

# TCS House Management System

Design Project

April 2022

Group 1:

Brand Hauser - s2234823

Fieke Middelraad – s2297833

Theodor-Fabian Niculae – s2264323

Martin Stoev - s2392593

Katja Wevers - s2298155

Supervised by Yeray Barrios Fleitas

## Abstract

The TCS House Management application was developed to provide a clear and easily accessible overview of the division of Computer Science students into houses. Due to significant growth in enrollments, these houses are necessary for the administration of the bachelor program at the University of Twente. Users of various roles require access to the data contained within the system, to perform the duties specific to their position within the university. One of which is that of the teachers who use this division for the scheduling of classrooms and group formations. The responsibilities of study advisors toward the students are also segmented based on the house division. The authors of this report developed the application as a final project for their Bachelor's TCS program.

The process began with the collection of requirements through interviews with prospective users. Many requirements were discussed and documented. Not all of these requirements were selected when outlining the scope of the product. From this basis, the design of the product was conceptualized through flow charts and use case diagrams. This, in turn, lead to the production of the application using frameworks such as Spring Boot, PostgreSQL, Svelte and more to create a user-friendly system for the purpose of managing the houses of the Bachelor TCS program.

Through unit, end-to-end and system tests, the application was verified to function as expected. Furthermore, usability tests were conducted with prospective users to verify that the user interface was designed in such a way that it would be easily understood by those for whom it was designed.

Although desirable features remain to be developed to make the TCS House Management System as useful as it has the potential to be, the end result has been described as a wonderful minimally viable product with added benefits. This report documents the design choices and development process of the application.

# Table of Contents

Abstract.....	1
Introduction .....	4
Problem description.....	5
Requirement specification .....	6
Stakeholders .....	6
Functional requirements - MoSCoW .....	8
Quality requirements.....	10
Risk analysis .....	11
Design.....	13
Conceptualization .....	13
Canvas plugin .....	13
System overview .....	13
Use case diagrams.....	14
Sequence diagram.....	17
Prototype .....	22
Technical details.....	24
Spring Boot.....	24
PostgreSQL .....	24
Hibernate .....	24
LTI Login .....	24
Security through Role-Based Access Control.....	24
RESTful services.....	25
Frontend libraries.....	25
Database class diagram.....	25
Testing.....	27
Unit testing.....	27
Test plan.....	27
Test results.....	27
End-to-end testing .....	27
Test plan.....	27
Test results.....	27
Usability testing .....	28
Test plan.....	28
Test results.....	28

System Testing .....	30
Test plan.....	30
Test results.....	30
Future work.....	31
Conclusion.....	33
Reflection .....	34
Planning .....	34
Task and work division .....	35
Appendix .....	36
Appendix A – Test results test session 1.....	36
Appendix B – Test results test session 2.....	40
Appendix C - System Testing.....	44
User .....	44
Student.....	51
Prefect.....	56
Housekeeper .....	57
Admin .....	60

# Introduction

Over the last few years, the number of first-year students in the Technical Computer Science (TCS) bachelor program at the University of Twente has grown significantly. A system was set up to divide these students into smaller groups, which would provide an easy way to assign students to classrooms, as well as give students a fixed group of people to have classes with. This makes it easier for them to get to know each other. These groups were given the name 'houses', inspired by the popular book series *Harry Potter*. Each house was named after a signature color. This first resulted in the red, yellow, blue, and green houses. In later years, these were joined by orange and violet houses. A teacher was assigned to each house, given the name 'housekeeper', who would act as the head of the house and a contact point for students. As additional points of contact, students in higher years were assigned as 'prefects'.

Currently, keeping track of which house students are in is done using spreadsheets. These spreadsheets also keep track of any requests made by students to change their house. The house division is also used by some teachers to form project groups. Keeping track of these spreadsheets and sending the right information to the right teachers takes a lot of work. That is why the Formal Methods and Tools group at the University of Twente has commissioned this Design Project. It aims to create a system that can keep track of student information and allows for easy viewing and exporting of this information. Additionally, students can use the system to view information about their house and make a request to change the house that they are in.

This project is a continuation of a project made in 2020 during module 4 of the Technical Computer Science bachelor program. The group assigned to that project produced a system that was able to take uploaded student data and use an algorithm to sort the students into houses. Furthermore, it included the ability to view statistics about student information. The system was not put into practice due to lack of functionality regarding the various roles aside from the admin. The original project was built specifically to fit the needs of the Sorting Hat who assigns students to houses. Thus, the project was recommissioned as a design project, allowing more time and expertise to be put into it. This project puts its focus more on managing the student data and the houses, rather than sorting students into houses.

This report documents the design process of this system, starting with a more elaborate problem description and the system requirements, then explaining the design and the decisions made while designing the system. Additionally, the testing process is explained, as well as any possible improvements that could be made in the future, before ending with a conclusion and a reflection.

## Problem description

The TCS houses system is relevant for several stakeholders within the study. This includes students, prefects, housekeepers, module coordinators, teachers, study advisors, the secretary and the “sorting hat”. The so-called sorting hat represents an admin user. An informative overview of these different stakeholders is given in the Requirement Specification, under Stakeholders. All stakeholders require various levels of information on the students and their corresponding houses.

Currently, this information is tracked with various spreadsheets. There is a spreadsheet for student information of different house cohorts. A spreadsheet for student house change requests. There is another one for students who switched houses, for house points, etcetera. This results in a chaotic overview of the information, without the possibility to filter out information depending on a person’s role in the study. Another issue with this approach is that it is quite hard to keep it up to date. All sheets need to be updated regularly, while filtering out doubles, noticing errors and inserting missing information. For example, when a student requests to change houses, they are added to the change request sheet. Next, someone must find their entry in the student information list and change their house. Now the student must be removed from the change request sheet and added to the switched houses sheet. The study advisors must then check this sheet to know which students belong to the respective houses they oversee. Meanwhile, teachers need this information to allocate classrooms properly. The house policy is used to group students together. Assigning the same group of students to the same room for every class is a big part of that. Therefore, teachers need to know who is in which house.

Overall, this process creates a lot of room for error. Requests could be handled doubly or not at all. Information may not be up to date when the users need it, creating problems in their workflow. Apart from that, the current method is not very intuitive or organized for multiple role access, which makes it difficult to use. Furthermore, this process is also has quite a few privacy concerns. Anyone with access to the spreadsheets can view all available data. This means only admin users should have full access, which is inconvenient. The admin has to share data with other stakeholders. Doing this manually means that a small error could give people access to data they were not allowed to have.

Another problem currently lies with the TCS houses competition. This is a new policy that is currently on trial for the computer science program. When students perform exceptionally well on a project, e.g., implementing extra features or handing it in early, teachers and teaching assistants can give points to the house those students belong to. At the end of the year, the house that has accumulated the most points wins the competition. Currently, only the sorting hat knows how many points each house has. Feedback from module coordinators and housekeepers has shown that house points are often forgotten due to the lack of a clear space to assign or keep track of these points.

## Requirement specification

The functional requirements for this project were gathered by interviewing all prospective stakeholders. First, the stakeholders were established, to know who would play a role in using the system. Then, through interviews and follow-up meetings, requirements were set up and regularly updated, resulting in a final list. This list was then prioritized using the MoSCoW methodology. This method separates requirements into four lists based on whether the developers Must, Should, Could, or Won't implement the respective requirements. Besides these functional requirements, quality requirements were also specified, showing goals that are specific to the system's performance and usability.

The division between functional and quality requirements was made to give the developers a clear overview of the requirements. The functional requirements are features that can be implemented one by one. However, the quality requirements need to be considered for every feature. By making a clear distinction, the developers are less likely to overlook any requirements. All functional requirements are written as user stories. This helps to create a good understanding of why users want or need a feature in the software. It also encourages the developers to always view the features from the user's perspective. This makes it easier to implement the feature in a way that is useful for the user. The functional requirements are prioritized by the earlier mentioned MoSCoW methodology. This makes it possible to develop the end product in smaller iterations. It tells the developers which requirements are most important, and which can wait for a later stage to be implemented.

### Stakeholders

Below, table 1 provides an overview of the stakeholders involved in the system and a description of their needs.

Stakeholder	Description
Sorting hat/secretary (Admin users)	The admin users of the system. They are responsible for keeping all the information in the system up to date. Therefore, they want to be able to easily upload new data to the system, edit, and delete it. This goes for student information, but also for staff members and houses. Furthermore, they want to decide who can view which information, and turn on and off the students' ability to request a house change. Admin users should also be able to help other roles, for example by accepting a house change request.
Housekeeper	Housekeepers are the contact point for the students in their house. Therefore, housekeepers want to view student information. Furthermore, housekeepers are responsible for accepting or denying students' house change requests.
Prefect	Prefects are assigned to help the housekeepers in their jobs. Therefore, they want to view student information and submitted house change requests. However, they are not allowed to accept or deny these requests.
Teacher	Teachers need student information and their respective houses in order to create schedules and project groups. Furthermore, they want an easy way to submit house points for the competition.
Teaching assistant	Teaching assistants need the same student information and house point abilities as the teachers.

Study advisor	Study advisors have responsibilities towards students in specific houses, therefore they need to know which student is in which house. They also want to be updated on changes.
Student	Students should have an overview of who their housekeeper is, who their prefects are and how they can contact them. They should also have the ability to request a house change. Moreover, students want to see the current state of the house cup competition.

*Table 1. Stakeholders and their use of the system*



## Functional requirements - MoSCoW

### **Must**

#### R1. As an admin

- R1.1 I want to see all information about all students: name, student number, email, gender, house, do-group, date of birth, nationality, prior experience, and the year they started the program
- R1.2 I want to upload data of new students
- R1.3 I want to update data of students
- R1.4 I want to be able to manually add and delete students
- R1.5 I want to be able to manually edit student data
- R1.6 I want to be able to add or remove houses
- R1.7 I want to approve and deny house change requests
- R1.8 I want to be able to assign user roles
- R1.9 I want to be able to add and remove users
- R1.10 I want to see a history of all house change requests
- R1.11 I want to be able to manually assign students to houses

#### R2. As a housekeeper

- R2.1 I want to see which students are in my house

#### R3. As a student

- R3.1 I want to see the house I'm currently in
- R3.2 I want to have the ability to request changing my current house
- R3.3 I want to be able to see who my housekeeper and prefect(s) are
- R3.4 I want to login through Canvas to access the system

#### R4. As a user (any user, excluding student)

- R4.1 As a user, I want to filter the information I am allowed to view
- R4.2 As a user, I want to export the information in the table
- R4.3 As a user, I want to login through Canvas to access the system
- R4.4 As a user, I want the system to automatically limit the information viewable to me based on privacy regulations when I log in

### **Should**

#### R1. As an admin

- R1.12 I want to be able to adjust privileges based on user roles
- R1.13 I want to be able to turn off the ability of students to submit house change requests

#### R2. As a housekeeper

- R2.2 I want to receive notifications when students change house

#### R5. As a teacher

- R5.1 I want to get weekly notifications of house changes

#### R6. As a study advisor

- R6.1 I want to receive weekly notifications of students who have changed houses in the shared study advisor email address

R7. As the secretary

- R7.1 As the secretary, I want to receive email notification when students change houses at a rate determined by the immediacy of the change

R4. As a user (any user, excluding student)

- R4.5 I want to be able to filter the data based on which students have changed houses since I last viewed the data
- R4.6 As a student I want my page to be mobile friendly

## Could

R1. As an admin

- R1.14 I want to be able to efficiently sort students into houses in a balanced way
- R1.15 I want to view statistics about student information

R5. As a teacher

- R5.2 I want to submit house points

R8. As a prefect

- R8.1 I want to submit house points

R3. As a student

- R3.5 I want to be able to view the current points of the house competition
- R3.6 I want to see the top three people who have received the most house points in my house

## Won't

As an admin

- I want to see which courses students are currently in

As a housekeeper

- I want to view past groups students have been in
- I want to add private notes to my students
- I want to add notes shared with other various roles to my students
- I want to view and edit the combinations of students that should not work together in the future
- I want to chat with students

As a student

- I want to chat with my housekeeper

### Quality requirements

- QR1. The system should be easy to use  
*The system should be intuitive and ideally not require a manual to use*
- QR2. The system should be easily maintainable  
*Code structure should be clear, and code should be well documented, so it is easy to maintain the code by someone who has not written the original code.*
- QR3. The system should be accessible at any time
- QR4. The system should be easily viewed on different screen sizes  
*The system should be readable on different PC or laptop screen sizes, and ideally on mobile devices as well*
- QR5. The data should be accessible only to the users that have the privilege in viewing the confidential personal data
- QR6. The system should be efficient and rapidly show the information that the user wants to view

## Risk analysis

In order to identify risks and implement possible solutions, a risk analysis was done. A Strengths, Weaknesses, Opportunities, and Threats (SWOT) matrix was used to find possible risks. This matrix contains strengths and weaknesses. Those are all internal factors, meaning the development team had control over them. Furthermore, the matrix contains possible external factors, represented as opportunities and threats. The SWOT matrix seen in table 2 helps to identify possible risks and suitable solutions.

Strengths	Weaknesses
<ul style="list-style-type: none"> <li>- The development team consists of people with diverse interests and knowledge and skills on a variety of subjects</li> <li>- The team is motivated to achieve high quality results</li> <li>- The product is very suitable for expansions and improvements</li> <li>- The product is customized to the exact needs of the client</li> </ul>	<ul style="list-style-type: none"> <li>- The development team consists of students with minimal experience</li> <li>- The team members have no prior knowledge on each other's strengths and weaknesses</li> <li>- The product development is vulnerable to team members falling ill</li> <li>- The product is not useable in settings other than the one it was made for</li> </ul>
Opportunities	Threats
<ul style="list-style-type: none"> <li>- The house policy could be adapted by other studies, creating more demand for the product</li> <li>- Further development and product expansions could make the product available for wider use</li> </ul>	<ul style="list-style-type: none"> <li>- The available time may not be sufficient to create a minimum viable product</li> <li>- The university might change or remove the house policy, rendering the product useless</li> <li>- The university could refuse to host the software</li> <li>- Hackers could try to steal personal data contained in the product</li> </ul>

Table 2. SWOT matrix

Using the SWOT matrix, several risks were identified. To explore the danger of those risks they were assigned a probability and impact rating. The probability rating represents the chance that the risk event will happen. The impact rating shows how much effect that would have. Table 3 contains the criteria of the ratings.

Probability rating	Probability of occurrence	Impact rating	Impact description
0	Less than 10%	0	Minimal impact, emerged problems can be solved on the spot
1	Between 10% and 25%	1	Noticeable impact, will require some time to find solution
2	Between 25% and 50%	2	Significant impact, problems can be solved, but require drastic actions
3	Over 50%	3	Heavy impact, problems probably cannot be solved

Table 3. Criteria of the probability and impact ratings

The total risk score of each risk is the accumulated value of the probability and the impact rating. The total risk score then indicates the significance of the risk. Table 4 gives an overview of the risk scores and the actions required.

Total risk score	Required action
0 – 1	Minimal risk, no action required
2 – 3	Non-critical risk, possible to explore solutions, not essential
4 – 5	Significant risk, take action to reduce it or have a contingency plan in place
6	Critical risk, immediate action required

Table 4. Implications of total risk scores

Considering the total risk scores, strategies were drawn up to try and minimize the risks. The final risk assessment is shown in table 5.

Risk	Probability rating	Impact rating	Total risk score	Strategy
The team falls short in knowledge and skills to develop the product	0	2	2	Have daily standups to talk about what team members are struggling with. Timely ask other members for help.
The time period proves insufficient to deliver a minimum viable product	2	1	3	Keep the product small to ensure there is enough time for an mvp. Rank the importance of non-essential features, and add them only if there is time left.
A team member falls ill causing a gap in the project planning	1	2	3	Have multiple team members work on the same topic, keep track of each other's progress by daily standups. This way, a different team member can take over the work if necessary.
Personal data leaks from the product	1	3	4	Take security into account in every step of building the product. Only use dummy data during testing. Host the product inside the protected canvas environment.

Table 5. Risk assessment

# Design

The following chapter is composed of three subchapters: *conceptualization*, *prototype* and *technical details*. *Conceptualization* presents an outline of the system's features. It explains why it has the form of a canvas plugin, gives an overview of the features per role, and discusses the use case and sequence diagrams. Next, *prototype* discusses the layout of the system. It explains how the graphical interface came to be and illustrates this with screenshots of the system. Lastly, the subchapter *technical details* provides more technical insight into the system. It discusses the used libraries and methodologies, explains the security implementations and shows the database class diagram.

## Conceptualization

### *Canvas plugin*

The TCS House Management System is a plugin for Canvas, a Learning Management System used by the University of Twente. Canvas contains study material, assignment hand-ins, grades, etcetera. Therefore, it is widely used among all students and teachers. This makes it the perfect place for the TCS House Management System. Users can access it with their existing account, without an additional login, and using a system they are already familiar with.

Another reason to use Canvas is security. The system holds and handles a lot of sensitive information. For example, names, email addresses and other personal data are saved in the database and viewed in the interface. Proper security is needed to protect that information. User accounts have to be verified on login, to make sure every user is who they say they are. To illustrate, say that someone without administrative rights would somehow be able to log in as an admin user. They would then be able to download data that should not be accessible to them. But they would also be able to allow all other users to view this data. This is, of course, unacceptable. Canvas handles sensitive data as well, such as grades and submitted assignments. To protect this data, Canvas already has many security features built in, for example two-factor authentication. By placing the TCS House Management System inside of the Canvas environment, it benefits from these security features. In this manner, the data is protected.

### *System overview*

Depending on the role of the user, the content of the plugin differs. Every role can sort, search, filter, and export the data they have access to. The most extensive and complete view is that of the admin users, the sorting hat, and the secretary. Admin users hold administrative access to the system and are the only users that can view all data. They can also add, edit, and delete this data, and have access to all features that other staff roles have. Furthermore, the admin users control some settings that influence the view of other users. They can turn on and off the students' ability to submit house change requests. They can also restrict data access of the various other roles in the system. For example, the admin users can control whether the housekeepers can view the nationality of the students in their house.

Housekeepers and prefects have access to the same information unless an admin specifies differently. When opening the plugin, they see a list of all the students in their house. They can also open a tab that shows relevant house changes. In this way, the housekeepers and prefects can easily see who recently joined or left their house. Furthermore, these roles can see a list of pending house change requests. Housekeepers can deny or accept any request that contains their house either as origin or target house. Both the origin and target housekeeper must accept the request before the house change is finalized. If an admin accepts the request, the change happens immediately. Both

housekeepers and prefects can view a history tab with all past requests. Furthermore, they can award house points to students.

Study advisors, module coordinators, teachers and teaching assistants can all see the same student information and house change history as the housekeepers and prefects, unless an admin specified differently. However, these roles do not have the house change requests view. They can see whether students switched or were added to houses in the history tab. Apart from the study advisors, these roles can also register house points that the students earned.

The students themselves have the smallest view in the system. They cannot see any information about their fellow students. However, they have access to the main page of their house which contains the names and contact information of their housekeeper and prefects. Students can also view the current status of the house cup competition. In addition to the current number of points per house, students can see a leader board for the students in their own house. Furthermore, students can submit new house change requests (unless an admin switched this ability off). They can select the house they would like to be in and provide an explanation of why they want to switch.

### Use case diagrams

Use Case Diagrams give a schematic overview of the system at the highest level. They are used to show the possible user interactions with the system. The following diagrams were made to improve the understanding of all stakeholders involved in the project.

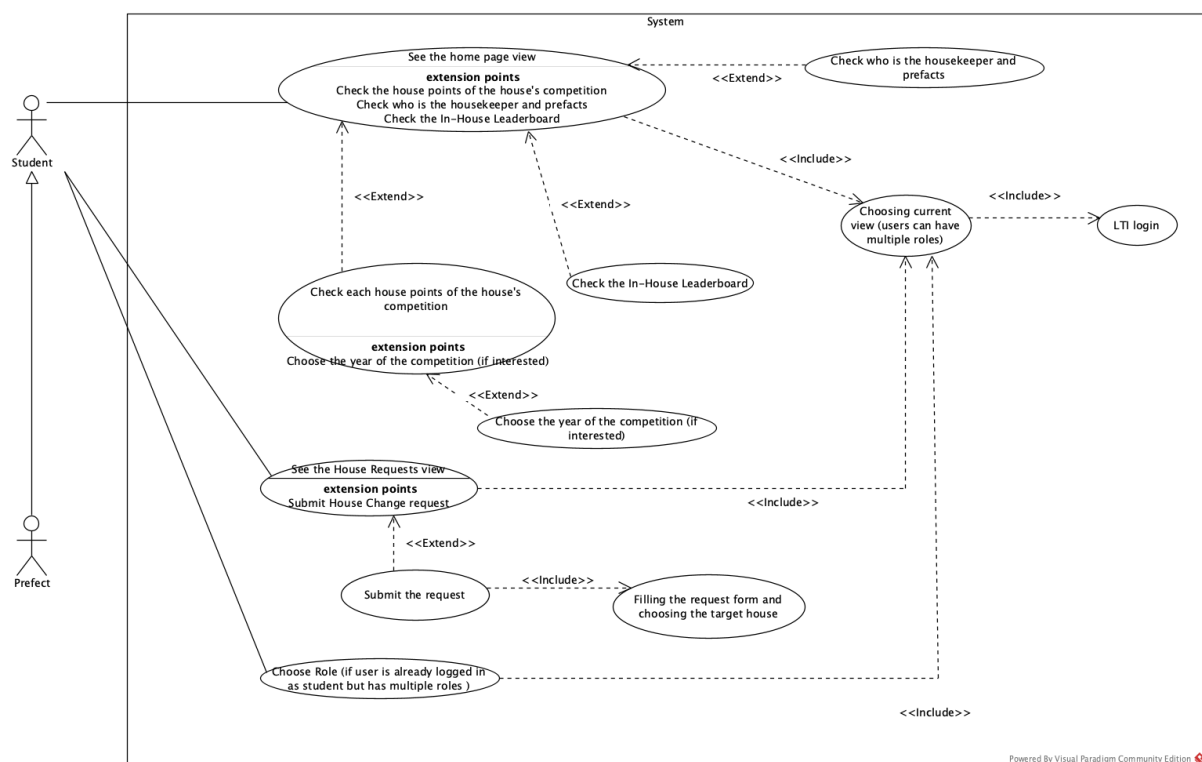


Figure 1. Student-Prefect Use Case diagram

The first diagram, shown in figure 1, represents the possible interactions and views of the student. All use cases include the LTI login, meaning that every action is done after a user successfully logs in. Additionally, the prefects will likely be students, but not every student is a prefect. This is shown with the generalization arrow from the prefect to the student. After logging, in the user is redirected to the home page appropriate to their assigned role. Should a user have more than one assigned role, they have the ability to choose the role through which they want to view the system. The user can always

switch to a different role view by clicking the change role button. On the home page, the student can see who the housekeeper of their house is, along with the prefects. The student can also view the houses in the current year, together with their points in the house cup. A student can change the year in the dropdown menu if they are interested in seeing the leaderboard of previous years. The In-House Leaderboard shows the top 3 earning students. Finally, there is the house change requests view. A student can write down the reason they want to change houses and indicate their preferred house. After the form is submitted, it can be accepted or denied by the housekeepers of the respective target and origin houses.

