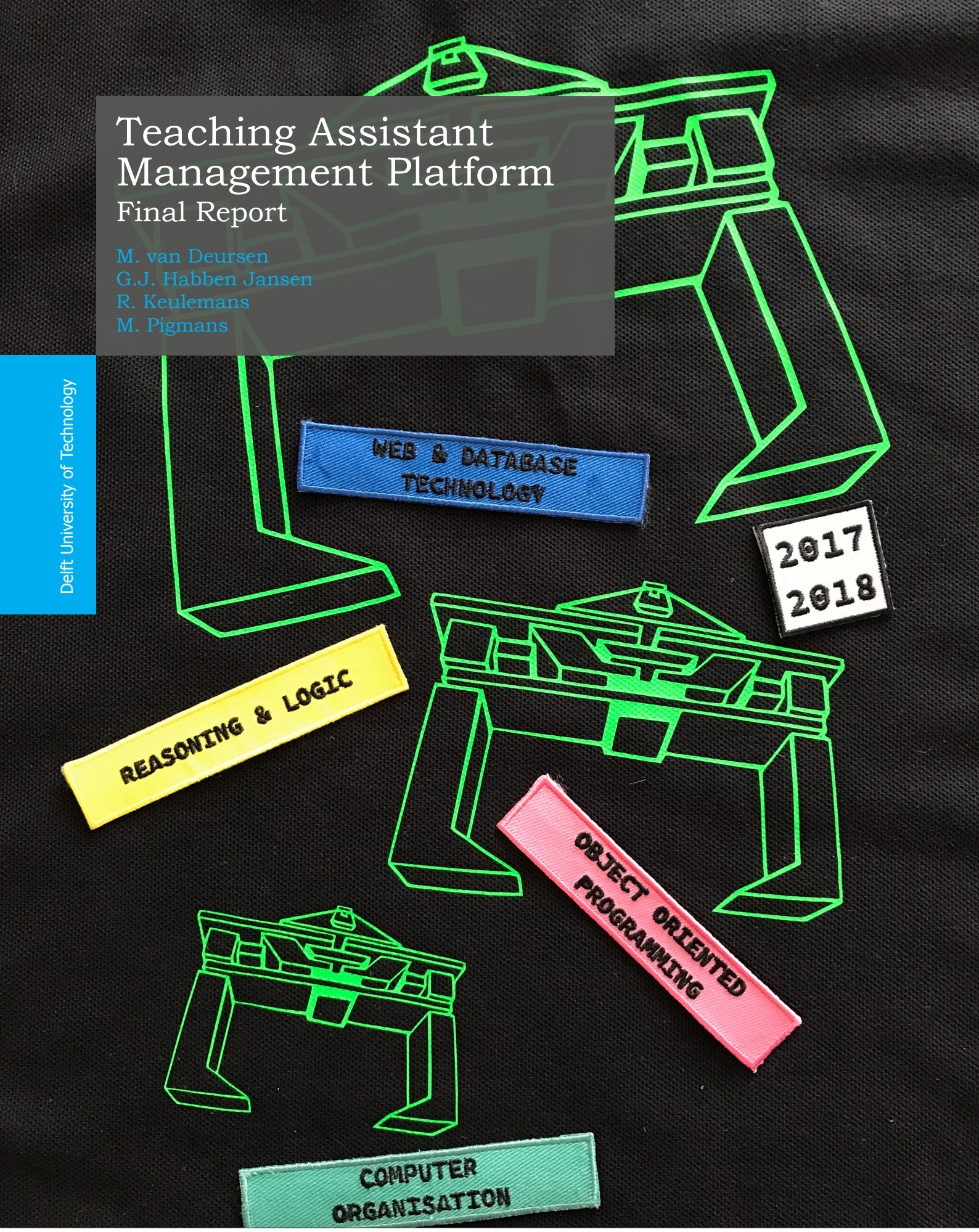
Teaching Assistant Management Platform

Final Report

M. van Deursen
G.J. Habben Jansen
R. Keulemans
M. Pigmans



Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

Teaching Assistant Management Platform

Final Report

by

M. van Deursen
G.J. Habben Jansen
R. Keulemans
M. Pigmans

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at the Delft University of Technology,

Project duration:	April 23, 2018 – July 2, 2018	
Client:	MSc. Stefan Hugtenburg	Education Innovation Projects
Coach:	Dr. Xavier Devroey	Software Engineering Research Group
Bachelor Project Coordinators:	Ir. Otto Visser	Distributed Systems Group
	Dr. Huijuan Wang	Multimedia Computing Group

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft Delft University of Technology

Preface

For our Bachelor End Project, we chose to create the Teaching Assistant Management (TAM) platform with a clear reason: we wanted to build a platform that is put into production and actually going to be used. Instead of 'hobbying' on a project for 10 weeks after which it is probably forgotten, we wanted to build a maintainable and extendable product that would stay useful for years. Considering what has happened over the past 10 weeks, we can say that we achieved our desire of putting our code into production. The first version of our platform was already deployed at the end of week 4. We are happy to see that TAM has already been used by over a hundred students to indicate the courses they want to assist as well as their availability for first two quarters of the coming year. We think that our platform is going to streamline the process of appointing TAs and improve quality of life for students and staff alike.

We found a great deal of joy in the process of developing TAM. Special thanks go to Stefan Hugtenburg, our client. His positive attitude helped to make the project enjoyable.

Special thanks also goes to Xavier Devroey, our coach. By providing useful feedback on both the project and our interaction with the client, he helped us to deliver a great product.

While there are still a lot of extensions possible, we are satisfied with the result we achieved and delivered.

During the project, multiple people supported our development process, and we would like to thank them. We are grateful to Dr. M.M. de Weerdt of the Algorithmics Group for his input on the design of the scheduling algorithm. Gratitude also goes to Dr. A. Katsifodimos of the Web Information Systems group for evaluating our initial database schema. With a special mention to BSc. Gijs Weterings and BSc. Tim Speelman, two students who provided us with useful insights for the development of our system.

M. van Deursen
G.J. Habben Jansen
R. Keulemans
M. Pigmans
Delft, June 2018

Summary

The majority of the courses in the Computer Science Bachelor at the Delft University of Technology use so called *lab sessions* to provide an opportunity for students to ask questions about course material and get feedback on their assignment. In order to optimally support the students, teaching assistants, or TAs, are appointed to assist the lecturer during the lab sessions. With the number of students in the Bachelor quickly growing, the process of manually recruiting students to become a TA and assigning the TAs to lab sessions is becoming infeasible.

This project aims to ease the process of gathering and scheduling TAs. In order to achieve this goal, the Teaching Assistant Management platform, or TAM, has been developed. All parties involved in the process of appointing TAs can use TAM to provide their input. Lecturers can register their courses on TAM, students are able to indicate their interest to help with different courses and representatives from Education and Student Affairs can validate the application of the interested students. Using this input, TAM is able to automatically create a schedule by assigning TAs to lab sessions.

To provide an algorithm for the automatic generation of schedules, a model based on the minimum-cost max flow problem is created. Due to complications with the implementation of the minimum-cost max flow model, the schedule generation is handled by a linear solver: Gurobi. By modeling the constrains for a schedule to be considered valid, Gurobi is used to process the input of the users into an optimized schedule.

TAM consists of three components: a MySQL database, a backend written using Spring, containing the business logic and the implementation of the scheduler, and a frontend website created with Vue to provide an interface to the users. The frontend and the backend are connected using a REST API.

A unique aspect of the project is the live deployment of TAM. At the end of the fourth week, the first version was deployed, allowing interested students to submit their course preferences. Subsequent features have been deployed iteratively.

During the development, muliple problems have been encountered. The team underestimated the time required to learn the new technologies, as well as the time needed to maintain a system in production. Furthermore, configuring Single Sign-On required more time than expected.

Contents

List of Figures

1

Introduction

The Teaching Assistant Management platform (TAM) is a platform designed to improve quality of life for students and staff of the Delft University of Technology by streamlining and automating the existing workflow of recruiting, appointing and informing Teaching Assistants.

Quality of education at the Technical University of Delft relies upon so called *lab sessions*. In the Computer Science Bachelors program, almost 90% of the courses make use of these lab sessions. During the lab sessions, students have the opportunity to ask questions about course material and get feedback on their assignments from experienced students, called teaching assistants or TAs. For each course, TAs need to be informed, recruited and assigned to the lab sessions. The current process of hiring and appointing TAs is opaque to students and requires a significant amount of work from staff for each step, with an estimated 200 hours spent on checking if students are fit to become a TA and over 80 hours being spent on assigning TAs to lab sessions.

TAM aims to encapsulate this entire process by creating a structured and largely automated replacement for the current system. This not only involves the process of gathering interest from potential TAs, lecturers responsible for the lab sessions, and study advisors responsible for validating if each TA has sufficient skills to become a TA, but also making a schedule, ensuring that each lab has enough TAs, and informing users of the created schedule.

The first version of TAM, which was capable of gathering interest in courses from TAs, was deployed in the fourth week of the project. Successive weeks have added additional functionality to the deployed system, including authentication with Single Sign-On, the ability for lecturers to input their courses and lab sessions, pages for the study advisor to give their input on TAs, and the availability to automatically assign TAs to lab sessions.

This report gives a complete overview of the TAM platform created during our Bachelor End Project. Chapter 2 begins by expanding upon the problem TAM solves, gives a requirement overview and details the research done to ensure the created product solves the requirements. Chapter 3 explains the design decisions that were made during the process of creating TAM, ranging from programming languages and frameworks used, to database design, to overall design goals. The workflow and the process of the project are explained in Chapter 4. An overview of the entire product is given in Chapter 5, including a detailed overview of the frontend, backend, scheduler and database. This chapter also covers code quality, test coverage and the feedback received from the Software Improvement Group (SIG). The ethical implication of TAM are discussed in Chapter 6. In Chapter 7, a reflection on the project as well as a suggested list of future improvements to the TAM platform is given, inspired partly by the initial requirements and partly by our findings during development. Finally, in Chapter 8, a conclusion is given on the intended goal of the project.

We strongly recommend anyone interested in understanding the initial goals and research that shaped TAM to read Chapters 2, 3, and 6. Anyone interested in continuing development should also read Chapters 5.2 through 5.5, in combination with Chapters 7.1, 7.4 and 7.6. Appendix E contains a list of all technologies and frameworks mentioned in this report.

2

Research

To solve the problem presented by the client, research into the problem and solution has to be done. The research has been performed during the first and second week of the project. This chapter gives an overview of the findings made during the research phase. First, a definition and an analysis of the problem is given. Then, our solution for the problem is presented together with a set of requirements. Finally, our solution for the scheduling problem is explained.

2.1. Problem Definition

Almost 90% of all Bachelor courses in the Computer Science program given at the Delft University of Technology make use of so-called *lab sessions*. These sessions provide an opportunity for students to work on assignments and receive support when necessary. To support the lecturer(s) in facilitating these lab sessions, other students are employed as Teaching Assistants (TAs). These TAs have completed a previous iteration of the course and have shown interest in assisting the lecturer. Selection of TAs is done based on multiple criteria, such as their proficiency in the course, measured by their grade, and past experience as a TA. Once assigned to a course, the TAs can help the students by answering questions during lab sessions or other tasks like grading assignments, answering student mail or proofreading and grading exams.

The number of students for Computer Science is quickly growing. Where around 280 students started the study in 2015, an estimated 800 new students will start in 2018. Because of the growing student population, the demand for TAs is growing and the process of manually gathering and scheduling TAs has become unsustainable. The current process takes over 80 hours each year to create a schedule for each quarter and an additional estimated 200 hours for representatives from Education and Student Affairs (ESA) to verify if each student is fit to become a TA. This process will take even longer if the number of TAs increases with the increasing number of students attending lab sessions. Besides, certain tools currently used to perform these tasks are not compliant with the recent changes in legislation concerning the protection of personal data, the GDPR [1].

2.2. Problem Analysis

The goal of this project is to simplify and automate the process and reduce the amount of time needed to gather, assign and schedule TAs. To understand how this can be achieved, this section provides an insight into the process as it currently stands.

Since the platform will be used by different types of users, five roles are identified to provide a clearer overview. These roles are:

1. Teaching Assistant: a student who wants to assist during lab sessions.
2. Course Manager: a person responsible for a course. This role could be fulfilled by a lecturer or an assistant of the lecturer.
3. Verifier: a representative of ESA responsible for checking if a student may become a TA.

4. Scheduler: a person responsible for gathering, appointing and scheduling TAs.
5. Admin: a person responsible for the maintenance of the platform.

The process starts with the central scheduler listing courses requiring TAs and distributing a form among students to ask for which courses they would want to become a TA. The form is filled out by students and the information is processed, duplicate entries and invalid entries are removed. An initial shortlist with accepted students is created and an availability form is sent to them. The shortlist is also sent to the verifier to check if each student has passed the required TA Training and English Test. After the availability form is filled out by students and is processed, a list of labs per course is created with the required amount of TAs per lab. The students are then manually scheduled. Once the schedule is finished, it it sent to students and course managers. Scheduled students are registered to FlexDelft[1], the organization managing their payment. Finally the course manager contacts the TAs of the course with specific instructions. This process is visualized in Figure 2.1.

The preference gathering process covered two quarters per iteration. Using the information gathered during this process, the schedule for the two quarters can be created.

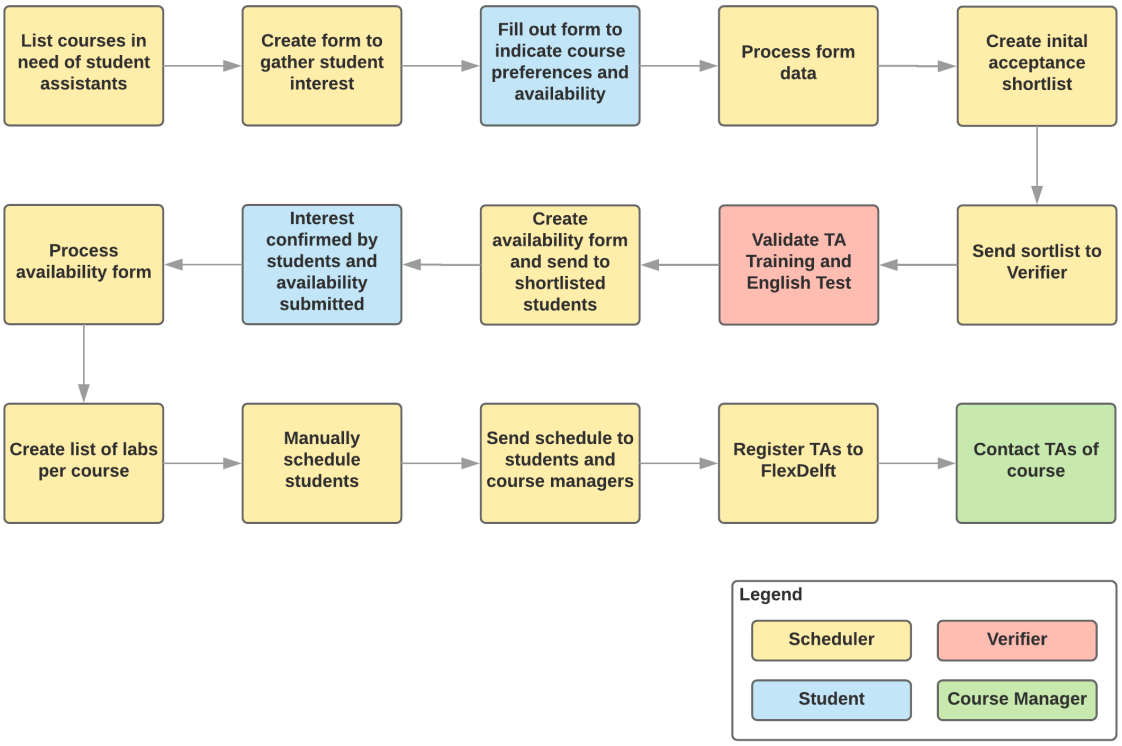


Figure 2.1: Current recruitment and scheduling procedure

2.3. Solution

In order to reduce the amount of time and work needed by staff to gather and schedule TAs, TAM is developed during this project. TAM aims to encapsulate the entire process for all parties involved. The process begins by course managers (CM) registering courses in the system. Students can then indicate their interests for working as a TA on the registered courses. Once the students have indicated their preferences, a representative from ESA is able to verify if each student has passed his or her TA training and English test and if each student is capable enough to become a TA based on his or her personal progress of their study. Finally, the scheduler can use TAM to recruit students as TAs and automatically generate a schedule by assigning the newly recruited TAs to lab sessions.

