UNIVERSITY
OF TWENTE.

Design Report - Group 7

SmartTeam Webapp for creating effective teams

Authors:
Cristian Trusin
Luuk Willems
Tudor Nastase
Kristupas Raicevicius
Cosmin Ghiauru

Supervisor:
Yeray Barrios

University of Twente

Faculty of Electrical Engineering,
Mathematics & Computer Science

November 11, 2022

Abstract

Throughout life, especially in the academic environment, people are part of tens, maybe hundreds of teams. Every team is created with the purpose of achieving a common goal that is shared between its members. The ideal team will be able to take advantage of each member's strong points and make up for their weak ones, so team diversity and completeness is essential for achieving a desired outcome. How good the chemistry of the team is depends on various criteria such as their nationality, spoken languages or their natural team role (such as Belbin role). In order to not just randomly assign students to teams, teachers have to put considerable effort in creating teams with students that they consider to be compatible. Such a task is very time-consuming. On top of that not all teachers are experts in maximizing team compatibility. An app that can successfully compute students teams with maximizing effectiveness in mind can prove to be a big help for plenty of teachers.

Contents

1 Introduction

In this section, we will introduce the main factors that stand behind the implementation of our solution and why such a product is necessary.

SmartTeam was an application developed through the course of the Design Project module of the Technical Computer Science programme at University of Twente. The project was done by: Cristian Trusin, Luuk Willems, Tudor Nastase, Kristupas Raicevicius and Cosmin Ghiauru, under the supervision of Yeray Barrios.

SmartTeam can be found at: `https://smartteam.utwente.nl/`

The repository can be found at: `https://gitlab.utwente.nl/s2162652/smartteam`

1.1 Background

During the Computer Science bachelor's degree at the University of Twente, working in groups is not a choice, but an important skill that every student has to develop from day 1. It is a requirement to work together in teams because the workload and complexity of projects and assignments become higher with each module. A student can be expected to be in up to 20 different project or lab groups during their first year alone. The process of matching students into groups is left for the teachers to decide, so most of the time students can match up with whoever they want, or they are randomly assigned. Such approaches to creating teams can create various conflicts or imbalances in teams, which the teacher sometimes must fix in order to not compromise them. The lack of any matchmaking criteria results in arbitrary teams, which have random compatibility and effectiveness.

As a result, the experience of the students is, naturally, quite mixed. Some people might get to work alongside their friends, while others might create long-lasting friendships or collaborations with the people they were randomly assigned with. But not every student will necessarily have a pleasant experience when it comes to team creation. Sometimes, a student matching up with friends does not lead to a positive result due to a lack of diversity or focus on the goal. Teams that are randomly assigned can suffer from conflicts due to lack of compatibility. Since the first year is erratic and students must adapt to many changes and overcome many challenges, the proverbial "bad luck" of having an ineffective or unreliable team or teammate can make the difference between passing or failing the first year and receiving a BSA.

1.2 Problem statement

If a teacher wants to create a matchmaking that is more efficient than the random or student-driven ones, then the time and effort invested increases drastically. After gathering up all students that are going to be matched, the teacher has to actively come up with criteria on which to match each student group in order to have a higher than random efficiency. Without careful consideration and relevant knowledge in matchmaking

techniques, this process can prove to do more harm than good or not provide relevant improvements. The next challenge that the teacher would encounter is gathering relevant information about the students that are about to be matched. On top of having to manually gather the data for each student, there is a very limited amount of student information on teacher platforms, such as: name, study, student number, email or courses followed. If a teacher would prefer to use additional student data for matchmaking, the most common solutions are to use either personal experience, speculation, or a survey. The first of the two tend to be highly biased, so a survey remains one of the only ways to get accurate student data, assuming the students will answer the survey truthfully. Creating such a survey takes time, effort and technical knowledge. Finally, the task which is usually the most resource-exhausting is successfully matching all the students of a course based on certain criteria. It might be reasonable to do so for a small number of students (ex. 30 students), but when a teacher has to pair up a whole course of over 400 students, the task can become very demanding. In such a case, a great amount of hours is invested in the matchmaking, and the resulting teams can also suffer from human error due to various reasons. After considering the process stated above, we can conclude that a tool which can simplify each step in the matchmaking task and achieve higher overall team effectiveness can provide teachers with a much needed assistance.

1.3 Current Solutions

According to the research that was conducted both by interviewing teachers and talking to fellow students, there are 3 main approaches that module coordinators take when it comes to setting up project teams:

- students choose their own teams
- teacher randomly assigns teams
- spreadsheet with all students and their information, followed by manual matchmaking

Following, is an overview of each of these approaches, how they fall short and how they can be improved upon. The main takeaway is that there might be more need for an automated, centralized approach than module coordinators currently may think.

1.3.1 Student driven matchmaking

This approach is favoured especially by coordinators of higher year modules and there are a few clear reasons for this. Firstly, second and third year students area already acquainted with a good portion of their peers and are likely to make good or at least informed decisions when it comes to choosing teammates. Secondly, these students are experienced enough to get around the shortcomings of a less then ideal teammate, generally by dividing work according to everyone's capabilities. Finally but importantly, this approach for the most part does not require any intervention on the side of the coordinator, unless problems arise(e.g.: students without a group, in wrong groups etc.).

However, when using this approach for first year courses, there are a few issues that emerge. First, when a significant number of students participating in a course are not sufficiently acquainted with the other so it might be difficult for effective teams to emerge naturally under these conditions. Usually a WhatsApp or Discord group is created by a teaching assistant with the purpose of finding teammates. Alternatively, team formation sessions can be held in order for students to get acquainted. Both of these solutions work but it seems clear that some degree of randomness exists. What happens is that someone is likely to go to the first team they find that is looking for members. Conversely, a team looking for members will just accept the first person who wants to join.

The second main issue with letting students choose their own teams is that restrictions are difficult to enforce sometimes. It may be desirable, for instance, to not have more than three people which speak the same language in a group of six. However it may be difficult to dissolve groups that don't meet the criterion, as the teacher will have to actively oppose the wishes of the students.

A possible major improvement to this approach would be to let teams searching for one or two more members specify which type of member they are looking for. From the other side, the student looking for a team would be put into a better position if she knew what kind of teams are available.

1.3.2 Random matchmaking

As with the previously mentioned approach, a big advantage of this method is the ease and efficiency with which team can be created. The teacher does not need to interfere too much, again, unless problems arise (e.g.: students that are not supposed to be in the course, dropping out of the course etc.). Random is generally fair but it can also be troublesome and this method may lead to conflicts. This is the likely reason why this method is mainly used in Module 1("Pearls of Computer Science"). As a new partner for the lab assignments is chosen on a weekly basis, having a less than optimal partner is acceptable, as it will be changed the following week.

The process of making the teams can still be optimized, as Canvas does not have the option to create random groups automatically. A spreadsheet or python script can probably be used to assign students in random pairs or teams, but while this is not too difficult for a computer science professor, this task could made easier by means of automation.

1.3.3 Manual matchmaking through a student information spreadsheet

This strategy for making effective student groups is used in Module 4 ("Data and Information"), where teamwork is very much one of the learning goals. The great thing is that teachers can have full control over groups and can make sure that all restrictions are put in place(e.g. two BIT students and four TCS). Additionally, more information about each student can be stored, things such as Belbin role scores, past grades, whether

or not the student is doing the module as a minor, any such information that might help match students together.

Nevertheless, this control comes at the price of complexity. The task of managing potentially four hundred students manually can be tedious, which is why currently multiple teachers and teaching assistant are in charge of ensuring the smooth cooperation of project teams. Furthermore, the process of matchmaking can be difficult with such a large number of students. A backtracking strategy that attempts all possible matchings of 400 students into teams of 6, until all restrictions are met, might take over 24 hours to run, even on computationally powerful machines. More likely, though, teachers would just manually match students themselves, a task that can take several hours to complete.

Even though in terms of group effectiveness, this solution seems to produce the best results, the technical challenge difficult enough to deter module coordinators from using it. A great improvement would, thus, be to hasten the process, as well as facilitate it.

2 Proposed Solution

After consulting various relevant research papers and interviewing with multiple teachers, we came up with our own solution for the matchmaking problem: SmartTeam. SmartTeam strives to solve the problem of maximizing team performance and adequately grouping students based on certain criteria while also simplifying the workload of the teachers. The teacher or coordinator of the module still retains a lot of control over the formation and managing of groups while using SmartTeam. They can upload the list of students, select the matchmaking criteria and the groups will be automatically formed in a way that maximizes their expected effectiveness.
Teachers have the option to move or remove students from groups and make any changes they wish before deciding to export the groups and upload them to Canvas. As for the students themselves, all they have to do is login on the platform using their university credentials and answer a few questions about them and their preferences through a survey. At the end of this process, we expect that both students and teachers experience of group work will be positively impacted

2.1 Overview

As we mentioned before, facilitating the task of creating groups for courses stands to benefit everyone involved. The module coordinator will save time and actively help students who, in turn, will also benefit from being in more balanced, heterogeneous and overall more effective teams. We are achieving this by means of automating the matchmaking, but still preserving freedom for the teacher, as well as the student. We will go again trough the three main strategies of grouping students and explain how our system facilitates and enhances each of them.

2.2 Students choose their own teams

While finding a team is not always a hard task for students in advanced years of the study, first year students and minor students in general usually struggle when looking for teams. What we aimed to do in order to help them find the perfect team is let them have some a priori knowledge of all teams that are not yet complete and which type of member those teams are looking for. To this avail, SmartTeam offers a simple workflow. The teacher needs to first set-up a course, select the maximum group size choose criteria that will affect the overall team score. After the course is launched, any student enrolled in a course can simply log onto the platform with their university account, click on the course in question and start browsing through prospective teams. On this page, each team will initially have an associated number and a score will be computed once more than a student joins. This way, students can have the full benefits of the team score so that they can choose their teams in a smarter way.

2.3 Students are randomly assigned

In case the teacher doesn't wish to let the students choose for themselves and also doesn't want to use the criteria as a basis for the matching, then the random matchmaking is the offered solution for the task. The teacher has to choose a group size at the time the course is created and can choose criteria if needed, or no criteria can be used at all. In case the criteria are used, the groups will still receive a Group Score that might let the teacher know if some of the groups are highly incompatible. In either case, after going further with the matching, teams will be randomly assigned based on the team size chosen by the teacher.

2.4 Students are smartly assigned

In the case in which the teacher doesn't want to let the students choose their own teammates, but wants to have more efficient teams than randomly assigned ones, the teacher can choose to create teams with a "Genetic" matchmaking. The name of the matching comes from the genetic algorithm used as a means to discover the best possible student combinations in a relatively short time. Once the teacher sets up the course with the preferred criteria and team size, students will receive a survey in which they must provide some information about themselves and their preferences. The system will then use the provided answers as a means for computing the Team Score for each team. Once the students have completed the survey, the teacher can choose to match students with the "Genetic" matchmaker. The matchmaking algorithm automatically matches all the students that completed the survey within seconds, all while providing a tremendous increase in Team Scores compared to the random matchmaking. After the teacher is happy with the results and considers that no changes need to be made, the teams can be exported to a .csv file and imported in Canvas.

3 Global Design

In this section we will analyze the requirements of the system, offer an overview of the key design choices we made, ranging from the programming languages we used to planning, as well as give a rundown on how the system works.

3.1 Requirement Analysis

In order to determine the requirements of our app, we had meetings among the team members, with our supervisor and with various teachers that might be interested in our solution.

Yeray Barrios, our supervisor, who also served as a product owner for the purposes of this Design Project is our main stakeholder. Most of the initial requirements, as well as the initial idea for the Smartteam application comes from the project proposal he formulated. The proposal demanded Smartteam to be web-based application that will create teams of students that are more effective than the teams that teachers can currently create. After our first meeting with our supervisor, who is our main stakeholder we got some ideas of what the system should looks like, that we were subsequently able to separate into clear and concise requirements as we advanced through the project.

Teachers and module coordinators represent the main users of the application, they are also very important stakeholders. We hosted interviews with each one of them, to talk about how they create teams in their module and to ask for functionality they would like to see in our application. Their input proved very valuable and was taken into consideration when designing the functionality of the app. A summary of each interview with the teachers can be found in the appendix A, B, C, D. They all have slightly different approaches when it comes to making student groups but we could find some similarities between their respective views. Teachers either make the teams themselves, or allow students to choose their own teammates. In the case when teachers make the teams themselves, our application would help them with this process since it will automate this process, thus saving precious time, and also improve it by using criteria supported by research, and removing any room for user error.

