Educational Information System for the DMB Group

Design Report

Kağan Gülsüm Alex Petrov Vasko Pirinski Robert Ignat Liran Neta

UNIVERSITY OF TWENTE.

EEMCS
University of Twente
Enschede, Netherlands
April 2023

Contents

1 Introduction

The Data Management and Biometrics group (DMB) wants to have an Educational Information Systems (EIS) in which students and members can register and update the educational, supervisional and managerial activities and can get an overview of all of these activities of the DMB group. This system is required as to facilitate better management of the group and its members. Statistical overviews of the work that is handled by a certain teacher is important both for the management of the DMB group, as well as the teachers themselves. Management is concerned with how tasks need to be distributed in the best way and outlining contributions, whereas teachers are interested in following their activities per period and understanding their load.

This report aims to cover the design and implementation process of the new information system for the DMB group. This system served as the assignment for the bachelor's design project module.

To begin this report, the first section after this introduction provides a detailed specification of the requirements 2 of the system. The requirements underwent a number of design choice cycles and were modified to reflect to the best extend the wishes of the clients.

After the requirements specification section, the global design of the system is introduced 3. In this section a general overview of the system is provided with all of its components and the connections between them. Topics such as the technologies used and methodologies are covered.

The detailed design and the choices that lead to it are presented in the design choices section 4. This section is split into multiple parts, as the EIS system itself has 3 integral components, namely front-end, back-end and database design. There is one additional section, related to the virtual machine design, which relates to the deployment of the system and its usability by the DMB group. The section aims to provide a thorough explanation of the process behind the design of the system and how it lead to a functionally appealing product.

A software product is only as good as its testing procedure. And our test plan and results are reflected in the next section of the report 5. Unit testing, network API and automated system tests were the main test tools the team used to validate the correctness of the system. After a detailed explanation of the test plan and the technologies used for testing, the relevant results of the whole testing procedure are highlighted.

To finish such a project, our team had to be set up in a good environment, with constant communication and understanding of the tasks at hand. For that, we allocated tasks to every member and split the team into two sub teams, focused on respectively front-end and back-end functionalities, constantly working in close cooperation. This led to the efficient execution of the requirements set up by the stakeholders. The contributions of each member are highlighted in the contributions section of the report 7.

Finally, this report is concluded 8 with a reflection on the whole process, where the question 'What could be been done better?' is answered, followed by 'What went well?'. Further, final remarks illustrate the opinion of the whole team on this project and finish off with ideas for further work that can be done on the EIS.

2 Requirements Specification

The proposed system consists of a front-end, a back-end, and a database. The front-end will be a single page web application written in Angular. It will make calls to a back-end server API, which will be built using Python Flask. The database will be created with PostgreSQL. All three components will be hosted on the DMB group's intranet.

Below are the functional and non-functional requirements with respective categorizations. The prioritization of these requirements are color coded with respect to the MoSCoW principle (**Must**, **Should**, **Could**, **Won't**).

2.1 Functional Requirements

2.1.1 Account management & access levels

Requirements regarding accounts of teachers and different access levels.

2.1.1.1 Access levels

Teacher — completes forms for activities, views and edits data concerning them.

Admin — completes forms for activities, views and edits all data. (Consider responsible chairs of the DMB group and the technical personnel.)

2.1.1.2 Account management requirements

Front-end

- Teachers & Admins should be able to log in using their UT account.
- Teachers & Admins should be able to log out.

Back-end

- The server should process the Microsoft Azure AD token received in requests and only process the request if the token corresponds to a teacher/admin with the right access level for that request.
- Teachers should be able to have their accounts registered using their UT account.
- The admin should be able to change the access levels of accounts.

Database

- The DB should store, for each account, the corresponding Microsoft unique ID used for authentication (which is not confidential).

2.1.2 Data collection & forms

Requirements regarding forms through which teachers report their activities and consequent data collection procedures.

2.1.2.1 Educational component

Front-end

- Teachers should be able to view their educational-related activities in a table. Each activity in a different row.
- A teacher should be able to enter all relevant data for the educational activity.*
  - A teacher should be able to select the teacher that would participate in the educational activity.

- A teacher should select only one teacher from the options provided by the server.
- A teacher should be able to select the academic year for this educational activity, from the options provided by the server.
- A teacher should be able to select at least one quartile in a year, including Q5.
- A teacher should be able to mark the selected teacher as a main contact person.
- A teacher should be able to select the degree of the course, such as Bachelor or Master.
- A teacher should be able to specify another degree other than the ones that are presented by the application.
- A teacher should be able to select a module name from the options provided by the server.
- A teacher should be able to to select the participating studies (one or more) from the options provided by the server.
- A teacher should be able choose a course name, from the options provided by the server.
- A teacher should be able to specify course credits.
- A teacher should be able to specify approximate number of students.
- A teacher should be able to specify approximate contribution to modules in percentage.
- It is mandatory for all the fields to be filled by a teacher except the approximate number of students and the contribution to modules.
- If a teacher selects a Master degree, the module name field should be skipped.
- If a teacher selects a PhD degree, there should not be a participating study option.
- If a teacher selects a Master degree, only masters participating studies should be available for selection.

- Teachers should be able to add an educational-related activity, through an "add button" underneath the activities table. Pressing this button leads to the above mentioned form, and its completion should result in a new activity.
- Teachers should be able to edit and delete their educational-related activities through buttons, which should be located at the right side (end) of the row.
- Admins should be able to perform all of the aforementioned actions (view, edit and delete all the educational-related activities) for all the teachers in the system.
- Filtering of the data should be done by choosing out of a drop-down list, for different fields of the activity (for instance filtering by Module). The exact parameters of filtering will be discussed in more detail with the product owner and updated later.

Back-end

- All the form data that is given as input to the front-end, is parsed by the back-end and sent to the DB.
- The statements to retrieve data from the DB are handled in the back-end by a request handler.
- The statements to update or insert new data to the DB are handled in the back-end by a request handler.
- The request handlers should support filtering based on the front-end input.
- The server should contain a request handler for getting all possible module names, teachers, academic years, participating studies, and course names.

- The server should contain a request handler for getting the educational activities of one teacher (if Teacher access level is given) or all teachers (if Admin access level is given).
- In case the user has the Teacher access level - the activities will be retrieved based on the teacher's access token which is sent from the frontend to the backend.
- Only modules and courses that are stored in the DB should be accepted as valuable input.

Database

- A database table should hold all data for the educational activities.
- The educational activity table should have a foreign key email that references the teacher table.
- The referenced teacher is the one to participate in this educational activity
- A module table should store names for all modules.
- The module table should be referenced by the educational activity table.
- A course table should store names for all courses.
- The course table should be referenced by the educational activity table.
- A period table should store a triplet of quartile, year and a reference to an educational activity.
- The period table should be used to give quarterly information for educational activities.
- A participating study table should store a reference to an educational activity, the study degree and name.

2.1.2.2 Supervision component

Front-end

- Teachers should be able to view their supervision-related activities in a table. Each activity in a different row.
- A teacher should be able to submit a new supervision activity by entering all relevant data (listed below) in a form.*
  - A teacher should be able to enter the information of the supervised student (e.g. name, student number, email).
  - A teacher should be able to enter information regarding the thesis.
  - A teacher should be able to enter names of the people working in a group (if it's a group project).
  - A teacher should be able to enter the emails of the teachers that supervise the thesis.
- Teachers should be able to add a supervision-related activity, through an "add button" underneath the activities table. Pressing this button should lead to the above mentioned form, and its completion should result in a new activity.
- Teachers should be able to update the status of their supervision tasks.
- Teachers should be able to edit and delete their supervision-related activities through buttons, which should be located at the right end of the row.
- Admins should be able to perform all the aforementioned actions (view, edit and delete all the supervision-related activities) for all the teachers in the system.
- For entering new supervision data, teachers should also be able to upload a Google Forms results document filled in by the student.

Back-end

- The server should be able to query the DB for teachers' supervision activities
- The server should contain a request handler for getting the supervision activities of one teacher (if Teacher access level is given) or all teachers (if Admin access level is given).
- In case the user has the Teacher access level - the activities will be retrieved based on the teacher's access token which is sent from the frontend to the backend.
- The server should contain a request handler for adding a supervision activity. The teacher responsible for the activity should be deduced based on the access token.
- The server should contain a request handler for editing supervision activities. Requests with teacher level access should only be able to edit activities of that teacher, while admin requests should be able to edit any supervision activity.
- The handler for adding supervision activities should also accept a Google Forms results document, which it should parse for filling the activity's values.

Database

- The DB should be able to store supervision activities of teachers.
- The database should allow for retrieving all supervision activities corresponding to a teacher.

2.1.2.3 Management component — generic activity

Front-end

- Teachers should be able to view their management tasks in a table. Each task in a different row.
- Teachers should be able to add a management task, through an "add button". Pressing this button should lead to a form, and its completion results in a new management task being stored.
- Teachers should be able to update the status of their management tasks.
- Teachers should be able to edit and delete their tasks through buttons, which are located at the right end of the row.
- Admins should be able to perform all the aforementioned actions (view, edit and delete all the tasks) for all the teachers in the system.

Back-end

- The server should be able to query the DB for teachers' management activities
- The server should contain a request handler for getting the managements activities of one teacher (if Teacher access level is given) or all teachers (if Admin access level is given).
- In case the user has the Teacher access level - the activities will be retrieved based on the teacher's access token which is sent from the frontend to the backend.
- The server should contain a request handler for adding a management activity. The teacher responsible for the activity should be deduced based on the access token.
- The server should contain a request handler for editing management activities. Requests with teacher level access should only be able to edit activities of that teacher, while admin requests should be able to edit any management activity.

Database

- The DB should be able to store management activities of teachers.
- The database should allow for retrieving all management activities corresponding to a teacher.

2.1.3 Data overview & querying

Requirements regarding fetching data and presenting it, be it tabular overview or concise graphical views.

Front-end

- Teachers & Admins should be able to view statistics of the overall data through graphs (will be discussed in more detail with the product owner).
- Teachers & Admins should be able to filter the data, and create new statistics, by using the filter bar, which allows them to filter by quartile, activity type, teacher (for admins only) etc.
- Teachers & Admins should be able to download any of their overviews in an agreed upon format.

Back-end

- The server should be able to query the DB for teachers' activities.
- The server should contain a request handler for getting the activities of one of the or all teachers (based on the access level, as mentioned before).
- The handler for getting the activities should support filters such as quartile, activity type, teacher (admin access only, teachers only see their data) etc.

Database

- The database should allow for retrieving all the data corresponding to a teacher.

2.1.4 Notifications

Back-end

- The system should be able to notify teachers in the form of an email.
- The system should automatically send a notification before the start of each quartile, reminding teachers to fill in their data.
- The system should notify teachers when one of their supervision/management activities passed the due date.
- The system should be able to query the DB in order to get supervision activities that passed the due date.

2.2 Non-Functional Requirements

- When querying the database, the result should not be influenced by the number and type of concurrent queries.
- Data in the database must not be corrupted or affected by any query, even if a query fails.
- The system should be easy to maintain, meaning that adding changes in the future or fixing bugs should be easily done. This requires clean code and modular architecture.
- The system should be easy to manage by the admin, so that someone taking over the role of Dr. Bukhsh can also work with it.
- The average time the system needs to accomplish a user's task (uploading a form to the server, showing results, showing filtered results, etc.) must not take more than 5 seconds.
- The webapp's interface should be easy to understand, so users can learn to use it within a day.
- The pages' load time must be no more than 5 seconds for users that access the website.
- The server should log all data manipulation actions, in order to ensure non-repudiation.

- The server should be able to revert changes made to the data if needed, based on the manipulation logs in the DB.

Remark on Requirements

It should be noted that this document was prepared early on and many things have changed. Initially a waterfall development framework was assumed but both the client and our team realized that there were a lot of tweaks necessary to make the system a desirable one.

We still present the document as it sheds light on design decisions discussed in later chapters and it captures the essence of the system quite well. Also the prioritization was largely respected even though some requirements have changed.

3 Global design (general design of the system)

3.1 Overview of design

Designing a system, such as an EIS, involves a crucial process of architecture design that enables the compilation of requirements at a macro-level and lays the foundation for the system's structure. This process helps to establish the essential pillars of the system, which are critical for its success. After eliciting to the requirements mentioned by the clients (DMB managers), our team sat down together and planned the structure of the system, division to components and the communication between them, as shown in the figure below.

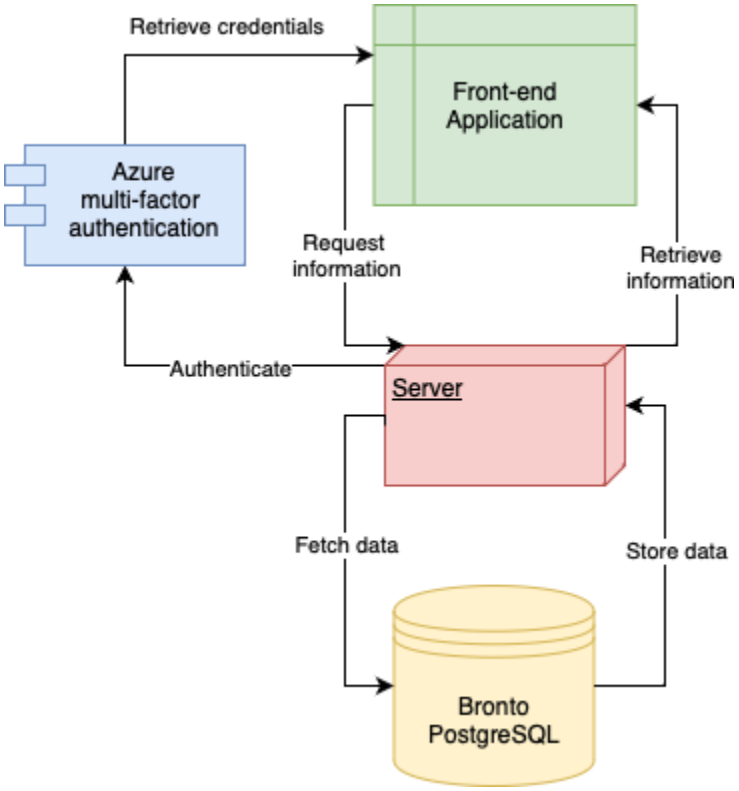

Figure 1: EIS Architecture Design

As you can see, we have designed the system in the following manner:

- **Front-End Application**

  The user interface of a system is the frontend, which acts as a platform for users to interact with the system. It is responsible for displaying information and data to the user. The frontend acts as the bridge between the user and the backend, receiving user input and sending appropriate requests to the backend for processing. In addition to that, the frontend handles user authentication and retrieves a token from Azure (through the backend), which it attaches to each request to the backend. Lastly, it is also in charge of showing to the user the appropriate information received as a response from the backend.