[1]https://flexdelft.nl/

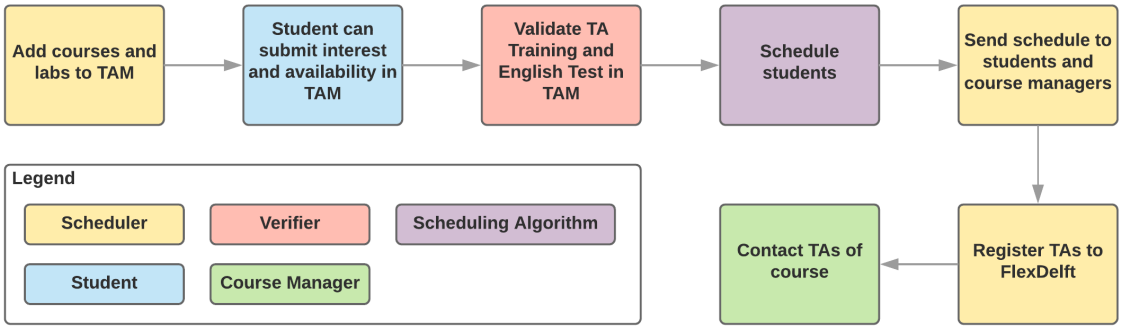


Figure 2.2: Recruitment and scheduling procedure with TAM

Generating this schedule by assigning students to labs is an important feature of TAM. These students first have to be selected for TA roles and then assigned to courses, such that there are enough TAs present for each lab session of each course. Besides indicating their preferences, students are also able to register their availability during each quarter. The course managers are also able to indicate their preferences for students interested in their course. They can veto students they do not want as a TA, they can 'manually' accept TAs without involvement of the scheduler and they can indicate preference for other interested TAs (scale from 0 to 4), all based on previous experiences with the TA. TAs who are preferred by the course manager should be prioritized over TAs that are not preferred. Another goal for the automated scheduler is to manage the assignment such that both the number of TAs per course and the number of courses per TA are minimized. This is to minimize the overhead involved for each TA, including factors such as hiring and training.

An overview of the proposed process flow TAM aims to achieve can be found in Figure 2.2. The proposed process flow contains less overhead than the original process, mainly for the scheduler. The automated scheduling process also replaces a part of the work done by the scheduler.

2.4. Requirements

In order to design TAM, a series of requirements is created. These requirements are split into three categories: functional requirements for the platform itself, requirements for automatic schedule generation and non-functional requirements for the codebase.

2.4.1. Functional Requirements

The functional requirements of the platform describe the features to be implemented in the platform. A list of these requirements has been compiled together with the client. The full list of features can be found in Appendix A.

Since TAM will be used by users in different roles, the requirements are split by role using the roles described in Section 2.2.

The features are also grouped using the MoSCoW method to determine the importance of each feature for the working of the platform. The MoSCoW method groups features into four categories: *Must have*, *Should have*, *Could have* and *Would have*. These four categories indicate a decreasing level of importance to the functioning of the platform. Features in the *Must have* category describe the core functionality of the platform and the most desired features once the core features are implemented. Implementation of these features has the highest priority. Features in the *Would have* category are the least essential features and should not get priority during development. The *Should have* and *Could have* categories are used to prioritize the features whose priority falls between the two other categories.

2.4.2. Schedule Generation

The generated schedule must also follow a set of requirements. These requirements, expressed as constraints, are split into hard constraints and soft constraints. Hard constrains must hold for a schedule to be considered valid, while soft constraints describe optimization criteria.
The following hard constraints have been identified:

1. A student cannot be scheduled for multiple labs during the same timeslot.
2. A student can only be assigned as a TA to a course if they want to TA for that course.
3. A student can only be assigned as a TA to a course if the CM allows the student to be a TA.
4. A student must be assigned as a TA to a course if the CM has indicated so.
5. A student can only be assigned as a TA at a timeslot if they are available during that timeslot.
6. A student can only be assigned as a TA to a lab if they are assigned to a course of that lab.

The following soft constraints have been identified:

1. It is better for a course to have fewer TAs.
2. It is better for a student to be assigned to fewer courses.
3. It is better for a course to have TAs which the CM prefers having.
4. It is better to have the same amount of TAs assigned to a lab to be the same as the amount of TAs required for the lab.

Most of these constraints are straight forward, but two require additional clarification.

The last hard constraint, *A student can only be assigned as a TA to a lab if they are assigned to a course of that lab*, means that a student cannot be assigned to a lab session for a course where he or she is not assigned to. This is because the model has two separate decision variables, one for assigning students to courses and one for assigning students to labs. This constraint ensures that these two decision variables are linked, and also supports shared labs.

The last soft constraint can be formulated as a hard constraint in the following way: *A lab must have the same amount of TAs assigned as are required for the lab*. The reason this constraint is modeled as a soft constraint is that it allows for an incomplete schedule to be computed if no full assignment is possible (due to a lack of TAs in order to meet the requirements of each lab session). By providing an incomplete schedule, the scheduler can identify the gaps and find extra TAs to fill the gaps in the schedule. When modeling this constraint as a hard constraint, no schedule will be generated since no schedule could fulfill all hard constraints and no explanation on the courses or lab sessions that could not be filled is given.

2.4.3. Non-Functional Requirements

Besides the requirements in the functioning of TAM, non-functional requirements are also determined. These requirements are split into three categories: privacy, security and maintainability.

Privacy includes the protection of the personal data of the users of TAM. The platform must follow the requirements stated in the new GDPR. This means that a user must be able to receive an overview of all of the data stored which they are subject to. Also, a user must be able to opt-out of the system and have all his or her personal data removed.

Security means that TAM should be protected against common and known attacks. With TAM running on the university's own servers, protection against attacks aimed directly at the server is not an issue. Protection against attacks through TAM, like cross-site scripting, cross-site request forgery or unauthorized-access attacks should however be provided by TAM itself.

Maintainability includes the possibility to update and extend the codebase. After the project is finished, development of TAM will likely be continued by a new developer. To minimize the amount of time this developer will need to start working on TAM, the structure of the code should be clear and the code itself should be well documented. This documentation should contain explanation on the working of each component as well as a detailed description on how the different components interact with each other. This way, development should be easy to pick up without requiring the help of the original development team.

2.5. Scheduling Algorithm

The requirements for generating a schedule have been expressed in Section 2.4.2, but to actually create a schedule, an algorithm has to be created. The following section describes the process of going from the identified requirements to a design for a suitable algorithm.

Initial research indicates that this problem is a variant of the Teacher Assignment problem [2]. The teacher assignment is an instance of a scheduling problem, which is NP-complete[3]. Although scheduling problems are an ongoing area of research, there has not been much research on the Teacher Assignment problem specifically [4]. The teacher assignment problem that TAM tries to solve has several key differences with the various teacher assignment problems studied in literature. First, in our instance, each TA has their individual availability which must be respected whereas most literature assumes that teachers will fit their schedules to the courses they are assigned to. Second, in our case, multiple TAs are required for each lab session rather than one teacher per course, as assumed by most literature. Finally, our problem size of 200 students, 20 courses and 200 labs is substantially larger than the examples studied in literature as well.

A common method of solving Teacher Assignment problems is by using linear programming. Linear programming involves minimizing an objective function, while keeping the variables bound to certain constraints. An important consideration of linear programming are the runtimes. Domenech and Lusa[4] considered instances with 20 teachers and 80 courses and concluded computation times for optimal solutions of nearly one hour, with runtime quickly growing along with the input size. Because our instances have a different form, having more TAs (teachers) than courses rather than vice versa, it is difficult to extrapolate from this what we our runtime would be when using such a solution. This was discussed with the client and it was concluded that long runtimes are acceptable, if it could not be avoided.

Another approach taken for a similar problem, which combines Teacher Assignment with Course Scheduling, is proposed by Gunawan et al. [5]. They use a hybrid approach, first calculating an approximate solution, and then using greedy heuristics and simulated annealing to quickly find near optimal solutions. They report runtimes of 2 minutes with instance sizes of 30 teachers and 60 courses, with results of about 85% of the known optimal value. This is substantially faster than the results in other papers. Since their problem also considers Course Scheduling, they do consider it to be more complex than just teacher assignment. It is therefore likely that applying a similar method to our problem would not result in significantly worse results, making it a possible solution. However, this approach is markedly more complex and produces worse results, albeit faster. These downsides outweigh the advantages for TAM, so at least for the initial version this is not considered to be a better fitting solution than just a linear solver.

While the initial research resulted in several possible solutions, more research was needed before committing to one of these options. As part of our further research we contacted Dr. Mathijs de Weerdt (Algorithmics Group, Delft University of Technology) and discussed possible other solutions. This discussion resulted in several other potentially fitting mappings to well known problems. In the following sections, each of these mappings is discussed and finally our decision is explained.

2.5.1. Matching

Matching appeared to be a good problem to map to. Matching problems refer to where a subset of edges have to be selected, matching certain criteria. Matching with preferences[6] seemed to be able to optimize for all our soft constraints; Course Manager preferences can be accounted for easily and the other minimization constraints could be solved via smart ordering of matching. However, there was no suitable way of expressing the hard constraint of *only schedule a TA for one lab during a certain*

timeslot. This is a constraint that could not be neglected, making matching an unsuitable mapping.

2.5.2. Max Flow

Max flow[7] problems involve finding the highest amount of flow that can be sent through a graph, where each edge has a capacity on how much flow can be sent. Max flow easily models the hard constraints; capacities on edges between TAs and timeslots force TAs to only assist one lab at a time, while the other constraints can be expressed through the construction of the graph. Similarly, the required number of TAs in a lab session can be modelled via edge capacities. Initially, there appeared to be no good way to model the various soft constraints. It seemed possible to modify the order in which paths are considered, based on the soft constraints. This solution might work but it would be difficult to guarantee that the solution obtained is of high quality. In order to solve this problem minimum-cost max flow[7] seemed more suitable.

2.5.3. Minimum-cost Max Flow

Minimum-cost max flow is a variation of max flow that adds costs to edges, and aims to both maximize flow and minimize cost. It appeared to be able to solve all the constraints. However, we realized that shared labs were not considered during the listing of our constraints. Shared labs are labs where students can ask questions about multiple courses at the same time, meaning that the solution needs TAs for multiple courses to be present. In fact, the goal for shared labs is to have as many TAs as possible for each of the courses - making TAs that can assist with multiple courses better than TAs that are only available for one of the courses.

Ideally the scheduler should support the ability to specify how many TAs are needed in total for a shared lab, and specify for each of the courses how many TAs should be present as well. Unfortunately, minimal-cost max flow is not capable of completely expressing this. It is possible to guarantee (a limit) of the total amount of TAs in the shared lab. However, for the amount of TAs per course in the shared lab, the best possible guarantee is a total number (between all courses) equal to the total number of TAs of that shared lab. To give an example, for a shared lab with 30 total TAs and 3 courses, it is possible to promise that at least 10 TAs are present for each of the courses, but not promise that 11 TAs are present for each of the courses. However, it is possible to adjust the order in which TAs are selected in order to first select TAs that are available for multiple courses - giving us a result that is likely near optimal for the total amount of TAs that can do each course. We discussed this solution with the client and they agreed that this would be good enough to solve the problem of shared labs, making minimal-cost max flow a suitable mapping.

Minimum-cost Max Flow mapping

To solve our problem through the min-cost max-flow algorithm, a suitable modelling has to be made. We create construct the model as follows:

1. Create a source and sink node.
2. For each student, a node is created.
3. Each student-node is connected to the source node, with capacity of the edge equal to the amount of hours the student wants to work.
4. Per student, a node is created for each timeslot he is available in.
5. Each timeslot-node is connected to the student with a capacity of 1 and no cost.
6. For each lab, a node is created.
7. A lab-node has an incoming edge from each of the timeslot-nodes that represent the timeslot of the lab, but only if the student is allowed and willing to TA for the course that the lab is from. The capacity of this node is 1 and the cost depends on multiple criteria, such as the preference of the Course Manager for the student.
8. Each lab-node is connected to the sink, with a capacity corresponding to the amount of required TAs and without cost.

An example of this model is shown in Figure 2.3.

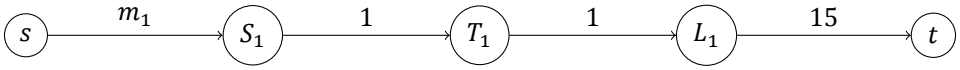


Figure 2.3: An example of a lab session L_1. In this case, student S_1 is available during timeslot T_1 and is interested and allowed to TA for course corresponding to L_1.

Shared lab sessions

When a lab is a shared lab, however, the creation of nodes and edges for that lab is different. Lets take an example with a shared lab of two courses: A, and B. The first five steps are the same as the process for modeling a regular, non-shared lab. After step 5, the procedure is changed as follows:

6. For each subset of the courses of the shared lab, omitting the empty set, a node is created (i.e. the nodes $\{A\}$, $\{B\}$ and $\{A, B\}$).
7. A subset-node has an incoming edge from each of the timeslot-nodes that represent the timeslot of the lab. This edge is only added for the subset-node which best fits the student (e.g. if student S is interested in becoming a TA for both A and B and is allowed to do so, the outgoing edge from the timeslot goes to the node $\{A, B\}$). The cost is again determined by preference and reflects the preference for TAs who prefer more courses as well.
8. For each course of the shared lab session, a node is created and connected to all subset-nodes containing the course. These edges have an infinite capacity and zero cost.
9. Each course-node is linked to the sink node with a capacity equal to the number of TAs needed for that course during the lab.

An example of shared labs is visualized in Figure 2.4. It involves a lab for courses A and B, where 15 students are requested for course A and 5 for course B.

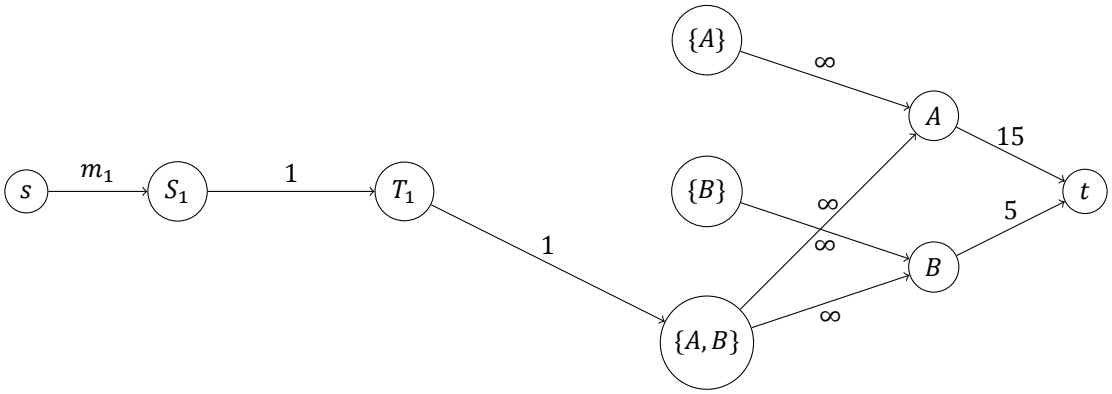


Figure 2.4: An example of a shared lab with courses A and B. In this case, student S_1 is available during timeslot T_1 and is interested and allowed to TA for both course A and course B.

In Section 2.4.2, several hard and soft constraints are described. In order for our model to work, each of the hard constraints have to be enforced. The constraints for adding edges to the model cause the second, third and fifth constraint to be enforced. The capacity of the edges from a student to a timeslot make it so that a student cannot be assigned to multiple labs in the same timeslot. Moreover, the fourth constraint, which requires that a student must TA for a certain course, can be enforced by pre-processing the input of the model by removing the hours used by this assignment from the students' maximum hours and removing all edges dealing with this course. Lastly, the sixth constraint follows immediately from the assignment, because a student is assigned to a course if he is assigned to a lab from that course. Because of this, all hard constraints in the model are satisfied.

