We used Jira for tracking working time, communicated via Discord and held physical meetings.

2.2 Meetings

We knew that we had to do regular physical meetings every week to get an estimate on how we were doing, and figure out what the next step should be. We discussed about whether we would have one or two physical meetings a week, and eventually decided on having two meetings, one on Mondays and one on Thursdays at 10:00 O'clock.

The first 5-10 minutes of every meeting was an evaluation of the current, or last weeks sprint. We decided that Mondays was the day we created a new sprint which took us about 15 minutes. On Thursdays we had an update and evaluation of the tasks done in the current sprint. After the evaluation we worked together between 1-4 hours on the different tasks that were assigned. The physical meetings provided a good way for the team to have a dedicated time set aside for the project where everyone was gathered. Here we could also discuss problems that we individually had and look for solutions. We could also have discussions on how the front-end and back-end was supposed to look like, in a way everyone could agree.

2.3 Scrum

For this project, we were recommended to use the Scrum development method, which we decided to use. This is the Agile development method, which is one of the most used agile methods when developing software[1]. This method has an iterative and incremental approach and lets us work in short sprint cycles, which gives us the possibility to be flexible to changes and adaptations during our development process[1]. Scrum also gives us a framework with common rules and roles, which makes it easier to split the tasks among the various group members.

2.4 Group roles

When we decided to go for the Scrum method, we tried to set some defined Scrum roles to build a Scrum team. Since Charlotte was the one with the most experience in Scrum, she was dedicated as the Scrum master. She did most of the work with the backlogs and sprints and took lead when we had the sprint meetings. Since this was a school project we had no product owner and had to fill those requirements ourselves, we decided against having other roles because everyone wanted to learn a little bit of everything that we implemented.

Instead of setting roles, we distributed 100% responsibility on different part of the system.

Nikolai had responsibility for the design, Markus had database, Sivert had queue and tickets, Ole had login, Charlotte had SignalR communication and Sondre had testing responsibility.

2.5 Sprints

The process of developing our product was divided into short cycles, known as "sprints". First we wrote down the majority of user tasks into our backlog with priorities. We then added them into the sprints as we continued with the development. These sprints can vary between one week to a month, with different tasks which we assigned to different team members [1]. As the project is over a short time period, we aimed at going for one sprint per week. The sprints were a great way to track our progress and pushed us to get the different tasks in our project done on time. Sprints are an easy way to document what has been done, and who has done it.

We decided on writing one sprint per week with various tasks to be done. On Thursday each week we evaluated them, and on the following Mondays we went over them. It wasn't always easy to know how much time each task would take, so some of the tasks ended up going over multiple sprints. These were typically the report writing sprints because we filled in the report as we went by.

2.6 Version control

Version control is the practice of tracking and managing changes to the software code [2]. For this, we used Git and Bitbucket to see our changes and updates. We had mainly two types of branches that we worked on.

To be able to work independently, we made a branch for each Jira task. This branch would contain the project name, CHAN, and the associated Jira ticket number. Before merging to main, the branches had to be approved through a pull request. Pull requests are important for checking any error and get input from others.

The main branch was the branch where everything should be tested and working, and would produce our working website. This was only updated when the task branches were fully completed, and had the main branch merged into it. If a merge into the main would happen, we had set a bottom limit to at least two reviewers(members of the group) to approve it before the task branch could be merged into the main. The main branch is blocked if there is any merge conflicts.

2.7 Usage of Discord bot

To keep everyone up to date on the project, and keep track of who was working on what, we created a Discord bot with a connection to Bit-bucket. This bot would notify us every time a change happened in a project, either in the form of a push to a branch or a pull request. Every member of our team would then have the ability to quickly review the pull request to allow it to quickly be merged into the main branch.

2.8 Design applications

2.8.1 User Interface

The user interface was initially designed in Microsoft PowerPoint, but was fairly quickly moved over to Google Slides. Google Slides allowed us to work together easier, as every member of the team would have access to the file. This would allow all team members to join the design process, follow it along the way, and even make their own changes. As the system would be using simple design mechanics, Google Slides was a very adequate candidate for the job.

2.8.2 Flow

The whole flow of the system is designed using Lucid-Chart [8]. Lucid-Chart provides a service that allows the user to create different diagrams like UML diagrams and flow charts. This service is simple and intuitive to use, and worked perfectly for our project.

3 Requirements

Since this is a student project, 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 [1]. 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.

3.1 Functional requirements

3.1.1 Must have

Must have requirements	
Requirement-id	**User stories**
[Req-1]	As a student assistant, I must be able to log in as a student assistant.
[Req-2]	As a student assistant, I must be able to mark tickets as resolved
[Req-3]	As a student assistant, I must be able to assign tickets
[Req-3-A]	This must be done from the help list board or in the issue ticket
[Req-4]	As a user, I must have the ability to create a ticket
[Req-5]	As a user, I must have the ability to view where I'm in the lab queue after writing the ticket
[Req-6]	As a user, I must have the ability to log in if I want to
[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
[Req-8]	As a user, I must be able to register an account:
[Req-8-A]	As a user, I must be able to register an account through email:
[Req-8-B]	As a user, I must be able to register an account through discord:
[Req-9]	As a user, I must be able to confirm my email after registering:

3.1.2 Should have

Should have requirements	
[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
[Req-11]	As a user, I should be able to leave the queue.
[Req-12]	As a user, I should be able to update the description to my related tickets

3.1.3 can have

Can have requirements	
[Req-13]	As a user, I can view the help list from discord
[Req-14]	As a user, I can rate the student assistant that helped me.
[Req-15]	The system can create a prioritize queue
[Req-16]	As a student assistant, I can set tickets on hold.

3.1.4 Wont have

Wont have	
[Req-17]	The program won't have all the Universities class available

3.2 Technical requirements

Must have	
[Tech-Req-1]	As a user, I must have the possibility to have two-factor authentication
[Tech-Req-2]	As a student assistant, I must have the possibility to have two-factor authentication

4 User interface design

The focus of the application design is to make the user interface as simple and intuitive as possible, and is designed using a minimalistic approach. This approach uses large elements and has a very simple information flow on each screen. The design has also been created with accessibility in mind, as the large text makes it easier to read, and the large buttons are easier to press for users that may have disabilities. This also makes it easier to use the application on smaller screens, like for example on a phone.

The use of consistency, where the different screens use the same UI elements and base layout has also been an important aspect in the design process. The use of drop-down search boxes makes finding your room easy, and larger buttons make the system easier to navigate through. There should never be any doubts as to what button you should press to get to the desired screen.

The design takes heavy inspiration from simple UI applications, a specific example being the Kahoot application [9]. The color blue was selected by listening to scientific analyses concluding with blue being the most calming color[6]. Each designed page has been given a unique ID, starting with "UI" followed by their corresponding number, to easily identify the page later. The initial UI design is shown below.

4.1 Registration and login methods

4.1.1 The universal login screen

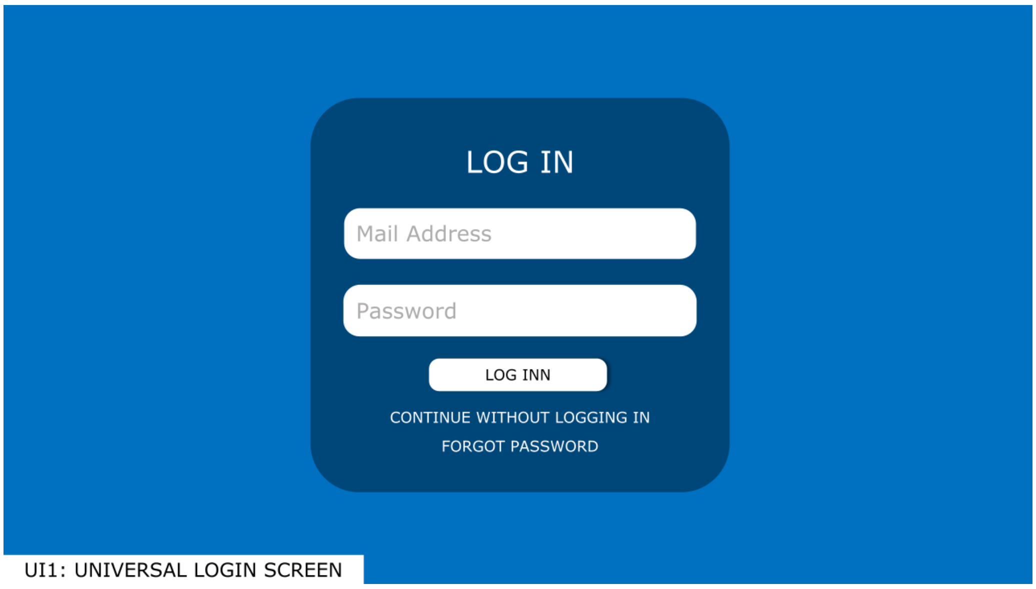


Figure 1: The universal login screen

The universal login screen is the page that allows the user, student assistant, and admin to log into the system. We decided to call it universal, as it was initially planned to use multiple login pages. The login page is designed to consist of a form where the user will input their email address and password, and a button displaying "Log in". Pressing this button with the correct input will log the user into the system. The user can also continue to the page without logging in, or pressing a link to reset their password. This page was planned to be the landing page..

4.1.2 Register new user

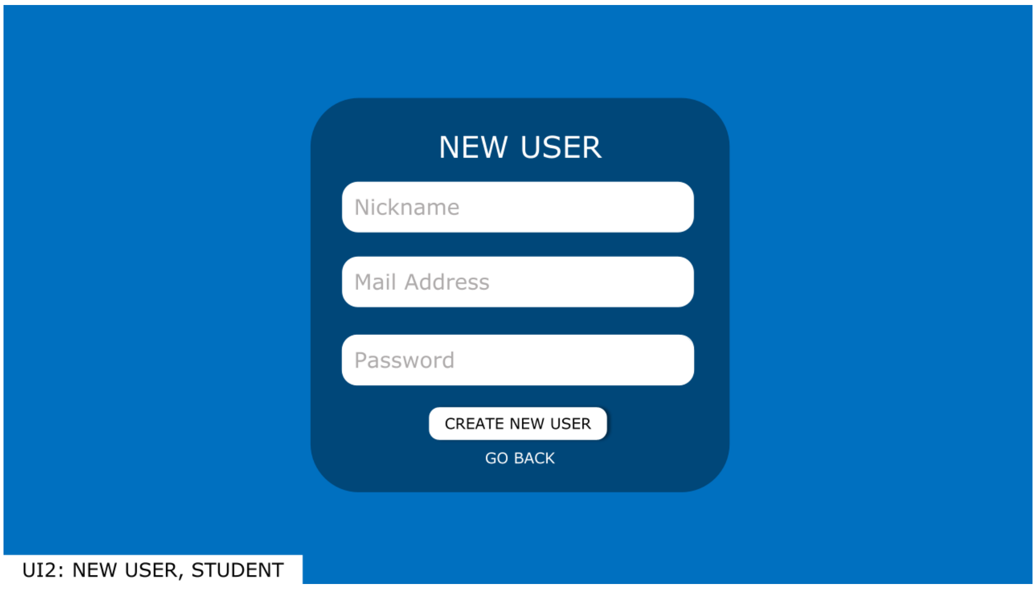


Figure 2: Page for creating new user

This page is designed to let the user create a new account if they want to. It is planned to consist of a form where the user will input their nickname, email address, and password, before pressing the "Create new user" button. Below this button, we also wanted to include a link to go back to the previous page, in case the user changed their mind.

4.2 Student view

4.2.1 Create new ticket, user logged in

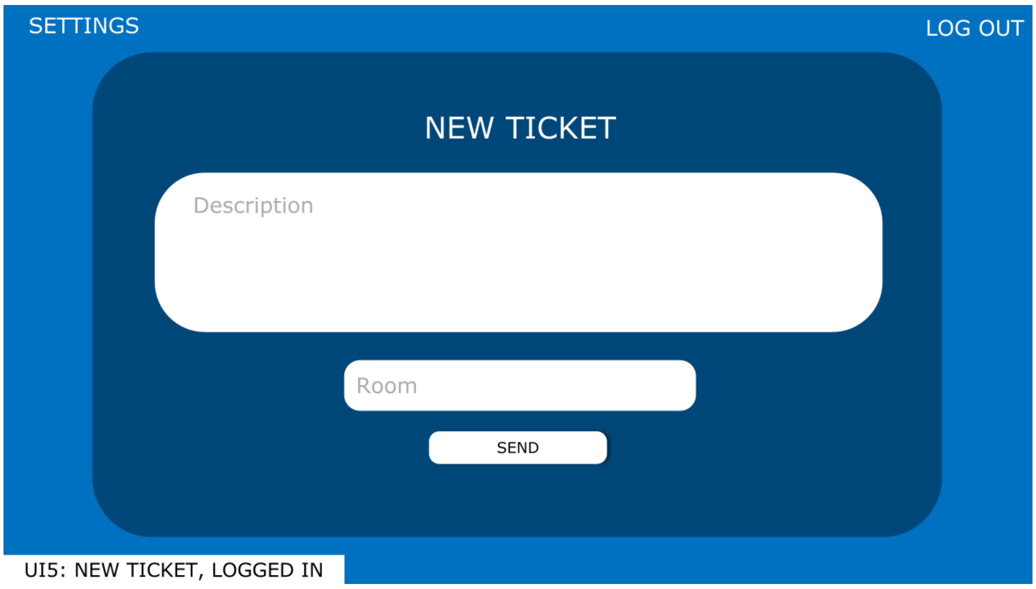


Figure 3: Page for creating a new ticket, if you are logged in

This page is designed to let a logged-in user create a new ticket in the system. The page is designed with a large description box, where the user can input their description, as well as a smaller box where the user can insert the room they are located in. After inputting this, the user can press the "Send" button, to submit their ticket to the system.

4.2.2 Create a new ticket, anonymous user

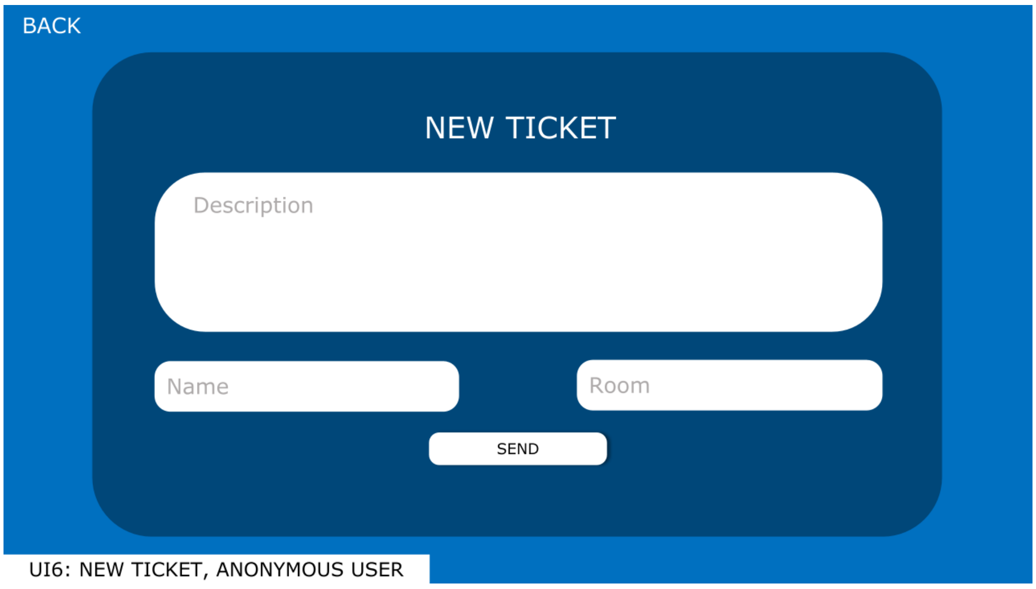


Figure 4: Page for an anonymous user to create a new ticket

This page is similar to the page in the previous section, with one important addition. As the anonymous user is not logged in, they will have no nickname and will have to provide one to the system. This is why we have added an input box beside the "Room" input, where the user will be able to add their name. Other than that, the page will look the same.

4.2.3 The user settings page

The user setting page is mainly designed to let the user change their nickname and password, as well as add their Discord username. The page consists of a form where they can edit their nickname, add their Discord username, and a save button to submit the changes.

