Design Project

Department of Computer Science

Design Report - TA Database

Authors:
Arda Konça
Ervīn Zvirbulis
Marius Pană
Mark Troicins
Vladislav Mukhachev

Client & Project Supervisor:
David Huistra

April 19, 2024

UNIVERSITY
OF TWENTE.

Abstract

The final project report is written as part of the “Design Project” course at the University of Twente. A web application system called the “Teaching Assistant (TA) Database” has been developed for the Module Supports and Module Coordinators to mainly arrange the formalities of determining the module eligibility of TA applicants and creating their job contracts. This report demonstrates the design process of the developed system, the planning of each system development phase, information about the aims and objectives of the proposed system, and how it could be a better tool to use than the current system the Module Supports and Module Coordinators utilize. If the proposed system satisfies all the functional requirements that the client has set, it is planned to be used by the University of Twente in the future for the optimal efficiency and convenience of usability of the designed web application. The layout of this report has used the BOZplanner report on Canvas as a template for reference.

Chapter 1

Introduction

At the University of Twente, there is a need for Teaching Assistants (TAs) to play a role in the educational modules to assist the students' understanding of topics covered within a module. The TAs are not only responsible for assisting students but can also grade the assignments/exams of students depending on their past experiences. To allow the process of involving such TAs within the modules to improve the quality and pace of education, the university gives responsibilities to the Module Support and Module Coordinator officers. One of these responsibilities is determining which applicant(s) could be a good fit to become a TA for a certain module considering their academic qualities.

1.1 Current Methodology of Officers

To determine the qualification, the officers first gather information from the students who volunteer to become a TA by conducting applicant surveys for certain modules. Via the filled-in surveys, they check all the relevant information about an applicant regarding their past academic achievements and whether they need a work permit. However, even though the background information checks are quite straightforward, the process becomes tedious since determining the eligibility of a TA process is manually done and held on Excel sheets at the moment.

1.2 Purpose of the Proposed System

To make the manual check activities faster and more efficient inside a safe and reliable system, the officers have decided to have a web application created that allows them to easily determine the eligibility of TA applicants by multiple users on different machines. During the eligibility assignment process, the proposed system aims to automatically find, retrieve, and save information related to an applicant after being added by a Module Support officer on the page where applicants' eligibility is determined. On this page, the Module Support and Module Coordinator officers could add their comments for an applicant as well as assign them training(s) that an applicant needs to follow. Ultimately, this application aims to resolve the current problem with manual applicant background information checks by utilizing a database for persistent storage. In addition to that, the creation of job contracts for the selected applicants and the hiring process aspect is also determined in the development phases of the proposed web application for the officers to easily input the information of how many total hours a selected TA should work within a module.

1.3 Content Order of the Design Report

To address all the issues regarding the current methodology that the officers follow, more detailed information will be provided in Chapter 2 to address how the proposed system could be beneficial to use. In Chapter 3, the proposal of the system and high-level descriptions will be described. In Chapter 4, the functional and non-functional requirements of the proposed system, the requirement prioritization, and the stakeholder and system requirements will be determined. In Chapter 5, the global design choices and the design considerations will be mentioned. In Chapter 6, detailed architectural design choices such as user interface design and the taken design decisions to enhance the user experience will be elaborated. In Chapter 7, the approaches for testing both frontend and backend components will be considered as well as their test results to understand the overall quality of the functionality of the proposed web application. In Chapter 8, the future planning for the system will be taken into account to possibly further improve the usability and effectiveness of the proposed system. Lastly, in Chapter 9, the evaluation of the overall planning, responsibilities of team members, and the final functional version of the proposed design project will be inspected to derive conclusions.

Chapter 2

Domain Analysis

This chapter will intricately present the system's domain in its current state, through which the highlighted problems might become self-evident. A keen awareness of such matters will ground the project's direction and guardrails for the topical and prospective stages of development.

2.1 Introduction to the Domain

The domain in question involves the frequent distribution of student information between Module Support and Module Coordinators to add, modify, and delete student identifiers, their module eligibility, and potential contract details. Therefore, the system must enable staff of both roles to manipulate their pertinent data.

2.2 General Knowledge of the Domain

Each faculty has a different Module Support staff, so the requirements of a TA database can vary among its users. Grasping the differences and commonalities between desiderata is crucial to effectively deploying the application across the entire university.

2.3 Client, Users, and Interested Parties

The primary stakeholder is Tina Holtkamp-Marti, representative of the Module Support role from the TCS study programme, of the EEMCS faculty, however, the system is expandable to be used by staff members from all the university's faculties. The possible user roles of the system are Module Coordinator (MC) and Module Support (MS).

2.4 Procedure of the Current Situation

Presently, the Module Coordinators and Module Support exchange student data tables over email, which is a privacy hazard. Additionally, we have captured the different activities involved in hiring TAs in Figure 2.1, where it can be seen that the process of deciding a student's total and weekly work hours can be a tedious and highly inefficient process, leading to a loop with an indeterminate end. This happens because a Module Coordinator is the one responsible for assigning their module's future TAs' total work hours and a Module Support is responsible for validating the chosen amount of hours after converting them into weekly hours. Moreover, Module Support must check the validity of all received student data from the Module Coordinator each module, and correct all erroneous information. All of these steps involve a lot of manual labor, leading to much time and effort being spent rather wastefully.

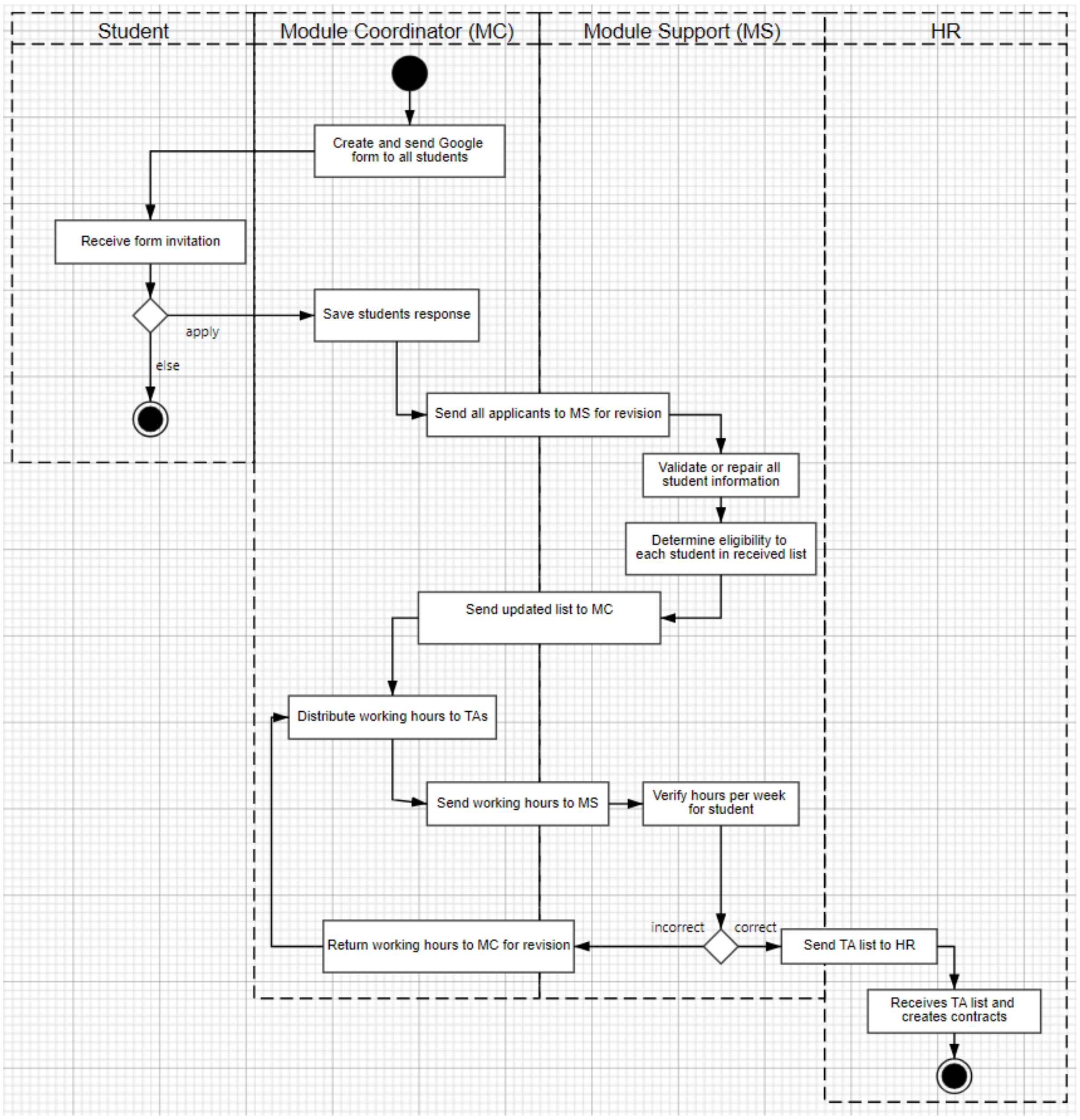

Student
Module Coordinator (MC)
Module Support (MS)
HR
Create and send Google form to all students
Receive form invitation
Save students response
apply
else
Send all applicants to MS for revision
Validate or repair all student information
Determine eligibility to each student in received list
Send updated list to MC
Distribute working hours to TAs
Send working hours to MS
Verify hours per week for student
incorrect
correct
Return working hours to MC for revision
Send TA list to HR
Receives TA list and creates contracts

Figure 2.1: The current activities of all participating agents in hiring students as TAs.

2.5 Commonalities of UT Software

The system was designed to share the aesthetics of the University of Twente's currently deployed application suite to maintain navigational user affinity. Direct inspiration was taken from Horus, Canvas, and Khonsu respectively, by integrating the UT SSO Authentication and replicating common elements between their web page layout designs.

2.6 Software Environment

The project supervisor has some preference specifications, such as building an SQLite or PostgreSQL database, utilizing the Django REST framework under the Python programming language for the back end, a front-end SPA with Vue.js, and using the UT GitLab repository. Some additional options were offered, such as having a Spring Boot back end instead, or using React or Angular for the front end, but our team stuck to the favored development environment to be consistent with the rest of the university's suite of applications, and as all the team members had little to no experience with either tool, relying instead on concurrent on-the-job learning and the supervisor's occasional guidance during development. Additionally, data model development started with SQLite, with the hope to switch to using PostgreSQL near deployment, but after considering there will not be many application users at any one time due to its target audience, the database engine library was maintained.

2.7 Conclusions

In conclusion, the system's domain analysis exposed the application's motivations, founded on the ineffectual procedure of the involved stakeholders, which, in turn, have also been recognized to have their needs catered to.

Chapter 3

System Proposal

In this chapter, the proposal for the system will be provided by including brief information about the requirements set by the client and a user, the proposed mock-ups, an introduction to the system, and lastly the findings of the continuous meetings.

3.1 Requirements Proposal

During the project's development phases, various on-campus meetings were held with the client to ascertain whether the developers firmly grasped the needs and requirements of the client for the proposed system. In the earliest phase, where the concept of the project was introduced, the client held a presentation about how the system is expected to function from the perspective of Module Supports and Module Coordinators by referring to their current issues with the way they use Excel sheets to perform their background information checking task when determining the eligibility of applicants. Although the discussed expectations and requirements were explained to determine the eligibility task and job assignments of selected applicants in a clear way to the developers, there occurred times when developers misunderstood some of the required functional requirements because of the complex relational nature between processes and actions that the users should be able to perform. To fix those misunderstandings, the client and the developers ensured being on the same page by communicating throughout the project's development by following the Agile methodology to avoid any possible delays in delivering a satisfactory functional software product.

At the end of the day, it has been acknowledged that the proposed web application should support the option to choose different academic years, study programmes, and study modules for the application to be used at any time by the officers to capture data about the students. To capture data of the students and register every change, it is decided to develop the concept of "badges" inside the web application. The concept simply refers to the training(s)/specifications (such as passing a certain module, requiring a work permit, etc.) that an officer can assign while determining which applicant could be a TA on the designated TA application page where they determine the eligibility of TA applicants for specific modules. With the support of including such a concept, the officers would easily be able to store relevant information for each of the applicants and they would easily manipulate the records of applicants and/or see/edit which applicant needs to follow which training, which modules they have passed/became a TA in the past. If the mentioned ideas could be done in the proposed web application, then this would allow officers to perform their tasks easily and reliably by being able to assign a dynamic list of student attributes instead of assigning a hardcoded fixed list of student attributes in the system which optimizes the efficiency of the determining eligibility process.

On top of that, additional considerations have also taken place for the job contracts that need to be created for the eligible applicants. The developers suggested that it would be a good idea to have officers have the ability to import the total hours for a TA who could work in a specific module in an Excel format to improve the pace of inputting information in the applicants' job contracts in another dedicated web page. The details of the web pages and their benefits will be given in detail in Chapter 5.

3.2 Mock-ups Proposal

In the meetings, the mentioned requirements of the client were taken into account when trying to create different designs of how the software product should look from the users' perspectives in terms of the user interface of the proposed web application. A team member suggested the front-end developers follow the hand-made templates that he created on the Figma application for the front-end developers to follow what kind of components could be used in the design of the web pages. After that, the team members responsible for working on the front end agreed to follow the constructed designs by making progress every week of the project development phase. To follow and verify the developers were on the right track with how the user interface was starting to look, the mock-ups from Appendix A were presented to the client for feedback on the web pages' design. With the support of showcasing mock-ups, it was made available for the client to observe the thoughts and decisions of the developers to set additional (or remove) requirements for possible system usability improvements.

3.3 System Introduction

In the project's initial phase, the client proposed a face-to-face meeting for the developers to meet with a Module Support officer to get valid insights about the way the application shall function from her perspective.

During the first meeting held with the Module Support officer, she mostly elaborated and focused on mentioning her current problems with using Excel sheets in the process where she fills in the necessary information for each TA applicant (such as which module(s)/course component(s) an applicant has passed in the past, whether they have passed all the study components from their first-year study, whether an applicant needs to follow a teaching minor/facilitator training or whether an applicant needs to have a work permit into a sheet) who would like to become a TA for the certain module(s). She highlighted that filling in information for each applicant takes too much effort and time, and she mentioned that working on Excel sheets leads to organizational efficiency, resilience, and security issues considering the overall student data management. She therefore came up with some ideas of how she would be willing to use an ideal web application to prevent the mentioned issues.

The developers noted down all the mentioned functional requirements within the session to start right away working on the desired web application product. At the end of the meeting, the technical sides of working with the given frameworks were discussed between the developers and client, for the developers to make sure all the aspects that need to be covered within the development of the software product were understood well.

3.4 Results of the Meetings

After the first meeting with Module Support, it was decided between the developers and the client to organize weekly meetings to keep track of the software-wise progress of the developers to possibly focus on/resolve the arisen issues to prevent any delay in producing the proposed product in the given time. It is essential to state that each of the meetings with the client (both online and on-campus) was very beneficial from the perspective of the developers for them to understand they were on the right track. The general conclusion of the regular meetings is the following: Conducting regular meetings critically supported the way the development of the software components be implemented in a fast and organized manner to satisfy the requirement of developing a system that makes the manual background checking on a fitting interface. The client was so supportive, giving ideas of how to solve such problems that the developers were facing from time to time. Additionally, he was so open to being communicated.

On top of the meetings held with the client, and the first meeting held with a Module Support officer where the developers compiled all the requirements related to the software product, the developers wanted to conduct one more meeting with the Module Support officer again on the product to receive her feedback to get an insight of the general satisfaction level about the way how the developed application functions from her perspective at the very end of the time of the project development. In the meeting, the project team first presented a presentation about their progress and then verbally introduced the concepts (badges and module-specific decisions related to eligibility and student jobs) that they additionally applied to the project. After that, one of the team members did a demonstration of the product. At the end of the session, the following feedback was received by the officer and noted down: “The Module Support officers would like to maintain the functionality of Excel, therefore it is a good idea to let the system allow users the ability to control most of the fields (customizability) relevant to TA applicants.” Upon receiving the suggestion, the developers worked on the fields relevant to the data of TA applicants (information about nationality, the year one started their study, start date, end date, and hours per week information of their created student job contracts) to be editable by users to follow the given feedback.

Chapter 4

Requirements

This chapter will present the project’s requirements, guided by the Agile methodology.

4.1 Requirement Specification

This section presents the approach to structuring the system’s requirements that best fit the stakeholders’ demands. The utilization of Agile provided a gradual means in terms of the appropriate way to represent the iterations.

4.1.1 Agile Project Management Approaches for Requirement Specification

The project’s team opted for pursuing Agile management, specifically, weekly SCRUM sessions with the coordinator, because of the wide range of stakeholder needs, a tight deadline, and the team’s unfamiliarity with the development tools, necessitating a continuous reassessment of timeliness and priorities, leading to a frequent reformulation, validation, and effectuation of the requirements.

4.1.2 Requirements Formulation

Expounding upon the second chapter's stakeholder analysis, their requirements take the form of user stories, which have been grouped into four identified feature concepts: user roles, badge system, student management, and student jobs. Moreover, these stakeholder requirements are scrutinized from another perspective: into more practical and detailed system requirements.

4.1.3 Requirements Prioritization

The requirements are sorted by importance, utilizing the MoSCoW prioritization method. They were first identified and later grouped, so not all categories will be present. Furthermore, the MVP would have constituted every feature from Must, and the final product aimed to have all from Should. Additionally, the four feature concepts take no priority over one another, as they are independent, having been worked on in tandem, yet are just as important, being ordered by logical consequence of a user’s application interaction experience.

4.2 Requirement Analysis

The following is a list of stakeholder and system requirements courtesy of the stakeholders' input during meetings. They are illustrated as user stories, divided into functional and non-functional, and by the MoSCoW method.

4.2.1 Stakeholder Requirements

Functional requirements:

I. *User roles:*
   - **A. Must**
     1. As a user, I want to log into the system.
     2. As a Module Coordinator, after logging in, I want to select and see the modules of the study programme that I am responsible for handling TA applicants.
     3. As a Module Support, after logging in, I want to select and see all the modules of the study programmes that I am responsible for checking the requirements of TA applicants.
   - **B. Should**
     1. As a Module Support, a Module Coordinator should not have the possibility to manipulate data outside their coordinated modules.
     2. As a Module Support, I want to have access to other study programmes that I am not responsible for.
   - **C. Could**
     1. As a TA, after logging in, I want to modify my profile information.
     2. As a TA and Module Support, I want to document my experience with the system.