To optimize on the soft constraints, the costs of certain edges will be adjusted throughout the solution computation to incorporate the constraints into the algorithm. Moreover, the picking order will be adjusted to also serve the optimization of the soft constraints.

2.5.4. Implementation

The Minimal-cost max flow solution, using the model described above, was to be implemented at the start of the project. Even after extended research we could not find any research on such an algorithm that supported changing the costs of edges edge during execution. After creating a basic implementation of the algorithm, we came to the conclusion that too many requirements were being integrated into the changing-edge-costs. We were unsure that our solution was going to scale to fulfill all requirements, and if it did, whether it would be easily understandable and extendable. Because of these reasons, we chose to go back to using a linear solver. We felt that this choice makes it clearer how the various requirements interact, makes it easier to judge the correctness of the modeling, and makes it easier to modify or extend without unexpected changes in results. In addition, using an existing linear solver rather than our own minimal-cost max flow implementation means that the scheduler will benefit from any advancements made in linear solver technology, without requiring work from TAM. The team does not have any experience with linear models or using linear solvers, since this is not covered by the Computer Science Bachelor. However, we felt that the advantages over a Minimal-cost max flow are substantial enough to warrant this learning process.

2.5.5. Linear Solver

The first step was to ensure that a linear solver could model all constraints, by creating a suitable modelling of our problem. Such a modelling was made in the style of Gunawan et al.[5] and can be found in Appendix B. In this model all requirements from Section 2.4.2 are listed and for each requirement a formulation of how we could model it as a linear constraint or objective is given. Based on this initial modelling an implementation could be made, but there was one more decision to be made. A linear solver implementation had to be selected, that TAM would make use of. Initially, the primary options considered were open-source solvers such as *LP_solve* and *Coin-Or's Clp*. However, we found that these did not have documentation that was easy to use with our limited experience. It was not clear how to begin implementing the modelling using these tools. We decided to check the options available with commercial solvers and found Gurobi to relatively clear documentation, especially useful was the large set of example programs available. On top of this, we found that Gurobi does very well in benchmarks [8]. Since Gurobi is not free to use, we asked our client whether we could use Gurobi. He informed us that the free academic licence was suitable for our system, making Gurobi an applicable tool.

3 Design

The set of functional and non-functional requirements as discussed in Chapter 2 have to be considered during the creation of TAM. TAM consists of three major components: a database to store the data, a backend application containing business logic and the automated scheduler, and a frontend website the user can interact with. These three components together form the stack. The technologies used in the various components of the stack need to synergize.

The design decisions behind the stack are discussed in this chapter. First, the design of the database, backend and frontend is explained. This is done by listing the important requirements, identifying the different languages, frameworks and technologies to be considered and finally picking the best fit for the project. A complete list of technologies and tools, including links to their webpages, can be found in Appendix E. Then, an overview of the complete stack is given. The chapter is concluded with a description of the additional tooling used to support the development process.

3.1. Database

Since TAM is mainly based around gathering, aggregating and updating data, the database is an important part which has to be chosen carefully. There are two phases that the research process for the database has to go through. Firstly, the database software has to be chosen. Moreover, once a database is chosen, the schema has to be designed carefully to cater to the envisioned features of TAM.

3.1.1. Database System

The main criteria which influence the decision of the database system are the scale of the information which has to be saved, as well as the relation between saved information.

When looking at the Bachelor courses, a round of interest gathering covering two quarters results in under 200 interested students. Also taking into account that student who have indicated their preferences during one round of interest gathering are also likely to indicate their preferences during subsequent rounds, the number of registered users will be limited. This small user base means that the impact of the expected size of the database on the decision for the database system is negligible. However, the data which is stored in the database is closely related and coupled. For example, the relation between the courses and students assigned to them is a relation that needs to be mapped well in the database. Therefore, the database has to be able to map these relations well and this is a requirement which has to be taken into consideration.

There are two categories in which databases are divided: relational and non-relational databases. A relational database stores the data and the relations between this data in a clear way. A non-relational database instead focuses on support for large-scale (distributed) deployment and sacrifices the functionality to efficiently save relations between data.

As mentioned before, the primary requirement for selecting a database is that relations between data in TAM can be mapped correctly and easily. The main advantages that non-relational database

provide are speed and scalability. Based on these factors, a relational database is the best fit for TAM.

There are multiple different prevalent databases of the relational database category. The three different databases that are considered are: PostgreSQL, MySQL and SQLite. These three databases are the most used relational databases according to the StackOverflow survey of 2018 [9]. They are similar in functionality and have extensive documentation and references. MySQL is already used by the client for an existing in-house application, Queue. This means that it already is installed on the server, making the initial deployment of TAM more straightforward and no extra maintenance will be added to the server. Since there is no functionality missing from MySQL which is available in one of the alternatives, MySQL is the most suitable database system for TAM.

3.1.2. Database Design

To determine the structure of the database, a conceptual model is created based on the requirements. An appointment with Dr. Asterios Katsifodimos (Web Information Systems Group, Delft University of Technology) was made to receive feedback on the initial model. Dr. Katsifodimos said that there were no structural problems with the conceptual model and he provided a lot of useful insight and advice for the database component. He emphasized that the created model is inevitably going to change during the project. Using this conceptual model, a logical schema for MySQL is created in Vertabelo. This logical schema can be seen in Figure 3.1.

Core Data

What becomes immediately apparent from the database schema is the central role of both the `User` and `CourseEdition` tables. The former embodies a User in the database and the latter embodies a course in a given year and quarter. These two entity types form the basis around which TAM revolves, namely assigning students to course editions. There are multiple design decisions made during the creation of the database schema.

To begin, a user has both a `gender` and `t-shirt size` field. These fields are used to hand out the correct TA t-shirt of the specified gender to the student.

Another notable aspect is that a user can have multiple roles, described in the `User_Role` table. These roles are the same as the roles described in Section 2.4.1. This design decision is made because a person can attain multiple roles in the current process as well and it was decided to keep this possibility intact.

Preference Tables

The user gets connected through numerous linking tables, although two of these tables contain fields that require some more explanation, namely the `v_rank` field for the `PersonalPreference` table and the `p_rank` field for the `CoursePreference` table.

The `v_rank` field stands for verifier rank and is the rank that is assigned to a student for a certain course by the verifier. This rank is either $0, 1$ or 2 and embodies the judgment of the verifier on whether the student is suited to TA for the specific course.

The `p_rank` field on the other hand is preference rank of the course managers. This rank ranges from 0 to 4 and specifies the preference of the course manager to have the specific student as a TA, with 0 meaning the student is not wanted and 4 meaning the student is really preferred.

Scheduling

Both the assignments of students to courses and students to labs are linked to a `schedule_id` from the `Schedule` table. The primary key of the entity is an id and not a combination of year and quarter. This is done to allow the creation of multiple schedules for one quarter. In turn, the `final` boolean in the same table marks whether the created schedule should be seen as final, meaning that it is the schedule that will be used for that quarter and could be shown to other (non-scheduler) users.

Another notable field from the `Schedule` table is the `interest` field. This field specifies whether the students are able to submit preferences for the courses in the same quarter and year as the schedule.

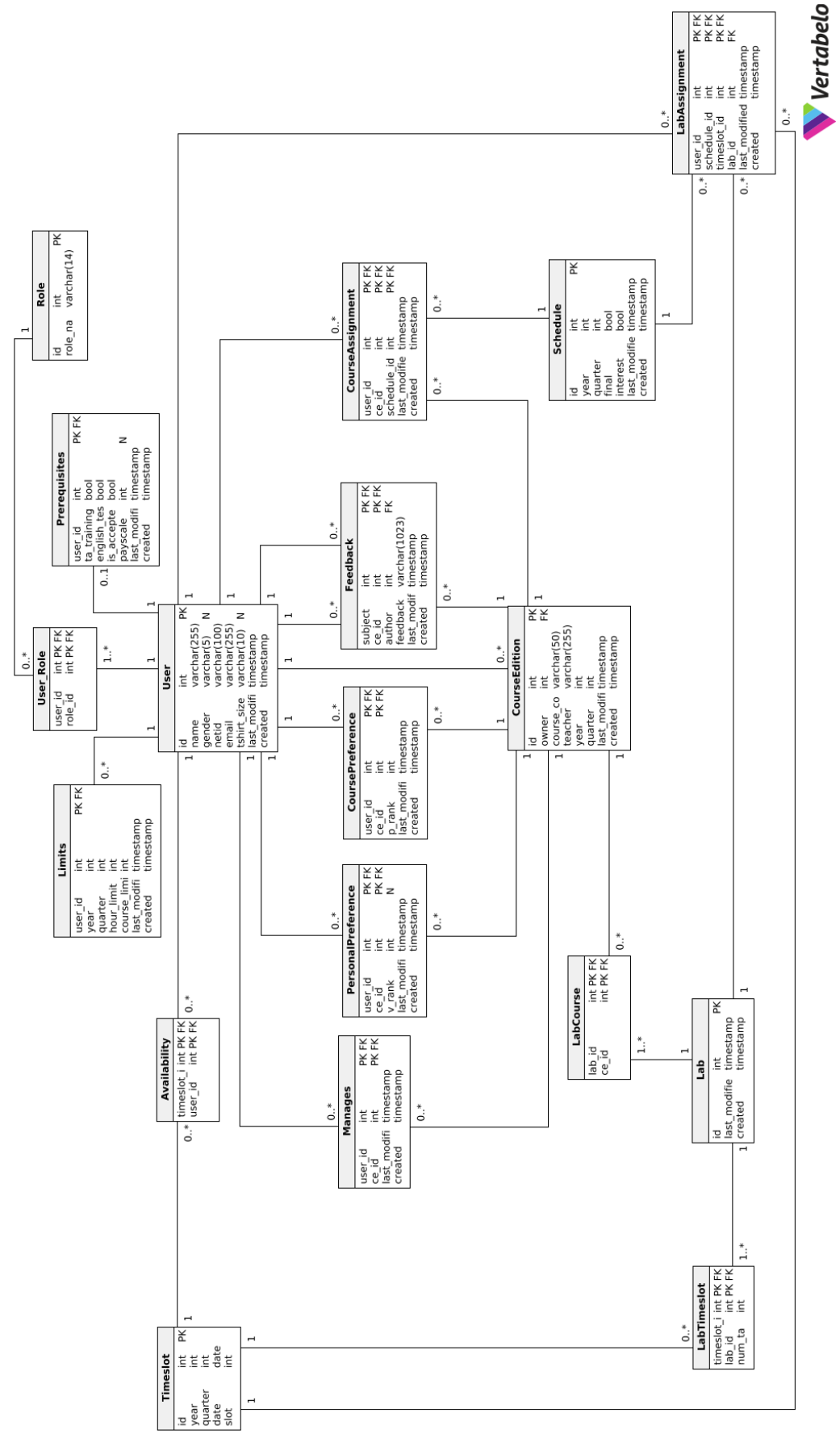


Figure 3.1: Initial design of the database schema

Slot ID	Starting time	Ending time
1	8:45	10:30
2	10:45	12:30
3	13:45	15:30
4	15:45	17:30

Table 3.1: The slots used in the *Timeslot* table.

Additional Fields

Almost all tables have both the `last modified` and the `created` timestamp. These two fields can be used to track and solve possible errors in TAM which cause the database to come in an unwanted state, usually by user input. Through these two fields, the changed time can be extracted and through these timestamps, the changes could be more easily restored via partial recovery from backups. Notice that the `Timeslot`, `Slot` and `Role` do not have these two fields. This is because these two tables are not to be changed by user input and therefore do not require this extensive tracking.

The `Availability` table also misses the `last modified` and the `created` timestamps. Because the entries in the table are binary, a record is created or removed if a user changes their availability, a `last modified` timestamp would not add any information. A `created` timestamp should be present in the `availability` table. This issue has only been noticed while writing this report and will be resolved as soon as possible.

Another important decision was made for the `Timeslot` table, by adding the `slot` field. This field resembles the four timeslots during which lab sessions can take place. These timeslots can be seen in Table 3.1. An entity of the `Timeslot` table therefore represents a two hour slot at a certain date, in a certain quarter and year.

3.2. Backend

In order to connect the user interface to the database, a backend is needed. This backend should both include a connection to the database as well as proving a web server to host the frontend of TAM.

3.2.1. Language

Since the duration of the project is only ten weeks, the primary factor for deciding on a backend web framework is the experience of the developers. However, it has to be noted that there is already a working prototype of TAM, written in Python using Django, which could be extended during the project.

The team is well familiar with Java and somewhat familiar with Python, the former being used throughout the Computer Science bachelor. Since in the future this project is most likely to be maintained and extended by students, choosing a familiar language is preferred. Moreover, the development team prefers a statically typed over a dynamically typed language for the backend. Using the above preferences and requirements, only Java and Python are considered for backend languages.

3.2.2. Framework

When using Python, the most used option is the Django web framework. Django uses the Model View Controller (MVC) pattern, separating views from logic. Furthermore, Django allows easy SQL data manipulation using Django Object Relational Mappers (ORM), instead of writing complex SQL queries. Another advantage of Django is that there is already a working prototype written in Django as mentioned before.

The most popular choice when using Java is the Spring framework. Within the framework, two alternatives exist: Spring WebFlux and Spring MVC. Spring WebFlux is reactive, non-blocking and supports huge amounts of concurrent connections. Spring MVC uses a synchronous blocking I/O model supporting one request per thread. Spring also features ORM capabilities. Although the prototype of TAM is in Python, another in-house project, the aforementioned Queue, uses Spring MVC as well.

3.2.3. Conclusion

Because a working prototype is already provided using Django, a decision could be to expand upon this prototype. However, one of the design goals of the team is to have a maintainable codebase. Upon reviewing of the existing codebase of the prototype, the team decided that the refactoring of the

prototype would take too much time and a complete rewrite would therefore be necessary. With the option to extend the prototype discarded, the decision is whether to write a new application in either Python or Java.

Because of the familiarity of new Computer Science students with Java and the preferences for Java in the development team, Java is chosen as a backend language. Spring MVC will be used as backend framework. The MVC version is preferred over the WebFlux version since the development team is more familiar with the conventional MVC model than the reactive model and support for large amounts of concurrent connections is not needed due to the small expected user base.

3.3. Frontend

The frontend serves as an interaction layer between the user and the backend. To ease multi-platform support as well as maintainability, the frontend of the platform consists of a mobile-friendly interface. Development of this interface requires both a programming language for the frontend to be written in and a system for the frontend to communicate with the backend.

3.3.1. Language

Just like the development of the backend, the limiting factor for the decision on a frontend language is the lack of time to learn new languages within the project. The team is solely familiar with JavaScript.

However, there are multiple options for the frontend language. First, other languages such as TypeScript or Elm can be used. These language provide type description or inference, but would take more time getting familiar with.

Second, the frontend can be integrated with the backend using Thymeleaf, a server-side Java template engine which has modules for Spring MVC. Using this method, the functionality of the frontend is implemented on the backend and a compiled webpage is sent to the user. Given the lack of experience with this option of creating webpages using this method, it is not desired to use for development.

Due to the lack of experience with other languages, JavaScript is to be used for the frontend.

3.3.2. Framework

To improve both the speed of development and reduce the boilerplate code for the frontend, a library or a framework is desired. The most common JavaScript libraries and frameworks for creating interactive web pages are Angular, React and Vue.

The most important factor to be taken into account in the selection of a framework or library is the lack of experience in frontend development within the team and the limited time available to learn new technologies. For this reason, Angular will not be used as multiple sources indicate that it has the steepest learning curve from the three [10, 11].

After reviewing both React and Vue, it was clear that one of the two has to be used. However, the development team could not come to a unanimous opinion on which of the two to use. After reaching consensus by majority voting, Vue was chosen as the framework for the frontend. The main deciding factor was the clear documentation of Vue, which could be beneficial in supporting the frontend development.

To speed up the process of styling the web page, Bootstrap will be used, since it is the most prevalent styling framework [12, 13]. Moreover, Bootstrap provides great support when it comes to different screen sizes. This means that it is relatively simple to create a frontend application which can be used on both desktops and mobile devices. Lastly, by using a package developed by the Vue community, Bootstrap-Vue, the integration between Vue and Bootstrap is simplified, easing the process of developing and styling the web pages even more.