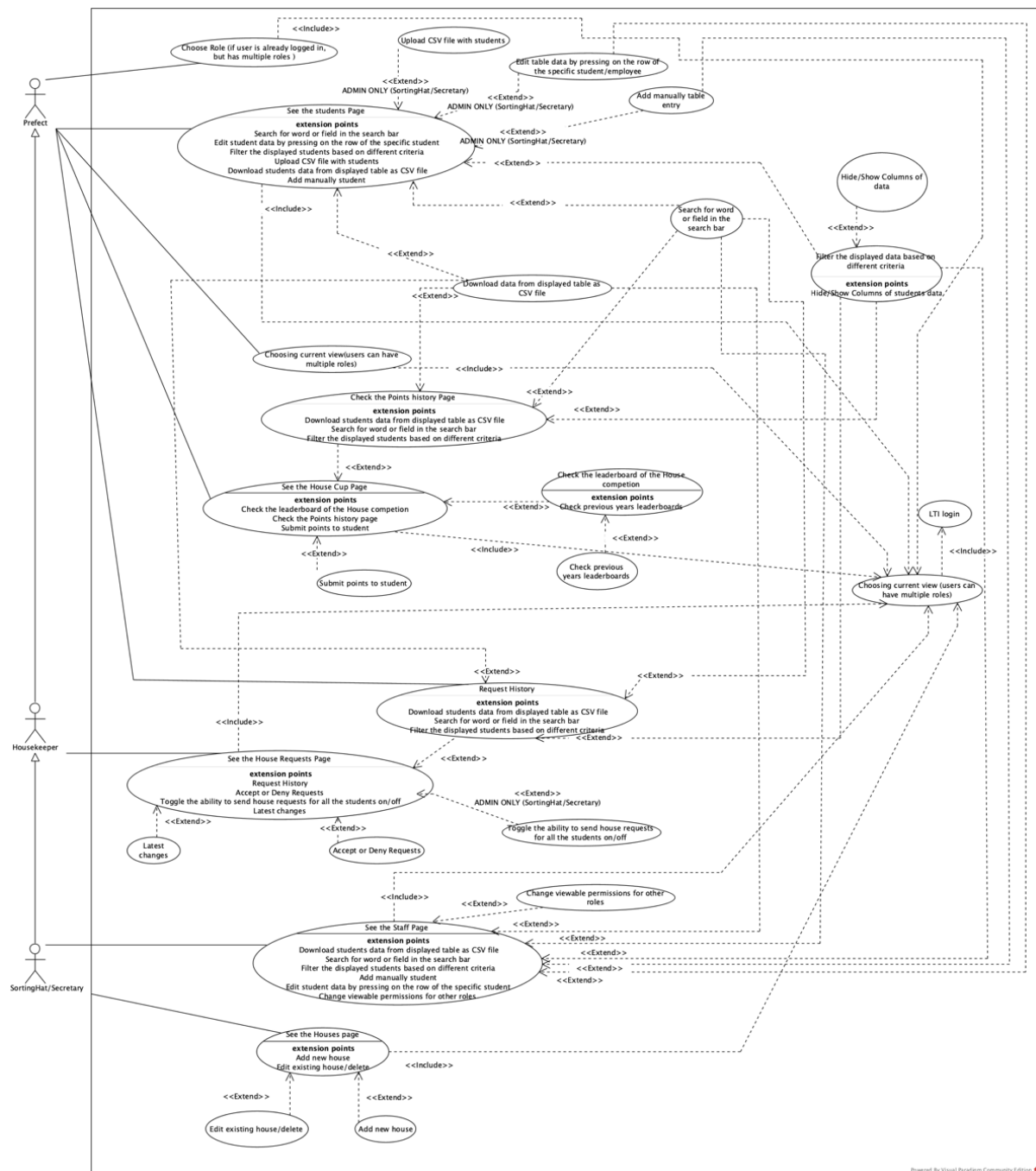


Figure 2. Use case diagram – Sorting Hat/Secretary – Housekeeper – Prefect

The second use case diagram, seen in figure 2, intends to show the interactions between the different user roles. The generalization is made to show that the admin users have access to all the columns of



information, as well as all views. The columns of information that users are allowed to view are not displayed in this diagram, only the views and tabs they may interact with. The student table contains all student data that the specified role is allowed to view. The only users able to edit or add to the student data are the admin users.

Starting from the top, each user is redirected to the student table after logging in. Prefects and all the higher roles such as housekeepers and admins can view the columns of data that they are allowed to and filter the data by choosing the columns they want to show or hide in addition to various other filters. The filters that are available can be used to filter the students by house, status and the start or end date of modified student's data as well as other data points. They can also download the data in the table as a CSV-file and search for names and student numbers using the search bar. Another feature that users have is the house cup view. Here the user can see the leaderboard of the house cup competition. They can also change the year in the dropdown menu if they are interested in past results. They have access to the points history to view different submissions along with their explanations and access to the request history of the students. All viewable data can be downloaded as a CSV file, taking into account the search criteria and filters. The last tab is the submit points page. By entering the number of points and an explanation for giving the points, they can submit earned house cup points.

As a super-class of the prefect role, housekeepers have access to the same features as the prefects with the addition of approving house change requests. As is written in the description of the first use case diagram, both housekeepers of the origin and target house need to approve the request, otherwise, the request is denied. Additionally, for the house change requests, a table with the history of all requests and explanations is available.

Admin users of the system have complete control over the student and staff member data. They can modify, add, and delete students and employees. On the Houses page the admin users can view all the houses, edit the current ones, add new houses, and assign housekeepers and prefects. Furthermore, they can turn on and off the viewable data of other roles in the permissions view on the staff page. In this way, they can modify which attributes of the student data are viewable to certain roles. On the house change requests page, they can toggle on and off students' ability to submit new requests.

### *Sequence diagram*

The proposed sequence diagram shows all the use cases of the possible users based on their respective roles and is made to get a better overview of the more in-depth interactions of the users that will work with the interface. The sequence diagram uses 3 different fragments and one interaction use. The first fragment is “loop” with a given operation to show that the specific messages between the user and web interface may continue or end at some point. The “alt” fragments have the function of an IF-statement. All the messages contained within these will be executed if the operation is fulfilled. The third variety of fragments is the “opt”, which denotes an optional path. These fragments reduce the complexity of the diagram by eliminating the need for multiple alt fragments. All the actions that the users may take on a page are optional. These are exemplified to show the complete scenarios that may happen. The ref fragment is a reference to interactions contained elsewhere in the diagram to eliminate repetition. Such repetition comes from features available to multiple user’s roles.

Starting from the top, each user must enter their credentials in order to gain access to their respective pages. Upon failure, this must be repeated until the username and password are correct (this is shown by the loop fragment). All interactions are contained in a loop since they can be made until the user is no longer active on the platform. If the user has multiple roles, he can choose the one that he wants to use at that specific moment. Messages passed between the user interface and the server contain information on button clicks as well as views and tabs displayed. Each view contains multiple tabs, for example, the Houses Cup view has the Houses Cup, Points History and the Submit Points tabs for users that have access to them. All tabs, aside from the default of the page, are contained within the opt fragments. This is called in the views that contain tables for simplifying the sequence diagram.

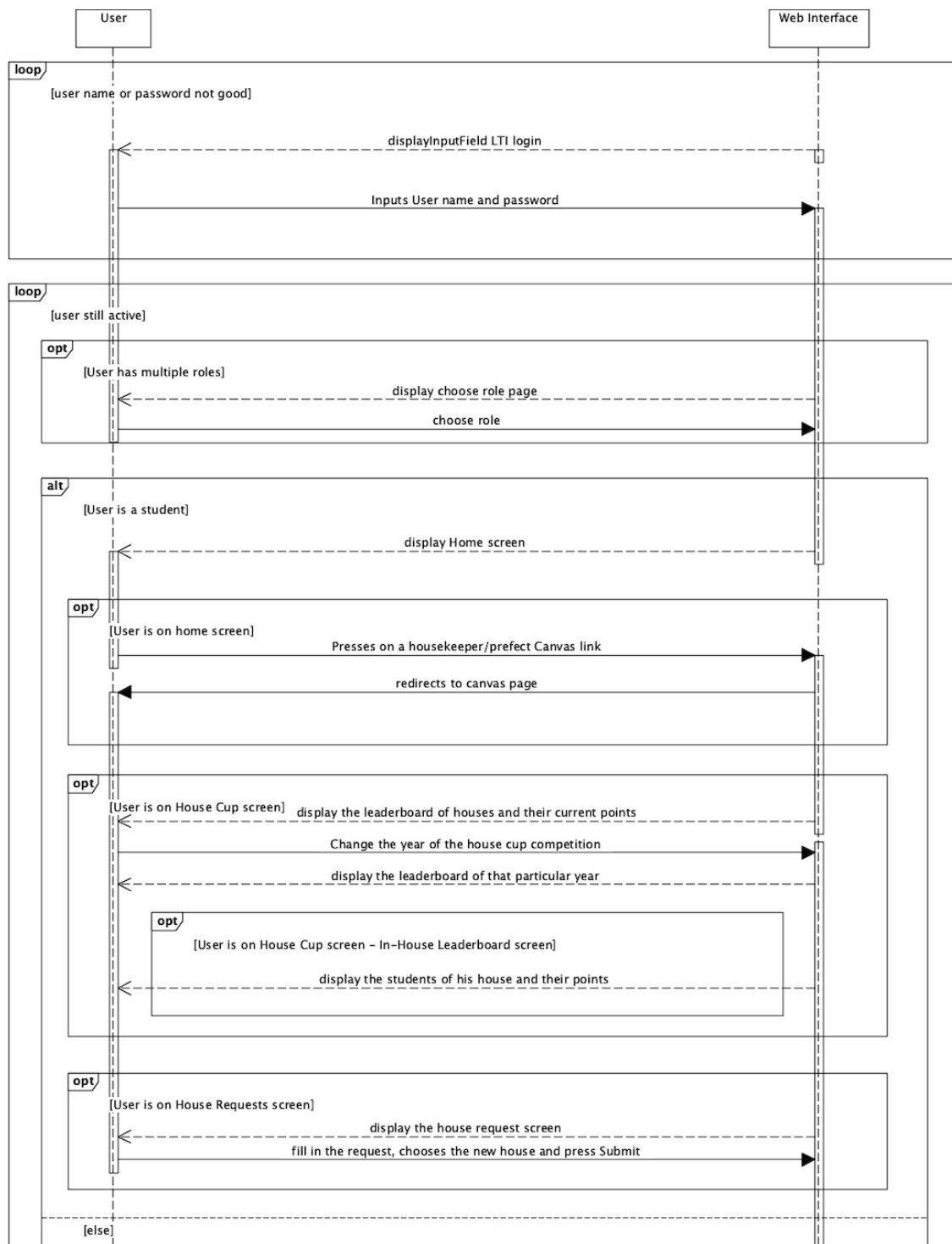


Figure 3.1. User – web interface sequence diagram

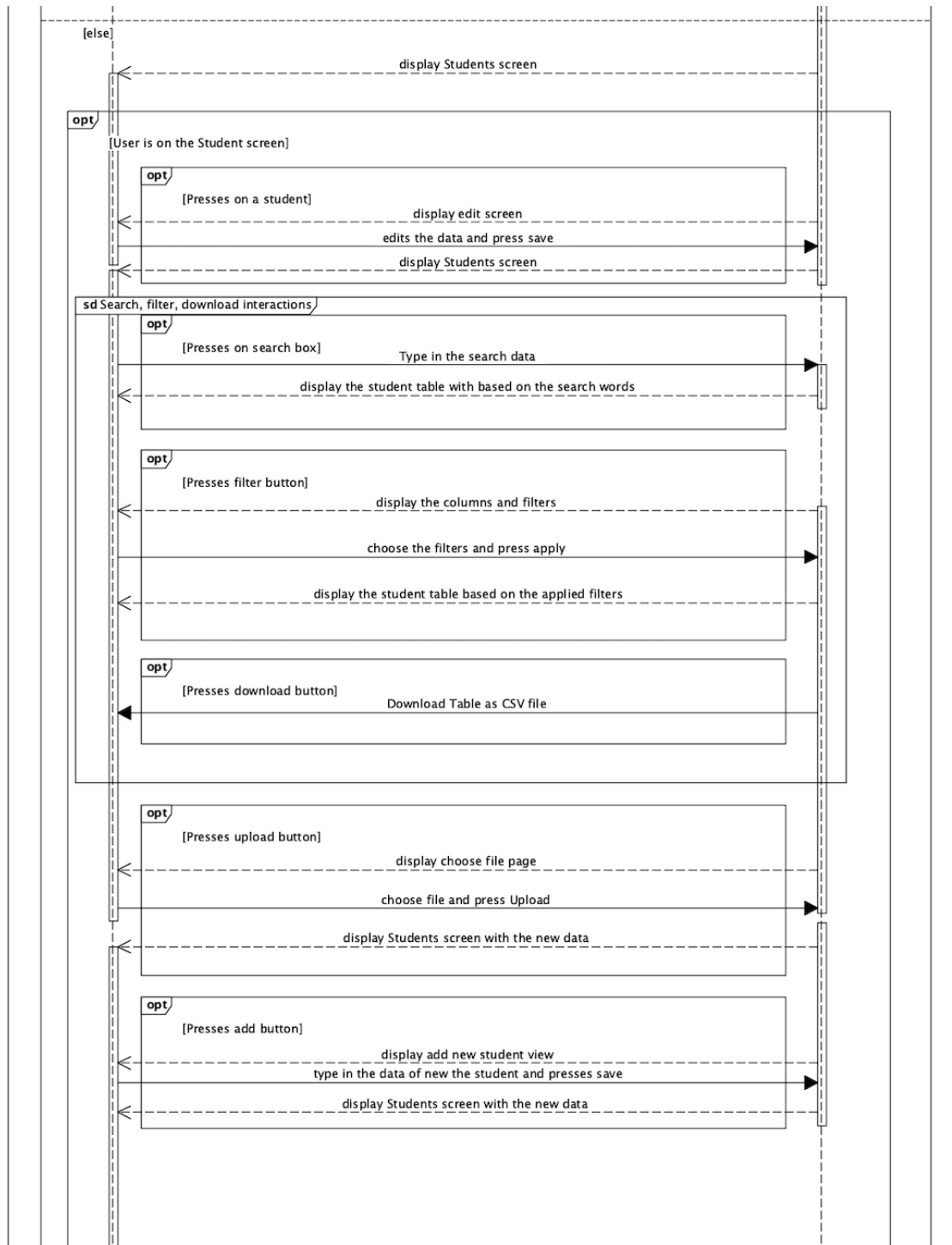


Figure 3.2. User – web interface sequence diagram

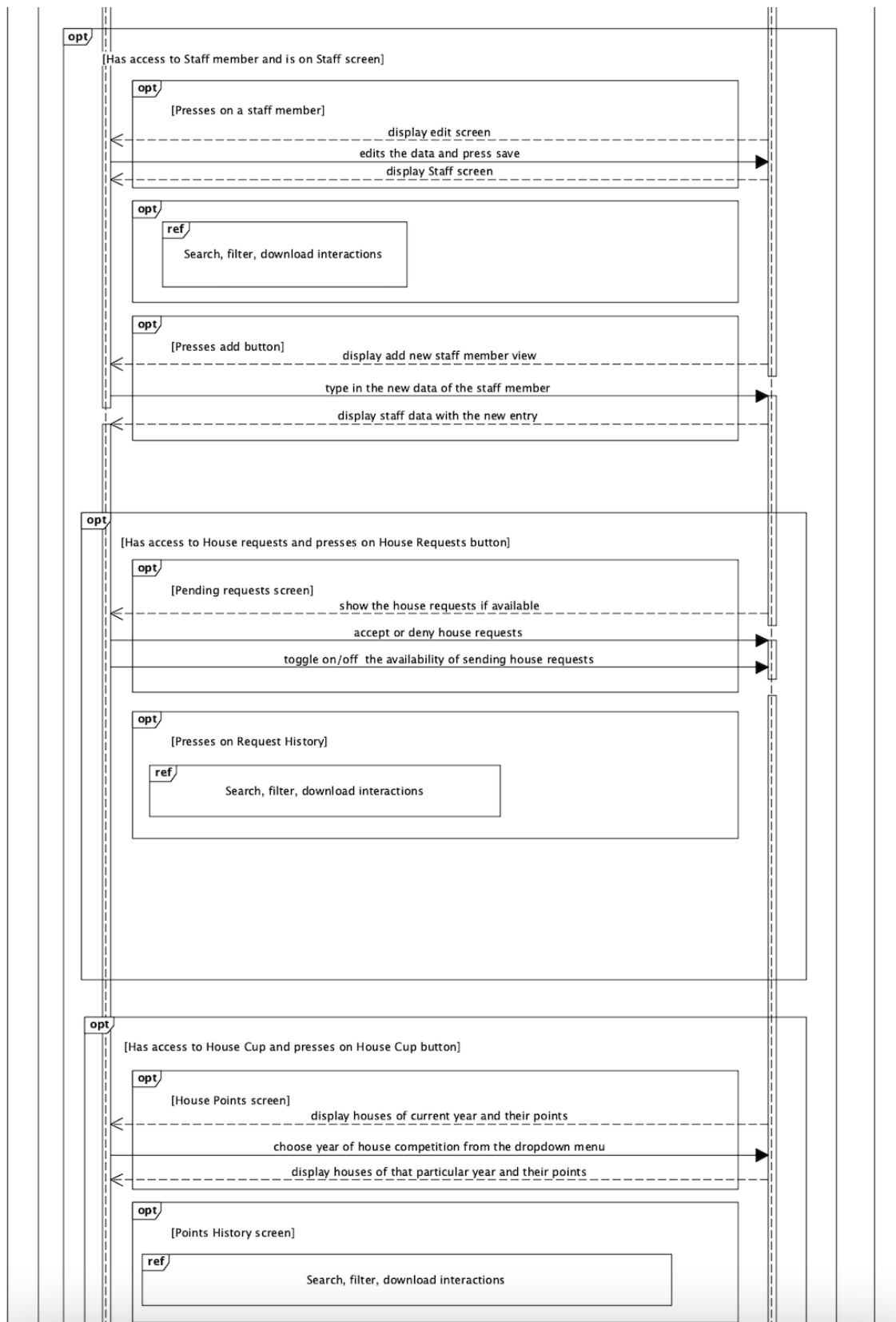


Figure 3.3. User – web interface sequence diagram

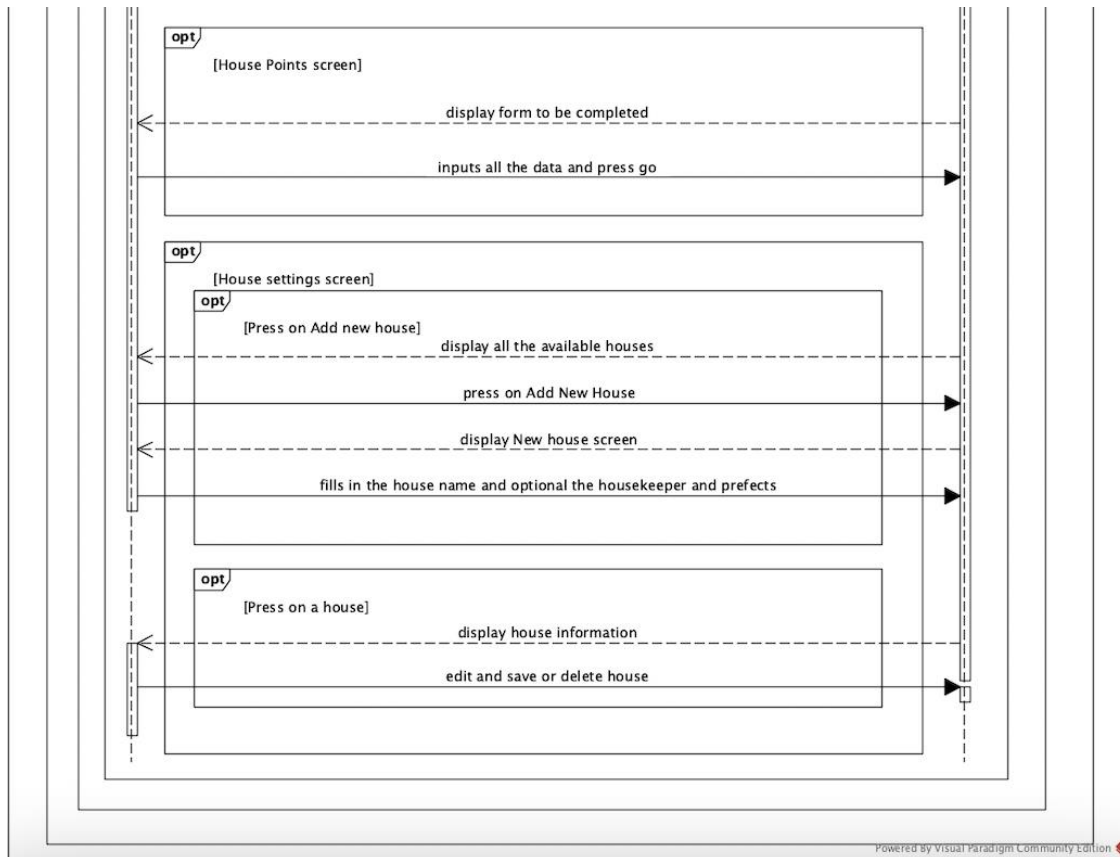


Figure 3.4. User – web interface sequence diagram

## Prototype

Every page has a header with the name and currently used role of the logged-in user. It also has a navigation menu on the left with different sections, categorized by the main features of the system.