II. *Badge system:*
   - **A. Must**
     1. As a Module Support, I want to see if a TA candidate has completed the module they registered for.
     2. As a Module Support, I want to see if a TA candidate has all first-year credits.
     3. As a Module Support, I want to see if a TA candidate requires a work permit.
     4. As a Module Support, I want to manage what completed training types a TA candidate has.
   - **B. Should**
     1. As a Module Support, I want to upload the overall grade of the module taken by a recurring TA.

III. *Student management:*
   - **A. Must**

1. As a Module Coordinator, I want to import an Excel file of TA candidates.
2. As a Module Support, I want to set the module eligibility of a TA candidate.
3. As a Module Support, I want to export chosen TAs in an Excel document.
4. As a Module Support, I want to see a recurring TA candidate's previous teaching experience.

B. Should

1. As a Module Support, I want to manage the details of a TA candidate, such as their study programme and cohort.
2. As a Module Support, I want to manage eligibility comments attributed to TA candidates of a module.
3. As a Module Support, I want the system to flag certain TA applications in mild or harsh warnings, depending on the eligibility requirement violation severity, rather than having them disregarded.

C. Could

1. As a Module Support, I want students who graduated to automatically be archived.
2. As a Module Coordinator, I want to distribute my module's TAs between different tasks.

IV. Student jobs:

A. Must

1. As a Module Support, I want to view a TA's working hours.
2. As a Module Coordinator, I want to import an Excel file of TA jobs.

B. Should

1. As a Module Coordinator, I want to specify working hours and their period for selected TAs of a module.

C. Could

1. As a Module Support, if I decide to raise an issue with a proposed TA contract, the system could send an email about this rejection to the Module Coordinator.
2. As a Module Coordinator, I want to see each of my module's TAs' logged hours.

Non-functional requirements:

A. Must

1. As a user, I want the system to be available at all times.
2. As a user, I want the web application to be safe and secure.
3. As a user I want the application to have very low page load times.
4. As a user, I want the system's tables to function with incomplete information.

B. Should

1. As a user, I want the visual design of the web application to be easy and pleasant to use.

C. Could

1. As a user, I want the web application to have the feature of dark mode.

4.2.2 System Requirements

Functional requirements:

I. *User roles:*

A. Must

1. The system must enable Module Support or Module Coordinator users to log into the system.

   Description:

   The standard access of The University of Twente's staff to the institution's applications has been through its SAML SSO Authentication, through which user roles can be assigned. For our system to be integrated into the university's suite of applications, a consistent design is required. The functionality implementation was achieved quite early with okta, in the fifth week, and the final modifications were made after the application's deployment, at the end of development.

2. The system must grant Module Support users access to all modules within a study programme.

   Description:

   While Module Support users have access to all study programmes, they generally only work on one, and a single module at a time, but must still always have access to all a programme's modules. Therefore, each Module Support acts as an admin of the system; the default way to view all of the application's functionality, which was used for the majority of its development. This feature was completed in week five.

3. The system must have a mechanism through which a Module Support or Module Coordinator user can select a "Study Programme", "Academic Year", and "Study Unit" to be able to do their pertaining TA application management operations for each module.

   Description:

   Both user roles must have access to modules, where applying students are handled, this being a core feature of the system. When there isn't already a module entry with these three specifications, a new one will have to be created. This feature was completed in week six.

B. Should

4. The system should have a section in which a Module Support or Module Coordinator user can modify a module's properties, such as its name, start, and end date.

   Description:

   Users should also be able to modify various properties of the modules they create; their main workspace within the application. This feature was completed in week eight.

5. The system should have a mechanism which enables a Module Support user to assign the role of Module Support to another staff member.

   Description:

   A Module Support user should be able to share their admin privileges with other UT staff members, generally to manage other study programmes they do not focus on. This would be done by associating the person's UT credentials with our system's Module Support access level. This feature was completed in week eight.

6. The system should have a mechanism which enables a Module Support user to assign the role of Module Coordinator for a specific module to another staff member.

   Description:

   The Module Support is the one that assigns Module Coordinators to selected modules for managing TAs. This would be done by associating the person's UT credentials with our system's Module Coordinator access level. The abilities of a Module Coordinator are identical to that of Module Support, except for having access to a select few of the application's web pages, chosen by our primary Module Support stakeholder, Ms. Holtkamp. This feature was completed in week eight.

C. Could

1. The system could have a mechanism which enables students to add and modify their personal information.

   Description:

   If we were to extend our user roles to support students who want to become a TA, they would want to ease the workload of Module Support and Module Coordinators by directly filling out their student information in their profile, rather than filling out a teacher's form. Their modifications, however, would still have to be checked by one such user. After weighing the benefits and disadvantages, as well as the development costs this feature would bring, given the project's purpose, being to increase the UT staff's work efficiency, we do not consider this feature important enough to be implemented within the initial development cycle. Therefore, this feature was not implemented, as giving students access to the system is out of the project's scope and set time frame.

II. *Badge system:*

A. Must

1. The system must have a mechanism which enables Module Support or Module Coordinator users to assign badges to any of the students who would like to become a TA for a certain module of an academic year.

   Description

   The concept of badges is primarily designed around efficiently defining student study attributes and assigning them to those who possess their characteristics. In practice, all students' badges should be updated yearly by a user with their study progress, except for those with conditional eligibility or newly imported into the system. This feature was completed in week seven.

2. The system must have a mechanism which enables a Module Support or Module Coordinator user to assign badges to a module to define the requirement options of its student eligibility.

   Description

   By assigning relevant study programme badges to each module, users would be able to easily remind themselves what possible attributes of a student should be taken into account to become eligible for that particular module. This feature was completed in week eight.

3. The system must have a mechanism which enables a Module Support or Module Coordinator user to create badges inside a programme.

   Description

   We define badges as persistent attributes that a student either does or does not have at any one time. These could be completed trainings or hireability requirements, such as needing a work permit or having completed enough faculty courses. These badges would be created per study programme, as the requirements could vary between them. We started by making pre-built badges in the system through the Django admin panel from week five to use for more important features until badge creation could be handled in the front end. This feature was completed in week eight.

III. *Student management:*

A. Must

1. The system must have a mechanism which enables a Module Support user to see the list of all students applying as a TA within a programme with their relevant study information inside a table.

    Description

    We completed this goal in week five simply by displaying on the front end the written dummy data from Django's admin panel, though the overall aesthetics of our website were later improved to fit the Figma designs.

2. The system must have a mechanism which enables a Module Support or Module Coordinator user to see the list of students applying as a TA for a module with their relevant study information inside a table.

    Description

    Much like the full list of students, our application must also have a designated area for each module to display a table of students filtered by their solicitation to labor in it. This feature was completed in week five.

3. The system must have an option which enables a Module Support or Module Coordinator user to search for a student by identifiers such as their student number, full name, badges, and eligibility status.

    Definition

    In both the study programme and applied-to module-specific tables of students, one must be able to filter the entries by searching for certain student identifiers. We used the common approach of implementing a search bar to this end. This feature was completed in week seven.

4. The system must have a mechanism which enables a Module Support or Module Coordinator user to perform eligibility checks for the students in a certain module.

    Definition

    For every student in the system, whether they have been imported or manually added, there are some fields attributed to them for the module to which they have been assigned, such as an eligibility status, with the option of conditional eligibility, an eligibility comment, as well as a record of the source of the student entry's automatic eligibility creation. This feature was completed in week eight.

5. The system must have a mechanism which enables a Module Support or Module Coordinator user to select certain students to become a TA for a certain module of an academic year.

    Definition

    From the front end's available tables of students, one must be able to select and deselect an array of students. This has later proved useful in performing en masse operations with the intended people, such as deleting them or exporting their information. This feature was completed in week seven.

6. The system must have a mechanism which enables a Module Support or Module Coordinator user to export a list of selected students with their information such as their student number, full name, study programme, comments regarding their possibility of becoming a TA, and their eligibility status into an Excel spreadsheet.

   Description

   Selected student data must be able to be exported at the press of a button in a table as an Excel file, to be compatible with other university data processing systems. This feature was completed in week eight.

7. The system must have a mechanism which enables a Module Support or Module Coordinator user to import a list of students who would like to become a TA for a certain module of an academic year.

   Description

   That will be the primary method of adding new students to the system, as asked for by the stakeholders. To the alternative of manually creating student entries, importing people will only be possible within a module list, as students only apply to become a TA to one module at a time; therefore, this limitation removes any potential confusion. Additionally, we leverage the university's LDAP system to fill in some pertinent information automatically, solely based on student numbers. This feature was completed in week eight.

B. Should

1. The system should have a mechanism which enables a Module Support or Module Coordinator user to manually create, update, and delete students from a certain module of an academic year.

   Description

   This feature was less prioritized than importing a student list, as the stakeholders currently handle their data in Excel files, a common format for the distribution of tables, and have expressed their desire for efficient spreadsheet transfer into the system. Nonetheless, manual control is persistently vital for the speed and simplicity of entry management. This feature was completed in week nine, except for manually adding new students due to time constraints and because the alternative of importing users had a higher priority.

D. Could

1. The system could have a mechanism to automatically gather all relevant student information from Osiris into the application for Module Support or Module Coordinator users.

Description

With OSIRIS, we can automatically extract more extensive student information than LDAP. However, gaining connective access to the platform is considerably more difficult, so we have left this as a feature we could implement after the application's release.

2. The system could have a mechanism to automatically determine if a student passed the first year, has 60 EC, or has passed a specific study unit for Module Support or Module Coordinator users.

   Description

   By leveraging the breadth of data OSIRIS offers, it would be possible to mechanize the determining process of such queries. Given the requirement for such a feature, it follows that it can only be achievable after the previous user story's implementation.

3. The system could have a mechanism to archive students.

   Description

   Student information should be archived once out of general use, such as for those who graduated or dropped out of the university. This could be executed manually by a Module Support, or automatically with an established connection to OSIRIS.

IV. Student jobs:

A. Must

1. The system must have a mechanism which enables a Module Support or Module Coordinator user to view all student jobs from a selected module.

   Description

   Each module must also have a designated area to display a table of its students' jobs. This feature was completed in week five.

2. The system must have a mechanism which enables a Module Support or Module Coordinator user to import within the system a list of students, their job period, and their total work hours so that a job would be automatically created.

   Description

   Just as with importing students by their student numbers, jobs can be imported through a few details. However, the given details don't have to be correct until a Module Support checks their validity. This feature was completed in week eight.

4. The system must have a mechanism which enables a Module Support or Module Coordinator user to export a list of selected student jobs.

   Description

Student job information must be able to be exported for transferral to one of the university's other applications to be officially formatted and signed as a contract. The date of such exports is recorded to ease contract history administration, profitable in clearing up potential mix-ups. This feature was completed in week eight.

B. Should

1. The system should have a mechanism which enables Module Support or Module Coordinator users to manage a job from a selected module for a student, specifying the contract's start and end dates and their total hours, out of which total weekly hours are inferred.

    Description

    Student jobs must be able to be created, updated, and deleted from the front end within a module. While the weekly hours are derived from a specific formula, we started by dividing the total hours by the number of hours in work days between the selected date interval. Later in development, our team could have received the actual formula to replace the placeholder mechanism. However, this was left for after the project's submission deadline. This feature was otherwise completed in week nine, except for manually adding new jobs due to time constraints and because the alternative of importing jobs had a higher priority.

Non-functional requirements:

A. Must

1. The system must be available at all times.

    Description

    The application should not crash or be out of service. In other words, the system has to be available at all times. Naturally, new features may introduce issues during development, and the system's optimizations continue until its release, so our team must stay wary in maintaining system availability and reliability throughout its course. As far as we have tested, there are no threats to the system such that this requirement isn't fulfilled.

2. The system must have good privacy and security features.

    Description

    The system should implement basic security practices and be deployed as a Docker container on the University of Twente's servers for increased security. This feature was completed in week ten.

3. The system must have the entire front end loaded at once for increased browsing fluidity.

    Description

The system must have the front end loaded at once for increased browsing fluidity, through a SPA. Since such a feature is a part of the initial design, it was present from the early stages of front-end development, even from week five, when the front-end groundwork was set.

4. The system must be able to work with incomplete information.

   Description

   Where possible, table entries should support leaving as many fields empty so a user can save their progress at any time and return later to complete their work. Though this was an ongoing feature to implement throughout the system in the front end, the back end architecturally supported this from week five, when we set the front-end groundwork.

B. Should

1. The system should have good usability, achieved through simplistic visuals and a user-friendly design.

   Description

   Using a similar aesthetic to other systems from the university's suite has offered the application a sense of familiarity and follows the same visual design principles. While the design of the front end was done in week three, the continuous development of the final product has occasionally changed some aspects.

C. Could

1. The system could have the feature of light and dark mode.

   Description

   Different luminosity modes are popular color scheme styles that could increase enjoyment and comfort for many users. However, they had a lower priority than the other vital non-functional requirements, having fallen out of the initial development cycle's scope.

Chapter 5

Global design

This chapter presents and defends the high-level design decisions made throughout development. Additionally, its second half outlines blueprints of the system’s web page content structure.

5.1 Global Design Choices

The purpose of the system is to decrease work redundancy in the TA hiring process by aggregating module-specific applicant data, divided by user access, and storing it for more accessible and efficient management: to Module Support first and second to the Module Coordinator. Chapter 2 has revealed the current processes and procedures, which this chapter expands upon with their updated replacements.

5.1.1 Revised Work Procedure

Chapter 2.5 analyzed the current activities, identifying many logistical issues between Module Coordinators and Module Support. In light of the stakeholders’ proposed project task, depicted in Figure 5.1, the TA hiring process has not reached peak efficiency, not addressing the indefinite loop across the two primary roles using the system, but yet offers great flexibility in task preference of the new approach and cuts down the overall workload of Module Support by minimizing repetitive endeavors. Firstly, while the distinctions between roles remain, each member may choose to assist in some way the opposing group. The system separates Module Coordinators from Module Support based on access to general study program web pages and data, alongside management privileges granted to different modules, but not based on their features. Therefore, users are only limited by factors outside the application, thereby supporting any permission exception. On that note, the stakeholders hold the future dispersal of user responsibilities in "good faith," resorting to a simple authorization retraction to those who misbehave, a fact that lifts this burden off the system and onto the administrative users. Second, since all registered students' data is saved in the new system, Module Support need not check its validity, except for new student entries and occasional updates, the latter of which generally happens once a year. Besides these modifications, the general workflow remains identical, which should ease the system’s integration into the users’ onuses.

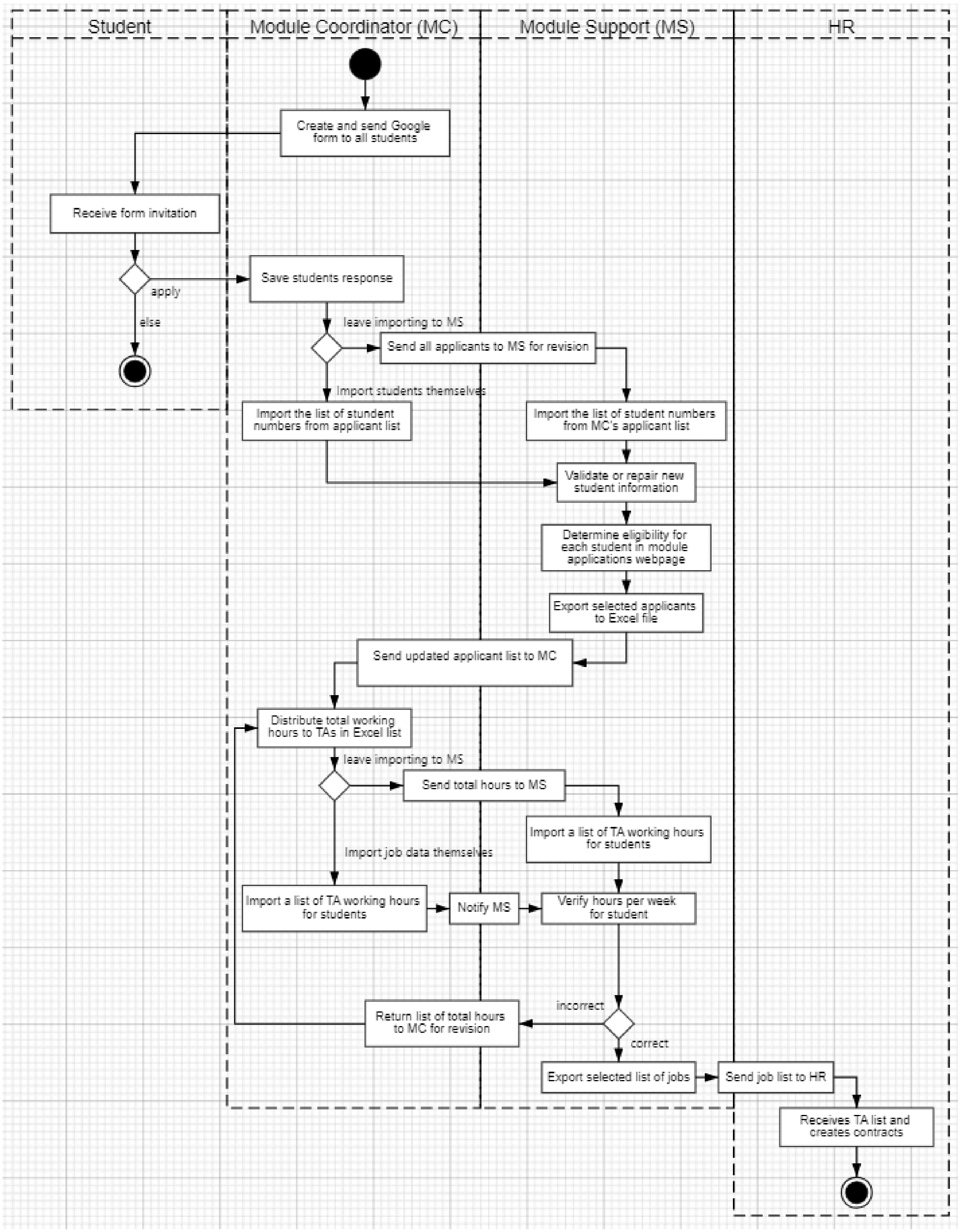

Student
Module Coordinator (MC)
Module Support (MS)
HR
Create and send Google form to all students
Receive form invitation
apply
else
Save students response
leave importing to MS
Send all applicants to MS for revision
Import students themselves
Import the list of stundent numbers from applicant list
Import the list of student numbers from MC's applicant list
Validate or repair new student information
Determine eligibility for each student in module applications webpage
Export selected applicants to Excel file
Send updated applicant list to MC
Distribute total working hours to TAs in Excel list
leave importing to MS
Send total hours to MS
Import a list of TA working hours for students
Import job data themselves
Import a list of TA working hours for students
Notify MS
Verify hours per week for student
incorrect
correct
Return list of total hours to MC for revision
Export selected list of jobs
Send job list to HR
Receives TA list and creates contracts

Figure 5.1: The updated activities of all participating agents in hiring students as TAs.