3.3.3. API Specification

Because the frontend uses JavaScript with Vue and Bootstrap, there is no way to integrate the frontend directly into the backend. This means that there has to be some kind of communication between these two components, through which data is shared. There are two major web service communication protocols, namely the *SOAP* (Single Object Access Protocol) and *REST* (Representational State Transfer) protocol. REST allows a great number of different data formats, such as JSON, whereas SOAP only allows XML. Since JavaScript is used for the frontend, the choice for sending JSON object is a natural fit, which is why the REST protocol is chosen.

In order to keep a contract between the backend and the frontend in the terms of what data is sent to and from which endpoint of the backend, an API specification will be written. In this specification, all available endpoints and their path are documented. For each endpoint, documentation should explain what type of requests are permitted, which query parameters can be used on the endpoint and what is returned by the endpoint. Moreover, if a JSON payload is required or returned, the format of the JSON is documented as well. This API specification will be written using Swagger IO, since the team has experience with this tool.

3.4. Stack Overview

Using the languages, frameworks and technologies chosen in the previous sections, the full stack is visualized in Figure 3.2.

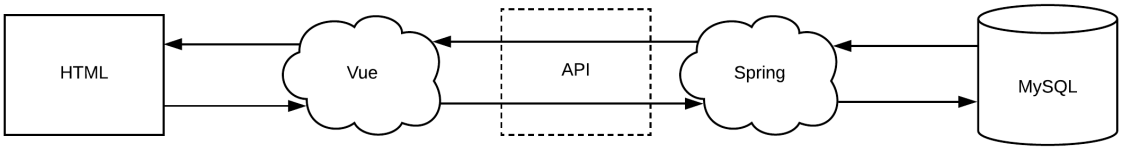


Figure 3.2: Summarized stack overview.

The user interacts with a website created using Vue. This website is styled using Bootstrap-Vue. This frontend interacts with the backend, which created in Spring, using a REST API. All communication is to be specified in an API specification, to be written using Swagger IO. The Spring MVC-based backend serves as an interface between the API and the MySQL database. The backend also holds the business logic of the system, including the automated scheduler.

3.5. Development Tooling

In order to ease the development process, whilst complying to code standards set by the team, the team uses a range of tools. The planned tooling is shown in Figure 3.3a. However, some of the tooling has changed throughout the project. The final tooling is therefore displayed in Figure 3.3b. The bold tools indicate tools that have changed during the project. These changes are discussed in this paragraph.

Function	Backend	Frontend
Editor	IntelliJ IDEA	IntelliJ IDEA
Testing	JUnit	Jest
Code Coverage	Cobertura	Jest
Code Quality	Checkstyle, PMD, FindBugs, SonarQube	Flow, SonarJS, SonarQube
Version Control	Git (GitLab)	Git (GitLab)
Dependency Manager	Gradle	NPM
Build Tool	Gradle	Webpack
Continuous Integration	TravisCI	TravisCI

Figure 3.3a: Intended tooling

Function	Backend	Frontend
Editor	IntelliJ IDEA	**VSCode**
Testing	JUnit, **Mockito**	Jest
Code Coverage	**Jacoco**	Jest
Code Quality	Checkstyle, PMD, **SpotBugs**	**ESLint**
Version Control	Git (GitLab)	Git (GitLab)
Dependency Manager	Gradle	NPM
Build Tool	Gradle	Webpack
Continuous Integration	**GitLab CI**	**GitLab CI**

Figure 3.3b: Actual tooling. Tools that differ between planned and actual are bold.

The team had past experience with VSCode for frontend code. VSCode provides multiple plugins to ease the development in JavaScript and Vue. Because of this, VSCode is used instead of IntelliJ IDEA for frontend development.

For backend testing, Mockito is used to mock objects fields and methods during tests. This is handy because behavior of external components can be controlled using Mockito. Moreover, the code coverage tool was changed from Cobertura to Jacoco because Jacoco works better with the other tooling used.

Both the backend and frontend had changes for their code quality assurance. The idea was to use a SonarQube server to keep track of our code quality over the course of the project. However, the team deemed it too time consuming to set up this server, especially since other tools already provide the same analysis on the current code version. After informing with fellow developers of other in-house products, the team came to a conclusion not to use SonarQube. The other change for the backend is the use of SpotBugs instead of FindBugs. This is simply the result of the deprecation of FindBugs, which is succeeded by SpotBugs.

The frontend was supposed to use SonarQube combined with the SonarJS plugin for static code analysis, but after the decision was made to not use SonarQube, integration with SonarJS was not viable anymore. As a replacement for static code analysis, ESLint is used.

The frontend was also supposed to use the static typing system Flow. However, because of the amount problems encountered during setup with getting Flow and Vue to work together, and the deadline early in the project (see Section 4.5), the decision was made to drop integration with the tool.

Lastly, the idea was to use TravisCI for the continuous integration during the project. However, GitLab provided the project with a built in continuous integration tool as well: GitLab CI. Because of this, the decision was made to abandon TravisCI and use GitLab CI instead.

4

Process

In this chapter, the process of development during the project of the team is highlighted. This chapter includes the internal structure and overall organization of the team, including communication with the client and coach. After this, the set up procedure, feature development and deployment procedures are more closely examined, as these three processes played a major role in the development of the final product.

4.1. Internal Organization

The time span of this project is relatively short. One of the challenges that comes with this short time span is to reduce unnecessary overhead. This can be achieved through having an organized work flow. In this section, three different factors which helped creating this work flow are described. These factors made sure that the work flow is organized whilst keeping an overview of the progress of the project.

4.1.1. Scrum Framework

The first factor that has greatly improved the organization of the team was the use of the Scrum framework. Scrum is a framework that is focused around the incremental and adaptive design of a product, in this case a software product. It uses sprints, a period of time in which a working product with new functionality is created. This was adopted in weekly sprints during this project. At the end of the week, the Sprint Retrospective is used to reflect on the past sprint and to word improvements for the upcoming sprint. Moreover, the plan for the product is revised each week as well, in the Sprint Review, together with the client. These two events were done by the team together with the client to revise the goals whenever necessary and to improve the work flow of the team each week. Moreover, the team has Daily Standups in which they reflect on what has been done the day before and what will be done during the day.

Although some time is spent each day to revise and discuss the planning of the upcoming day or sprints, this helps the team keep focused. Moreover, the daily standups work as a great tool for the team to stay updated on the work of the other team members, as well as on the overall progress throughout the week.

4.1.2. Issue Tracking

Another factor which came into play was that of the user stories. A user story is a simple description from the perspective of a user who desires new functionality. These user stories are then used to determine how much time is required to implement such a feature.

The team adapted this idea of user stories, but altered it to their own needs. The flaw that became apparent was that not all code enhancements could be translated into user stories Instead of rewording these enhancements into a user story format, possibly creating a 'developer' user, the team chose to instead describe the required enhancements and use that as their user stories.

These adapted user stories can be used to track the enhancements or issues that each developer is working on. There are multiple different platforms on which issues can be tracked. The intended platform to be used was the GitLab issue tracker. Since the team already decided to use GitLab as a

remote server to handle their version control on, the choice to use GitLab as well to track their issues was obvious. However, at the start of the development of TAM, a workspace was provided by the client. The team had then decided to use one of the walls as an issue tracking board instead of GitLab, because the physical issue board gave the team a better overview of the project. An example of the issue board can be seen in Figure 4.1.

The issue board had an organization of post-it notes. The top left is the legend and indicates that each color of the post-it note indicates a particular type of issue. Moreover, each small colored stripe of paper indicates who is working on an issue. Below the legend, the board is separated into three different sections. The top section indicates that an issue is still to be done. The middle section contains the issues that are currently being worked on. The bottom section holds the issues that are finished and have been added to the system.

4.1.3. Team Member Roles

The intended division of roles within the team was to have all team members contribute to all components of the end product. Although a developer would always have a specialization of some kind, the intention was to have each team member at least develop one feature for both the front- and backend. In this way, each team member ideally has a sufficient level of knowledge of each component of the end product. However, it quickly became apparent that the time to learn the used frameworks and technologies, especially on the frontend, was significant for each team member. Because the process of becoming proficient in the frameworks used across the whole application would be too time consuming for a project of this duration, the decision was made to instead split the team into two pairs. One pair puts the focus on the frontend, and the other pair focuses on the backend. Because of this split, the team required less time to become proficient in the used frameworks, which in turn meant that the team has more time to develop features. Moreover, there are still two people to discuss on matters for each component, which means that an issue does not have to be tackled by solely one person, but can instead be solved by a pair. However, each team member still has some cursory knowledge of both sides.

4.2. Communication

Communication makes or breaks group projects. Both communication within the team as communication with other involved parties plays a vital role in the outcome of the project. This section briefly discusses internal communication, and how communication with the client and coach was managed.

4.2.1. Team

The communication within the team plays a great role in a successful project. Different members of the team can exchange ideas for new features. Moreover, if a team member struggles with a particular problem, albeit with a conceptual or with a coding problem, another team member can help in solving a problem. Above all, communication within the team improves the cohesion and team effort throughout the project.

During the project, the aforementioned work space is available to the team. Because all team members are present throughout the day at almost all working days, communication can be done instantly and no overhead is introduced by using for example instant messaging. Moreover, the work space includes whiteboards, which are used by team members to illustrate their ideas in a more visual manner. Therefore, the team can easily brainstorm about complicated topics at any time when working on the project. For example, the whiteboard was used to draw wireframes, the initial design of a new webpage; discuss API specification or plan out documents. This leads to quality communication in the team throughout the course of the project.

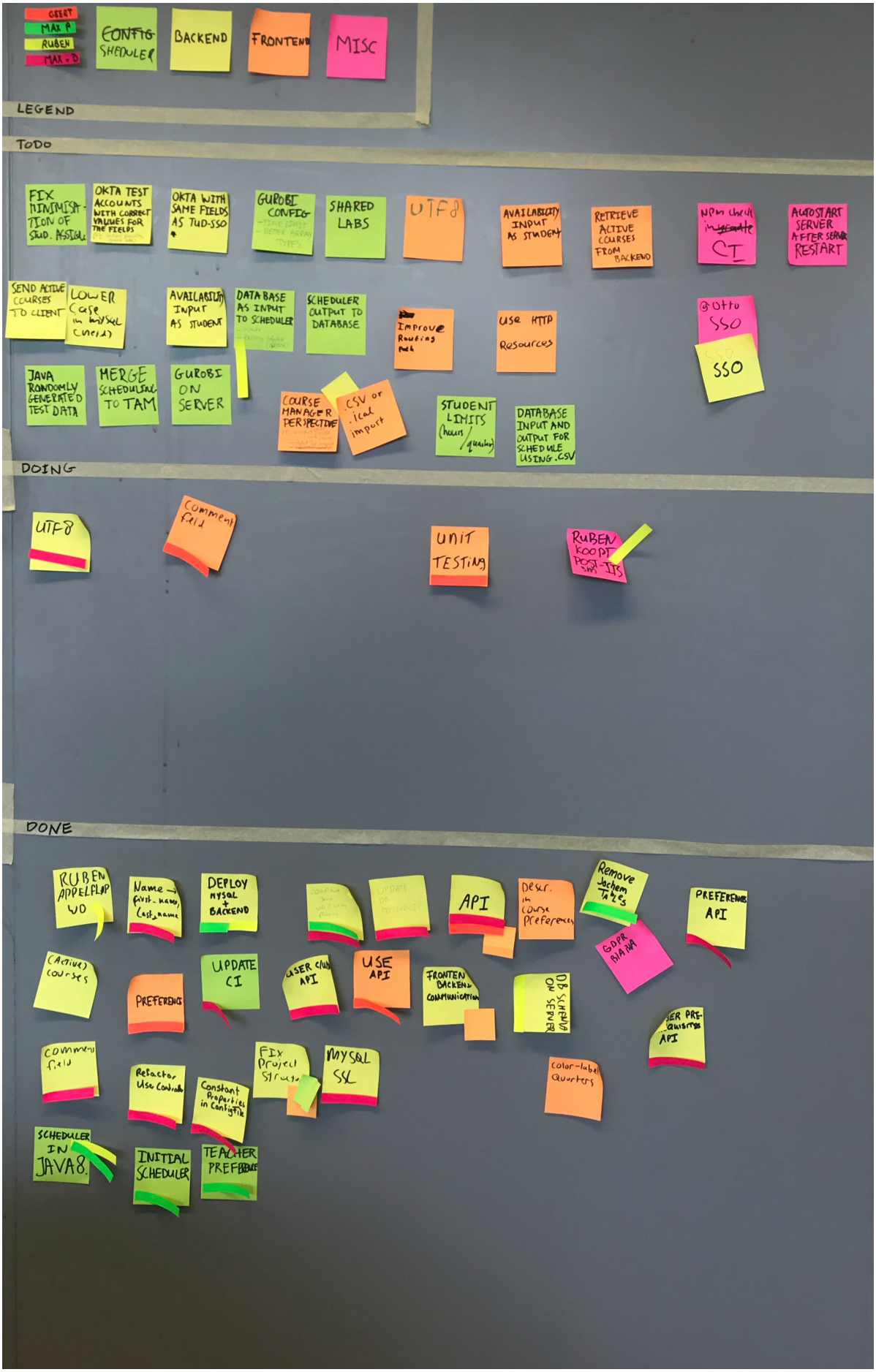


Figure 4.1: The issue tracking board. On the top left, the legend of the colors can be viewed

4.2.2. Client

To make sure that the requirements and the vision of the client for TAM are not lost in translation during the creation of TAM, the client is able to give feedback throughout the project. To accommodate this, the team plans a meeting each week.

This meeting is held on Friday by default and is based on an agenda made by the team. An example of the template used for the agenda can be seen in Figure 4.2. One of the recurring topics is a live demonstration of the new features which are developed during the week. During these demonstrations, the client is able to use and give feedback on the newly created features. The given feedback is then discussed within the meeting and is taken into consideration for the next week of development, where the features are adjusted when necessary. The team also uses these meetings to discuss possible issues encountered during the week and discusses with the client on how to handle these issues. The meetings are concluded with the client and the team discussing the new planned features for the next development week. Moreover, a consensus is reached on the prioritization of these planned features.

Hi Stefan,
We want to discuss the following points during our next meeting.

1. Demos of new features
2. Subjects to discuss
3. Direct questions
4. Planning for next week

Figure 4.2: Agenda template for meetings with the client

However, at times the team requires intermediate feedback between meetings during the week. In this case, the team communicates to the client through an instant messaging platform called Mattermost, which is hosted by the university. Through Mattermost, the client could provide feedback on the developed features when requested, even during the week.

Using both these measures, the client is closely intertwined in the development process.

4.2.3. Coach

To make sure the team is functioning correctly, the coach evaluates the team on their performance. As with the client, the team plans a meeting each week with the coach.

This meeting is held on Monday by default, where the meeting was planned and directed by the team. During these meetings, any issues encountered during the previous week are discussed with the coach. Moreover, the team asks the coach for advice on encountered issues, to receive another perspective on the solution. For example, the coach advised the team to use a more visual approach to show the newly developed features to the client. That way, the client can get a better idea of the work done during a week of development After using this new strategy for the client meetings, the team noticed that the overall structure of the meetings improved. The meeting with the client is discussed as well and the prioritization of the upcoming week is discussed.

4.3. Set up

Before work could start, a project setup had to be made such that the entire team could work with the same setup. This section briefly documents the set up process that occurred for the TAM platform.

In order to build the codebase, various technologies are used. Gradle is the main build tool used to build the complete project. Because the frontend is served by the backend, the frontend has to be built first. This is done by Gradle invoking Webpack, which builds and bundles the frontend into vanilla `html` and `js` files, which are ready to be served by the backend. Once this is complete, Gradle bundles the complete project into an executable `jar`.

Both frontend and backend rely on external libraries and packages for functionality. Gradle manages the dependencies for the backend. The frontend uses NPM as the package manager. On the frontend, NPM is used as package manager. It manages both development tools, such as ESlint, and

production dependencies, such as Vue, and ensures the entire team is using the same versions of these dependencies.

In order to ensure code quality, continuous integration is needed. For this purpose Gitlab CI is used. It is configured with tasks for both the front- and backend, and runs all available test suites and code analysis tools for both.