However, for teachers that allow students to choose their own teams, our application wasn't useful, so we had to change it to satisfy the needs of these stakeholders as well.They would like to have at their disposal an app similar to the group management tool from Canvas, but with a few additions that would help students make more informed decisions on who they are choosing to group up with. Such additions could include a more dynamic group formation system, where students could input some preferences, a tag system with the help of which someone that makes a group could specify what kind of team member they are looking for. Conversely, the tag system can also be used by someone who would be looking for a group with preferences similar to their own.

Another important stakeholder for the application is represented by the students that will have to log in and fill in a survey for each course. As students ourselves, we realized

that to optimize their experience we need to make it so they have to spend a minimum amount of time and effort on the platform. Peer review sessions have also been very helpful, as we got a multitude of useful tips. For example, we realized that some students may try to cheat the system by lying in the survey. This led to the design decision to not use "desired grades" as a question in the survey, as it would be too easy for anyone to just say they want a 10, in order to get matched with an already strong group. Another key insight was represented by the fact that many students might simply forget or not have time to answer the questions. To mitigate this problem we store past responses and automatically fill them in.

Now that we have explained the method we used to find the requirements, we can finally list them. An overview of the requirements can be found in the tables below:

Functional Requirements
The system must provide login functionality for both students and teachers, using the University of Twente login credentials.
The system must persistently store teachers together with the courses they teach and students, together with the data they provide, as well as the courses they take part in.
The system must allow uploads only from teachers and the accepted files must be of type ".csv".
The system must save the changes that teachers make to the teams, by means of a "save" button.
The system must generate a ".csv" compatible with Canvas.
The system must not allow students to write text or unsanitized input in input fields.
The system must show both students and teachers the groups that were formed, as well as the associated score.
The system must accommodate a student and a teacher being in multiple courses at the same time.
Non-Functional Requirements
Teachers should not need to spend more than 2 minutes setting up a course
Teachers should be able to find a student within 10 seconds, if they use the search or filter functionality.
The function of automatic matching of students into groups should not take more than 5 minutes, regardless of the team size and number of students.
The teacher's interface must present all the necessary information without feeling too cluttered.
The student's interface must be easy to navigate.
The database should have enough capacity to store multiple courses at the same time.
Transactions between front-end and back-end should not take more than 2 seconds in total (i.e. the page should not take more than 2 seconds to load).
The automatic matchmaking should provide groups with an effectiveness score higher than random matchmaking.
In the teacher's view, the teams should be sorted by their "effectiveness score".

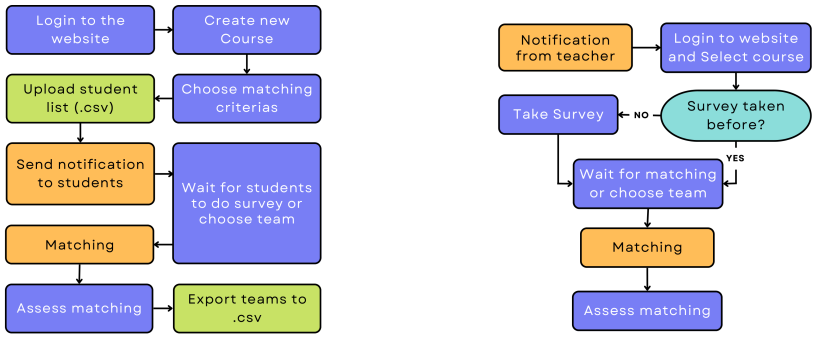
Table 1: System side requirements

Teacher
As a teacher, I want to login with my university account
As a teacher, I want to create and delete courses.
As a teacher, I want to give my course a name in order to identify it.
As a teacher, I want to upload a list of students from Canvas so I can start creating teams.
As a teacher, I want to be able to manually create or edit teams, so that I can manage edge cases or individual problems (e.g. students dropping out).
As a teacher, I want to decide whether to create teams manually, automatically based on criteria or let students join teams by themselves.
As a teacher, I want to select the maximum number of students per team.
As a teacher, I want to select which criteria matchmaking uses when creating teams automatically.
As a teacher, I want each team to have an "effectiveness score", regardless of the method used to create it.
As a teacher, I want to be able to filter students based on different criteria (e.g. gender, study, tags)
As a teacher, I want to be able to manage multiple courses at the same time.
As a teacher, I want to be able to export the teams as a ".csv" file so that I can upload it to Canvas.
As a teacher, I want to be able to click on the student to see information about them.
As a teacher, I want to be able to Lock a course so students cannot join/change teams anymore
Students
As a student, I want to login with my university account, so that I don't have to create a new account.
As a student, I want to be able to see a list of courses I am registered for.
As a student, I want to select a course and see the current paired and unpaired students.
As a student, I want to take the Belbin roles test inside the application.
As a student, I want to be able to click on a team that still has empty spots, in order to join it.
As a student, I want to see the effectiveness score of the team that I want to join or have already joined.
As a student, I want to use a tag system to see which teams fit my preferences.
As a student, I want to filter potential teams, based on tags.
As a student, I want my answers to be saved and auto-filled in future surveys, so that I save time.

Table 2: Requirements expressed as User Stories

3.2 Proposed Workflows

Figure 1: Proposed workflows



(a) Teacher workflow

(b) Student workflow

For Teachers *Figure 1a*
The teachers will login using the University of Twente Microsoft login. They will select or create a course for which they wish to make teams for. Afterwards they will specify the team size and the maximum number of people that can speak the same language in a team. The teachers will be asked to upload a list of students in a .csv format which can be obtained from Canvas. Then the teacher should notify the students to complete the survey or choose their team if the teacher decided to do so.
After the students have filled the survey, the teacher can perform the matching, either by hand or automatically using one of our matchmakers. If the teacher decides to let the students choose their own teams, they can just wait for them to do so. After any of these is done, teacher can assess the matching, make any adjustments they deem necessary, and lock the course so students cannot change teams anymore. After this step is completed the matching is done and the created teams can be exported as .csv file, that later can be uploaded to Canvas.

For Students *Figure 1b*
Students will receive an email notification from the teacher asking them to fill in the survey or choose their team. The students will then login to the website using the University of Twente Microsoft login, and choose the course which they have to respond to. If they have not taken the survey before - they will be asked to do so. After taking the survey or if they have taken it before they can either wait until the teacher does the matching or choose their team if the teacher allows this. The system will perform the matching, and after any adjustments made by the teachers, the students can assess it, and if the teacher hasn't locked the course the students can change teams.

3.3 Preliminary Design Choices

3.3.1 Architectural Design Choices

We decided that the system will be a web based application, since we had experience creating web applications, and web applications can be used on any platform. Moreover this was the initial requirement from our supervisor. We considered making this system a Canvas plugin, but this idea was rejected by our supervisor since it would have been more resource intensive, would have imposed more limitations and would have required permission from LISA, which would have slowed our project down since requesting permission, testing and deployment would have to go thought a third party.

3.3.2 Programming Languages

Java was chosen as the language for the back-end. This choice was made, because all of us had experience using it through projects for university. This also meant that everybody would be able to create code for the back-end and we would not have to depend on only a few team members.

SQL is the language used to communicate with the database. This is the language we all have had to work with before and is the default language for the Postgres database we used.

JavaScript/CSS/HTML This combination of programming languages is utilized almost universally when developing anything web related, which is why we also choose to utilize them. We also used JavaScript with the jQuery library which implements many key features we required for our project, such as AJAX requests and selectors.

3.3.3 Frameworks

Spring Framework

For the back end the Spring framework was chosen. The Spring framework is one of the most popular frameworks for enterprise Java development. Some of the benefits of using Spring are:

- Spring is used by millions of developers and makes it easy for developers to create code that is reusable, high performing and easy to program.
- Spring helps programmers by promoting good programming practices by having a lot of detailed documentation online with examples and information. This also helps to easily implement new functionality to your Java application without having to write much tedious code.
- Because of the wide use of Spring framework there is a lot of answers online for when you run into problems, which makes solving problems a lot easier and faster.
- Spring uses a modular approach which makes it easy to only import the parts you need and makes sure you don't have a lot of redundant libraries that only take up space.

- The Spring web framework that we are using, is a well-designed web MVC framework that provides a lot of build-in functionality which takes a lot of the work away from the developer.
- Spring is also build to be lightweight and so can also be run on lower power devices without much trouble.
- Spring Boot makes it possible for a developer to setup a Java web application in minutes by giving developers a ready to use application that provides defaults for code and annotation configuration for starting projects in no time.

Another benefit that using Spring had for our projects is that we already had some experience with using it and where able to quickly get started building our web application. All the benefits above and the fact that we had some experience with it made it an easy choice for us.

Bootstrap

We choose to use bootstrap for designing the front-end. The reasoning behind choosing bootstrap was that our team was already familiar with the framework and we had already used it to build similar applications in the past. Bootstrap is a free and open-source CSS framework that is aimed at enabling responsive, front end web-development and as such was a perfect fit for what our project aimed to achieve. Some key features of bootstrap that we utilized are the flexbox system, its table and card elements as well as all the CSS styling that it provides.

Figure 2: SmartTeam Logo

3.3.4 Planning and organization

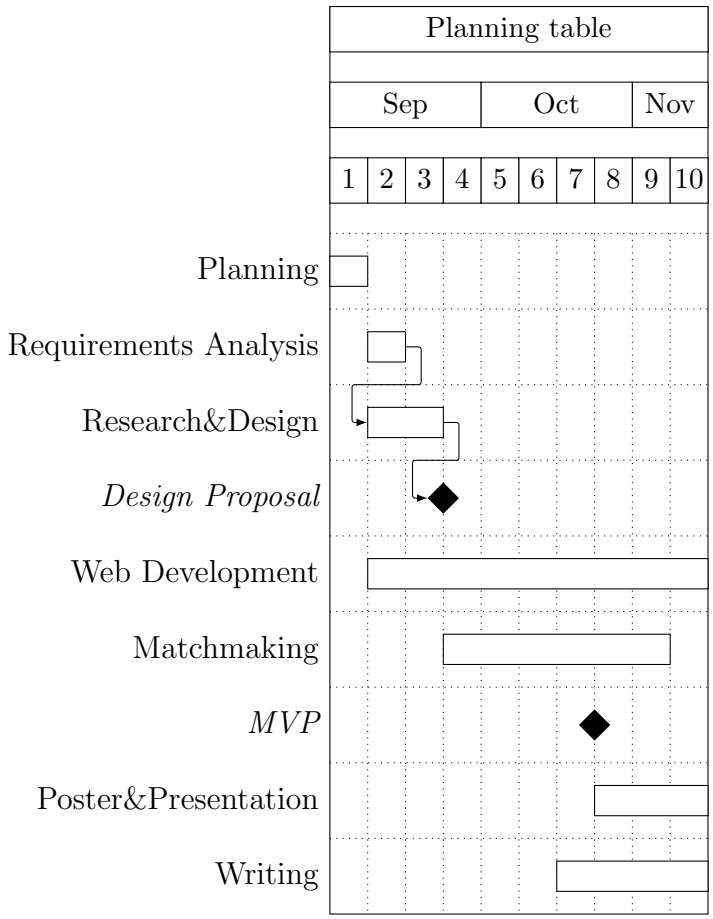


Figure 3: Planning

We used a SCRUM like methodology thorough this project, where we had a deliverable every 2-3 weeks.

The project was split in three stages: the core functionality stage - **Week 2 - Week 6**, the matchmaking stage **Week 7 - Week 9** and the refinement stage **Week 9 - Week 10**. An overview of our planning can be found in Figure 3.

At the start of the first stage (**Week 2 - Week 3**) we split in two teams:
Team 1, consisting of Kristupas, Luuk and Tudor started working on implementing the core functionality of the System. Mainly login and registration - Luuk, Database structure - Tudor, Frontend - Kristupas. Team 1 finished the core functionality by **Week 6** and was improving on it over the course of the next weeks.
Team 2, consisting of Cristian and Cosmin, looked into relevant research about matching and creating efficient teams, and choose algorithms and criteria which were used to match students in the second stage of the project. Besides this Team 2 hosted interviews with stakeholders to establish requirements for the system. Team 2 finished this in week **Week 5**, and afterwards they started implementing and testing the algorithms they chose. We had a minimal viable product (MVP) by the end of **Week 7**.

The second stage consisted of Team 2 making a random matchmaker that Team 1 integrated into their system, while Team 2 continued working on a better matchmaker. Besides this members of Team 1 and 2 started writing in the Design Report during the second stage of the project. By the end of **Week 9** Team 2 had a good matchmaker, that was integrated into the main system by Team 1, while Team 2 continued working on the report, poster and presentation.

The last stage only lasted a week, in which both teams ironed out any remaining bugs, and finished the poster, presentation and report.

3.4 System Overview/Manual

In this section we will show and describe the functionality of each page of the web application.