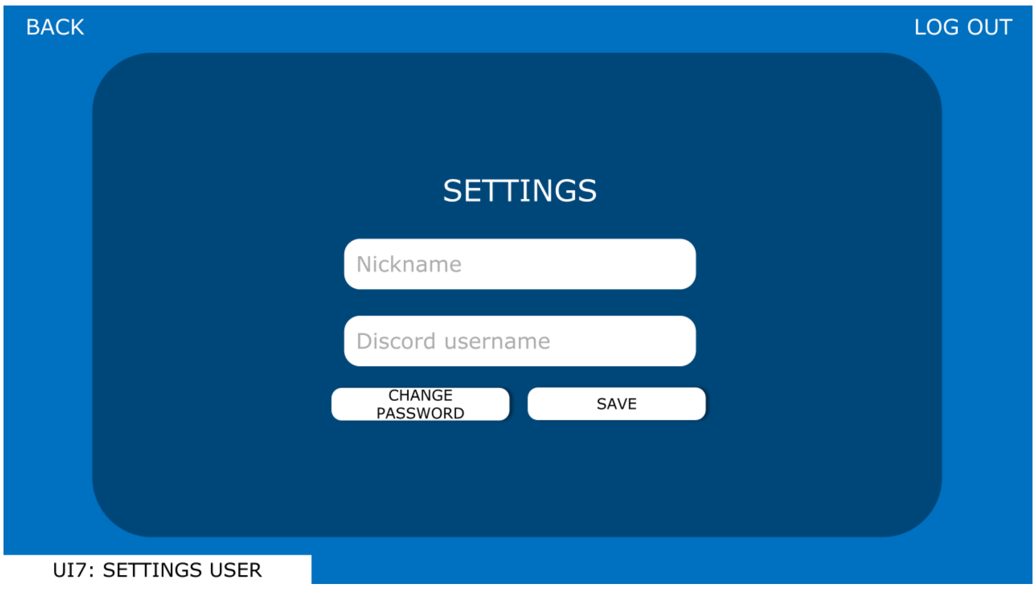


Figure 5: The user settings

4.2.4 The Ticket queue

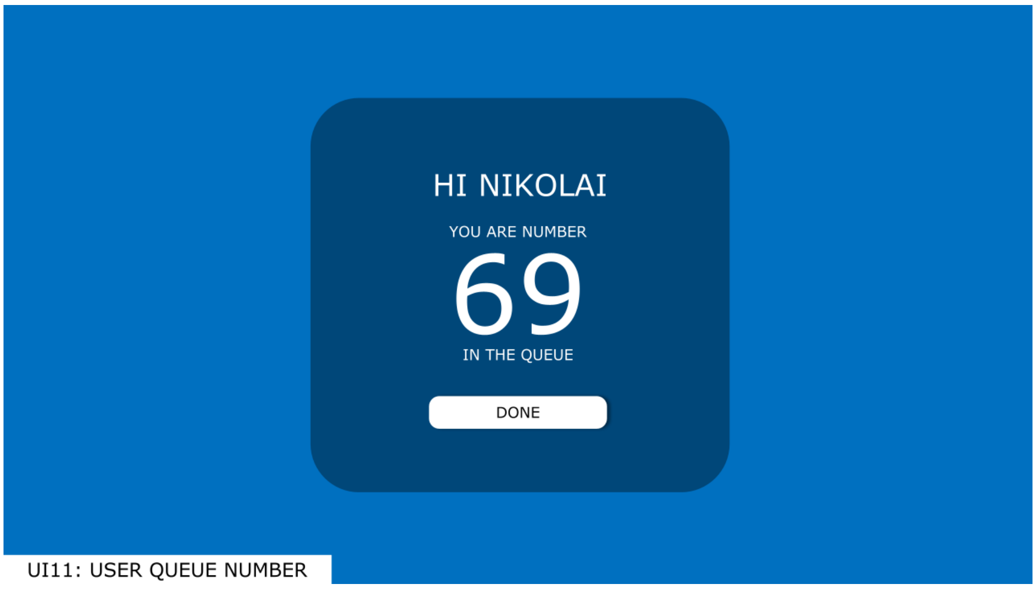
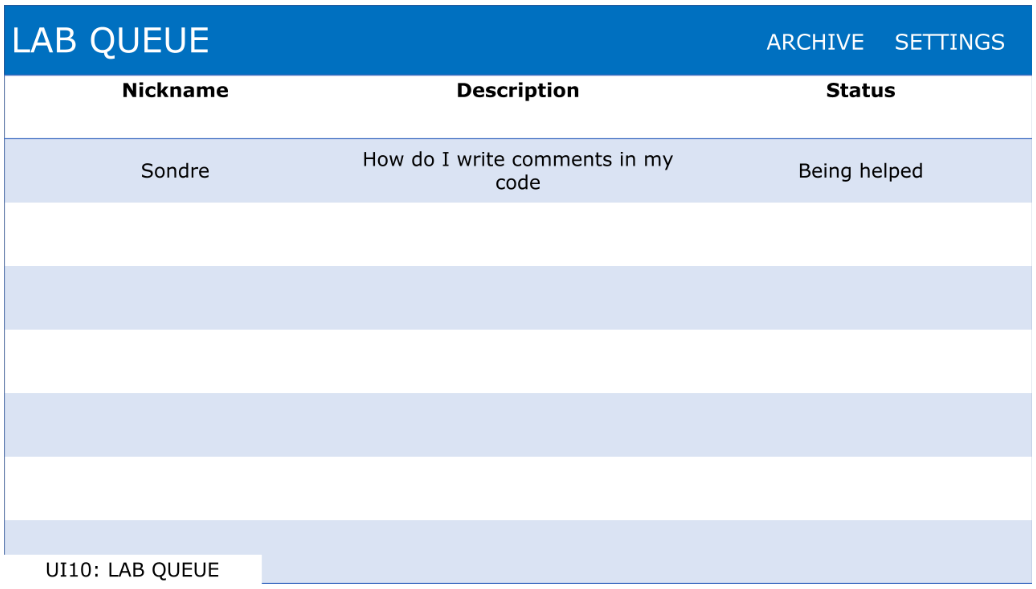


Figure 6: the page for showing the user what number he / she is in the queue

The ticket queue page will show the user their current position in the queue. It consist of a text-box greeting the user as well as the actual position of the user in the queue. If the user decides that they don't want help anymore, or if the assistant forgets to put the ticket as resolved, the user can press a button that will remove ticket from the queue, and take them back to the landing page.

4.3 The Student Assistants view

4.3.1 The Helplist



Nickname	Description	Status
Sondre	How do I write comments in my code	Being helped

Figure 7: UI10: The lab queue showing the tickets

The help list is a design derived from the Excel spreadsheet used by UiA as the current implementation for lab help. This page is designed to consist of a list with the different tickets, and will contain the nickname, description and status of the tickets. It will also have a title, and the possibility to enter the archive. Because this is thought to be the landing page for the student assistant, the page will also contain a button to enter the settings.

4.3.2 Archive

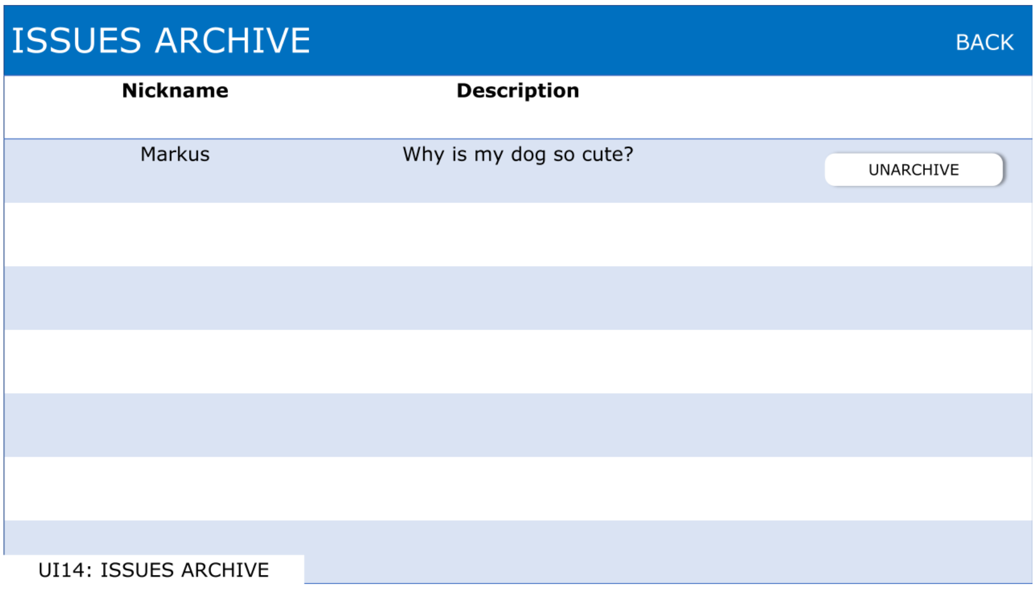


Figure 8: UI14: The list of archived tickets

The archive is designed to look almost exactly like the help list. The archive will contain all the finished tickets, and will let the student assistant send the tickets back into the queue. This is possible by pressing the "Unarchive" button placed at the side of the ticket. At the top, the "Settings" and "Archive" buttons are also swapped out with a "Back" button, to redirect the assistant back to the help list.

4.4 Admin view

4.4.1 The admin settings page

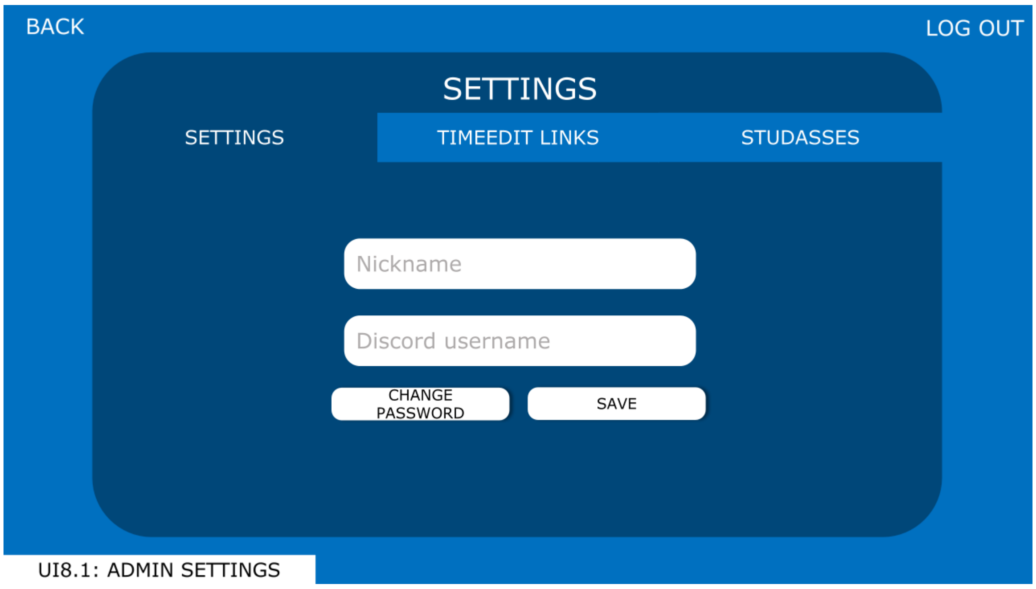


Figure 9: The admin settings view

The admin settings page is designed to look similar to the settings page provided to the users and student assistants, with the exception of an additional navigation bar. This navigation bar makes it possible for the admin to access more settings.

4.4.2 Time-Edit links

The "Time-Edit links" section of the admin settings lets the administrator add new links from Time-Edit into the system. The links are added by typing the link into the top window, and pressing the "Add" button. The links will then be shown be in a list below this window. The admin can later remove this link to remove courses from the system by pressing the "Remove" button beside the saved link.

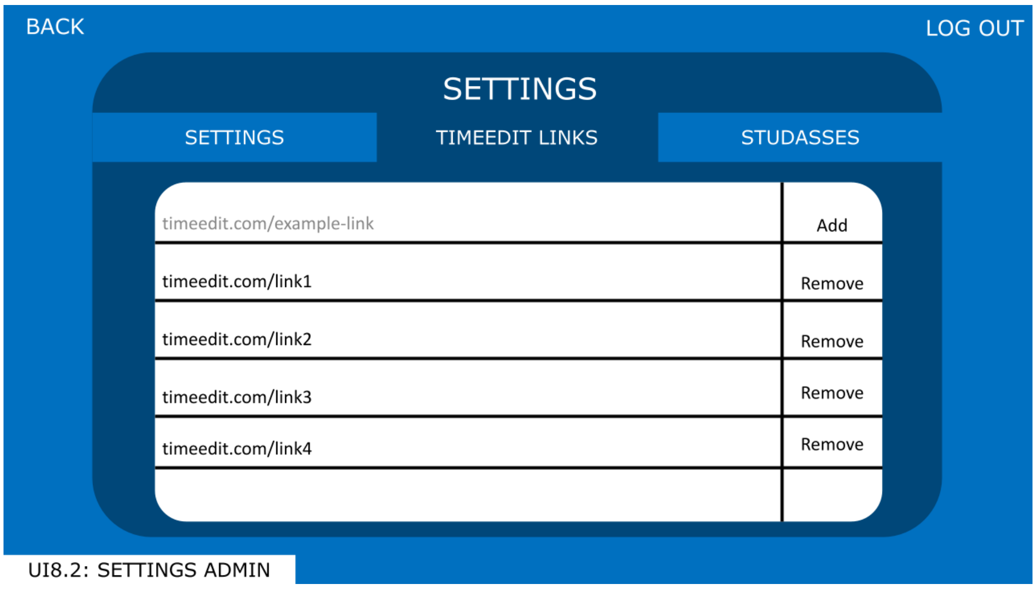


Figure 10: The view where the admin can add or remove courses

4.4.3 The Roles settings

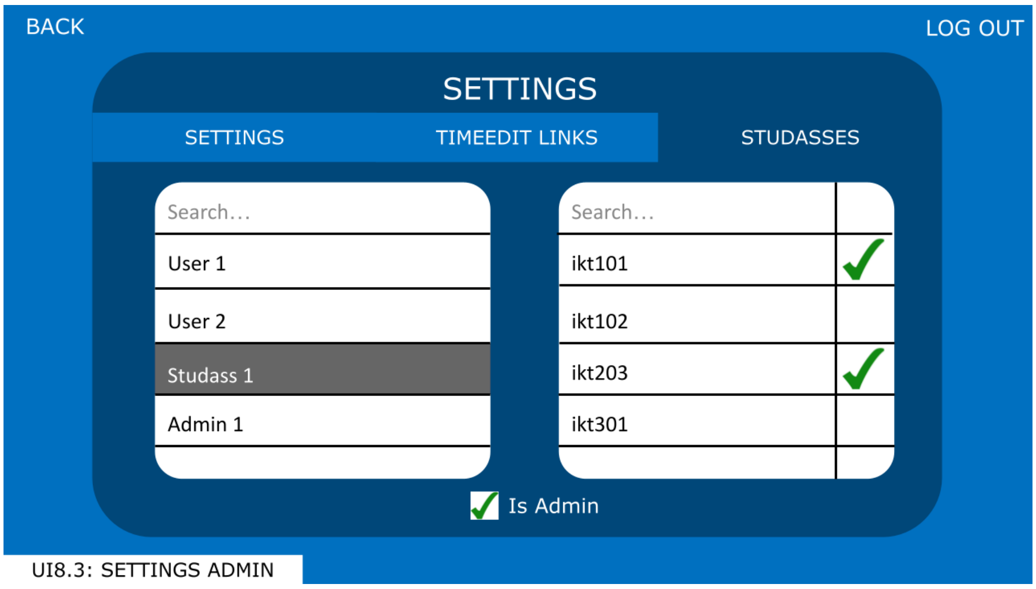


Figure 11: The view where the admin can manage user roles

The "Roles" section of the admin settings page is designed to let the admin choose the roles of the users in the system. This page consists of two tables, one showing the names of the users, and the other showing the courses. At the top of each table is a search bar, to make it easier for the admin to find the user and course. At the bottom of the page, we have also included a checkbox at the bottom of the page combined with the text "Is Admin", that the admin can tick off to change the role of a user to admin.

The left table consists of a list of all the users present in the system. The right table consists of the courses contained in the system. This table also contains check boxes where the admin can choose courses that a student assistant will work in.