When using the scheduler, a license file is needed for Gurobi. This licence file is device-specific and can be obtained by requesting an academic licence on the Gurobi website.

The backend relies on a MySQL database, for which the database schema and a user account must be configured before starting the system. For this purpose, a table creation script is provided with the system and user details can be configured in the `application.properties` file.

Configurations that must be done by new developers or for deployment are also detailed in a `readme.md` file, that describes the development setup and the deployment procedure. Besides the readme, a `development.md` is also present, which contains information regarding topics that could form obstructions for new developers.

4.4. Feature Development

The project was a constant process of creating new features for TAM, in order to enhance its functionality. These features required changes on both the front- and backend and therefore communication within the team was needed. Moreover, the client played an important role in the creation of the features as well. The client extensively communicated with the team through weekly meetings, as well as instant messaging on Mattermost. Using this procedure, the team could quickly communicate envisioned changes and ensure that consensus with the client was reached on new features.

A graphical overview of the feature development process can be seen in Figure 4.3. When a new feature is first identified, relevant user stories are created in the form of post-its for both the front- and the backend.

The initial step of creating a new feature is to reach consensus on the data required and sent by the frontend for the new feature, and the specific format to be accepted by the backend.

The backend then starts it full development cycle to create the endpoints to handle the requests of the frontend. This development cycle consists of writing, testing and documenting the code additions. Moreover, if the requested endpoints required modifications in the database schema, these were put in place as well. Documenting the code additions included creating the API specifications of the newly added endpoints, in which the consensus between the data format was documented. After the new feature is implemented it is evaluated through a peer review by another team member. This team member can request changes on the current implementation, which are then fixed by the developing team member. This feedback cycle continues until none of the team members have anything to request to be changed, after which the feature is added to the codebase.

The frontend on the other hand, starts with a more analog approach. Firstly, a wireframe is created that displays the new functionality. This wireframe is a raw sketch of the webpage, providing a concept of the layout to base the actual webpage on. After consensus on the layout is reached within the team, a basic implementation of the wireframe is created. This initial version does not include any interactivity. Once this initial version is approved, the interactivity is incrementally added to the webpage. This comes together with hooking up the agreed data format to the in- and output of the webpage, to add loading in and outputting the data from the backend. After the new functionality is developed, the code is tested to ensure that the functionality stays the same after adding new code in the future. Similarly to the backend, the new feature is evaluated through a peer review by another team member after the feature has been fully developed and tested. Just as in the backend, the cycle of requesting changes and implementing these changes continues until no further changes are requested, after which the feature is added to the codebase.

Finally, the two components are checked together to ensure they work and then the new feature can be deployed.

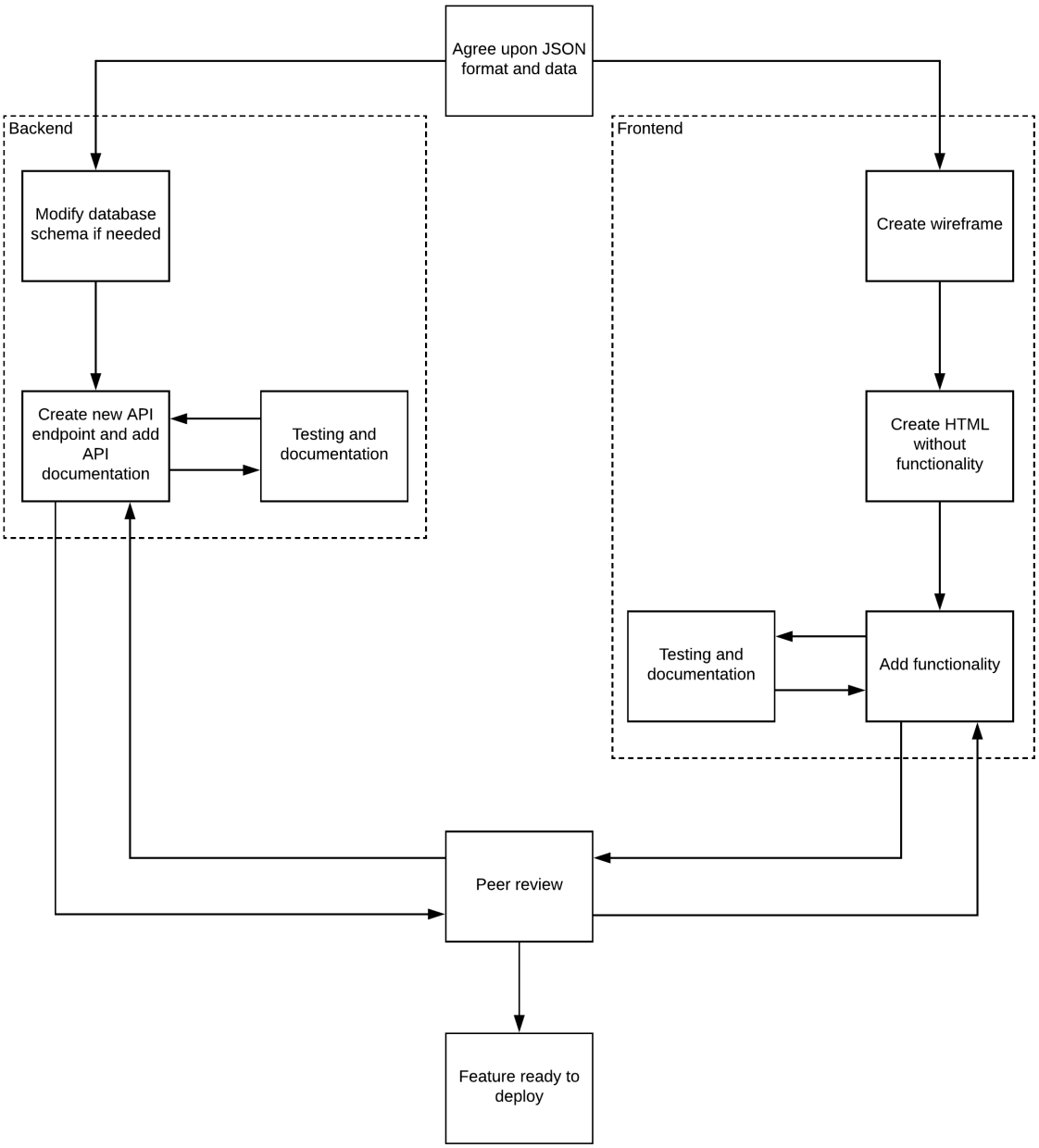


Figure 4.3: Overview of feature development process.

4.5. Deployment

The initial idea of the team was to have Single Sign-On, the SAML[1] protocol used by the university for authentication, working before deploying TAM. However, the information on configuring the protocol for TAM proved to be more difficult than foreseen. After the team had invested a lot of time into the process of configuring Single Sign-On, the client requested that a primitive version of TAM, without authentication, was to be deployed.

This was needed because of deadlines faced by the client; the information gathering process had to begin. This meant that the team had to create a placeholder with partial functionality of TAM until Single Sign-On was configured correctly. The team successfully deployed the placeholder version as soon as the fourth week of the project, at the end of the second sprint.

There were several downsides to such an early deployment. The deployment of newer versions of TAM had to be done with extreme care, as they were deployed to a live environment with real users. For example, when the database schema was modified, the team had to make sure that the existing data was not altered or lost. If the system did not need to be available live to users, the team would not have had to focus on these aspects.

A downside of the early live deployment without Single Sign-On is that newly created features could not be always deployed to the live environment, since user authentication was not possible. This meant that non-student functionality could not yet be deployed, and students could not see their personal information in the system. Care had to be taken to ensure that no functionality that could reveal sensitive data or damage the integrity of the system was available to any user.

After Single Sign-On was configured, the team could work on easing the deployment of TAM. This is because all new features could now rely on authentication of the user, allowing for the deployment of new features with appropriate set of access rights. After this, deployments became easier; while care still has to be taken to preserve and protect user data, all functionality could be deployed by protecting endpoints with authentication. Before, care had to be taken to not deploy endpoints that revealed sensitive data. Switching from development and production configuration now involves only setting TAM to production mode by adding one argument, with no other changes needed.

[1] https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language

5

Final Product

This chapter gives an overview of the final product that is delivered. First, an overview of the functionality in the final product is given. Next, the codebase for both the backend and the frontend is evaluated. This evaluation covers the structure, testing and documentation of both the frontend and the backend, together with the API specification. These sections center around Figure 5.1, which is an in-depth version of the original stack design originally described in Figure 3.2.

After the sections on the front- and backend, the final model of the database is explained and compared to the initial design. Fourth, the initial modelling of the scheduler, its conversion to code and integration into the backend are explained. Finally, the feedback from the Software Improvement Group (SIG) is discussed and reflected upon.

5.1. Functionality Overview

The final product supports the core functionality required to encapsulate the process of gathering and scheduling TAs. The supported process begins with a course manager creating a new course. This course contains basic information, like the name and the course code, the year and quarter the course is given in and the lab sessions associated to the course. Once the course is created, it is added to the overview of the course manager, showing all course editions he or she is managing.

After the courses are created, the scheduler can indicate that TA interest can be gathered for given quarters. With interest gathering enabled, students can use the platform to see all courses looking for TAs and indicate their preferences for courses to help with. The students are also able to indicate their desired size for a TA polo to wear during lab sessions.

Once students have indicated their preferences, the verifier gets an overview of all students who wish to become a TA. Using this overview, the verifier is able to register if a student is fit to become a TA and rate the students on their proficiency during previous editions of the course.

When the scheduler enables availability gathering for a quarter, the students are also able to fill in their availability on the website. Using the gathered interest and availability of TAs and the confirmation of the verifier, the scheduler is finally able to start generating a schedule for the selected quarters. Once the linear solver is finished, an optimized schedule is stored on the server.

A screenshot of the page where a TA can fill in their preferences in included in Appendix F.

5.2. Backend

The backend of the platform consists of a set of API endpoints, from which data can be requested and updated. The purpose of the backend is to supply the frontend with a way to retrieve and update data in the database, in a standardized way. Moreover, it handles the authentication of the user through SAML, the protocol used for the user authentication of the university. It is written in Java, using the Spring MVC framework, as explained in Section 3.2. This section first gives an overview of the structure of the backend. Next, the created API specification is discussed. Finally, there is a brief discussion on the test coverage and documentation of the backend.

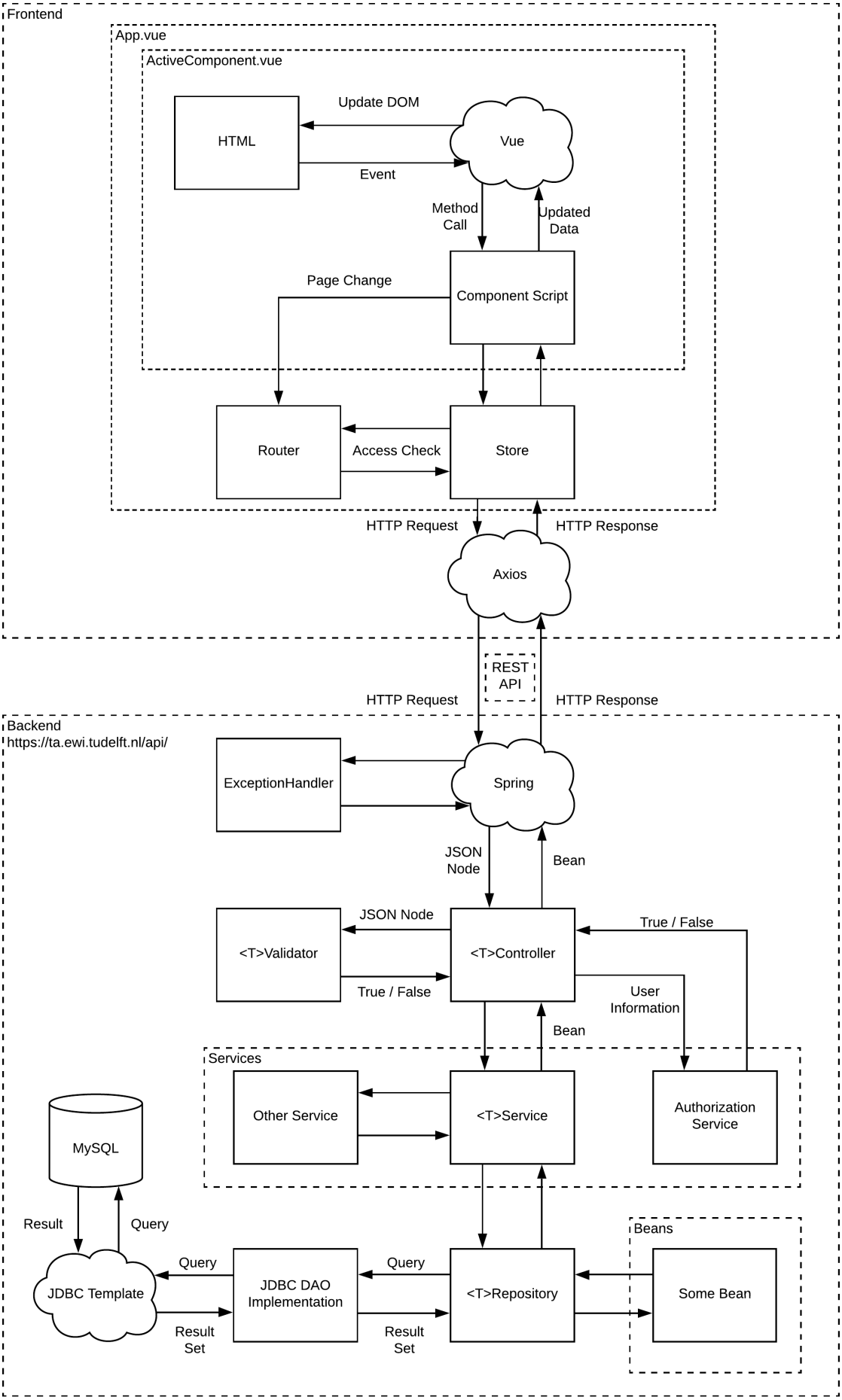


Figure 5.1: Detailed structure of the front- and backend

5.2.1. Structure

The overall structure of the backend can be seen at the bottom of Figure 5.1. In the backend, some components are prefixed with `<T>`. This indicates that there are multiple components which share the same organizational structure, but which handle different types of request. For example, there is a `LabController` which manages all requests which request, create or alter lab information. The prefix `Lab` is shared across all prefixed components, i.e., there is a `LabValidator`, a `LabService` and a `LabRepository` as well. In the description of the structure below, the example of creating a new course will be used. This example is used because it covers all relevant aspects of the structure of the backend. The HTTP request that is done for the creation of a course is the POST request. Because it is a POST request, a JSON object is provided in the body of the request, describing the course to be created. Although this subsection focuses solely on a concrete example, this is merely to clarify the process of processing a request. The other components and endpoints function in a similar way as described below.

Controller

The Controller processes the requests of the user and delegates to other components of the backend to send an appropriate response. It consists of multiple methods, with each method linked to a particular HTTP request. The Spring framework handles these HTTP requests and calls the correct method of the controller, together with the JSON data and the given query parameters if these are available.

For the example request the JSON, describing the course to be created, is sent to the `CourseEditionController`. This Controller handles every request for retrieving, creating and altering course information. The `CourseEditionController` first checks using the `AuthorizationService` to verify that the user making the request is properly authorized to do so. This depends on whether the user is logged in and whether the user is either a staff member or an admin.

When the user has either of these roles, the `CourseEditionController` continues to validate the sent JSON. This is done by handing the JSON data to the `CourseEditionValidator`. If the JSON data is formatted correctly, the Controller creates a `CourseEdition` instance from the given JSON.

Once the request has been completely validated the `CourseEditionController` passes the request to the `CourseEditionService`, which returns the requested or updated information after its computations and calls are finished. This returned information is then used to create a correct response according to the API description and this response is returned to Spring.

Validator

The Validator receives the JSON sent to the HTTP endpoint and validates this against the set format. This format is set in in the JSON Schema[1] format. JSON Schema is a standardized way to specify the format that JSON must adhere to. This technique is used to strictly set the format that was agreed on by the frontend and backend. If the received JSON does not adhere to these specifications for the HTTP endpoint, the Validator throws an error.