5.1.2 Architectural Design Choices

This sub-chapter delves into the strategic decisions driving the architectural design of the project. Each choice was carefully considered to align with the overarching goals of promoting modularity, enhancing user experience, ensuring data flexibility, and optimizing the development and deployment processes.

5.1.2.1 Separation of front and back

Since the project was developed by a team, separation of concerns was inevitable. One of the ways the team did this was by separating the front and back ends of the application. This enables modularity, relaxes merge conflicts in the version control, and enhances overall maintainability.

5.1.2.2 Single-page application (SPA)

We decided to adopt a single-page application (SPA) architecture for our solution to streamline user interaction and enhance efficiency. This decision came from several considerations.

First and foremost, SPA architecture offers a more fluid and seamless user experience by removing the need for a page reload during navigation through the application. With our application, the module supports and coordinators can access functionality, such as managing upcoming TAs or adding a new badge, without experiencing interruptions or delays commonly associated with traditional multi-page architectures.

Additionally, SPA architecture leverages client-side rendering and data retrieval techniques, allowing faster and more efficient data processing. By dynamically updating content without refreshing the entire page, our solution offers real-time access to up-to-date data.

5.1.2.3 REST API

For the application to interact with the database, we decided on the REST API interface. A few of the main reasons for such a decision include the interface's simplicity and data flexibility, as well as the team’s and client’s familiarity with the subject.
By adhering to the REST principles, we ensure modularity, maintainability, and alignment with established best practices, which improves the backend’s maintainability.

5.1.2.4 Docker

Since the system is meant as a tool for Module Support and Module Coordinator officers, deploying it on the university’s infrastructure is an obvious choice, both for security and convenience. Docker helps to ensure the integrity of the project, as well as test deployment stages locally, which increases development efficiency.

5.1.3 Technological Stack

In this part of the report, we will discuss what frameworks and libraries we are using and why we chose them. As our application is split into frontend and backend, we will discuss those parts separately here as well.

5.1.3.1 Frontend:

The first, and most important framework that we will be using is **Vue**, which is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative, component-based programming model that helps you efficiently develop user interfaces of any complexity. Its component-based architecture promotes reusability, making development faster and more efficient. This is very useful and helpful as we are building a single-page application, all the mentioned features just make the development process easier.

Another important framework that we used is the **Vue Component Framework (Vuetify)**. It is directly connected with the Vue framework and it allows us to create small modules to be used and re-used throughout an application. Vuetify is a collection of pre-made components paired with powerful features such as dynamic themes, global defaults, application layouts, and more. This framework is very useful when it comes to adding and adjusting any user interface components.

Second, we decided to use **Axios** which is a promise-based HTTP Client for node.js and the browser. We use it to send different HTTP requests to the backend.

Third, we used **Xlsx** which is an open-source solution for extracting useful data from almost any complex spreadsheet and generating new spreadsheets that will work with legacy and modern software alike. It is extremely useful as one of the requirements is to import and export Excel Sheets in and from the system.

Fourth, we used **Selenium** which is an open-source umbrella project for a range of tools and libraries aimed at supporting browser automation. It provides a playback tool for authoring functional tests across most modern web browsers, without the need to learn a test scripting language. We used it to test the front end properly.

5.1.3.2 Backend:

The backbone of the system is built using the **Django REST** framework. There are multiple reasons why this was the choice. First, Django has an extremely convenient admin panel for the database. Second, the data model is configured via Python objects, which eliminates the need for a separate database manager and allows one to choose an appropriate database type for the purpose. In our case, since the database is fairly small, we settled on the SQLite3 format. It is the default Django option and the project does not need anything more complex.

University tools must be accessible with the university's credentials. Microsoft is the credential provider of choice, it uses the SAML 2.0 standard. So, we have implemented login functionality using **python3-saml-django.** Besides complying with this requirement, SAML 2.0 facilitates single sign-on, which means that users can seamlessly use the application alongside other university tools, without having to log in to each service.

In conjunction with the SAML library, **OKTA** was chosen as an identity provider. University gives access to authentication systems only for mature apps for security and reliability reasons. Additionally, having the production identity provider redirect to localhost, which is needed while the system is in development, is not possible for the aforementioned reasons. To solve this we set up OKTA to mimic Microsoft's implementation as much as we can. This allows for an easy swap of the identity provider by simply changing the link in the Django settings file.

Students are usually identified simply by their assigned university id, called "ut id". Because not every student will be in the system it is nice to reduce the amount of needed work from the module support by populating known data, namely name, surname, and email address. For this purpose the university's lightweight directory access protocol (LDAP) system was integrated, using the **ldap3** library. Making the system require less manual labor.

Documenting API is extremely important. It helps both current and future developers to keep track of which features are done and how the algorithms work internally. For this purpose, we decided to use the **django-rest-swagger** library and its dependency **drf-yasg**. Swagger automatically creates an overview of all calls present in the system, which simplifies the development process. The developers can easily identify unnecessary calls, as well as notice missing calls from the list.
Swagger has the additional benefit of being interactive. Not only does it state existing calls, but also allows the developers to easily execute and test the call response for correctness. This is useful for validating the appropriateness of the newly created code and manual testing of the calls.

Visualizing the data model is both important for overall understanding of the system, and a manually intensive task. So, to streamline this aspect we used **django-extensions** and **graphviz** libraries. It allows us to create a class diagram of our data model using one command. We made use of the diagrams during development, for intermediate feedback, and in the report, to show the final version.

5.1.3.3 Deployment

Django has a built-in server, which facilitates development steps, however, the server provides the bare minimum for its use case. It is single-threaded, not scalable, and lacks some necessary security features, such as SSL termination, rate limiting, and access control. To address these issues, the following libraries were chosen.

Gunicorn takes the responsibility of managing user requests and executing Python code. Which in turn allows for many concurrent users without the performance hits. Additionally, Gunicorn enables easy horizontal scaling if that becomes necessary for the application.

The use of Gunicorn, unfortunately, doesn't resolve all issues of the Django deployment server. So, an additional library, namely **NginX** was introduced. NginX acts as a load balancer, SSL decryption point, and server of static files. This reduces the load on other parts of the system and blocks bad requests before they can reach the processing point.

Docker allows us to encapsulate all necessary components into a self-contained, easily deployable package. By containerizing our Django application alongside Gunicorn and NginX, we create a cohesive environment that promotes consistency and reliability in our deployment process. Docker keeps our application separate from the underlying system, reducing compatibility issues and making it easily fit within the university's infrastructure. It simplifies managing dependencies and speeds up deployment. With Docker, our application follows the university's standards and ensures better performance with optimal use of resources.

5.2 System Pages

As this application will serve as a replacement for Microsoft Excel, we had to carefully analyze the current workflow and come up with the design that is going to optimize the process. Initially, all data was presented in one Excel sheet, which is very inconvenient and cumbersome. We decided to split all the data into 3 logical parts (pages): "Module management", "Student list" and "General data". Before we get into details it is important to mention that all the pages are program-specific. It means that before the user can see any of the pages they have to choose a program, on the top right on Fig. 5.2.1, they are working in, so they don't see irrelevant information. Images for each page are included in Appendix B.

Now let us break down the purpose of each page:

5.2.1 Module Management Page

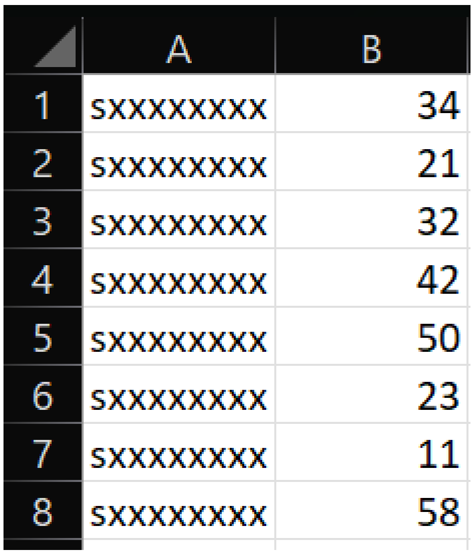
The "Module management" page is what the user interacts with when they need to process TAs for the module the user is responsible for. When the user needs to add TAs that have applied for a module, they don't want to see any information related to other modules. Due to that, the page serves the purpose of working with a specific module.
When the user first enters this page before they can see any data they have to choose a year they are working in and a specific module. Year selection is crucial as modules repeat every year and we need a specific one to display the data. After the user has chosen a year and module they have 3 "tabs" to choose from: "Applications", "Student Jobs" and "Module Settings". Each serves a unique purpose in the process of TA enrollment.

5.2.1.1 Applications Tab

The first tab - "Applications", shows the list of applicants for the selected module and contains all the data required to determine if the applicant is eligible or not. Applicants are the students who have applied to become a TA for the module the user is responsible for. By interacting with this page the user can import the list of student numbers of applicants (in an Excel format - one column of selected s-numbers of students) and then for each student they can assign badges and determine whether a student who applied to be a TA, is eligible or not. As the user has finished the process they can select applicant records in the table and export them as an Excel file. All the info that the user has filled in for the students is saved and will be shown on this page by selecting the same specific module. Furthermore, in our application, any modification to the data is saved and organized. Therefore, if a change was made to a student in one module, it would reflect in all the other modules this student is present in. Unlike Excel sheets, if a user has multiple tables, they wouldn't be able to preserve the student information between them.

5.2.1.2 Student Jobs Tab

The second tab - "Student Jobs", shows all the students who were chosen by a module coordinator to be a TA for a selected module. It contains all the information relevant to the work contract such as start date, end date, working hours per week, and total hours which are made editable for the officers to be able to update information on job contracts. Similar to the "Applications" page the user can import the list of students here. But on top of just importing a list of student numbers, for this tab, they also need to determine students' total hours and include it in the imported Excel file (the required Excel format for creating jobs for corresponding students/applicants could be examined in the figure below).



	A	B
1	sxxxxxxxx	34
2	sxxxxxxxx	21
3	sxxxxxxxx	32
4	sxxxxxxxx	42
5	sxxxxxxxx	50
6	sxxxxxxxx	23
7	sxxxxxxxx	11
8	sxxxxxxxx	58

Figure 5.2: The required Excel format for creating student jobs indicating which students need to work for how many total hours in a selected module.

Our reasoning for the creation of this page comes from the fact that all this information is irrelevant to the user while determining eligibility.

5.2.1.3 Module Settings Tab

The third and last tab on the "Module Management" page is "Module Settings". This tab allows the user to modify any data that the representation of the selected module is built from such as badges, start and end date, and name. It also has a button to delete the module. Furthermore, as our application allows module coordinators to use the system to work on the module they are responsible for, this page has a table of people who have access to the module. To prevent the module coordinator from being able to give access himself, this table is not editable, it is possible to delete records from it but not create new ones.

5.2.2 Student List Page

The "Student list" page is a place where the user can overview all the students in the system of the selected program. That includes students who have been imported into "Applications" or "Student Jobs". The view of the page is taken by a single large table with student records. By interacting with the search bar of the table, users can easily check whether the student is present in the system. If the record is found, by analyzing its data the user can see how this student was added to the system and, therefore find him on a "Module Management" page.

Moreover, it is the main place where the user can modify any information regarding the student.
Unlike “Module Management”, “Student List” is not module-specific. We use this distinction to show all the non-module-specific information about students like completed study units. We are planning on including information about which modules the student has applied for and in which modules they worked as a TA, however, the implementation will be done after the report submission, so it is not present in the images in Appendix B.

5.2.3 General Data Page

The “General data” page is the central hub to observe and edit concepts used in the application. To not overcrowd the view of the page we have decided to split it into “tabs” similar to how “Module Management” is done. It contains 3 tabs: “Badges”, “Study units” and “Accesses”.
Each of these tabs serves a similar function and possesses identical capabilities, yet operates for distinct concepts. They consist of a table to overview the present data, a search bar to easily find specific records, create a new record button, and delete selected ones. Furthermore, every attribute in each table is modifiable for a pleasant experience of editing the data.

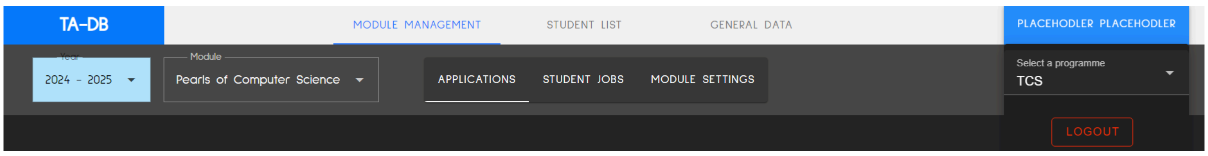


Figure 5.2.1: Program selection and year, module, “tab” selection on the “Module Management” page.

Chapter 6

Detailed Design

This chapter discusses all the low-level and important design choices in detail. The chapter contains a detailed description of the data model, the structure of API calls, and the details of both the user interface and the user experience parts of the front end. The chapter should give the reader an understanding of how the system works from a technical perspective. Additionally, it should give valuable insights into the visual aspect of the application and explain the selection of specific interactive elements.

6.1 System Description

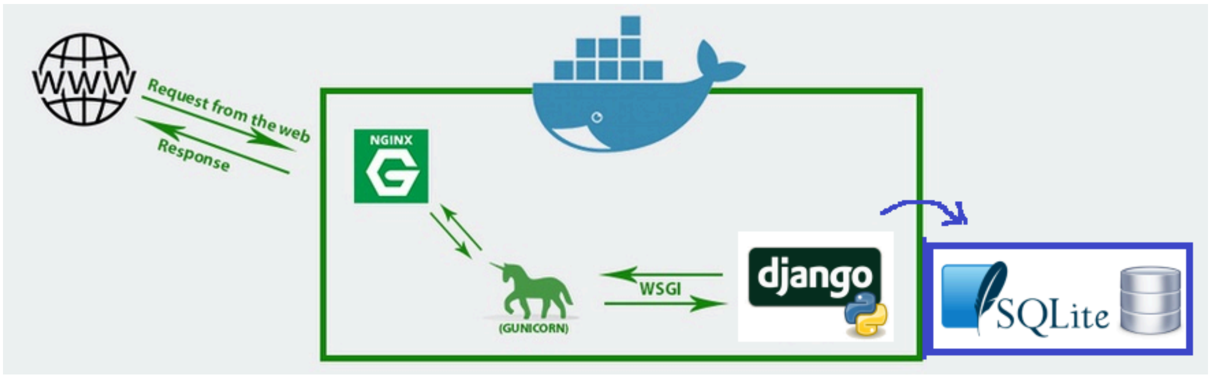


Figure 6.1: system overview

Figure 6.1 illustrates a complete overview of the system. Everything except the database is encapsulated into one Docker container, while the database resides in a persistent volume in order to keep information after application reloads and redeployments.

The first entry point is NginX which decrypts incoming requests, distributes work for the workers in the next step, and forwards the HTTP request to the Gunicorn. After the request is validated it is assigned to a Gunicorn worker which then executes necessary Django functions via Web Server Gateway Interface or WSGI. The resulting response is propagated back through the chain in the reverse order, ending with returning the HTTP response back to the awaiting user.

The simplicity of such a setup increases maintainability and acts as a solid core for future features and uses.

To see more information about the deployed application and necessary credentials please refer to chapter 9.4.

6.2 Design Choices

6.2.1 Data model

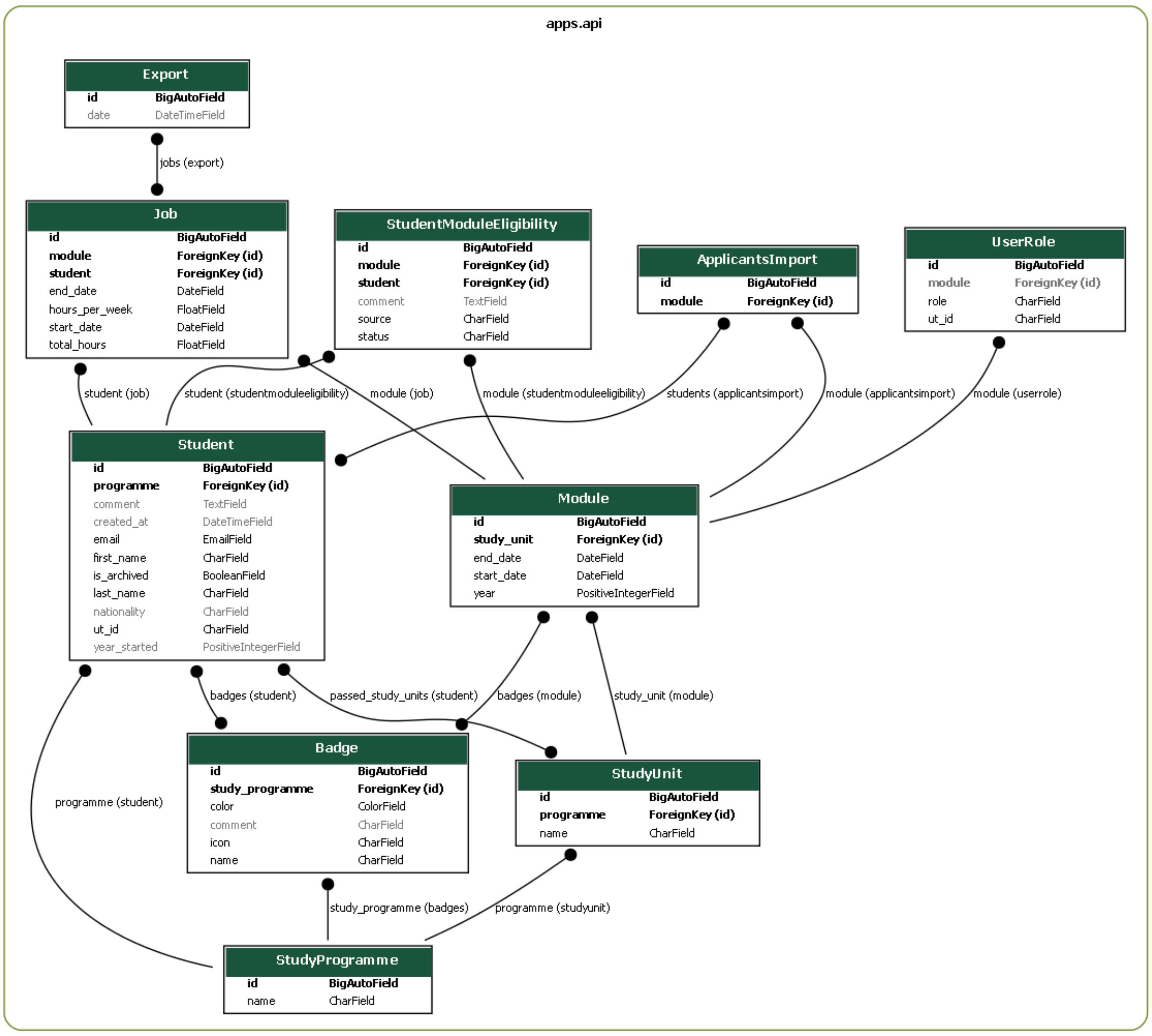


Figure 6.2: The full graph with all class models.