3.4.1 Authentication

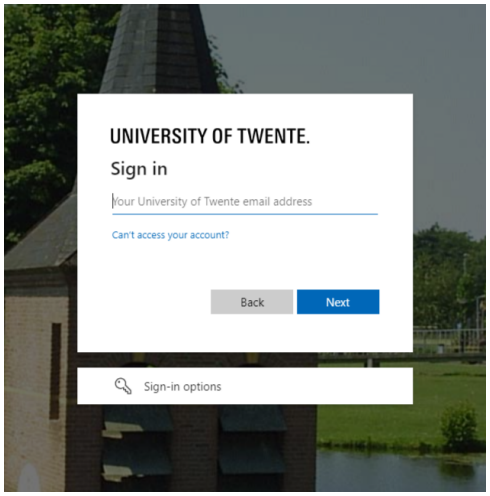


Figure 4: Login Page

Students and teachers can login into SmartTeam using their University of Twente account (Figure 4). The system will automatically fetch if the user is a student or teacher and redirect them to the appropriate view.

3.4.2 Course Dashboard

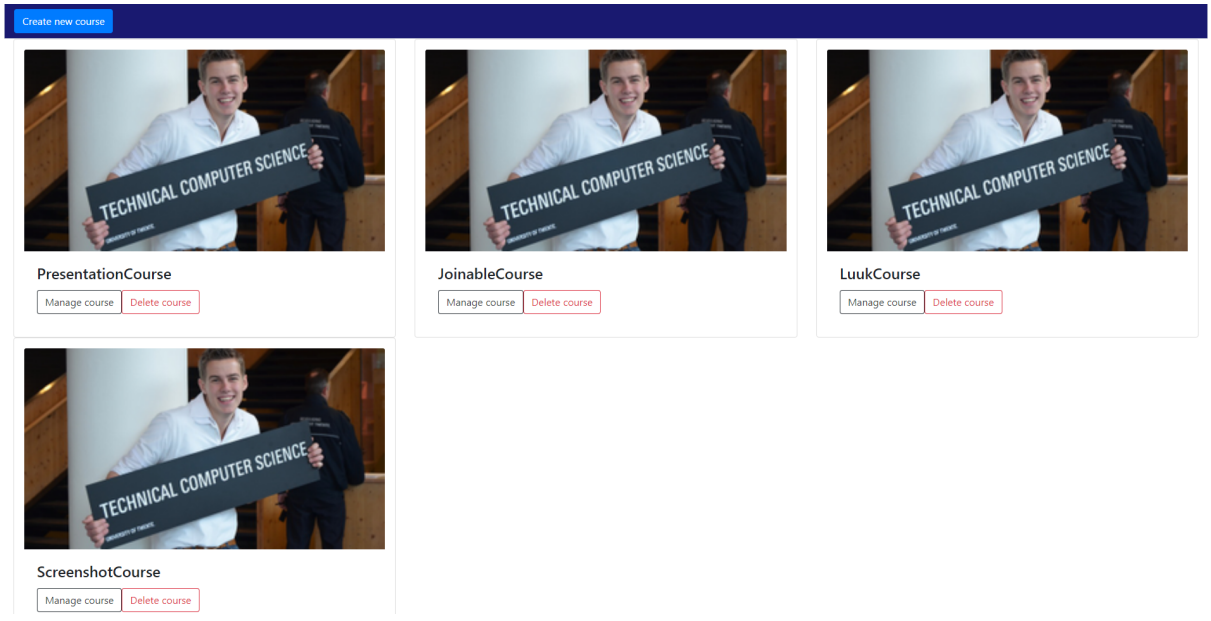


Figure 5: Course Dashboard Teacher

A course represents a module taught at University of Twente. It stores all the students participating in it, their data, the current groups and their scores.

Teachers

Teachers can choose which course they want to manage on this page (Figure 5) or create a new one using the **Create new course** button (Figure 6). Where they can specify a name for the course, the team size, the maximum number of people speaking the same language, and they can also upload the list of students from Canvas as a .csv file.

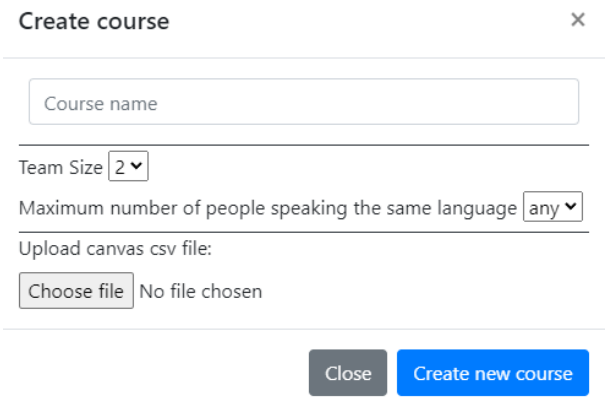


Figure 6: Creating a new course

Students

Students can also choose which course they want to view on this page (Figure 7). They can also see their personal details on the right, and they can also retake the survey by clicking the **Redo survey** button.

Figure 7: Course Dashboard Student

3.4.3 Survey

The first page that the student will see if this is the first time they are using the system is the survey page (Figure 8). On this page they can enter their study, house, spoken

language and gender. They can also take a Belbin Inventory Test if they haven't done so yet by clicking on the **Take Belbin Test** button. This will redirect the student to the Belbin survey page (Figure 9). The data gathered from these surveys will be used for the matchmaking.

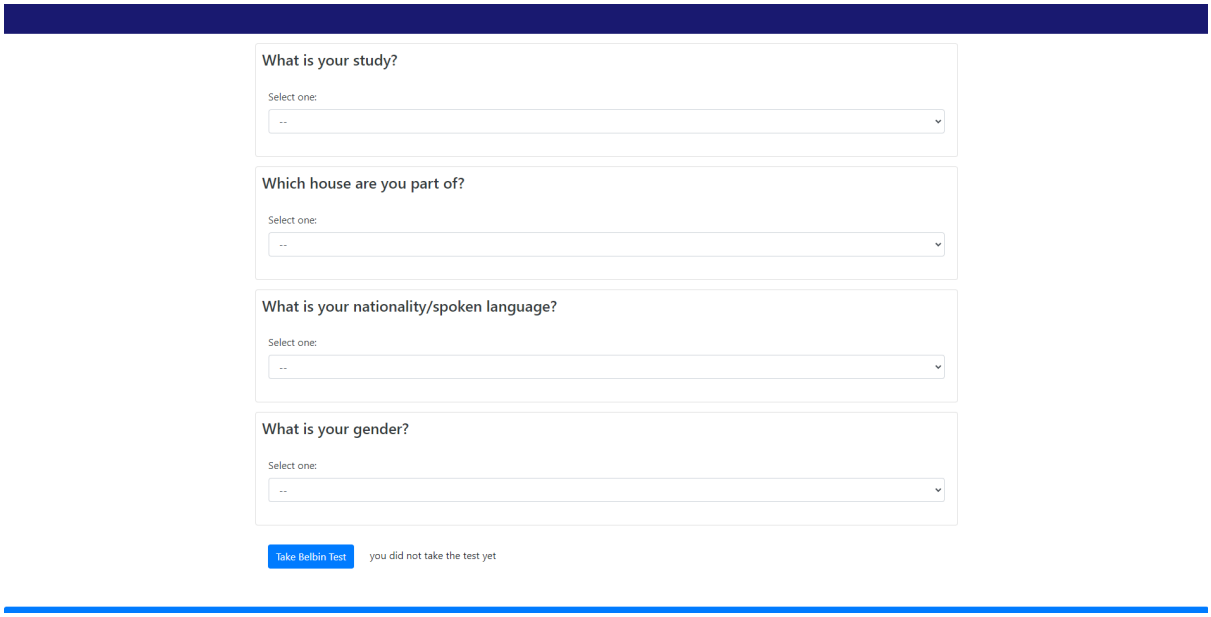


Figure 8: Survey Page

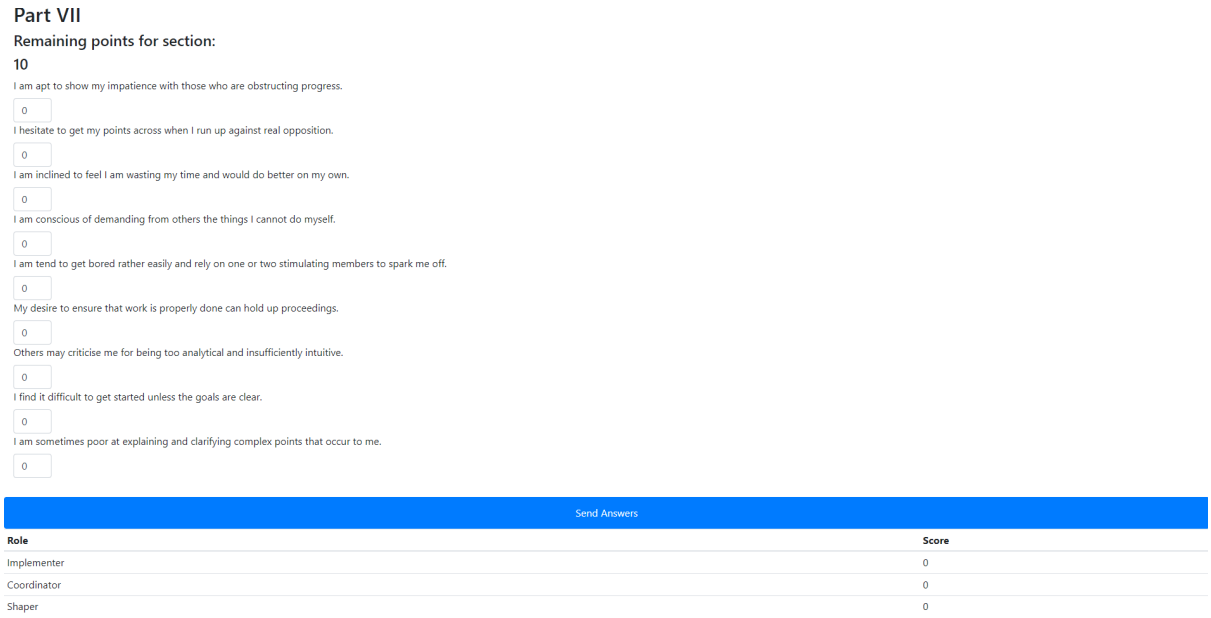


Figure 9: Belbin Survey Page

3.4.4 Course View

The course view shows up-to-date information regarding the selected course. You can access this page by selecting a course on the Course Dashboard page (Figure 7, 5). On

these pages both students and teachers can click on an unpaired student or a student inside a group, to view the student's profile (Figure 11c)

Teachers

Figure 10: Teacher Course View

On this page (Figure 10) teachers can see up-to-date information about the composition of the groups, alongside their score, as well as unpaired students on the right. They can search through the unpaired students by using the **Search bar**. Teachers can change group compositions by dragging and dropping students between groups, they can remove a student from a group by dragging them into the unpaired students section, and they can add a student to a group by dragging them from the same section into a group.

Teachers can also lock or unlock a course by pressing the **Lock/Unlock** button, which will prevent students from changing groups, they can save the current team compositions by pressing the **Save** button, and they can download the current team composition as a .csv file which can be uploaded to Canvas by pressing the **Download (.csv)** button.

Besides this they can automatically match students by pressing the **Match teams** button, which will open a pop-up (Figure 11a) where they can choose which matchmaker to use. The loading indicator (Figure 11b) shows that the matchmaking is working.

Figure 11: Match teams and Student Profile

(a) Matchmaker selection

(b) Loading indicator

(c) Student Profile

Students

On this page students can also see up-to-date information about the composition of the groups, alongside their score, and the searchable list of unpaired students. At the top of the page students can also see whether the course is locked or not. If it isn't, then they can leave or change their group by clicking the **Leave Group** button or clicking on an empty spot in a group.

Figure 12: Student Course View

3.4.5 Matchmaking

The system offers two matchmakers:

- Random Matchmaker - fills a group by picking a random student. The purpose of this matchmaker is to illustrate how much the genetic matchmaker improves the score.

- Genetic Matchmaker - fills the groups by choosing the best members to maximize the score that group can get. It uses an evolutionary algorithm as described in [1] to do so.

Heterogeneous - Consisting of dissimilar elements or parts; not homogeneous.

Research shows that heterogeneous teams tend to perform better [2,3]. In this system we chose 3 criteria to clasify students and ensure heterogeneity:

- Language - Teachers at University of Twente prefer to have multinational groups of students, and usually they set a limit on how many people can speak the same language in a group. We reward groups that don't break this limit, and punish groups that do. Moreover, research shows that multicultural groups tend to promote behavioural and cognitive engagement [2].
- Gender - Teachers at University of Twente prefer to have groups of mixed gender. And again, research shows that teams of mixed gender show increased productivity [2].
- Belbin roles - These roles prove to be an effective way to identify strengths and weaknesses of a student, and they can be used to categorize students. As seen in [1] having at most one person that naturally plays a given Belbin role in a group can be a good criterion for creating effective groups.