Depending on the chosen section, there might be a top bar with different links grouped by the title of the section. An example of the navigation can be seen in figure 4. The components were positioned in such a way that the user could navigate easily between pages.

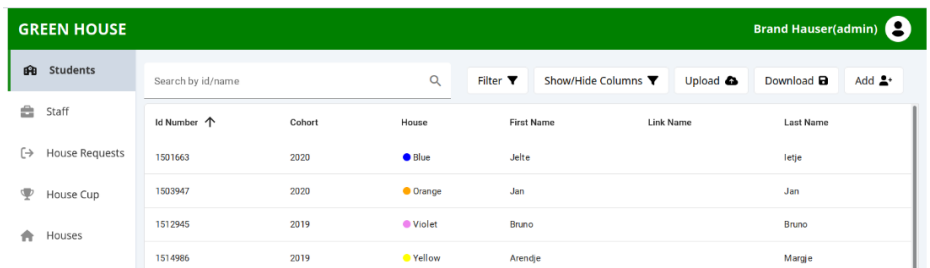


Figure 4. Admin - header and left side menu

The student and staff data as well as any historical data are shown in tables to give a clear and structured overview. An example of the student table can be seen in figure 5. All data tables have a search bar, a filter button, and a download button above them that allows for easy selection and export of data. An upload button for the student table is visible to admin users. The Show/Hide Columns button allows the user to select which columns they

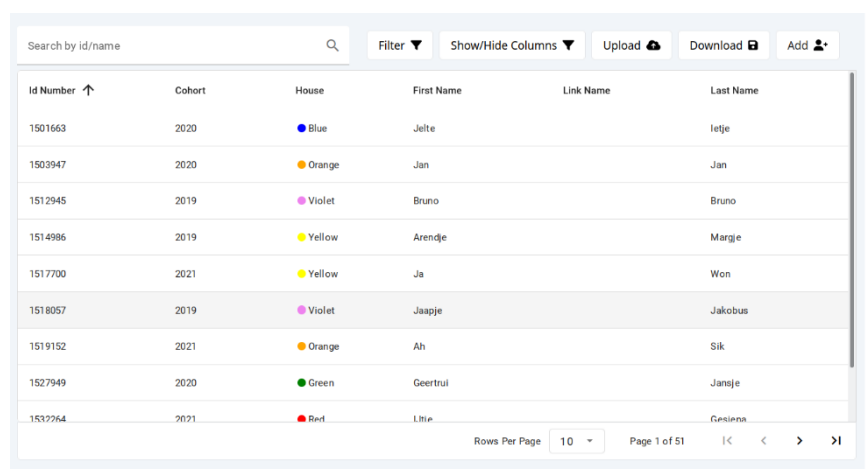


Figure 5. Admin - student table

wish to view and which to hide through checkbox selections, as shown in figure 6. The filter button displays an accordion menu with the different sections of the filterable columns. Each section contains the values which can be filtered in the form of checkboxes, as seen in figure 7. Above the accordion, a list of all the applied filters and a "Remove All Filters" button allows for an easy viewing of all the applied filters as well as a simplified method of their removal. The download button will retrieve the data with the current filters applied. As such, users can export only the data that they deem relevant.

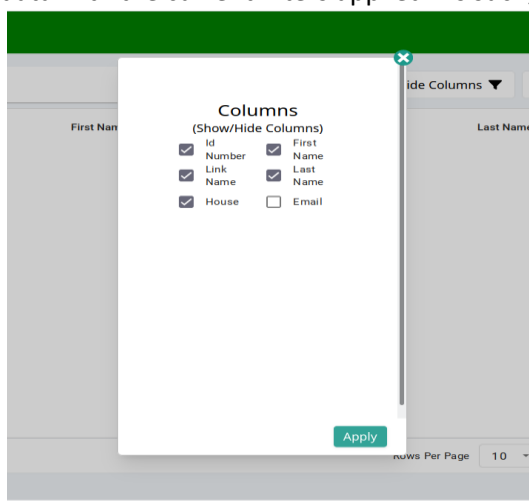


Figure 6. Admin - Column selector menu

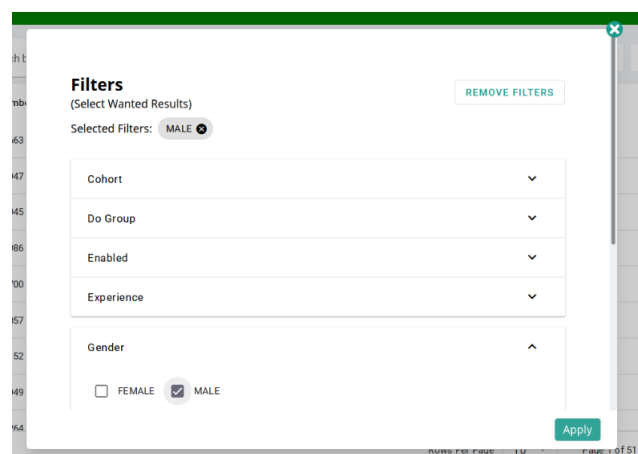


Figure 7. Admin - Filter menu

Upon clicking on an item from the data table, a window shows up with the data of the user. If the data is not read-only and the user has the role of admin, there will be delete and edit buttons. Upon deleting any item, a warning will pop up requiring confirmation to delete the data. This helps to prevent unintentional destruction of data. Within the admin view, the student and user table have an add button, allowing for new students/users to be added to the database as seen in figures 4 and 5. If the addition overwrites other changes, such as removing the previous housekeeper of a house and assigning the newly created one, the user will be warned, and confirmation will be required.

The layout of house change requests has been kept simple; an example can be seen in figure 8. Next to the name of the origin and target house is a circle with the color of the respective house. This makes it easier to see the involved houses at a glance while keeping it accessible for users who might be color blind. House change requests can either be accepted or denied, but when denied the user is asked to give an explanation as to why the request is being denied. This helps the student who made the request to understand the decision. When a request is handled, a notification is sent to the student by email containing the decision and reasoning as specified.

Jimmy Page | s2175791 12-04-2022

Reason: My house mates are in green and it would make life simpler if I could switch

From: Blue To: Green Status: Undecided

Approve Deny

Figure 8. House change request

The home page of a student has three sections: house information, top three students in the student's house based on the house cup, and a leaderboard with a ranking of the houses based on their points. The academic year dropdown allows the option to view the ranking of past academic years, as seen in figure 9. For non-student users, the leaderboard includes the top earning student from each house instead of showing the top three. Most staff roles can assign points to a student or a house, the input window of which can be seen in figure 11. The student can be searched based on id or name, allowing for the user to easily find them. Points can be assigned in combination with a brief explanation.

RED HOUSE Brand Hauser(student)

Home

House Change

House Details

Green House: 259 points

Housekeeper: Yera Barrios-Fleitas

Prefects:

Top 3 in your House at the moment

1. Sammeli Riisa 27 Points

2. Everarda Torniko 24 Points

3. Niini Meriluuli 24 Points

Leaderboard: 2022

1. Yellow House 615 Points

2. Blue House 504 Points

3. Red House 293 Points

4. Green House 259 Points

Change Role

Figure 9. Student - Home page

STUDENTS HOUSES

Student

Submitted Assginment Early(5pts)

Did Bonus Assginments(10pts)

Did Amazing Work(25pts)

SUBMIT POINTS

Figure 10. Multiple roles - Submit house cup points



## Technical details

### *Spring Boot*

The Spring Boot framework was chosen as the basis for the back-end as it is a well-known and easy to use platform with far-reaching capabilities. Simply put, Spring Boot allows the developer to quickly move past setup and configuration to product development. In addition, it provides security integration and abstraction in a manner that does not distract the developer/maintainer from the less boilerplate parts of the product. For example, the code necessary to confirm user roles prior to calling the method, for the purpose of security, can be abstracted through method annotations.

### *PostgreSQL*

As PostgreSQL is an extensive and efficient open-source database management system that was previously in use by the client, The University of Twente, it was an obvious choice for managing the data inherent to the TCS House Management system. Data tables were constructed for student, staff and house data. Additionally, data tables containing all information of submitted house change requests, house cup entries, user roles plus select incidental tables were created to enable the features outlined in the requirements. As PostgreSQL is a relational database, many connections exist between the individual tables such as the student id number, which is the primary key of the students table, used as the identifier (foreign key) in the house change request and house cup tables.

### *Hibernate*

Due to the juxtaposition of a relational database such as PostgreSQL and the object-oriented Java programming language that Spring Boot is built around, it was necessary to find a bridge between the two. For that purpose, Hibernate ORM was chosen to map the relational database entries to Java objects as it is well known, documented, and commonly used in conjunction with Spring Boot. This simplifies the production, as well as the maintenance of the system since the mapping from a row in the data table to a Java object is abstracted.

### *LTI Login*

As the application is intended to be used as a Canvas plugin, the OAuth2-based LTI login that is utilized by Canvas was a necessary addition. The user authentication provided through LTI easily incorporates into Spring Security allowing for the use of role-based access control described below. Through code gained from the University of Oxford LTI demo, the TCS House Management system verifies that the user has been authenticated by Canvas prior to allowing said user to access the application. Public and private keys are used to ensure that the authentication cannot be faked. After successful authentication, the user is redirected to a specified URL. To ensure that this process functions properly the keys and URL must be set in both the *application.properties* file of the TCS House Management code and the Canvas course from which the user is attempting to access the application.

### *Security through Role-Based Access Control*

Each user who accesses the TCS House Management application must be assigned at least one role within the system. Roles are assigned to individual staff members through the API by an admin user. The role of student is automatically assigned to students as they are added to the database through the application.

Tracking of the logged-in user through the combination of Spring Security and session cookies enabled a novel approach to data read access protection. This innovative approach allows the admin users to determine which data points of the student information will be accessible to the distinct roles enabled in the system. As such, only the information that the user is allowed to view is passed to the front-end user interface.

To further protect the data, regarding methods that should only be enacted by certain users, all REST methods are protected by Spring Security's `PreAuthorize` annotation. This methodology ensures that the user calling the method through an HTTP request has one of the specified roles before that method is called. In this way, write access is mainly limited to the admin users while allowing read access to all other data based on the user's roles.

#### *RESTful services*

HTTP requests through RESTful services and CRUD protocols were implemented as the bridge between the front-end user interface and the back-end as they represent the standard best practices in web app development.

For purposes of efficiency and data protection, all filtering, pagination, searching and further manipulation of the data to be presented to the user is handled either by the database through SQL queries or by the back end prior to the transmission of data over the network.

#### *Frontend libraries*

The main library used for implementing the front-end was SvelteKit. This is a library by Svelte, which was chosen for its ease of use, allowing for higher productivity. It is efficient in its way of defining routes to the web pages, as well as its way of allowing data to be persisted between pages. SvelteKit also provides reactive statements and variables, which upon a change will cause the application to re-render and update the user interface accordingly. Furthermore, it allows prefetching on links, allowing the page to be fetched when a user hovers over a link. This greatly improves the user experience in terms of loading times.

Next to SvelteKit, a library called MaterialUI was used. MaterialUI provides many basic components that were used in the front-end: buttons, switches, data tables, pagination, search bars, text fields, select menus, and checkboxes. This helped in easily creating a layout with a universal and coherent design.

Lastly, a library was used to aid in formatting the pages. TailwindCSS is a CSS utility library providing many CSS classes. It allows for quick implementation and adjustment of the visual format of components.

#### *Database class diagram*

In figure 11, a diagram can be seen that shows the structure of the database, as well as the relations between classes. For example, it can be seen in the diagram that students are a subset of users, and that each user has a many-to-one relationship with the houses table. The database was built in this way to reflect the relationships that exist between the various entities represented by the data.

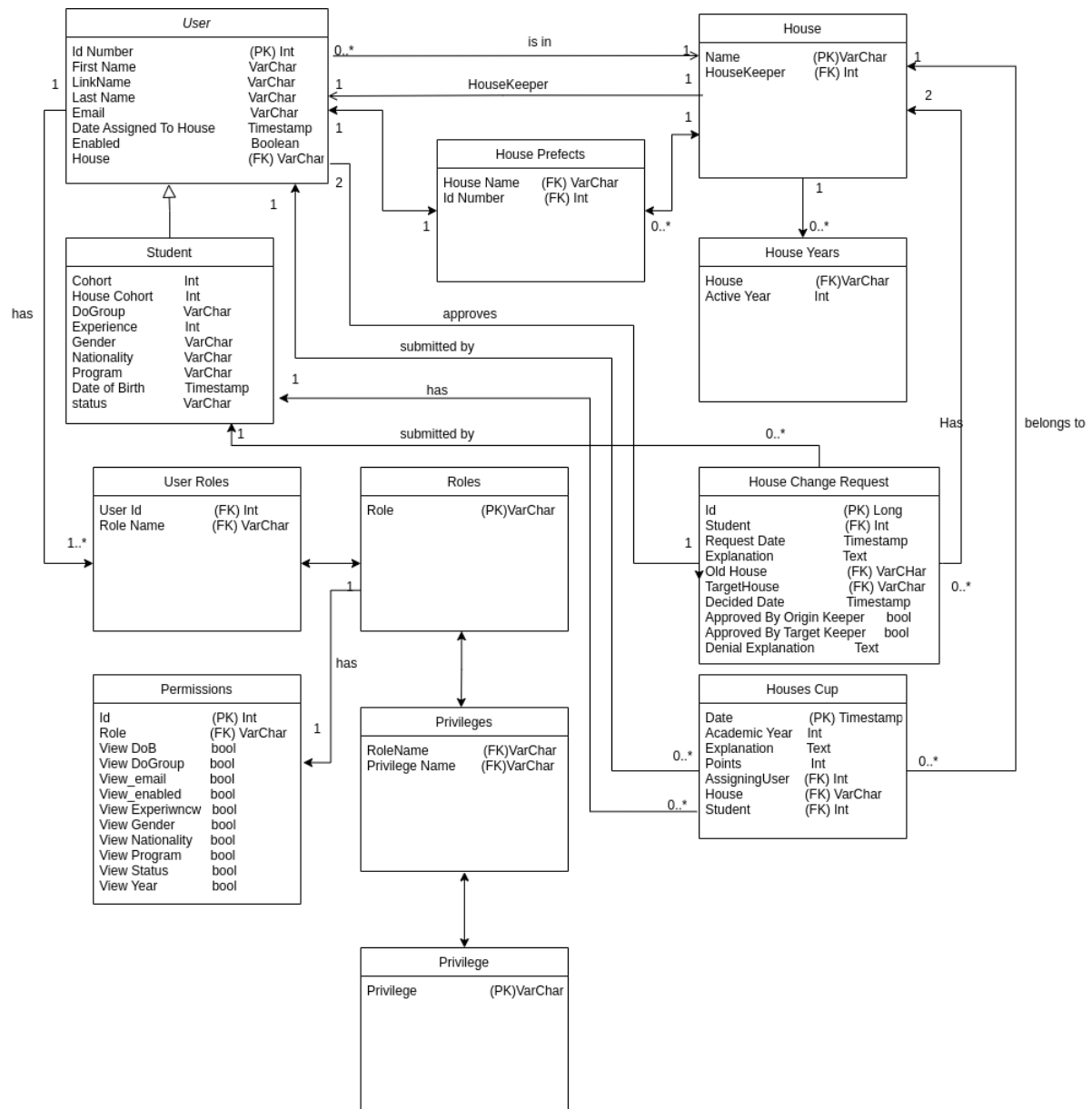


Figure 112. Class diagram

# Testing

Multiple testing methods have been used to make sure various parts of the system worked as intended. These different methods include unit testing, end-to-end testing, system testing, and usability testing. All methods proved to be very helpful in locating bugs and improving the overall performance of the system. A test plan was made to create a clear overview of the expectations and goals for the testing process. Below, this plan and the test results are explained for every different type of testing that was performed.

## Unit testing

### *Test plan*

For a system that relies heavily on communication with a database it is important to ensure that every method functions as desired. All CRUD methods will be tested using MockMVC, a library that sends mock HTTP requests. Helper methods are then tested as parameterized tests to test multiple inputs for the expected outputs and to check edge cases as well as possible points of access for malicious intent. Where possible, methods will be tested for both success upon correct inputs as well as for proper exceptions on incorrect input. The goal for this section of testing is to reach at least 90% line coverage, where all tests give the expected outcome. Once these criteria have been met, the unit testing will be deemed successful. Unit tests will be written along with the development of the methods. Once a method is finished, a unit test will be written for it. This method of testing throughout the development prevents unexpected bugs from appearing at the end, which could have been intercepted much earlier in the project.

### *Test results*

Of the sections of code that were able to be tested through unit tests 90% line coverage and 96% method coverage were achieved. Testing was not possible for the parts of methods involving role-based access as this requires a session cookie obtained through Canvas LTI login which cannot be done in Junit tests. All performed tests resulted in the expected outcome, thus successfully completing the unit testing.

## End-to-end testing

### *Test plan*

The end-to-end testing will be done manually during the development. It will act as a precursor to the system testing, to remove as many bugs as possible beforehand. After the completion of a feature in the frontend, it will be tested thoroughly by trying out several variations of the task and making sure no errors occur. This type of testing will be less structured and will be more of a method that helps prevent large bugs from occurring later.

### *Test results*

Overall, the end-to-end testing mostly helped in ironing out the bugs that arose while integrating the frontend and backend. It helped, for instance, in realizing pages should be automatically refreshed after making a CRUD request that would affect what the page would look like. It also helped in making sure that the filtering of the data table was fully functional. For example, having the correct options appear in the filter window, as well as columns being in a certain order.

## Usability testing

### Test plan

Tests will be performed with people outside of the development team who are all prospective users, to test how intuitive the system is to use. Presenting the system to people who have not seen it before will help find bugs and improvements that the development team could not have foreseen because of their continuous involvement in the system. The amount of usability testing sessions will depend on the availability of the users, thus there is no set amount to be performed.

### Test results

In total two test sessions were done, one testing the usability of the student table page and one to test the usability for admin users. More were planned, but due to lack of response from possible testers, the total stayed at two sessions. For each session, the results are explained below.

#### Test session 1 – Usability of the student table page

Tests were done with two users to assess the intuitiveness of the design choices made for the student page. This test was performed during sprint 5, and proved to be quite useful, resulting in many small changes that helped improve the system. Users were given five tasks to do without any help or instruction on how to do it. Through observation as well as a small questionnaire regarding the participants' opinions on their experience it was possible to gather points that the system could improve upon. The full test results can be found in Appendix A. The users were asked to rate each task on a scale from 1 to 5 on how easy it was, 1 being very hard and 5 being very easy. The results of these questions can be found in table 2.

Task	Answer user 1	Answer user 2	Average
Adding a new student	3	5	4
Editing student info	4	5	4,5
Deleting a student	5	5	5
Filtering data	4	5	4,5

Table 6. Usability testing session 1 - Results on how easy the tasks were to perform

Overall, both users were very positive about their experience. The users were asked to perform the following tasks: adding a student, editing the information of a student, deleting a student, and filtering the data in the table. Both executed the tasks with ease, only making small remarks about details of the process. Most of these remarks were about the input fields for adding a new student. The *id number* and *experience* fields should always have integer values, but in the system, it was possible to write letters as well. While this does not necessarily form a problem when creating a student, since a non-numerical value would not be accepted by the backend, having the option to type letters made the field less intuitive to use. These two fields were changed to only allow for numerical values to be typed into them.