Figure 6.2 encompasses the attributes of and connections between all data model classes. We will cover each class to describe its purpose and use cases in detail.

6.2.1.1 Study Programme

The “StudyProgramme” class records the names of all study programmes in the system. Consequently, due to the EEMCS faculty having requested the project, probable examples of such names would be “TCS” and “BIT.”

6.2.1.2 Badge

The “Badge” class, or, more specifically, the concept of badges, is defined as optional but permanent student attributes that can be dynamically created, updated, or deleted directly in the front end. The design arose from the desire to support the concepts of having completed specific training, passed the first academic year, gained a particular number of ECs, and required a work permit; notions stored as name strings. While the concept of having passed certain study units also fits in this permanence category and is represented in the front end this way, study units are distinguished into a separate class in the back end to support additional functionality.

Moreover, while a Module Support acts as an admin to the system, having access to all programmes, they generally only govern the administration of one, relinquishing the responsibility of others to different users with the same role. As a result, different users may choose to define their badges distinctively, and separate programmes may have dissimilar requirement sets. For these reasons, the decision was for badges to depend on a study programme.

Furthermore, badges may be customized by color, icon, specified by name from a list, and comment, for faster recognition and differentiation.

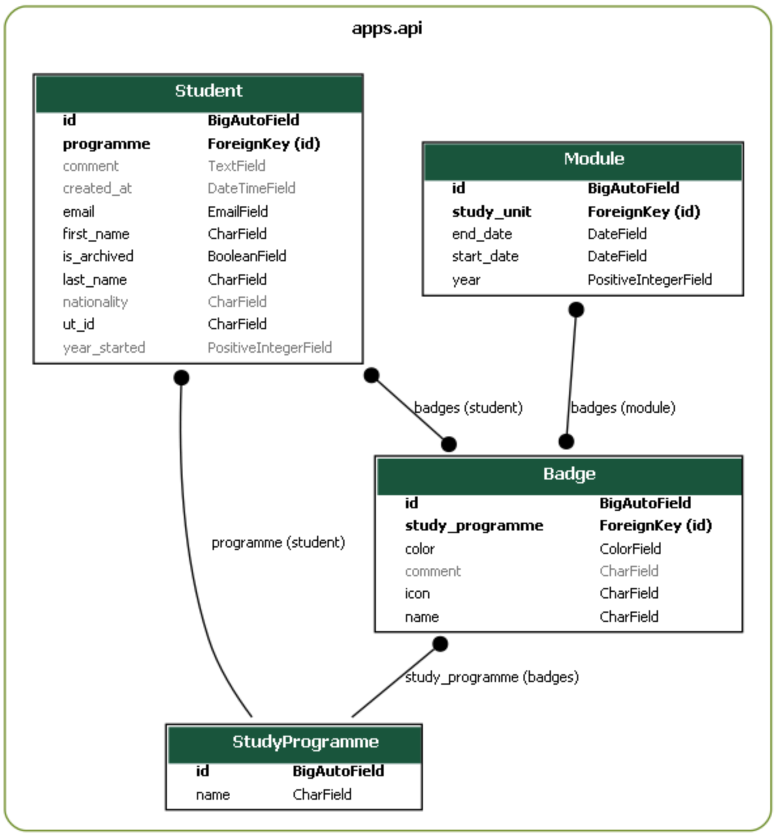


Figure 6.2.1: Partial class diagram including badge and all directly connected models.

6.2.1.3 Study Unit

The "StudyUnit" class represents year-independent modules, defined by name and their study programme.

6.2.1.4 Module

The "Module" class is a year-dependent version of the study unit. Each entry inherits a study unit, meaning a name and a study programme, as its base, with further details such as year, start and end dates, and a list of badges. These badges act as a relevance filter in the front end for student attributes to each module. They limit the available badges for viewing and assignment to TA applicants.

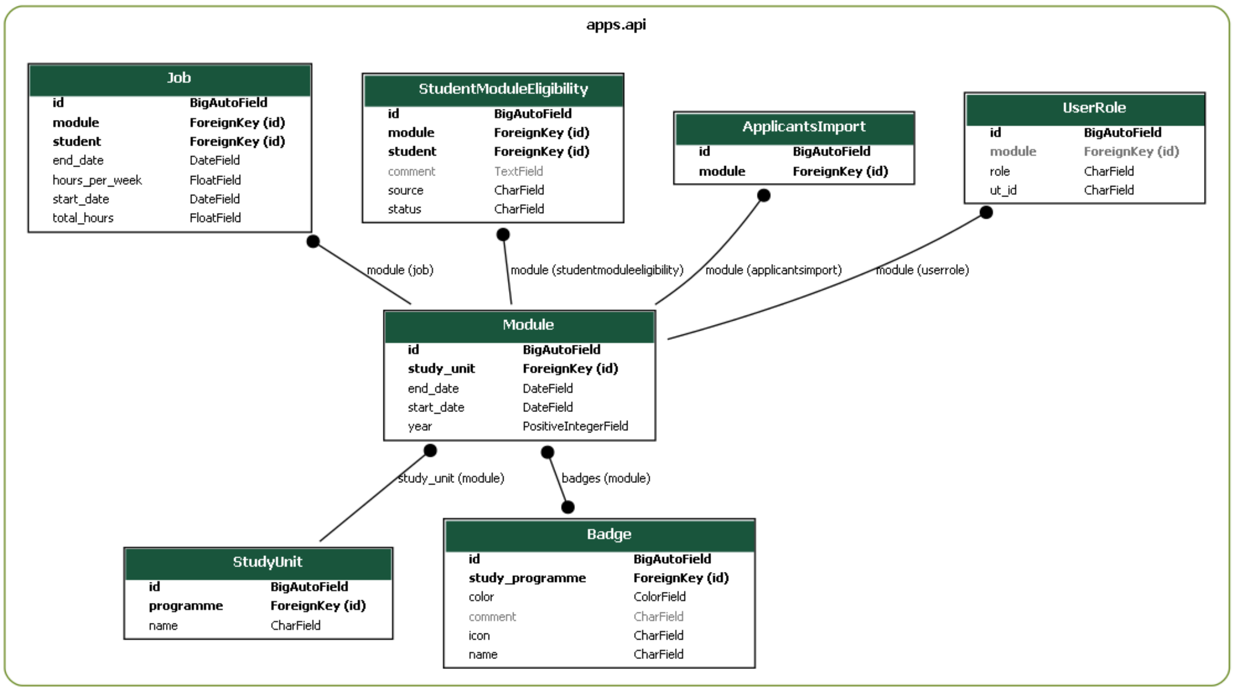


Figure 6.2.2: Partial class diagram including Module and all directly connected models.

6.2.1.5 User Role

The "UserRole" class enables the bifurcation of role permissions between Module Coordinator and Module Support officers. Each UT staff user has their university ID recorded for identification purposes in role distribution. Currently, the system supports the roles of "Module Coordinator" and "Module Support", the former of which has a field for module access assignment, whereas this will always be empty for the latter, as they have admin access throughout the system. These two entry requirements are ascertained in the front end, while the back end responds with an appropriate object based on a user and their permissions. Moreover, a Module Coordinator has a different entry for each module access, rather than a single entry with a list of modules. This design principle was chosen for future-proofing, where new roles added may introduce modular permissions. While

imperfect, it is enough with the current implementation's scope and a step in the right direction.
The decision to use string fields instead of foreign keys to Django's User model for the ut_id was to make the task of assigning coordinators more streamlined. Using our solution, the support can give permissions to the person by knowing their ID without the need to log in (or create a user via the admin panel) beforehand.

6.2.1.6 Student

The "Student" class captures a lot of sensitive student information. After much deliberation with the Module Support stakeholder, we have agreed that all mandatory and static student attributes, as opposed to the concept of badges, that are recorded in the system, comply with GDPR regulations. This fact is achieved by only handling and storing necessary data, which is the case. Regarding good data retention, however, while students can be marked as archived in the system, a proper storing mechanism still needs to be implemented.

Back to student information, the default stored fields are the student ID, first and last name, email, nationality, studied study programme, and first study year. Additionally, UT staff users can add comments directly to a student, pertaining by design to relevant supplementary remarks to their study programme hire ability, not their eligibility to any specific module.

Lastly, there is also a distinction between "passed study units" and "badges," two fields also present in the "Student" class. In the front end, within a module management page, a user is limited by viewing and assigning only the presently selected study unit, to avoid clutter, as the other dozens of modules are extraneous to the deciding TA hiring factors for that module. The needed distinction comes from the necessity to separate between normal and "study unit badges;" while avoiding redundant data duplication when defining modules. Despite presenting each study unit as a badge, users cannot modify or delete them as other badges, and they are formed automatically in the front end based on back-end information. Also, it is worth noting that a student passes study units and not modules, as the courses completed are what matters, not their completion date or age.

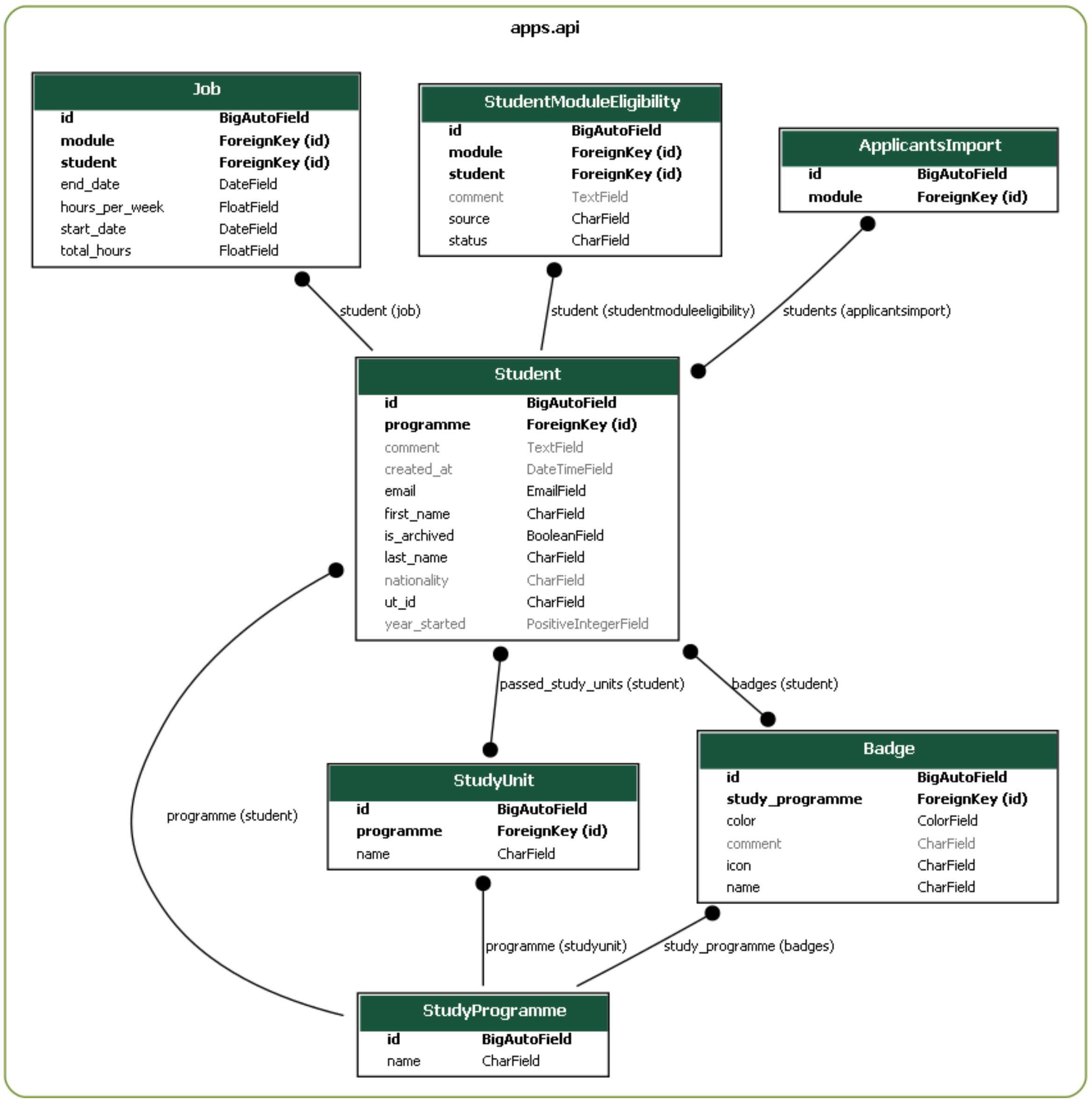


Figure 6.2.3: Partial class diagram including Student and all directly connected models.

6.2.1.7 Applicants Import

The “ApplicantsImport” class pairs student TA applicants to a module. The basis for one of their classes not including a direct reference to the other class’s entries is that each such class defines a self-contained concept, and this approach improves the system’s readability and modularity.

6.2.1.8 Student Module Eligibility

The “StudentModuleEligibility” class pairs a student and a module to assign some eligibility properties to an applicant for a course. Such an individual can receive either of the following four eligibility status types, specifically: "UNKNOWN" for newly entered participants in the system, whom a Module Support officer has not yet checked, and "ELIGIBLE" for passing all the module's hiring requirements, which does not imply being hired yet, "CONDITIONALLY" for a student who does not pass the hiring requirements but is still deemed a potential candidate, and "INELIGIBLE" for those with a declined TA application. Moreover, the source of said status can either be "APPLICANTS" for new students imported into a module as applicants, "JOB" for new students imported alongside their job details, or "OTHER" for students that would be manually added in the system or as a base case.

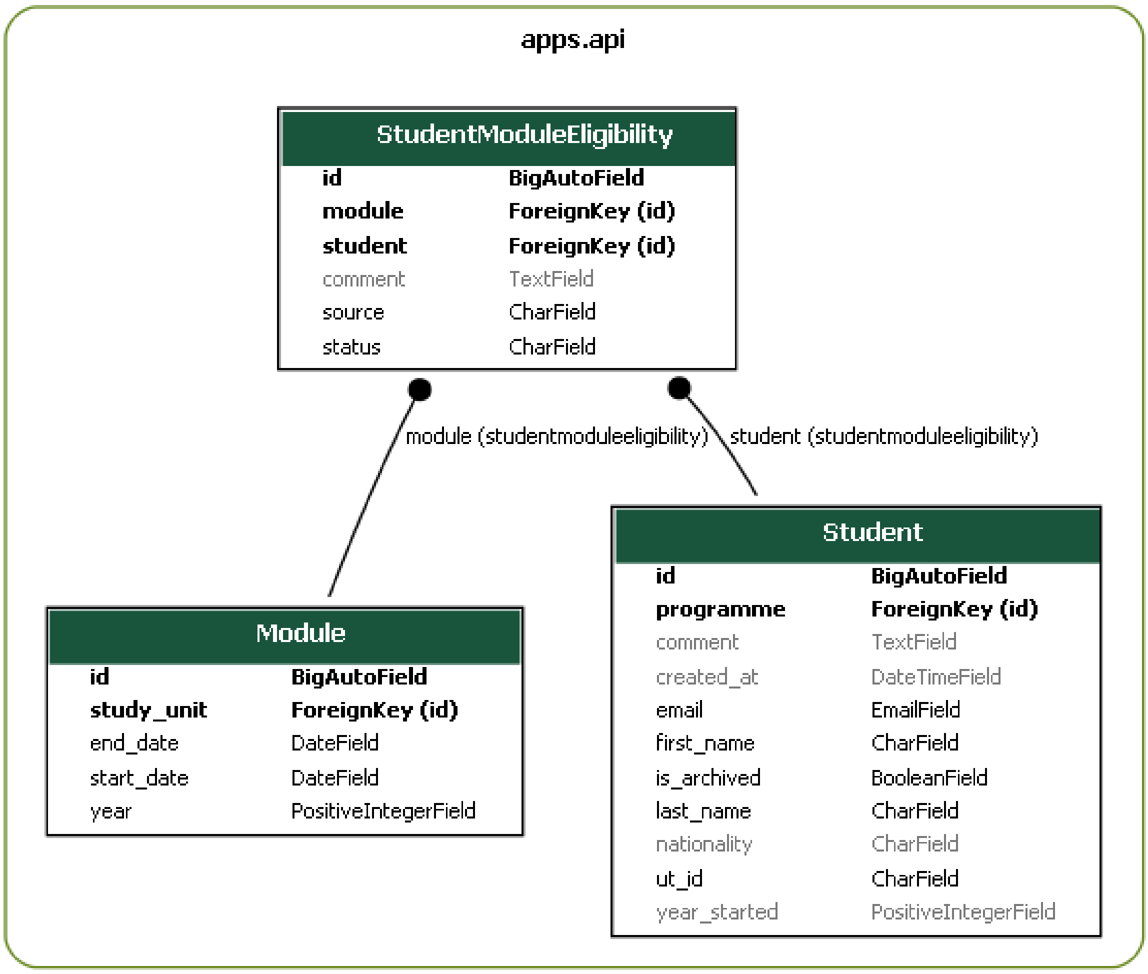


Figure 6.2.4: Partial class diagram including Student, Module and StudentModuleEligibility models.

6.2.1.9 Job

The “Job” class contains a student, a module, a start and end date, a field for total hours, and one for hours per week, where "hours per week" is initially derived from "total hours" for every job entry imported into the system.

6.2.1.10 Export

The “Export” class records the date and time of each group of jobs exported into Excel files from the system.

6.2.2 API call structure

Every single model has CRUD operations implemented, many with additional parameters to facilitate additional features. When it came to realizing the need for filtered outputs and additional information in the response depending on the circumstance we heavily leaned towards adding in query parameters. Depending on the value and if the parameter is present, adjusting behavior and response accordingly. An alternative was to create new requests for each additional response option, but we decided against that since in our opinion keeping the request count low simplifies the system, and keeping all logically connected calls under a single request with parameters was the more compelling option.