The group score is a number from 0 to 100 that represents the heterogeneity of the group according to the criteria specified above. We weight some criteria more than other, and by default the weights are distributed as follows:

$$Language > Belbin > Gender$$

4 Detailed Design

4.1 Look and Feel

Before anything was implemented in the application itself we made figma mock-ups (F) that had early designs of the course view, survey and student dashboard pages. Having these early designs were a big help in keeping the design uniform and consistent. These designs were mostly implemented in full, some features ended up being scrapped such as filters and profile pictures, but they ended up being a close representation to the final product.

The front-end of the project was created with the goal of simplifying and streamlining team creation. Our priority was creating a responsive and quick application, as making teams using the application is meant to be more accessible and faster than any previous method our users had available. For this we utilized AJAX and REST making any calls to the back-end seamless for the user. Furthermore, following this design philosophy we designed the looks of the website as well, names pop out when they're hovered over, buttons give immediate visual feedback, dragging students towards and between teams has a slot-in effect, clicking on any student immediately returns a modal with their

information. All these design choices come together cohesively to create an experience that never leaves the user wondering if the web-app is stuck or malfunctioning.

Another aspect of much consideration was the colour scheme, our aim was to create a modern, minimalist design that does not clutter the screen. For this a small sample of colours were used, a gradient of green, yellow, and red, to represent the expected performance of teams, and our signature navy blue used for elements such as the banner or empty teams, to create a uniform contrast to the white background of each page.

All pages, excluding the survey page, follow a similar design to give the user a similar experience, a banner, with some functionality at the top, with the content on the left and any information displayed on a column on the right. We choose to place most functional buttons on the banner at the top, while only leaving basic functionality in the content and information windows, such as dragging and dropping students for teachers, joining teams for students, or clicking on a user card for more information for any user.

Lastly, our last core goal was to make information easily accessible, for this we employed the aforementioned information cards that pop up whenever a student is clicked, and our team score displays for both students and teachers. We choose to sort the teams on the teacher side in order of the best scoring team, as we believe teachers would prefer an overview of how the course is evaluated as a whole. While students get a colourful mix with teams being displayed in the order they were created, so when students are given the option to create teams themselves, they are more fairly presented with options.

4.2 System Architecture

For our project we used the Spring Initializr website which makes it very easy to create a Spring Boot project with all the dependencies you want. We choose to use Maven for our project, because of the experience we have and the easy of using it. Maven is a build automation tool that provides a lot of functionality for building projects. The main one being that libraries are specified in a POM file that will make sure that on every computer where you open the project it will download all the correct libraries needed with the specified version, so you do not have to add them manually for each project and makes sure everybody is using the same version. When the project was created we setup a Git repository for the project using the universities Gitlab. Git is a version control system that makes it easy for multiple people to work together on the same code, but also gives to option to revert back to older versions of your code.

For our sever we are using a DigitalOcean droplet which is just a VPS(Virtual Private Server) that we are running Ubuntu Docker 19.03 on. Which is just a version of Ubuntu 19.03 where docker is preinstalled. Docker is a containerization platform, which means that it allows developers to easily package applications into a container. This container is a complete package that contains everything the application needs to run and can be deployed anywhere. With this approach you can run multiple docker containers on the same server each for there our own application. For example also our database runs in a docker container. Because we are using Maven it has a build in command to build a docker container for our project without needing to set anything up. This container

would be setup with a Tomcat web server. Tomcat is a web server for Java programs and makes it very easy to setup a web server for your Java application, this is also used by Spring Boot This container we would upload to our Gitlab container registry and can then be downloaded on our server and started. For handling the communication between the docker container and the internet we decided to use NGINX. NGINX is a web server and reverse proxy server. The second part is what we used it for. Our docker container with the project only runs on localhost:8080 but cannot be accessed from the internet. To make sure we have a secure connection we used NGINX as a reverse proxy where all requests first go through NGINX and get send to our container after. NGINX is also where our SSL certificate is provided to our URL. The URL we decided to use was smartteam.utwente.nl, to get this domain we had to contact LISA and do a domain request. When we got approved all they needed was the IP of our server and our server was connected.

For creating our web application we decided on using the MVC (model-view-controller) model. This model a commonly used model that divides the programming in three parts. The models are the classes we have for Student, Teacher, etc. Then the view are our HTML pages with same build in pages from Spring Web and the controller are the controller we have to link URL's to the right web-page and the REST controllers we have for requesting the models from the database.

Class Diagram

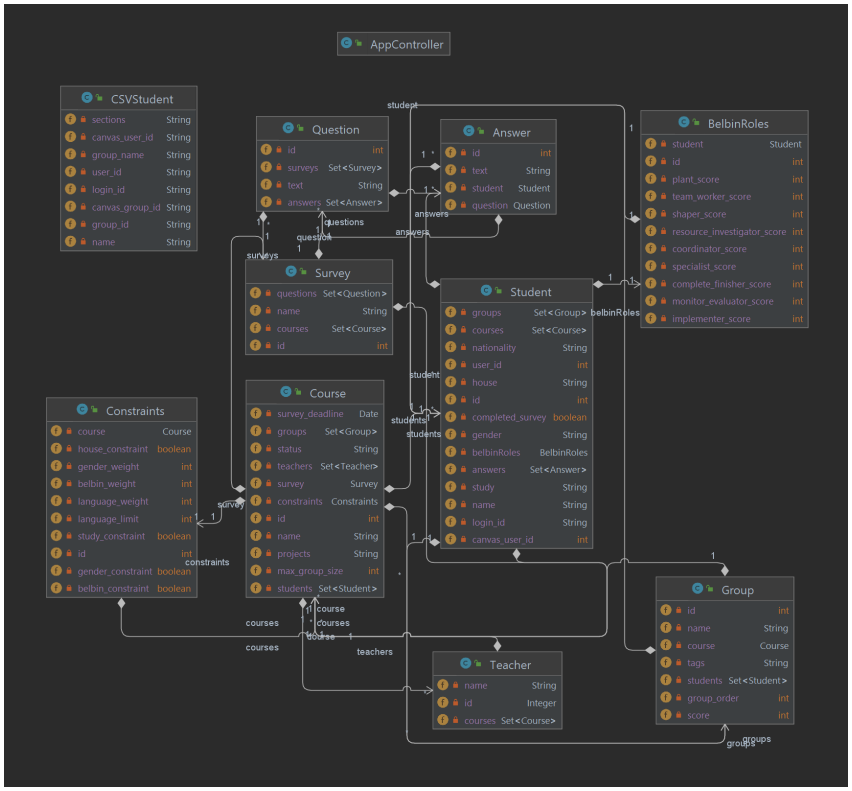


Figure 13: Part 1 of Class Diagram

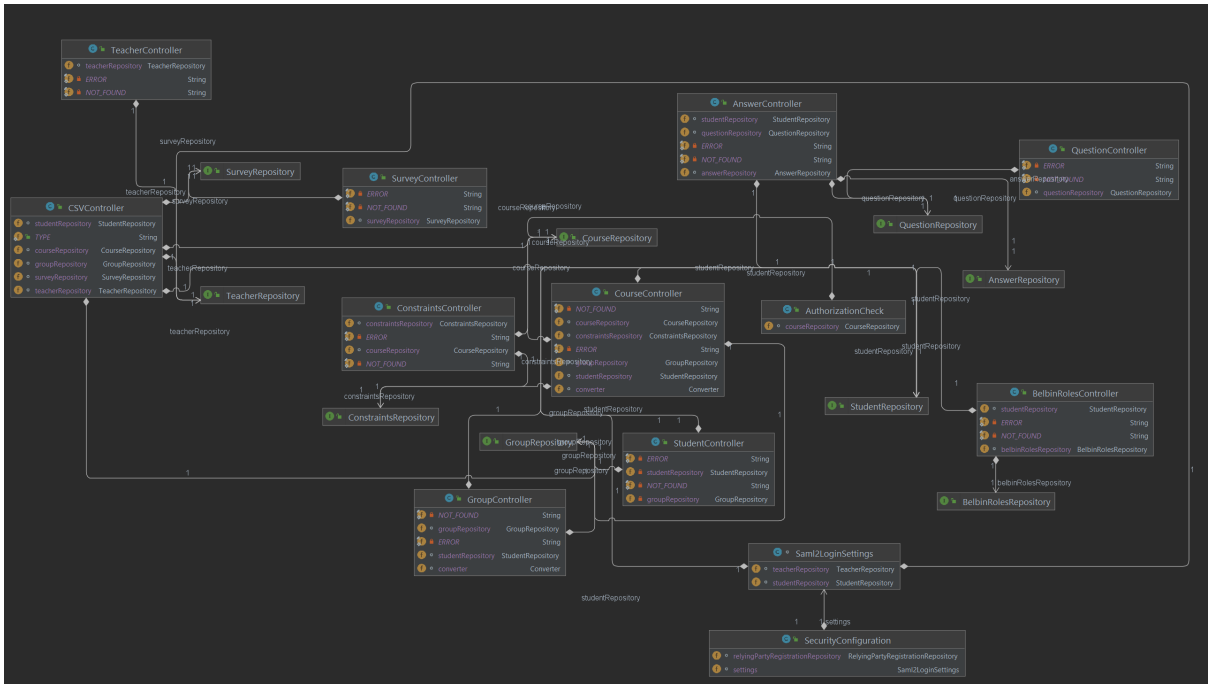


Figure 14: Part 2 of Class Diagram

4.3 Database and Data Access

Postgres

As discussed in previous sections, multiple features of our system employ user data. For example, the matching of students into groups makes use of data such as gender or Belbin roles. We get this data by directly asking it directly from the students trough surveys that can be launched at the inception of each course. Once the student gives answers to these questions, these answers need to be saved in a database. The Relational Database Management System (RDBMS or simply DBMS) that we are using in this system is PostgreSQL, also known as Postgres.

The reason we chose this particular SQL architecture is related to a few key advantages. The main advantage of Postgres is its wide acceptance and compatibility with a plethora of systems. Because of this, it was simple to create the connection with Spring Boot. Furthermore, as the system is designed to be used by multiple courses that contain hundreds of students each, we chose Postgres because it offers extensibility and scalability. The amount of data that is stored is expected to grow as the system gets deployed first programme-wide and subsequently University-wide. We, thus, needed to make sure that the DBMS could handle such growth. Additionally, Postgres is free and open-source, fact which insures that there will be no economic incentive that would require a sudden change in systems.

More concretely though, we created a database schema that includes 13 classes which efficiently store the data necessary for the features of our web-application. Firstly, a student and a teacher need to log in to the system. Since they do this trough the University Microsoft account, the system does not need to store any usernames or passwords, hashed,

secured or otherwise. The student class is the most important class as information stored in this class is used in virtually every view. The class has the student number, without the "s", as an integer primary key. Other information that we store about the student is their gender, house, nationality and study, which are all use in the matchmaking process. Other information used in matchmaking is stored in the "belbin_roles" class, which has a one-to-one relationship with Student, as insured by the use of student_id as both primary and foreign key for the belbin_roles table. Our application is supposed to produce project teams and the way we store them is with the table based on certain criteria. The criteria is stored in the table constraints. The constraints are attached to a unique course, fact which allows the teacher of that course to customize exactly what he wants to take into consideration when creating a team. Each group has a primary key that is generated at its creation and, importantly, an integer "score" attribute which stores the effectiveness score of the respective group. This score is not initialized, but is instead calculated on the back-end, by a specialized method in the back-end. The tags field will be useful in future versions for tag system that the students might use to improve the matchmaking process.

Students have a many-to-many relationship with groups, as a group contains multiple students and students can be part of multiple groups (e.g.: from past courses, or if they take part in multiple concurrent courses). Each group is associated to exactly one course trough the use of the foreign key course_id. The table course stores a number of important details, such as the survey deadline, after which the survey answers will not be logged anymore. The teacher's information are not exactly relevant to our system so we only store their name and employee id. We do, however, need to know which courses they have access to. For this reason, we used the table teaches_course which employs two foreign keys to connect the teacher and course tables. The student table is analogously related to course, with the takes_part_in table.

While the survey is static in this iteration of the system, we designed the database to scale as the app evolves. To this end, we made sure each course uses exactly one survey, but the same survey can be used by multiple courses (one-to-many). This survey can contain multiple questions, but questions can be reused in multiple surveys(many-to many relationship described by the association class contains_question). Finally, each answer is given by exactly one student to one question. We made sure this is the case with a combination of a primary key answer_id and to foreign keys, one which references the student table and one which references the question table. In this fashion, the surveys can be modular and designed dynamically by the teachers.