4.4.4 Statistics

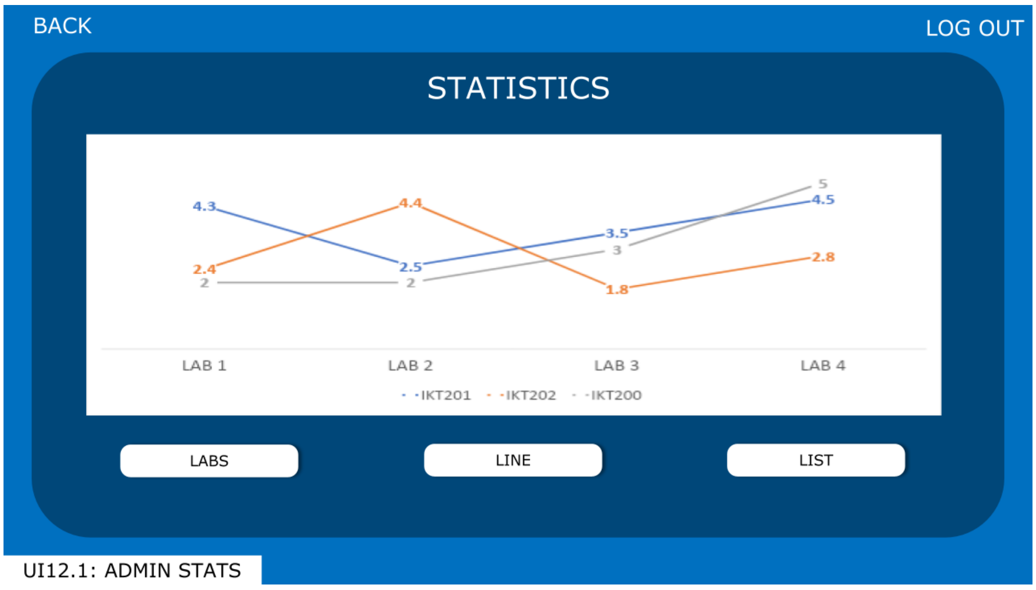


Figure 12: The admin statistics view

We thought it would be fun if the administrator could see what labs were most frequently used, and where students got the most help, as well as just seeing how many people were using the platform. We therefore planned to create a "Statistics" page, where this information would be shown to the admin. This page would consist of a large diagram, which could be changed by pressing buttons on the side. Here the admin would be able to choose which statistics they wanted to see, and if they wanted to view it as a line diagram or a list. The "Labs" button was planned to show a popup window where the admin could choose what labs to see in the charts.

4.5 Other pages

4.5.1 Change password

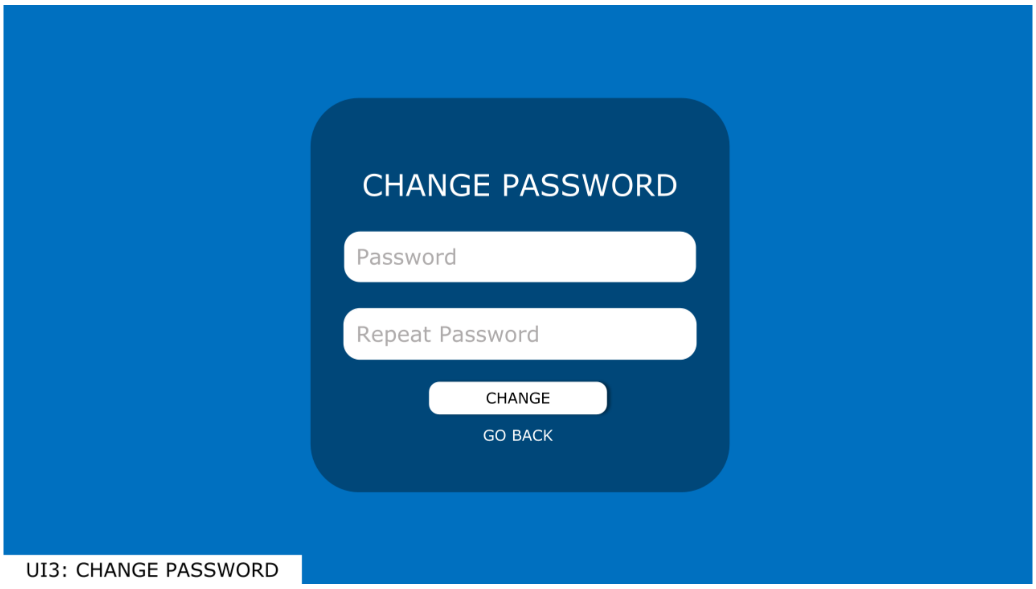


Figure 13: Page for changing the password

The "Change Password" page is designed to let the user change their password effortlessly. The page consists of two input boxes, one where the user can enter their new password, and one where the user repeats it for security reasons. After having inserted the new password, the user can click the "Change" button. Below this button, we have also added the functionality to go back to the previous page if the user changes their mind.

4.5.2 Reset password

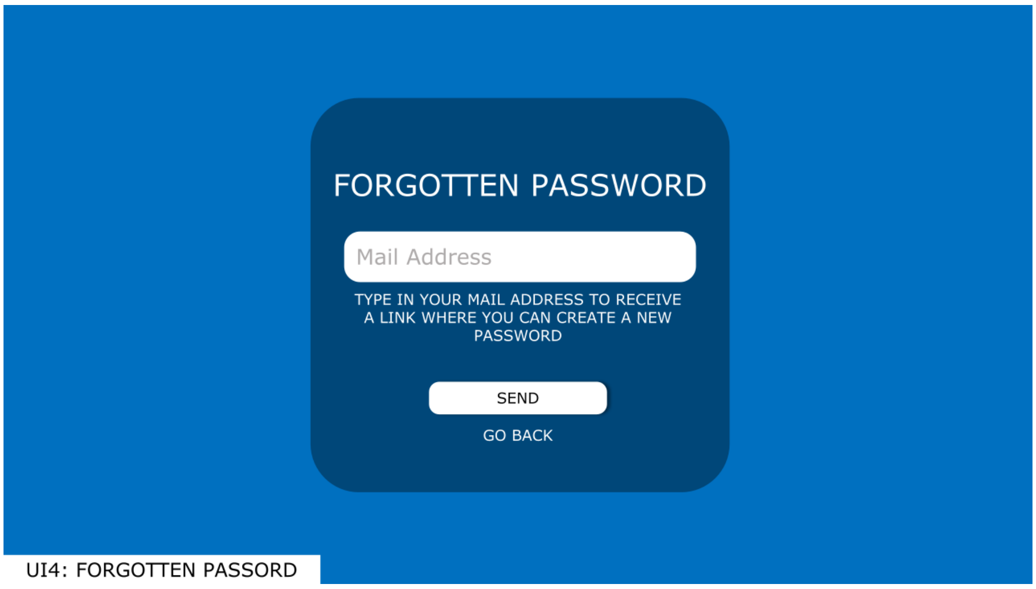

FORGOTTEN PASSWORD
Mail Address
TYPE IN YOUR MAIL ADDRESS TO RECEIVE A LINK WHERE YOU CAN CREATE A NEW PASSWORD
SEND
GO BACK
UI4: FORGOTTEN PASSORD

Figure 14: UI4: Page for getting a new password if you have forgotten it

This page is designed to make it possible for the user to get an email with a reset link to a new password. The design consists of an element where the user can insert their email address, and a button labeled "Send", to submit. We have also designed the page to include a label with instructions on how to reset the password, as well as a button where the user can go back to the previous page.

4.5.3 Generic error

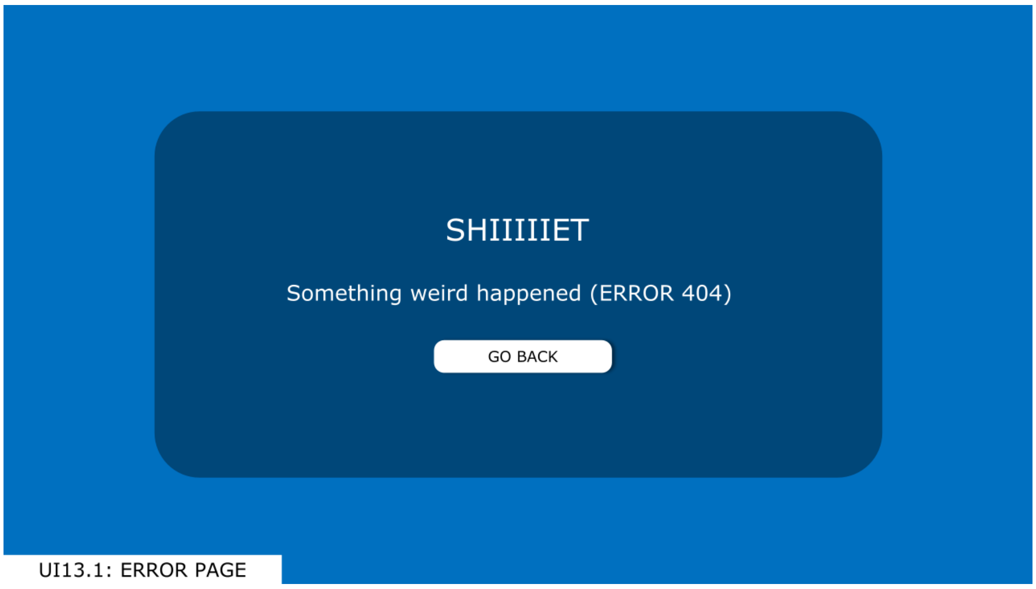


Figure 15: UI13.1: The generic error page

The generic error page is a page that will show up if an error happens. It is planned that the error code and shown message will be different for each different error that occurs. This page will also have a button to take the user back to the landing page.

4.5.4 Error 401: Unauthorized

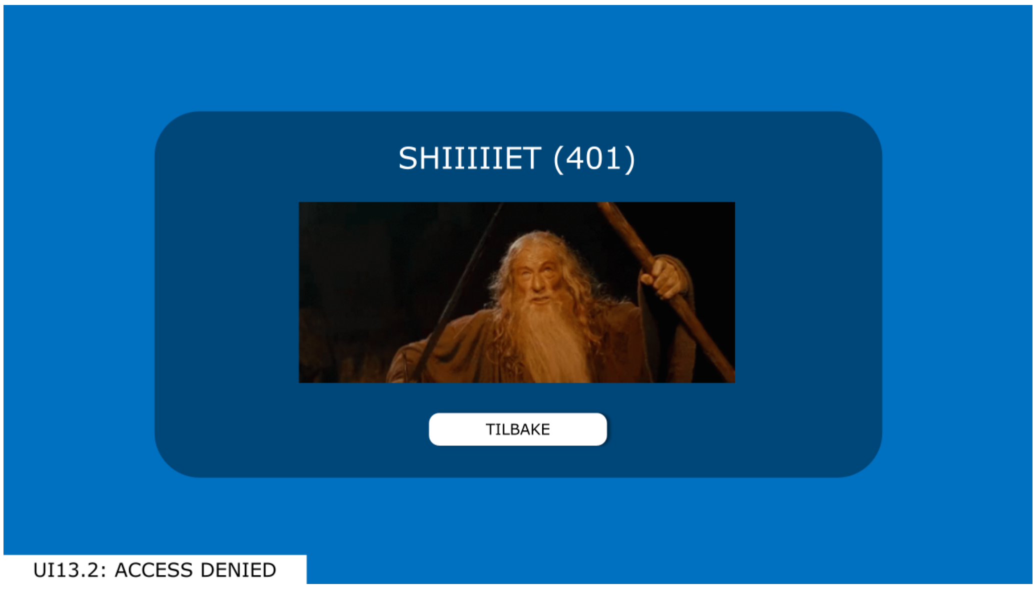


Figure 16: UI13.2: The error page of error code 401: Unauthorized

As this is a school project, we opted to add in a little fun Easter egg to it. This Easter egg is confined in the "401: Access Denied" error page. The access denied page is shown if someone tries to access a domain they do not have access to. This page is designed to consist of a little GIF of Gandalf from Lord of the Rings shouting "You shall not pass", and a button that will take you back to the landing page.

4.6 Application flows

The initial design of the flow of the system is shown in the figure. We created this design before starting the implementation process, to have a template to look at while designing the software. The created software does not exactly correspond with the initial design, as we have made some changes where it has been necessary along the way. More on this in the "Discussion" chapter.

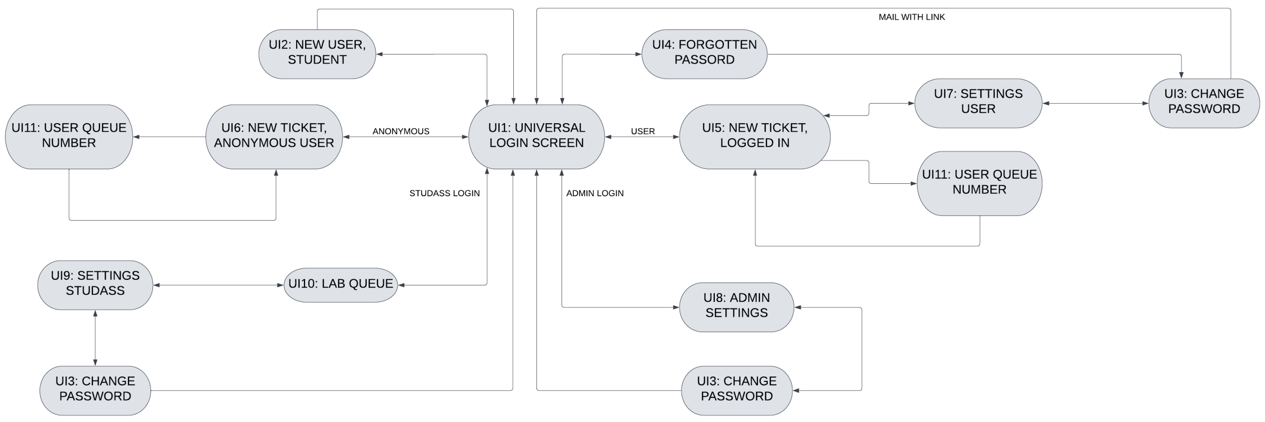


Figure 17: Flowchart showing the initial flow design of the system

The above flowchart shows the initially designed application flow, complete with the previously designed views. Each view is identifiable by the UI code.

5 Technical background

5.1 Tools and technology used

5.1.1 Jira

Jira is a software tool that helps teams to structure and plan their project, and is primarily used among software developers. Here we can add tasks for the user requirements that we made, and weekly sprints with tasks to be done. A sprint is a list of tasks that has to be done within a set period of time. We tried to do one sprint per week. For every meeting we had, we went through the current sprint and updated it. We could also use Jira for tracking our time spent working on a specific task.

5.1.2 Bitbucket and Git

Bitbucket is a Git-based code collaboration tool that lets developers easily work together with code in teams [3]. We had a variety of choices at the start of the project for which kind of cloud we wanted to use, and eventually settled for Bitbucket and Git. We decided to use Bitbucket and git because they were tools we had used before, and they works well in tandem with Jira. On Bitbucket we stored our code in the main branch and set up other branches for more specific tasks.

We made the main branch, and created a readme file about what to name our branches, how to add a branch, how to merge a branch, and how to commit changes. We assigned tasks on Jira and the developer that worked on a specific task made their own branch corresponding to that task. When the task branches were completed and working, they were merged in the main branch. The individual branches had to be reviewed and accepted by at least 2 other team developers before they were allowed to be merged into the main branch. This way we could easily track our changes, and get a good overview of what had been done.

5.2 Frameworks

5.2.1 ASP.NET

ASP.NET is an open source web framework that is created by Microsoft, that we used for building the web application[15]. We used the framework's template and setup that was created for us, and modified or added other tools as we wanted.

We also choose to use Razorpages instead of using MVC which was already implemented in the template. Razor pages have a more coding page focus, and it was stated by Microsoft the creators of asp.net that using razor pages could be easier and more productive than using controllers and views[14]. There is auto-generated code in razor pages, like defined urls. We found this easier to use.

5.2.2 Identity Framework

Identity Framework is a framework that ASP-NET Core identity introduced, that contains a membership system and allows a built-in login function. The users that create an account will have their data stored in a SQL database that Identity provides.[13] We used the Identity framework as the base for our login functionality, but also added in some additional optional attributes like nickname and discord tag. These additional attributes were also stored in the same database. Besides the Identity login function, we added a discord login and two-factor authorization that we will describe later.

5.2.3 Entity Framework

Entity Framework Core is a framework that Microsoft uses for .NET developers and is an object-relation mapper (ORM). This allows us to work with databases using .NET objects.[12] We used Entity Framework core to create models, that made up the classes that were represented in our database.

Entity Framework has three different approaches when constructing a database; Database first, Code first, and Model first.[12]. Here we went for the code first, where we created the code for the models and Entity Framework created a database from our description of the models. This was the approach that the whole group was already familiar with, because we have all experience with object-oriented programming.