Moreover, responses also rely on the authenticated user. If the current user is module support, they will have access to all information stored in the database. On the contrary, if the user is a coordinator, they will be able to see and manipulate only information connected to their module.

To see all calls and test them go to "**server ip**/docs" to see the Swagger page. Examples are also provided in the Appendix C.

6.2.3 User Interface (Connect with figures)

The initial design was one of the starting points of our application. By working on it during the requirement collection and project proposal phases, we were able to determine what functionality and data we might need in the future. Working on the prototype made us question each possible button and table, which at the end of work on the prototype, resulted in a satisfying result.

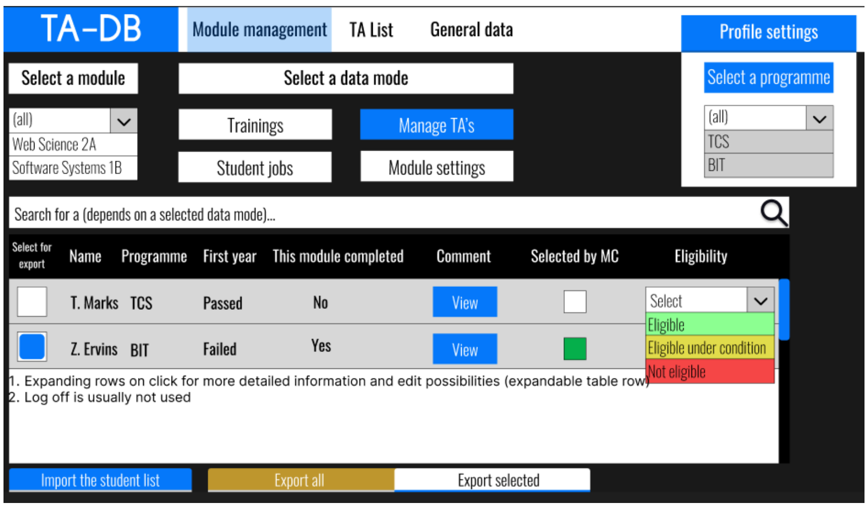
The first page that one will see is the login page. After succeeding with the login, the user will be able to see the application itself. It contains 3 main navigation tabs, which are "Module Management", "Student List" and "General Data". The prototype for the "Module Management" page can be seen in Figure 6.3. In this part we will not discuss the meaning and the functionality of them and other solutions, we will do that in the "User Experience" section, located below. For the "Module Management" and "General data" pages there are multiple sections of data we need to display. To not overcrowd the look of the page and to represent the distinction more clearly these pages have "tabs", pages inside pages. This way only one table can be displayed at a time and each of the tables has its data inside of it.

The selection of color schemes was straightforward. We decided to pick the same blue color as the one used in "horus.apps.utwente.nl" as an additional color and selected a commonly used dark gray hue for the background, which is prevalent in modern applications. Figure 6.4 shows the selected colors and their codes.

For the font, we decided to pick one of the "Google Fonts" available fonts designed by Dale Sattler. The font is called Sulphur Point, it is a geometric sans serif typeface, with low contrast stems, high x-height, restrained ascenders and descenders, and minimal optical adjustments away from pure geometric form. This font is minimalistic and rounded, which helps it exist in harmony with our selected color palette and design. Figure 6.5 illustrates the font in different weights.

There were quite a few interactive elements that we were planning to use during the prototype phase, such as drop-down buttons, buttons, and checkboxes inside the table rows and a search bar for the table. While we kept the placements from the prototype, for the final look of elements we had some concerns. Luckily we came around the idea of using a component library - Vuetify, which has all of the necessary components in the same style. This way we were able to keep stability in the design throughout the development, which benefits a lot to the overall look of the application. Furthermore, due to the appearance of the components we have decided to change our approach for the application design and have preferred to adhere to mainly dark colors for the components.

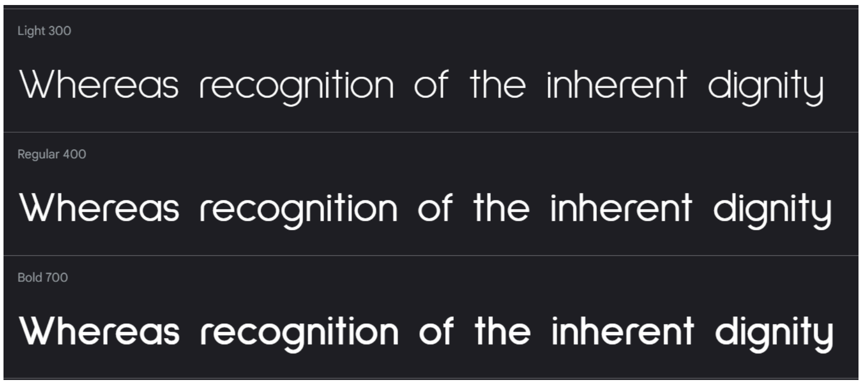
Our design has slightly changed towards the completion of the system. This happened because of the introduction of new functionality and the nuances of the Vuetify component library. For example, the mock-up of the “Module Management” page looked like Figure 6.3, while at the end it looked like Figure 6.6. The color scheme, font selection, and most of the interactive elements stayed the same, but they became more minimalistic, clean, better aligned, and peculiar.



1. Expanding rows on click for more detailed information and edit possibilities (expandable table row)
2. Log off is usually not used

Figure 6.3: Prototype of the “Module Management” page

Figure 6.4: Dark gray color with hex code “#252526” used as the main color, and the blue color with hex code “#087CFC” used as an additional color in the application


Light 300

Whereas recognition of the inherent dignity

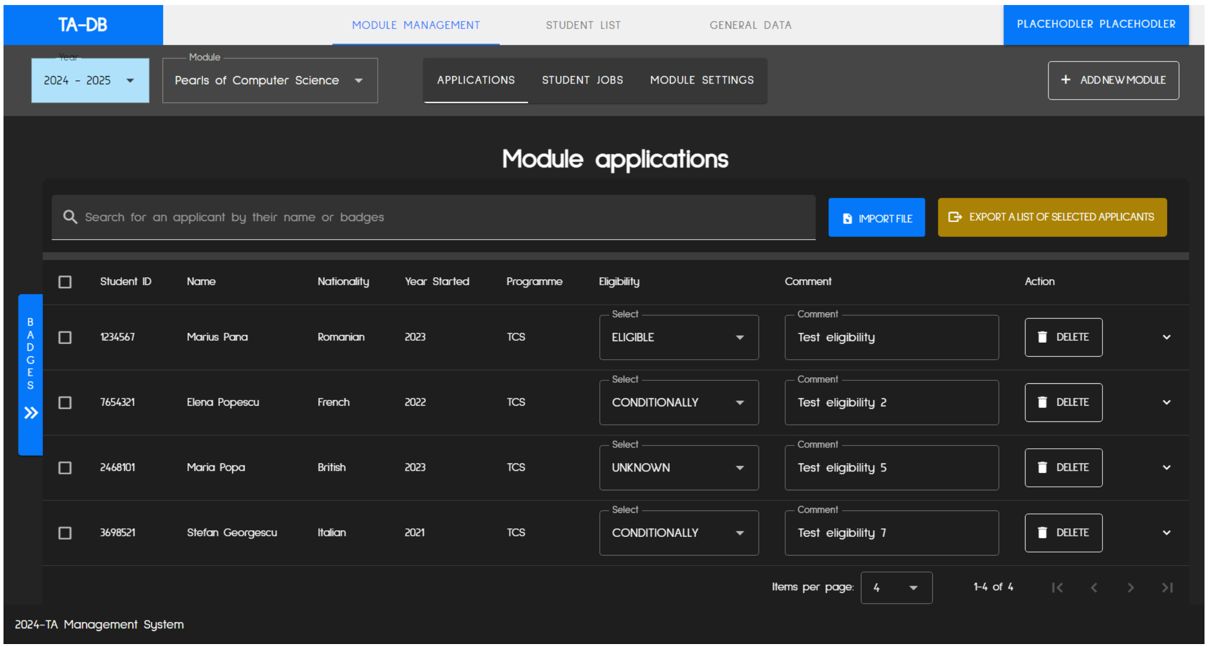
Regular 400

Whereas recognition of the inherent dignity

Bold 700

Whereas recognition of the inherent dignity

Figure 6.5: "Sulphur Point" font example phrase in different weights



Module applications

Search for an applicant by their name or badges

Student ID	Name	Nationality	Year Started	Programme	Eligibility	Comment	Action
1234567	Marius Pana	Romanian	2023	TCS	ELIGIBLE	Test eligibility	DELETE
7654321	Elena Popescu	French	2022	TCS	CONDITIONALLY	Test eligibility 2	DELETE
2468101	Maria Popa	British	2023	TCS	UNKNOWN	Test eligibility 5	DELETE
3698521	Stefan Georgescu	Italian	2021	TCS	CONDITIONALLY	Test eligibility 7	DELETE

Items per page: 4   1-4 of 4

2024-TA Management System

Figure 6.6: Final version of the "Module Management" page

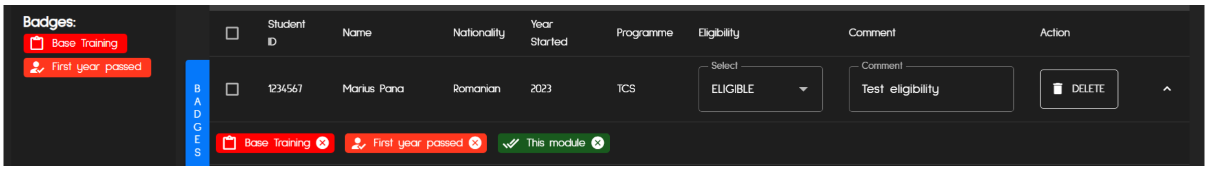
6.2.4 User Experience

Our primary goal for enhancing user experience was to offer a more enjoyable interaction with data compared to the current system, which is based on Excel use while retaining the user's abilities intact. Excel by its nature is a big table with each cell being easily editable, however, the experience of using it is greatly influenced by the structure of the data the user is working with. If the data is inconsistent when certain records have some attribute, but others don't, it leaves an empty cell in the row of records where these attributes are irrelevant. This can make the table look incomplete and hard to distinguish relevant attributes.

6.2.4.1 Badges

To keep the ease of editing data in Excel we have decided to use table components as a main way to show data around the website. This is due to the table component from Vuetify being the best option to include inputs while keeping data structured and readable.

However, as we decided to use tables, we came to the same problem that Excel has when displaying inconsistent data records. For this reason, we have come up with the concept of “badges”. Badges are small pill-like chips with information representing student’s attributes which can be assigned to students. This concept enables a user to assign an attribute to the student without the need for it to be present in all other students. They are program-specific but can be easily created, modified, or deleted on the General Data page. Furthermore, we had to think of a way to represent all the badges to be able to assign them to students without cluttering the view of each student object. At first, we thought of putting all the badges inside the row, and the user would need to press on them to assign them to a student. However, we have decided to change our approach, here is why. In Figure 6.7, you can see a view of a single student record. Imagine if a program had ten badges and all of them would be shown inside the record, some activated and some not. This would also create a cluttered look, similar to Excel. So, we have conceived to have a sliding panel that would have all the badges and if users want to assign a badge to a student they can simply drag and drop a badge to the student's row.



	Student ID	Name	Nationality	Year Started	Programme	Eligibility	Comment	Action
☐	1234567	Marius Pana	Romanian	2023	TCS	Select: ELIGIBLE	Comment: Test eligibility	DELETE

Figure 6.7: Badges panel and badges assigned to a student on the Module Management page, Applications tab

6.2.4.2 Information relevance

Another design decision we had to make was based on how we wanted to build our data model to save information on students. We thought of a way to make it more structured compared to the Excel example sent to us by Module Support. We have noticed that the file contains all the students who have ever applied to save their information for future use, which is a good way to avoid retrieving the same student multiple times. However, as it is also the same place where Module Support decides on the eligibility of students, the view is cluttered with all other students who have not even applied to a TA role in the current module. Therefore, we have decided to limit the view of the data to a specific module, to display only relevant information while deciding on eligibility. As modules repeat every year we had to formulate a strategy of distinguishing them. For that matter, modules are connected to study units by inheriting their name, but the year is different, so a specific module is a combination of year and study unit.

6.2.4.3 Import and Export

In addition, from the beginning of our project, we acknowledged the possibility of a module coordinator using our system. However, we also recognized that some of them may be unwilling to do so. If so, module support and module coordinator would keep information exchange through Excel files via email. We needed a way for Module Support to easily put records from Excel to an appropriate table and after modifying it in the application, send it further as an Excel. For this reason, the “Module Management” page in the tabs “Applications” and “Student Jobs” tables have import and export functionality. Users are able to import data to the table or select records and export them. Furthermore, as our primary objective is to reduce the amount of manual work done by the users of the system, when students are imported into the system, we are using a connection to LDAP (system by which we request information from the university described in chapter 5.1) to retrieve data that is available for the student and fill it automatically.

6.2.4.4 Storage and state management

Moreover, we have included a storage and state management feature in the application. The main reason for this is to improve the user experience while using the application. By saving data such as settings that the user has chosen on the pages, and the information required for authentication we minimize repetitive steps while working with the application. For example, without this feature, if the Applications tab on the Module Management page would be reloaded, the user would need to select program, year, and module again. By saving some server variables in the front end on application load we reduce the total amount of API calls made while the application works to keep the application fast and its latency small.

Chapter 7

Testing

This chapter is dedicated to the testing of our system. The testing is split into two parts: frontend testing and backend testing, as they were tested in very different ways. The frontend testing plan includes the scope of testing, mentions the used framework, and describes different testing scenarios. The test results are then presented and analyzed. The backend testing plan explains how and why the tests are divided into specific categories, explaining the scope and purpose of each test. Results include an analysis of what the success of each test implies, as well as how the system could be tested in the future

7.1 Test Plan

7.1.1 Frontend Testing Plan

The purpose of this test plan is to ensure the quality, functionality and correctness of the front end of our application. The tests will cover different aspects of the front end, starting from the login page and ending with the correctness of module creation…

Scope
This test plan covers testing of the Vue.js, HTML, and CSS frontend application developed under the name of "TA Database". The test will include only a desktop view, as it was not required for the application to work on any other devices. All the tests will be conducted in the browser Google Chrome with a default scoping of 100 per cent.

Testing framework
Testing the front end is a very time-consuming and difficult task to execute properly. For this purpose, we had to select a specific framework that is effective and efficient. There is an open-source project aimed at supporting browser automation, which is called "Selenium". We decided to base our frontend testing on it because it executes tests quickly and accurately, reducing the likelihood of human mistakes and ensuring consistent test results.

There are several test cases that we prepared to test the most possible aspects of the front end.

User Interface Testing
One of the first and most obvious aspects that we are testing, is the functionality, and presence of the specific UI elements. This is done via our selected framework. If any of the tests do not execute till the end, that indicates that there is a problem with finding or interacting with a UI element that is used throughout the specific test. The tests show if the element is not present or not interactable, in case that is the cause of failure.

Functionality Testing
There are several workflows that we are testing with Selenium.

First, we are testing the functionality of the login page and the logout button. The test is trying to log in with a specified OKTA account and if the login is successful, it logs out immediately. The test requires being connected to the University of Twente network or to be connected to the eduVPN (University of Twente). The purpose of the test is to see if the login page is functioning and redirecting correctly, check if the okta credentials are still valid and authorization works, and check if the logout button properly sends requests and cleans cookies. The successful test execution should leave the testing environment on the login page.

Second, we are testing the “Module Management” page. This test bypasses the login sequence, as it is tested separately, and uses the “bypass login” button that is only available in the development environment. The test checks if the different data tables are displayed on the pages. The purpose of the test is to test the navigation of the system, the module selection and academic year drop-down buttons, and the data table mode navigation.

Third, we are testing badge creation and removal. That is part of the functionality of the “General Data” page. This test also bypasses the login sequence and immediately goes to the relevant page, clicks the “Create badge” button, and fills in the opened dialog. After creating the badge, it compares the amount of badges in the system before and after creation. The initial amount should increase by one. After comparing, it removes the created badge and checks the total amount of badges again, now it should be the same as at the start. The purpose of this test is to see if the tables are updated properly and if all the data contained in them is up-to-date.

Fourth, we are testing the assignment of badges to a student on the “Module Management” page. This test also ignores the login sequence and immediately navigates to the mentioned page, opens the badge panel, and drag and drops one of the badges to the first student in the table. Then the test verifies that the student now has a new badge with the same ID as the one selected from the panel. The purpose of the test is to verify the correctness of the drag-and-drop approach to the badge assignment and to check if the badges are updated in the data table in real time.

7.1.2 Backend testing plan

To ensure the integrity of the API calls we devised four categories of tests, each serving a specific purpose ensuring reliability, resilience, security, and functionality. All tests were implemented via the integrated Django testing framework.

Basic tests:

- **Objective:** Validate the functionality of HTTP requests on all Django models.
- **Scope:** Test CRUD (Create, Read, Update, Delete) operations on each model, ensuring that data can be manipulated correctly via HTTP requests.
- **Coverage Criteria:** Aim for full coverage of basic CRUD operations for each model.
- **Examples:** Test creation, retrieval, updating, and deletion of model instances via HTTP requests.

Error tests:

- **Objective:** Verify that API calls return appropriate errors and handle exceptions gracefully.
- **Scope:** Test scenarios where erroneous input or conditions may occur, ensuring that the system does not break and responds with the correct error messages.
- **Coverage Criteria:** Cover a range of potential error scenarios, including validation errors, database constraints, and unexpected inputs.
- **Examples:** Test error handling for invalid input, database integrity errors, and server-side exceptions.

Access tests:

- **Objective:** Validate appropriate access controls for module support and module coordinators.
- **Scope:** Test user permissions and access levels, ensuring that module support and module coordinators can perform their designated tasks without unauthorized access.
- **Coverage Criteria:** Cover all roles and permission scenarios.
- **Examples:** Test appropriateness of information given for module support and module coordinators. Module support should have access to any information, while coordinators must be restricted to their respective modules.

Integration tests:

- **Objective:** Test the integration of multiple components and requests involving multiple tables.
- **Scope:** Validate sequences of the API calls that often will be used together in a typical use case, focusing on scenarios where data manipulation across multiple tables is necessary.
- **Coverage Criteria:** Cover integration points and data flow between models, ensuring that import requests function correctly and maintain data integrity.
- **Examples:** Test user login and confirm that the user is authenticated.

Our strategy is to run the test regularly during development, especially when working on a new feature, as well as before each deployment. To enforce testing, we used GitLabs ci/cd pipeline, to run automated tests on each push request.