Another remark was that the *nationality* field was a dropdown menu, containing all values that already existed in the database. This implementation of the field made it impossible to add a new nationality manually, which was not the intention. Therefore, this field was changed to an open field, but still retains the dropdown that shows possible auto-complete suggestions.

One small, but important comment was made about the *id number* field. It was unclear whether the user should input an 's' at the front of the student number or whether that would be handled automatically by the system. To avoid any confusion, the field was changed so the 's' is already shown at the start of the input field. This way, no doubts would be able to arise about having to add it. The same was done with regards to the staff table, adding an 'm' to the id number field.

Lastly, a change was made to the way that house change requests are accepted. This was a topic that was extensively discussed with the client but was also dependent on the opinion of the people who would be using the system as housekeepers. The original plan was to allow house change requests to either be accepted by an admin or the housekeeper of the destiny house. After asking the opinion of the head of the housekeepers, who was one of the users involved in this test, he shared that this had been discussed among the housekeepers already. They concluded that requests should be accepted by either an admin or both the housekeeper of the old house and the housekeeper of the destiny house. This request was then implemented.

A full list of the changes that were made after the tests can be found in Appendix A.

### Test Session 2 – Testing with admin users

Usability tests were performed with the two University personnel who are intended to be the admin users of the application. The intention of this test was to analyze how user-friendly and intuitive the designs regarding the tasks of the admin users are. This test was performed during the final sprint. Many small changes that helped improve the system resulted from the input of one of the two test participants. As with the previous tests, users were given five tasks to do without any help or instruction on how to do it. Through observation as well as a small questionnaire regarding the participants' opinions on their experience valuable information was gained to improve the application.

Similar to the previous usability test, the users were asked to rate the tasks on a scale of 1 to 5 on how easy they found the task. The results are displayed in table 3. The condensed transcript and further test results can be found in Appendix B.

Task	Answer user 1	Answer user 2	Average
Uploading a CSV file	3	5	4
Filtering data	5	5	5
Adding a staff member	4	5	4,5
Approving/denying a house change request	5	5	5
Change viewing permissions	5	5	5

*Table 7. Usability testing session 2 - Results on how easy the tasks were to perform*

The tasks that the participants were asked to complete were uploading a csv file of students, filtering the list of students visible on the web page for a requested sub-set of students, adding a new staff member, approving/denying house change requests, and modifying permissions to view student data by role. Both participants completed the tasks with ease. Participant 1 provided a significant quantity of input to improve the user friendliness of the application. A repeated comment was the desire for further confirmation notifications to acknowledge that the user has completed tasks correctly. Many actions that may be taken by the admin user did not present messages to the user declaring the success of the action. In some cases, this was due to errors in the code that prevented the messages from displaying correctly. Following this test many corrections and additions were made to make confirmation messages more abundant and clearer.

Another important improvement that came as a result of the test with participant 1 relates to how students are selected in awarding house cup points. Initially this was done by entering the id number of the selected student. The participant made it clear that this is not a preferred method as it takes extra work to find the appropriate student on the student page in order to find the correct id number. This id number would then have to be copied into the field on the submit points page. The solution was to modify the submit points page so that users can search for a student by name with an auto-complete option. Upon submitting, the data passed to the server for point submission would contain the id number procured from the search but hidden to the user. Additionally, many minor changes

were made such as the ordering of columns in the tables, added color coding, simplification of text items, etc.

Participant 2, on the contrary, had little suggestions toward the improvement of the software. Comments from this tester mainly expressed how easy and intuitive the software was. As such, no changes were made following the usability test with participant 2.

## System Testing

### *Test plan*

System testing will be performed in the final stage of the project, to test whether the user stories gathered in the design phase of the project are implemented and functional. Each user story that refers directly to a capability of the system will be tested. These tests involve specific user roles and the actions available to them. This testing method will act as a final check to make sure the system works as intended. Therefore, it is essential that all system tests pass.

### *Test results*

System testing was performed in the final sprint of the project. As such, most errors and oversights that would have resulted in failed tests had already been corrected through the unit, end-to-end and usability testing. Few changes resulted from the system testing. One such change revolved around a null pointer error that resulted from attempting to find the house of a study advisor who is not assigned to one specific house. This problem is avoided for the admin and teacher roles who also do not belong to a house through an if-statement in the code. The study advisor role had been left out of this if-statement as an oversight. The correction was made following the test.

Each system test was documented. The test procedure was recorded with all input values used as well as the expected behavior of the action being taken. Actual behavior was recorded through screen shots. The full documentation of the system tests can be found in Appendix C of this report.

## Future work

The current application has enough implemented features to make it a functional and useful system for all the roles involved. However, improvements and additions can always be made. While gathering requirements, several features came up that fell outside the scope of this project due to the limited time available. They are listed in this section to give an overview of the future additional possibilities of the system.

The first improvement would be to host the application in a Docker container on the university server. The application would then connect to a PostgreSQL database, also on the university server. Due to the LTI connection, this setup would have further complicated testing and development. The connection to a database on the University server, however, was implemented.

One extra feature could be the ability to save private and shared notes on students. This feature was requested by housekeepers during the interview process. They mentioned that earlier correspondence with students often got lost in their email inboxes. If they could add notes to their students in the plugin, it would be much easier for them to keep track of earlier communication and agreements. This could be implemented by adding a clickable notes column in the student table. Users could view notes here and add new ones. These notes would then be saved in a separate database table. Each note entry would hold a student number, date, and the text itself. Every entry would also hold an attribute that indicates who is allowed to see the note. In this way, private notes can only be viewed by the author.

A different idea to improve student correspondence was to add a chat functionality, as suggested by the client. By using a chat, all communication concerning the houses would be in one place for the staff members. This should make it easier for them to respond to everyone personally and prevent messages from getting lost in emails. It would also enable students to reach out to their prefects and housekeeper in an easier, less formal way. Detailed implementation options for these features would first need to be explored further.

Another feature suggested by the client is to automatically assign students to houses. In other words, the sorting hat would be able to tell the system that all first-year students need to be assigned to a house, after which the system would give a suggested division based on students' kick-in do groups, nationality, gender, and experience. The sorting hat could then review the division, edit it where necessary, and accept or deny it. This can be done by writing an algorithm that has access to all the student data and divides the attributes equally over the available houses. As an extra option, different attributes (for example experience) could be prioritized over others in the division, as specified by the sorting hat.

Another possible feature requested by the sorting hat is house statistics. Adding statistics to the sorting hat view would provide a better overview of the houses. For example, they could see that the green house contains many more students than the red house (possibly due to house changes after the initial assignments). This information could prevent the housekeepers from accepting more students to switch from the red to the green house, to protect the balance. Without these statistics, this data must be determined manually. In the implementation, this would include an extra tab in the interface. This tab would then show multiple graphs and charts that are refreshed on every page refresh to keep them up to date. The backend would gather this data from the database on every frontend request.



The last set of suggested features originates from the module coordinators and teachers. They requested that the system also contains information about which module the students are enrolled in. Ideally, they would like to know which courses the students are actually joining. This would give them a much clearer overview of which and how many students are participating in their course. Unfortunately, it is not possible to obtain this data directly from Osiris, due to security concerns. The best way to implement this would be to add a course column to the student table in the database. Admin users can then download the course information from Osiris and upload it to the plugin.

Lastly, the teachers suggested keeping track of student conflicts in the plugin. This would mean that when a group breaks up or has a significant conflict, the teachers can enter in the plugin that these specific students should not be forced to collaborate again. This can prevent future conflict in modules that use teacher-assigned groups for their projects. To implement this, an extra database table is required. This table would have a many to many relationship, linking together pairs of students that should not work together anymore. Users would then be able to see a list of students upon opening a student's information in the plugin.

## Conclusion

This project aimed to develop a product that eases the process of managing students in the TCS houses. The original approach involved many spreadsheets that needed to be accounted for and sent to teachers. This approach left a lot of room for error and proved to be extremely inefficient. In response to this issue, the TCS House Management system, a Canvas plugin, was developed. It aims to reduce the workload and responsibilities of the admin of the system while also allowing for other participants to be more involved. The system allows many users to perform their intended tasks: viewing students, assigning points to a given student as a part of the house cup and, given that the role allows, responding to house change requests. All the features were discussed and accounted for, allowing for the development of a well-functioning product while not sacrificing privacy. A lot of user and system testing was done to develop the product with the best possible user experience and with the highest level of functionality. Prospective users of various roles have taken part in user testing and were very satisfied with the newly developed product, confirming that the goal of the project was achieved.

# Reflection

## Planning

The original planning to conduct the work of the project was made for the Project Proposal. Looking back at the proposal, the development closely followed the plan and steps laid out at that early stage. The main idea was to divide the team into two sub-groups where each sub-group needed to take care of either implementing the frontend or the backend. The next step was to set up a list of functional and non-functional requirements and use cases by interviewing the stakeholders. This helped to make an overview of all parts that need to be implemented and assists in the division of tasks. Also, the Scrum methodology was followed, dividing the requirements into sprints, assigning tasks to project members, and keeping track of each member's progress during daily standup meetings. As stated in the proposal, communication with the stakeholders continued during the development weeks to get the requirements of each distinct role and to satisfy the needs of the users of the system. The stakeholders and supervisor were kept apprised of the progress and multiple user tests were conducted in order to gather opinions of the design and to find possible faults in the system.

MAAT Project	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
Explore										
Ideate										
Conceptualize										
Prototype										
Implement										
Evaluate and Testing										

Figure 123. The original project plan Gantt chart

The Gant chart seen in figure 12 shows the original planning as taken from the project proposal. Phases like ideation, conceptualization and prototyping went smoothly and the two sub-groups made a significant effort in finalizing them before the original planning. This was valuable for having a larger window for implementation phase. Prototypes made in Figma were successfully designed from the beginning and the sub-group used them as a guide for implementing the frontend. Evaluations of the system parts, such as tabs and views, were done incrementally as the code was developed. Testing was done by each team member for their specific work and later multiple tests were made to check if indeed everything functioned as intended.

### Task and work division

While developing the project, the team was separated into two parts, owing to the fact that the project could be divided into two main aspects – the frontend and the backend. Fieke Middelraad and Brand Hauser were assigned the task of developing the backend, while Katja Wevers, Theodor-Fabian Niculae and Martin Stoev were assigned to the frontend.

<b>Backend</b>	
Fieke Middelraad	Setting up Docker containers, project proposal, implementing backend pagination, filtering and searching, creating endpoints, email notifications, report, poster
Brand Hauser	Creating endpoints, CRUD and RESTful services development, database design and integration, LTI access, role-based access control, user data permissions, frontend to backend integration, user testing, system testing, report, presentation
<b>Frontend</b>	
Theodor-Fabian Niculae	Designing the frontend, developing the house cup page, exporting data from a table, design diagrams, connecting to backend, frontend bug fixing, project proposal, report
Katja Wevers	Designing the frontend, developing the house change requests pages, developing houses page, MoSCoW, user testing, connecting to backend, frontend bug fixing, project proposal, report, presentation, poster
Martin Stoev	Designing the frontend, developing data tables, developing houses page, developing the permissions page, connecting to backend, user testing, frontend bug fixing

*Table 8. Team workload division*

# Appendix

## Appendix A – Test results test session 1

Both tests were performed on 18-03-2022

### User test 1

#### *Notes during testing*

Very quick in use of the system

Does not know what the *enabled* field is (field should not be there → take it out, same for *Last modified* and *status*)

Email field is putting in automatic capitalization (bug)

When adding student expects to be able to go to the next input field by pressing Tab

Not able to add new nationality, since it is a dropdown with options pulled from the database

Used Add button, not the Add student tab

Calls the process of adding student standard (in a good way)

Suggests an 'all' option for pagination

Would like to be warned about unsaved changes when clicking out of a student modal

*Last modified* column is not clear that it is only for house changes

Deleting is too easy, would like there to be a confirmation

When filtering, presses INHOUSE unnecessarily

Thinks it would be better if 'null' is replaced with a dash

#### *Questionnaire*

##### Task 1 - Adding

On a scale of one to five, one being very difficult, 5 being very easy. How easy was the task?

3, average easiness. It wasn't easy, but there were some small troubles.

What did you find easy/difficult about this task?

The capitalization of the email address and not being able to add a new nationality.

For the nationality, it would be annoying if you would have to scroll to the right one in a dropdown. It would be nice if the most common are on top, or if nationalities are suggested based on what you have already typed.

Regarding this task, are there any changes that you would make to the page is involved?

What he said previously about the nationality field. Also remove the unnecessary fields like last modified and enabled. Another thing is that it is unclear what experience can be. Can it be 3? 1000? Should be made clearer.

##### Task 2 - Editing

On a scale of one to five, one being very difficult, 5 being very easy. How easy was the task?

4

What did you find easy/difficult about this task?

It was very easy to do it, since it is similar to adding a student. It took me a second to notice that the fields become editable when clicking the Edit button because there's no visual cue for whether it is read only or can it be edited. For a typical interface you would have them gray if you can't edit it and white if you can, something like that so that the changes are very visible. Now only the drop-down boxes became gray. And initially I thought this was the only thing I was allowed to edit.

Regarding this task, are there any changes that you would make to the page is involved?

See previous question

### Task 3 – Deleting

On a scale of one to five, one being very difficult, 5 being very easy. How easy was the task?

5

What did you find easy/difficult about this task?

It was too easy, one click and the student is gone. There should be a confirmation window or an undo option

Regarding this task, are there any changes that you would make to the page is involved?

See previous question

### Task 4 – Filtering

Questions were not asked, but general consensus was that it was easy to use, but some filters like *Last modified* and *enabled* did not make sense and should be removed

### Final questions

How would you describe your overall experience with the system?

OK. It seems to work, also clicked on some other pages and everything looks nice.

What do you like most about the system so far from what you've seen?

It seems pretty standard, which is good.

What did you like the least about it?

The whole email thing. That it can clearly be incorrect because I can enter a two here (he added it after the @ sign). It's weird. And of course, the fact that you have multiple ways to add students.

Was there anything that surprised you about the system?

No.

### Changes made afterwards:

- *Last modified* changed to *Date* assigned to house
- Nationality is no longer a dropdown, is now a text entry that makes suggestions while you type
- *Last modified*, *status*, and *enabled* fields were removed from Add student feature
- *Experience* field is now a dropdown with a fixed set of options
- Gender dropdown now includes "other" as an option
- Null values now appear as a dash

## User test 2

### *Notes during testing*

Wants to put an s before student id number, is unsure whether it would be needed

Confused by the 'no matches found' text when adding a new nationality

Goes very quickly through all tasks

Just like user 1, clicks INHOUSE when filtering

*Experience* and *id number* field should be integer only

### *Questionnaire*

#### Task 1 – Adding a student

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

A 4 because I was having a fight with the nationality thing, but that's just minor. I would give it a 5. I feel like it's the only reason it was confusing because the only nationality added there seems to be Netherlands or Dutch.

And whenever I completely went back it would just autofill. Except for that it was fine like it was very easy.

Regarding this task, are there any changes that you would make to the page is involved?

Maybe mention whether the 's' is mandatory or not, because I can guarantee you that a lot of people will add it if they had to use the system. Maybe you can implement it so if a person has put it in, it gets removed. If it is not there, it doesn't matter. Something like that.

#### Task 2 – Editing a student

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

5, very easy.

Regarding this task, are there any changes that you would make to the page is involved?

No, I think the design choices make sense. My first instinct was OK, it would be nice if I could go in here and then just immediately edit it, but then I was realized that the last thing you want is if someone clicks it, it accidentally removes something and then automatically gets saved. So I like this choice, it is a good choice.

#### Task 3 – Deleting a student

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

5, very easy.

#### Task 4 - Filtering

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

5, very easy.

Regarding this task, are there any changes that you would make to the page is involved?

Maybe it would be nice to be able to filter on students who are not in a house, would be nice to have that as an option in the house list. It also could be nice to have a question mark in front of the search bar that explains what the search function can be used for

Changes made afterwards:

- S- and m- prefix are automatically filled in for *Id number* edit and add field
- *Id number* and *Experience* fields were made to be integer input only
- Changed house requests so they are authorized by either just an admin or both the old housekeeper and destiny housekeeper
- Changed dropdown in edit fields so 'no matches found' is not there anymore
- Added an asterisk to the *Id number* header of the edit field to show it is mandatory



## Appendix B – Test results test session 2

Tests performed on 12-4-22 and 19-4-22

### User test 1

#### *Questionnaire*

##### Task 1 – Uploading a CSV

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

I will say that in the first I will say five, because it was really intuitive was really easy to make. So using the same system works like this, that I didn't receive any confirmation messages. So I am not sure that the changes was made. Also, I expect something like whenever I introduce a list of students some information about okay, you are inserting this specific number of newer students and you are re-writing this specific number of students. I feel unconfident. I am not sure if I'm breaking the database or not. So instead of five, I will say three. Okay.

And are there any further changes that you would make regarding the pages the page involved in this task?

I don't think there are any minor changes. I think it's pretty good. It's pretty nice. Everything is just compressing one whole page. I guess I can even increase the pagination so that I can scroll down and see. Yeah, exactly. All the students are this is really comfortable for the people who is using the system. I don't think you need to change that. And also the design is kind of attractive. I can see that also, you have the binary in green because I belong to the green house. So this is pretty personalized, and I like it. Okay, oh one, if you are asking me for an evaluation from 1 to 5.

##### Task 2 – Filtering

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

Okay, maybe before I understood the question, because if this is about how easy it is, I would say this, this is between four and five. So if let's say five for being caring This is about how you see this, but I have Yeah, minor changes that I could mention you because I was suspecting

First of all, this specific will not say we know it's a pop up. It's a little bit confusing for me as I told you because these are real filters. And this is this looks like it's a different type of filters. So maybe a short explanation in this specific you know sides about what is happening here and what is happening here should be convenient. On the other hand, I understand why you use the drop down buttons for you know, just putting all these filters it groups convenient, but at the same time it's a little bit annoying because you mark something here you hide it and now you don't see it anymore unless you make the click here right so I don't know if there is a better option of share showing the other filters this looks convenient because of the number of filters but don't look don't seems convenient in order to have a big overview of what type of filters you are having. So my suggestion would be that as soon as you do click in one specific filter in certain specific books he'll on top or something like that appear something like filtering by whatever filter by whatever filter by whatever. I don't know if you understand the for example when you go to Amazon and you select one specific filter on top of the search box appear that a specific filter, I understand. So I have the overview of what type of filters I am applying right now because Okay, only filtering by gender and color house is not a big deal. But whenever you have to filter for more things, maybe it is start to be a little bit more difficult to track. Okay, another scene that I will watch. Sorry for the amount of comments. It's I don't know how difficult it is but maybe changing the color of the text. Depending on the house. Good big opinion taking into account that this is a very Pre primary information. I mean, the this is about houses. So the house is one of the most important data, it could be convenient to mark at least this specific text with the color so that I can identify super quick. But it's definitely a suggestion.

### Task 3 – Adding Staff

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?  
This is about how easy it was I will say four this time

What changes would you make regarding this

Okay, starting from the beginning. As I told you before, I think this is kind of confusing I would I will use the space for putting the name but also next to the name a short description about what the specific role is this something like the check listed? Just sort of in a vertical way. The other half of the space just a short description about what the specific permissions has this specific role. okay, this this specific message is confusing for me because I am not adding nothing but still. It says that I am overriding Samsung. So apparently I cannot escape from here. Yeah. And then then the way in which the results are added should be, in my opinion a little bit more explicit. Again, like with the students a message saying this specific person was added as a blue housekeeper sounds like that is a is both is a confirmation of your actions. And the, let's say the summary of the changes that you did. But that's okay. So also I was not able to identify a yet here is the house yet? Nothing, nothing. Okay, again, taking into account that the house is one of the most important data, I don't understand why this is in the last place, the order of the columns, it's quite important. As the is this number, that number is the first one, right? Because it's a key, I think the house should be C for even the name. But this is my opinion. So column should be shorter taken into account importance. Okay.

### Task 4 – Approve/Deny house change Requests

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?  
Five

Any changes you would suggest regarding this task.

I am unconfident so I would need a confirmation of everything so I know I am not making a mess. Errors in this system could lead to a big mess. As an admin I feel the pressure of making sure I do things in the right way.

### Task 5 – Change permissions

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?  
I would like to say 10 because it was super clear and easy. This is really clear.

Any changes that you would make?

Yes. I would like a confirmation. I think the admin should be able to see every role in the system so that I can check on my own the results of the changes. I would like to double check that what I am doing is correct.

### Final questions

How would you describe your overall experience with the system?

I think that what I have seen is beautiful. Its nice. Its interactive. It seems practical, still there are certain usability bugs that need to be polished. So I would say that this is a very good minimum viable product but it is not mature yet.

What do you like the most about using the system?

I remember the previous version and I am really happy because it is a big change. I think that one of the most difficult parts of the software is managing the different roles. Last time it was not clear but this time it is quite clear. You did a nice job.

What do you like the least about the system?

The way which you are showing the information is not useful. You would have to expect lots of students so the scroll bar is not useful. Also the ordering of the columns is not the correct one.

Were there any other frustrations you experienced?

No. It's nice.

Changes made afterwards:

- Show selected filters
- Add remove all filters button
- Color circle next to house name
- Default options for drop down selectors
- Removal of "ROLE\_" prefix
- Confirmation/Error modals added
- Columns reordered
- Select student by search for house cup points submission
- Show/hide columns in separate button
- Search Pending house change requests
- Search box states "Search name/id"

## User test 2

### Task 1 – Uploading

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

I think that was very easy.