- **Azure multi-factor authentication**

  The main purpose of Azure multi-factor authentication (MFA) is to add an extra layer of security to user authentication and authorization. When a user sends a request from the frontend, the backend checks if the Authentication header exists in that request. In that header data such as a user's name,

email and unique identifier (oid) are stored. In the beginning of a user session on our application, the user is prompted with the Microsoft Azure login page. Azure MFA comes into action by requiring two kinds of authentication, a password (of their UT account) and a PIN code provided to the users' phones or emails. If everything goes well, the Authentication token is stored as a cookie and is passed to every request, to ensure proper authentication. In the case that the user does not provide correct credentials, Azure will prompt them to try again and eventually will block their account. The beauty of this is that we have completely outsourced this process to Microsoft and our system only gets feedback from Azure whenever a successful login has commenced. After acquiring the auth token, it is decrypted in the backend and based on the oid and the access level, stored in our DB, a user is identified as either a teacher or an admin. Based on the access level, different kinds of functionalities and overviews are provided to the user. This procedure helps to guarantee that the user is who they say they are and that they are not an unauthorized user attempting to get access to the system (or to specific routes). Once the user has been authorized, the backend may obtain and return the necessary information to the frontend. We prevent unauthorized access and protect sensitive data by introducing Azure MFA into the system. Using Azure for account management is also convenient for end users, as they will not need a separate account to use this application.

- **Server (backend)**

  The system's backend will run on a server that is installed on a virtual machine maintained by UT. Currently it is hosted locally. It communicates with the database and serves the Frontend. This is accomplished by a RESTful API that receives API requests from the Frontend, directs them to the relevant function, obtains the necessary information from the database, and sends a response back. In addition, the backend verifies the user's credentials from the Azure token attached to every API request, and proceeds accordingly, based on the user's access level. The server contains the **business logic**, where the advanced calculations and processes are taking place. The business logic is typically implemented in the backend because it involves sensitive data that should be kept secure and hidden from the end-users. Placing the business logic in the frontend can make it vulnerable to manipulation and security breaches. Additionally, implementing the business logic in the backend ensures that it can be reused by different frontends, such as web and mobile applications, without having to duplicate the code. It also provides a centralized location for managing and maintaining the business logic, which can make updates and bug fixes easier to implement.

- **Bronto PostgreSQL**

  The database server, which stores the data, retrieves and sends it to the backend.

3.2 Technology stack

The aforementioned requirements of the system implies on the kind of technologies needed to build the EIS system.

The system had to be built within a period of approximately 10 weeks, which led us to choose technologies we are familiar with, or technologies with a fast learning curve.

For that reason, we decided to use the following:

1. **Frontend - Angular (TypeScript)**

   The user interface of the system is a website, due to the convenience of access and data representation. We have two members in our team with a prior experience in Frontend development using Angular. Angular enables an easy scaling-up and improvements to the website, thanks to the architecture of the framework itself. Angular is a good choice of use when there is a need to build a (potentially) large system. Based on the requirements, and the possible use of EIS, we decided that Angular is a good fit to be the frontend technology. Angular also uses Typescript, which allowed us to create different types of Models to represent the data in the database, and work with it more safely (considering that the data should be sent/received in a particular format to/from the server).

2. **Frontend Authentication mechanism - Microsoft Azure**

   Users can log in and sign-up to EIS by authenticating their UT account. This process is done using the cloud service of Microsoft (Azure), therefore we had no choice in that aspect. We configured the relevant details in Azure, and combined them with imported modules from @azure/msal-angular we have built the authentication system of the website.

3. **Backend - Flask (Python)**

   Flask is a framework that could be used to easily build a RESTful API using Python. Due to our time limitation, and the need to reach the MVP in a few weeks, we chose to use Flask, which is relatively easy to learn. This decision helped us to increase our system quickly and reach our goals within the short time-frame.

4. **Database - Postgresql**

   While designing the database of the system, we realized that the data within the system is very related to one another: teachers could be also supervisors, Courses and Modules have a strong relation, and more. In other words - there is a strong need of a Relational-Database, where we could apply this relations in a logical way, and then easily manipulate the database when needed. Hence the choice of using SQL as the programming language to interact with our databse. The choice of using PostgreSQL stems from the fact that UT has already PostgreSQL servers which could be used and deployed more easily, with the support of the UT.

5. **Backend & Database integration - Psycopg2**

   Psycopg2 is an easy-to-use library in Python to interact between the server (in Flask framework) and the database itself. It enables us to write SQL queries in our code in an understandable and easy-to-read manner.

6. **Visual Statistics - ApexCharts**

   We were looking for a free and easy-to-use library to visualize the statistics in the form of various charts (pie-charts, bar-charts and more). We found the library called ng-apexcharts, which gave us plenty of charts to use. All we had to do is to adjust the required data into a specific format (of ApexCharts) in order to represent aesthetically the necessary statistics about teachers' activities.

3.3 Methodologies

In our project, we have decided to use several methodologies such as Waterfall & Agile (combined), Scrum, and Security by Design. In this section, we will explain the reasoning behind following those methodologies.

- **Waterfall**

  Waterfall is a traditional software development methodology that follows a linear and sequential approach to software development. In contrast to the Agile methodology, the Waterfall approach emphasizes a structured and organized approach to software development, where each stage of the development cycle is completed before moving on to the next. We followed this methodology by first planning the system, then designed it, developed, and eventually moved to the testing and maintenance phases. We did not follow the Waterfall approach entirely, due to the very flexible nature of this project. However, this methodology helped us structuring and organizing the project slowly and safely, making sure that all the teammates are involved in the process and understand the required steps to be taken. We provided a clear and well-defined path for the system's development. The initial UI design of the system was pretty general, but we made sure with the client that the idea of the project is clear for us and for them. On the other hand, the database design was done more precisely and carefully, taking all the possible data problems in mind. We wanted to make sure that the complex database design will be as fit as possible to the current and future requirements of the clients. Therefore we see the database design as more strict, thus followed by the Waterfall's sequential approach.

- **Agile & Scrum**

  Due to the inflexible nature of the Waterfall approach, we had to use the Agile methodology in the development phase itself. Agile is an iterative and flexible approach that allows for changes to be made throughout the development process. This means that the project can adapt to changing requirements and stakeholder feedback. We have followed Agile by breaking down the development process into smaller, manageable sprints (approx. 2 weeks each) with specific goals and deadlines. We have also re-prioritized features based on stakeholder feedback, allowing for changes to be made throughout the development process. Requirements had been added or modified based on our progress and after getting feedback from the clients.

  In addition, Scrum is an Agile methodology that emphasizes team collaboration and communication. This helps ensure that everyone is on the same page and that the project is progressing smoothly. We communicated among the team on a daily basis, sharing the progress, the issues that were raised, and the modifications of requirements. This agility contributed to our fast development and led to a better MVP, with features that help users even more than the system we had in mind (and the clients) in our first couple of meetings.

  All of that implies the flexible manner of this project, hence our choice was to combine Agile and Scrum when it comes to the development phase and requirements elicitation.

- **Kanban**

  Kanban is a project management methodology that focuses on visualizing and optimizing workflow to improve efficiency and quality. We followed this approach by using Trello - A visual board with columns indicating each phase of a process and cards representing the tasks to be done. As tasks are accomplished, they are moved to the next column on the board until they reach the last column, which represents completion. This allowed out team to simply track our progress and identify process bottlenecks.

- **Security By Design**

  Security by design is a method of incorporating security issues into all stages of the development process. This helps to guarantee that the system is secure from beginning to end, reducing the possibility of security breaches.

  In our project, we followed this methodology by taking into account any security issue that might arise in any of the system's components. We used prepared SQL-statements to prevent SQL injection, we also ensured that the GIT repository is clean from any configuration or other private credentials.

  In addition, we perform the filtering in the backend so that users will not be able to see the data format through the web-console.

  We also followed the practice of never trusting the frontend, which means that the backend validates the data received from the frontend, including the user's access level, before proceeding with any action.

  Lastly, we outsourced the authentication procedure to Microsoft (using Azure MFA), to guarantee more secured user management.

4 Design choices (detailed design of the system)

4.1 Front-end application

The front-end single page application (SPA) design began shortly after the requirements elicitation. It was clear that the app needed to provide a convenient way for teachers to accomplish tasks, while also working around data limitations.

Early on, it was established that there are 3 types of activities (education, supervision, management), and each type required different data to be stored. What that meant was that we had no way of nicely putting all of a teacher's activities in one single table, and that we would instead have to separate the activities by type. We therefore designed three pages, each containing a list of the teacher's activities of that type.

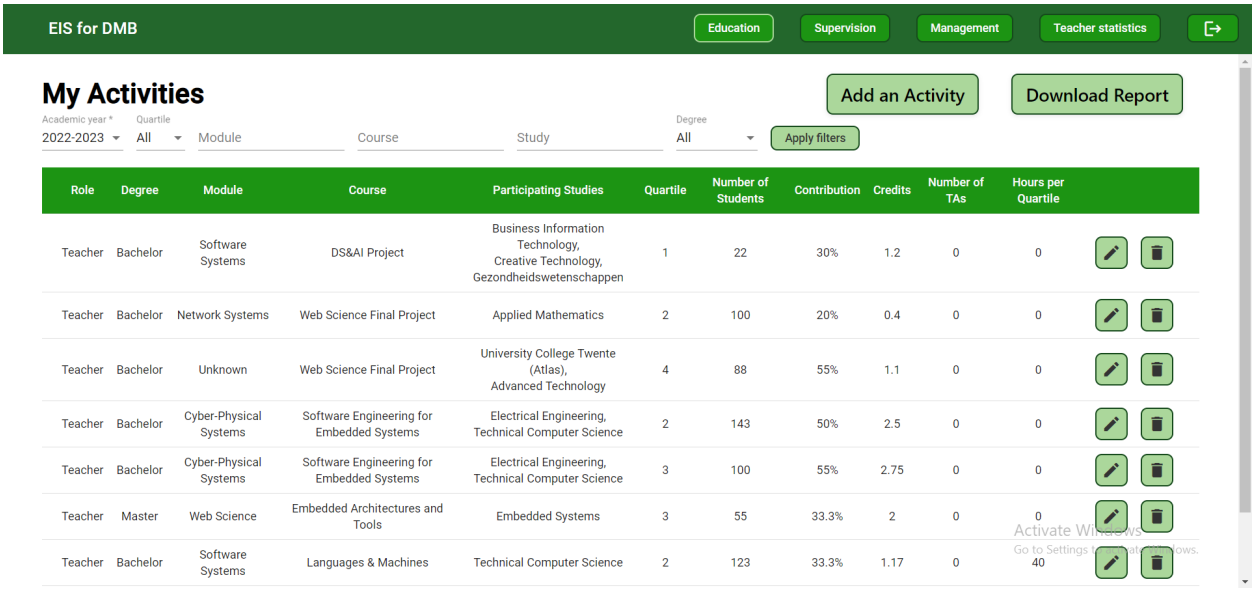

Figure 2: The education activities table page

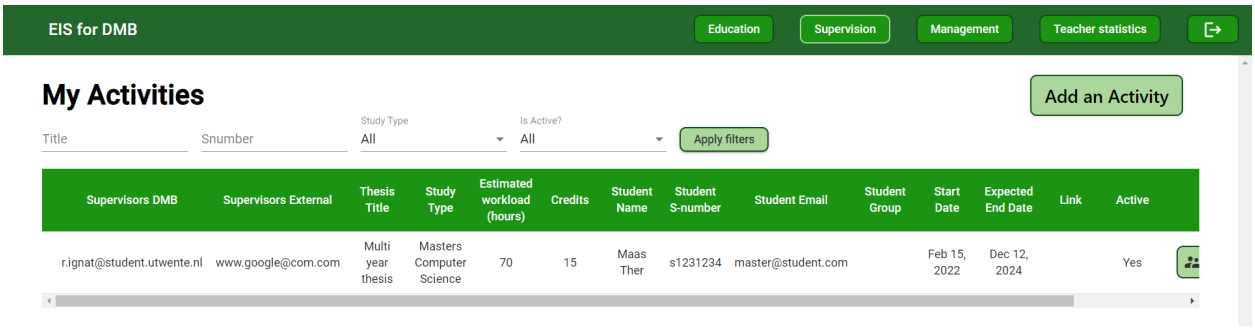

Figure 3: The supervision activities table page

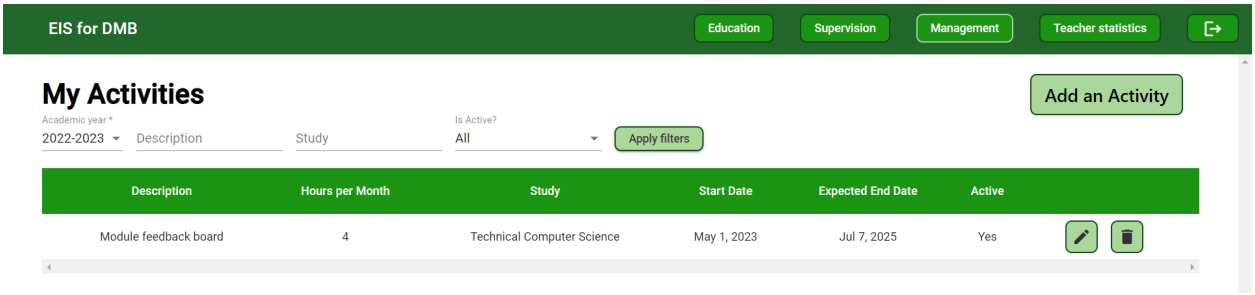

Figure 4: The management activities table page

On each page, a button was placed that would open up a form for inputting a new activity. While thinking about the form design, we took inspiration from the already existing Google forms that were used for gathering activity information. By looking at those, we came up with the idea of making the form dynamic - so that previous questions would influence what fields you see next. For that reason, we split the forms into steps, with each step collecting part of the data of an activity and influencing what is displayed in the following steps.