7.2 Test Results

7.2.1 Frontend testing results

Testing environment
All the front-end tests were conducted with the functionality of the Selenium framework. Here are more specific execution details:
- Operating System: Windows 10;
- Browser: Google Chrome (Automation mode), version: 124.0.6367.60/61;
- Devices: desktop, laptop;
- Window scale: 100%.

Testing results
- Login and logout: executed correctly;
- Module Management page: executed correctly;
- Badge creation and removal: executed correctly;
- Badge assignment to student: most of the time executes correctly.

Summary
The Vue.js frontend application has been partly tested and meets the specified requirements. Almost all test cases have been executed successfully, only one test scenario is not executed correctly due to imitation of drag & drop action throughout the Selenium framework. Minor issues were discovered during testing and have been logged for resolution in the future.

All in all, many other tests can be written to test different aspects of the front end, but even with the automation framework, the tests are very time-consuming and difficult. One small change in the visual part of the system can stop all tests from executing correctly. Our conclusions after testing are:
- Create more deep and detailed tests;
- Use current tests in the development environment to check the correctness of the most important functionality of the system;
- Optimize tests and make them future-proof.

7.2.2 Backend testing results

Testing environment

All backend tests are implemented using Django's testing framework. Each test class is inherited from ‘APITestCase’ with setups for test data.

Testing results

- Basic tests raise no issues, which indicates solid ground for future features.
- Error tests validate the appropriateness of error codes, making the system resilient to errors and informative in its error messages.
- Access tests are complete without errors and with expected outputs, giving us assurance in the correctness of access levels.
- Integration tests execute all necessary steps flawlessly, confirming the validity of the implementation.

Summary

All crucial aspects and developed features of the system are successfully tested. Validating the implementation and giving reasons to be sure of it.

Conclusion

Despite all tests executing with no errors, many other aspects of the system could and highly likely should be tested in the future. Metrics like time performance or resource intensity are not relevant for the application now due to the small target user base. However, this might not always be the case.

Additionally, with the introduction of new features, new error, integration, and access tests must be written. While updates to the data model might require adjusting or adding tests to reflect new changes.

Chapter 8

Future Planning

8.1 Utilization and Support of the System

The developed application is not perfectly polished and lacks some built-in hints to help users navigate through the functionality. Therefore, one possible future improvement could be creating hints in the application which could help users orientate around the website. In addition, a manual which contains information about how the overall web application can be written from the perspectives of both Module Support and Module Coordinator officers could be constructed by providing details about the use cases of the web pages.

8.2 University-wide Enrollment

Currently, the functionality of our application is adapted to the needs of the EEMCS department, and TCS and BIT programmes, to be more specific. This leaves a lot of space for future improvements, as we have not gathered any requirements from the Module Supports and Coordinators of other faculties and programmes. Such requirement gathering could provide a lot of valuable insights and the potential to implement new features.

The purpose of the application would not change with new requirements, but its functionality would expand, which might make it more complex. This will lead to a need to implement our first future improvement.

8.3 Expanding access to the system

From the beginning of the developing requirements for the system, we counted on Module Support and Module Coordinators being the sole users of it. However, as we got to know the full flow of the current process from the Module Support, we have noticed that even with our system process still requires Excel file exchange on the step of student information gathering. Moreover, during presentations about the progress we've made, students have provided feedback indicating their willingness to use the system. They would appreciate it if the system could store the information they input in their TA applications for each module they apply to.
For that reason, we have thought of a “Student” role in the system. With this in mind, applying to become a TA, availability scheduling, tracking the progress of the application process by the student, and even communication between students and Module Support could

be done through the system in the future. It would significantly reduce the need for file exchange and the chances of missing any data while doing so.

8.4 Remainder Requirement Integration

There are still a few requirements that were unable to be implemented due to the complexity and lack of time. Now we will go over the requirements that were left over and discuss them.

First, we did not implement the requirement to archive students who graduated or dropped out. This is an important possible future improvement, as it is required by GDPR, which states that the system should only store relevant data or have it stored adequately.

Second, we have not implemented a feature that could be done on the Student Jobs tab on the Module Management page is the calculation of hours per week, which is done through the form on the university HR website in the current system. However, due to a lack of time and difficulties in connecting to the HR system, we haven't implemented the calculation but allow the user to modify the field. By connecting our system to the HR system we could avoid the need for Module Support to use another system, consequently improving user experience.

Third, the current logging system could be improved for easier tracking of user actions. Currently, the Django REST framework saves all user API calls with timestamps, but this format of logs is not practical as it contains too many details. One of the improvements could be to refactor Django REST logs or to log the actions on the server side by ourselves, which would allow us to have an adjustable logging system.

Chapter 9

Evaluation

This chapter will evaluate the project's planned direction and progress, state the assigned team member responsibilities, and conclude with an introspection on the final result.

9.1 Planning

We planned biweekly stretch goals, at the end of which took place a Peer Feedback Session to showcase our progress and receive critical advice on our work. All tackled requirements in Chapter 4 have followed this timeline for their components in planning. However, the full completion of some has deviated from their proposed deadline due to certain features taking longer than expected and having handled extraneous work to the project during the module:
Week 1 - stakeholder introduction & requirement round-up
Week 3 - front-end Figma design and back-end groundwork & project proposal and planning
Week 5 - front-end groundwork and back-end finalization & test plan
Week 7 - application's front-end core features' functionality & testing beginning & design report
Week 9 - feature polish and transition to production & testing finalization
Week 10 - project and poster presentation & system and report finalization and distribution

Intermediate goals were established weekly with the supervisor and updated in group meetings when necessary. Testing was done alongside development.

9.2 Responsibilities

The group members have chosen the sections they feel most comfortable working on, and each has agreed with the others' decisions while considering the workload proportions divided among ourselves. While we have generally stuck to the following distribution of responsibilities, the frequent discourse has ensured an adequate integration between each component:

Front end: Arda Konça, Mark Troicins, Vladislav Mukhachev
- **Design**: Mark Troicins, Vladislav Mukhachev
- **Design implementation**: Arda Konça, Vladislav Mukhachev, Mark Troicins
- **API Request handling**: Arda Konça, Vladislav Mukhachev
- **Testing**: Mark Troicins

Back end: Ervīn Zvirbulis, Marius Pană
- **Authentication**: Ervīn Zvirbulis
- **Data Model**: Marius Pană
- **Building API Requests:** Ervīn Zvirbulis, Marius Pană

- **Deployment**: Ervīn Zvirbulis
- **Testing**: Ervīn Zvirbulis, Marius Pană
- **Swagger documentation:** Ervīn Zvirbulis

All five group members contributed equally to the Design Report by documenting their work and splitting shared topics. Likewise, participation in the bi-weekly presentations and stakeholder meetings was also a joint effort.

9.3 Team Evaluation

Most of the time our team looked and progressed in the same direction, but not everything was perfect. There were many cases of miscommunication or lack of communication which led to each of us having slightly different visions of the requested features or the ways to design and implement them. This led to some small conflicts inside of the team, but the conflicts resolved quickly and everybody was able to agree on something. We did not dedicate any time to having special team evaluation sessions, but those maybe could have been useful. There were no serious disputes during the project, and in the end, the team is satisfied with the results and the spirit that has been with us throughout this time.

We could have had more structured and strict planning, could have better split some responsibilities, and could have better used some of the feedback from the supervisor. We are not perfect, so such problems are normal, though we should always strive for the best.

9.4 Final Result

The system exceeds the status of an MVP, having implemented all the must requirements, and the client is pleased with the final result. But despite that, there is always room for improvements in how the system could further extend its current functionalities. For that reason, chapter 8 has been contemplated as the steps that could be considered to further enhance the ease of usage and the benefits of the proposed system.

To see the deployed application go to https://tadb.apps.utwente.nl/.
Okta usernames:

- **Module support:** *s@m.com* (full access)
- **Module coordinator:** *c@m.com* (access limited to assigned modules)
- **Student:** *fs@m.com* (no access, but can log into the system because of the use of university credentials)

Password is the same for all: *TestIWantTo* .
Keep in mind that the deployment is not 100% stable, if the system shows any signs of unexpected behavior, please, reload the page.

Due to the limitation of using developer OKTA accounts, it does not always work on the deployed prototype. Alternatively, the login page allows to bypass the identity provider and gain access as a module support. The final production version will have such functionality removed.

9.5 Conclusion

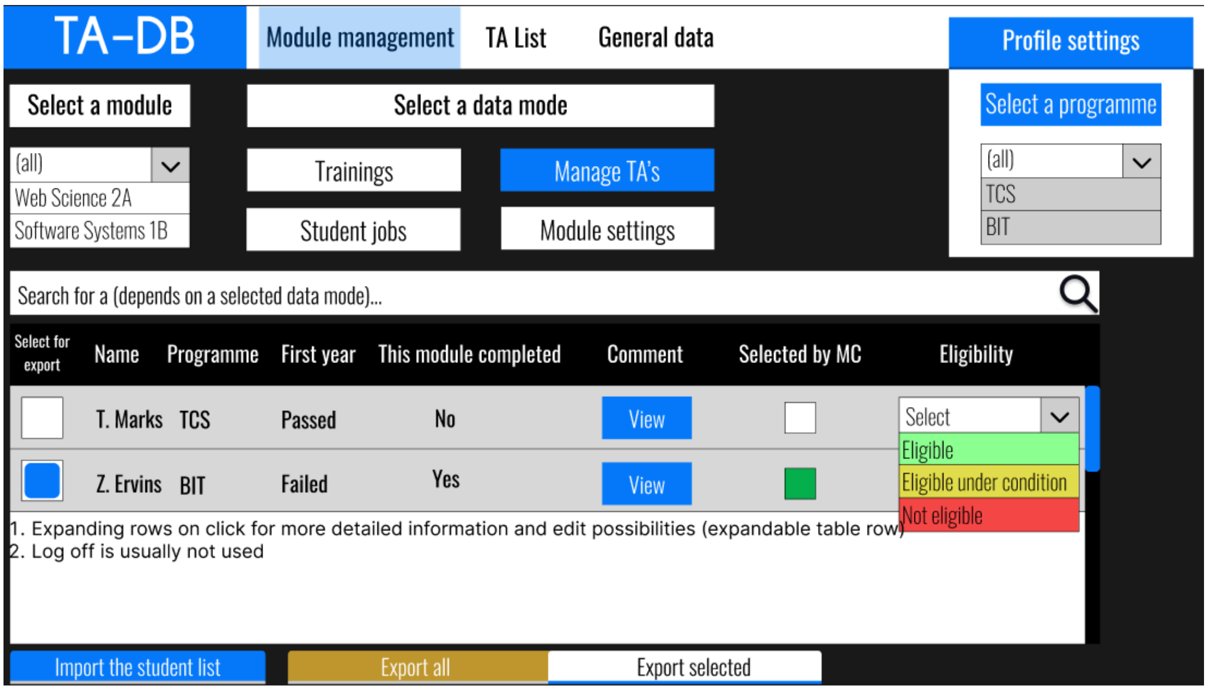
As previously indicated, to avoid the privacy, resilience, and efficiency deficiencies with the current methodology of UT staff in managing students to hire as TAs, it has been decided to come up with a proposed web application system which allows Module Support and Module Coordinator officers to easily determine the eligibility of applicants and arrange the number of hours the students need to work within a module. The proposed application at the end of the project works in such a way that the officers could log in to their OKTA account to let the system be aware of their user roles, and accordingly, the officers only see the relevant pages of the web application. On top of that, the application has the main functionality of whenever an officer imports a TA applicant, the officer can edit and assign them their attributes, the previously mentioned badge concept, and the information related to an applicant gets stored in the designated page (Student List) automatically by the system, where their information is always accessible and editable by the appurtenant officers at any time. This is done in an approach that saves the applicant's information - which can be related to their eligibility or job details; right away. Furthermore, considering development progress, the proposed system's expected functionality is ready and can already be observed in the deployed web application. However, there still are some additional features, covered in Chapter 8, that could still be implemented in the system to further improve the serviceability of the overall system.

Lastly, it is important to highlight that besides the advantages of users utilizing the proposed application - its primary purpose, the developers gained much knowledge from their work on the project. We wrap up the written report by stating they learned to work within the provided frameworks from scratch and developed their time management, teamwork, problem-solving, and collaboration skills.

Appendices

Appendix A.

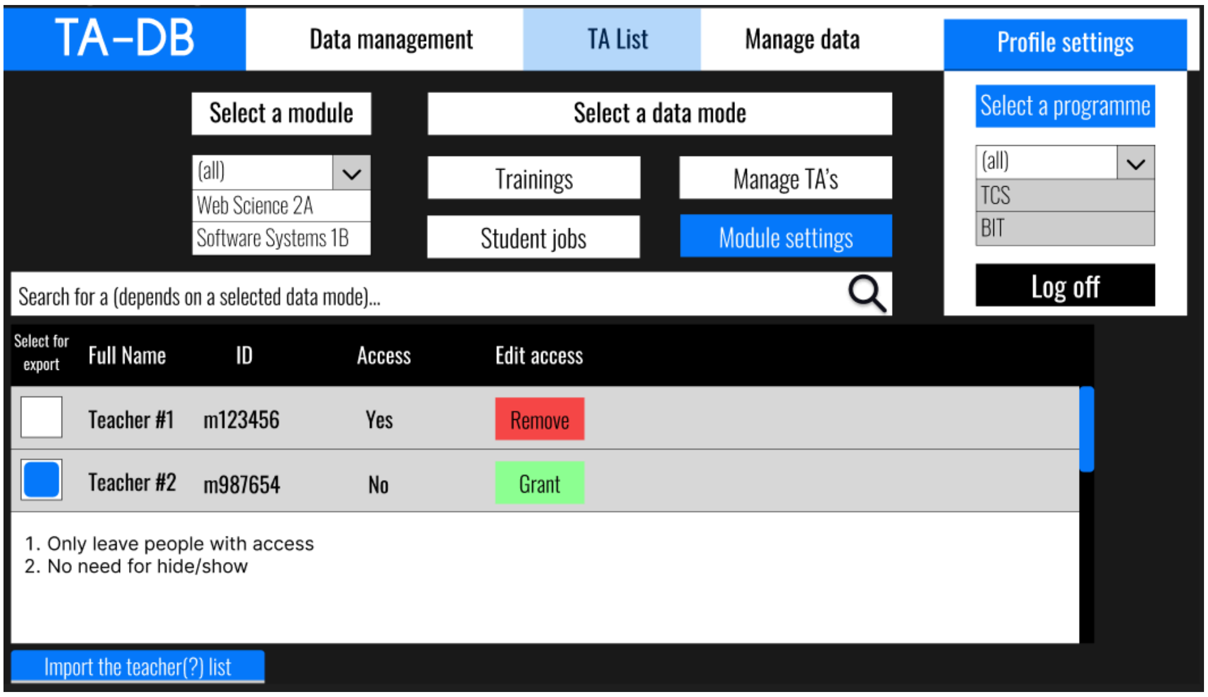
Figma Design Mock-ups



Select for export	Name	Programme	First year	This module completed	Comment	Selected by MC	Eligibility
☐	T. Marks	TCS	Passed	No	View	☐	Select
■	Z. Ervins	BIT	Failed	Yes	View	🟩	

1. Expanding rows on click for more detailed information and edit possibilities (expandable table row)
2. Log off is usually not used

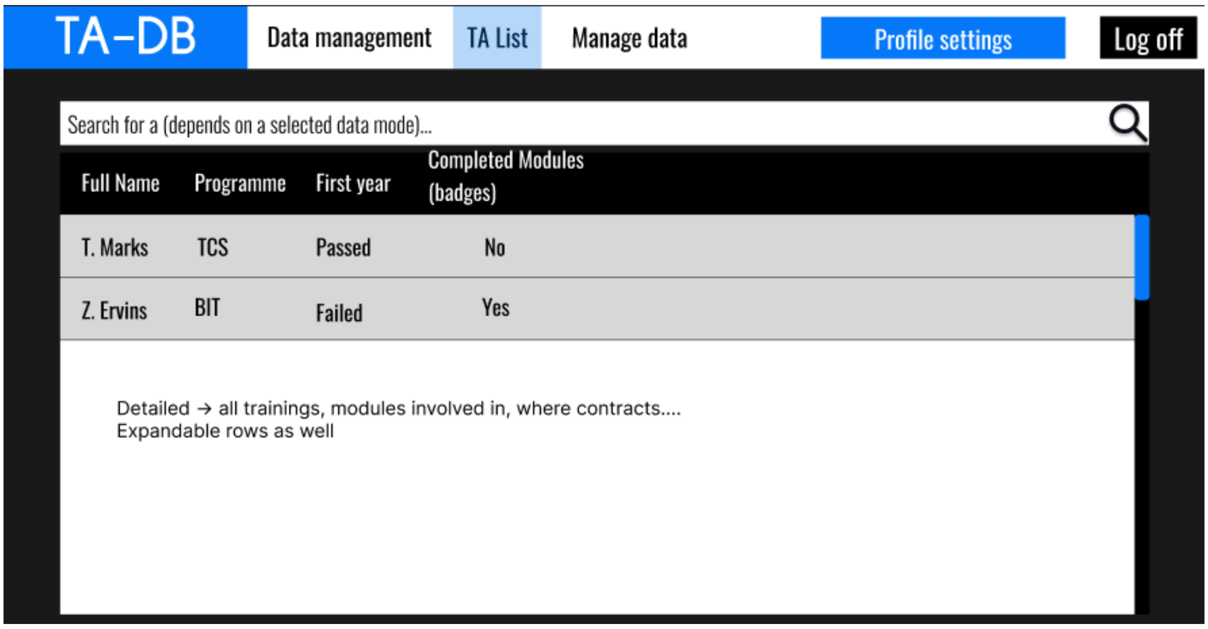
Figure A.1: Mockup of the Module Management page with some comments from the supervisor.



Select for export	Full Name	ID	Access	Edit access
☐	Teacher #1	m123456	Yes	Remove
■	Teacher #2	m987654	No	Grant

1. Only leave people with access
2. No need for hide/show

Figure A.2: Mockup of the Student List page with some comments from the supervisor.