An overview of the schema can be found below:

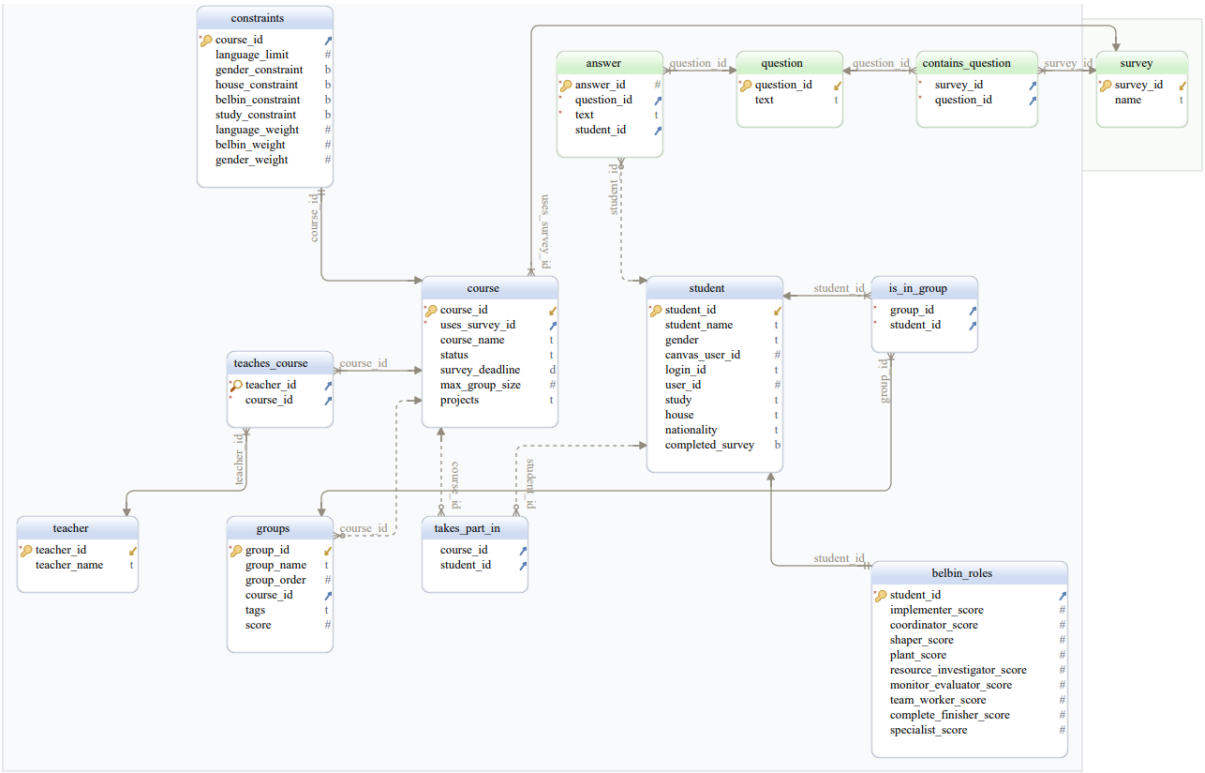


Figure 15: Diagram of database architecture

JPA

For accessing our database we choose to an ORM(object-relational mapping) called JPA(Jakarta Persistence API). JPA is concerned with persistence, which can be described as a mechanism that makes it so that Java objects can outlive the application process that created them. Not all Java objects have to be persistent, but with JPA you can choose on which objects should be persisted and even how. JPA is used with databases like PostgreSQL. What JPA does is giving a Java object representation for a database table. So for each database table you build a Java class that reflects that table. So each column will become a variable and table relationships can be represented by for example having a set of students in your course class. Each of these objects that will be in that set will present a row from the student table in your database. This makes using database information in the Java back-end a lot easier and automates a lot of functionality that normally would have to be coded. Using JPA almost eliminates the need for having to make SQL statements, this now only has to be done for specific cases where JPA does not have a build-in solution. In our application this made it very easy to use information from the database in our methods with being able to use Java objects to retrieve values, change values and create entries in the database. Just creating a Java object and saving that object with JPA puts the information set in the object into the database. We used Spring Data JPA for our back-end which made integrating JPA in our project even easier, because the library is build by Spring we know that all the functionality will be compatible with our other Spring libraries and makes setting it up very easy. And for creating the connection to the database we used the Spring Data JDBC library that

makes the actual connection to the database and is used by Spring Data JPA. Using the Spring Data JPA we can use annotations for setting up the models that will represent that database tables. These models can be found in the Class Diagrams under figure 13 and 14. These annotations are used to specify information that is needed to map all the classes and variables to the right table and columns, also by using build-in annotations we can for example modify how a set should be sorted or how a variables relationships to another table should be setup. This makes it so you don't have to write a lot of code for achieving a lot of functionality.

REST API

The connection between the front end and back-end has been build by using REST (representational state transfer). REST is a set of constraints that can be used by an API to create a REST API that uses HTTP request to access data from the back-end, but also send data to the back-end. It makes it easy to communicate with each other by being sure that the API calls you do will always have the same format. The advantage of using a REST API is that almost all programming languages can work with it and supports it. Some design principles that REST uses are:

- An uniform interface, which means that all the API requests for the same resource must look the same. So using a specific URI(uniform resource identifier) should always return the same piece of data, like when we request a student with id 1 at /api/students/1 it should always return the same student no matter from where the URI is called.
- The server applications and client applications should be completely independent of each other. Which means that the client only should have to know the URI for the requested resource and from the server side sending the request should not modify the client in any way other then giving it the requested data.
- A REST API should be stateless which means that that one request should have all the information that is needed to process it. We should not be dependent on multiple request to complete a REST call. REST APIs use HTTP communication to perform the standard CRUD functions, creating, reading, updating and deleting on a database. A basic example would be the use of doing a GET request which is intended for retrieving information, like the URI specified before /api/students/1 would return the student, POST to create one, PUT to edit a student and DELETE to be delete a student from the database.

For our application we mainly used JSON(JavaScript Object Notation) to represent the data from the database, because it is easy to read and work with in both JavaScript and Java. Also because we are using the Spring Web library we can easily build a REST controller by using annotations. By making methods for our REST API and specifying the URI and the information that we expect to get from the front end we can bind this to variables and use this in our code. In combination with JPA we can use the JPA models we build before to represent the data in JSON. This is done automatically by Spring and makes it so you can just return a Java object that will be represented in JSON. This also works the other way around, where if the client send a JSON that matches the model

we expect on the server it will be automatically converted into the right Java object and can be directly used as that object with the correct information set. For this we use the REST controllers that can be found in the Class Diagram under figure 13 and 14.

CSV Converter

Because wanted to make it easy for teachers to create a Course in our system we decided to use the Canvas csv file as a way of importing all the students that are in a course. For this we had to build a converter that would take a csv as an input and convert it to our JPA models and save it to the database. By using a CSVStudent model(visible in figure 13) that would represent each row from the CSV file we would be able to first convert the CSV file to a list of CSVStudent objects and then from these objects create JPA Student objects to save in the database. These students would be automatically assigned to the created course that will also be saved in the database and based on the given parameters the right amount of empty groups will also be created and added to the course. So with this one method a teacher would be instantly able to create a course in our system and start using it. To be able to create a csv file from a Course object the opposite has to happen. The object is retrieved from the database, all the Student objects get transferred to CSVStudent objects where their associated group id and group name will be added to them and then these objects will be transferred back to a csv file. This file can then be used by a teacher to import the groups into canvas.

Matchmaking Converter

For being able to use the matchmaker we need to be able to convert our JPA models to the models used by the matchmaker. To do this we build a converter class(figure 14 that has multiple methods for converting a Course, Group and Student to the models used by the matchmaker. So what it does is take the required values that are needed from the JPA models, to create the matchmaking models. This class is also used for computing the scores of a group, because the groups scores get calculated in the matchmaker to be able to get these scores in the database and show them on the front end we made an easy method that when you give it a JPA Group model will return the score. The reason for using different models is that we decided to make the matchmaker a standalone system that can also used separately from our application and does not depend on having the same structure as the database.

4.4 Authentication and Security

Authentication and security are important aspects of every web application that is accessible from the internet and has functionality that not every user should be able to use. Because of this reason we implemented an authentication system with method level security to be able to specify for each web page request and REST request who should be able to access it. For our authentication we quickly decided we wanted to use the

universities Microsoft login. We chose to use this, because our application is build for the university we do not have to keep our own database with usernames and passwords and make it very easy for teachers and students to start using our application. To use the Microsoft login we used the Spring Security library which has a lot of build-in features to secure web applications without much effort. For connection our application to the universities Microsoft login we decided to use SAML(Security Assertion Markup Language) which is a standard for exchanging authorization and authentication identities between security domains. It enables people to use SSO(single sign-on) where you have one account that can be used for multiple web applications. It also is able to provide some user information like for example in our application we can get the id of the user and use this in our application. For setting up SAML you have to exchange metadata between the SP(service provider) that is our application and the IdP(Identity provider) this is the universities Microsoft server. We contacted LISA to request access to the test IdP first, they asked us to fill in a request form with the metadata. When our request was processed LISA provided us with metadata that we had to import and after this a connection was established. Later in the project we did a new request for access to the production/normal IdP that all university applications use. For this we only had to import the new metadata that they send. Now that we have the authentication we can use this for our security. When an user logs I not our application we first check if they are a teacher or student. Based on this they will get assigned the corresponding authority that we can use in our method level security to check if they should be able to access. We also use this authority to redirect an user to their corresponding page, teachers to /teacher and students to /student. Also when a web page or REST request is received by the server it will first check if the user requesting it has the right authority and for some methods we added some extra checks. These checks are there to make sure that, for example a student is not trying to access a course that it is not in or tries requesting or sending certain data for another user. We applied these checks also to the teacher side, because we tried to make sure every user does not have more access then needed. To do this we again used Spring annotations that make it very easy to set the security checks per method without much code.

4.5 Course Dashboard

The dashboard is the landing page for any user logging in into the system, here an AJAX call is sent to the back-end to retrieve information about the user, such as the courses they are included in. This information is sent by the back-end as a list of course objects represented in JSON.

4.5.1 Teachers

Teachers however, have a more interactive dashboard, as they are able to create and delete courses, this is done utilising the **Create course** button on the page banner, which opens a popup modal object. Then the course name, team size and the maximum number of people who speak the same language in one team, must be included as parameters for

the course. Furthermore, a csv file, exported from canvas, must be attached to the course creation modal. Then, after the **Create course** button is pressed in the modal, all of this information is stored as a formdata object, which is then sent to the back-end as a AJAX POST request. The POST request has the formdata object as request parameters that get assigned to objects in the receiving method on the back-end, the csv file returns a course object with all the students as student objects, and the teacher that uploaded the file is added as an object as well. Furthermore, groups objects are then created with the given parameters and in the number required to contain all the students in the course. This method then saves this course object with all of its children objects and saves them to the database. After this process on the front-end an alert is displayed, informing the user that the file has been uploaded successfully and then displaying the course object on the front-end. Teachers then have the option to either enter the course, which simply redirects them to the teacher course view associated with the course id. Another option teachers are given is to delete a course that is no longer in use, or simply has been generated incorrectly, this is done through an AJAX DELETE request to the back-end, which then removes the course associated by id and all of its groups, from the database however, not students, rather disconnecting them from the deleted course. After which the front-end is updated to no longer display the deleted course.

4.5.2 Students

On the student side, when the student dashboard is entered, a check using an AJAX GET request to assure whether the student has completed the survey and if not they are rather redirected to the survey page. If the student has completed the survey, they are greeted by a page, displaying all of their associated courses and information about the student themselves, which has been collected through the aforementioned survey. This information is retrieved from the database using AJAX GET requests. In the page, the students have limited options of interaction, namely being able to go to the survey page, to update or change any inaccurate information or simply enter any courses they are associated with, being redirected to the student course view associated with the course id.

4.6 Survey Page

When entering the survey page, the student is presented with a series of questions, if any of them have already been answered, their answers are displayed as well. These answers are obtained using an XML GET request. The possible answers students are able to enter are sanitised on the front-end and back-end, only allowing a predefined set of answers. Before being able to submit the answers the application performs a check on whether the student has completed the belbin roles test, which is accessible through a button on the survey page, redirecting the user to a similar survey page, focused only on belbin roles. In this page, students are able to take a test, to determine their belbin roles, the front-end ensures that the obtained results fit the belbin criteria. Such as, making sure every section's answers add up to at most 10 points and that the total does

not exceed 70, this is done through simple JavaScript logic. Lastly, once the belbin test has been filled out according to the requirements, it is then sent to the back-end and saved into the database utilising an XML PUT request. If the answers in the survey page are all filled in according to the parameters and the student has completed the belbin role test, they then may submit their survey answers to the back-end, using an XML PUT request. The back-end creates answer objects, which can then be saved into the database, however if some answers were already present, the objects are updated rather than being remade. This also flags the student as having completed the survey and they are no longer redirected to the page when trying to access the student dashboard.