Are there any changes you would make regarding this task?

As far as I can see I'd be fine with this.

### Task 2 – Filtering

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

Once you know what to do it is easy. That's really nice.

Are there any changes you would make regarding this task?

I like it, cause I can filter a whole variety of things. That looks good.

### Task 3 – Adding User

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

I would still say very easy.

Are there any changes you would make regarding this task?

No

### Task 4 – Approve/Deny house change requests

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?  
Very easy.

Are there any changes you would make regarding this task?  
Nope.

#### Task 5 – Changing permissions

On a scale of 1 to 5, where one is very difficult and five is very easy, how would you rate this task?

Very easy.

Are there any changes you would make regarding this task?  
No changes

#### Final Questions

How would you describe your overall experience with the system?

It is easy and I like it. It's very neat. It's very clear.

What do you like most about the system?

The easy access to adding people changing roles. It should make life much easier.

Was there anything you do not like about it?

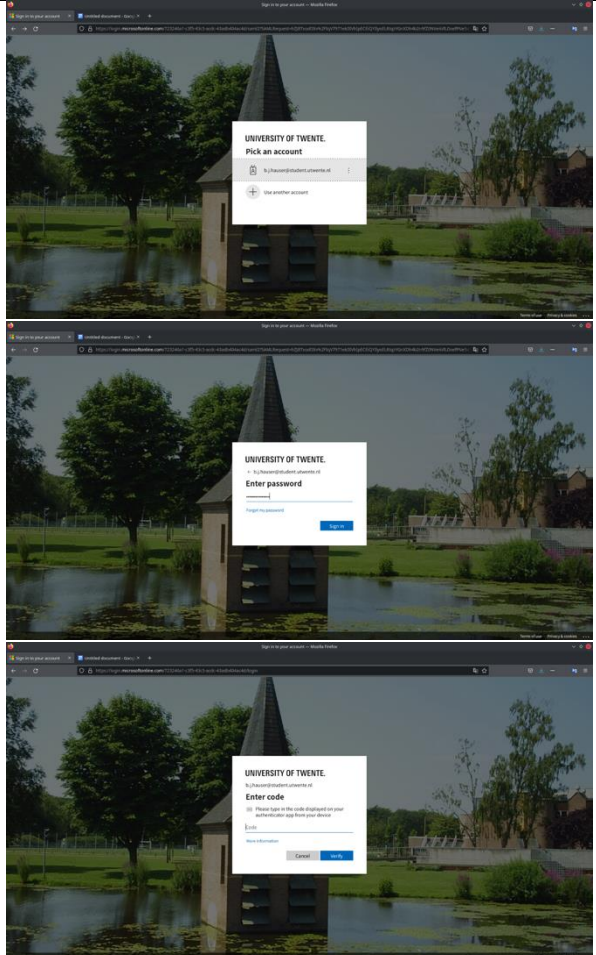
At the moment not.

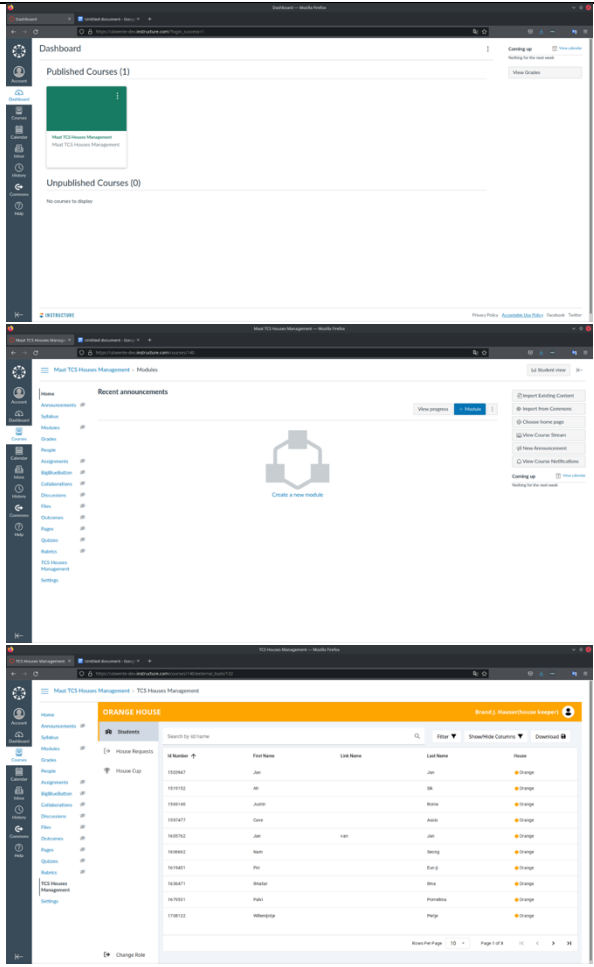
Changes made afterwards:

- None

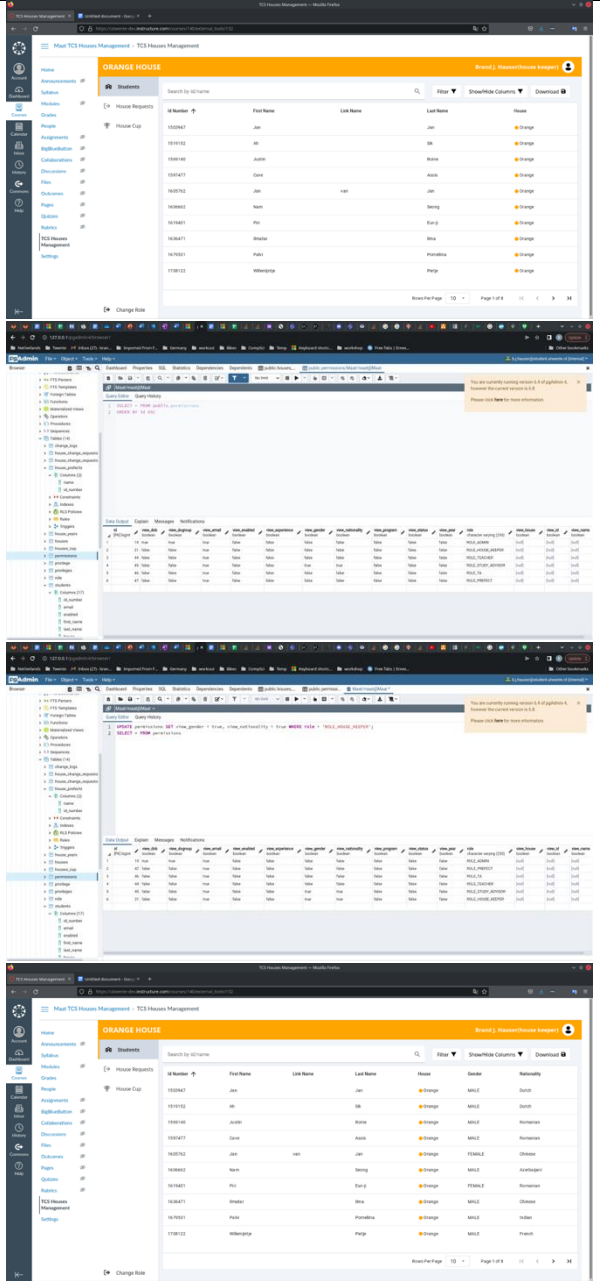
## Appendix C - System Testing

### User

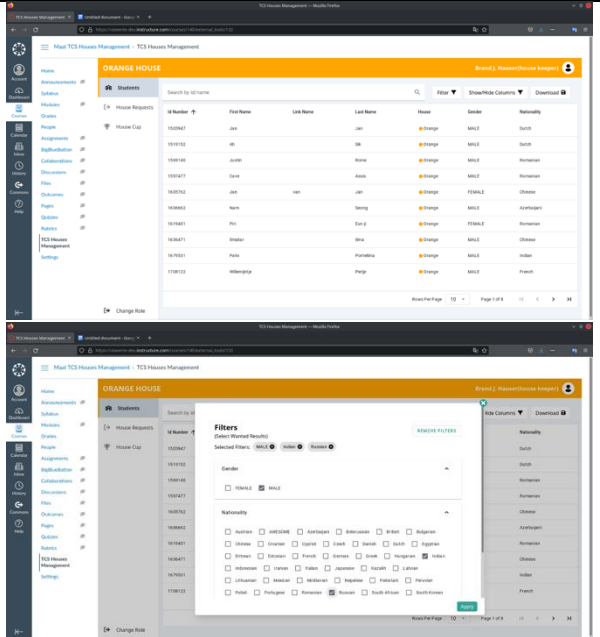
User Story	As a user, I want to login through Canvas for accessing the system
Test Description	Login through the Canvas environment using an existing UTwente account and verify that the application is accessible.
Initial Conditions	Existing UT Canvas account. Frontend and backend servers running and connected to database.
Test Input	Valid UT username and password
Test Procedure	In Firefox web browser, navigate to the Canvas development course at <a href="https://utwente-dev.insrtucture.com/courses/140">https://utwente-dev.insrtucture.com/courses/140</a> . Log into Canvas using valid UT credentials. Click the link to the TCS House Management plugin. Verify that the plugin is accessible.
Expected Behavior	Login credentials are accepted. Upon clicking the plugin link, application will open within Canvas window and display the logged in user's homepage with username and color-coded banner according to house.
Actual Behavior	 <p>The actual behavior is captured in three screenshots of the University of Twente login process. The first screenshot shows the login page with a 'Pick an account' dialog. The second screenshot shows the 'Enter password' dialog. The third screenshot shows the 'Enter code' dialog for two-factor authentication.</p>

	 <p>Actual behavior as expected</p>
Evaluated Success	100%
Errors Observed	none
Errors fixed	N/A

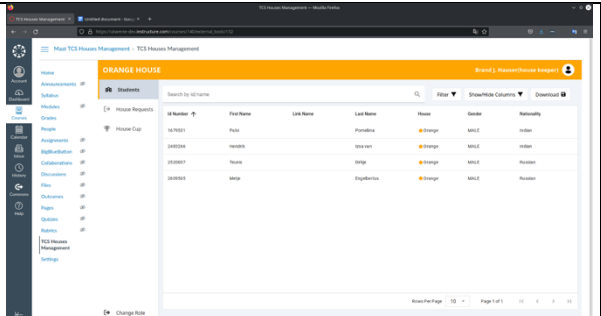
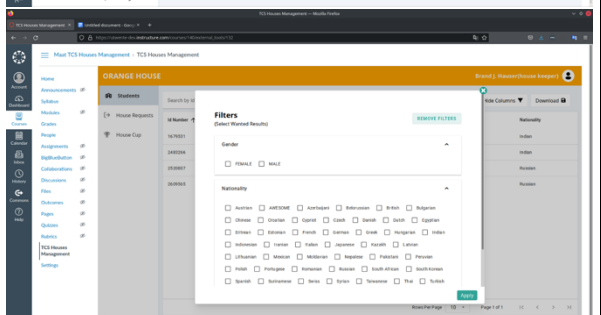
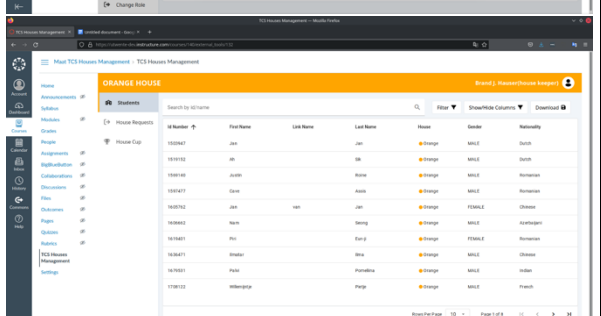
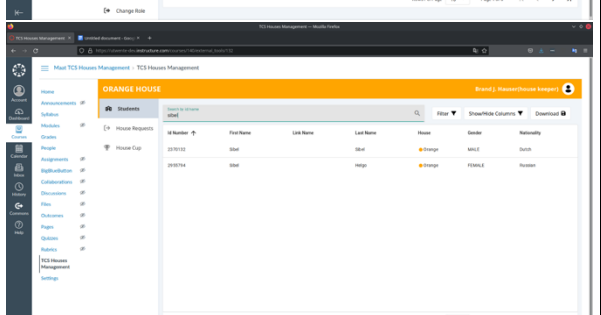
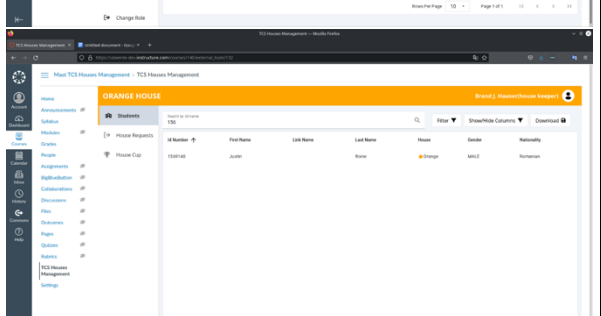
User Story	As a user, I want the system to automatically limit the information viewable to me based on privacy regulations when I log in
Test Description	When a user logs into the system the page showing student information should only display the information that is allowed to be displayed to a user of that role as set by the admin user.
Initial Conditions	Test user set as Housekeeper. Permissions of housekeeper set to allow student id, name, house and email.
Test Input	N/A
Test Procedure	Log into application with housekeeper role and verify that only the allowed information is visible. After verification, modify permissions of housekeeper directly in the database to allow

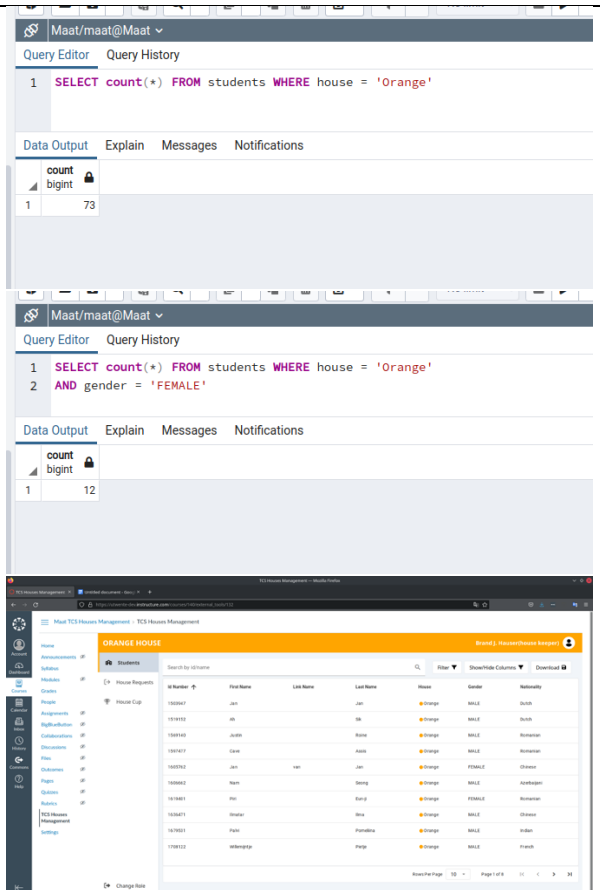
	viewing of student gender and nationality. Refresh webpage and verify that change has occurred. Repeat for teacher role.
Expected Behavior	On initial login Student id, name, house and email will be only data points visible. After modification of permissions, gender and nationality will be visible as well.
Actual Behavior	
Evaluated Success	100%
Errors Observed	None
Errors fixed	N/A

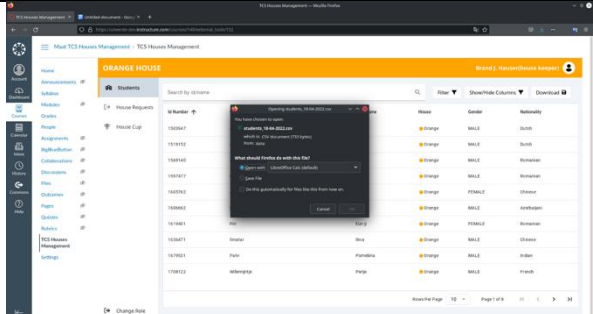
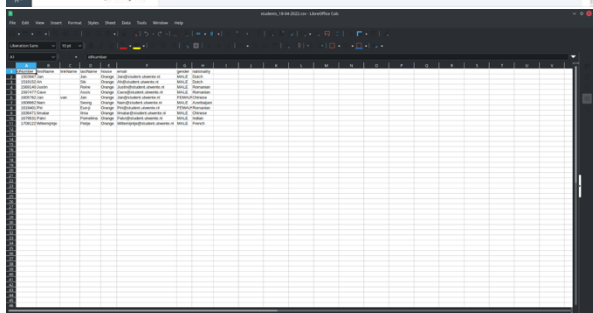
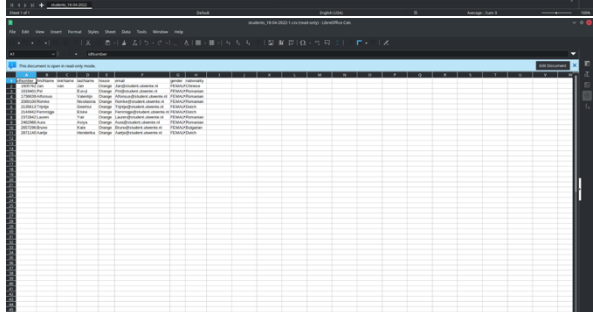
User Story	As a user, I want to filter the information I am allowed to view
------------	--

Test Description	The student data table should be filterable by all columns visible to the user with the exception of name and id and email. Name and id columns should be searchable.
Initial Conditions	Logged into application as housekeeper with permissions to view student id, name, email, house, nationality and gender. Fictitious student information with variety in fields loaded into database.
Test Input	Filter selections: gender = male, nationality = Russian and Indian Search student name Sibel Search student id 156 . . .
Test Procedure	Click filters button. Verify that values from gender and nationality columns are selectable. Select values and click apply. Verify that data is filtered according to selections. Open filter menu and click remove all filters. Verify that values are unchecked. Click apply and verify that resulting table is identical to initial table. Verify that searching for student name and id number results in the filtering of data according to the input values.
Expected Behavior	Filtering by selected values results in a list of students who match the selections. All other students are removed from the table. Searching results in a list of students that match the search values entered. All other students are removed from the table.
Actual Behavior	

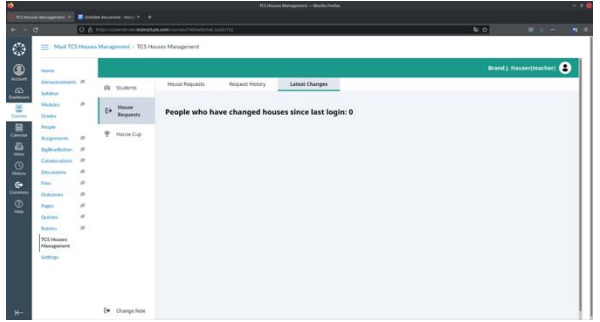


	    
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

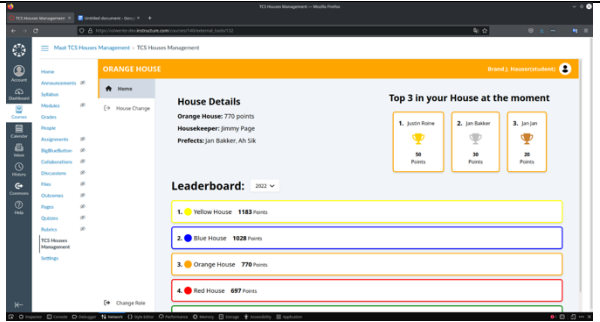
User Story	As a user, I want to export the information in the table
Test Description	While viewing any table in the application the data visible should be exportable to CSV by clicking the download button. The resulting CSV should contain only the information in the visible table including all subsequent pages. Filters applied to viewed table should be applied to CSV as well.
Initial Conditions	Logged in as housekeeper. Fictitious student information loaded into database. No filters applied. 73 students are assigned to the orange house, 12 of which are female.
Test Input	Filters: gender = female
Test Procedure	Begin with view of unfiltered student information. Click download and verify that resulting CSV contains all data from current table. Filter data based on gender and click download. Verify that exported data matches data of viewed table.
Expected Behavior	Initial CSV export contains all data of students in the orange house which corresponds to the house of the logged in user. Second CSV contains only female students.
Actual Behavior	 <p>The screenshots illustrate the test process. The first screenshot shows the Query Editor with the query: <code>1 SELECT count(*) FROM students WHERE house = 'Orange'</code>. The Data Output shows a count of 73. The second screenshot shows the query updated to: <code>1 SELECT count(*) FROM students WHERE house = 'Orange'</code> and <code>2 AND gender = 'FEMALE'</code>. The Data Output shows a count of 12. The third screenshot shows the 'Orange House' student list in the application interface, with a download button visible.</p>

	  
Evaluated Success	25%
Errors Observed	Data is downloaded to CSV but only the currently viewable page is downloaded containing 10 students.
Errors Fixed	This error was not fixed.