5.2.4 Blazor

Blazor is a framework that allows the user to use C# code when loading the page, instead of using JavaScript in the browser [16]. This allows both the frontend and the backend to be created using .NET. Blazor is used for loading all pages with data from the backend controller. This is possible by creating a public variable in the controller, and then passing the variable into Blazor. This variable can be everything from a simple integer, to an entire model. Using an "@" tag in the Blazor page, this variable is then accessible, and can be used to access the value of the variable.

5.3 Libraries

5.3.1 SignalR

SignalR is an open-source library created by Microsoft for ASP.NET Core that makes real-time two-way communication between the server and the client possible [7]. The library uses multiple technologies, among them Websockets, to create the communication layer.

SignalR is in our application used for communicating live updates between the server and the client. This library is used for when the client needs to be able to show a new ticket on the help list when it has been created and also update the queue view. We opted to use SignalR over client polling, as client polling is slow and regarded as bad practice. This enables a snappy experience for the end user of the product. The library uses hubs on the server side to communicate with the clients, which in our case run javascript equipped with the SignalR javascript library. These hubs are part of the backend codebase and is the unit that initiates connections and handle communication with the clients. The hub has access to the database, user model, role model, and is initiated at startup to immediately connect with clients. The client then initiates the connection to the hub, and they can communicate freely through this open socket.

5.3.2 Bootstrap

Bootstrap is known for its responsiveness and its scalability. These are two of the reasons why we chose it. Bootstrap is also an open-source framework and has many predefined components[4]. The library makes it easy to define different designs for different screen sizes. This makes mobile viewing possible. We have used a combination of Boostrap mixed with standard CSS for the webpage design. Bootstrap has with its ease of use helped us a lot with the front end, and provided great value to the application.

5.3.3 OAuth Discord

As most of the IT subjects at UiA use discord as a means of communication, we decided that it would be practical if the students could use this service to log on. OAuth Discord is a middleware package accessible from GitHub or through NuGet that allows the programmer to easily implement the discord login service through the program.cs [10]

5.3.4 QR Codes for JavaScript

In order to generate QR codes for our 2FA implementation, we used a library called QR Codes for Javascript. The library is free to use and has no dependencies. It works by sampling some data and formatting it into a QR code format. [5]

5.4 Languages

In this project, we have used a variety of different languages for different parts. These languages include:

5.4.1 C#

C# is used in both the back and front end of the system. On the backend, it is used in controllers and entities to provide the server processing needed to run the application. In the frontend it is used in the Razor pages to process variables coming into the HTML from the controllers, and in some places to communicate further with the controllers by collecting input from the user of the system.

5.4.2 Javascript

Plain Javascript is used in the front end to provide capabilities such as interactivity to the pages. Javascript also plays an important part in the SignalR communication for instant updates on the pages, without the need for a full page reload

5.4.3 HTML

HTML is used on the front end, to provide the elements that the user will see and interact with.

5.4.4 CSS

CSS is used to design the HTML elements and create the layout of the whole front end. The CSS shows the HTML elements where they should be, and what they should look like.

6 Product

This section will describe in more detail what the product is going to look like. Here we will show the different pages that you can visit through our website. The product is going to serve the admins, students, and student assistants. The view for both the customer and the assistants are going to be an easy-to-use service, where the students can create ticket fast and easily, while the student assistants can easily manage these tickets.

6.1 User interface design

The completed user interface design, follows the initial design very closely. The focus was to create a simple and intuitive design for our product. Our completed design follows the design presets initially created. The finished and implemented design is shown in the chapters below.

6.2 Registration and login methods

6.2.1 Register new user

This is a page that allows a student to create a new user. The page consists of a form that allows you to input your email, nickname, Discord tag, password, and confirm password. After a user has registered, they will be prompted to verify their email account, and after verifying it they will be able to log in. When the student has logged in, they will be redirected to the "Create ticket" page.

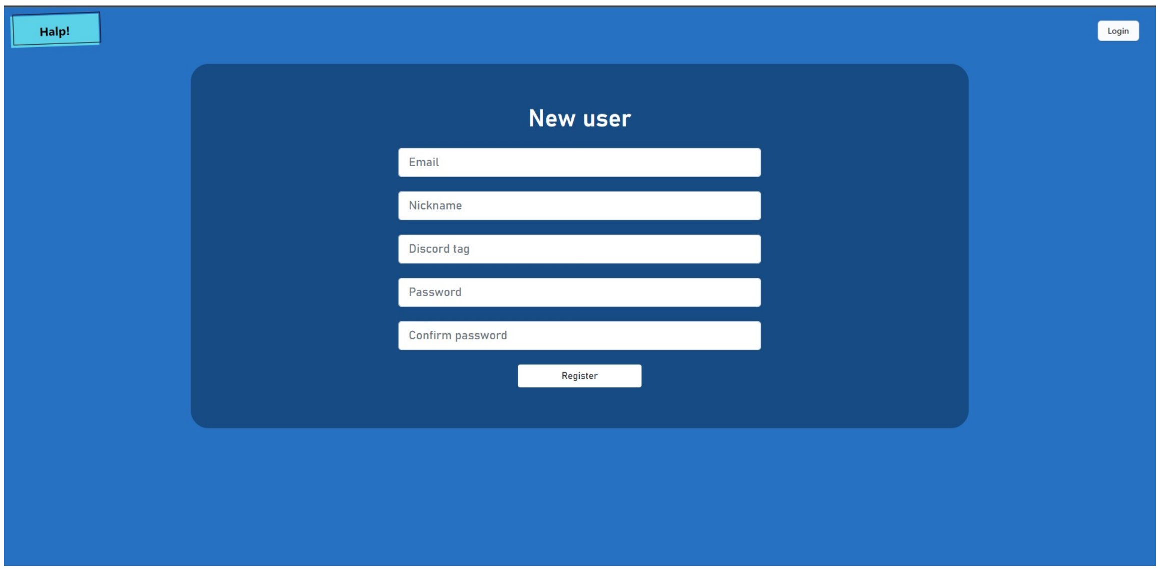
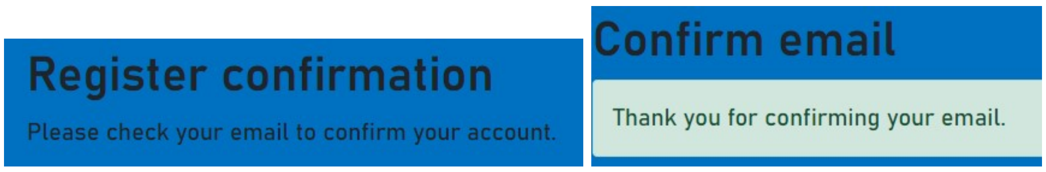


Figure 18: Register user

After when you click the "Register" button you will be asked to confirm the email address that they entered.


Register confirmation

Please check your email to confirm your account.

Confirm email

Thank you for confirming your email.

Figure 19: Mail Confirmation

6.2.2 The universal login screen

The universal login screen is a login page where all users can log in. It is an intuitive screen, consisting of a login form, and buttons to redirect you to the page for registering as a new user or to reset your password. Additionally, it consists of a button for logging in using Discord. On the discord login you will be redirected to Discord to authenticate yourself, and then be redirected to the website to enter a nickname (only happens the first time you use this method of logging in). After that, you won't be prompted for the nickname again. When the student has logged in, they will be redirected to the "Create ticket" page.

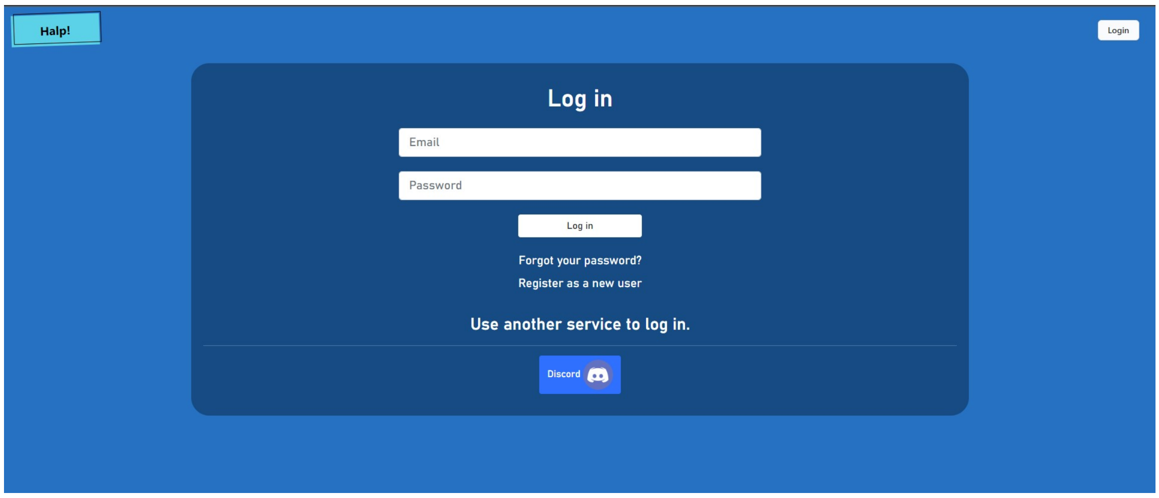


Figure 20: Login

6.3 Student view

6.3.1 Create a new ticket, anonymous user

The first page you will arrive on when you enter the website is our "Create ticket" page. Since the students that are attending the lab don't necessarily need to be logged in to use the system, we want it to be as fast and easy as possible to get help right away. On the page, you will find a box to enter a name that the student assistants will see, a description of the problem, and the ability to choose what room you are located in. The room choice is a drop-down menu, where you can pick from available locations. Pressing the "Send" button lets the user submit the ticket to the system, and they will arrive at the "User Queue Number" page.

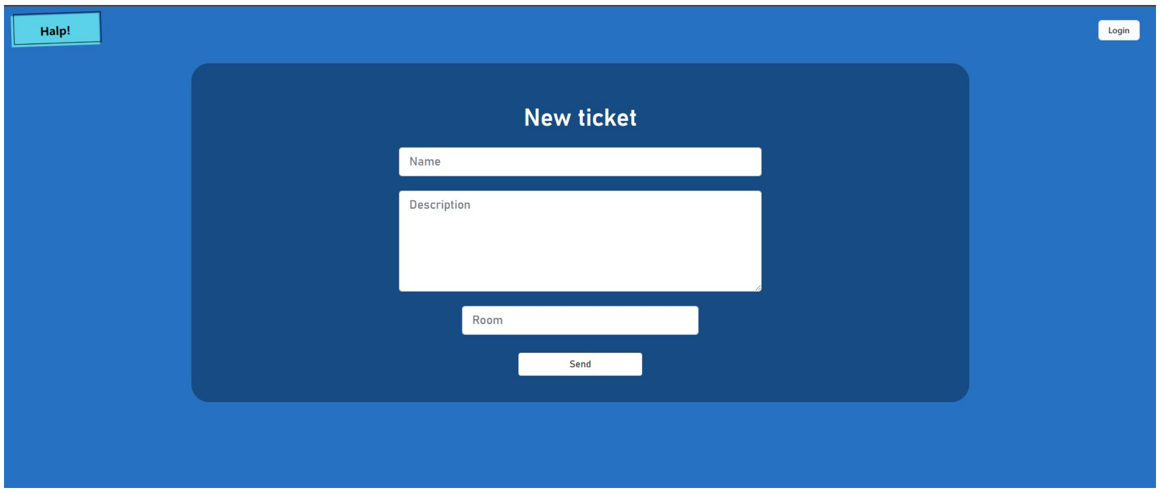


Figure 21: Create ticket

6.3.2 Create a new ticket, user logged in

This page basically has the exact same layout as if the user was anonymous, with one key difference. If the user is logged in the "Name" field will already be filled out for them. Other than this, the page has the exact same functionality. The crate ticket page is made to be as user-friendly and easy to use as possible. It is important that the design is straightforward and intuitive to give the students an incentive to use our services. If using our service was a hassle people would go for other programs that they are already familiar with. An example of our user-friendly design is that if the user is already logged in the Name box will automatically fill in their name when they enter the "Create ticket" page. We also purposefully made the description box cover the majority of the page as this is the most important feature of our application, and should be the first thing our users sees.

6.3.3 Ticket queue

The ticket queue page is where the user gets redirected after submitting a ticket. At this page, the user can see their position in the queue as they wait to be assisted by a student assistant. This page is dynamic meaning their place in the queue will update in real-time. This page shows the user's nickname and their current position in the queue. Furthermore, the user is given the option to edit their current ticket or quit the queue if they no longer need help. All the features within the queue page will work even if the user is not logged in. Similarly to the create a ticket, the queue page is made with user-friendliness in mind. There are purposely not a lot of options for the user to interact with the queue, making it simple and intuitive.

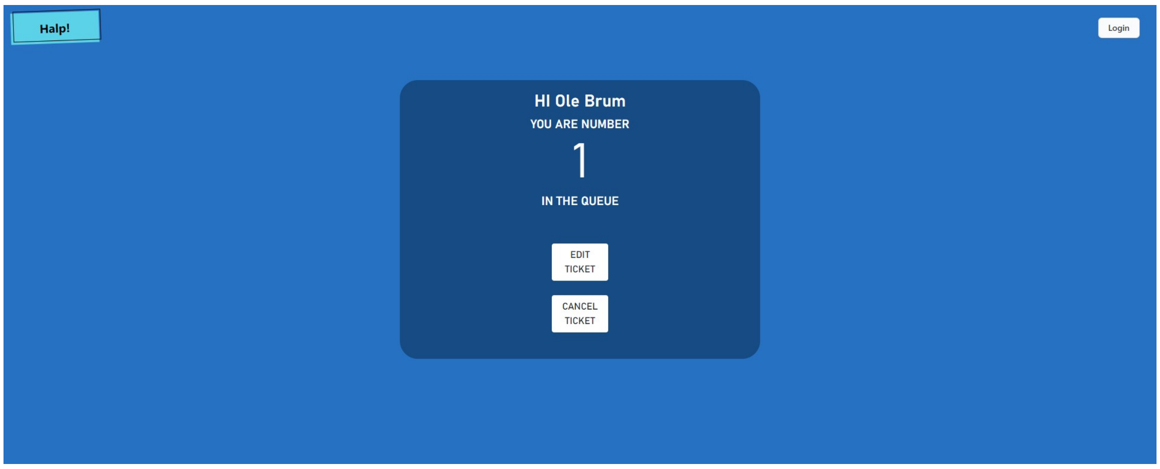


Figure 22: Ticket queue

The user has two options for interacting with their currently active ticket, these two options the "Edit ticket" button, and the "Cancel ticket" button.

Edit ticket:
If the user presses the "Edit ticket" button they will be redirected to another page where they have the ability to alter their currently active ticket.

Cancel ticket:
If the student presses the "Cancel Ticket", their currently active ticket will be deactivated and put into the archive. Afterward, the user will be redirected back to the create ticket page, where they are free to create a new ticket.

6.3.4 Edit ticket

As explained in the queue page information, the user has the ability to edit their currently active ticket. If the user presses the "Edit ticket" button within the ticket queue page they get redirected to the edit page. Here the user has the ability to edit any mistakes in their ticket before they get assistance.

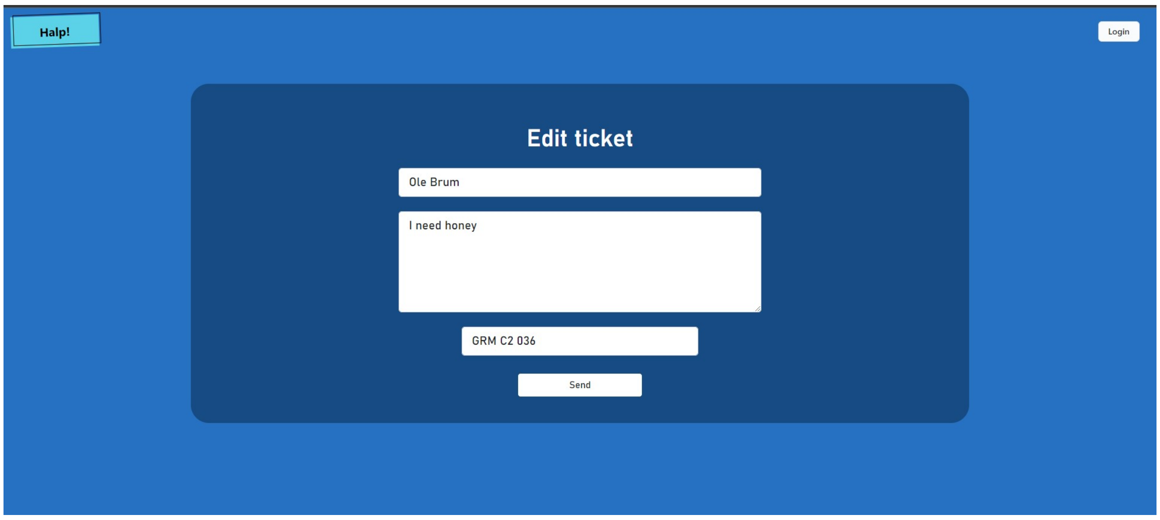


Figure 23: Edit ticket

The edit ticket page has a layout that closely resembles that of the create ticket page. This is done to make it feel similar and intuitive. When the user enters the edit page all the current data of their ticket will be displayed in the text boxes. This way the user won't have to retype their entire ticket if they have made a small mistake. Once the user has entered his updated information he can press send to get returned to the queue page. This will update the ticket for himself and for the student assistants instantly. Editing a ticket does not put the user any further back in the queue, they will keep their queue number throughout editing their ticket.