4.7 Course View

The course view pages give a visual representation of all the groups and their members and score for a specific course, which is indicated by a course ID in the course view URL, this is attached to all requests sent to the server from the course view page as a path variable. This information is obtained by sending three AJAX GET requests, to obtain the status of the course, to obtain the information of all the groups in the course and to get a list of all students currently not in groups. For these three calls the information is simply fetched from the database and sent to the front-end in the form of objects represented in JSON format. Utilising this information, representations of groups are created on the front-end using jQuery to generate HTML elements. Another method called colorGradient is used to generate a colour representation of the team score. The students are represented as clickable elements, which can be used to bring up a popup modal, which executes a AJAX GET request to obtain information about the student associated with the clicked element by the associated student id. This information is fetched from the database and sent to the front-end as a student object represented in JSON. Furthermore, the list of students not in groups is represented in an HTML table element on the right hand side of the page, This information is retrieved with one of the three aforementioned set-up AJAX GET requests, to obtain all the students not currently in groups rather than using the built in call in JPA, a custom SQL query had to be implemented. It is further possible to search and filter these students, utilising a search bar, which by using JavaScript removes any student elements with text not matching the text currently entered into the search bar.

4.7.1 Teachers

The teacher course view's main distinction from the student course view is the ability to manipulate the teams and the four buttons in the banner element, namely: **Save**, **Lock/Unlock**, **Download(.csv)** and **Match groups**. Teachers are allowed to manually manipulate groups and move students into or from groups, by simply dragging and dropping student elements from the student list into a group element or vice versa. This is achieved utilising jQuery, by making every element with the class .connected-sortable, have the jQuery attribute .sortable enabled. This is why unlike the student course view,

where groups are represented as cards, the teacher course view represents groups as tables, as table bodies can be connected using jQuery allowing for table rows to be dragged from one table to another. The **Save** button allows for the changes that have been made manually utilising the drag and drop functionality, to be saved into the database. This is achieved, by firstly, parsing all the student elements stored in their respective group table elements utilising jQuery. Then a map is created mapping every student id in a group to its respective group id. This mapping is then converted into a string which is a JSON representation of the mapping. This mapping is sent to the back-end using a AJAX PUT request, which uses this map and goes through all the group ids and gets the representing object from the database, then the students ids are used in the same way to retrieve their objects and add them to the corresponding group object. After this is done the changed group object will be saved to the database, sending an HTTP OK, back to the front-end. Once the front-end receives the HTTP OK, it refreshes the page to display the changes in group score and group formation. The **Lock/Unlock** button utilises the status request done in the set up, to change it's label, when the page is first loaded it has a label of **-**, however once the status is obtained it changes to Lock if the status is the string "OPEN" or Unlock if the status is anything else. If the button is clicked, the application stores the string "OPEN" when the button is set to Unlock or "CLOSED" when the button is set to "LOCK", into a formdata object. Which is then sent to the back-end utilising an AJAX PUT request. The back-end uses the path variable provided in the call to retrieve the associated course object, set the new status and save it to the database. The button tag is then set to the inverse of what it was when clicked using JavaScript. The **Download(.csv)** button allows for a csv representation of the course view to be stored locally and due to having identical structuring to the canvas .csv files it can directly be uploaded onto the canvas platform to generate the teams there. Once clicked an AJAX GET request is sent to the server, it has a preset data type of "text/csv". On the back-end a temporary .csv file is generated and then sent as a return value to the GET request. The back-end instead of going from a csv file to a course object does the opposite. It gets the Course object from the database using the path variable as an id and transfers the students and into a csv file, while retaining the groups that they were in. Once received on the front-end it transforms the received file into a blob object which is then saved on the local system as a .csv file. The **Match groups** button, is used to match the students into groups utilising one of our two matchmaking options, either the genetic matchmaker or the random matchmaker. Once the button is clicked, the front-end sends an AJAX POST request to the server, this request includes the type of matchmaker as a string, either "random" or "genetic" as well as the weights to be used for the matchmaker namely "language", "belbin" and "gender" and their respective values as integers, these are all stored as a formdata object. On the back-end the matchmaking process is initiated using the aforementioned Matchmaking Converter class and once done the new group objects are stored in the database, while a HTTP OK request is sent to the front-end. Once received the page is reloaded to display the newly generated teams.

4.7.2 Students

The student course view facilitates students joining and leaving groups as well as obtaining information on their group members and other people participating in the course. The student course view only allows for students to interact with groups, when the course status is set to "OPEN", when the page is loaded a AJAX GET request is sent to the back-end to retrieve the status of the course. The back-end uses the path variable to obtain the course object, then gets the status of it and returns that value. If the course status is not "OPEN", text is displayed on the banner indicating that team formation has closed. For students to join a team, card elements are generated to represent each group in a course, this is parallel to the group table elements in the teacher course view, with a few key differences. The card has a numbered row for every available slot in the group, each of these rows is a span element, which has a .clickable parameter, once clicked if it is not currently occupied by another student an AJAX PUT, request is sent to the back-end, with the group id of the group element as a path variable. The back-end when receiving the request uses the group id to obtain the group object from the database and does a couple of checks before letting a student join. It first checks if the course has status "OPEN" after that it checks if the student is in any other group at the moment, if this is the case it will first remove the student from the old group and add them to the new one or if the student had no group yet just add them to that group. Also when joining a group the new score gets calculated so also when you get removed from an old group that score gets updated as well. If the back-end responds with an HTTP OK, the page is reloaded to display the new status of every group. If the group that the student is joining is full, the back-end responds with an HTTP ERROR and the page is also reloaded to show why the join request failed. If students wish to leave a group, they have two options, they can simply join another group or they can utilise the **Leave group** button, This button once pressed, sends a AJAX PUT request to the back-end. On the back-end the group the student is currently in gets fetched from the back-end. The course that the group is connected to gets checked if that has status "OPEN" and if this is the case it will remove the student from their current group and update the group's score. When the student is removed from the group in the course they are currently in, once the front-end receives an HTTP response, the page is reloaded to display the new status of groups. If the student attempts to leave a group while the course status is not "OPEN" or they are not in a group, they are informed of this via an alert on the front-end.

4.8 Matchmaking

The goal of the matchmaker is to automatically make groups that have a high score overall. In order to design a matchmaker we had to define a formula for the score of a group, and in order to do so we had to choose which criteria do we care about when forming groups. To achieve this we conducted research into this topic, as well as hosted interviews with our stakeholders - teachers to see which criteria is relevant for the groups in their modules.

4.8.1 Research

As mentioned in 3.4.5, heterogeneous teams tend to perform better. We arrived to this conclusion after reading 7 paper regarding this topic:

[1] This was the most important paper of all. It showed that using Belbin roles to create effective teams works. The created teams are heterogeneous according to Belbin roles, meaning that there is at most one person that plays a given Belbin role naturally. Ideally resulting in a team where each of the 9 Belbin roles is played by one member. This approach implies that we need teams of 9 people, which is not the case for us, so we had to adapt the method. This paper also shows the use of a genetic algorithm to solve the matching problem and provides a good insight into how it work. This information was used to implement our version of this algorithm.

[2] This was the first paper we read that supported the idea that diversity in a group correlates to better performance. This paper looked specifically at cultural diversity, which does apply to our use case since University of Twente is trying to have multinational teams as well.

[3] This paper again supports the idea that heterogeneous teams perform better. Moreover, they showcase an interesting approach on how teams can be formed. This approach proved hard to adapt to our use case so it was disregarded.

[4] This paper showed a way to solve the complex problem of matching by using meta-heuristics and student preferences. While we did have to take student preferences into account for this project, we used a meta-heuristic approach in our system as well.

[5] Provided some good points on what constitutes an effective team, and a way to evaluate a team. However this information is very outdated and does not apply to our use case.

[6] Did not provide anything worth noting. It was more about what a team can do to improve their performance and relationship, but not much about how to create the team

4.8.2 Interviews

Our main stakeholders are our supervisor and the teachers at the Technical Computer Science programme at University of Twente, so naturally we had to group students according to criteria that fits their needs and preferences. In order to find these we conducted interviews with 5 teachers summaries of which can be found in the Appendix:

Faizan Ahmed (C) In Module 1 students should be from different countries and not speak the same language. They should be from the same house, and must be paired with a new student every week.

Tom van Dijk (A) In Module 2 our system would be applicable since Tom uses the grades from the Python pearl which is sensitive information that he doesn't want to upload anywhere.

Piete-Tjerk de Boer (E) In Module 3 Pieter-Tjerk could use a system that helps students make more informed choices on who they team up with. This could be done

with the use of a tag system where students can specify tags on when they prefer to work, or if they prefer to work remotely etc.

Faiza Bukhsh (D) Module 4 is our main client. Belbin roles are a learning goal for this module, and our supervisor is a teacher responsible for the Academic Skills component in Module 4, so we can use Belbin roles as a criterion. The criteria used in Module 4 is: no more than 2 students speaking the same language, students should be from the same house, and there should be 4 TCS and 2 BIT students.

Rom Langerak (B) Module 12 has a similar situation as Module 3. They allow students to choose their own teams, and Rom would benefit from a system that allows students to make more informed choices.

We noticed that all teachers wanted diverse teams, diverse nationality and diverse genders. Modules that have teams of size 4 or greater would be interested in using Belbin roles to increase team efficiency, but modules with smaller teams would not find much use for this. Teachers who allow students to form their own teams, are not interested in our solution so it has to be adapted to let students choose their own teams and make more informed choices by showing them some data about the students they are grouping up with, or the predicted effectiveness of the group.

With this done we can finally choose which criteria we will use for this application, and it is language, Belbin roles and gender. We have disregarded the study criteria since it only applies to module 4. We also omitted the house criteria due to the time constraints of this project. However these 2 criteria are implemented on the back-end and are ready to be used with a couple of additions i.e. the functionality is there, we just did not have enough time to fully implement it.

Besides these interviews we also had meetings with our supervisor Yeray Barrios, and in one of them he showed us the way he was creating effective teams in Module 4 using a spreadsheet. This meeting was particularly useful since he explained the theory behind it and this helped us use the Belbin roles efficiently in the process of creating groups.

4.8.3 Framework

With the criteria chosen, we could finally start programming the framework that we will use to create the matchmaking. This framework is thoroughly documented, so we won't go into too many technical details in this report. Below you can find a class diagram of the matchmaker package, and we will now break it down class by class.

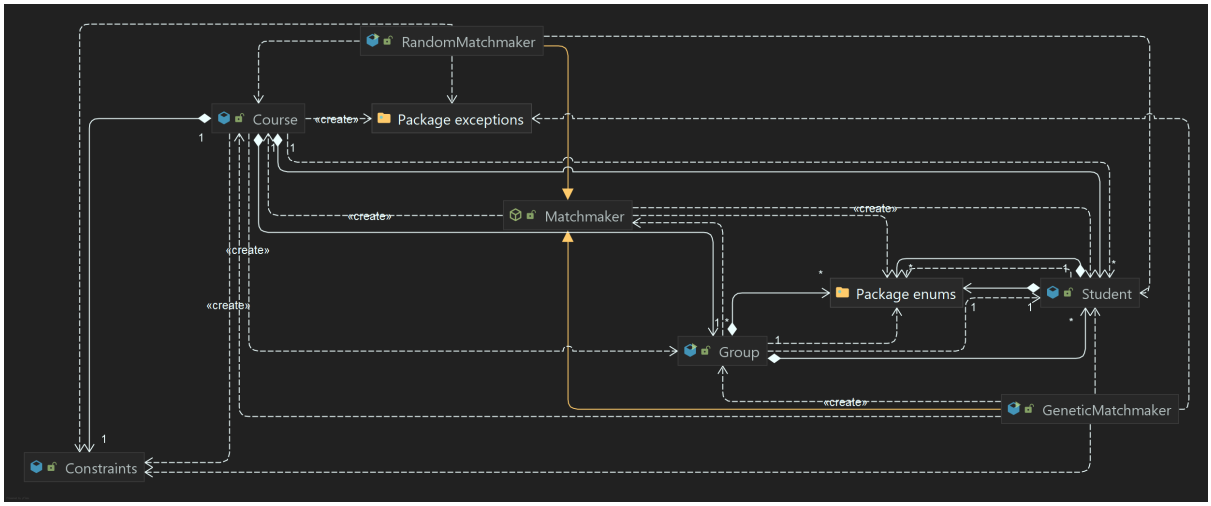


Figure 16: Matchmaking package class diagram

The criteria tells us what information do we need from the students, so we can create the student class. But before that we had to define some classes that will be used to store data regarding the students. These classes are stored in the enums package.

enums Package:
Language stores all the languages a student can speak.
Gender stores the genders a student can have.
Role stores the name of the 9 Belbin roles.
Study stores the study a student can be part of (TCS, BIT).
House stores the houses a student can be a part of.
Category stores the 3 categories of Belbin roles (Action, Thought, People).