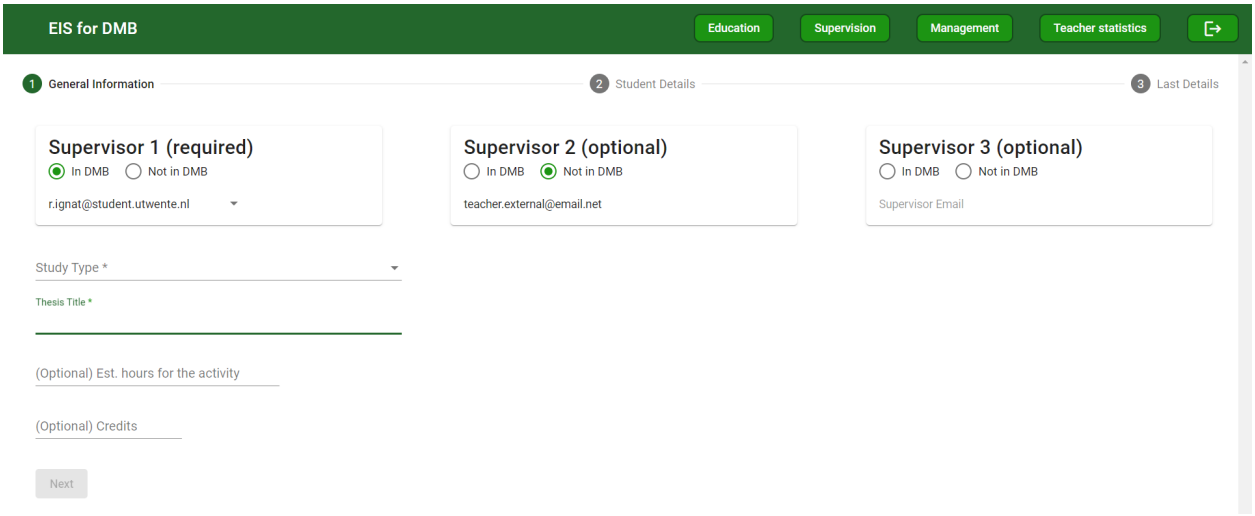

Figure 5: The first step for submitting a new supervision activity

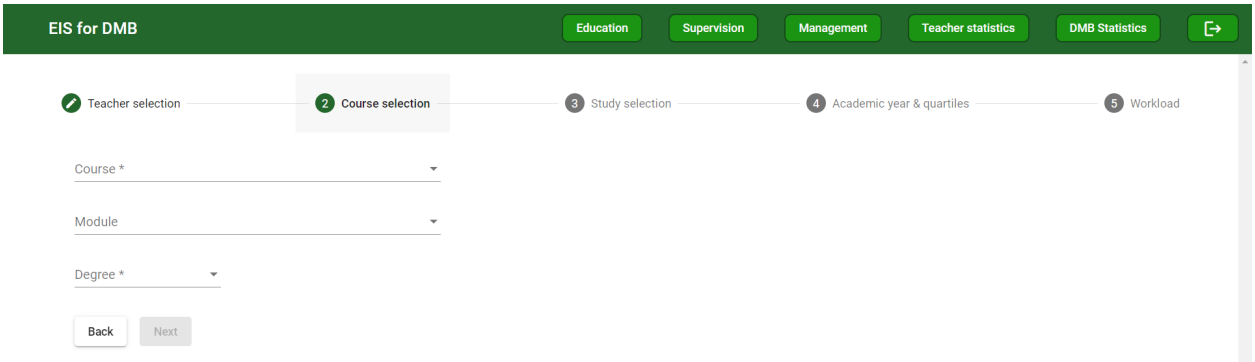

Figure 6: The second step of the education form, allowing users to select a degree

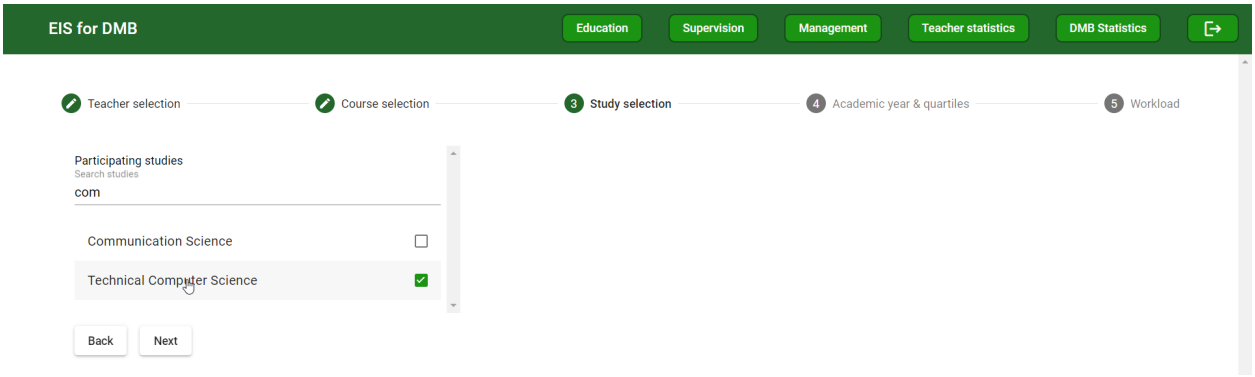

Figure 7: The third step of the education form, allowing users to select participating studies from a list, based on the previously selected degree

Another requirement that came up was that users should be able to edit certain fields of an activity. For achieving this, we considered 2 approaches. One was to have the edit button send the user to a pre-filled form that they could then change, while the other approach was to simply make the rows in the table editable. What we ended up doing is close to the second approach. When a user wants to edit an activity, they can click on the edit buttons, which transforms that table row's cells into editable fields. Once the data is modified, the action can be confirmed, in which case the update would be sent to the server. A unique situation was when a user would want to edit what supervisors are assigned to one of their activities. We found it impractical to make this edit in-table, so we opted for having a different button that opens a popup. This popup contains a small pre-filled form, in which users can select the new supervisor values for the activity.

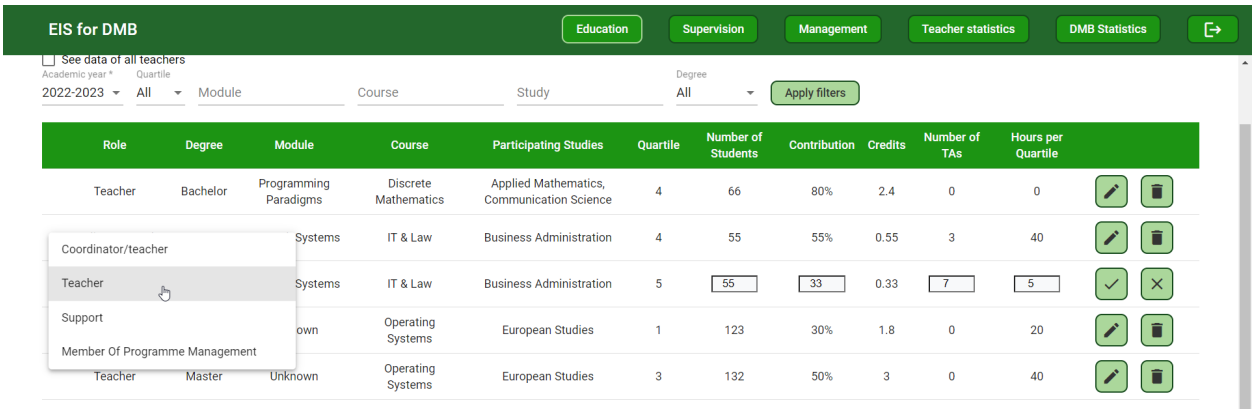

Figure 8: An education activity being edited in the table

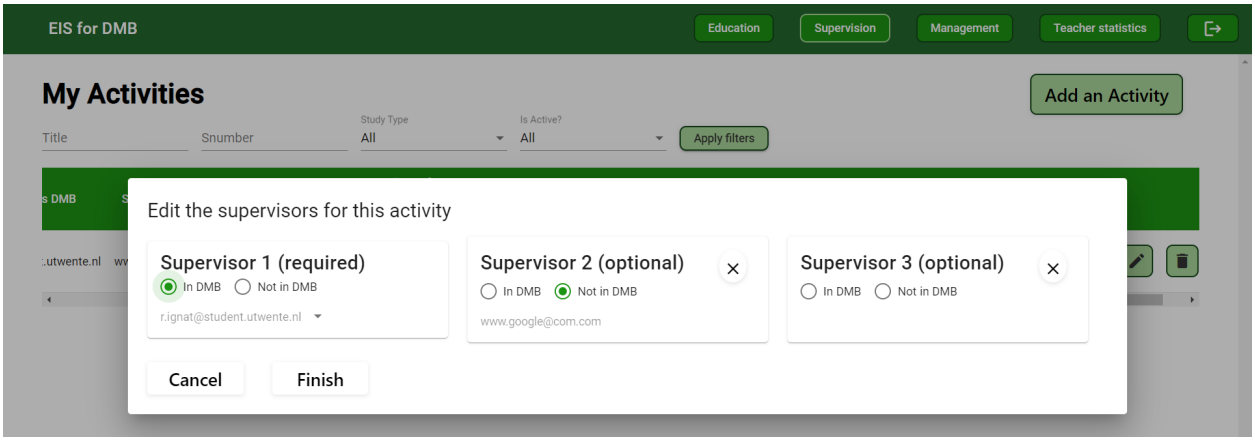

Figure 9: The popup window for editing the supervisors of an activity

One of the main reasons of this application is to have a convenient overview of DMB members' workload. Therefore, we included another separate page for statistics of the entire group. In this page, we strived to cover all the useful stats of the group, such as how busy the group is in each quartile, who does a lot of work during a certain part of the year, or how many credits each teacher gets in a year. For the sake of visualization, we also included suggestive graphs - pie charts, bar charts and line charts - in order to neatly display the data. While the types of charts that should be used was debated with stakeholders, we decided it was best to make everyone happy by implementing all three types, and by making them minimizable. Of course, because of the many differences between the data of activities of different types, we were once again forced to split the stats page into 3 different tabs, one for each type. This does come with benefits though, as these pages already display a lot of data, and having them split into 3 prevents them from being overwhelming.

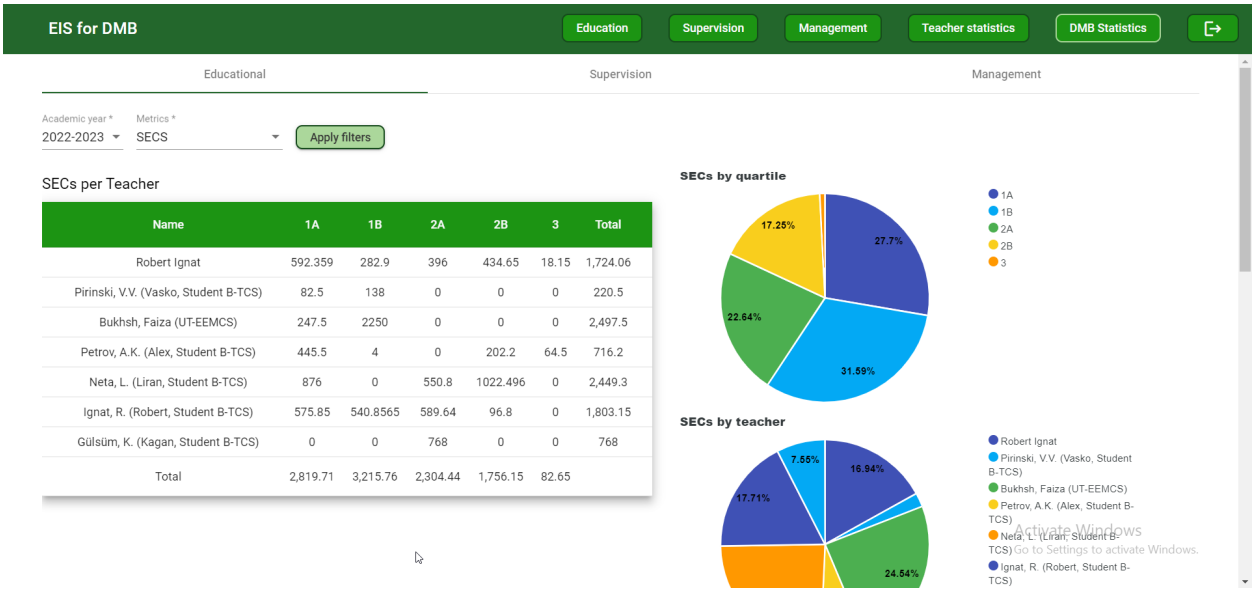

Figure 10: The education stats page, showing the SECs (Credits times number of students) of each teacher for each quartile, together with total stats