6.3.5 The settings page

If the user is logged in with only a user role, the user will get a simple view consisting of a button to change the nickname, a button to change the password, a button to log out from external services, and a button to delete all their data. These buttons will redirect to the different views, where the user can perform these operations. These individual views are explained further later on in the report.

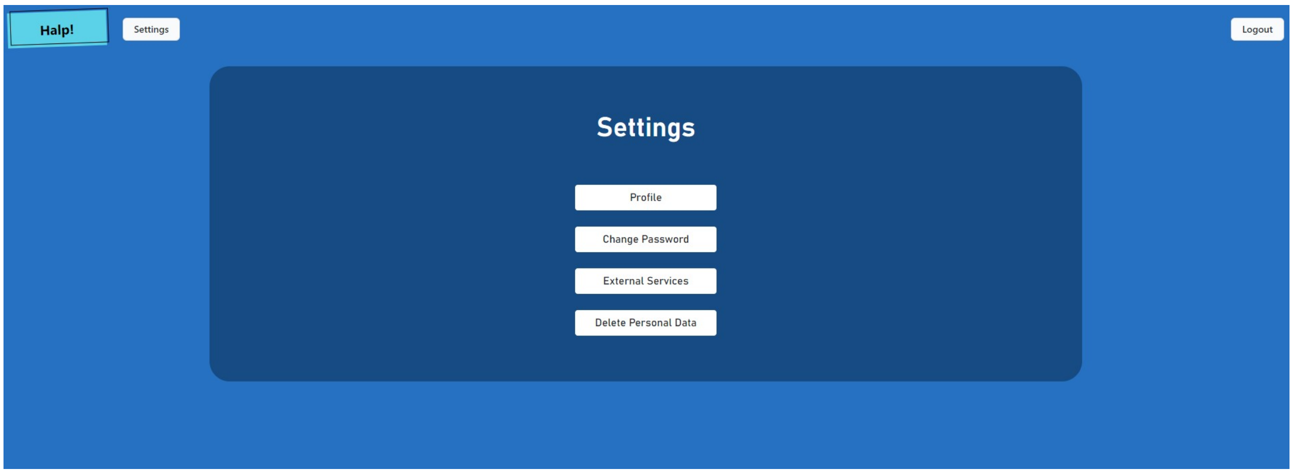
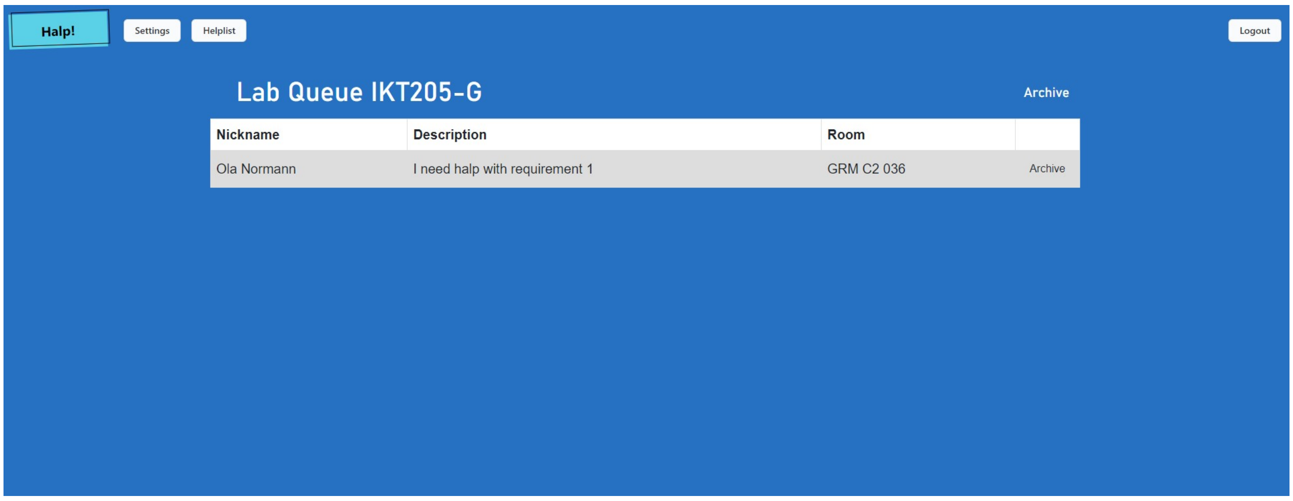


Figure 24: Settings view for users

6.4 Student Assistant's view

6.4.1 Overview of lab queues

The student assistants have a web page where they have an overview of their assigned labs and that they currently are a student assistant. Here they can press on the current lab they are attending and get the list of the students that have created a ticket.



Lab Queue IKT205-G

Nickname	Description	Room	
Ola Normann	I need halp with requirement 1	GRM C2 036	Archive

Figure 25: Overview labqueue for a course

6.4.2 Help-list for courses

When a student assistant has entered a help list they will get an overview of all the tickets that are currently posted. Here they will have some functions that are available to them. On the right hand, there is a button named "Archive" which the student assistants press when they are done with a ticket. After pressing "Archive" the corresponding ticket gets put into the archive, and the student assistant can move on to the next ticket.

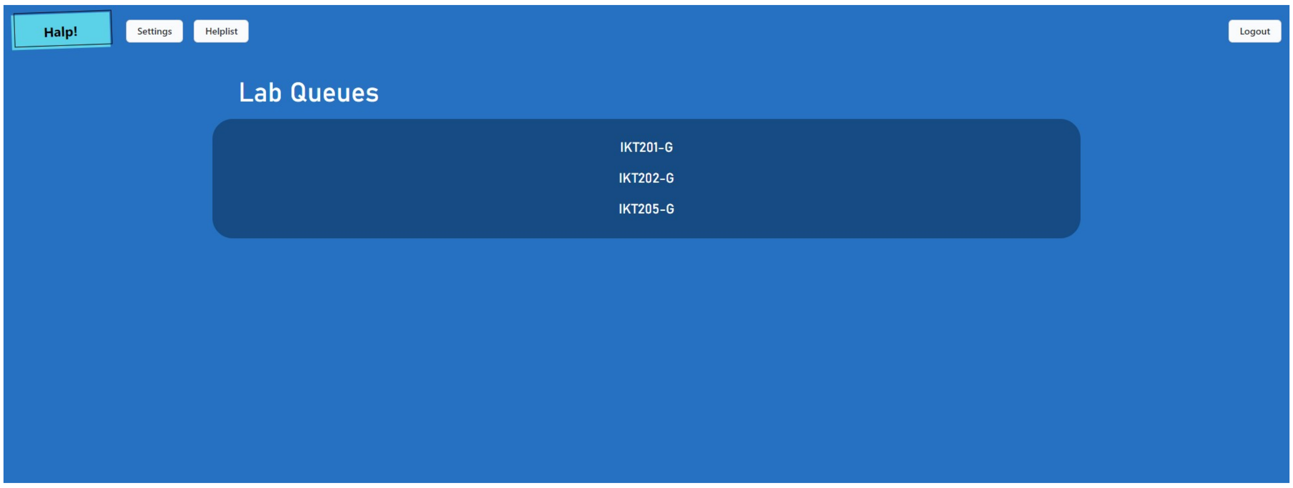
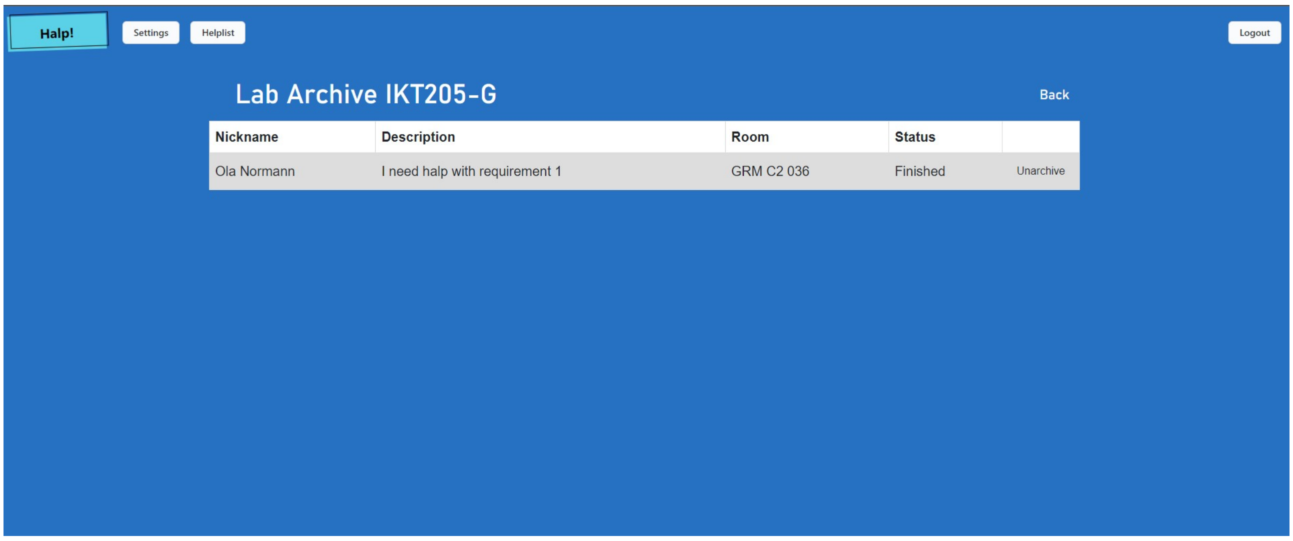


Figure 26: List over helplist courses

6.4.3 Archive

The Archive is where the tickets that are completed by a student assistant are stored and listed. If for some reason a student assistant regrets that they were done with a ticket they have the ability to go to the archive and "unarchive" the ticket to get it back into the queue. A ticket that gets unarchived will pop back into the queue at the first position.



Lab Archive IKT205-G

Nickname	Description	Room	Status	
Ola Normann	I need halp with requirement 1	GRM C2 036	Finished	Unarchive

Figure 27: Page for the archived tickets

6.5 The Admin view

6.5.1 Pages

The admin will have access to all pages in the entire system. There is no limit to what he can access if the page exists. All these pages look the same to the admin as they do to everyone else in the system, aside from the fact that the admin has a more advanced settings page.

6.5.2 Settings

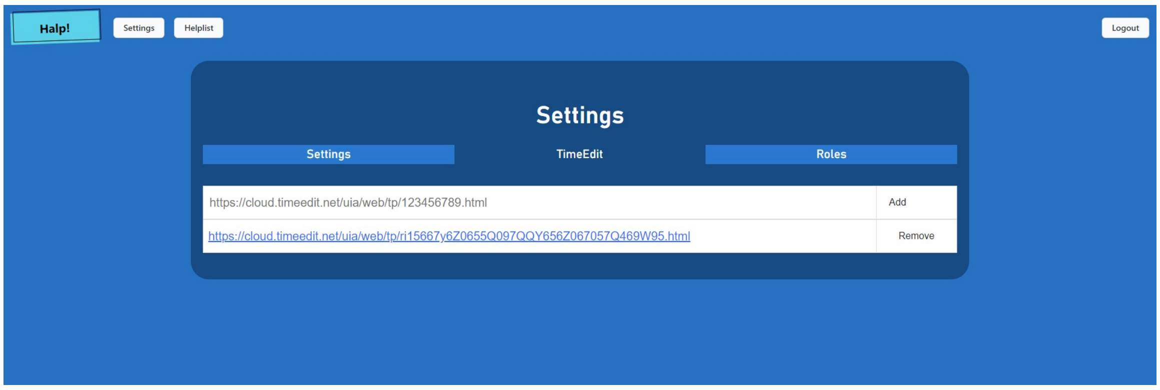
The settings page is divided into three parts, depending on the logged-in user. The page is intuitive, and contains only buttons if logged in as a user, but also a top navigation bar if logged in as an admin. This navigation bar enables the admin to easily switch between the different views.

The admin will have access to the "Time-Edit" section of the settings. Here the admin will be able to add more links from Time-Edit, containing more courses. The links are easily added and removed by using the "Add" and "Remove" buttons to the right of the link. This page uses SignalR to communicate with the server, through the "SettingsHub". Every time the admin presses a button, a message is sent to this hub, allowing it to immediately update the database. After having updated the database, the hub sends a response to the server, which lets it verify that the operation was successful.

This page has a search function that makes it possible to search for students and course codes. This search function is created in the frontend javascript and consists of only showing the names or course codes that contain the search words put into the search form.

The full functionality of the "Roles" page: A user can be added as a student assistant in any course, and have multiple courses. This is done by first pressing the name of the user, and then the checkboxes of the courses to be given student assistant permissions. A user can be given admin privileges by pressing the checkbox at the bottom. A student assistant can be removed as a student assistant in all courses and is thus be demoted to a user. A student assistant can be promoted to admin by pressing the checkbox at the bottom. This results in the student assistant losing all the courses, as the admin will already have access to all courses, not just the ones they were a student assistant in. An admin can be demoted to a student assistant or user by any other admin, but if there is only a single admin, he will not be able to demote himself without first promoting someone else to admin. Attempting to do this will result in an error being shown between the navigation bar and the tables. This error is sent by the hub to the client, after verifying that the admin cannot be removed.

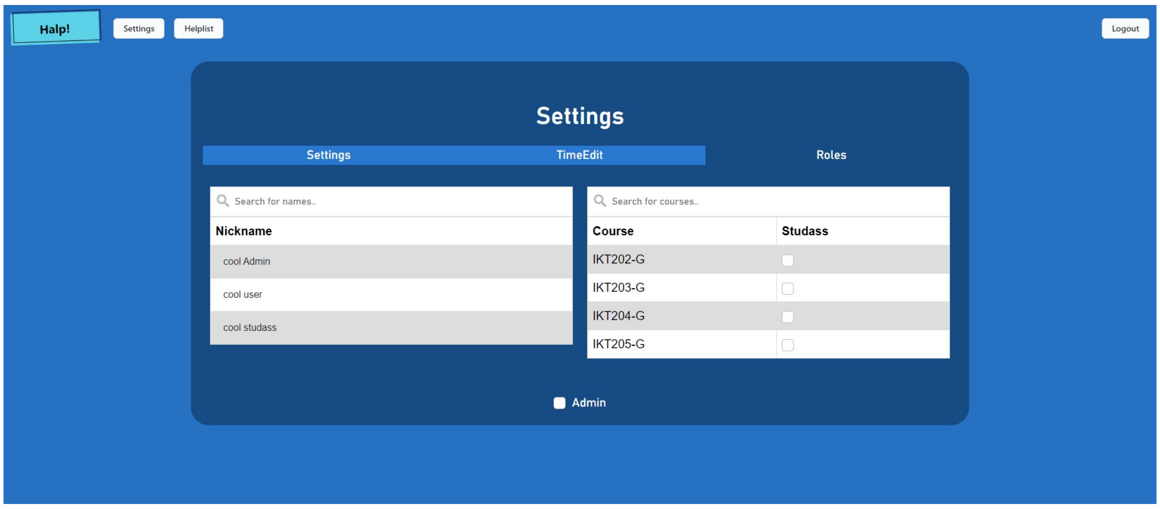
The TimeEdit section communicates with the server using HTTP requests, more specifically POST requests. This was deemed as the simplest approach, over using SignalR. When pressing the "Add" or "Remove" button, a POST request is sent to the controller on the backend of the system, which processes the request. This is done by removing or adding the link to the database. After having done this, a response is sent to the client, updating the page.



https://cloud.timeedit.net/uia/web/tp/123456789.html	Add
https://cloud.timeedit.net/uia/web/tp/ri15667y6Z0655Q097QQY656Z067057Q469W95.html	Remove

Figure 28: Settings view for Timeedit links

The registered admin of the page will have access to the "Roles" section of the settings. Here the admin will be able to assign roles to other users. To assign a user to become a student assistant, the admin will first click the user they will assign, before checking the courses they want to assign the student assistant to. There is also a checkbox at the bottom to give a new user administrative abilities.