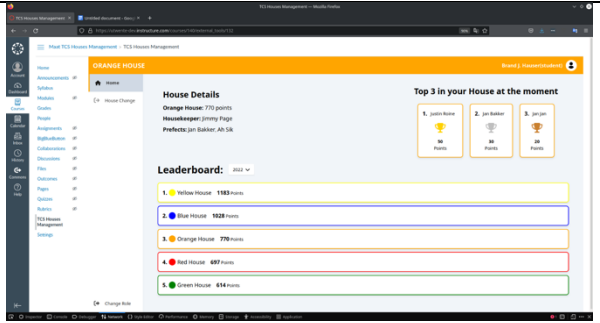
User Story	As a user, I want to be able to filter the data based on which students have changed houses since I last viewed the data
Test Description	Users should be able to view the students that have changed houses since the user last logged in.
Initial Conditions	Logged in with teacher role. No house changes have been approved since last logged in. Fictitious student data loaded.
Test Input	None
Test Procedure	Click the link to the house change requests view and then click latest changes. Verify that page shows latest changes. In this case empty list as none have been approved. Previous testing has shown accurate behavior when changes have happened.

Expected Behavior	Display a list of the latest house changes.
Actual Behavior	Error message received After fix: 
Evaluated Success	0%, 100% after fix as stated below
Errors Observed	Null pointer error
Errors Fixed	Source of null pointer was an if statement that had an incomplete list of roles to check. Upon fixing, test works properly.

### Student

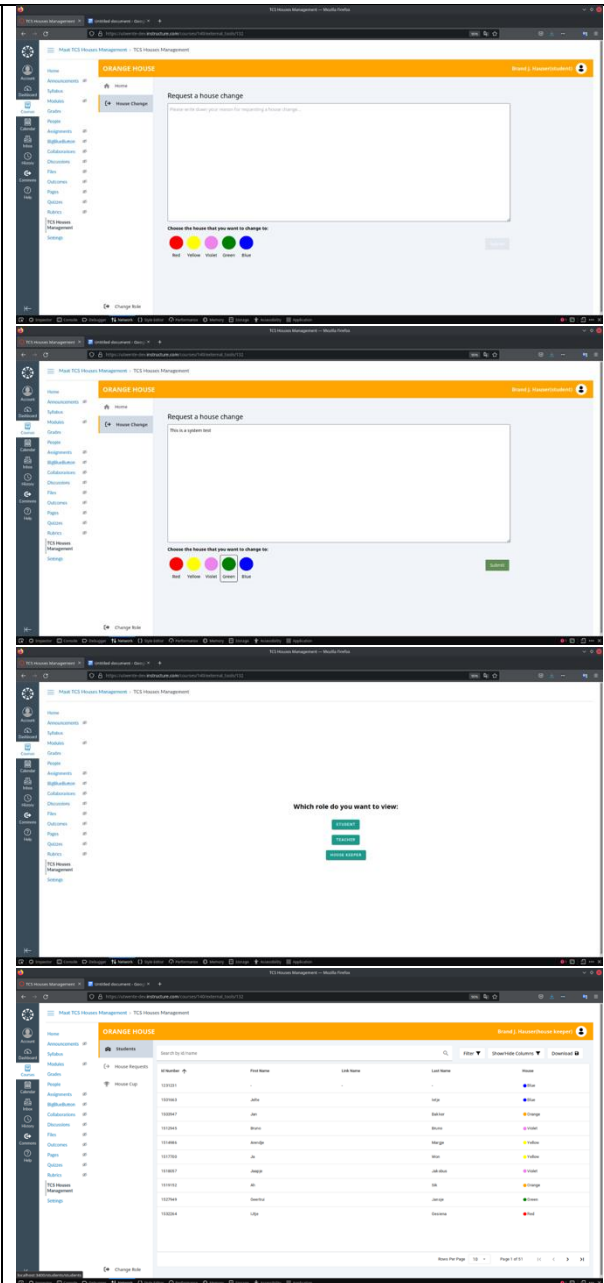
User Story	As a student, I want to be able to see who my housekeeper and prefect(s) are
Test Description	When students log in, their home page should display the names of their housekeeper and prefects.
Initial Conditions	Logged in as student in the orange house. Housekeeper is assigned to use with name Jimmy Page. Prefects assigned as Jan Bakker and Ah Sik.
Test Input	None
Test Procedure	Log into application as student in orange house
Expected Behavior	Display student home page with housekeeper name and prefect names.
Actual Behavior	
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

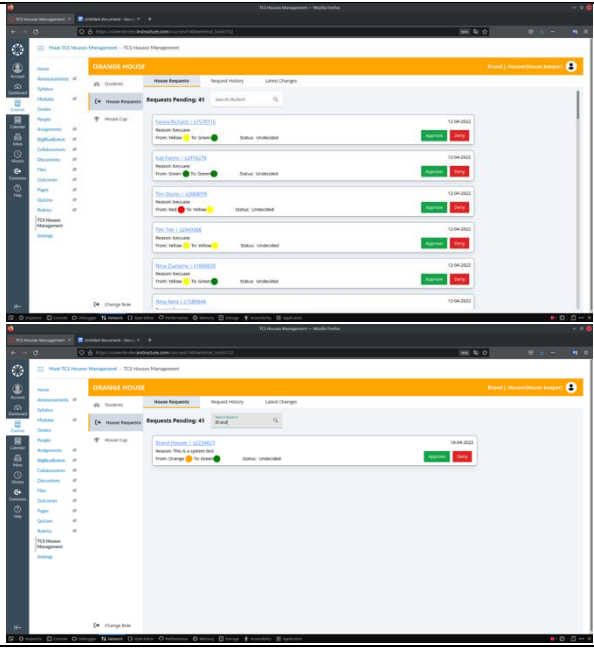
User Story	As a student, I want to be able to view the current points of the house competition
------------	---

Test Description	When students log in, they should be able to see the current points per house of the house cup competition.
Initial Conditions	Logged in as student in the orange house. 1000 house cup points entries in database representing 5 houses and many students.
Test Input	None
Test Procedure	Log in as student and view home page. Verify 5 houses and their scores are displayed.
Expected Behavior	Scores of the 5 houses are displayed upon successful log in as student.
Actual Behavior	 The screenshot shows a web application interface for 'TCC House Management'. The main content area displays 'House Details' for the 'Orange House' with 770 points. Below this is a 'Leaderboard' for 2022 showing the top 3 houses: Yellow House (1183 points), Blue House (928 points), and Orange House (770 points). Other houses listed are Red House (697 points) and Green House (614 points). The interface includes a sidebar with navigation links like Home, Accounts, Users, and a top navigation bar with a 'House Change' button.
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

User Story	As a student, I want to have the ability to request changing my current house
Test Description	Students should be able to click the link to submit a house change request. Upon submission, the house change request should be present in the database as well as in the pending house change requests view for other user roles such as housekeepers.
Initial Conditions	Logged in as student in the orange house as Brand Hauser.
Test Input	House change request variables: Old house = Orange Target house = Green Explanation = This is a system test
Test Procedure	Log in as student in the orange house with name Brand Hauser. Click House Change link. Fill in information and click submit. Switch to housekeeper role and view pending changes. Verify that new request matching given variables is present.
Expected Behavior	After submission, the new house change request should be visible in the housekeeper role pending request view.

## Actual Behavior



	
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

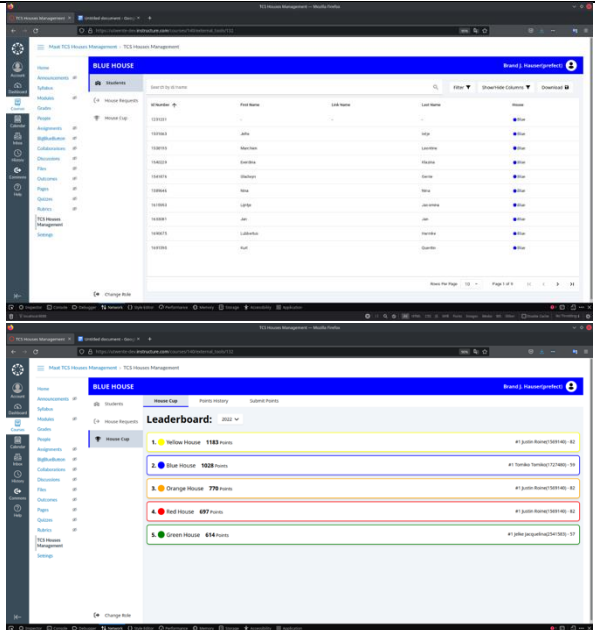
User Story	As a student, I want to see the house I'm currently in
Test Description	When a student logs in to the application the banner at the top of the page should display both the name and color of the house that the student is in.
Initial Conditions	Log in as user in the orange house.
Test Input	Change house to blue house
Test Procedure	Log in as student in the orange house. After verifying that banner color and house name are correct, change house of student to blue house and refresh. Verify that color and name have changed appropriately.
Expected Behavior	Banner color and house name reflect orange house assignment. After change of house assignment in database both banner color and name have changed. Housekeeper and prefect data displayed should also change.

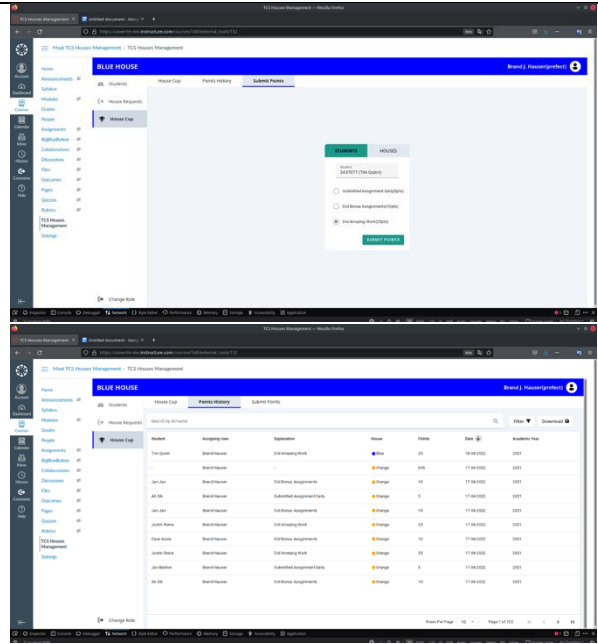
Actual Behavior	
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

User Story	As a student, I want to see the top three people who have received the most house points in my house
Test Description	When a student logs in the home page should display the names and points of the top three students within the same house.
Initial Conditions	Student set to blue house. Fictitious student data in database with 1000 entries in the house cup corresponding to student data.
Test Input	None
Test Procedure	Log in as student in the blue house and verify that the top three students are visible.
Expected Behavior	Top three students with their names and points are visible upon successful log in.
Actual Behavior	
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A



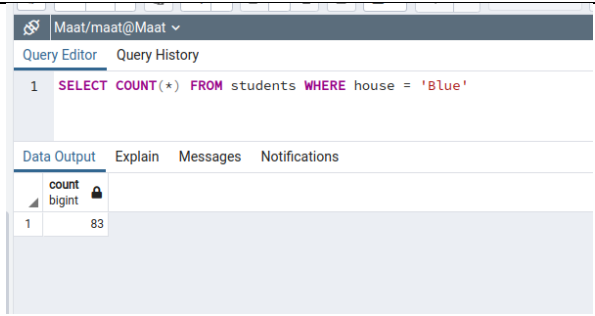
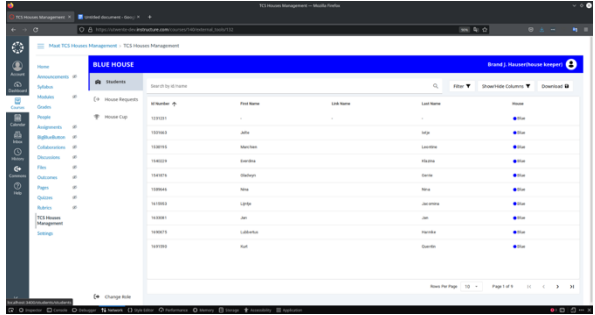
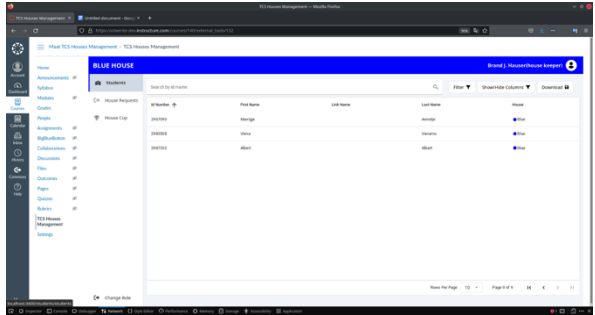
## Prefect

User Story	As a prefect, I want to submit house points
Test Description	Prefects should be able to submit house cup points to students.
Initial Conditions	Logged in as prefect user. Fictitious student data loaded into database.
Test Input	Student Tim Quinn House blue Points 25 for submitting early
Test Procedure	Log in as prefect user and click button for house cup. Click link to submit points. Enter data as above and click submit. Check house cup points history to verify points submitted as described above.
Expected Behavior	Upon completion of points submission, clicking link to history should display list of points entries including an entry to match the one submitted with variables as described above.
Actual Behavior	

	
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

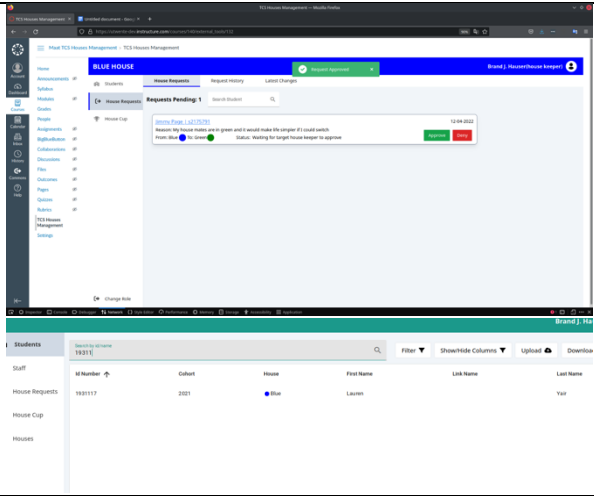
### Housekeeper

User Story	As a housekeeper, I want to see which students are in my house
Test Description	Housekeepers should be able to view all of the students in their house on their home page.
Initial Conditions	User set to housekeeper of blue house. Fictitious student data in the database.
Test Input	None
Test Procedure	Log in as housekeeper. Verify that students in view are all in blue house and comprise the total of the blue house students.
Expected Behavior	List of students will be displayed. All students are in blue house and total number of students matches the number in the database that are in the blue house.

Actual Behavior	   <p>8 pages of 10 students each plus 3 on final page = 83 students</p>
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

User Story	As a housekeeper I want to approve and deny house change requests
Test Description	Housekeepers should be able to click the deny button on a house change request and have that request be finalized as being denied. Approved request must be approved by both the target housekeeper and the old housekeeper. A request that has been approved by one will have a status that it is pending approval by the other.
Initial Conditions	Logged in as blue housekeeper. Fictitious data in database including house change requests.
Test Input	Denial explanation = This is a system test
Test Procedure	Log in as a housekeeper. Click link to house change requests page. Select pending request to deny. Give denial explanation as listed above. Verify that request is in house change history as

	<p>being denied after submission and that it has disappeared from the pending page. Select another request to approve. Verify that status has changed to pending other housekeepers' approval. Select another request that has been approved by the other housekeeper and verify that it is marked approved, and that student has changed house.</p>
Expected Behavior	<p>Denied request is set to status = denied          Approved request is set to pending other          Approved request is finalized as approved and student has been moved to another house.</p>
Actual Behavior	

	
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

### Admin

User Story	As the sorting hat I want to be able to adjust privileges based on user roles
Test Description	Role based permissions to view various attributes of the student data are set by the admin user on the permissions page.
Initial Conditions	Base permissions are set. Logged in as admin.
Test Input	Set permissions for Teacher: view gender and nationality Prefect: view none Housekeeper: view dogroup
Test Procedure	Log in as admin user and click staff link. Click link to Permissions page. Set permissions as described above. Verify changes in database.
Expected Behavior	Permissions in database change as described.

## Actual Behavior

The screenshots illustrate the T3 House Management application interface. The first screenshot shows a list of house change requests with columns for ID, Status, House, and Request Name. The second screenshot shows a detailed view of a specific request, including fields for Requester, Requested House, and Requested Date. The third screenshot shows the 'Viewable Permissions' section, which lists various roles and their associated permissions. The fourth screenshot shows a SQL query editor with a query that selects data from a table named 'house\_change\_requests'.

Evaluated Success

100%

Errors Observed

None

Errors Fixed

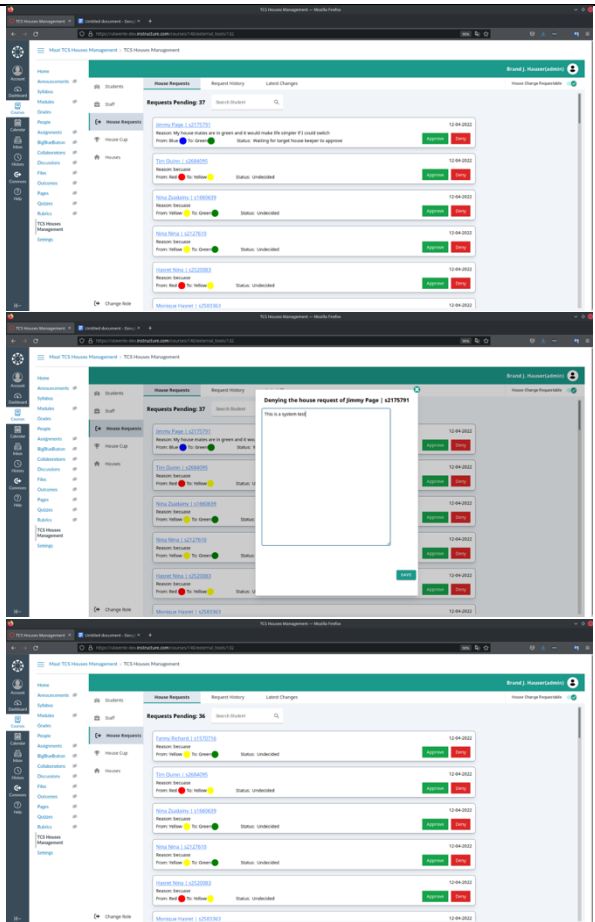
N/A

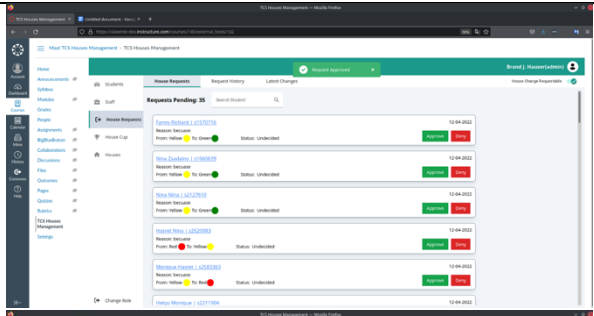
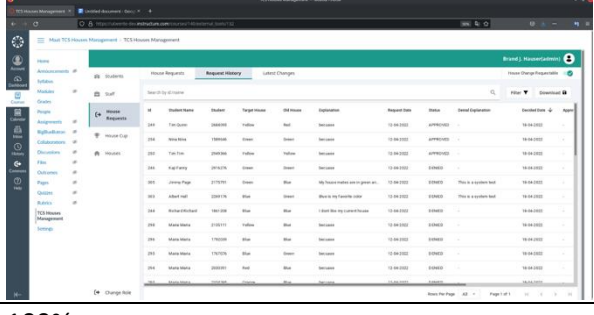
User Story

As the sorting hat, I want to approve and deny house change requests

Test Description

Admin users can approve and deny house change requests by clicking the appropriate buttons.