For the example request, the `CourseEditionValidator` validates the sent JSON. The particular JSON schema used checks whether the sent JSON specifies an owner, course code, name, teacher, study year, year and quarter. Moreover, it only allows for a description to be specified, but this is optional. If either of these fields are not specified, or other fields are specified within the JSON, the JSON is marked invalid and an error is thrown by the `CourseEditionValidator`.

Service

The Service receives the parsed information by the Controller and makes sure the request is possible. After this check, the Service can perform other business logic before sending the information to the Repository.

In the example, the parsed `CourseEdition` instance is sent to the `CourseEditionService`. Before this `CourseEdition` is added to the database and the application, a range of conditions have to be checked. The first condition which is checked is whether the `owner` variable of the `CourseEdition` exists as an user. This is because the `owner` variable describes a netid of an user, which has to be registered in the system to not create erroneous behaviour in the database. The `CourseEditionService`

[1] http://json-schema.org/

uses the `UserService` to check this condition, because the `UserService` controls the user information in TAM. Secondly, it is known that the combination of course code, year and quarter is unique for each course. Therefore, the `CourseEditionService` checks whether this combination of parameters is unique as well. This check is done through calling a method in the `CourseEditionRepository`, which returns whether a `CourseEdition` with such combination exists.

After these two conditions have been checked, the `CourseEditionService` sends the `CourseEdition` instance to the `CourseRepository`, which then returns the unique identifier of the newly created `CourseEdition`. The `CourseEditionService` then adds this identifier to the `CourseEdition` instance and returns this instance.

It has to be noted, however that one Service does not comply with the above functionality: the `AuthorizationService`. As mentioned before, the `AuthorizationService` verifies whether the user is correctly authorized for a particular request. This is done through checking whether specific properties, such as certain roles, are in the user object associated with the user making the request. In the example, this means that the `AuthorizationService` checks whether the user is either an admin or a staff member. The `AuthorizatonService` then returns whether this is the case.

Repository

The Repository receives validated data and stores this data inside the database. This is done through communicating with the database through the `JdbcDaoImpl` class, which acts as a wrapper for the `JdbcTemplate`[2] class supplied by the Spring framework. This class connects to the MySQL database through the Java Database Connectivity (JDBC) API.

In the example, the `CourseEditionRepository` receives the `CourseEdition` instance which is to be added to the database. In the `CourseEditionRepository` a SQL query is formulated with which the `CourseEdition` can be added to the database. The `CourseEditionRepository` specifies the types of the fields of the `CourseEdition` and sends this, together with the fields of the `CourseEdition` and the SQL query, to the `JdbcDaoImpl`. After the new `CourseEdition` is added to the database, the `CourseEditionRepository` returns the identifier of the newly created course.

5.2.2. API Specification

In order to document the consensus on the data transfer between the backend and the frontend, an API specification is created, as described in Section 3.3. This specification is conform the intended described functionality, as it does describe each endpoint in the backend.

The full list of API endpoints is included in Appendix C. As can be seen in the expanded `GET:/me` request in Figure 5.2, is that each endpoint comes with a description and summary of the functionality. Moreover, all parameters which are used by the endpoint are described. Lastly, a detailed overview of all the possible responses the endpoint can return is given. When JSON is returned, in the code 200 response of the request, an example structure is given. Instead of this example structure, the model can be displayed as well, which describes the data types and additional properties per set property.

Overall, the provided API has helped with providing an overview of the available functionality of TAM. Moreover, using the API, the frontend and the backend had common ground to discuss the way data was transferred between the two components. For each endpoint described in the API specification, the possible status codes which are thrown can be used by the frontend to display detailed error messages to the user.

However, in some cases, the created endpoints are too specific for the frontend. For example, the `GET:/students` request tailors to only one use case; that of the verifier. This can be generalized by creating more general endpoints on which filters can be applied to obtain the specific needed functionality.

Moreover, there are some minor inconsistencies in the use of the API. For example, the unique identifier property of a course has the `id` property at times, but sometimes `ce_id` is used instead. Although this is the case, these inconsistencies are always documented well and when the user follows the documentation, they will not encounter unexpected behavior.

Besides these minor extensibility and consistency improvements, the API specification is robust and extensively detailed, which serves as a core pillar of the future of the project.

[2] https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html

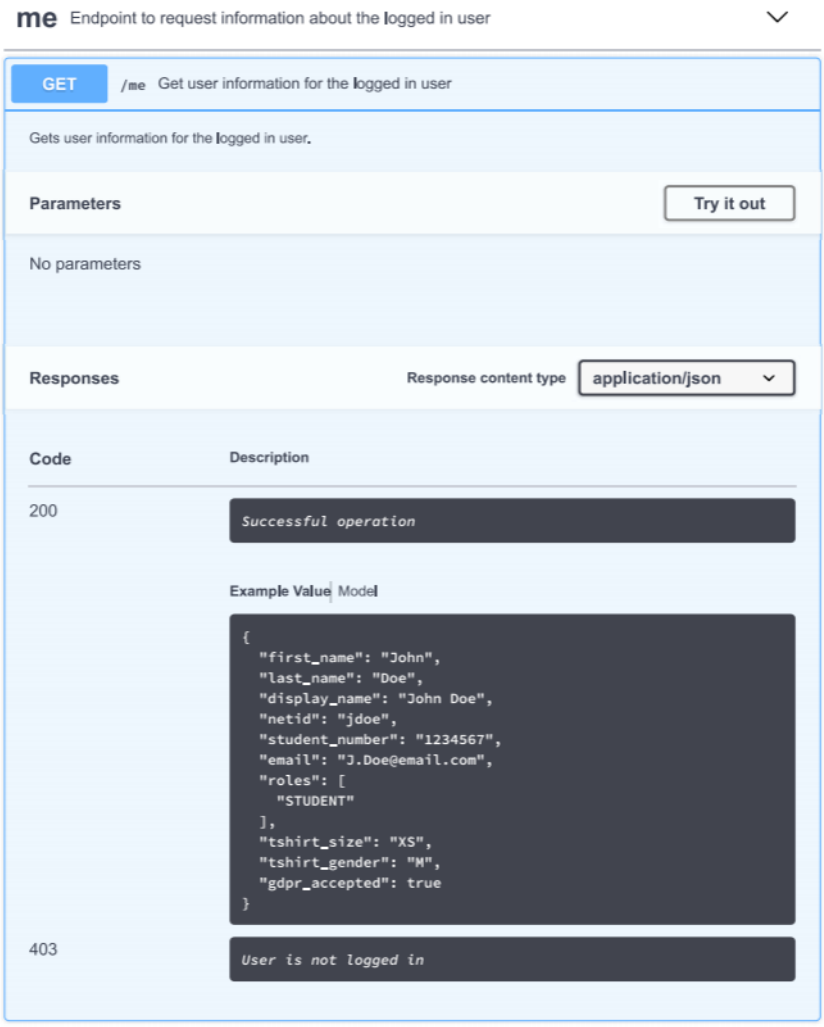


Figure 5.2: The complete /me endpoint in the API specification

5.2.3. Testing

One of the design goals set in Section 2.4.3 was to have a maintainable codebase. This in turn means that the code is both testable and tested. In the backend, the goal was to test as much as was possible using the prescribed tools in Section 3.3b. A focus was placed on branch coverage, as the team feels that this is the most accurate indicator of which code is worth testing.

The testing in the backend was done using the JUnit testing framework together with the Mockito framework. These tools were powerful enough to test the majority of the code, which can be seen in Figure 5.3.

Most of the backend has been tested. There are two reasons code is still untested. Part of the codebase interfaces with Spring and is very difficult to test, while simultaneously not being very interesting to test due to the lack of complex logic. This includes parts of the `AuthorizationService` and the SAML configuration.

Part of the scheduler-related code is also untested because it is in need of refactoring. For example, the endpoint to create a new schedule should create a new thread that will perform the actual scheduling, such that the endpoint can reliable return a response to the user instead of a timeout. The team agreed that testing this code is not worthwhile, when this structural change would break these tests immediately.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.ewi.tam.validators		0%		n/a	31	31	116	116	31	31	8	8
nl.tudelft.ewi.tam.services		78%		97%	33	176	90	389	31	103	1	13
nl.tudelft.ewi.tam.scheduler		85%		97%	40	213	88	500	34	98	2	8
nl.tudelft.ewi.tam.repositories		60%		93%	50	139	149	329	47	95	2	12
nl.tudelft.ewi.tam.exceptions		78%		n/a	6	23	12	47	6	23	5	19
nl.tudelft.ewi.tam.controllers		36%		60%	49	70	99	163	45	60	6	11
nl.tudelft.ewi.tam.beans		86%		100%	28	161	36	252	28	154	1	17
nl.tudelft.ewi.tam		0%		n/a	65	65	190	190	65	65	4	4
Total	4,074 of 10,857	62%	23 of 498	95%	302	878	780	1,986	287	629	29	92

Figure 5.3: The test coverage created by Jacoco

5.2.4. Documentation

In the process of making the codebase simple for extensibility for new developers in the future, it is key to document the codebase. In the backend, this was primarly done through JavaDoc documentation, together with some explanatory in-line comments. During the project, the tools Checkstyle and PMD, as mentioned in Section 3.5, together enforced a strict policy on the documentation amongst other code styling. This strict policy involved having JavaDoc documentation above each class, method and variable. Moreover, the method documentation includes description for all parameters, the return value and the thrown exceptions, if any.

Apart from the enforced policies by the static analysis tools, an effort is made to have a consistent way of writing documentation. This consistency is kept during the peer reviews before new functionality is merged into the system.

As mentioned before, the code is commented with in-line comments as well to clarify any parts of code that might be confusing. This makes it so that new developers do not get confused when reading the existing implementation for existing methods.

To ease the process of getting into the codebase, the backend features additional documentation, in the form of a 'developers.md', as well. This document describes the quirks of the platform which might seem counter-intuitive. Moreover, it documents workings of the product that might be overlooked by new developers and could lead to frustrations and debugging.

5.3. Frontend

The frontend of TAM consists of a website. This website is written in JavaScript using the Vue framework and styled using Bootstrap, as explained in Section 3.3. This section first gives an overview of the structure of the frontend. After, there is a brief discussion on the test coverage and documentation of the frontend.

5.3.1. Structure

The frontend of TAM consists of three pages. First there is the index page, which is served when a user visits the website. This page is a static page which explains the goal of the platform and links to the Single Sign-on service. If the user is already signed in using SSO, which is the case if he or she visited another website using SSO during the same browser session, the index page is not shown and the user is sent directly to the next page.

When a user visits TAM for the first time, the credentials page is served after logging in. On this page, the user can check if their credentials are correct and have to accept the GDPR. After the user has confirmed their credentials and has accepted the GDPR, the full frontend application is served.

This is a single-page application (SPA) as the Vue framework focussed on the development of single-page applications. With an SPA, the entire webpage is sent to the user only at the start of the session. Any requests to the server change the data that is displayed, but these requests do not result in a new webpage being sent. This means that the amount of data being sent from the server is minimized.

Components

The application is structured using components using the Vue framework. By using these components, the logic and design of different pages or items on a page can be split and reused. This results in the code in a components only being responsible for one subject and thus helps the understandability of the codebase. If the amount of logic included in one component becomes too large, a part of the logic is moved to a child component. An example of this is the editor for the lab sessions on the overview

page of a course. By moving the logic for this editor to a child component, the logic involved in the overview page stays understandable and keeps its focus on only one subject.

Routing

Because the developed web page is an SPA, changing perspectives has to be handled within the application. To provide the logic of replacing the component shown to the user, also referred to as the active component, the Vue router is used. This router is responsible for swapping out the active component. The router is also responsible for preventing the user from visiting a component he or she is not allowed to visit, like a student accessing the component used by the verifier. Before the router links to a component reserved for a specific role, it queries the store to check if the current user has the role required. If this is the case, the component is shown to the user. Else, the request to show the component is rejected and the current component is shown. Components not reserved for a specific role are always directly shown to the user.

Store

Shared data, like the information about the current user, is saved in a central store. This store is provided by Vuex, an extension of the Vue framework. Besides saving the data used by the components, the store is also responsible for retrieving data from the server and storing the data retrieved from the server. Once the data is retrieved from the server, the store saves it and provides access to it to all components. When sending new data to the server, the store also updates the local copy to provide the components with the updated data.

HTTP Requests

The application communicates with the server through a HTTP adapter, Axios. Axios provides a standalone HTTP adapter which can easily be configured to our needs. This instance has to be decoupled from the Vue components because it also has to be accessible form other parts of the frontend, like the store. Upon recieving a response from the server, Axios automatically converts the response body, which is in the JSON format, to a JavaScript object. This way, the responses can easily be used by the application.

Styling

To provide styling for the application, Bootstrap is used. Using the Bootstrap-Vue package, complicated objects, like interactive tables, can easily be controlled from the component while also being styled by Bootstrap. The themes can also easily be modified by changing the default settings from Bootstrap.

5.3.2. Testing

For testing the frontend, the Jest framework is used, as described in Section 3.5. This framework supports all required aspects of testing: asserting conditions, mocking components and collecting coverage of all executed tests.

The test suite for the frontend consists mostly of unit tests for the frontend. The reason for this is that most of the components are designed to work independent of each other. Testing of the components focuses mainly on the logic inside the component. This logic is responsible for linking the data from the server to the DOM with which the user interacts. By mocking external dependencies, the tests can focus completely on the logic inside the component being tested. Since the Vue framework handles the interaction between the logic in the component and the DOM, only a limited amount of time is spent testing this interaction. The coverage report generated by Jest is shown in Figure 5.4.

File	% Stmts	% Branch	% Funcs	% Lines
All files	89.6	87.54	85.27	89.81

Figure 5.4: The test coverage created by Jest

5.3.3. Documentation

The frontend is documented using JSDoc, a documentation standard similar to JavaDoc. Documentation is added to each function in the codebase and includes a description of the behavior of the function, the input arguments and the returned value. For methods linked to DOM elements via Vue, the element linked to the method is also stated. Comments on inline functions, like functions passed when sorting, are only provided when the behaviour of the function cannot easily be derived from the context.

By configuring ESLint (see Section 3.5), both the presence of documentation and the style of documentation is enforced. The configuration of ESLint is not able to enforce the presence documentation for functions inside the modules of the store. Documentation in these parts has to be enforced via peer review.

Additional comments are provided on select parts in the codebase, explaining the handling of corner cases or clarifying complex parts of code. This eases the process of understanding the full behaviour of all functions.

5.4. Database

The initial database schema, found in Figure 3.1, was designed before any features were implemented. This design was used at the start of the project, but was altered during the project because additional functionality was required. The final database schema can be viewed in Figure 5.5. In this section, the initial design the database schema is contrasted with the final version. The areas of change are identified and the reasoning behind these changes are explained. These changes are split in two categories: additions and structural changes.

5.4.1. Additions

The changes in this subsection are changes to the database schema that are simple additions to the initial database schema. These changes catered to functionality which was introduced during the product and which was unforeseen when designing the initial database schema.

The majority of the changes have to do with submitting preferences by the students. For this part of the process, the client requested two additions to the original functionality. The students have to be able to submit notes for each quarter, which can be used for additional information or questions. Moreover, the students have to be able to submit whether they have been a TA for the course they prefer to become a TA for, because no information of previous years is known.

To accommodate these two additions, two additions to the database were made. The `Note` table is added to the database, as can be seen in the top right of Figure 5.5. In this table, the notes of a user can be saved for each quarter. In addition, the `last_year` column is added to the `PersonalPreference` table. This column contains a boolean value and resembles whether the TA was a TA for the course in a previous year.

Another change was made after the SSO worked in TAM. At this time, TAM could receive user information from the university. This information did not only include netid, first name, last name and email, but a display name and possibly a student number as well, if the user was a student. The latter two were not present in the `User` table of the initial database schema. However, the student number is commonly used by the Verifier in the process of looking up students. Apart from this, the display name might be different of the combination of first and last name and as the name suggests, this field is to be used as a display name in applications. To account for these two previously unsaved fields, they are added to `User` table.

Lastly, the `Timeslot` table used integers for slots, which were then defined by the team as in figure 3.1. However, the team thought it was better to make these slots explicit in the database. Therefore, the `Slot` table is added, which includes the start and ending time of each slot. The `Timeslot` table now uses the identifier of the `Slot` table as a foreign key and is therefore forced to only use the defined slots.