Nickname
cool Admin
cool user
cool studass

Course	Studass
IKT202-G	☐
IKT203-G	☐
IKT204-G	☐
IKT205-G	☐

Figure 29: Role settings view admin

6.6 Other pages

6.6.1 Change password

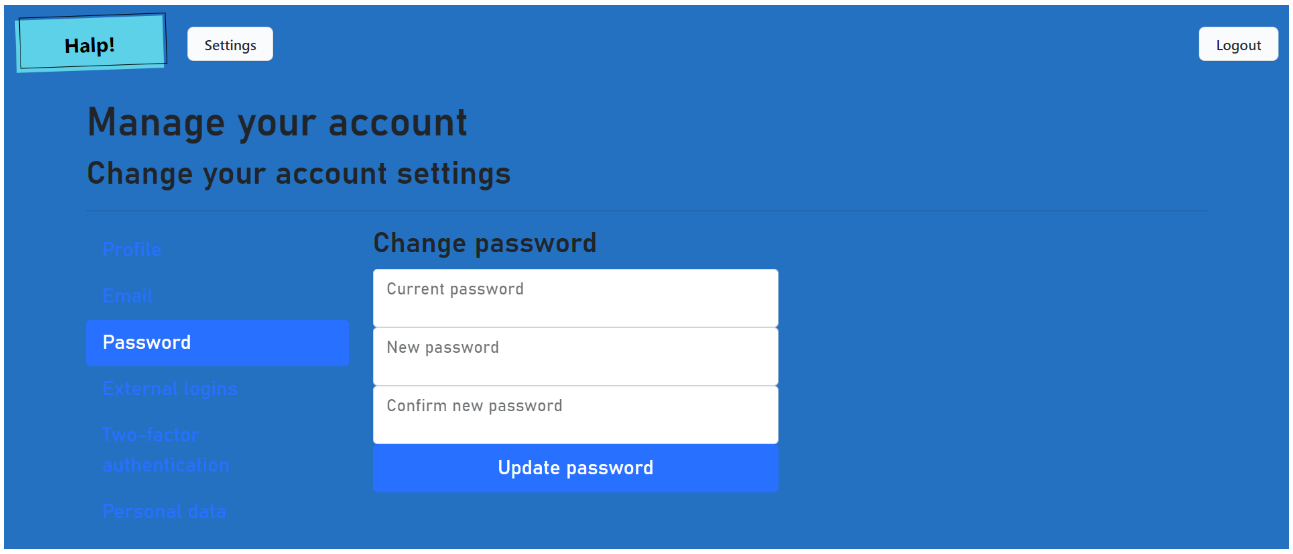


Figure 30: Page for letting the user change their password

If the user wishes to change their password they can press the setting button, then the change password button to get to the change password page. At the change password page, the user can fill in their current password, their desired new password, and their new password again to confirm it. After entering this information the user can press the Update password button to change their password.

6.6.2 Forgotten password

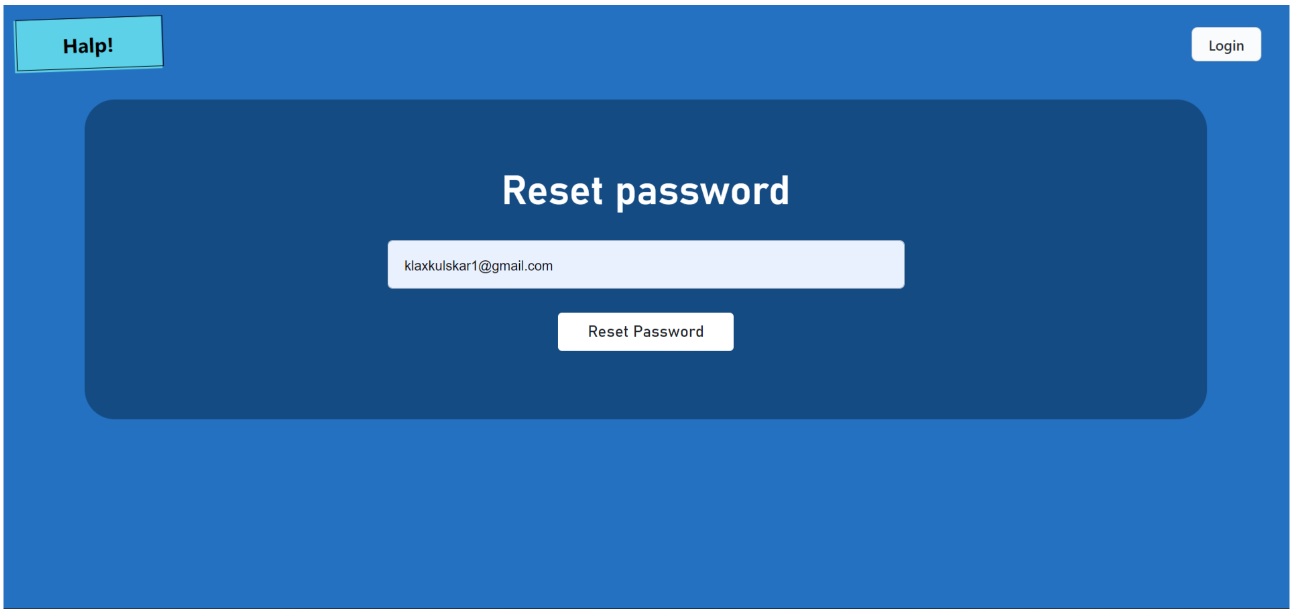


Figure 31: Page for users having forgotten their password

If the user has forgotten their password they can press the login button followed by the forgot your password button to get directed to the forgotten password page. At the change password page the user can enter their email. After filling in their email and pressing the reset password button they will get an email with a confirmation email. Pressing the link in the email takes you to the reset password page.

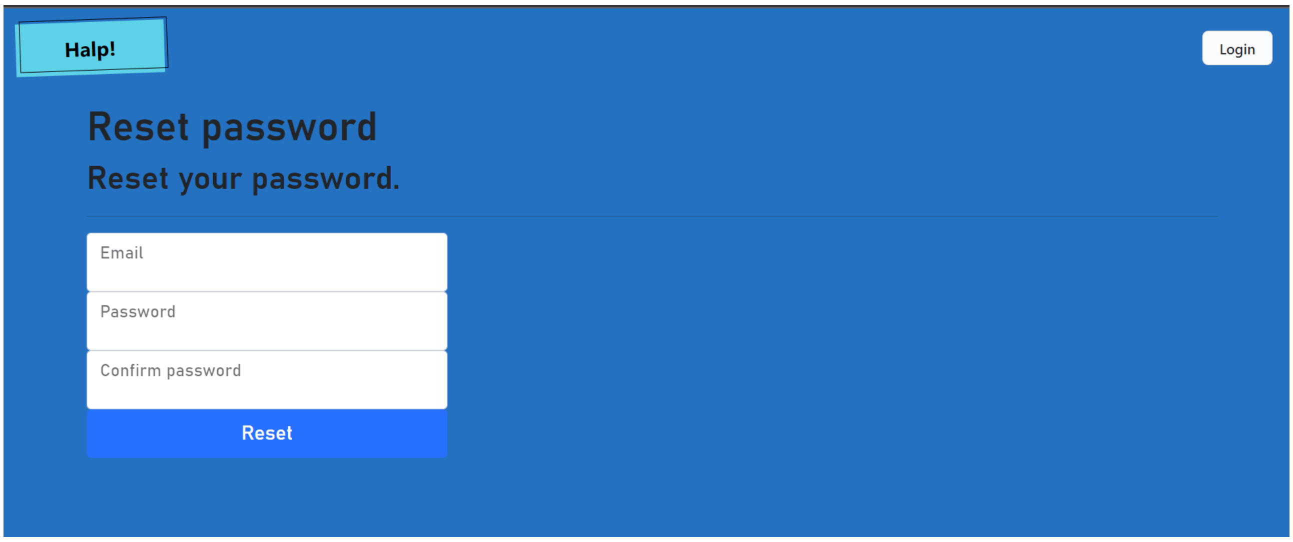


Figure 32: Page for users to reset their password

At the reset password page the user can fill in their email, their desired new password, and their new password again to confirm it. After pressing the reset button their password will be updated, and they can log in again with their new password.

6.6.3 External login

If a user want to add their discord login they can go under settings and press external login button. Here they will have a possibility to add the discord login. You will be redirected to discord when you press on "Discord".

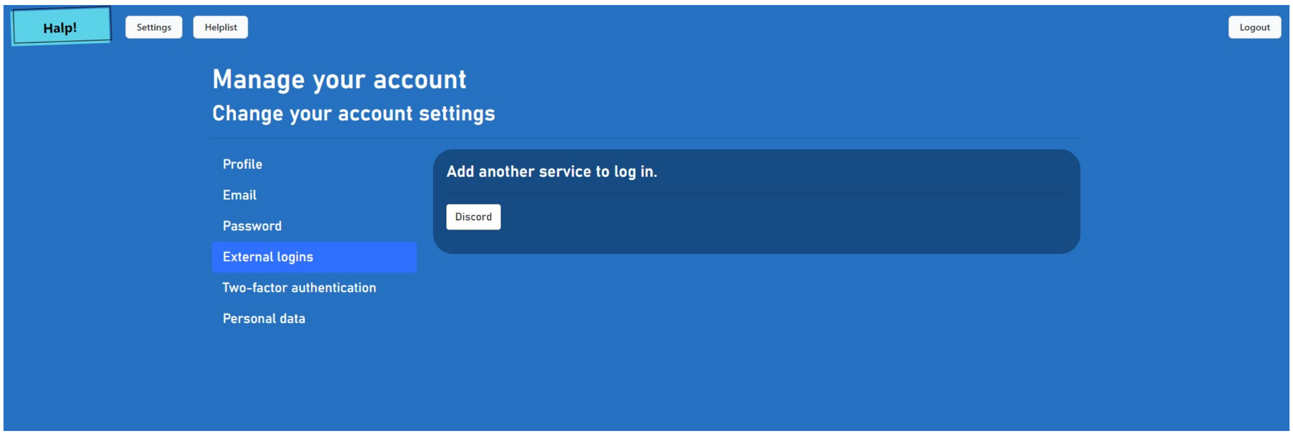
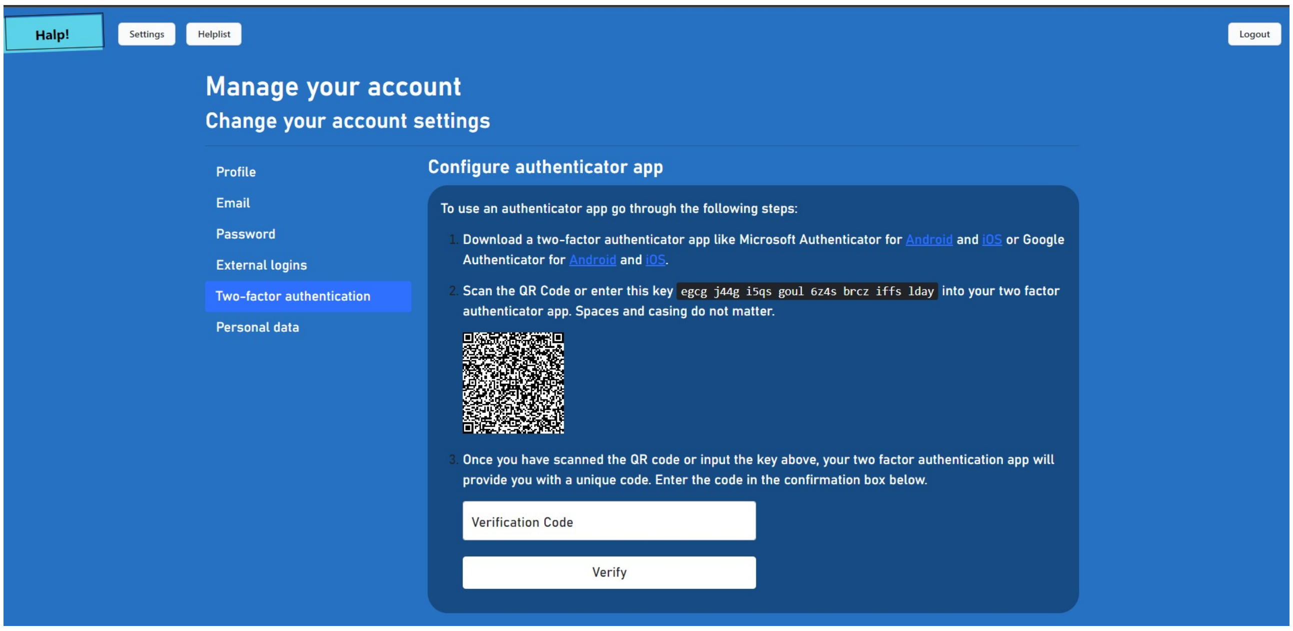


Figure 33: Page for external login with discord

6.6.4 Two factor authentication

On this page there is the option for the user to set up two factor authentication. When you press the "Two-factor authentication" button, you will get a choice to either reset or add the authentication. When you press "set up authentication app" you will get a picture with how you can set it up for your account.



To use an authenticator app go through the following steps:

1. Download a two-factor authenticator app like Microsoft Authenticator for Android and iOS or Google Authenticator for Android and iOS.
2. Scan the QR Code or enter this key `egcg j44g i5qs goul 6z4s brcz iffs 1day` into your two factor authenticator app. Spaces and casing do not matter.
3. Once you have scanned the QR code or input the key above, your two factor authentication app will provide you with a unique code. Enter the code in the confirmation box below.

Figure 34: Page for set up the two factor authentication

7 Implementation

7.1 Database design

The core model is based on the ApplicationUser, which is created for the users and contains the attributes that came with the Identity Framework as well as discord tag and nickname. Between the ApplicationUser and the student assistant, there is a one-to-many relationship, so that a student assistant can attend many courses without having a lot of duplicates in the ApplicationUser data. The student assistant class contains the id of the student assistant and the course he attends.
The help list and ApplicationUser have a one-to-many relationship so that the help list can have many ApplicationUsers, but an ApplicationUser can only attend one Help list at a time. This decision was made because a student can't be at two places at a specific time. The Help list contains the needed data for the lab purposes.
Courses and course links were all fetched from Time-edit and did not need to be connected to any other classes. The course link and course model are just classes that contain the course links and courses that the admin will add to the list at the beginning of the year, and assign the student assistants to where they will attend.
The role class that came built-in with Identity Framework contained the different roles that were needed. We made a Student assistant role; "student assistant" and an "admin" role to take care of the different authorization tasks.

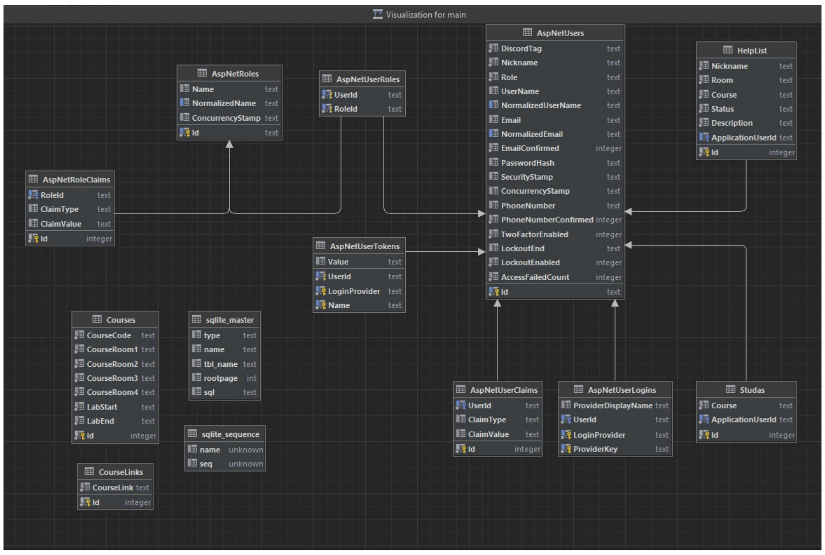


Figure 35: Database design

7.2 Web design

We started off by implementing the primary design and common components in a shared CSS file. The background, text color, font size, and font got changed and a component named "box" was created. Next, we set some guidelines for how to set up a responsive page using the grid and made everyone responsible to meet the design requirements (making every site look similar to the design).