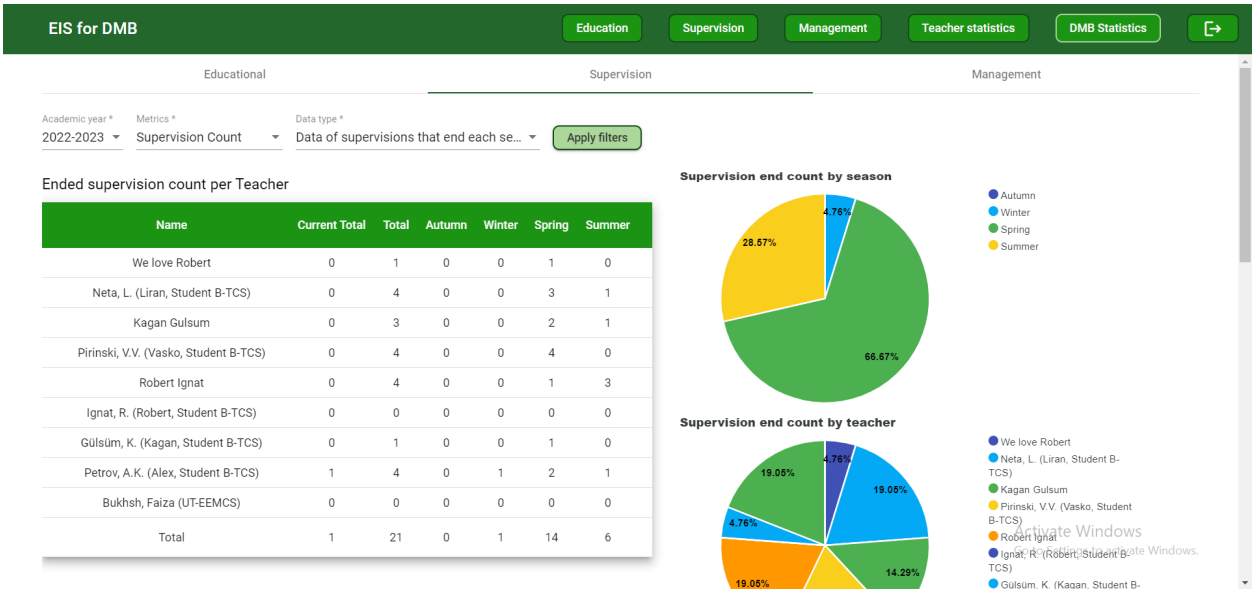

Figure 11: The supervision stats page, showing how many superivision activities end for each teacher each season, together with total stats

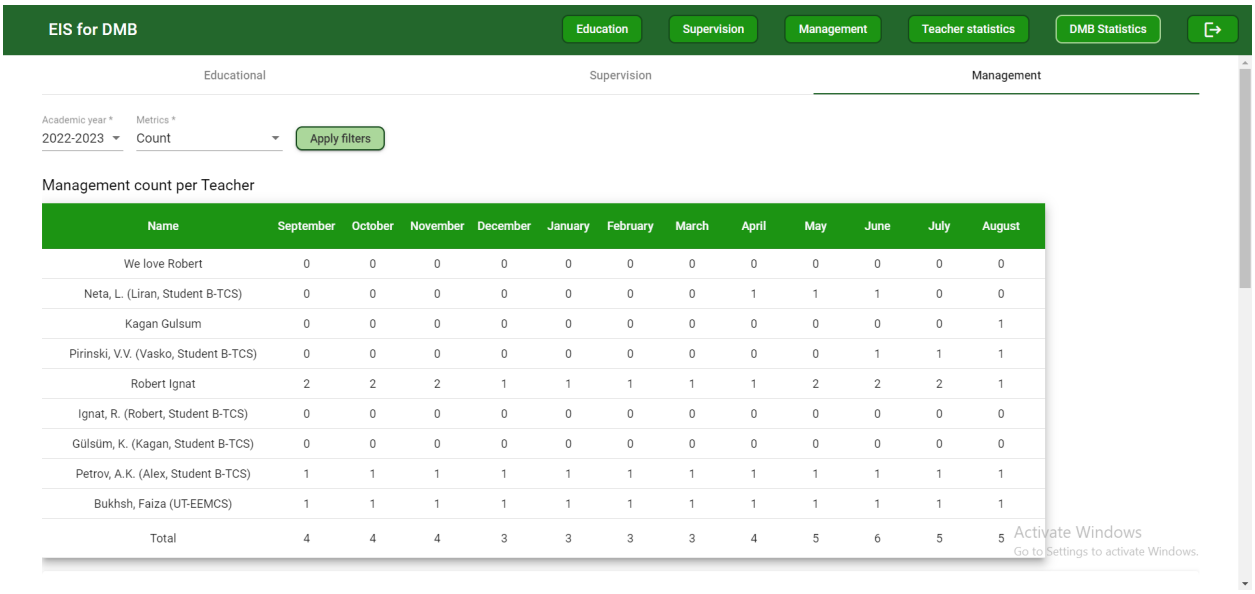

Figure 12: The management stats page, showing how many management activities each teacher has in each month (graphs available on scroll)

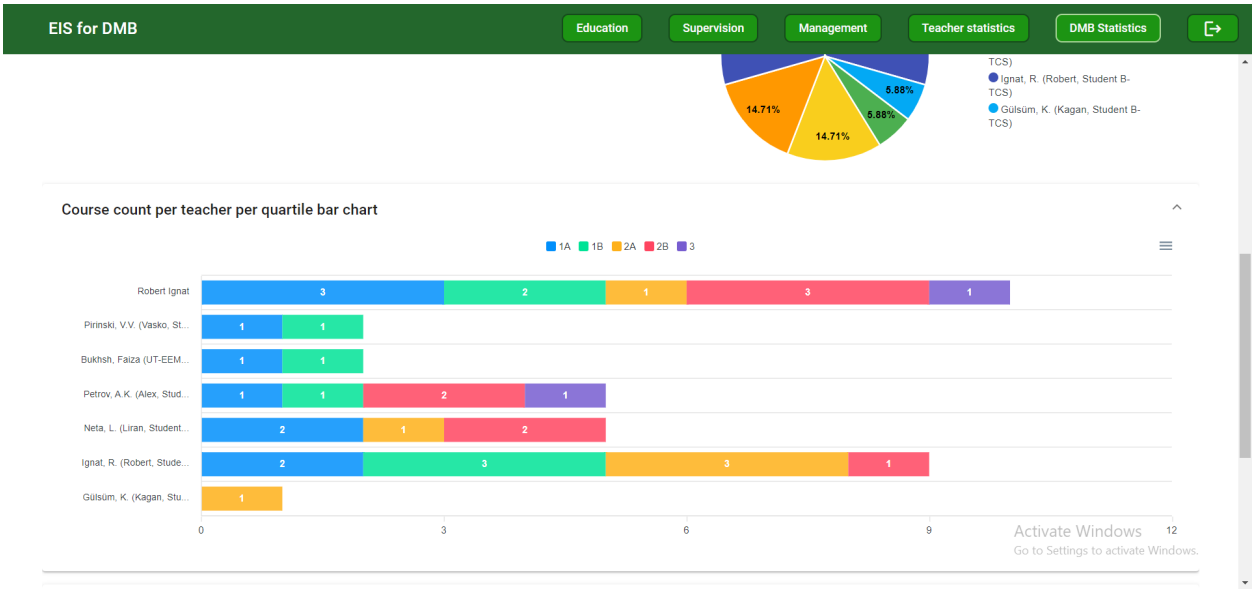

Figure 13: Example graph: Bar chart showing for each teacher, how many education activities they have each quartile

One issue that emerged was that, because the above-described stats page was gated behind admin access, regular users did not have any way to conveniently see all of their stats. In addition to this, while the DMB stats page also presented all the data of each member, it proved a bit tedious to actually extract the information about a specific teacher. This was a problem, because stakeholders expressed their desire to have an easy way of seeing all the stats of one teacher. That being the case, at a later point in the project we also managed to implement a teacher overview page, which would show the same types of stats as the DMB stats page, but only for one teacher and in a more compact manner.

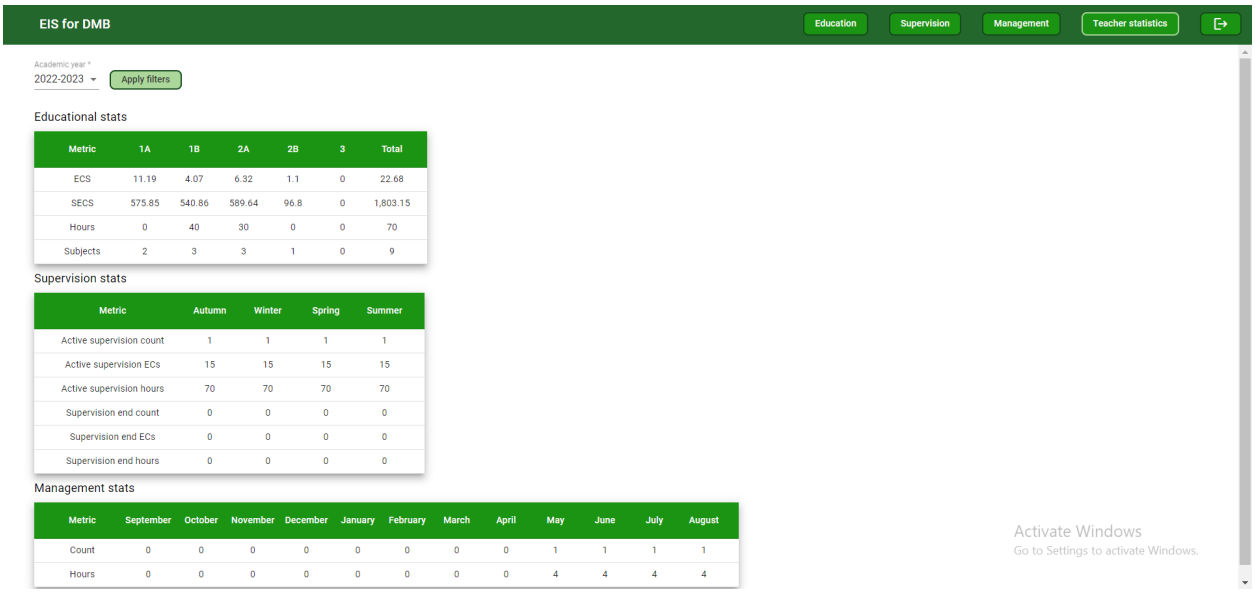

Figure 14: The teacher overview page, showing all metrics for all 3 types of activities for the logged in teacher

The difference in access levels between admins and regular users (which we often simply referred to as

teachers) was noticed at an early point in the design process. While the exact limits of what a teacher could do were subject to debate throughout the module, it was clear that admins should have access to additional features, and that the interface should reflect that access. It was easy to see how implementing completely different pages for admins was not practical, given the time constraints, so when designing each page, we also thought about what both access level users would see. For example, the admin sees the same table of their educational activities as a teacher, but they have an additional toggle that makes it so that they see the activities of all teachers, instead of just their own. In another instance, the teacher overview contains a filter to select the teacher in the admin view, but regular users do not have that choice - the page directly shows their own data.

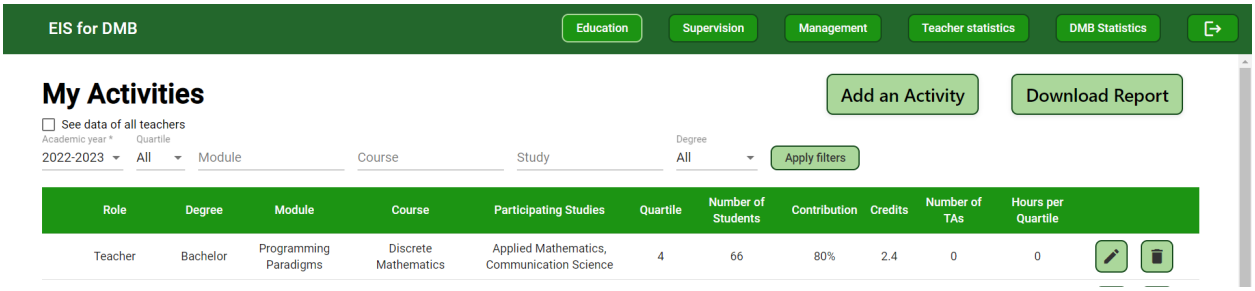

Figure 15: Admins have a checkbox that allows them to make the tables show the activities of all teachers, instead of just their own. Also, only they have the button in the header for DMB stats

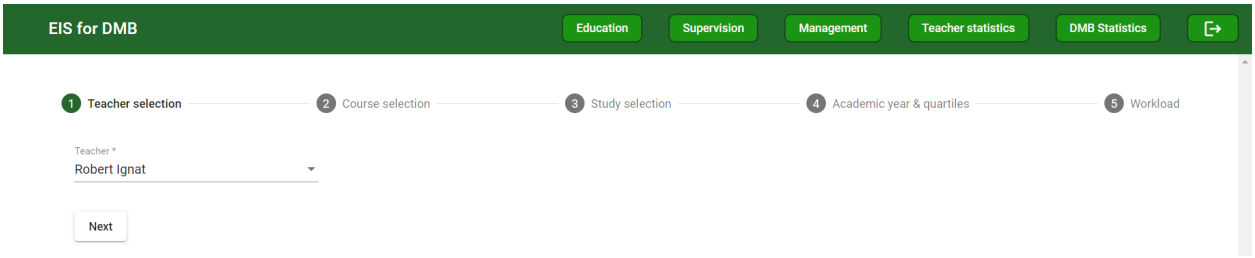

Figure 16: The first step in the education form for admins - this teacher selection step does not appear to regular users

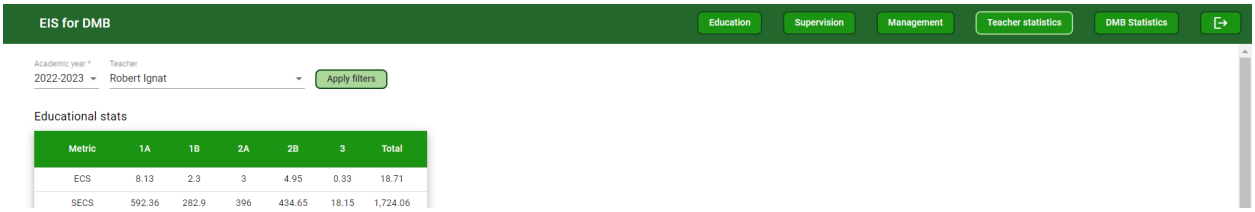

Figure 17: For admins, the teacher overview (showcased earlier) also contains a filter which allows the admin to show the data of a different teacher

4.2 Database

The design of the database took multiple iterations throughout the span of 8 weeks. It was changed due to factors such as stakeholder wishes, optimisation, proper design methodology and completeness.