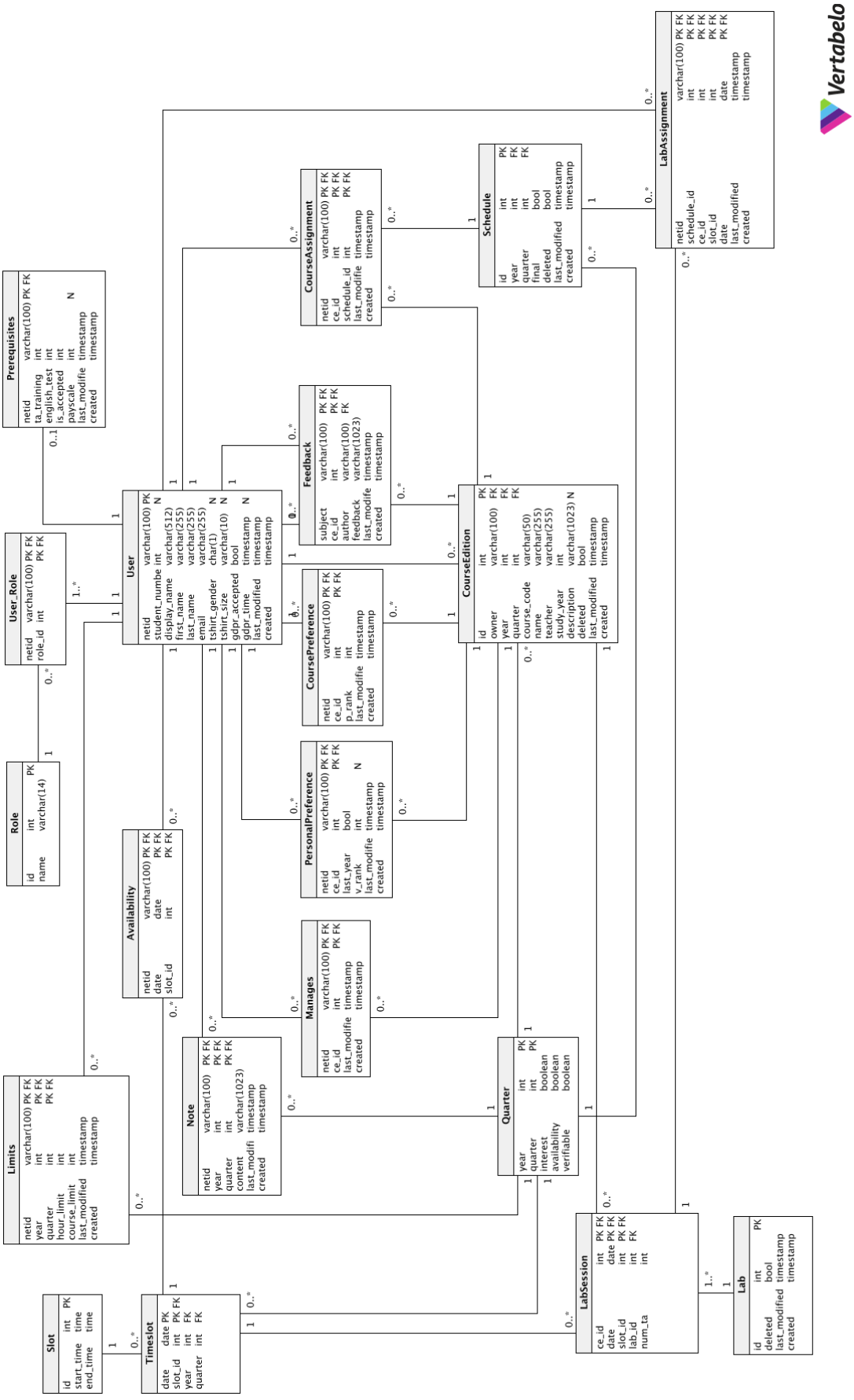


Figure 5.5: The final database model.

5.4.2. Structural Changes

Apart from the additions to the database schema, some structural changes were made during the course of the project. These changes were made when the initial schema required redesigning before the envisioned functionality could be implemented.

The first improvement which was made is the introduction of the `Quarter` table. In the initial schema, the `Schedule` table contained instances of schedules, which in turn described the state of the quarter. For example, the `Schedule` table stated whether the quarter allowed students to submit preferences for courses in the given courses. This database schema did not take the existence of multiple schedules for one quarter into account, which would lead to unexpected behavior. Moreover, multiple tables contained a `year` and `quarter` column, but these tables were not linked to the `Schedule` table explicitly in the initial database schema.

To make the link explicit between year and quarter, as well as a unique state per quarter, the `Quarter` table is introduced. This table had the `year` and `quarter` columns as primary key. This composite primary key is used in each table which contains a `year` and `quarter` in the form of a foreign key. The tables which include these two columns are: `Limits`, `Note`, `CourseEdition`, `Timeslot` and `Schedule`. Moreover, the state of the quarter is moved from the `Schedule` table to the `Quarter` table as well. This state is included in the `interest`, as well as the newly introduced `availability` and `verifiable` boolean columns. The `interest` column indicates whether students are allowed to submit preferences to courses in the quarter. The `availability` column indicates whether students are allowed to submit availability for the quarter. Lastly, the `verifiable` column indicates whether the Verifier is allowed to rank students for courses in the quarter.

Another small improvement with respect to quarters was made within the `Limits` table. The team added the `year` and `quarter` column to the composite primary key to make it possible for a student to submit different limits per quarter.

A redesign for the sake of clarity was made to the `Timeslot` table. In the initial schema, the `Timeslot` table had a `id` column as its primary key. However, this column had no relation to the contained information that it referenced and the `Timeslot` table had another unique key, namely the combination of `date` and `slot`. The primary key is therefore changed to be the combination of the `date` and `slot` column.

Another improvement made in the database was the representation of labs. In the initial schema, there was only the option to specify the amount of TAs a lab wanted per timeslot. However, the team wanted to provide the functionality to specify the amount TAs per course per timeslot. This became impossible if the lab consisted of multiple courses, since only one number of TAs could be specified per timeslot. Because of this, a `LabSession` table is created which resembles a timeslot per course per lab. Using this table, the number of TAs per timeslot per course for each lab can be specified. The primary key of this table, which is a combination of the primary keys of both the `CourseEdition` and `Timeslot` primary keys, is used in the `LabAssignment` table. In this way, it is known to what course and timeslot in the lab the student is assigned.

5.5. Scheduler

This section discusses the scheduler. First a reflection on the initial modelling is given. Next, the translation to Gurobi is discussed. Finally, the integration of the scheduler into the backend is discussed.

5.5.1. Modelling

Creating an implementation using Gurobi was done based on the modelling described in Appendix B. Over the course of implementing the model, we realized that the initial model was incomplete. In the following sections, the changes made to the model as well as possible future improvements are listed and discussed.

Two extra constraints were added to the model in order to make it more complete. These constraints deal with giving students the ability to impose limits on the amount of hours and/or courses they want to TA. The two constraints are very similar in structure; *A student cannot be scheduled to more courses than he wants to TA for* and *A student cannot work more lab sessions per week than he wants to*. For

these constraints, a matching table is needed that holds, for each student, their limits. Below is the constraint along with the needed data for the course limits; the hour limits are similar.

- SLC_i: Integer $S \times 1$ matrix that indicates the amount of courses a TA wants to assist at once.
- A student cannot be scheduled to more courses than he wants to TA for:

$$SLC_i > 0 \Rightarrow (\sum_{j \in C} TC_{i,j}) \leq SLC_i$$

The second soft constraint, (*It is better for a student to be assigned fewer courses*), was incorrectly modeled. In the original form, the constraint expressed the same as the first soft constraint. The constraint should have been modeled quadratically:

$$\forall S_i \in S: \text{ Minimise } (\sum_{p=0}^{|C|} TC_{i,j})^2$$

This expression means that assigning two students to one course each is better than one student to two courses, which was the intended design of the objective. While it is possible to solve problems with quadratic objectives, it is not possible to use a quadratic objective in a multi objective model using Gurobi. We could not find any solvers that have this functionality. Therefore, the current scheduler does not take this constraint into account.

5.5.2. Translation to Gurobi

It took some time to understand the code structure needed to translate these constraints into Gurobi, but once this was found translating most of the constraints and objectives was straightforward.

Converting the written model to an implementation that fit Gurobi was straightforward. During the process small changes were made to several constraints in order to improve accuracy or efficiency. These changes are listed in this section.

Several constraints made more efficient by adding sanity checks to check whether it is even possible to violate them. For example, look at the first hard constraint (*1. A student cannot be scheduled for multiple labs during one timeslot.*). If there is only one lab during a certain timeslot, this constraint can never be violated and does not need to be considered. Similar checks were added to the student hour limit and student course limit constraints.

Hard constraint 4 (*A student must be assigned as a TA to a course if the course has indicated so*) was changed to take into account the students preferences. It can only hold that a student must be assigned to a course if that student also indicates that he would like to assist the course, and is approved by the verifier.

5.5.3. Integration in System

The scheduler is integrated into the backend and works similarly to the other endpoints available. When the endpoint for the scheduler is invoked, the backend reads the data of quarter to be scheduled from the database. It converts this data into the matrix format that is expected by the scheduler to create the linear model, as described in the previous section, and then invokes Gurobi to solve the model. Once the model has been solved, i.e. a scheduling has been made, the scheduler converts the assignments back into the database. Finally, the student assignments and statistics such as the number of TA's per lab session and hours worked per TA can be printed to a CSV-file for further parsing. Integration into the frontend is not yet complete, as will be discussed in Section 7.1.

5.6. SIG Feedback

In this section the outcome of the code quality evaluation from the Software Improvement Group[3] (SIG) is discussed. The code base is evaluated two times throughout the project. The feedback on the initial version was received in week 8. The feedback on the improved version is still to be received and will be included before the report is uploaded to the TU Delft Research Repository[4].

5.6.1. Initial Feedback

The code base was uploaded to SIG on Friday June 1st for the first evaluation. On June 11th, the feedback from SIG was received; the original Dutch email is included in Appendix D.1. This section summarizes the feedback in English, and explains how the team handled the feedback.

The code base scored a 3.9 out of 5 on their maintainability model, indicating an average market maintainability. Unit Interfacing was the only aspect indicated as a point of improvement. In order to improve the code, the amount of parameters for certain constructors needed to change, because a lot of parameters indicate the lack of abstraction. For example, the `CourseEdition` class was described as a class to improve upon, since it has a lot of parameters, including strings which could be replaced by objects. Apart from the Unit Interfacing improvement, no other improvements were suggested. A remark was made that the test suite looked promising.

Based on this feedback, a reply was formulated regarding the Unit Interfacing issue. In this reply it was indicated that the large constructor was inherent on the architecture TAM uses, a hand crafted database schema and data classes (Beans) to map the data onto. It was not clear if this was taken into account during the evaluation process. Furthermore, a more detailed explanation of the composition of the grade was asked for. The advice to change the strings to objects was not followed up, since it does not make sense in our object relational mapping to make objects for these parameters.

SIG replied quickly with additional explanation regarding Unit Interfacing, they recommended the use of setters, whilst leaving the constructor empty. The team agreed that the proposed solution was indeed an improvement and it would make the code more clear, therefore it was adopted. Regarding the composition of the grade no explanation was given other than information on the general grading model used.

Another reply was sent, asking again for the composition of the grade and more aspects to improve upon, but a reaction on this email is not received.

5.6.2. Final Feedback

During the development of the rest of the project, further improvements on the codebase were made. The feedback on the final codebase stated that the codebase improved further in every aspect. This feedback is included in Appendix D.2.

[3] https://www.sig.eu/
[4] https://repository.tudelft.nl/

6

Ethical Implications

During the development of every software project, important ethical considerations are faced. This especially holds for a project involving live deployment, where it is important to handle user data responsibly. In this chapter, three topics with ethical implications are discussed. First is the topic of storing personal information. Second comes the topic of access to personal information. Finally, the potential bias in the scheduling process is discussed.

6.1. Storing Personal Information

TAM stores basic personal information such as name, student number and email as well as detailed information like payscale and shirt sizes. It is therefore important to design the system with the GDPR (General Data Protection Regulation) in mind. The GDPR is a regulation constructed by the European Union and concerns personal information storage and handling on (online) platforms, and was adopted on 28 May 2018. When a user signs up to use our platform, they have to actively agree with their personal information being stored. Only the minimal amount of data that is needed for our the functioning of our platform is stored, and this data is not shared with any third party. TAM complies with the various individual rights, such as the 'Right to Access' and the 'Right to Erasure' by an admin manually resolving such requests via direct database access. Due to the small amount of users, not more than a few hundred, an automated and integrated system was deemed to be unnecessary.

One point of concern we discovered is that the backups of the system database are not GDPR compliant. The client relies on database backups made by a central entity of the Delft University of Technology, and there is no way for the client to access and modify these backups to comply with, for example, the right to erasure. Restoring a backup could thus add data, that was deleted or modified by user request, back into the database. This was brought to the attention of the client, but solving this issue is outside of the scope of this project.

6.2. Access to Personal Information

As the project included a live deployment of the system managed by the team, the team had access to the production database containing personal information of over a hundred students. The team formally agreed with the client to handle this data confidentially and with care. Entries in the database, other than test entries created using the netid 'test' and our own personal entries, are not to be touched and only checked to verify the correct workings of the system. Access to and information from the database should not be shared with anyone outside of the client and the team. The team should also no longer have access to the live system after completion of the project.

Access to personal information is also a key component of the functionality of TAM. All data in the system is accessible by only the minimal set of people needed. For example, payscale information of a student is highly confidential and may only be seen by the student themselves, the verifier who sets the payscale into the system, and admins of TAM. Authentication requirements are checked for each request made to the backend. It is important to maintain strict access rights in case of integrations of TAM with other platforms, as will be discussed in Section 7.1.

6.3. Scheduling

The TA selection process includes factors such as course manager preferences and verifier rankings. An important note here is that the client is both the scheduler and the course manager for all courses in the Computer Science Bachelor. During the old scheduling process, the preferences of the course manager were not explicitly stated. They only appeared implicitly while the scheduler was choosing the students to assign to each course. When the personal opinion of the course manager or scheduler was given, it could be unclear why certain choices were made.

In TAM, the course manager and scheduler roles are split. The preferences of the course managers are submitted before the scheduling process begins, marking a clear distinction between the two roles regardless of who performs them. The automated nature of the scheduling algorithm means that all factors that influence the automated process need to be explicitly stored in the database, and the model is explicitly created in the code.

There are two important advantages of the automated system over the manual system. Firstly, it means that any generated schedule can be recreated to demonstrate that the schedule was fairly formed from its inputs by applying the model, barring randomness inherent to a linear solver. Secondly, it makes it possible to explain confidently to a student why they were (not) selected, if the need for such an explanation arises. We should note that it is still possible for the scheduler to make manual adjustments after the schedule has been generated. This does still leave the possibility of bias being introduced by the scheduler, but this option is necessary to accommodate for any features that the automated scheduler might be missing, or for small changes to input data which impact the schedule but do not warrant a complete rescheduling.

7 Discussion

Although our BEP project is now finished, there are many new features that could still be added to the TAM platform. This chapter begins with an overview of future work that could be done to improve TAM. This overview is followed by a reflection on the different components of TAM. After this overview, we discuss the problems we encountered and adjustments that we made during the project, involving Single Sign-on, the development process, the weekly meetings and the object-relational mapping.

7.1. Future Work

In this section, the recommended future work is presented. It starts by describing the additional functionality for each role described in Section 2.4.1. After this, the future work which has to be put in for continuous deployment is discussed. This section is concluded by discussing the possible integrations between TAM and numerous other applications.

7.1.1. Roles

In section 2.4.1, five roles are identified. For each of these roles, future work is addressed in this section. Note that the future work described in this subsection is not thought of at the end of the project, but was already considered at the start of it, as can be seen in Appendix A. Because of this, the database has been designed with these features in mind. Creation this features would therefore be considerably less work than when the database schema has to be modified as well.

Course Manager

Currently, the Course Manager has to manually add each lab to TAM, essentially copying the time and date from the MyTimetable of the university to TAM. To automate this process, functionality could be added that makes use of MyTimetable to load in the labs into TAM of the specified course.