7.3 Application flow

The below figure shows the finished flow of the program. The diagram shows how the different users can move through the system.

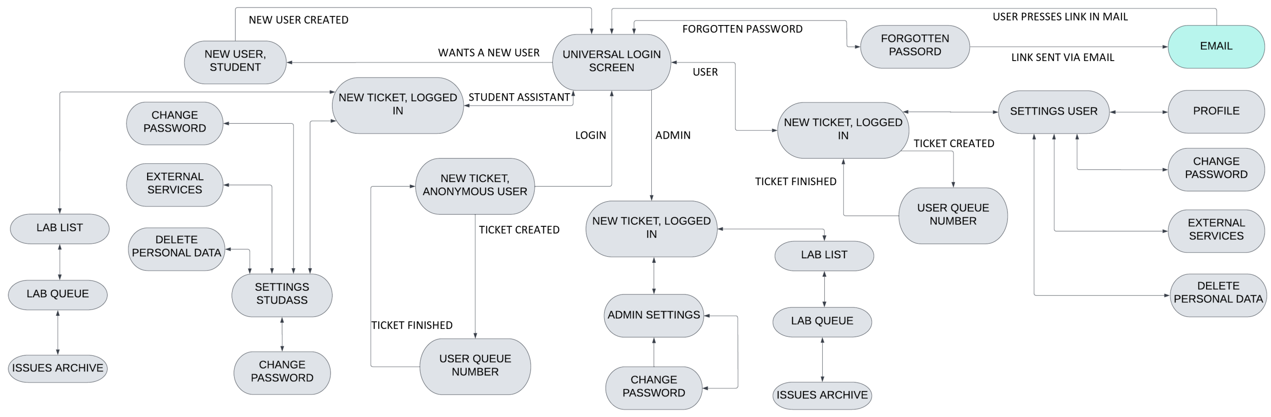


Figure 36: Flowchart showing the current flow design of the system

7.4 ASP.NET scaffolding

We have used ASP.NET scaffolding to automatically generate template files that already exists in the project but are hidden for us. We have generated everything under identity and login partial. This saves us time as we can alter already existing functionality without having to code everything from scratch.

7.5 Email-verification

The email-verification process is implemented by configuring an email provider. The email provider service used in the program is called SendGrid. To implement this email provider into the program we needed a SendGrid account and a key, then get an API that could be used in the code. The SendGrid service works by setting up a custom email that is used to send verification emails in collaboration with the API.

7.6 Discord-login

We implemented the functionality to login with discord by using the official Discord API. This API requires that we register our application through Discord to obtain a Client ID and a Secret ID. These ID's were then used through the ASP.NET OAuth2 Library [10]. Whenever the user would prompt a login with discord, we redirect the user to a Discord portal which then routes back to our application along with the users information in a JSON format which we then parsed to create the new user.

7.7 Settings

7.7.1 Retrieval of TimeEdit data

Data from TimeEdit is retrieved by first replacing the filename ending in the provided link from .html to .ics. This would let us download an iCalendar file, which we could then parse. The data is then parsed using iCal.NET, an open-source library for parsing Icalendar files [17]. The data is further edited in the system, to only get the lab times, and make sure that the data is gathered correctly. This data would then be inserted into a table in the database, where it is possible to retrieve it for use in the system. The retrieval of the courses is a service set to run at a certain interval, more specifically every 24 hours. This approach makes it possible to delete the whole table, and overwrite it with the new courses every day.

7.8 Lists

The "Lists" page lets the student assistant and admin chose what help list to access. The student assistant is limited to the courses he has been assigned as an assistant, while the admin can access every help list in the system. This limiting is done on the server side, and a student assistant will not have access to courses he is not assigned to, even if you type in the correct URL.

7.9 Help list

The implementation of the help list for the student assistants role is so that there is a list of the classes that a particular user is a student assistant in. And then they can view the site for the class they are going to work in. When they enter the site they will get a list with the current tickets for the class. They have a button to archive completed tickets, as well as restore archived tickets back to the original list. Each ticket has an ID, a Nickname, a Room, a Course, a Status and a description. Additionally, the row contains a button to send the ticket into the archive when it has been resolved.

The page communicates with the server using SignalR with grouping by courses, which provides live updates. Additionally, when the page is loaded the data is loaded directly from the controller using Blazor. When a new ticket is registered in the system, the helplist is automatically updated using SignalR. Sending a ticket into the archive is also done via the server using SignalR.

7.10 Archive list

The archive works similar to the help list but only contains archived tickets. These tickets can be unarchived to be placed in the original list where they came from. Active tickets can be archived by either the student assistant or the student who owns the ticket, and will thus be marked as "Finished" in the system. As in the help list, each row in the archive contains a ticket, and a button. In the archive, this button is labeled "Unarchive", and pressing it will unarchive the current ticket, and send it back to the help list. The student that made the ticket can also cancel the ticket, which will mark the ticket as "Removed". Tickets that are canceled by students can not be unarchived by the student assistant, and thus the "Unarchive" button is greyed out. The "Archive" page works using the same communication principles as the help list, by loading the page using Blazor, and loads live content using SignalR.

7.11 Tickets

7.11.1 Create ticket

The create ticket page was implemented using razor pages, where we have the view and controller working together for the page. In the view page, we wrote HTML code for styling the page to our liking. We also wrote HTML code for all the different text boxes, as well as the button for submitting the ticket. Lastly, we connected this page to various scripts written in Javascript. We set up this page to use ApplicationUser SignInManager to access information about the logged-in user. This can for example be used to fill out the name box of the logged-in user automatically. When we implemented the controller part of the page we connect the page to both the database, as well as the hub. Connecting to the database gives us the ability to update the database while connecting to the hub lets us update the help list in real-time. Moreover, the controller handles the post request when the user creates a ticket.

7.11.2 Edit ticket

The edit ticket page was implemented in a similar way to the create ticket page by using razor pages. We also wrote HTML code here to style the page to our liking. On the edit page, the values of all the text boxes will be filled with the original data from when the ticket was created. We get these values directly through the database. The id of the ticket we get through the usage of cookies. We use various scripts for JavaScript in the edit page as well. The controller part of the edit ticket page is done by and large identically as in the create a ticket controller. Furthermore, the edit controller handles the post request of the edit page. Here we also connect to the database and the hub, so we can update these when we submit our edit post request.

7.12 Queue

The queue page was implemented using razor pages. We wrote HTML code in the view page to style the page to our liking. In the queue view, we display the name, which is retrieved through the usage of the hub and help list in the create a ticket page. Furthermore, we display the real-time position of the user in the queue. This is achieved through using the signalr library. The view page is also coded to have multiple scripts running Javascript. In the controller for the queue, we connect to both the database and the hub. We connect to the hub so we are able to update the position of the user in the queue in real time. This is done through a count variable in the controller that gets sent to a Javascript script which is then sent to and updates the view.

7.13 Application security

7.13.1 User Roles

The system is implemented with a ASP .NET's user roles approach. With this approach a user is assigned a role in the system, which comes with different levels of functionality.

- **The Anonymous user** is a user that has not logged in, and therefore does not have access to the settings. The settings only provide capabilities that a logged in user should be able to have. The anonymous user is categorized as lowest on the list of privileges in the system.
- **The logged in user** has access to the user settings, and all the user pages. This includes every page that is not the administrator settings, or the pages containing any lists. The user is not added to a role in the system, and does not have any special privileges.
- **The student assistant** is given permission to access the help list, as well as the archive. The student assistant is given the "Student assistant" role in the ASP .NET UserManager. The student assistant will only have access to the help list in the courses he is assigned to. The only privileges that the student assistant is missing, is the ability to access the advanced settings screen.
- **The admin** is the highest elevated user, and thus has no restrictions as to what he can access. The admin will have access to the help list and archive in all courses, and has the ability to add new courses and assign roles to users through the advanced settings. The admin is given the "Admin" role in the ASP .NET UserManager

7.13.2 Password handling

Handling of passwords is done through ASP .NET or Discord, whichever the user has registered with. In ASP .NET the passwords are hashed, and inserted into the database when the user is created. [11]. If the user logs in with Discord, there is no password generated, and the user can not log in using the normal E-mail & password method.

7.13.3 Two-factor authentication

In order to offer more security to the user, we used the built-in two-factor authenticator service and added an auto-generated QR code using the "QRCode for JavaScript" library [5]. More about this library can be found under "Technical Background". This will allow the user to add 2FA using an authenticator app like Google Authenticator and require a code to log in. The user can opt to not require the code on the same device on future logins.

8 Testing and validation

8.1 Tests regarding "must have" requirements

1.

Test requirements	
Requirement id:	Req-1
Requirements description:	As a student assistant, I must be able to log in as a student assistant.
Pre-condition(s)	The student assistant must have an existing user ID and password registered.
Test-steps	• Presses login button • Enters email address • Enters password • Presses login button
Expected result	The registered user should be able to login and will be redirected to the creating a new ticket page.
Actual result	User is navigated to the create new ticket page, with successfully login
Status	Pass

2.

Test requirements	
Requirement id:	Req-2
Requirements description:	As a student assistant, I must be able to mark tickets as resolved.
Pre-condition(s)	- There must some ticket to be resolved - Must be in the right Lab queue
Test-steps	• The assistant presses "Archive" button • The ticket gets removed from the queue
Expected result	The student assistant should see that the ticket has been removed from the list
Actual result	Ticket got removed from the queue
Status	Pass

3.

Test requirements	
Requirement id:	Req-3
Requirements description:	As a student assistant, I must be able to assign tickets
Pre-condition(s)	- There must some ticket to be resolved - Must be in the right Lab queue
Test-steps	• "Student assistant presses "assign" button
Expected result	The student assistant should see that the ticket is assign to you
Actual result	The feature does not exist
Status	Failed

4.

Test requirements	
Requirement id:	Req-4
Requirements description:	As a user, I must have the ability to create a ticket
Pre-condition(s)	- Have entered the website
Test-steps	• Write their name/nickname • Enter a description • Choose the right room • press "send" button
Expected result	The user should be redirected to the queue, and the ticket has been sent
Actual result	Entered the queue
Status	Pass

5.

Test requirements	
Requirement id:	Req-5
Requirements description:	As a user,I must have the ability to view where I'm in the lab queue after writing the ticket
Pre-condition(s)	- Have wrote a sucsessfully ticket and sent it
Test-steps	• User has pressed the "send" button on create new ticket • Entered the queue
Expected result	The user should be redirected to the queue, with the right queue number and with the ticket sent
Actual result	Entered the queue in the right position
Status	Pass

6.

Test requirements	
Requirement id:	Req-6
Requirements description:	As a user, I must have the ability to log in if I want to
Pre-condition(s)	The student must have an existing user and password registered
Test-steps	• User enters the correct email address • User enters the correct password
Expected result	The user should be successfully logged in The user should be redirected to the create new ticket The user should have their nickname should already be filled
Actual result	Entered the queue in the right position
Status	Pass

7.

Test requirements	
Requirement id:	Req-7,A,B,C
Requirements description:	As a user, I must have the ability to know: What labs do I have What rooms the labs are in What time the labs are
Pre-condition(s)	The user must have an account and be logged in
Test-steps	• Press "my lab courses" button
Expected result	Should show a page with information about the labs that the user has
Actual result	No page found, not a feature implemented
Status	Failed

8.

Test requirements	
Requirement id:	Req-8
Requirements description:	As a user I must be able to register an account via email:
Pre-condition(s)	None.
Test-steps	• Presses login button • Presses register as a new user button • Enters email address, nickname, discord tag, password and confirmed password • Presses register button
Expected result	The user should be successfully registered and redirected to the registration confirmation page.
Actual result	The user is navigated to the registration confirmation page with a successful register.
Status	Pass

9.

Test requirements	
Requirement id:	Req-8-B
Requirements description:	As a user, I must be able to register an account through discord:
Pre-condition(s)	Must have a discord account. Must be logged in discord
Test-steps	• Press the discord icon • Press "Authorize" • "Enter nickname" and press the "register" button
Expected result	The user should be redirected to the create a new ticket with their new account, and the user should be stored in the database
Actual result	The user gets redirected to create a new ticket, and got stored in the DB.
Status	Pass

10.

Test requirements	
Requirement id:	Req-9
Requirements description:	As a user, I must be able to confirm my email after registering:
Pre-condition(s)	Has entered the registration information and pressed register
Test-steps	• Open personal mail • Press "clicking here" link
Expected result	The user should be redirected to email confirmation, and get the message that the mail has been confirmed.
Actual result	The user gets redirected and got the email confirmed
Status	Pass

9 Discussion

9.1 Process

9.1.1 Planning

Early on, we set guidelines for the product. How it would look, what functionality it would have, and our wishes. In hindsight, we were not good at setting requirements. We are missing, among other things, requirements to deactivate tickets for student assistants, set roles, and distribute classes to the different student assistants.

However, we were very good at having a common understanding of how the product would look and how the flow would be. The reason for this was probably that we created a draft of the design early in the phase.

9.1.2 Git

The collaboration using git has worked very well. In the beginning, we all sat down and agreed on how git, Bitbucket, the git server would be used. In developing the system we have used git diligently, and always made sure to follow the structure mentioned in the system's README file.

Not all of us had a lot of experience starting out with git, and we have helped each other learn the git system.

9.1.3 Challenges in the process

Though it worked well, it is still a challenge to work in such a big group. It was not always easy to find the time that we all could be together, and it was not always easy to know if everyone was on board with everything that went on. But it in the end it worked out good, and it was a good learning experience to know how to organize to get the job done.

9.2 Product

9.2.1 Achievements

Our initial product vision was to create an up-to-date program that would allow student assistants and students to interact as smoothly as possible. We wanted to revamp the current method of using spreadsheets as the means of organizing student help requests and this is exactly what we have achieved. Our new software *Halp!* streamlines the communication link between student assistants and students as well as offer additional features such as archiving completed tickets, registering as a user to avoid entering duplicate data and providing a much more friendly interface to all its users.

As members of the development team, we have individually achieved a greater understanding of how teams work in conjunction with each other, how tools such as Jira, Bitbucket, Git, etc. can be used in a development setting and how to properly document the development process.

9.2.2 Improvements

Improvements we thought about of everything we have finished, is the update of Time-edit data. As for now, the refresh of Time-edit happens every 24 hour when the program starts. The better solution we can think of is to refresh Time-edit at 00:00 O'clock.

9.2.3 Incompletions of requirements

We still have some implementations we could do to make the product more useful than just creating tickets and resolving them. We wanted to first focus on the most crucial parts of the product to make it do what is most needed, and then if there were time, we could introduce other elements on the web application.

- **The "must have" incompletions**
  We figured that the must-have for users/students to have the possibility to watch the different courses they could attend was more of a "quality of life" feature, so we just did the implementation of registering a new user, but decided that it was not the most important requirement and downgraded the priority on that one. The same went for the assigned ticket to a student assistant, and was not made under the development process.

- **Should, can have** We still have some implementations we could do to make the product more useful than just creating tickets and resolving them. The idea we had was to have some possibilities to have statistics on the labs and the student assistant so that the teachers can see how much the labs are used.

- **"Quality of life features"**
  We also thought that it would be nice to have the possibility to watch the queue from discord and get help through discord voice communication.

9.2.4 Finished Design

As mentioned in section 4, the overall design of the project is meant to be as intuitive and simplistic as possible. We had to make some changes along the way to the design, both because some things were missing initially, but we also added some functional and artistic changes. This chapter discusses the changes we made to the finished design, compared to what was initially planned. Images of the initial design can be found under the section, and the implemented design can be found under the section if you want a reference while reading.

- **The "login" page**
  We initially designed multiple login screens, one for the user, one for the student assistant, and one for the admin. We quickly decided against this, as it would be easier and more intuitive to just have a single page. After all, the student assistant and admin do not have any additional fields they have to fill out to be able to log in.

  The login page was initially designed without the ability to log in using external services, as this was not thought about in the design process. The finished design has been fitted with a button to log in using Discord, as we later in the process opted to include this external login method.

  In the process, we figured out that the requirements [Req-6] were not that crucial for our product to work properly. So we decided that we wanted to implement the possibility for a person to create a user and log in, to facilitate [Req-9, A, B, C]. We also know that if these requirements for the users should be a possibility, we have to implement a new class in the database that can store the user's courses.
- **The "New User" page**
  The initially designed "New User" page was missing some fields, both optional and required

to create a new user. After adding Discord as a sign-in option, we made it possible for the user to add their Discord username when they create a new user. We also added a field where they will have to confirm their password.