The first design was completed in the very first stage of our application design, after we gathered the initial list of requirements from the clients.

This section explains the final design of the database with its associated tables and fields, then goes through the different iteration of the design process that led to the final design, highlighting the differences introduced with each version. Furthermore, an explanation of the encountered difficulties is provided and the section is concluded with storage details and a few words on maintenance and maintainability.

4.2.1 Final design

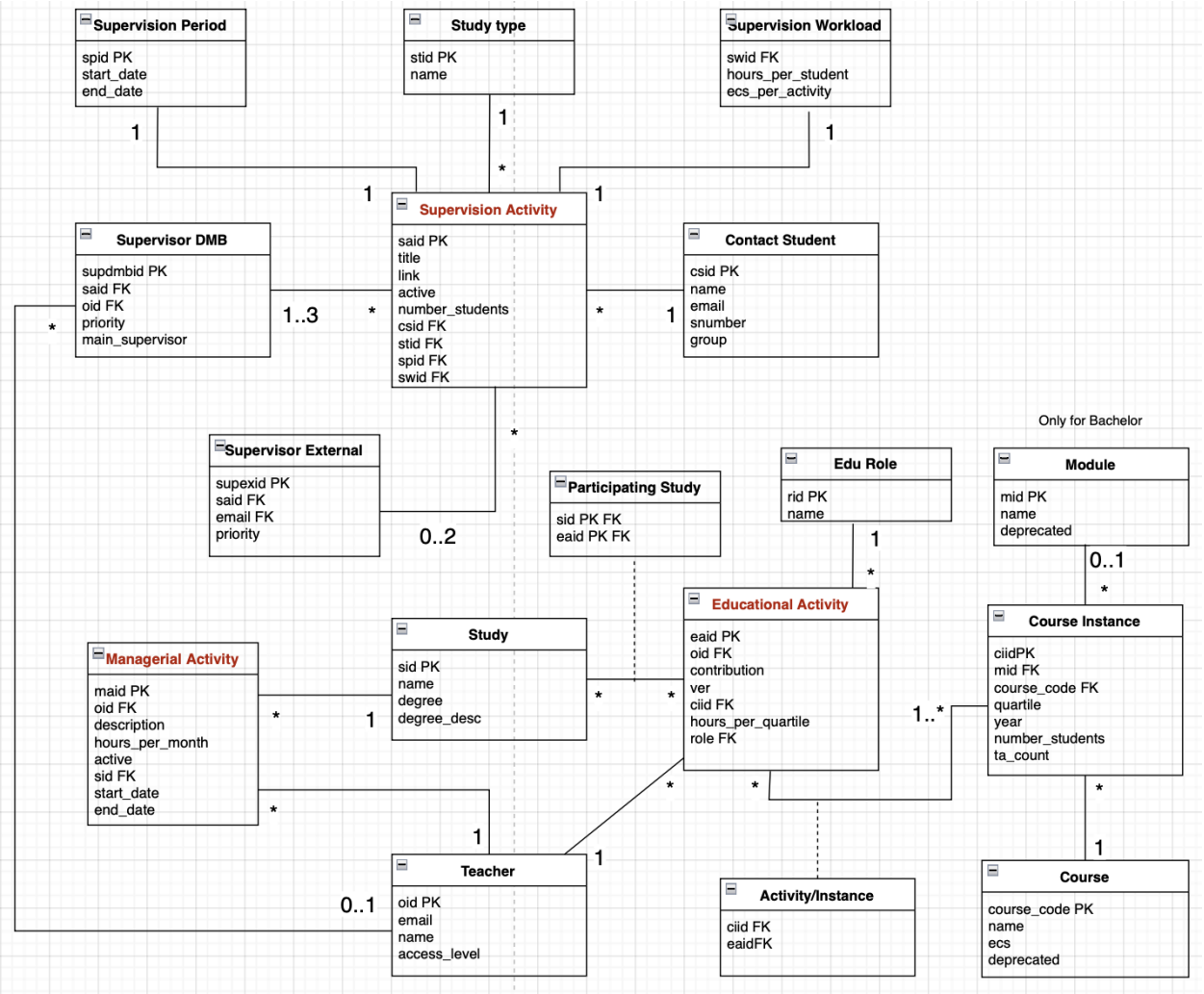

Figure 18: Database Schema Final

The main tables that define our database are the 'Educational Activity', 'Supervision activity' and 'Managerial activity' table. They represent the 3 main functionalities that were first requested by the clients, to store and show information about educational, supervision and managerial activities.

The main fields of the 'Educational Activity' table are a unique ID, an oid that references the teacher who is tasked with this activity, the contribution percentage of this teacher in the activity, an id to reference a

course instance, the hours per quartile that the teacher will put in the task and his role. The oid of the DMB teacher is generated from Azure and is registered in the database upon the very first login of that teacher. Since the oid is a reference to the 'Teacher' table, there information such as a teacher's email, name and access level are stored. The access level is used to distinguish normal teacher accounts from admin ones. Furthermore, the 'Edu Role' table stores all the roles that a teacher can take. The oid and the course instance ID have a unique constraint in the DB, which means that a course instance is associated with just a single teacher for a particular activity. There is also a check for the 'contribution' field that guarantees that a value would be between 0.0 and 100.0 which correctly represents the percentage.

An integral part of the functionality of the educational activity is the 'Course Instance' table. It is defined by the 'Course' and 'Module' tables and unique traits such as a quartile, year, number of students and a TA count. The combination of a year, quartile and course code is unique and this, combined with an educational activity, defines a unique row in the educational activity overview table.

The 'Course' table has a unique identifier, defined by the course code. It is also represented by a name, ECS for this course and a boolean that indicates if that course is deprecated. A course is marked as deprecated if it is no longer present in the university database. Usually, the overview will only show educational activities for non-deprecated courses and modules, since the boolean is also present in the 'Module' table. It is accompanied by a unique ID and a name for the module.

The course instance is usually a combination between a course and a module, but this is only true for a bachelor task. All other tasks, for example related to a master, are defined by a course alone.

An educational activity is associated with at least one participating study. The 'Participating Study' table defines the combination of a 'Study' table entry and an educational activity. The studies are recognised by their name and the degree they are taught in, for example, Technical Computer Science with a bachelor degree.

The 'Supervision Activity' table has a unique ID, title of the project, an optional web link, a boolean to indicate if this task is active, the number of students that will be supervised and multiple other ID's that reference other traits of a supervision task.

The student who will be primarily associated with the supervision task is stored in the 'Contact Student' table. Since it is possible for that student to be in a group, this group is only written as a field in this table, but what is important for the supervisors is the name, email and snumber of that contact student.

The supervisors in question can be part of the DMB group, represented by the 'Supervisor DMB' table or come from an external group, represented by the 'Supervisor External' table. Both of these tables have a unique ID, a reference ID to the supervision activity and a priority. Since a supervision task can have up to 3 supervisors, the priority integer shows if this supervisor is first, second or third. The first supervisor is mandatory, the last 2 are optional. From all the DMB supervisors, the one with highest priority will be the 'main' supervisor, which is stored in the respective table. Only the main supervisor for an activity can update or delete that activity. Supervisors from the DMB group have a reference to the existing teachers in the 'Teacher' table while the external ones are defined by an email that the user provides.

Supervision activities have different study types, stored in the 'Study type' table. This table should not be confused with the 'Study' table as it is only related with a supervision task. It only has a name and a unique ID and represents data such as 'Bachelor TCS', 'Bachelor BIT' but also 'Internship TCS', 'Internship BIT' and others.

A supervision activity is also defined by 'Supervision period' and 'Supervision workload'. The period stores the start date and end date for a particular supervision activity, while the workload defines the estimated hours that a student will invest in an activity and the corresponding ECS. A check on the start and end date guarantees that a supervision activity cannot exist with an end date coming before the start date.

The last task that is important for our clients considers the 'Managerial Activity' table. It is structurally and functionally the least complicated of the three. It has an oid, which as already mentioned, references the teacher who is employed with this task, a description of the activity, the hours per month, if that activity is active, the study which it is related to and a start and end date.

4.2.2 Iterations

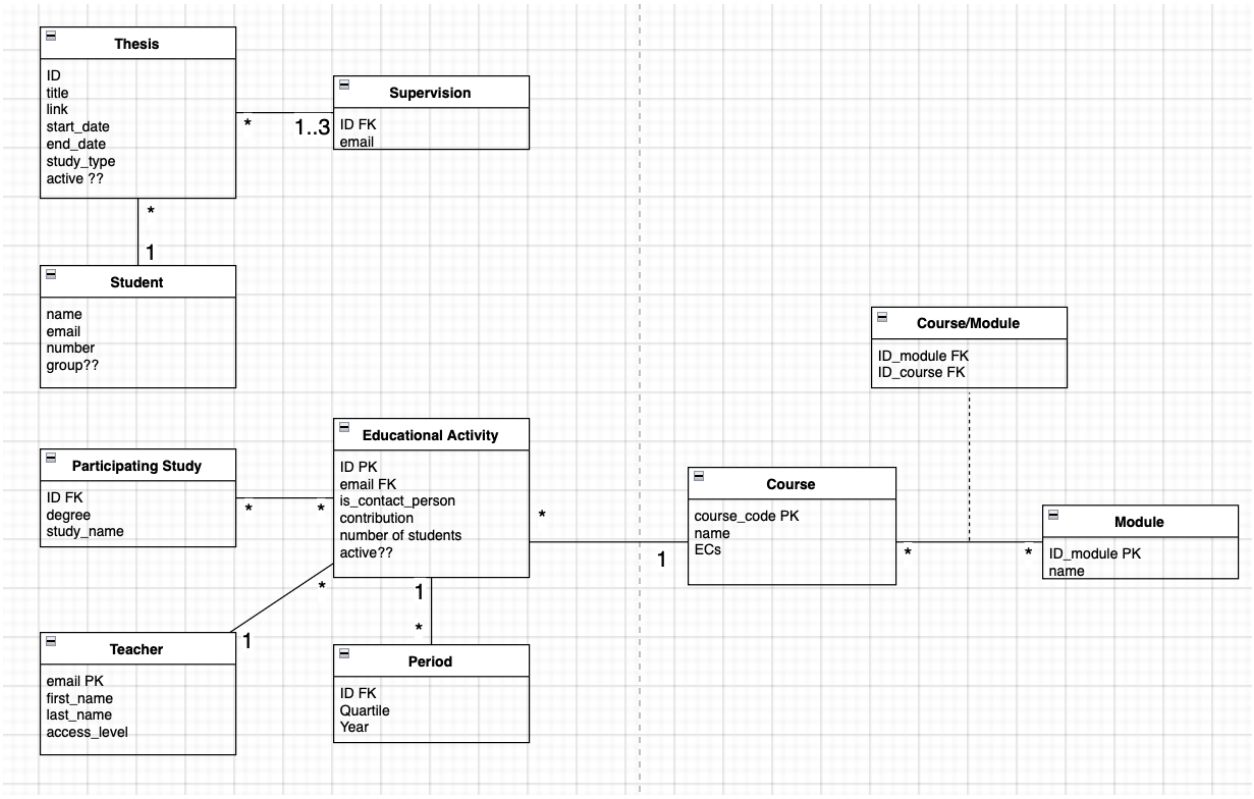

Figure 19: Database Schema v1.5

One of the very first designs of the database 19 is represented by the image above. It is version 1.5 as the initial draft was drawn on a white board and outlined a rough sketch of what the system should represent.

One of the main differences compared to the final design of the DB is the Course / Module structure related to the Educational Activity. As this was the earliest drawn design, courses were considered a mandatory part of an educational activity, while a module could be combined with a course in an association table where the ids of a particular module and its associated class would be paired together. But this 'Course/Module' table would not exist without a query join or be connected with any other table. There was also the concept of 'Period' which would identify one educational activity with a quartile and year. This was done to ensure the proper normal form of the DB so that there are no duplicate rows in the 'Educational Activity' table with just quartiles being different. The 'Period' table was later combined with the 'Course/Module' table into the 'Course Instance'. Also fields such as the number of students and the later introduced TA count are stored there and not directly in the 'Educational Activity' to adhere to the normal form. What is more, the 'Edu Role' table does not exists, as it was not considered in the requirements on that stage. The 'iscontactperson' field was added as the initial designs of the DB took great inspiration from the Google Forms that were used for gathering the data. Eventually it was considered not important for the table.