Full Name	Programme	First year	Completed Modules (badges)
T. Marks	TCS	Passed	No
Z. Ervins	BIT	Failed	Yes

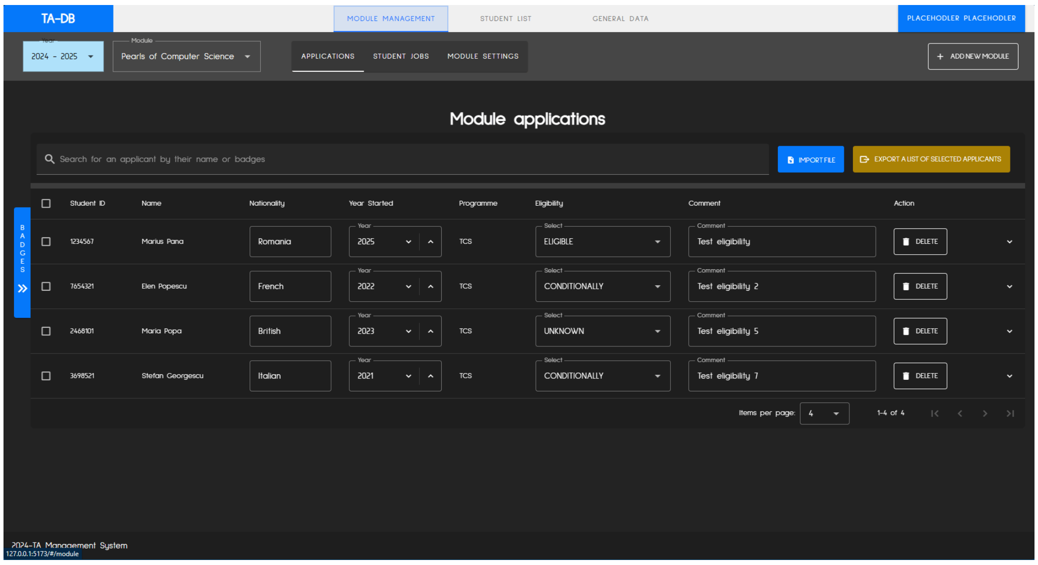
Detailed → all trainings, modules involved in, where contracts....
Expandable rows as well

Figure A.3: Mockup of the Student List page with some comments from the supervisor.

Appendix B.

Final Design Pages

Module Management



2024 - 2025

Pearls of Computer Science

Module applications

Search for an applicant by their name or badges

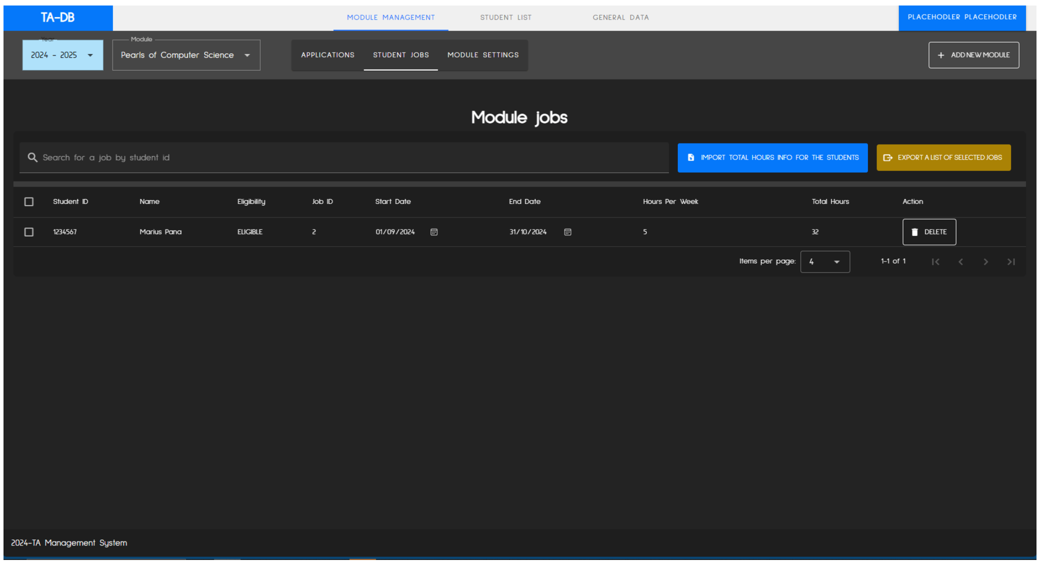
Student ID	Name	Nationality	Year Started	Programme	Eligibility	Comment	Action
1234567	Marius Pana	Romania	2025	TCS	ELIGIBLE	Test eligibility	DELETE
7654321	Elen Popescu	French	2022	TCS	CONDITIONALLY	Test eligibility 2	DELETE
2468101	Maria Popa	British	2023	TCS	UNKNOWN	Test eligibility 5	DELETE
3698521	Stefan Georgescu	Italian	2021	TCS	CONDITIONALLY	Test eligibility 7	DELETE

Items per page: 4 1-4 of 4

2024-TA Management System

127.0.0.1:5173/#/module

Figure B.1: Final design of the Applications tab on the Module Management page.



2024 - 2025

Pearls of Computer Science

Module jobs

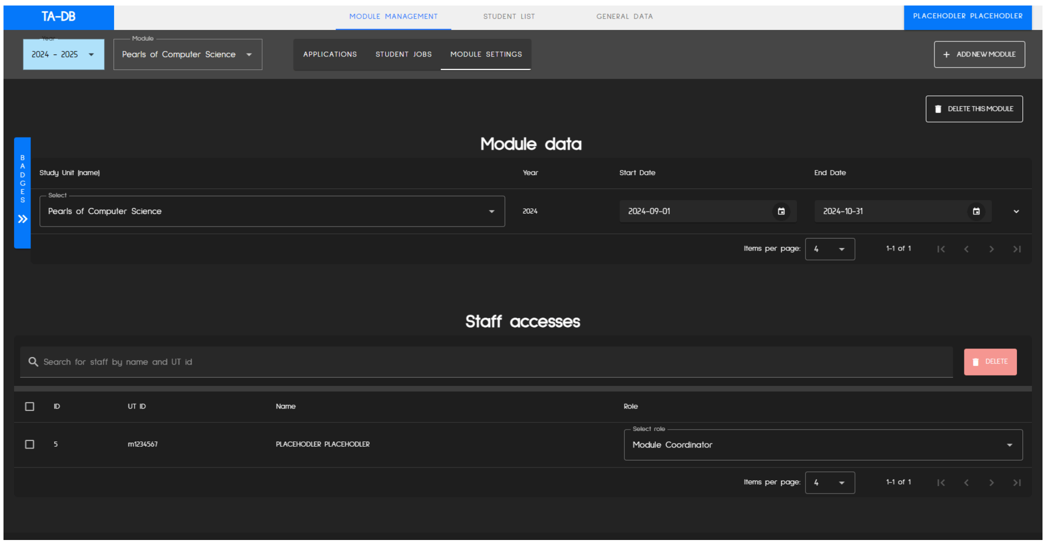
Search for a job by student id

Student ID	Name	Eligibility	Job ID	Start Date	End Date	Hours Per Week	Total Hours	Action
1234567	Marius Pana	ELIGIBLE	2	01/09/2024	31/10/2024	5	32	DELETE

Items per page: 4 1-1 of 1

2024-TA Management System

Figure B.2: Final design of the Student Jobs tab on the Module Management page.



Year: 2024 - 2025
Module: Pearls of Computer Science

Module data

Study Unit (name)	Year	Start Date	End Date
Select: Pearls of Computer Science	2024	2024-09-01	2024-10-31

Items per page: 4 1-1 of 1

Staff accesses

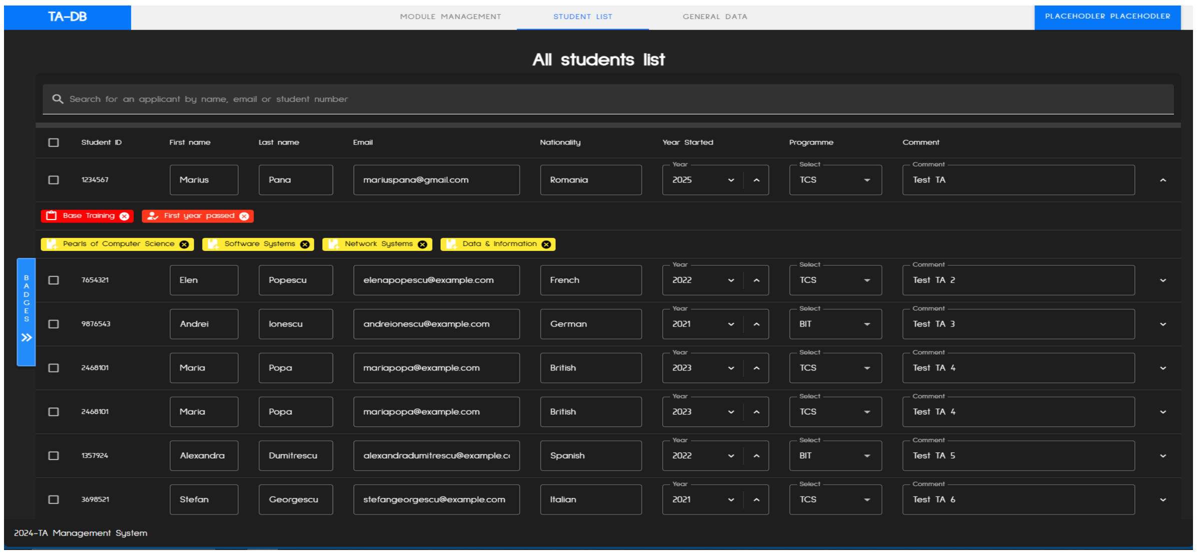
Search for staff by name and UT id

☐	ID	UT ID	Name	Role
☐	5	m1234567	PLACEHODLER PLACEHODLER	Select role: Module Coordinator

Items per page: 4 1-1 of 1

Figure B.3: Final design of the Module Settings tab on the Module Management page.

Student List


TA-DB | MODULE MANAGEMENT | STUDENT LIST | GENERAL DATA | PLACEHODLER PLACEHODLER

All students list

Search for an applicant by name, email or student number

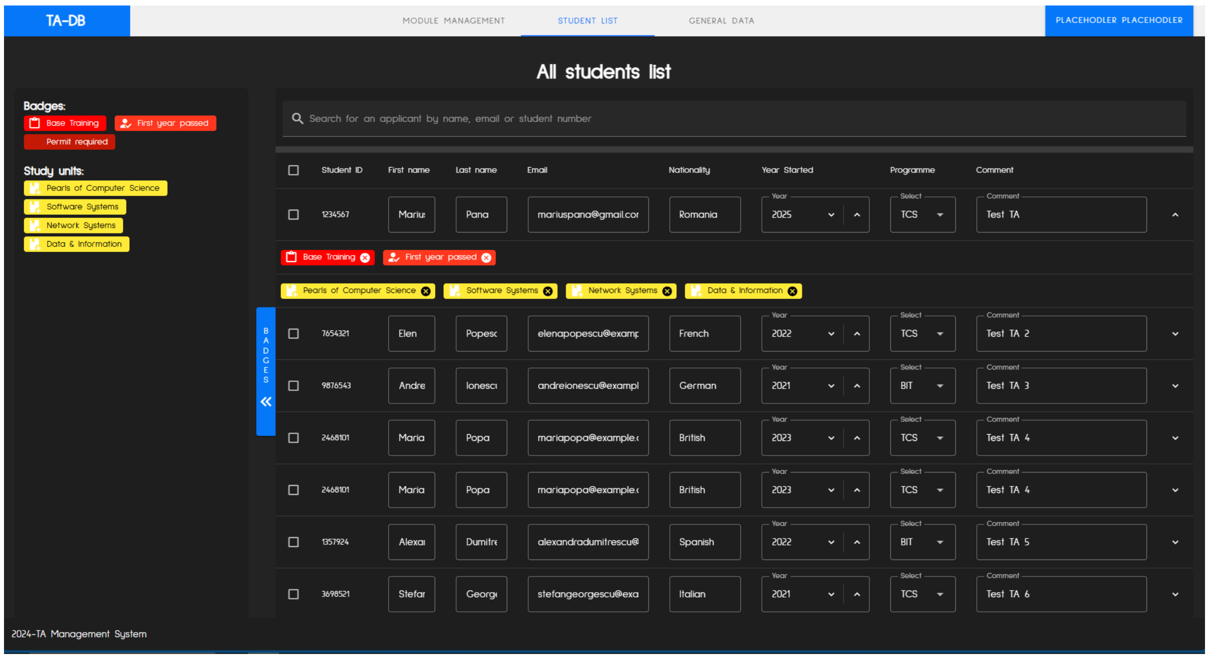
	Student ID	First name	Last name	Email	Nationality	Year Started	Programme	Comment
☐	1234567	Marius	Pana	mariuspana@gmail.com	Romania	2025	TCS	Test TA
☐	7654321	Elen	Popescu	elenapopescu@example.com	French	2022	TCS	Test TA 2
☐	9876543	Andrei	Ionescu	andreionescu@example.com	German	2021	BIT	Test TA 3
☐	2468101	Maria	Popa	mariapopa@example.com	British	2023	TCS	Test TA 4
☐	2468101	Maria	Popa	mariapopa@example.com	British	2023	TCS	Test TA 4
☐	1357924	Alexandra	Dumitrescu	alexandradumitrescu@example.c	Spanish	2022	BIT	Test TA 5
☐	3698521	Stefan	Georgescu	stefangeorgescu@example.com	Italian	2021	TCS	Test TA 6

Expanded row (1234567): Base Training · First year passed · Pearls of Computer Science · Software Systems · Network Systems · Data & Information

BADGES

2024-TA Management System

Figure B.4: Final design of Student List page with first student row expanded.


TA-DB | MODULE MANAGEMENT | STUDENT LIST | GENERAL DATA | PLACEHODLER PLACEHODLER

All students list

Badges: Base Training · First year passed · Permit required

Study units: Pearls of Computer Science · Software Systems · Network Systems · Data & Information

Search for an applicant by name, email or student number

	Student ID	First name	Last name	Email	Nationality	Year Started	Programme	Comment
☐	1234567	Mariu:	Pana	mariuspana@gmail.cor	Romania	2025	TCS	Test TA
☐	7654321	Elen	Popesc	elenapopescu@examp	French	2022	TCS	Test TA 2
☐	9876543	Andre	Ionesc	andreionescu@exampl	German	2021	BIT	Test TA 3
☐	2468101	Maria	Popa	mariapopa@example.c	British	2023	TCS	Test TA 4
☐	2468101	Maria	Popa	mariapopa@example.c	British	2023	TCS	Test TA 4
☐	1357924	Alexa	Dumitr	alexandradumitrescu@	Spanish	2022	BIT	Test TA 5
☐	3698521	Stefar	Georg	stefangeorgescu@exa	Italian	2021	TCS	Test TA 6

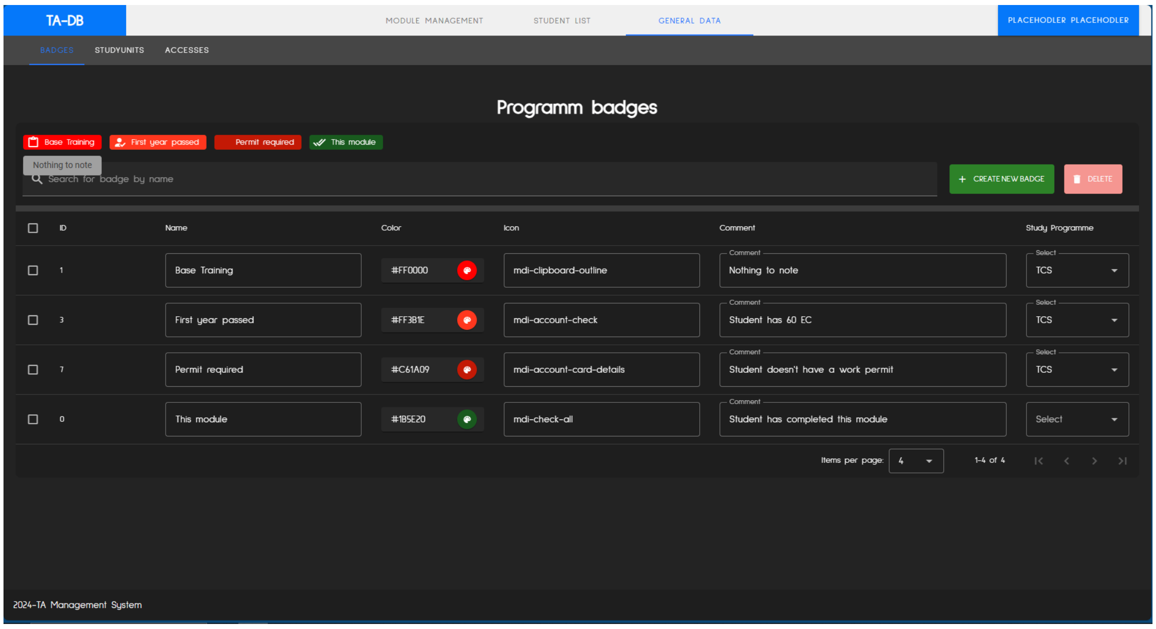
Expanded row (1234567): Base Training · First year passed · Pearls of Computer Science · Software Systems · Network Systems · Data & Information

BADGES

2024-TA Management System

Figure B.5: Final design of Student List page with first student row expanded and Badge Panel open.

General Data



Programm badges

Base Training · First year passed · Permit required · This module

Nothing to note

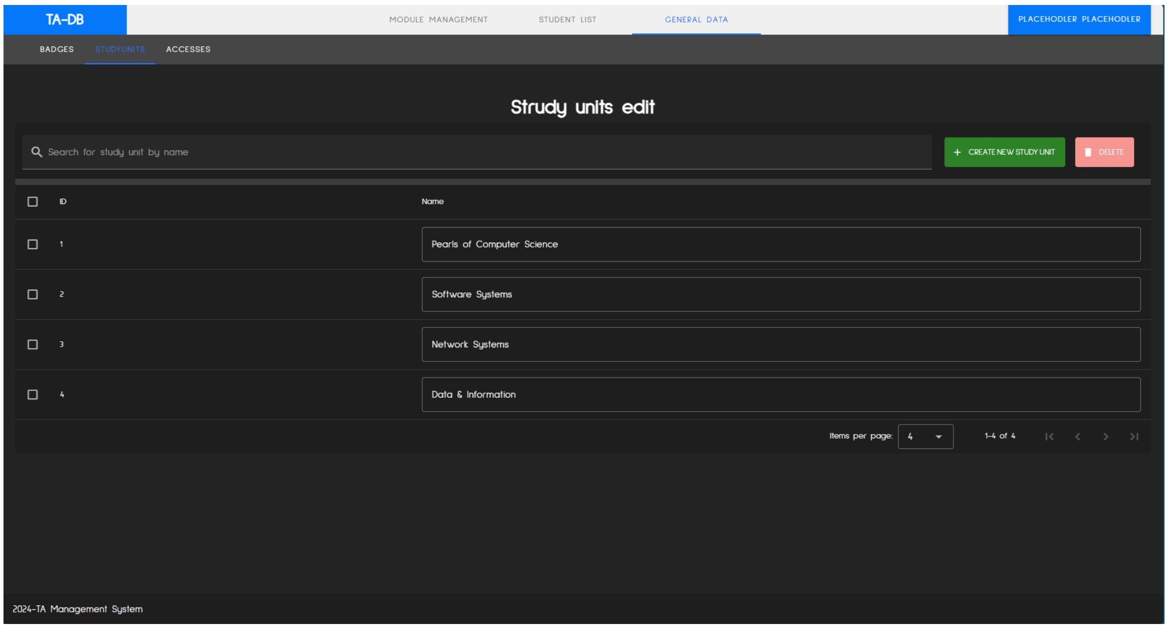
Search for badge by name

	ID	Name	Color	Icon	Comment	Study Programme
☐	1	Base Training	#FF0000	mdi-clipboard-outline	Nothing to note	TCS
☐	3	First year passed	#FF3B1E	mdi-account-check	Student has 60 EC	TCS
☐	7	Permit required	#C61A09	mdi-account-card-details	Student doesn't have a work permit	TCS
☐	0	This module	#1B5E20	mdi-check-all	Student has completed this module	Select

Items per page: 4 1-4 of 4

2024-TA Management System

Figure B.6: Final design of the Badges tab on the General Data page.