- **The "Reset Password" page**
  This page was initially designed to also contain a textbox, with instructions to type in the email address to receive a reset link via email. This textbox was not included in the finished design.
- **The "Create new ticket" page**
  This page ended up being very similar to the original design, but the layout of the contents were changed somewhat.
- **The "Settings" page**
  We had some discussion as to who should be able to add more Time-Edit links and landed on that that is a task for the administrators. This means that only the admin can add more courses to the system.

9.2.5 Design Incompletions

There are some pages that still are yet to be redesigned. These pages are evaluated as minor, and thus are not the ones mostly used by the users of the system. In a finished system, they should of course be redesigned to fit in with the rest of the design. These pages now use the layout automatically created by the ASP .NET Framework, but with the universal blue coloring scheme. The pages that are yet to be redesigned are:

Login
The login page works very well when resized to a smaller screen. The content moves and resizes exactly as planned, and the page is still as intuitive as before the resizing.

Settings
If logged in as a user or student assistant, the "Settings" page has no trouble resizing, because it only consists of buttons, that are not technically hard to resize.

If logged in as an administrator, the navigation bar will resize itself more than it should when the browser window is resized. This makes the bar a little narrower than it should be. Although this looks a bit weird, it does not actually reduce the usability of the navigation bar all that much.

The "Time-Edit" section of the admin settings consists of a table where the admin can add new links, as well as a table that shows what links are in the system. This table also contains a "Remove" button, and while the window is resized this button ends up being hidden, with the possibility to slide it back into view. As well as making the page less intuitive, the design is broken by removing the alignment between the "Add" and the "Remove" buttons, which no longer aligned correctly. This design could pass as being usable, although it is not at all optimal for a mobile user.

The "Roles" section of the admin consists of two tables placed beside each other that move to be placed on top of each other when the window is resized. This results in them being taller than the actual window, and also the background box. This view is too tall to fit on the mobile screen, and also does not look right. The "Roles" section is probably the view that would require the most work to pass as the design we want to have.

Help list and archive The help-list and archive resize nicely to fit the screen width. All buttons move as intended, and the tables are compacted without the need for any vertical scrolling. This makes it pretty easy for student assistants to use their phones to see who is next in the queue, although most use their computers to do this.

Generic error page The generic error page resizes nicely, as it does not contain any advanced elements that struggle with flexing. This page would therefore pass very nicely as being mobile-friendly.

9.3 Design anomalies

All implemented pages were initially planned, except for the page that shows the student assistants accessible help lists. The choice to insert this page was made through the discussion that the admin would need a list to chose courses from. Initially, the plan was to give the student assistant access to the correct help list based on the time, making it possible to automatically enter the correct list. To be sure that the student assistant would have access even if the time was changed, we decided to give this role access to this page as well.

9.4 The mobile application

In the beginning, we planned to create an application in addition to the web page. The application was planned to be created using Flutter to be cross-platform between Android and iOS and would provide the user with the same interface as the web page. This would be done by creating a simple web application, an app that ran a browser window connected to the correct URL. We, unfortunately, did not have time to create this app, and as stated earlier, the system was also not ready to perform on such a small screen.

9.5 Application Integration

As we designed the front end, we always had in mind that the project should work on mobile as well. This is why all the pages are made flexible using the CSS "flex" tag, through Bootstrap. This makes the page's content able to resize and move appropriately when the screen is resized.

We did not have the time to perfect this, so the views are not optimal when scaled too much. The advanced admin settings page is currently not very good at scaling down, which is why it would be recommended that the admin uses the system through a computer browser. Pages that require additional work, as well as pages that do work on the small screen, are listed below.

9.6 Implementation

9.6.1 Tickets

The ticket system implementation went well. We managed to access the database in order to get information about the user to auto fill the nickname of logged-in users as well as write the newly created tickets. There are only 2-3 things for the user to fill out, and that makes it very intuitive to use. It was important to us that the create ticket page is intuitive and easily understandable as we did not want to make it more complicated than the already old system. One problem with the create ticket system is that if the user enters a room that does not exist or if a lab has multiple rooms where only one/some of the rooms are implemented in our database, the user will get an error.

In the edit ticket we originally wanted to avoid appending the ticket id to the url when editing a ticket by using cookies. This worked fairly well, however, due to each browser only being able to store one ticket at a time, we encountered some problems where if the user would make a new ticket and open another tab to create another one, the old one would be replaced in the browser but not in the database. This way the student assistant would see the ticket, but not the student. We tried to circumvent this by adding a check if the browser contained a ticket in a cookie before displaying the create ticket page, but this is a problem because we reset the database each morning. And if the browser's cookie contains a ticket, but the database doesn't, the program crashes. So, because no sensitive information can be obtained by manipulating the url to access another ticket, we just rolled the program back to use url-based ids.

Overall, we are happy with how the ticket system turned out. We managed to fulfill all requirements related to tickets (REQ-4, REQ-5 (by redirecting), REQ-11 & REQ-12).

9.6.2 Queue

We initially encountered some difficulties as we tried to implement the queue. Getting the position of the user in the queue to update in real time without having to restart the website was challenging. Eventually, through trial and error we figured it out and we are satisfied with the result.

9.7 Satisfactions

Overall, we are pretty happy with what we have accomplished with the project. We set out determined to learn about working in a team environment, as well as developing software with professional tools. The overall system works with most of the "must have" requirement, which actually makes the system usable. We are very happy with the design, and the fact that it works just as well on the small screen as it does on the big screen.

One of our main goals of this project, was not to create software that would be put on a shelf after the project period was over, but to create software that could be used in an actual environment. We feel like our software is indeed ready to be put to use in a lab environment and thus we are very satisfied with how our software turned out.

9.7.1 Report writing

The writing of the report has gone down fairly well. We realize we could have been better at writing the report while developing, which would have helped us in the end. Some of us took notes from the meetings, and also while developing, and this helped us get a quick start

9.8 Challenges

During the development process, we have encountered some setbacks, which of course is also a part of the process. These problems are mentioned below. Bigger problems you have had, solutions, and eventual lack of solutions.

9.8.1 Approach for retrieval of TimeEdit data

As stated in the requirements, we wanted to deliver a product capable of getting data from Time-Edit, which is the time planner used at the University of Agder. Time-Edit contains all data about all subjects at the university and would make it possible for us to automate the process of getting data about classrooms and time for the specific labs. The Time-Edit API was not easily accessible, so we decided to use the .ics file that can be downloaded after adding the desired subjects to the website. This would then provide us with a file containing all the subjects, their start and end time, the lecturer, and more. We considered this an adequate approach for our purpose. Initially, the plan was to use python to gather this information, but we later learned that this approach was not easily integrated into .NET, as the framework is compiled into a .exe file with surrounding .dll files. We, therefore, had a little look around and found that the best solution was to retrieve and process the data from Time-Edit using native .NET.

9.8.2 Problems with git rebase

We first decided to go for the rebase merging method, but we quickly decided to go back to normal merging because we had too many conflicts when we tried to do it this way. The problem we had, was that Bitbucket would not allow us to merge the newly created branch after we had used git rebase. We are still not sure why this happened, as we did not want to spend too much time figuring this out. Because of this, the commits to git are not in a systemized order, and this makes the git log less clear. We evaluated the situation such that we opted to focus on creating the application, as this was the most important task.

9.8.3 Database design

Thinking about how we wanted our database was one of our first objectives when we started our project. The design of the database was not clear to us at the very first moment, because we did not know how and how well we could fetch our data from Timeedit. The approach we chose is a "Code first approach", which means that we started designing code before we started designing the database. The other connection we did between the users

9.8.4 HTTP Requests vs SignalR

We had some problems with POST requests, and used a lot of time on troubleshooting. In the process, we opted to use SignalR in some places where POST requests would have been better. The solution to the problem was of course extremely easy: We forgot to include the line "@Html.AntiForgeryToken()" in the HTML form. After finding the problem with the requests, we used some time to reevaluate and implement HTTP requests where it would be a better fit than SignalR.

9.8.5 External logins

We had some problems when implementing the login with discord functionality. Because we used a template that had built in functionality for logging in/registration, we had to generate the files using the scaffolding tool and undestand where to modify the code to accept the data sent by discord. This took alot of time and effort. There is little documentation on using the discord API with MVC/razorpages in ASP.NET.

9.9 Future work

In this section we discuss functionality that could be added in future versions.

9.9.1 Docker support

We initially wanted to add docker support to the application, to let system administrators easily spin the application up without any hassle. Docker would make it easy to run and maintain the program on the system server, and the administrator would be able to start the system with a single command. If we had more time, then maybe we would be able to get this off the ground.

9.9.2 More functionality for registered users

We would like to add more functionality to an already registered user than we currently have. Specialized accounts like student assistant, or admin accounts have special functionality when compared to a visitor. At the moment the only difference between a visitor and a logged-in user is that the logged-in user's name will automatically be filled in on the create a ticket page. We would like to add functionalities like adding favorite courses, rating student assistants, viewing older tickets, etc.

9.9.3 Mobile application

Adding a mobile application version of our website was one of our key ideas in the planning phase of the program. This idea was scrapped because of time constraints. In the future, we would like to implement an application that connects to the same service as our website. The mobile application would have the same services as our website, and we would like it to look as similar as possible to the website. We would very much like to get the Android / iOS application off the ground, to let the users easily create tickets on the fly.

9.9.4 Digital lab help

We would like to add functionality for digital help. Some students might not be able to meet up to a physical lab for a number of reasons for example illness. We would like to expand our system to include a way for student assistants to help students at home. This could be done by creating a custom chat service between the student at home, and the student assistant within the website, or through a connection with an external chat service like discord. We would like the students in the digital lab and the students in the physical lab to be placed in the same queue to make the integration as seamless as possible.

10 Conclusion

Our task was to make a digital ticket system for lab hours at our university. HALP is currently at a prototype stage, where only the crucial parts have been implemented and it is far from a finished product. Our solution covers the parts for an admin, student assistant, and users to use the web application, but still has some missing features for it to be a finished product. Compared to other products ours is a much simpler solution and differs from the other products because of our use of Time-edit integration. Because our schools current excel system is sub-par, we think that HALP could make it easier for the student assistants to have a good overview. It also makes it easy for students to write tickets to get help, and makes it easy to track where they are in the queue. This product satisfied our main goals, but we still want students/users that are logged in to have more features like; favorite courses, support for storing data for logging statistics, so that teachers can see how the attendance at each lab is.

References

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

[2] Atlassian. *What is version control?* URL: https://www.atlassian.com/git/tutorials/what-is-version-control.

[3] Bitbucket. *A brief overview of Bitbucket.* URL: https://bitbucket.org/product/guides/getting-started/overview#a-brief-overview-of-bitbucket.

[4] Bootstrap. *Get started with Bootstrap.* URL: https://getbootstrap.com/docs/5.2/getting-started/introduction/.

[5] davidshimjs. *QRCode for Javascript.* URL: https://davidshimjs.github.io/qrcodejs/.

[6] Frieze News Desk. *Blue is the Most Relaxing Colour, Scientists Say.* URL: https://www.frieze.com/article/blue-most-relaxing-colour-scientists-say.

[7] Brady Gaster. *Overview of ASP.NET Core SignalR.* URL: https://learn.microsoft.com/nb-no/aspnet/core/signalr/introduction?WT.mc_id=dotnet-35129-website&view=aspnetcore-7.0.

[8] Lucid Software Inc. *Intelligent Diagramming.* URL: https://www.lucidchart.com/pages/.

[9] Kahoot! *Kahoot!* URL: https://kahoot.it/.

[10] Kevinchalet. *AspNet.Security.OAuth.Providers.* URL: https://github.com/aspnet-contrib/AspNet.Security.OAuth.Providers.

[11] Microsoft. *Best practices for deploying passwords and other sensitive data to ASP.NET and Azure App Service.* URL: https://learn.microsoft.com/en-us/aspnet/identity/overview/features-api/best-practices-for-deploying-passwords-and-other-sensitive-data-to-aspnet-and-azure.

[12] Microsoft. *Entity Framework Core.* URL: https://learn.microsoft.com/en-us/ef/core/.

[13] Microsoft. *Introduction to Identity on ASP.NET Core.* URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio.

[14] Microsoft. *Introduction to Razor Pages in ASP.NET Core.* URL: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio.

[15] Microsoft. *What is ASP.NET.* URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet.

[16] ASP.NET Core Blazor / Microsoft. *Microsoft.* URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0.

[17] Rian Stockbower. *iCal.NET.* library. https://github.com/rianjs/ical.net.

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

[19] Zendesk. *Help desk ticketing system.* URL: https://www.zendesk.com/help-desk-software/ticketing-system/.

GRUPPEKONTRAKT

PROSJEKT HØST 2022

IKT201

1. Hvem er med i gruppen (navn, telefon, e-post)?

Horpestad, Sondre sondre-horpestad@hotmail.com 95304992
Thorjussen, Charlotte Celine charlotte.thorjussen@hotmail.com 41319579
Eidsheim, Nikolai Bunch nikolai.eidsheim@gmail.com 94032639
Husebø, Sivert sivertespelandhusebo@gmail.com 46855244
Hagli, Markus mhwan88@gmail.com 41546604
Øvreås, Ole-Johan oleelo123@hotmail.no 47702712

2. Hva er målene for gruppen?

Utføre prosjektet slik at alle i gruppen er fornøyd med det, samt lære og jobbe sammen som en gruppe i programmering og utvikle seg faglig.
Få best mulig karakter

3. Hva forventes av det enkelte gruppemedlemmet?

Alle bidrar og gjør en innsats.
Alle holder seg oppdatert og kommuniserer slik at resten kan få oversikt over arbeidet hver enkelt gjør.
Spør om det er noe!
Det forventes også at man jobber i bestemte tidspunkt og møter fysisk opp til møter.

4. Regler for oppførsel og samarbeid: Hvordan skal vi ha det i gruppa vår? Hva er god oppførsel og hva er ikke?

1. Møte presis til avtalt tidspunkt. Respekt for andres tid.
2. Forfall eller forsinkelser meldes på discord til **gruppen**
3. Vi deler eventuelle kostnader.
4. Plan og tidspunkt for **neste** samling gjennomgås
5. Taushetsløfte innad i gruppa, dette gjelder både arbeidsmessig og privat informasjon som måtte komme frem under samarbeidet i gruppa
6. Vi respekterer hverandre og tar vare på hverandre
7. Alle må skrive under gruppekontakt

5. Hvordan kommuniserer vi med hverandre?

Møter fysisk eller på discord.

6. Fremdriftsplan – definer tids- og ressursrammen for prosjektet

Innleveringsfrist 14. desember kl. 14:00
Timer per uke blir bestemt underveis samt hvilken dager

7. Konflikthåndtering

Hvis en eller flere opplever noe som ugreit, ta det opp med gruppen tidlig! Mange konflikter kan løses på en god måte om man tar opp det som er ugreit tidlig, før ting utvikler og eskalerer. Gi den det gjelder en ny sjanse – en sjanse til å endre seg!

Dersom ting ikke blir løst på en alminnelig måte, gjelder følgende regler:

§ 1-1 Hvis gruppemedlem(mer) *gjentatte ganger* bryter et eller flere punkter gruppekontrakten, skal gruppen sende en skriftlig advarsel per e-post til den eller de det gjelder. Advarsel må gis før 23. november 2022. Alle gruppemedlemmer må være enig i at det sendes advarsel, og signere den.

§ 1-2 Hvis gruppemedlem(er) etter skriftlig advarsel fortsatt bryter gruppekontrakten, kan gruppen sende en oppsigelse til gruppemedlemmet. Alle gruppemedlemmer må være enig i oppsigelsen, og signere den.

§ 1-3 Hvis et enkelt gruppemedlem er den eneste som oppfyller kontrakten (gjør alt arbeid alene uten hjelp fra gruppen), kan gruppemedlemmet sende et varsel til gruppen. Dersom situasjonen vedvarer etter sendt varsel, kan gruppemedlemmet sende en skriftlig oppsigelse med kopi til veileder og forlate gruppen.

§ 2-1 Ved oppsigelse og gruppen blir delt, anses alt arbeid gjort frem til tidspunktet for oppsigelse som felleseie, og kan fritt brukes videre av alle gruppemedlemmer.

9. Dato og signatur for alle gruppemedlemmer

04.10.22 Ole-Johan Øvreås
04.10.22 [illegible]
04.10.22 Nikolai Eidsheim
04.10.22 Markus Hagli
04.10.22 Charlotte C. Thorjussen
04.10.22 Sivert E. Husebø