Another difference with the final design of the system is the structure of the 'Supervision Activity' table. Besides the different naming, 'Thesis' being the 'Supervision activity', 'Student' being the 'Contact Student' and 'Supervision' being the predecessor of 'Supervisor DMB' and 'Supervisor External' there are also some functional elements missing. The supervision table was later split into two tables as the situation with the supervisors proved more difficult that first envisioned. More on this topic in the 'Difficulties' section 4.2.3. The 'Study type' table, as well as the 'Supervision period' and 'Supervision workload' did not exist as on this early stage the supervision functionality was not the first priority and the team did not consider that the number of columns would increase so much, so most of the information was kept in just one table.

Furthermore, the supervising teacher was not linked in any way to the 'Teacher' table, meaning that no validation could be done for DMB supervisors and probably repetition of data would exist with this design.

It is also obvious that a lot of features were still not clear, from all of the question marks in the diagram.

And finally, nothing about the managerial activity exists because at that stage the team had not gathered enough requirements from the stakeholders.

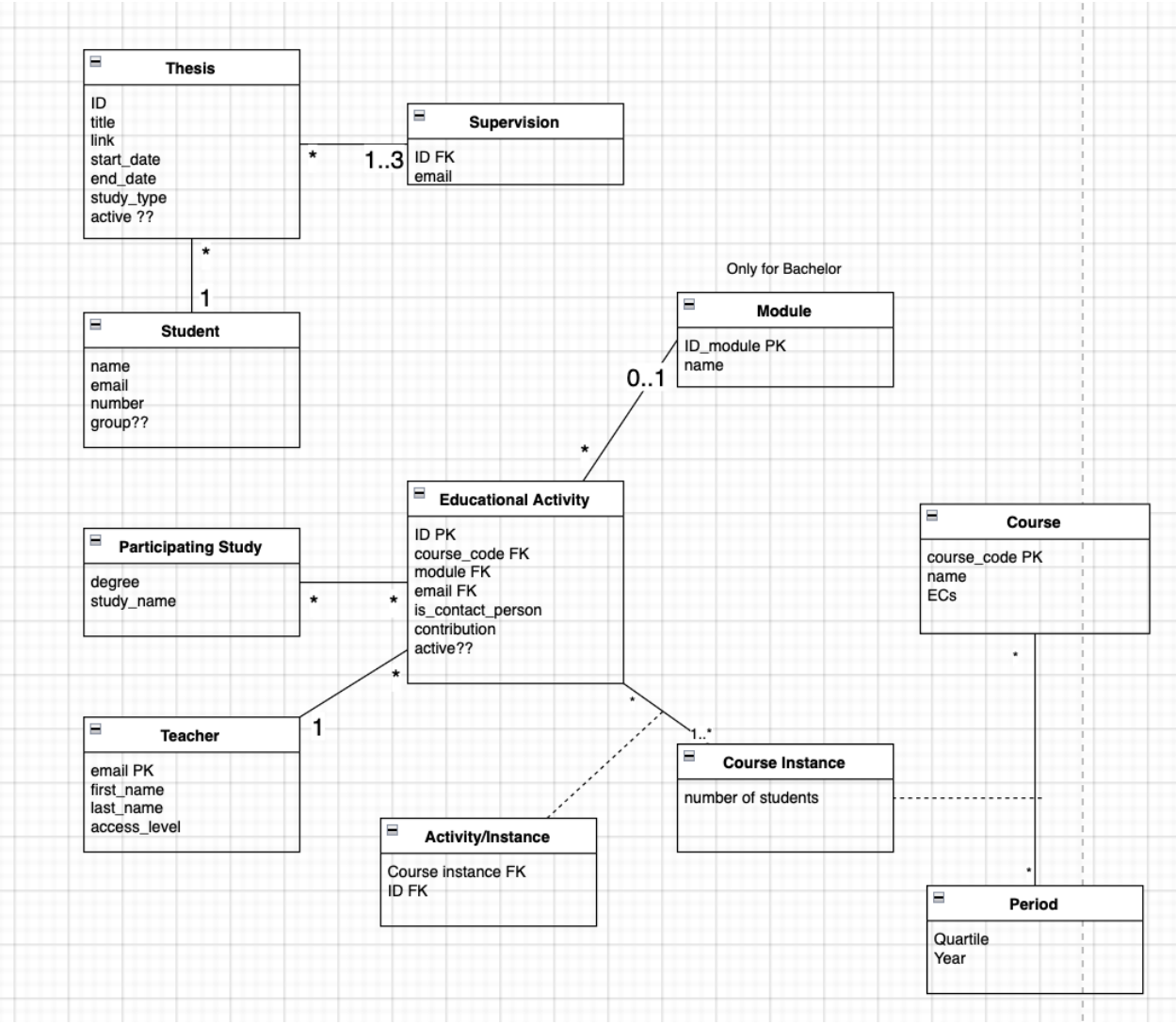

Figure 20: Database Schema v2.0

The second version of the design 20 took some steps forward into the development of the 'Course instance' table. But here, the 'Module' table was completely isolated as it was considered too unrelated with the courses. On the other hand the courses would be combined with the period table, and this combination would be identified as a 'Course instance'. This design was refactored in the next iteration as it would lead to a lot of null module foreign keys in the educational activity for all non-bachelor activities.

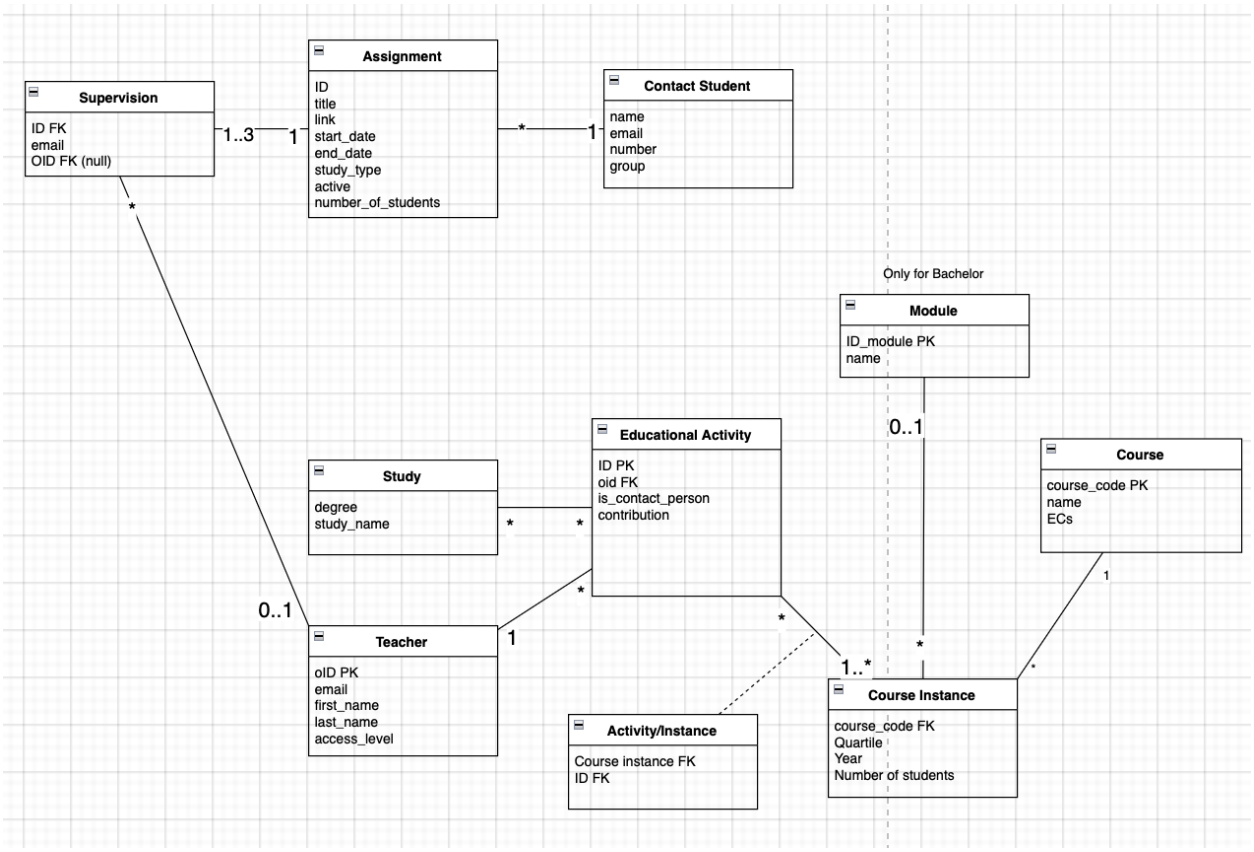

Figure 21: Database Schema v3.0

The third version of the design 21 has the almost complete up to date design of the 'Course instance' as it combines the module as well and has all the important information to isolate the educational activity to not have too many repetitive rows. For each educational activity, the association table 'Activity/Instance' would pair it with unique information such as a year and quartile, course, module and etc.

What is more, the 'Supervision' table would at that point be connected with the 'Teacher' to ensure validation with DMB supervisors. The referencing oid would be nullable to enable external supervisors to also be recorded in this table.

4.2.3 Difficulties

There was uncertainty around the design of the Modules and Courses since the team was not sure in how much detail we need to go to ensure the desired functionality.

The simplest version to implement would be to not employ any validation of the data input for Modules and Courses and just gather plain text information and store it as such.

The most difficult version would consider full input validation with connection between modules and courses. This would mean that after selecting a course, the user will be presented with modules that are only given in that specific course. Further, there would be periodic checks with Osiris to make sure the data is valid. This would require the storage of all modules and courses given by the DMB and the relation between them. What is more, this would have to be validated with quartiles and academic years.

The middle ground between these solutions is to have a database of all courses and separately, all modules, that is periodically maintained. Functionally, this means that a user can select only select modules and courses that are presented by the system. This ensures input validation to some extend as there will not be any misspelled or non-existing modules and courses. But still, it is up to the teacher who is inserting the activity to be sure the course specified is part of the chosen module.

Another difficulty that led to a lot of discussions within the team considered the supervisor situation. Initially, a supervisor would only be stored with an email that is provided from the user, without any validation. Then we decided to also relate the supervisor table with the teacher one, so that DMB supervisors can be pre-fetched and offered to the user as different options and not use plain text. This lead to the question if we should have entries in the table with null values or not. Since there would be both an oid and an email, one of the ideas was to store the oid if it considers a DMB supervisor and keep the email null or vice versa for external supervisors. Another option was to always fill the email, but for a DMB teacher that email would be fetched in the backed from the teacher table. The first option proved functionally problematic when updating the data and the second would have rows with data that have the same functional reference, which led to bad design. Finally, it was decided that it is a generally bad design choice to store two functionally different data sets in one table. One reason being the annoying amount of null values and also the added fact that a supervisor from the DMB group can also be a main supervisor, which adds another field that does not concern external supervisors. Thus, the final design includes two tables for each supervisor category, each referencing a supervision activity. This leads to a normally correct, efficient and easily traceable design and data flow.

4.2.4 Storage

All historic data of the EIS will be kept for security and overview reasons. This is due to the fact that very little personal information is stored, mainly names and emails. Upon request, data can be deleted, as to be compliant with the GDPR. In the event of hindered performance, very old and irrelevant data can be deleted as well. Most of the data about teacher tasks has 'active' and/or 'deprecated' flags that can be used to not overflow the user with activities that are too old, while they still exist and can be used if needed.

4.2.5 Maintenance

As the correctness of this system depends on the up-to-date information about modules, courses and activities, the timely maintainability of the system is of great importance. In the context of teachers, new teachers are added automatically when a new account is logged in the system. But modules and courses, for example, are left to the management of the technical staff and admins of the DMB group. They need to have access to the bronto server and understand how to edit the data there. In future versions of the system this could be made more user friendly and to be possible to edit from the system UI itself, but then again, it is up to the DMB teachers to keep track of having the latest information.

Furthermore, if the data is corrupted in any way, even though that should not happen, it is up to the technical staff to take care of that data in the database itself.

4.3 Back-end

As for the back-end, it has not deviated that much from what it is supposed to be. It delivers data between the database and the front-end, and, of course, handles its translation, since what the client sees is very different from how we actually store that data. First, we will explain how it is designed in general, and then go into specific issues and how we chose to resolve them.

So, our back-end has the typical API calls for fetching, updating and deleting data. For fetching we have also added support for the filters we want the users to be able to interact with in the application. It was a consideration to do that filtering in the front-end to simplify the system, but we decided to go with the standard approach of letting the back-end do this work. On the one hand, this is necessary because of security concerns - for example a non-admin user can easily see all sensitive data that was sent through the network even if it is not directly visible in the app. On the other hand, we considered this choice as much more logical, since the back-end can handle the data faster and with less computational work for the client.

There are also functionalities for providing statistical data about the three types of activities, which is calculated according to our client's wishes. This leads to the choices we made along the way.

4.3.1 Time intervals

First, there is the problem of how to separate the data in periods. We initially wanted to keep things consistent by just using quartiles, but that only made sense for educational activities since the teachers fill that data in. However, for supervisional ones we were asked to use seasons, and for management tasks - the hours per month metric, which means dividing those activities on a monthly basis makes more sense.

So, now we have a different time interval for each type of activity. This gives our stakeholders the opportunity to see all options and specify which ones are preferred for which situation, while also satisfying the current requirements. And since we already have the implementation done, these changes can be applied almost immediately.

4.3.2 Academic vs Calendar year

Another question of a similar type is if we should use academic or calendar years. Initially, all of our conversations were around academic years since the main point of discussion was the educational activities. However, closer to the end of the project once we had a functioning system, we were told that getting data for a calendar year is also of high importance because that is how salaries are calculated.

At the end, we stayed with representation based on academic years, mainly because this was brought up too late into the project. We considered adding an option to choose between the two, but as it was not part of our requirements, it was lower priority. And since we had to pick one of the two, based on our conversations with the stakeholders, we believe academic years are more useful for the teachers and for managing their workload.