Student class - represents a student and stores all the information needed about a student. The most important functionality in this class is the ability to automatically get the three dominant roles the student has from a Map of Roles.

Now that we have a way to represent students we can start putting them into groups.

Group class - represents a group of students. Important functionality is keeping track of the languages spoken in a group, of the Roles played in a group, and of the genders present in the group. This class also provides a way to calculate the score (KPI) of a group, and a way to adjust the weights for each criteria in this score formula. It also provides methods to add/remove members.

Having a way to represent groups and students we can now create entire courses in which we can store groups and unpaired students.

Course class - represents a course or module. It is the central class of this application. Important functionality includes:

- A convenient way to keep track of create groups, and number of groups yet to be filled.
- Various methods to create a group, access a group and remove a group.
- Convenient way to keep track of paired and unpaired students.

- A way to keep track of the current group that needs to be filled.
- A way to keep track of constraints a course might have, like maximum number of people who can speak the same language, if they want to use Belbin roles in the matchmaking, team size etc. This is done with the **Constraints** class.
- A way to compute the average score.

Matchmaker class - this class acts as an utility class and an interface that other matchmakers need to implement. It assures inter-compatibility between different matchmakers. It allows:

- To generate random Belbin roles
- To generate a random student
- To generate a random course
- To choose a random value from an enum (random gender, language)
- To export a course to a file
- To import a course from a file

Calling the *match(Course)* method on a course will match the unpaired students in that course, according to the specifications of the matchmaker that invokes this method.

RandomMatchmaker class - is an implementation of **Matchmaker** that randomly assigns students to groups.

GeneticMatchmaker class - is an advanced implementation of **Matchmaker** that maximizes the score for all groups in the course. We will explore this class in more detail in 4.8.5.

4.8.4 Group score

The group score is the root of our matching algorithm. This score can also be considered a diversity or heterogeneity score, since the more diverse each criteria that helps build it up is, the higher the score is as well. The three criteria that determine the group score are:

Nationality score

This score is based on the "languageLimit" attribute of the Group class. If the language limit in a group is exceeded, then no points are awarded for the language criteria. Otherwise, full points are awarded.

Gender score

This score is computed using a reversed Hamming Distance, which calculates how balanced the teams are in terms on gender. We have 3 genders: male, female and other. For the purposes of this project we consider the other gender as female. It gives full points if the number of males is the same as the females, and will decrease proportionally based

on how unbalanced they are (3 male 1 female will return 50% of the score and 4 females and 0 males will return 0%)

Belbin score

This score is computed by first iterating over all the Belbin roles and counting how many members of the team have this role as one of their three dominant roles. The dominant roles are the 3 roles that the student has the highest score for in the Belbin inventory test. We will define 3 functions:

$D1(Role)$ = counts how many people have the given Role as their first dominant role.

$D2(Role)$ = counts how many people have the given Role as their second dominant role.

$D3(Role)$ = counts how many people have the given Role as their thid dominant role.

After we counted this we can compute a score for each role that determines how much a role is played in a team. For this we sum the number of students that have this role as one of their dominant roles. We weigh the first dominant role more than the second, and the second more than the third. The formula for the score of each role is:

$$RoleScore(Role) = 3 * D1(Role) + 2 * D2(Role) + 1 * D3(Role)$$

Now having a score for each role, we separate the roles in three categories and sum the score for each category:

Action roles: Shaper, Implementer, Completer/Finisher
People roles: Coordinator, Team worker, Resource investigator
Thought roles: Plant, Monitor/Evaluator, Specialist

Having the total score for each category, we compute the euclidean distance between these three categories, and normalize the resulting score. It is a number from 0 to 100 that represents how heterogeneous the team is in regards to Belbin roles. Each criteria that is used for the computation of the team score has a weight. The weights must be at least 0 and will be automatically normalized, so more or less any ratio between the three can be realized.

4.8.5 Random Algorithm

SmartTeam provides the teacher with the option of creating teams randomly, without taking any criteria into consideration. Based on the group number, this algorithm takes random students out of the unpaired list and pairs them together until in teams until there are no more students left.

4.8.6 Genetic Algorithm

This section includes the whole process of developing the algorithm and all the decisions taken throughout.

Why a Genetic Algorithm?

The idea of creating a Genetic algorithm was introduced to us by our supervisor, Yeray Barrios Fleitas. After exploring the problem at hand, we found ourselves with a heuristic problem. If we wish to pair 6 students out of a course with 400 students, there are 5,478,557,838,600 different combinations in which the students can be paired. And that is for just the first out of 67 teams in this case. The Genetic algorithm seemed to be the best way to get results that are close to optimal in a reasonable time.

What is a Genetic Algorithm and how does it work?

A Genetic algorithm mimics the way children receive traits from both parents by mixing elements from two different sources. The more satisfactory an element is, the higher its chances to be selected as a source for creating the future generation is. The algorithm remembers the strongest out of each generation before overwriting the old population with a new one so that the result is the most satisfactory element out of all of the generations. There is a generation limit introduced to avoid an endless loop in a situation in which there is no perfect solution.

SmartTeam Genetic algorithm implementation

The algorithm implementation is based on the following steps:

- **An initial population is created:** Out of the whole list of unpaired students, groups of a predefined size are created and stored as arrays for easier data manipulation. The number of computed groups depends on the population multiplier constant. The higher the multiplier, the more groups are initially computed and the higher the chances are to find better teams.
- **The initial score is computed:** The score of each team is then computed and stored in an array. The groups with the two highest scores are saved so that they don't get lost in future generations if they remain the groups with the highest scores.
- **Groups are assigned weights:** Each group gets assigned a weight that is directly proportional with their score. The higher group's weight is, the more chances it has to participate in creating the next generation
- **Crossover:** Pairs of two groups are randomly chosen from a weighted list in order to create the future generation. The result of each pairing will be two new groups that have half of each parent group. This process is done until the new population reaches the size of the initial population.
- **Recalculate scores:** The scores of each new group is calculated and the best scores are compared with the old best scores. If any of the new scores is higher, that respective group takes either the place of the highest or second highest score group. The new population takes the place of the old one and and the process starts over until the generation limit is reached.

Once the generation limit is reached, the best group over all generations gets to become the group that gets added to the course. The process continues the same way with the

rest of the unpaired students until they are all paired. The last group just gets assigned all the unpaired students.

5 Testing

5.1 Web Application

For testing the front-end, we did not utilize any automated test classes, rather we hosted a local server, then accessing this server using our browsers, we used their inbuilt developer tools, while following the workflows we had designed for each user to test all of our implemented features. This testing worked for both front-end and back-end purposes as we implemented calls to the back-end through the front-end to be tested, then with verbose logging any issues were noted down and remedied. Another method for testing the back-end was simply utilizing the URL codes in the browser address bar.

The reason we decided not to utilize automated test classes is that we believed them to consume more time while being implemented than save in debugging issues. Our system has a small enough set of features that can be tested manually in a timely fashion.

5.1.1 Filling the Database with Random Data

As all the members of our team are students we did not have access to real students and to get them to answer surveys. Furthermore, in the early stages of development, there was no way to ensure that the data we would use in matchmaking would not be leaked. However, features still needed to be implemented and tested so, to solve this problem, we decided to generate dummy data to fill the database with. Python scripts were used for the generation of random data and the "csv" file format was used to store it, as it is compatible with both Python and the PostgreSQL server. Firstly, we needed to generate students and we decided that a set of a hundred would be sufficient for our purposes. Students have an id which is an integer and we generated with a simple counter. The names proved challenging to generate as we wanted them to look real. Luckily, there are online tools that generate random names. We used one of these tools and imported them into python. Each person was than assigned a gender between "male", "female" and "other", using the library "random" a sub-module of Python. In similar (random) fashion, each student was assigned a house("Blue", "Red", "Yellow" or "Green") and a study("TCS" or "BIT"). We made sure that all the data so far is distributed uniformly, in other word, there is an about equal amount of TCS and BIT students for example.

Generating nationality was done similarly, but we did try to simulate the situation of real life students. The closest model or approximation that we found comes from the "Paretto Principle", also known as the "80-20" rule. This principle states that 80% of outcomes come from 20% of all causes. In this case, we wanted to be as close as possible to the goal of having 4 fifths of students come from only 1 fifth of the countries. More concretely, out of 100 students, about 70 come from only three countries. The rest are

split between 5 other nationalities. This technique was necessary, to make sure that the matchmaker can handle edge cases and real life conditions.

5.2 Matchmaking

The matchmaker module was intensively tested with custom built methods. The challenging part was getting student data to test the matchmakers on. We resorted to creating random fake data for this purposes due to privacy concerns and because this would allow us to test more thoroughly and on various course sizes.

As mentioned in 4.8.3 the **Matchmaker** class has methods to generate random Belbin roles, students and courses. It does so by using a custom function that selects a random element from an enum. This allows us to select a random **Language, Gender, Study** and **House** for a student. Alongside the Faker library that can generate a fake name and random ID, we can create random and completely fake students. We have added readable and useful *toString()* methods for each class, which allowed us to manually test and debug each class we made. This was used to test the functionality **Group** and **Course** class.

Being able to generate random students allowed us to create courses of any size filled with random students. This proved very valuable in testing the matchmakers.

RandomMatchmaker was tested by checking if there are no unpaired students left, if there are no duplicate students in a group and if the program finishes executing correctly.

GeneticMatchmaker was tested with the generated student sets throughout its whole development procedure. As a result, constantly testing the matchmaking resulted in the following findings:

- The algorithm's results based on the generation limit stagnates at around 10. After 10, the results tend to be the same, but only take much longer to compute.
- The population multiplier has clear results between 1 and 5, and then very slightly improves the algorithm's results until around 9. Anything higher than 10 doesn't seem to provide any advantages.

The final **Team Score results** have seen a tremendous improvement in comparison to the random matchmaking.
For a course with 200 randomly generated students from four different countries, teams of six, a language limit of 2 and team score weights of 1:1:1 for language, Belbin roles and gender, the **average random scores** ranged between **28-35** and the **average Genetic scores** ranged between **72-75**. Each algorithm has been run 30 times each in order to compute these statistics.

6 Reflection

6.1 Individual Contributions

All members equally contributed to the Design Report and testing the parts they were responsible for.

Cristian Trusin - Research, Interviews, Matchmaker Framework, Random Data
Luuk Willems - Back-end, Database and Front-end connection
Tudor Nastase - Database, Survey, Random Data, Poster
Kristupas Raicevicius - Front-end, Connection to Back-end, Presentation
Cosmin Ghiauru - Research, Genetic Algorithm, Presentation, Poster

6.2 Limitations

Time

The first and most obvious limitation we faced is of course time, as this Design Project module was only ten weeks long, we were working on a tight schedule. As such we had to compromise on requirements, we had set out for ourselves as well as many nice to haves that would have improved the feel and flow of our system. Acknowledging this and moving forward we believe we made the decisions to focus on implementing truly critical features that allow for the application to function, while scrapping things that would have taken up a lot of our time and effort with little improvement towards our complete system.

Data

Currently we only get a limited amount of data when a student logs in, namely their student number and their email, while the rest we must acquire through the in-system survey. As such we did not have access to all the student data we would have preferred to have. For example, one of the features we had to scrap was asking for students expected grades to match them with people of similar commitment, but students might abuse this system to have an easier time with the project, if we were able to access their grades however this could be circumvented completely. Another great benefit of having access to more student data would be a simplification of the survey as most current fields could be removed and only the belbin role test would have to be taken saving users time when using the system.

Filters

The reasoning behind not implementing filters, which were included in our requirements analysis relates to how the generation of group elements is done, since groups are generated in completely exclusive div column elements, there is no simple way to implement the

display of the filtered teams, that is not just a highlight of the teams matching the filter, or by hiding all the teams not matching the filter as it leads to table and card elements being disconnected from view. As such we came to the decision to instead implement the filter feature in tandem with the tag system as filtering by tags would have been more intuitive and would justify a complex solution, however due to the aforementioned time limitations this feature had to be scrapped.

Custom matching

Allowing teachers to customize the matching algorithm was a feature we originally identified in our requirements analysis, however we were unable to find an elegant implementation that provides utility to the teacher, while also keeping their interface clutter free and intuitive. The problem arises when giving the user too much freedom it would be difficult to arrive at effective team formation, so while these kinds of features would be useful for research purposes, we did not believe them to be as useful in a practical use case. Which is why we decided to not allow teachers to freely pick criteria by which team scores are calculated.