Strudy units edit

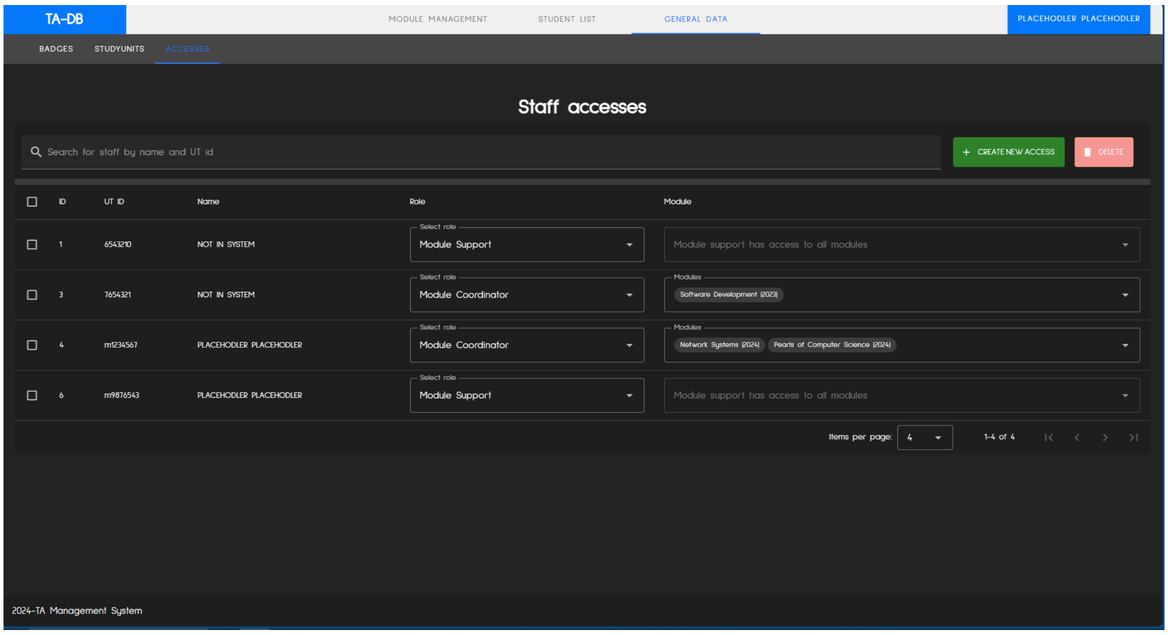
Search for study unit by name

	ID	Name
☐	1	Pearls of Computer Science
☐	2	Software Systems
☐	3	Network Systems
☐	4	Data & Information

Items per page: 4 1-4 of 4

2024-TA Management System

Figure B.7: Final design of the Study Units tab on the General Data page.



Staff accesses

Search for staff by name and UT id

☐	ID	UT ID	Name	Role	Module
☐	1	6543210	NOT IN SYSTEM	Select role: Module Support	Module support has access to all modules
☐	3	7654321	NOT IN SYSTEM	Select role: Module Coordinator	Modules: Software Development (2023)
☐	4	m1234567	PLACEHODLER PLACEHODLER	Select role: Module Coordinator	Modules: Network Systems (2024), Pearls of Computer Science (2024)
☐	6	m9876543	PLACEHODLER PLACEHODLER	Select role: Module Support	Module support has access to all modules

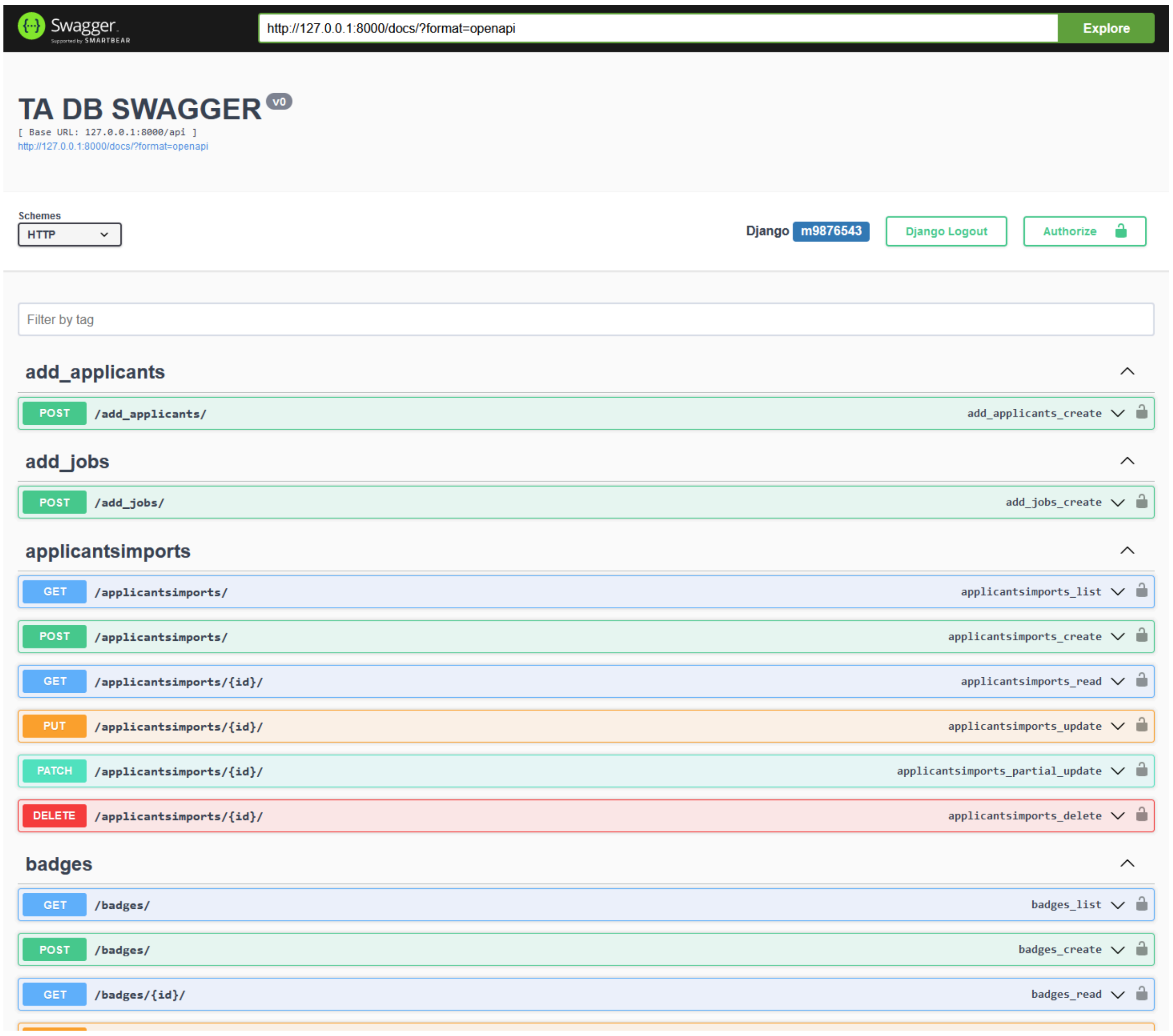
Items per page: 4 1-4 of 4

2024-TA Management System

Figure B.8: Final design of the Accesses tab on the General Data page.

Appendix C.

Swagger Screenshots



Swagger
http://127.0.0.1:8000/docs/?format=openapi
Explore

TA DB SWAGGER v0
[Base URL: 127.0.0.1:8000/api]
http://127.0.0.1:8000/docs/?format=openapi

Schemes
HTTP

Django m9876543 Django Logout Authorize

Filter by tag

add_applicants

POST /add_applicants/ add_applicants_create

add_jobs

POST /add_jobs/ add_jobs_create

applicantsimports

GET /applicantsimports/ applicantsimports_list

POST /applicantsimports/ applicantsimports_create

GET /applicantsimports/{id}/ applicantsimports_read

PUT /applicantsimports/{id}/ applicantsimports_update

PATCH /applicantsimports/{id}/ applicantsimports_partial_update

DELETE /applicantsimports/{id}/ applicantsimports_delete

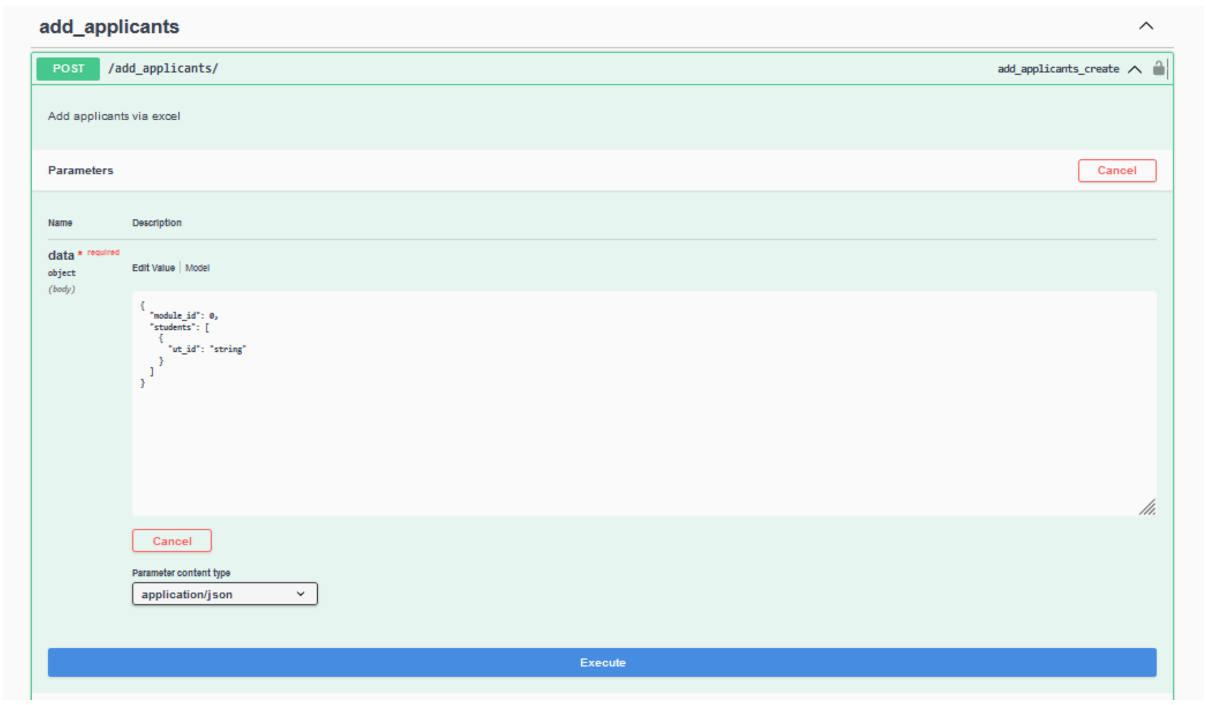
badges

GET /badges/ badges_list

POST /badges/ badges_create

GET /badges/{id}/ badges_read

Figure C.1: Showcase of the list of API calls.


add_applicants

POST /add_applicants/ add_applicants_create

Add applicants via excel

Parameters

Name | Description

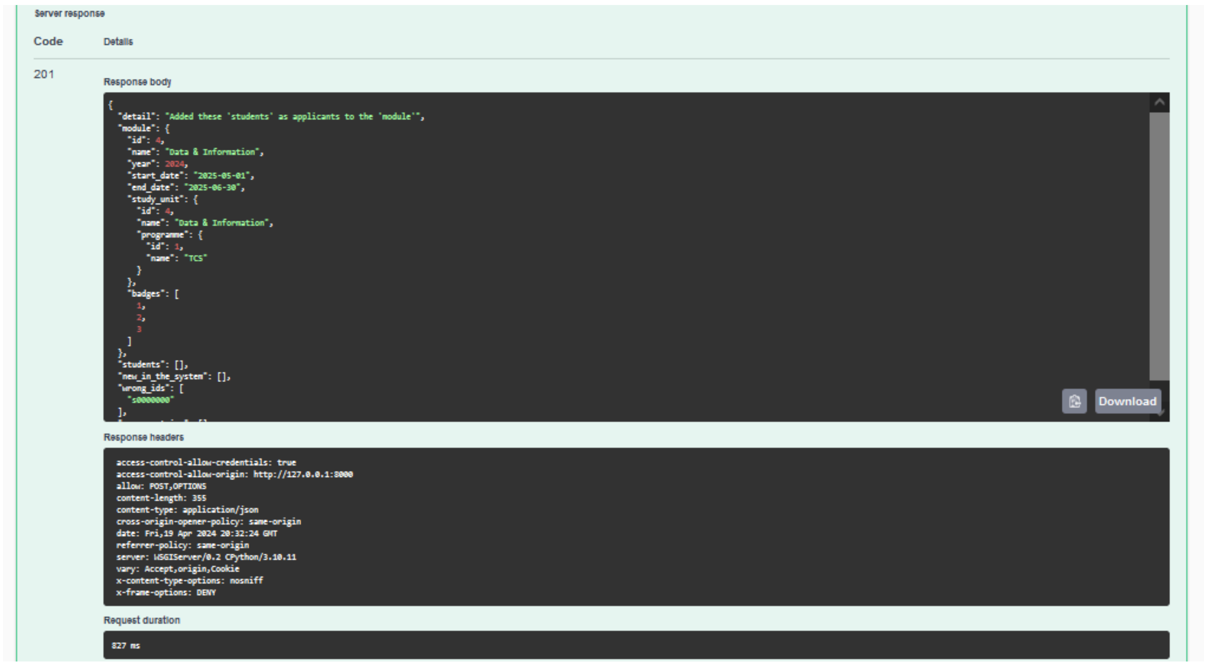
data * required
object
(body)

Edit Value | Model

```
{
  "module_id": 0,
  "students": [
    {
      "ut_id": "string"
    }
  ]
}
```

Parameter content type: application/json

Figure C.2: Template/structure for the POST requests (*/api/add_applicants* in this case)


Server response

Code | Details

201

Response body

```
{
  "detail": "Added these 'students' as applicants to the 'module'",
  "module": {
    "id": 4,
    "name": "Data & Information",
    "year": 2024,
    "start_date": "2025-05-01",
    "end_date": "2025-06-30",
    "study_unit": {
      "id": 4,
      "name": "Data & Information",
      "programme": {
        "id": 1,
        "name": "TCS"
      }
    },
    "badges": [
      1,
      2,
      3
    ]
  },
  "students": [],
  "new_in_the_system": [],
  "wrong_ids": [
    "s0000000"
  ],
```

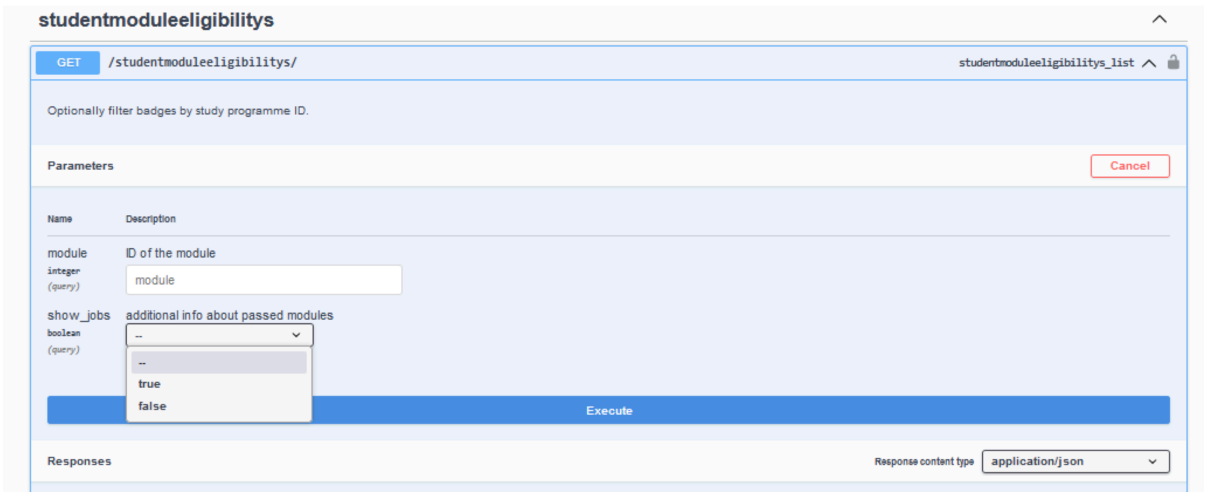
Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: http://127.0.0.1:8000
allow: POST,OPTIONS
content-length: 355
content-type: application/json
cross-origin-opener-policy: same-origin
date: Fri,19 Apr 2024 20:32:24 GMT
referrer-policy: same-origin
server: WSGIServer/0.2 CPython/3.10.11
vary: Accept,origin,Cookie
x-content-type-options: nosniff
x-frame-options: DENY
```

Request duration

```
827 ms
```

Figure C.3: Returned response of call */api/add_applicants*.


studentmoduleeligibilitys

GET /studentmoduleeligibilitys/ studentmoduleeligibilitys_list

Optionally filter badges by study programme ID.

Parameters

Name	Description
module integer (query)	ID of the module
show_jobs boolean (query)	additional info about passed modules

Figure C.4: Example of a call with options.