4.3.3 Counting activities

One additional issue that we discovered around the eight week of the project is that for payroll purposes it makes more sense to count supervisions that end in a given period, rather than all that are ongoing. Also, the issue of misinterpretation of the data was discovered, since if you want all the supervisions for a given year, you cannot just sum up all of the periods since these tasks most likely span among multiple periods.

However, we did not want to abandon our original implementation, since for the purpose of tracking workload, counting the active supervisions is a way more useful indicator than the number of ending ones. After all, a teacher might not have any ending supervisions, in the spring for example, but might have thirty ongoing ones, so if one is looking for a replacement, that teacher is certainly not a good candidate to assign more work to.

In the end, we chose to support both of these functionalities since they provide unique benefits to the teachers and managers. Now, the user can choose which type of counting to get statistics with, so our system is able

to help with both workload management and payroll data acquisition.

4.3.4 Workload metrics

One last point of discussion is the metrics we use to measure workload. Initially, we were mostly ironing out the details for educational activities, since those are most important. So, we ended up with four metrics - SECs, ECs, hours and count. For supervision, SECs do not make much sense since in general there is only one student and we do not have hours since that was not mentioned by our client. For management, we just have hours per month and count.

So as you can see our application uses a lot of metrics and they do not align between the different types of activities. This is not necessarily a problem, but it just means that in our back-end we cannot combine these activities under a common metric. The count is the only option, but we agreed that it does not make sense to do that, since it is unreasonable to equate, for example, coordinating a module and supervising a project in terms of workload.

It is also worth noting that our stakeholders do not have a clear view of how those measurements should be done, so for now we think the best approach is to provide all the options and data we heard during our meetings, which might actually help the client in picking an approach, while also making it easier for us since those functionalities are already there.

5 Test plan and results

Every product in our daily life is being tested many times before getting delivered to the customer. The same way, every software should be tested before being used by users.

This section gives a look at our testing process, including the tools and methodologies utilized, the types of tests performed, and the results that were obtained. The test plan and results are an important resource for us and our clients in assessing the system's quality and readiness for deployment in the production environment (where users can actually use the system).

5.1 Testing technologies

5.1.1 Postman

The main testing method for the backend was Postman. Postman is a popular API testing tool. The working principle is simple: Postman sends requests to the given URL and the request can be configured in many ways by the tester. After sending the request the response with its status code is presented. Different requests can be saved for automated testing in the future.

It is clear that Postman is a good choice for testing our various backend API endpoints. By forcing Postman to send incorrect requests, spoof requests, requests with injections, etc. we are testing the resilience of our backend logic.

5.1.2 Selenium

One of the test methodologies that were conducted on our system were Selenium automation tests. Selenium was used for integration testing and it served as a valuable tool to ensure the correctness of the whole system.

The method was used from the back end of the system and it consists of starting a driver (a Web browser of choice, such as Firefox) and conducting individual user tasks on the system. A Selenium script can click on objects and buttons and thus facilitate real user tasks that would be typical for normal usage.

5.2 Results

5.2.1 Postman

Early on we have decided on the formatting of the data that was expected by the backend and the formatting of the data that will be sent in response. API endpoints and their general pattern was agreed upon.

We have generated requests in Postman, correct and expected together with faulty and malicious for these endpoints. Valid requests were accepted and a correct, appropriate, response was returned. Invalid requests were rejected and an error response was sent back, and, most importantly, the integrity of the application was protected at all times.

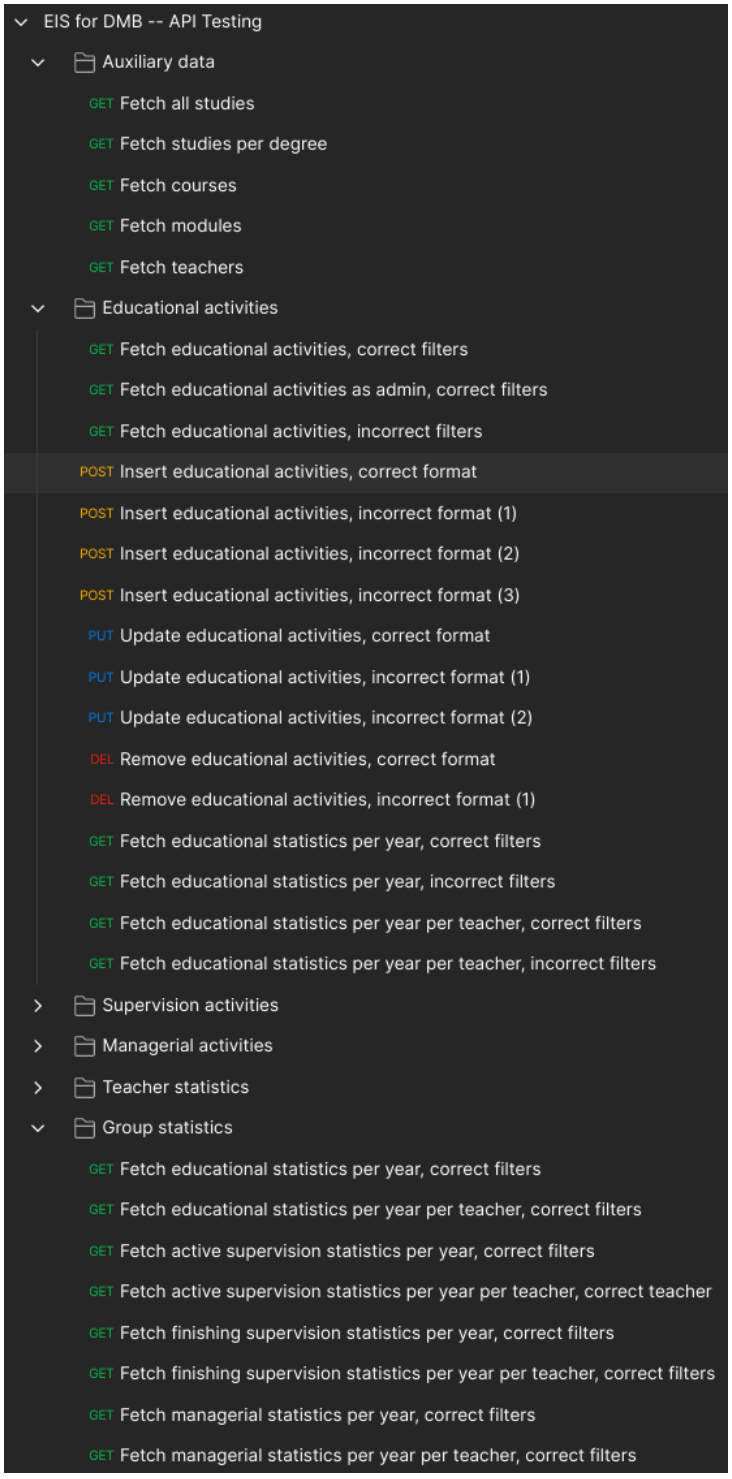

Figure 22: A snapshot of the collection of Postman tests

In the above figure we can see a section of our testing suite. All the entries are a request that we expect the system receive commonly. We have correctly and incorrectly formatted requests and expect appropriate handling. We can see the response that has been sent from the backend in the bottom screen in Postman and check the backend logs via the debugging terminal, as shown below.

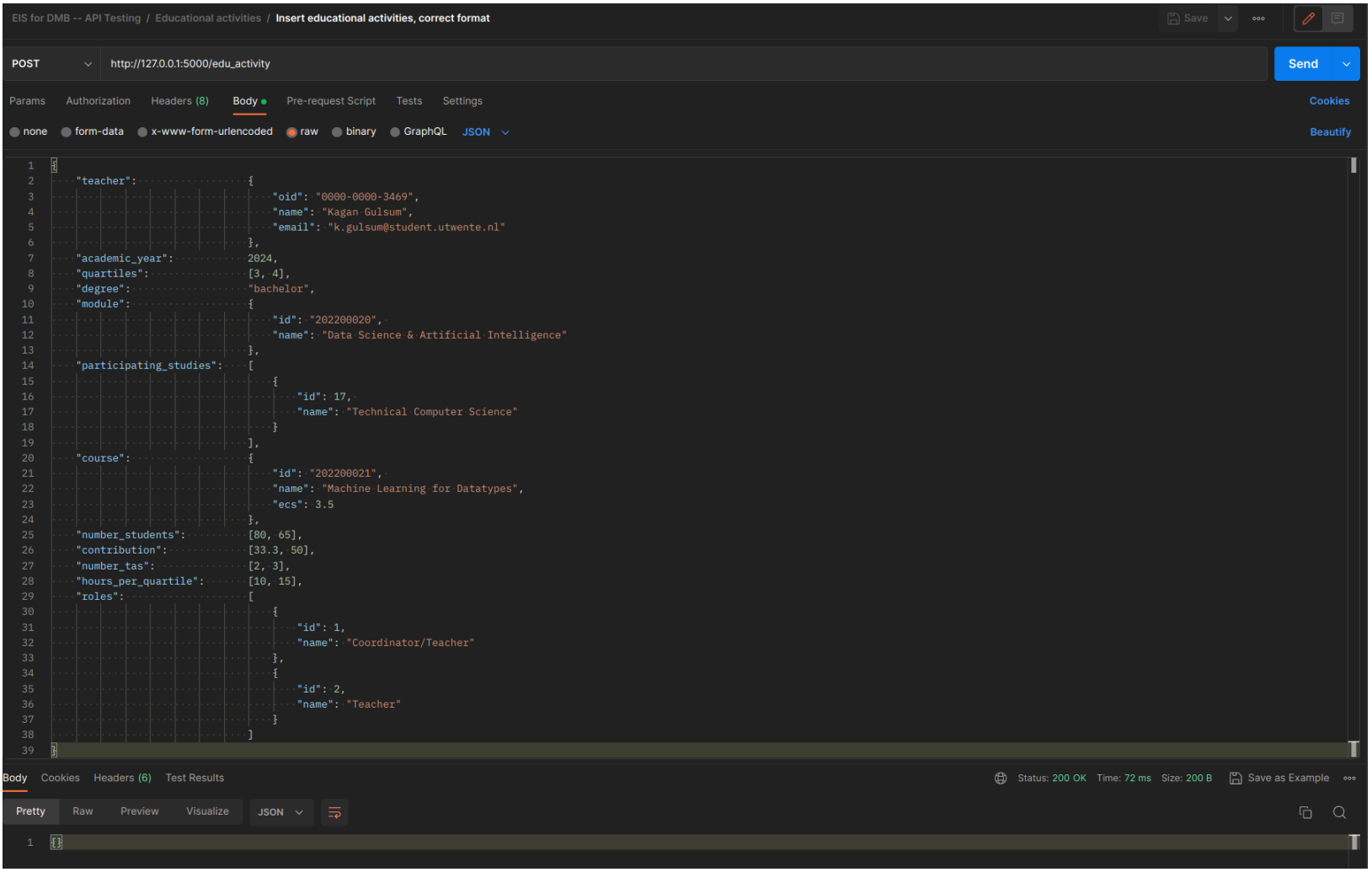

Figure 23: An example test case from the collection

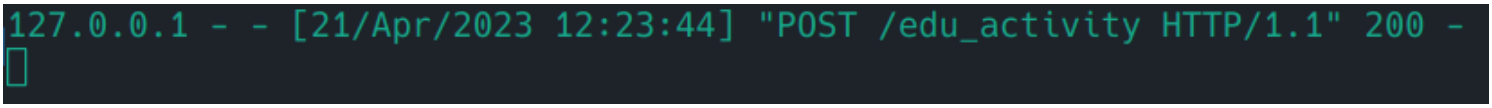

Figure 24: Log of the above test request, with the status code 200

5.2.2 Selenium

Typical tests that were conducted included tasks such as showing all activities, adding an activity, editing an activity and deleting an activity. All of these tests were conducted individually for educational, supervisional and managerial activities. These were separated for a teacher and for an admin user.

What is more, the filtering functionality was tested by providing different filter options and verifying the resulting data sets.

And finally, the overview functionality was tested. Unfortunately, there are no concise data visualizations indicative of the successful tests. We can only mention that Selenium helped with the overall functionality of the system and after identifying flaws, bugs could be fixed.

6 System limitations

Within less than 10 weeks, we managed to develop the desired MVP. However, due to some technical and time-wise limitations, there are still some undesired aspects of the system. In this section we will elaborate on these aspects.

6.1 General limitations

- We believe that fetching the activities could be optimized. Currently it takes a little bit of time, but nothing harming the experience majorly. Still, for the future we suggest feeding data partially.
- A loading indicator will be helpful for the user to understand that a process is running at the background, and they should wait until it is done. Processes such as: adding an activity, deleting an activity, fetching the activities, editing an activity and filtering.
- Inserting data to the database is currently handled manually via the database. This process should be either handled via an admin dashboard, or via a script (or both options).

6.2 Education Activities

- The filtering could become even more advanced, by filtering upon several modules or courses or studies. Currently the user is expected to write the name of the module/course/study, only one of each is acceptable, not multiple.
- Downloading the report of all the Educational activities that belong to the user is currently working on some of our local machines. On some, it doesn't. It depends on a specific Python module which should be installed and imported in a certain way on the machine itself. Therefore, we put it under this section, because this is a possible limitation. We cannot know it before we deploy the project to the Virtual Machine.
- In the process of adding an Educational activity, the user might expect the studies to be related to the course and module which has been selected. However, we decided to not link between them, due to the over-complexity it creates within the database (some studies belong to different courses, which belong to different modules, etc.)

6.3 Supervision Activities

- When the user is in trying to edit the supervisors via the pop-up window of Edit Supervisors, entering an invalid email doesn't turn the Finish button to non-pressable. This could lead to mistakes or confusion in this process. We did not solve the issue yet, due to technical limitations, but we believe it could be solved in the future. The described case is shown in the figure below.