	Admin, unlike housekeeper, is single approval role.
Initial Conditions	Logged in as admin. Fictitious data in database including house change requests
Test Input	Explanation = This is a system test Deny student Jimmy Page blue to green Approve student Tim Quin red to yellow
Test Procedure	Log in as admin and navigate to pending house change request view. Select a request to deny with the explanation above. Select request to approve. Verify both have disappeared from pending requests page and are in the history page as finalized.
Expected Behavior	Both requests are finalized with the appropriate status
Actual Behavior	 <p>The screenshots illustrate the system's behavior during a test. The first screenshot shows the 'Requests Pending: 37' list with a request for Jimmy Page (L4375791) from blue to green. The second screenshot shows a modal dialog for denying the request with the explanation 'This is a system test'. The third screenshot shows the 'Requests Pending: 36' list after the request has been denied, and the 'Request History' tab shows the request as finalized.</p>

	 
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

User Story	As the sorting hat, I want to be able to add and remove users
Test Description	Staff members exist in the user's data table. The admin user should be able to add and delete staff members as they are added to the team of UT staff that are involved in the management of the houses.
Initial Conditions	Logged in as admin. Fictitious student data loaded into database.
Test Input	Id number = 99999999 Name = John Fogherty Email = <a href="mailto:j.fogherty@utwente.nl">j.fogherty@utwente.nl</a> House = none Role = Teacher
Test Procedure	Log in as admin. Click link to staff page. Click button to add staff member. enter details as above. Click save. Verify staff member has been added. Search for staff member added above. Click Delete and verify that the data has been removed.
Expected Behavior	Staff member will be added with details as stated above. Staff member will then be deleted



## Actual Behavior

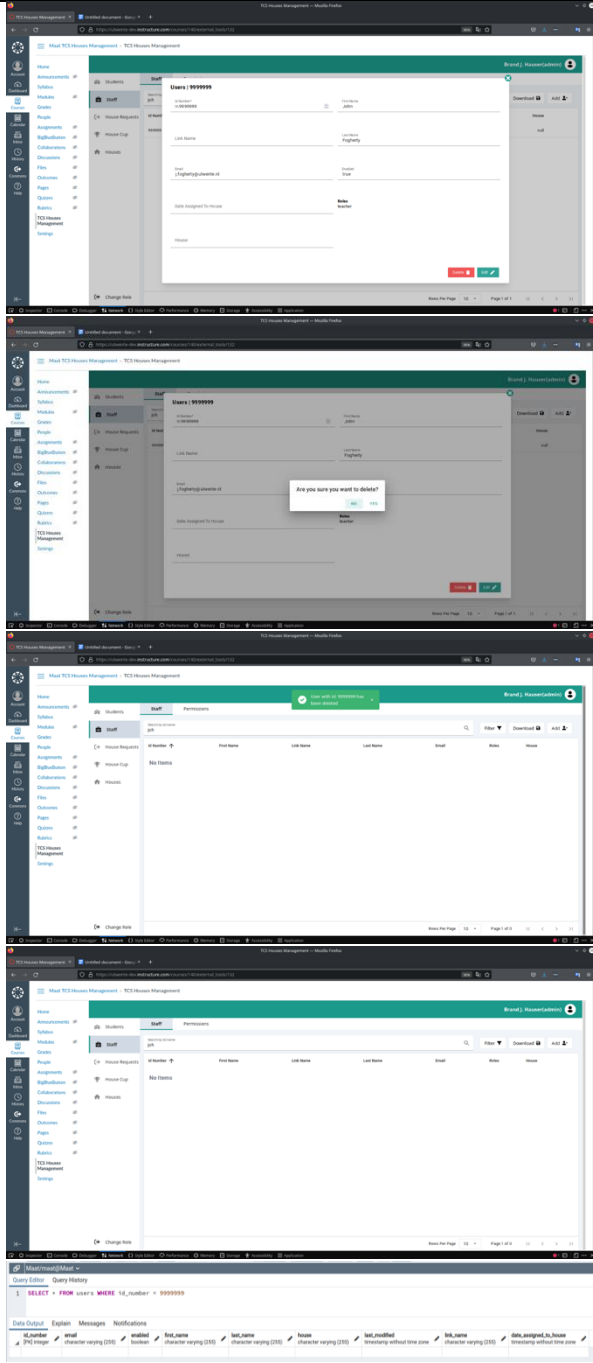
The screenshots illustrate the process of adding a new user in the T2T House Management system. The first screenshot shows a list of existing users. The second and third screenshots show the 'New Users' form where a new user is being added. The fourth screenshot shows the updated user list. The fifth screenshot shows the SQL query editor with a query to select users with ID 999999.

**Query Editor - Query History**

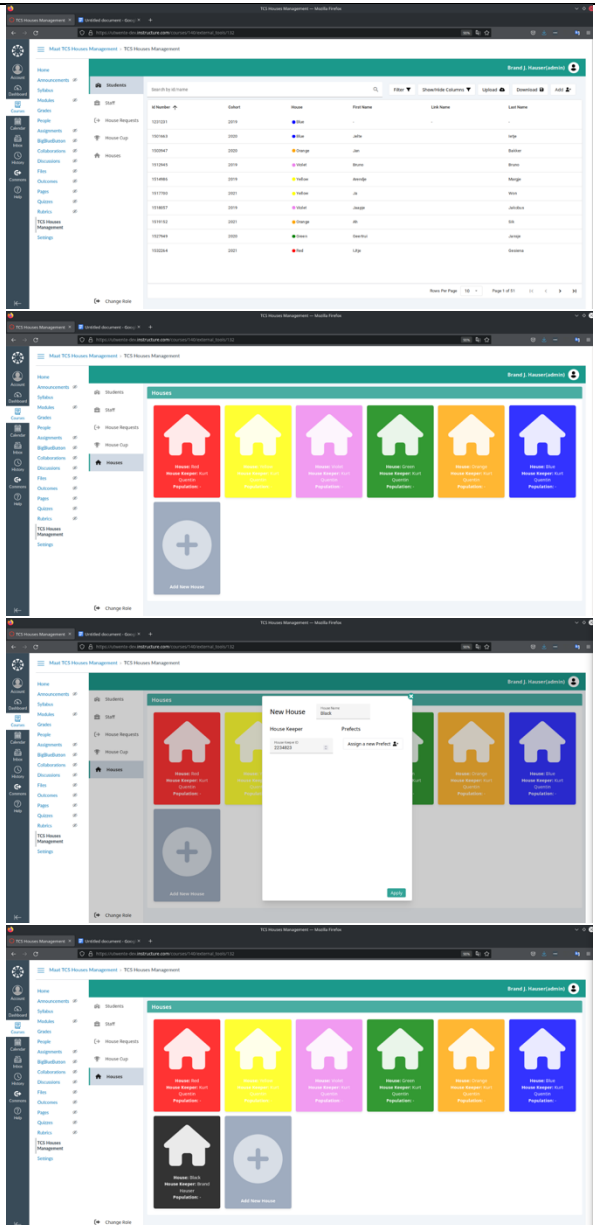
```
SELECT * FROM users WHERE id_number = 999999
```

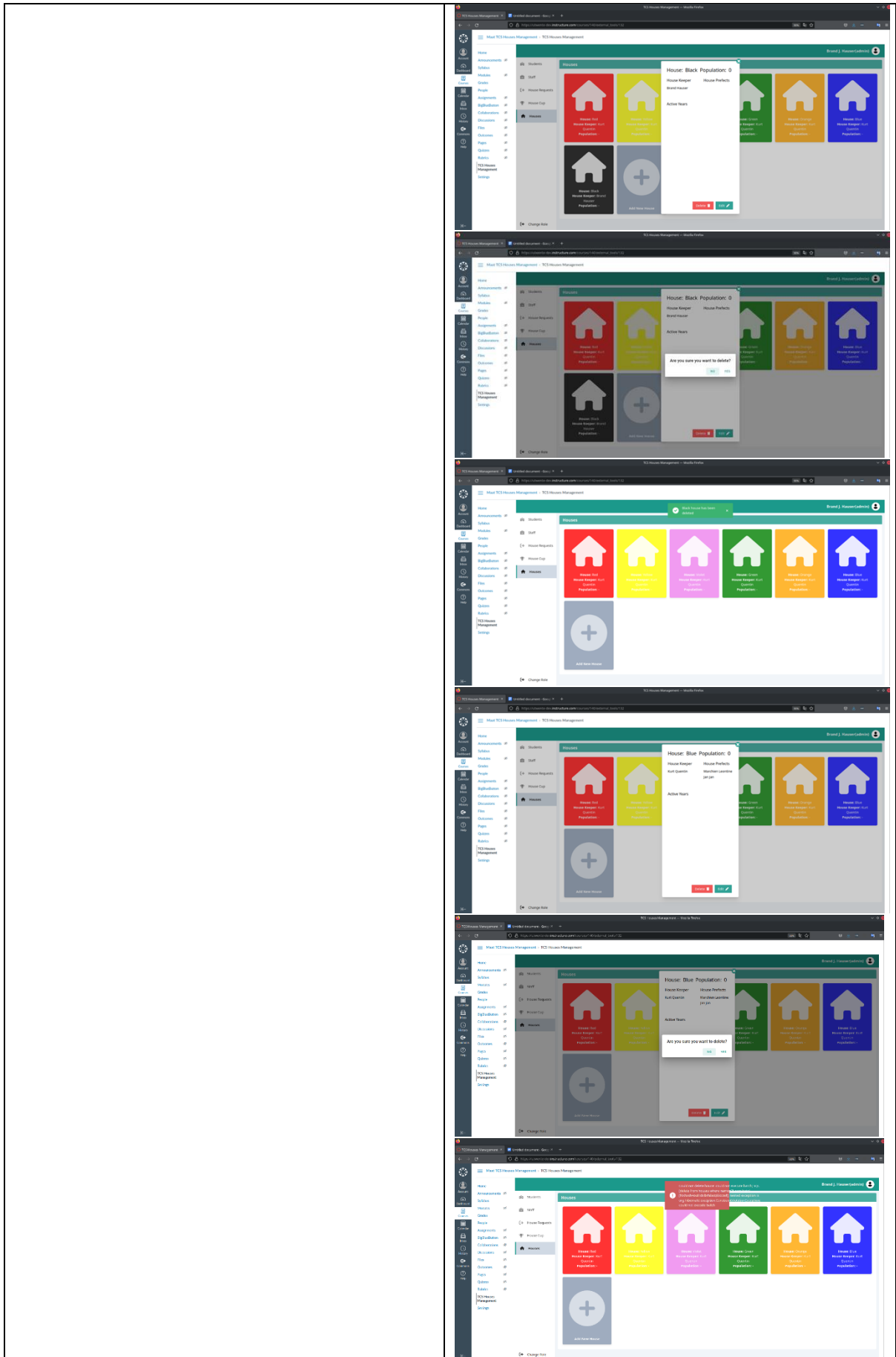
**Data Output**

id_number	email	enabled	first_name	last_name	house	last_modified	last_name	data_assigned_to_house
999999	john@t2t.com	True	John	Highly	House	2023-10-10 10:10:10	Highly	House

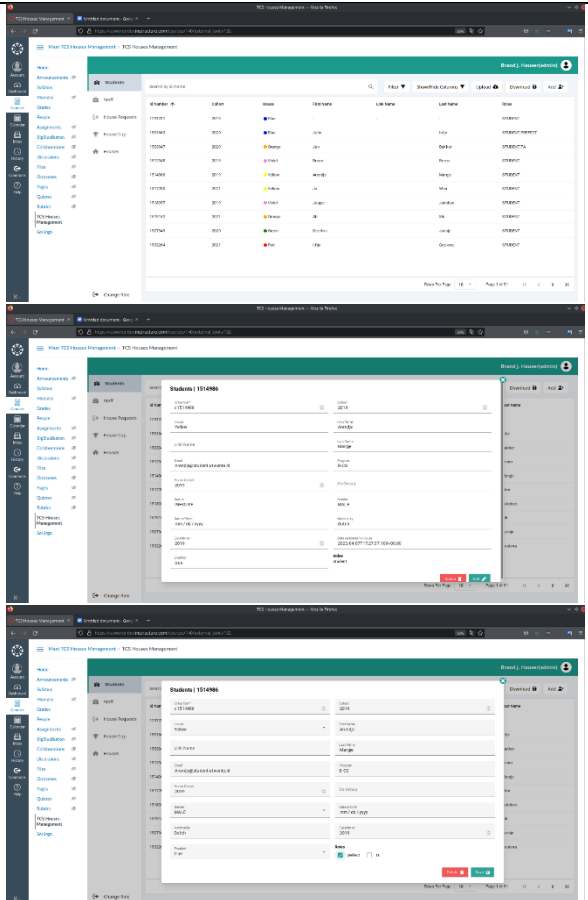
	
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

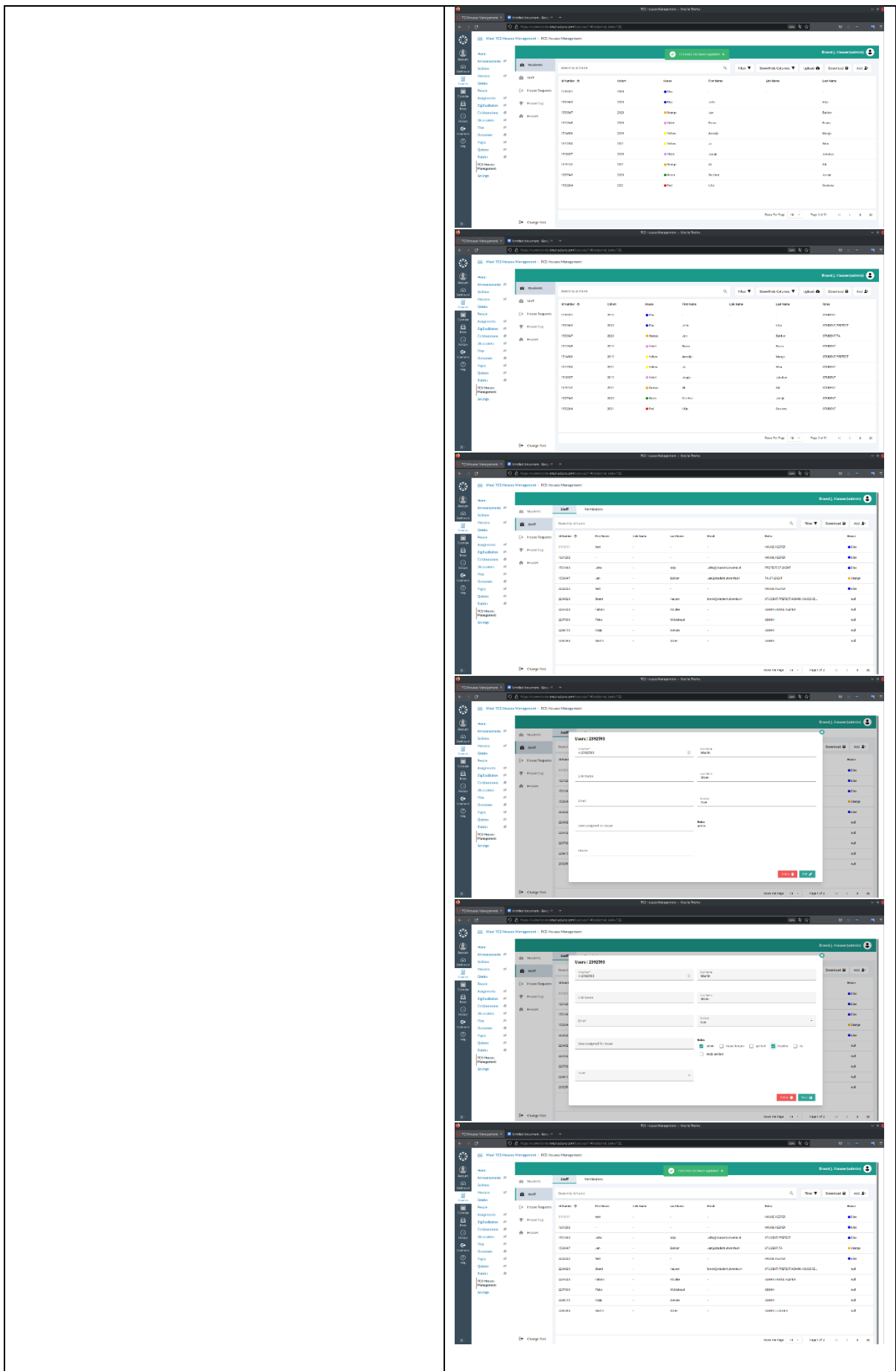
User Story	As the sorting hat, I want to be able to add or remove houses
Test Description	Houses can be added and removed by the admin user on the houses page. A house that has no data of students, users, cup points, etc. attached to it can be deleted. A house that has data attached cannot be deleted.

Initial Conditions	Logged in as admin. Six houses exist in the database with data attached.
Test Input	Name = black Housekeeper = Brand Hauser
Test Procedure	Navigate to houses page. Click button to add house. Enter details as stated above. Click apply. Verify that house has been created as specified. Select the new house. Click Delete. Verify deletion. Select Blue house. Click delete. Verify that house cannot be deleted.
Expected Behavior	House will be added with specified information. Same house will then be deleted. Attempting to delete the blue house will result in an error message stating that the house cannot be deleted.
Actual Behavior	



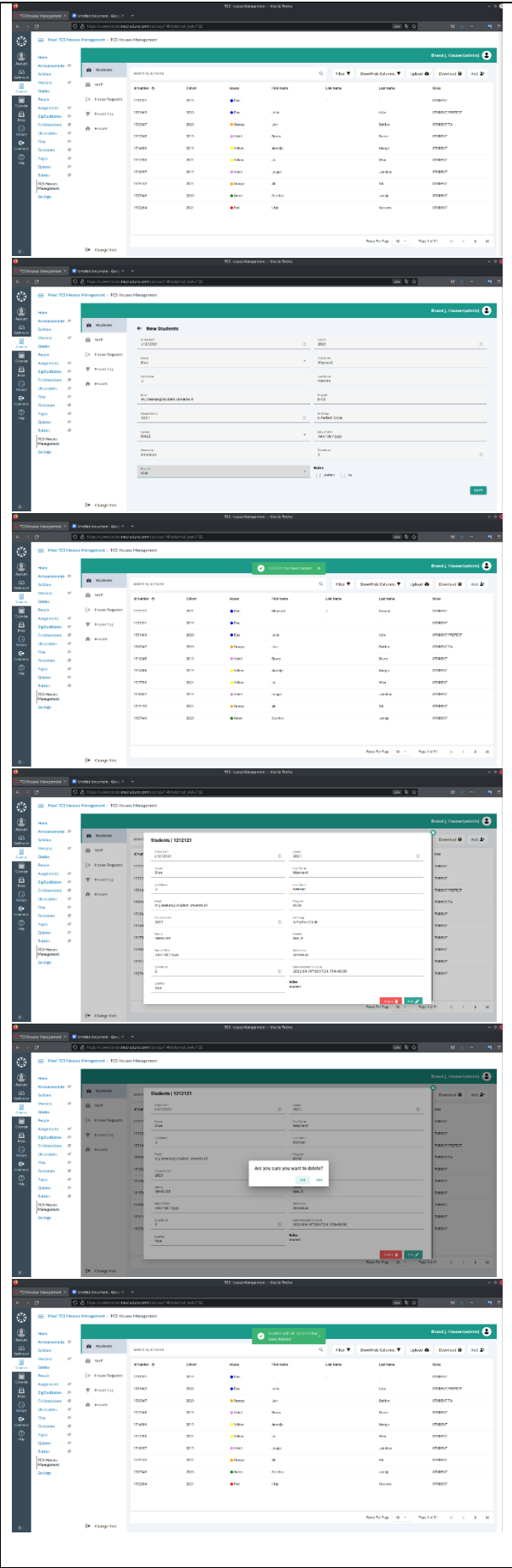
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

User Story	As the sorting hat, I want to be able to assign user roles
Test Description	Admin user can assign roles to staff and student users in the respective pages.
Initial Conditions	Logged in as admin. Fictitious staff and student data entered into database.
Test Input	N/A
Test Procedure	Log in as admin. Select student with id number 1514986. Click edit button. Assign role as Prefect. Click save and verify assignment success. Click link to staff page. Select user with id number 2392593. Click edit button. Add role as Teacher. Click Save and verify that change has been made.
Expected Behavior	Student will have the additional role of prefect added. Staff member will have the additional role of teacher added.
Actual Behavior	 <p>The actual behavior is demonstrated through three screenshots of the NIS Student Management system. The first screenshot shows the 'Students' list with ID 1514986 highlighted. The second screenshot shows the 'Edit Student' form where the role is set to 'Prefect'. The third screenshot shows the 'Edit Student' form with the role set to 'Prefect' and the 'Save' button highlighted.</p>



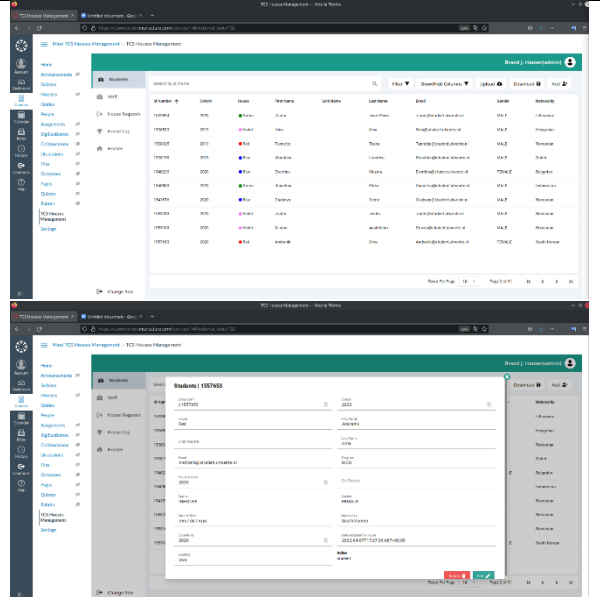
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

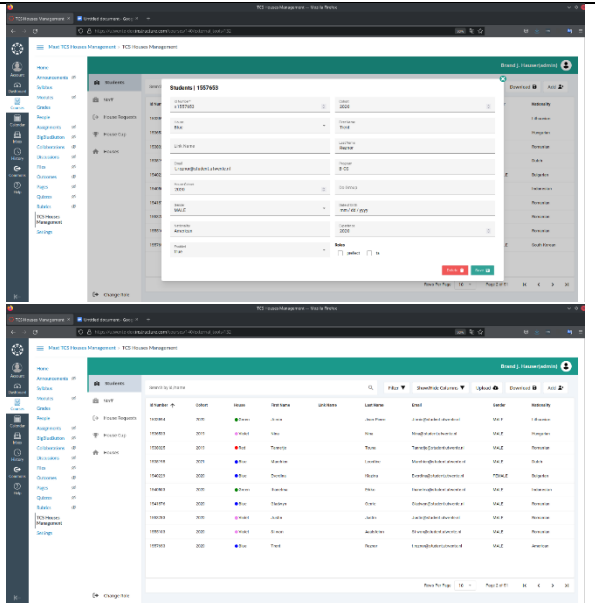
User Story	As the sorting hat, I want to be able to manually add and delete students
Test Description	Admin users can add new students using the add student button. All data entered will then be recorded and added to the system. Students can also be deleted by clicking the delete button upon which the data will be deleted.
Initial Conditions	Logged in as admin. Fictitious data entered into database.
Test Input	<p>Id number = 1212121</p> <p>Cohort = 2021</p> <p>House = Blue</p> <p>Name = Maynard J Keenan</p> <p>Email = <a href="mailto:m.j.Keenan@student.utwente.nl">m.j.Keenan@student.utwente.nl</a></p> <p>Program = B-CS</p> <p>House Cohort = 2021</p> <p>DoGroup = A perfect circle</p> <p>Gender = Male</p> <p>Nationality = American</p> <p>Experience = 2</p> <p>Enabled = true</p> <p>Roles = none</p>
Test Procedure	Log in as admin. Click button to add student. Enter details as specified above. Click save. Verify that student has been added. Select newly added Student. Click delete. Verify that student has been deleted.
Expected Behavior	<p>Student will be added with the specified data.</p> <p>Student will be deleted.</p>