6.3 Future Work

Tags

Tags would have been a great addition to our project, we had plans to implement both team and people tags that would give more information in a compact manner. Every person would be able to choose from a set of preset tags in the survey page. These tags could indicate how the user prefers to work such as online or in-person, similarly they could indicate when the user prefers to work – in the morning or in the evening. For groups any member could select tags to add to their group and some tags would be added automatically, for example a student could select a tag representing a project they would like to do in the module, this project tag would of course have to be provided by the teacher, some tags that could be auto generated, could be mixed-gender teams, full or empty. The reasoning behind implementing the tags is twofold, on one hand it gives more information to both teachers and students in a compact manner, on the other hand it allows for filtering according to tags allowing for easier access to desired teams.

House, Study and Previous Teammates criteria

All modules except Module 12 only allow groups where each member belongs to the same house. This feature is almost implemented (we are literally one day away), we do store and display information about the house a student belongs to, however we do not take this information into consideration when calculating the score of a team. If we did, then our genetic algorithm would have only made groups with members of the same house. Module 4 is the only module that requires teams to have 4 TCS and 2 BIT students.

Because it is the only one, this functionality was left to be completed last. The story is similar here, we store and display this data but it is not included in the score function. Module 1 and 3, use pairs of students every week to complete assignments, and besides the criteria we already use, a student cannot be matched again with a student he was previously partnered with. This feature can be implemented with the simple addition of a list in the **Student** class that will keep track of who this student was matched with, and reward points inside the score function whenever this criteria is satisfied. These features were scrapped due to time limitations, but they should be easy to implement in the future.

Social features

Social features would include things like an inbuilt group chat and direct messages as well as the ability to invite or kick people from groups. These features are of course not only not simple to implement they also pose a threat to degrade the experience of some of the systems users, which is why whether they should be implemented or not is still something that should be under careful consideration and requires at least preliminary feedback. On the other hand, it would also be very helpful for students, to be able to communicate and interact with each other without utilizing third party assistance, it would remove a big hurdle in forming teams when students are allowed to do so. Another benefit of having such social features is increased engagement with the system, there is reason to believe that being able to interact with other students would also act as an incentive for students to engage with the system.

Images

Images can be a powerful tool in conveying information and can be said to be lacking from the current version of the system. To give examples, the current course page uses a placeholder image that could be replaced with generated images containing the course name, to give more distinction for which course is which. More areas where images could be used are profile pictures to allow students to customize their pages. Displaying icons over teams to give a quick overview of what characteristics that team has, such as many speakers of the same language, or a mix of genders.

7 Appendices

A Summary Interview Tom van Dijk

Criteria: Similar level Pearl grades - Does not want to upload these anywhere since it is sensitive data) Nationality - As diverse as possible) Same House Gender - Diverse but no groups of 1 female and 3 males. Needs to be balanced e.g. 2F 2M Project groups are pairs of pairs Our system would not help Tom in M2. Belbin roles are not very useful in the context of a pair and matching them is not a primary concern in this module, experience is.

People will lie about ambition

Optimization/Complexity: Constraint programming Google constraint solver

Belbin Roles: Not applicable for M2, would be the last priority since grades are more important. Also not a good criterion for pairs, starts being viable at teams of 5+ people

Tom's Workflow: Sort dutch students according to grade on the left Sort international students according to the grade on the right Match dutch with international according to grade (similar experience) by hand

Edge cases: Match grades as close as possible Pairs of 1 student

Takeaways: Tom is interested in the system but will not end up using it in M2 since it will not save him much time and is not worth the trouble of risking sharing sensitive data. His priority in creating teams is having pairs of similar level, but still diverse in the sense of nationality and gender. Belbin roles would not help here since the module mostly uses pairs and not teams, and for 2 people it is not important to match the Belbin roles when you have more pressing criteria like experience and grades. Tom also warned us that students lie in surveys, and asking a question like: How motivated are you for this module? or What grade are you aiming for this module? Will lead to poor matches. He gave us advice on optimising the matching though: we should look into Constraint programming and Google's Constraint Solver (OR-Tools) however this begs privacy questions.

B Summary Interview Rom Langerak

Criteria

No criteria - Only the number of people per team

Ideas

Freedom for students to choose their own team according to a criteria like project preferences etc Constraints like avoid a certain project Suggestions for people to complete a team (i.e. have 3 ppl and need 2 more) See which people are free and they can invite them to a team See their profile See compatibility if they join a team

Optimization/Complexity

Perhaps contact people from the departments of Mark Uetz and Richard Buscheri if we get desperate

Belbin Roles:

Would be a good idea to let students see who they are choosing to group up with Tom's workflow:

Edge cases:

If they don't have a team - assign according to skill level/project preference

Interview students

MAKE IT HELP THE STUDENTS Let them choose which group they want to join if they fit in multiple (in mod4 for example)

Takeaways:

Rom is interested in the system for the utility it can offer students. Currently students choose their own team, if they have friends/acquaintances it is quite easy for them to form a team, but if they don't know anyone they are forced to use a Whatsapp group to find people (almost at random). Having a dedicated system where they can see which groups are still looking for people and which people are still looking for groups can simplify this process. Having access to the profile of a certain team or student can help them pick the best team according to project preferences or belbin roles. Also teams which are not yet complete for example 3 friends want to work together and need 2 more people in their team, can see which people are still not in a group, see their project preferences, belbin roles and invite them to join a team. Rom also had a suggestion for module like M4 where students get assigned to teams by teacher, but he says it would still be nice to have some freedom as a student to choose who you work with, so maybe we can allow the students that fit in multiple teams (due to skill level etc) to choose which one of those teams they want to join.

C Summary Interview Faizan Ahmed

Criteria: The pair should be from different country(as much as possible) Every week they must be paired with a new student(as much as possible) Students from different houses must not be mixed(as much as possible)

Edge cases: Minor and resit students are always an anomaly(since they are not in any house). I handle it two ways. Assigned them to a dummy house called Guest

However, this year I am just assigning them to one of the houses(so every week they join a different house).

Students make their own project teams so the suggestion from Rom applies here too

D Summary Interview Faiza Bukhsh

Criteria: 4 TCS 2 BIT Same house for TCS No more than 2 students with the same native language from both studies Edge cases happen when e.g. a House is 60

Belbin Roles: Belbin Roles are a learning goal of this module, students should be made aware of their use and existence. They serve as a suggestion and not a strict rule in making a team. So students take a survey and have the roles saved as tags to the profile. Teams were not broken up because their belbin roles did not match.

Edge cases: Split between TCS and BIT (use email address from outlook)

NOTES: History of teams that a student was part of - make the system so it is used to keep track of teams too. Canvas integration would be nice.

Takeaways: Faiza is very interested in the system. Belbin roles are a learning goal of M4 and she wants to make the students aware of their existence. However they are not used as a criteria for creating teams but rather a suggestion. She prefers letting the students make their own teams, since last year it proved to create less problems and more efficient teams. However, the teams still had to respect the criteria and many of them did not, so it led to a lot of manual work and breaking of teams. She wants this problem solved by enforcing these rules, BUT NOT TOO MUCH since it can happen that sometimes a rule has to be violated e.g. last year there was a house which was 60Kristupas' suggestion would solve this problem, that we let the students form bad teams, but we warn them about it and they can/have to request approval from the teacher.

E Summary Interview Pieter-Tjerk de Boer

Criteria: Number of people per team. All the people in the team use the same programming language. House.

Belbin Roles: Not applicable since students choose their own teams.

Takeaways: Pieter-Tjerk is not interested in the system as is. It would not help him since students choose their own team. However he is open to the idea of helping students make more informed choices on who they team up with, with the use of a tag system. Tags could be set by students and groups specifying when they would like to work, if they prefer to work remotely and so on.

F Mock-ups

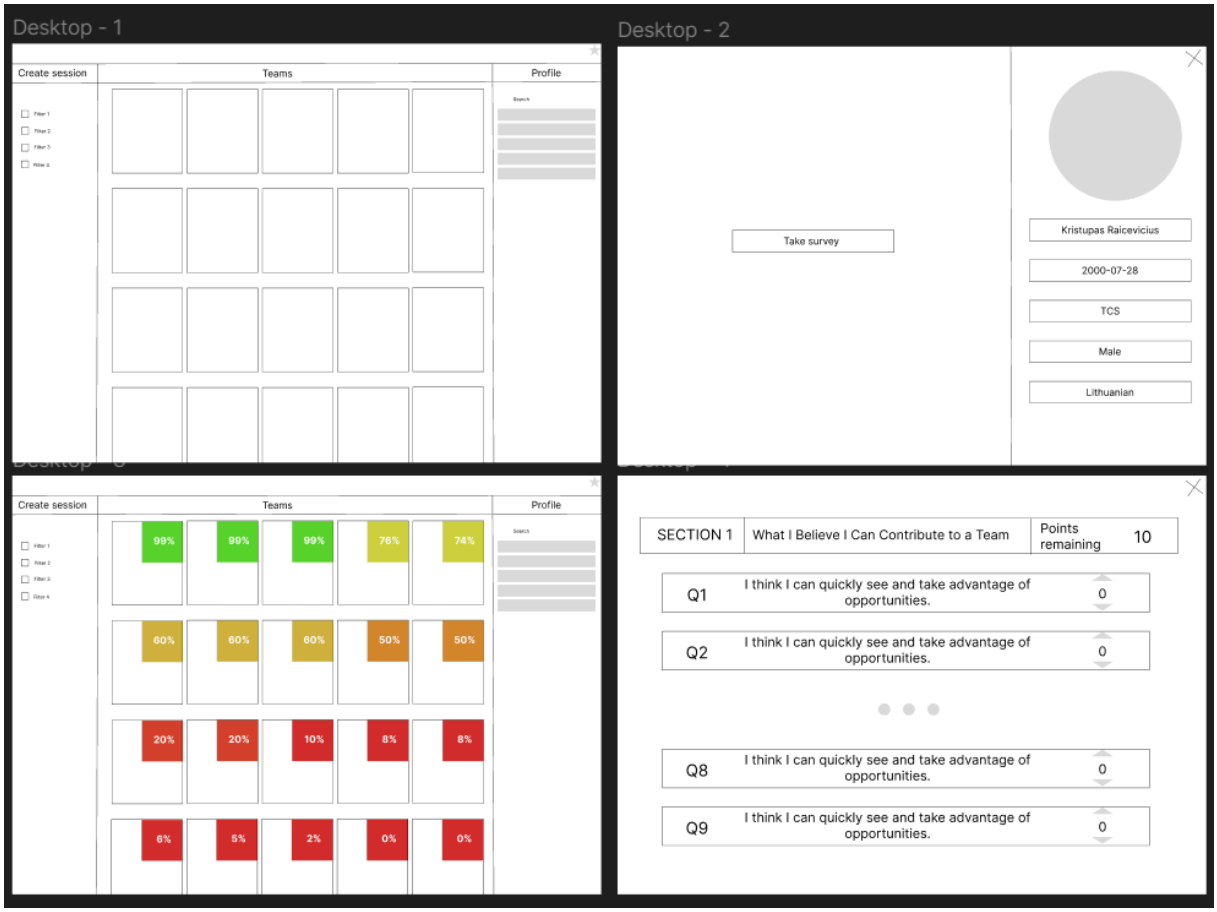


Figure 17: Initial mockups made in Figma

References

[1] V. Yannibelli and A. Amandi, "Forming well-balanced collaborative learning teams according to the roles of their members: An evolutionary approach," in *2011 IEEE 12th International Symposium on Computational Intelligence and Informatics (CINTI)*, pp. 265–270, 2011.

[2] I. Poort, E. Jansen, and A. Hofman, "Does the group matter? effects of trust, cultural diversity, and group formation on engagement in group work in higher education," *Higher Education Research & Development*, vol. 41, no. 2, pp. 511–526, 2022.

[3] D.-Y. Wang, S. S. Lin, and C.-T. Sun, "Diana: A computer-supported heterogeneous grouping system for teachers to conduct successful small learning groups," *Computers in Human Behavior*, vol. 23, no. 4, pp. 1997–2010, 2007.

[4] D. Lambić, B. Lazović, A. Djenić, and M. Marić, "A novel metaheuristic approach for collaborative learning group formation," *Journal of Computer Assisted Learning*, vol. 34, no. 6, pp. 907–916, 2018.

[5] V. A. Hoevemeyer, "How effective is your team," *Training & Development*, vol. 47, pp. 67–71, 1993.

[6] L. G. Bolman and T. E. Deal, "What makes a team work?," *Organizational Dynamics*, vol. 21, no. 2, pp. 34–44, 1992.