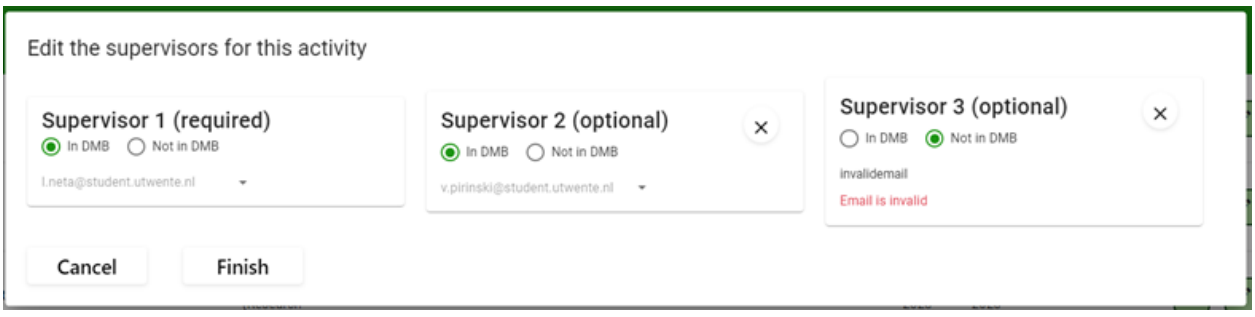

Figure 25: Edit Supervisors invalid email

- Another possible unexpected result of the edit-supervisors action occurs when the user switches the circular button from "In DMB" to "Not in DMB" and vice versa. If, in that case, the user doesn't

enter an email or select a supervisor within DMB from the list, then the previous value would remain the same as it was, and the type of supervisor would not change either.

- A loading indicator is missing when pressing on the Finish button when editing the supervisors of an activity.
- When adding a new Supervision activity, the Next button in the first page should remain non-pressable until a supervisor within DMB was selected. At the moment, it becomes pressable if the user just indicates that the second or third supervisor (without a value yet) is a DMB supervisor, ad the first one is an external supervisor. Hereby a figure to illustrate the case.

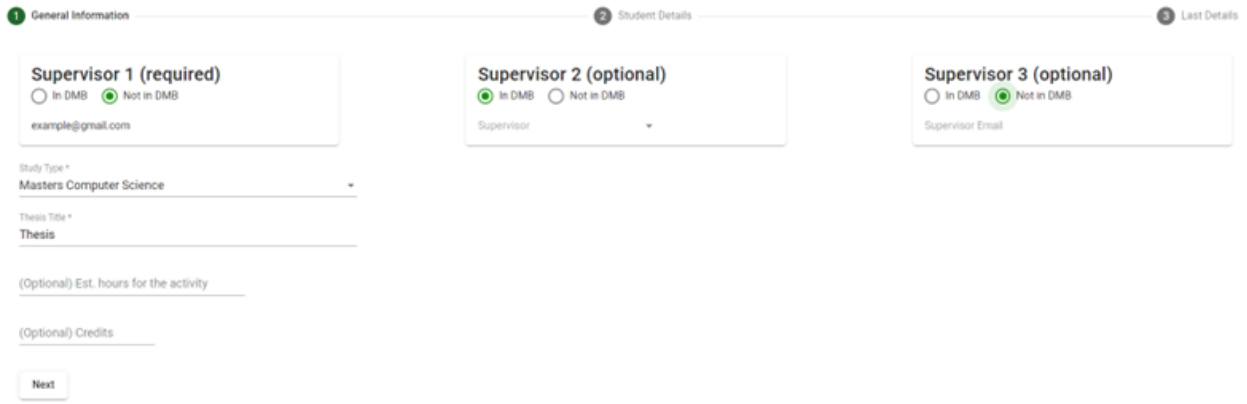

Figure 26: Add Supervisors

- Furthermore, the user cannot leave the DMB supervisor empty (as shown above) and submit the form. In that case, the submitting will not be processed by the backend. There is a need to handle this case.
- With relation to the previous limitations, the user might expect the option to remove a supervisor they added by accident. This functionality should be added as well.

6.4 Management Activities

- The admin user can see all the management activities of all the teachers within the DMB group in one table. However, it might be confusing for the admin that when they press on the checkbox called "See data of all teachers", the table changes its format first (another column named Teacher is added), and only then all the activities are fetched from the backend. Instead of these two steps it should become one step when both the data is fetched and the column is added to the table.

6.5 DMB Statistics

Currently there aren't any limitations on this page. It is completely in shape, as the clients suggested.

6.6 Teacher Statistics

Currently there aren't any limitations on this page.

6.7 User Management

Users cannot delete their own account via the platform yet. This is a privacy concern that should be handled.

7 Contributions

7.1 Alex

In therms of designing the system, I took part in discussions about the general design of the system, the technology stack and architecture. Since our team was split on two main sub teams, front-end and back-end, I was part of the back-end team. As such, my main tasks were related to the database, design of the database and schema and constructing SQL queries to fetch and manipulate data, and back-end, which included API communication with the front end and its inherent logic. My main focus was on the whole supervision activity functionality with its related tables. In the very beginning, I did some work on the educational activity to get used to the API communication.

Furthermore, I took part in ensuring the proper authentication of users in our system and built the authorisation mechanism that gives some rights to teachers and others to admins, of course, after joint discussions with the team. In this functionality, the automatic registration of new teachers in the DB, based on their Azure credentials, was also implemented.

The last implementation task that I had was to implement the report functionality. This was done for both educational and supervision tasks, even though for now it is present in the front end for the educational activity only. In short, a user can click the download button and this will download a report of all activities available to them in the main page table in both .csv and .pdf formats.

In terms of testing, I used Postman extensively to check the functionality of every API, before pushing the new changes to development. I also used Selenium for automation testing of the whole system.

Finally, I took part in constant team communication and discussions which are essential for the proper work of the team on a system of this size. I also took part in ensuring proper task division among the team and prioritisation for most efficient results.

7.2 Kağan

I was a part of the back-end team along with Alex and Vasko. I designed the API endpoints and the general pattern of communication. Most of the backend logic was designed and initially implemented by me. The back-end code for educational and managerial activities were largely under my responsibility. API endpoints to fetch statistics about educational activities were implemented by me which then became the template for all the rest of the overview related endpoints.

I established the connection between the backend and the database. Also, I took part in the initial design of the database and for each iteration, I closely worked with the rest of team. I designed the formatting of the data which is passed around in the communication of backend and frontend. Subsequently, I also implemented the parsing of this data. For all the other tasks related to the backend, I thoroughly checked and reviewed design decisions, lines of code, etc.

So, from a technical perspective I designed and initially implemented the architecture of the server, prepared coding guidelines, created the database structure and standardized data formatting. In terms of soft skills, I took extensive meeting notes and coordinated task division amongst the team members where possible. I engaged in fruitful discussions and started many of them to decide on design choices, be it aesthetics related or technical topics.

7.3 Liran

I belong to the frontend-team, alongside with Robert. In addition, we worked the entire team together to discuss the system's most important decisions - Database design, system's architecture, technology stack decisions, etc.

Robert and I split the work between us, and helped each other when needed. Robert had a prior experience in Angular, and my experience was in React. However, we decided to go with Angular because Robert

implemented similar website pages (as we were asked to develop) in the past, and we leveraged it to accelerate the development process.

I learned Angular and Typescript from scratch, despite its steep learning curve. Eventually, I developed the following pages / features:

- Educational activities page (including editing & removing)
- Management activities page (including editing & removing)
- Add new Supervision activity
- Editing & removing supervisors of a supervision activity
- Statistics graphics (on dummy data)

I focused on communicating with the backend team, to understand which API are missing for us (the frontend team), and helped to figure out the data models to be sent from/to the backend.

On a personal note, I really liked to work with my teammates. I feel blessed to work with Kağan, Alex, Vasko and Robert. They were all patient and kind, even when some of us had a busy deadline and contributed less for a week or two.

Along with a busy schedule for each of us, we worked as a team, and that is the reason we managed to accomplish everything in a short period of time.

7.4 Robert

In the first part of the module, like all the others, I contributed to discussions about general system architecture and database design. Together with Liran, I was part of the frontend team.

For the single page application, my main contributions were the following:

- Creating the initial project environment and implementing Azure authentication flow
- Implementing the form to add a new educational activity
- Implementing the form to add a new management activity
- Writing the logic for processing stats data and displaying graphs of different metrics in the DMB group stats page(s)
- Creating the teacher overview page
- Implementing the logic for filter fields

Other than that, Liran and I helped each other and worked together when needed. I also was involved with debating and establishing the API format with the backend. Lastly, I spent some time revisiting already implemented pages and making minor quality improvements to them.

7.5 Vasko

I was part of the back-end group. I was largely involved in the design and evolution of our database. Some of the API calls were implemented by me, like adding supervision activities. Also, for managerial activities the filters and the current functionality for the overview were done by me.

I also acted as the main contact person for most of the information gathering process, talking to our client, teachers, programme coordinator and key users of Osiris. This made the communication with stakeholders and getting feedback quite fast.

Of course, I also participated in most of the discussions we have had about the design of the system, and joined all activities and presentations we had for this module as a team.

8 Conclusions and future work

8.1 Future work

The system we have built bridges the gap we were asked to solve. However, there could be some extra work to optimize and improve the main purpose of the system - facilitating the process of managing the teachers' activities.

In this section, we will elaborate on the various possibilities of improving the platform, such as automation, scaling-up (i.e. extra features), scaling-out, and more.

8.1.1 Automation of tasks division

One of the main goals of our system is to comprehend the contribution of each of the teachers. Consequently, the admin divides the future activities among the teachers accordingly.

With an automation feature that facilitates this process even more, we believe that the system will have a great value for the admin user. An overview is still required to be done by the admin, to ensure that an equality is guaranteed among the teachers.

8.1.2 Notification mechanisms

Our clients told us that many teachers are facing the problem of forgetting to complete some activities before their deadline. Therefore, there is a need of a Notification system to remind teachers to fill in information.

8.1.3 Osiris integration

To minimize the amount of Education activities to be added manually by the teachers, the system should be able to fetch data about courses from Osiris, and as a result facilitate the process of adding the activities into our new platform.

8.1.4 Better data representation

In addition to the previous future work, we believe that an Osiris-like catalog of the data, fetched from Osiris itself, will make the user feel more familiar with the system. This data includes modules, courses, and studies. We believe that a combination of the already existing way of organizing the data (as in Osiris), plus using a contemporary design, will improve the usability of our system.

8.1.5 Publications/Canvas integration

To minimize the amount of Supervision activities to be added manually by the teachers, the system should be able to fetch data from Canvas (or other Publications) of research projects or other supervised activities.

8.1.6 Database management

The system should support an easier management of the database through an admin panel or other dashboard. It will facilitate the database management process. This will include features such as user-friendly editing of existing modules and courses as well as adding new ones.

8.1.7 GDPR compliance

To ensure we comply with the latest privacy regulations, we will add functionalities such as the easy deletion of a user account and all of the data related to them.

8.1.8 Deployment

The system is currently hosted locally, but for it to be ready for use by the DMB group, we need to deploy it in the virtual environment that was provided to us by the technical staff of the group. This will include a Docker implementation where the front end, server and database will be hosted.

In addition to deploying to the intranet servers of DMB we should use the Azure services owned by the university. Right now we use Robert's Azure account for this service.

8.1.9 Scaling-Out

The system currently supports the DMB group only. We believe that it could benefit other groups as well. Most of the data to be collected is the same, in terms of data types, tables, forms, etc.

Scaling-Out could be achieved by first reaching out to other groups within the UT, and presenting them the EIS system while explaining its advantages.

If the future clients are willing to have such system, the next step will be to technically prepare the system to scale-out, meaning to have more "instances" of the system, or else to add various access levels (for instance: User Group), so that teachers could access only the data that belong to their department/group.

8.2 Reflection

We learned a lot by working on this Design Project. We learned how to approach clients by ourselves, and how to manage the initial process of gathering requirements.

We also understood the importance of designing such an Information system for an organisation such as the University of Twente which was also mentioned in Shubham Gupta, Ashish Kumar and Roheet Bhatnagar case study 1. As students in our graduation phase, we experienced the chaos of choosing a graduation project. From a managerial perspective, this chaos is nothing compared to the complexity of the decisions that has to be taken in a research group. This gave us more motivation to build a good system that will facilitate efficiency and ease for teachers and managers.

Team work was an essential part for the success of this project and we learned a lot about the importance of constant communication and team discussions. But more important was that every member managed to have their own space of time which combined individual skills and team work for best results.

Furthermore, the newly acquired skills on this particular technology stack were both interesting to learn and research, and they are also a valuable asset in our skill set for future projects.

What could have been done better is our approach to development. The team could have adopted an even more agile approach by defining and following implementation cycles. A TDD (Test Driven Development) strategy could also help with identifying flaws in the system early on in the process, leading to less expensive mistakes. And lastly, even better documentation of our code would make the system more understandable and consistent for future work.

8.3 Final remarks

This project was very fun to implement and was also a great opportunity to gain knowledge. We believe the final product satisfied the needs of our clients and hopefully will become an integral part of the management of the DMB group in the University of Twente.

This challenge gave us a global perspective about the importance of Information systems in successful management of groups and in organisations in general and particularly in our institution.

We believe the project will have a positive impact on the workflow of the teachers in the DMB group, it will ease the management and in turn, will also benefit the students of the University of Twente.

9 References

1. Shubham Gupta, Ashish Kumar, Roheet Bhatnagar "Role of Information Systems in an University Setup – A Case Study", December 2016 https://www.researchgate.net/publication/317649791_Role_of_Information_Systems_in_an_University_Setup_-_A_Case_Study
2. GitHub repository of our project (ask Kağan Gülsüm for access) https://github.com/Chmlgy/eis_for_dmb