[illegible]



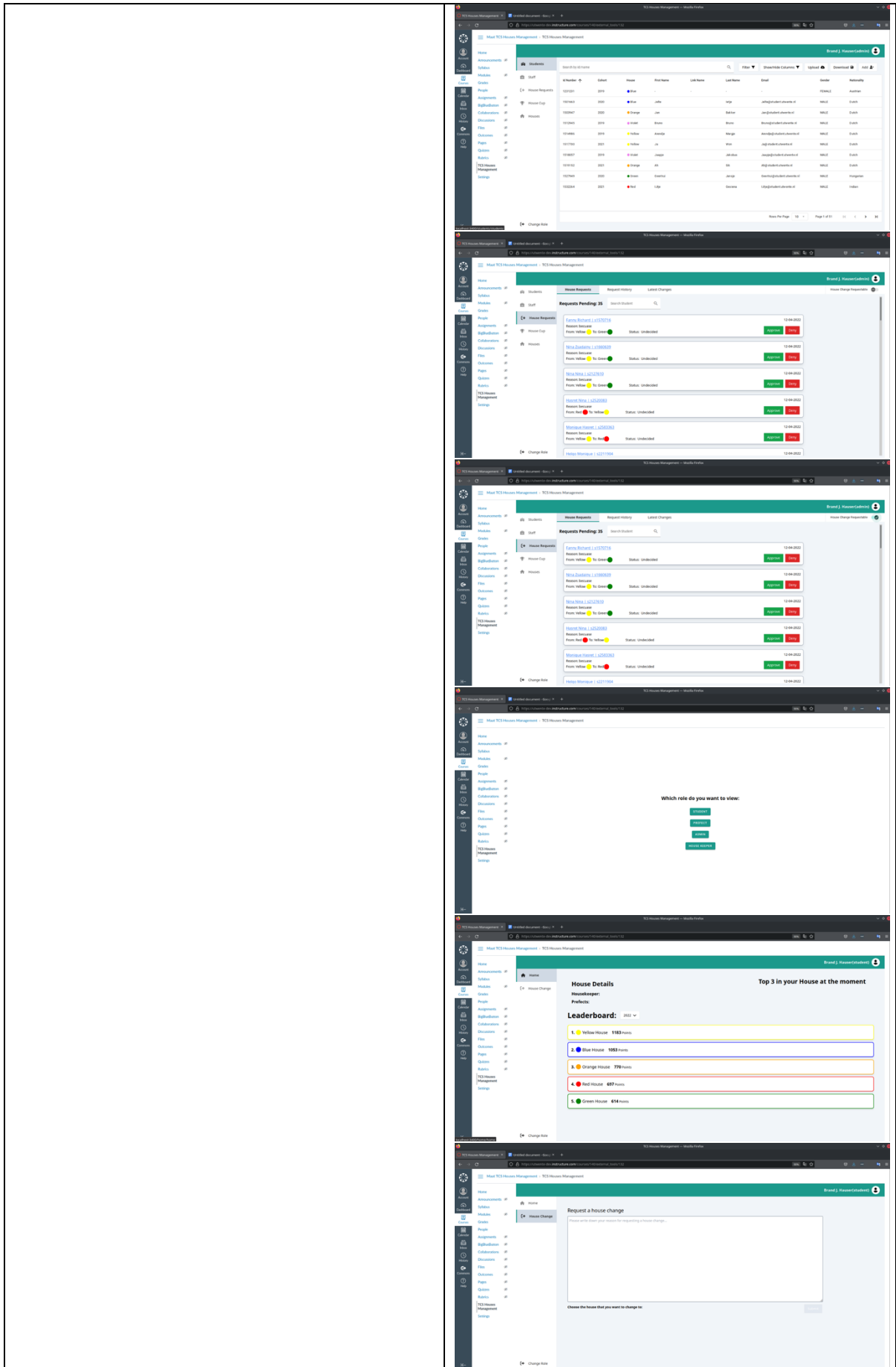
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

User Story	As the sorting hat, I want to be able to manually assign students to houses. As the sorting hat, I want to be able to manually edit student data
Test Description	Amin users can edit the data of a student. This includes adding a student to a house as well as changing the name, email, etc.
Initial Conditions	Logged in as admin. Fictitious student information entered into database.
Test Input	Name = Trent Reznor House = Blue Email = <a href="mailto:t.reznor@student.utwente.nl">t.reznor@student.utwente.nl</a> Gender = Male Nationality = American
Test Procedure	Log in as admin. Select student with id number 1557653. Click edit button. Enter data as specified above. Click save. Verify that student with id number specified has been changed.
Expected Behavior	Student information will be changed as specified.
Actual Behavior	

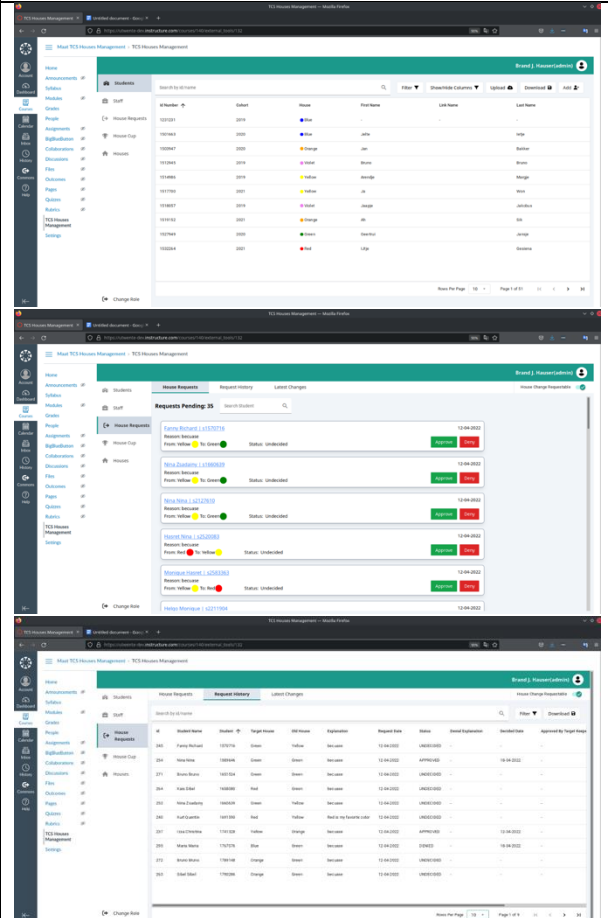
	
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A
User Story	As the sorting hat, I want to be able to turn off the ability of students to submit house change requests
Test Description	Admin users can click the button to turn on or off the ability to submit house change requests. When off students will not be allowed to submit new requests.
Initial Conditions	Logged in with both admin and student roles. Fictitious data entered into database.
Test Input	None
Test Procedure	Log in as admin. Navigate to house change request page. Click button to turn off house change requests. Change view to that of student. Verify that house change request cannot be submitted. Change role view to admin. Turn on house change requests. Change role to student and verify that requests can be submitted.
Expected Behavior	When HCRs are turned off a message stating that it is not allowed at this time will be shown to students. When turned on, the input fields for submitting a new request will be visible.

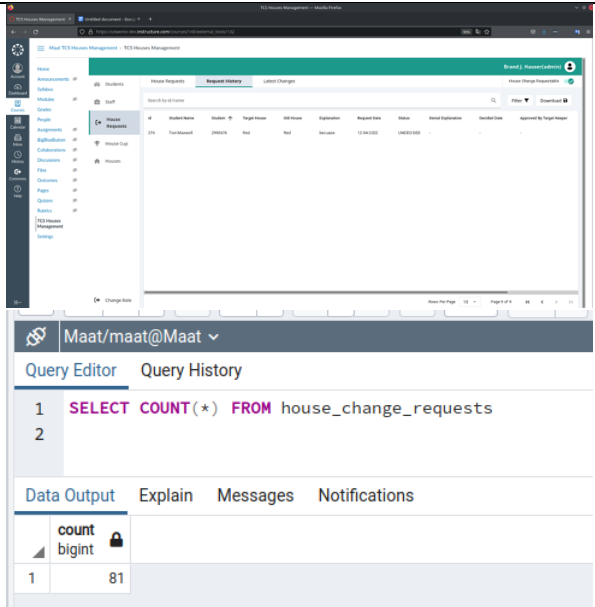
## Actual Behavior

The screenshots illustrate the 'House Requests' interface in a game. The first two screenshots show a list of pending requests with details like requester, item, and status. The third screenshot shows a modal asking 'Which role do you want to view:' with buttons for 'House', 'House Cup', 'House', 'House Cup', 'House', and 'House Cup'. The fourth screenshot shows the 'House Details' page with a 'Leaderboard' section listing the top 3 houses: Yellow House (1183 points), Blue House (1053 points), and Orange House (778 points). The fifth screenshot shows a message: 'The Sorting Hat has disabled house change requests temporarily'. The sixth screenshot shows the same 'Which role do you want to view:' modal as the third screenshot.



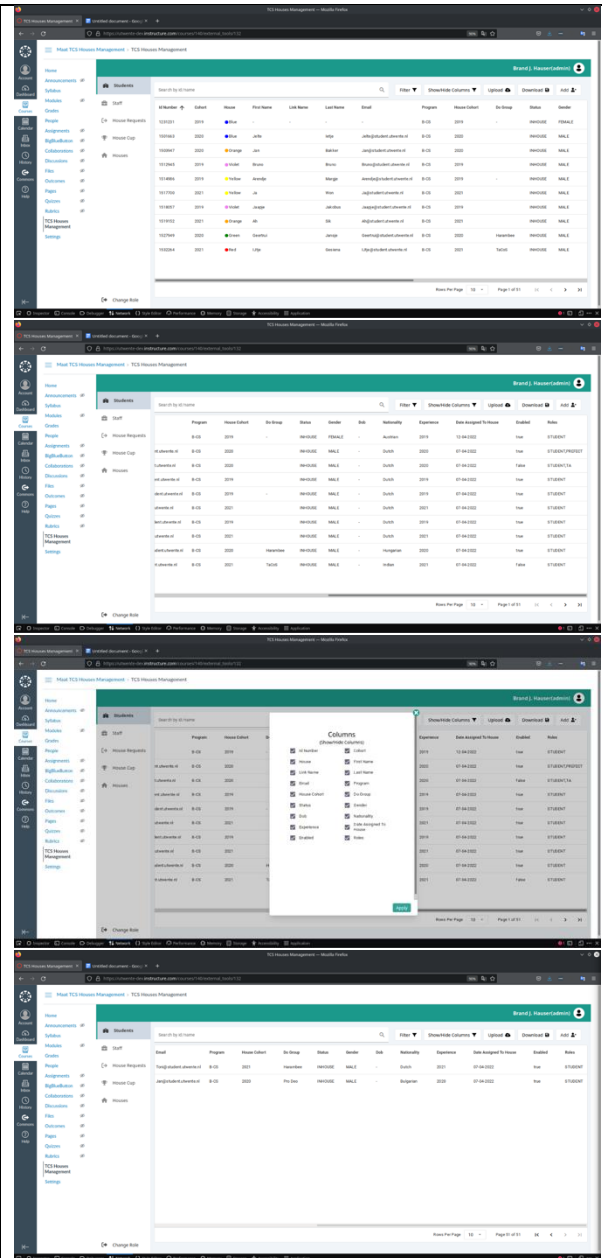
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

User Story	As the sorting hat, I want to see a history of all house change requests
Test Description	Admin users can navigate to the house change request history page on which they will be able to view all house change requests that have been submitted with all of the information including their current status.
Initial Conditions	Logged in as admin. Fictitious student data entered into database including many house change requests at different states.
Test Input	None
Test Procedure	Log in as admin. Navigate to house change request history. Verify that all house change requests in the database are visible.
Expected Behavior	All house change requests will be visible.
Actual Behavior	

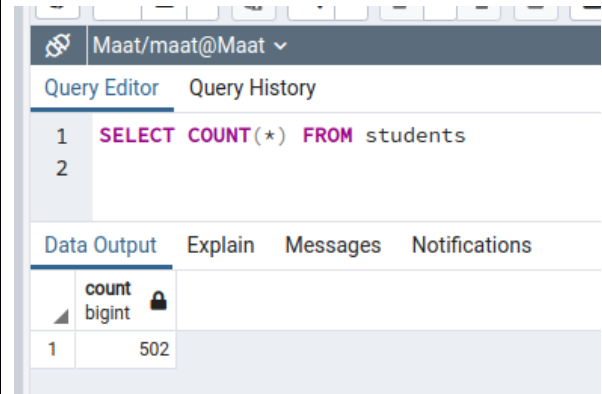
	 <p>8 pages of requests with 10 each plus one additional request = 81</p>
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A

User Story	As the sorting hat, I want to see all information about all students: name, student number, email, gender, house, do-group, date of birth, nationality, prior experience, and the year they started the program
Test Description	Admin user has the ability to view all columns of student data.
Initial Conditions	Logged in as admin user. Fictitious student data entered into database.
Test Input	None
Test Procedure	Log in as admin. Verify that all columns are visible. Verify that all students are visible.
Expected Behavior	All columns will be visible. The number of students viewable on the webpage will be equal to that of the number of students in the database.

## Actual Behavior



50 pages of 10 students each plus 2 additional students = 502



Evaluated Success

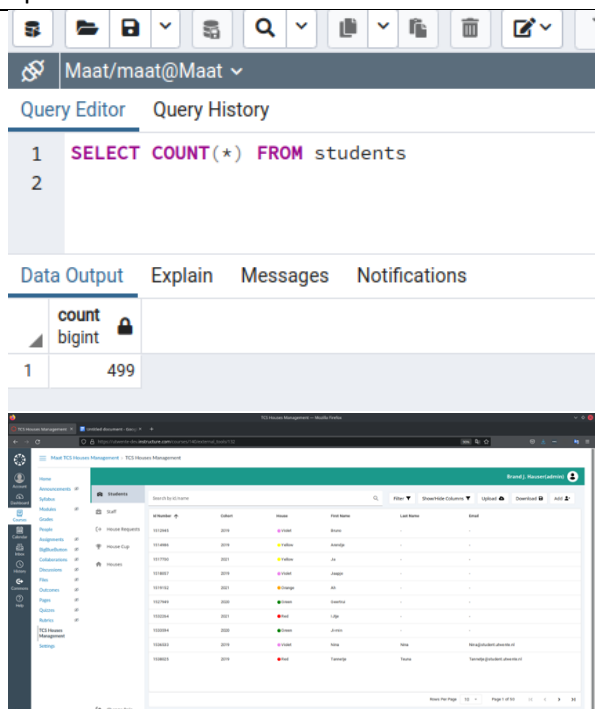
100%

Errors Observed

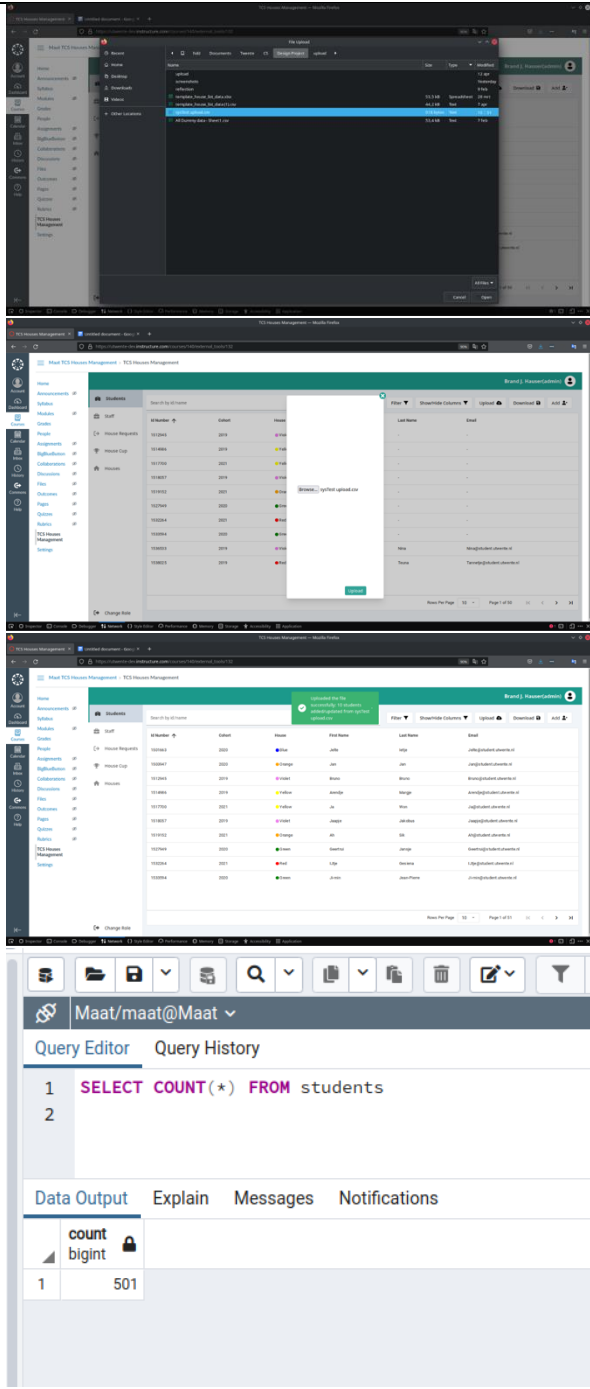
None

Errors Fixed

N/A

User Story	As the sorting hat, I want to update data of students (add csv with student data of students already in system, then it updates those students' data) As the sorting hat, I want to upload data of new students																																																																		
Test Description	Csv files can be uploaded to add new students or to update the data of existing students.																																																																		
Initial Conditions	Logged in as admin. Fictitious data of students entered into database. CSV file of fictitious data of 10 students prepared. First 2 students on csv do not exist in database. Remainder exist but without email and last name.																																																																		
Test Input	CSV file																																																																		
Test Procedure	Log in as admin. Click button to upload file. Select file and click submit button. Verify that 2 students with id numbers 1501663 and 1503947 Have been added. Verify that remaining 8 students have had their information updated.																																																																		
Expected Behavior	2 students will be added. 8 students will be updated.																																																																		
Actual Behavior	 <p>The screenshot displays a web interface for a database management system. At the top, there's a user profile 'Maat/maat@Maat' and tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, showing a SQL query: <code>SELECT COUNT(*) FROM students</code>. Below the query, there's a 'Data Output' section showing a single row with the count '499'. At the bottom, there's a detailed view of the 'students' table with columns: idNumber, cohort, house, first Name, last Name, email. The table contains 10 rows of student data.</p> <table><tr><th>idNumber</th><th>cohort</th><th>house</th><th>first Name</th><th>last Name</th><th>email</th></tr><tr><td>1501663</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr><tr><td>1501664</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr><tr><td>1501665</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr><tr><td>1501666</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr><tr><td>1501667</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr><tr><td>1501668</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr><tr><td>1501669</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr><tr><td>1501670</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr><tr><td>1501671</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr><tr><td>1501672</td><td>2016</td><td>Granger</td><td>James</td><td>-</td><td>-</td></tr></table>	idNumber	cohort	house	first Name	last Name	email	1501663	2016	Granger	James	-	-	1501664	2016	Granger	James	-	-	1501665	2016	Granger	James	-	-	1501666	2016	Granger	James	-	-	1501667	2016	Granger	James	-	-	1501668	2016	Granger	James	-	-	1501669	2016	Granger	James	-	-	1501670	2016	Granger	James	-	-	1501671	2016	Granger	James	-	-	1501672	2016	Granger	James	-	-
idNumber	cohort	house	first Name	last Name	email																																																														
1501663	2016	Granger	James	-	-																																																														
1501664	2016	Granger	James	-	-																																																														
1501665	2016	Granger	James	-	-																																																														
1501666	2016	Granger	James	-	-																																																														
1501667	2016	Granger	James	-	-																																																														
1501668	2016	Granger	James	-	-																																																														
1501669	2016	Granger	James	-	-																																																														
1501670	2016	Granger	James	-	-																																																														
1501671	2016	Granger	James	-	-																																																														
1501672	2016	Granger	James	-	-																																																														



	 <p>The screenshots illustrate the T33 database management interface. The top screenshot shows a file upload dialog with a file named 'students.csv' selected. The middle screenshot shows a table of student records with a modal dialog for editing a record. The bottom screenshot shows the 'Query Editor' with a SQL query 'SELECT COUNT(*) FROM students' and the 'Data Output' tab showing the result 'count bigint 501'.</p>
Evaluated Success	100%
Errors Observed	None
Errors Fixed	N/A