In the current version of TAM, there is only one user which can manage a course. In the future, this could be extended by allowing the creator of a course to assign manager roles to additional users in the system, which can then add labs and manage the course as well.

The Course Manager could be able to specify preferences for students which are interested in TAing for their course in the future as well. This can be done through requesting all students which submitted preferences for the course and setting a preference rank for each student.

During the course, TAM could provide the Course Manager with a way to notify all TAs for his course, either via email or via Mattermost. This could then be used to notify all students, for example about a new assignment that is up for review. This functionality could then be extended even more to notify all TAs in a specific map, which can then be used to message all students if for example the location of a lab is changed.

After the course is done, other functionality that the Course Manager could use is to provide feedback to students. An interface can be provided which allows the Course Manager to write an additional note to a TA wherever the Course Manager deems useful. This feedback can then be displayed to the Course Manager in the next edition of a course, which helps the Course Manager in selecting the correct TAs for their course.

Teaching Assistant

The main feature which the Teaching Assistant currently does not have is a complete overview on the dashboard. Although the dashboard now features all functionality available to the Teaching Assistant, there is no way of viewing the courses or labs they have been assigned to. The latter could be displayed in a personal schedule, in the form of a calendar. This calendar functionality could be improved even more by integrating this created calendar with assigned labs into other calendars such as Google Calendar. This can be done through exporting the assigned labs to the `ical` format, which can be used by all the major calendars.

If a Teaching Assistant is unable to attend a lab that they are assigned to, there is currently no way to inform the Course Managers through TAM. This means that the Teaching Assistant still has to manually contact the Course Manager at the moment. In the future, this could be improved upon by creating a button to notify the Course Manager in this event.

Verifier

The Verifier already has interfaces to completely rank and verify the prerequisites of a student. However, the Verifier can only verify that a student has done the English Test by receiving a certificate from the student. In the current version of TAM, this certificate cannot be sent through TAM, but instead has to be sent to the Verifier separately. This can be integrated into TAM in the future by allowing students to upload their certificate into TAM. These uploaded files can then be served to the Verifier, who can use them to approve or deny the English Test status of the student.

Scheduler

In the current TAM, the scheduler only has two endpoints in the backend, which can be used to respectively create a schedule and export a finished schedule to a csv file. Ideally, the Scheduler receives an interface in the frontend which can be used to create, alter and finalize a schedule. This page would then have an option to load a created Schedule, with an overview for each lab, which students are selected. If the Scheduler then wants to alter an assignment manually, he can get an overview of alternative students which are most suited for the lab. The Scheduler can then finalize this created schedule, which means the schedule is used in the other components of TAM.

Other than the interactive schedule creation, the Scheduler could get an overview before the schedule is created as well. This overview can include a statistics page, which offers insight on the amount of possible TAs per course. With this overview the Scheduler can estimate whether there are enough TAs to create a usefull schedule.

Admin

In the current version of TAM, the Admin is allowed to manually do all API requests which are available in TAM. Ideally, the Admin has a perspective in which he can get an overview of the whole system. In this overview, the Admin can moderate all information put into the system. Moreover, through this overview, the Admin is able to delete all user information from the system if this is requested by the user.

Other than this, TAM could offer an impersonation functionality for the Admin. This can be used to review optional problems that occur to a specific user. The Admin could then impersonate the user through TAM to investigate what problem occurs. This can help to debug TAM in case something goes wrong.

Scheduler

There are many improvements that could be made to the scheduler. As discussed in Section 5.5.1, the objective to minimize the amount of courses per TA is not currently in the implementation. Finding a way to either model this as a linear constraint or include a quadratic constraint in a multi-objective Gurobi model would be a good improvement.

Student experience is currently stored in the database, but is not included in the model, because the team, in conjunction with the client, came to the conclusion there is a simple universal metric is not possible. A possible implementation would be to minimize: $|\#experiencedTAs - \#inexperiencedTAs|$ to aim for a fifty-fifty ratio. This formula could be adapted for different ratios by applying factors to the two sides. It would most likely be desirable to supply the desired ratio on a course by course basis.

Another interesting addition would be the ability to load in an existing (partial) schedule, to use as a fixed base for further scheduling. Examples where this would be useful could be adding more

TAs to certain labs after a schedule was already made, or after manually removing some TAs who are no longer available and finding replacements for them. This could be implemented by loading in the assignment matrices of the existing schedule, and setting any made assigments to be forced to be taken again in the new schedule.

7.1.2. Continuous Deployment

The current deployment process is described in section 4.5. Ideally, this process is done automatically. This can be done by listening to the GitLab repository for changes. If the current version is changed, the server could be updated automatically using a script. This script could then do the actions done in the current deployment process, but without any human interaction.

Before this is done, the infrastructure to automatically update changes to the database schema to the database on the server as well. The options to do this automatically have to be investigated further before continuous deployment can be realized.

7.1.3. Integrations

At the moment, TAM is a standalone platform which improves the process of recruiting and scheduling TAs. However, there are multiple in-house standalone applications with which TAM could integrate to increase its functionality. Moreover, the data duplication across these applications can be reduced by introducing integration between different applications.

The first in-house application that comes to mind is Queue. Queue manages a question queue for each registered lab in its system. The TAs for the registered lab have to be assigned manually to the lab in Queue, after which they can answer and close questions in Queue. This same lab is registered in TAM and contains all assigned TAs. An obvious integration between TAM and Queue would therefore be to automatically assign the assigned TAs for a lab in TAM to the lab in Queue.

A similar integration can be achieved between TAM and the Complete Project Monitoring (CPM) application. CPM manages the signing off of assignments handed in by students for particular courses. The integration with TAM could be to automatically add the TAs which are assigned to a course in TAM to the TAs in CPM.

Another cumbersome process for courses is to pay the TAs. For each course, the student has to be registered as a TA. This has to be done through FlexDelft[1] and involves filling in a lot of information about the student. This registration can be fully automatized using the information already available in TAM. This automation can be done by registering a student for each course to which a student is assigned in TAM into the FlexDelft application. After the TA is registered, the TA has to submit their worked hours per week, which the course manager is then to confirm. This can once again be automatized by using the information within TAM to automatically approve or deny the submitted hours by each TA.

Lastly, all TAs currently employed use Mattermost as their instant messaging service to communicate with one another. TAM could integrate with this application as well by sending push notification when particular actions are required. For example, if a substitute TA is required for a particular lab, because one assigned TA was unable to attend this lab, TAM could automatically send a push notification to Mattermost. A more advanced integration could be to use emoticons or some other kind of response to automatically replace the TA in the system when another student is able to substitute the TA.

7.1.4. End-to-end Testing

TAM currently has unit tests for both the frontend and backend in place, however no end-to-end tests are included. End-to-end tests can help avoiding bugs during deployment, which cannot be solved by unit tests. For example, when a mistake is made when documenting the API, this can lead to the front- and backend being unable to communicate of that specific endpoint. With end-to-end testing, this phenomenon can be avoided.

[1] https://flexdelft.nl

7.2. General Reflection

During the development of TAM, several issues arose. This section describes both the problems encountered during the development live deployment of TAM and the points which were learned from.

7.2.1. Backend

The development process of the backend encountered a couple of issues, starting with the structure of Spring. Due to the lack of experience with the framework, the way Spring handles Java Beans was not understood. Besides, the default pattern used by Spring was not known, which resulted the controller, service and repository to be merged into one component (see Section 5.2.1). Once it became clear how the Java Beans and the structure in Spring should have been implemented, the codebase was refactored to incorporate the desired structure. After these problems were solved, the development process of the backend continued smoothly. Support for new components could quickly be added whilst keeping the codebase tested and maintainable.

7.2.2. Frontend

The development of the frontend also encountered multiple problems. To begin, any experience with frontend development was not present within the team, which proved to be a major issue. Combining this lack of experience with learning the framework used, Vue, caused the development process of the frontend to start of slowly. Once development got underway, multiple problems were encountered.

First, it was realized that some form of central storage was needed to share data, like information on the current user, between components. This meant that the codebase had to be refactored to incorporate the Vuex store. The initial implementation of the store turned out not to be completely desirable since the store wasn't connected to the server. This meant that any update of the data had to be manually updated to both the server and the store. Solving this issue was done by moving the connection to the server to the store itself. This way, updated data only has to be communicated to the store, which beside updating its local copy also relays the data to the server.

Second, high workload during the start of the project (see Section 4.5) caused a lack of documentation in the frontend. Since the documentation was lacking in the first version of the codebase, development of new components continued without documenting the new changes. Even though this issue has finally been solved once the development of new features was stopped, it should never have been an issue to begin with.

7.2.3. Deployment

Experience also lacked when it came to deploying software, which was a part of the project. This caused mistakes, like forgetting to update the database schema, to be made. As the project progressed, mistakes became less and less common. The deployment process also gave an insight in the processes and technologies used, like connecting to a server using an SSH-session and running processes through `tee` and `screen` to copy the output logs to an external file and keep the process running when the SSH session is closed.

7.2.4. Scheduler

At the start of the project the team had no experience with using linear solvers. Because the scheduler is only a component of the system, and time had also been spent on minimal-cost max flow, it was difficult to justify spending large sums of time on researching linear solvers. Fortunately, through studying example programs and some brief research it was possible to quickly create a working initial version of the scheduler. This initial version was made using Python, as this seemed to be both the easiest language to use for rapid development (that Gurobi was available for), and Python has the most extensive documentation on the official Gurobi website and other discussion forums.

While Python does still seem like a better choice for getting familiarity with linear solvers, the effects of this choice can be seen in how the final scheduler implementation is designed. For the Python implementation it was easiest to structure the input to the model as matrices (2d-arrays), which is also the way taken by most of the example programs. On top of being the easiest, it was also very similar in structure to how the original modelling expressed the constraints, making them easy to convert. However, after changing over to a Java based scheduler for easier integration with the rest of the backend, we kept this matrix-based structure. This means that when scheduling, the following steps

happen:

1. Database entries get converted to Java objects
2. Java objects get converted to matrices
3. Matrices are parsed to create the constraints and objectives given to Gurobi

In this pipeline, the second stage does not serve much purpose and could probably be removed. Rewriting the scheduler to instead parse the Java objects straight into constraints and objectives could improve code readability and remove a needless abstraction layer.

We are happy with the choice of Gurobi. The documentation was useful, although the Java documentation seems to be less complete than for other languages and has less example programs available than for example Python. One thing to note is that since this is our first experience with linear solvers, we might have gotten used to certain functionality that other solvers might not have, that are default in Gurobi. This will be something to find out in the future. Regardless, this experience will be useful as a foundation for the future.

7.3. Single Sign-On

The biggest issue in the project was integrating SSO into TAM. It was expected that the configuration used by the prototype version of TAM could easily be combined with the existing implementation in Queue to create a working version. Due to a bad configuration in the metatdata needed, this didn't work as assumed. Attempts to solve the issue took over one week, but a solution was not found and support for SSO was dropped for the first iterations.

Integration with SSO was picked up again when the final list of features to be implemented was created. In the mean time, contact had been made with the external team responsible for SSO and finding a solution for the problem was deemed possible. After contacting a developer responsible for SSO, a new configuration was created. Using this new configuration, the login service finally worked properly and could be integrated into TAM.

Because of the live deployment of TAM, the platform had to be designed to work without the authentication offered by TAM. This meant that most requests on the server had to be disabled to prevent users getting access to the data, limiting the capabilities of the frontend.

In the end, over 80 hours have been spent to get SSO working for TAM and even more time has been lost due to the consequences of SSO not working. Most of this time was lost during the initial phase of the project, causing other issues like a lack of documentation in the frontend. The problems could have been prevented if a working version of SSO in Java for the website was provided when starting the project. This should be a taken in consideration by the client if similar projects are organized in the future.

7.4. Development Process

The codebase is dependent on external resources, such as the database and Single Sign-On, to function properly. Usage of these resources is however limited to the production environment and can and should not be used during development. In order to develop new features, several alterations have to be made.

Backend

The first dependency of the backend is the database. In order to provide easy access to the database, a local instance is used, allowing for easy access to records stored and created by the backend. Setup of the database is described in a readme file and a script containing test data is provided.

After the issues with SSO were fixed, the backend also became dependent on SSO to provide information about the user making the request. Because SSO cannot be used for a local instance due to security reasons, the production configuration for the backend was not valid for development. To solve this issue, a separate development security configuration was created and used instead of the production configuration when the backend was started on a local machine. This development configuration disabled most security checks, resulting in access to the API without any authentication. Besides, a test user is loaded from the database to act as the current user, allowing the `/me` request to be used during development.

Frontend

The frontend depends on the API to function properly. During development, this API is provided by the backend, running in development mode.

To enable live reloading of changes in the frontend during development, the application is served through node instead of the backend. This causes warnings for cross-origin requests in the backend because it does not accept requests through an external port by default. In order to enable cross-origin requests, annotations have to be provided at each request controller in the backend. The process of adding all the required annotations is acceptable, but far from ideal. A solution where cross-origin requests can be enabled when running the backend in development mode would solve this issue, but no solution has been found.

Besides having to enable corss-origin requests on the backend, the frontend has to change the address to make requests to. This address has to be changed to `localhost` since the backend is active locally. Changing the address based on whether the frontend is served in a production environment should have been automated, but this has not been done yet.

Code Review

Before merging a new feature into the development branch on Gitlab, the code has to be reviewed by at least one other member. Reviewing the code should be done both statically and actively.

Static code review consists of looking through the new or changed code, checking for styling problems and parts of code that should be implemented differently. Gitlab provides a useful view for this type of code review.

Active code review consists of running the code and trying to find bugs or flaws in the design by using the new feature.

Running the code locally can cause issues like test data missing from the database. Besides, setting up the code locally could take a significant amount of time.

These issues caused active code review to be postponed to a moment the feature was deployed. By testing the actual functionality of the code only on the live environment, broken features were occasionally deployed. Preventing this unwanted deployment could be fixed by proving test data on merge requests when desired, which has not been done during development.

7.5. Weekly Meetings

In order to keep the client and the coach involved in the project, a meeting with the client and a meeting with the coach were planned on a weekly basis. During the initial weeks of the project, little to none preparation for these meetings was done. A short discussion was held before the meeting, but points to be discussed were not written down.

After a couple of weeks, the client suggested that topics to be discussed should be sent the evening before so he could prepare for the meeting. This resulted in an agenda being created each week, containing the new features to be shown, a list of topics to discuss, a set of questions for the client and a planning for the next week. Creating this agenda resulted in the meetings with the client being structured better and all points or questions to be covered during the meeting. Because of the improvements for the meetings with the client, an agenda was also created for each meeting with the coach.

Another improvement for the meeting with the client was suggested by the coach. By showing more of the work being done than just the resulting webpage, the client could get a better overview of what is being achieved during the project. This suggestion was quite useful as demos of the scheduler and attributes like an updated database schema gave the client a better understanding in the development process and helped planning new features to become more accurate.

7.6. Object-Relational Mapping

One recurring issue faced in the backend was the mapping between database and Java objects. This was especially prevalent when retrieving information needed for scheduling from the database. At the beginning of the project, the decision was made not to use a tool such as Hibernate, which is a tool that manages the mapping from database to Java objects. Instead, we chose to perform this mapping ourselves. We do this by performing SQL queries using JDBC, and having spring convert the result value of these queries to objects. By using this manual mapping , we have more control over the

database, especially given the live deployments we would face; we felt safer handling data migrations without Hibernate. Over the course of the project we found that most of our queries only convert tables to objects that represent most of the table, which suggests to us that Hibernate might have been suitable. It would also simplify the codebase, especially for the scheduling process. It would be worthwhile to research how Hibernate could fit into the system as it currently stands, and investigate whether replacing the current mapping system with Hibernate could be an overall improvement to the codebase.

8

Conclusion

TAM improves the teaching assistant recruitment and scheduling procedure, by improving the efficiency of the workflow and automating the scheduling process. Creating and updating forms to gather student interest is no longer needed, as these are now integrated in TAM. Once the courses are registered in TAM, the preferences can be submitted by students. Course managers put new courses and labs into TAM directly instead of informing the scheduler about their needs. This reduces the burden of work on the scheduler. Future work could mostly automate the addition of courses and labs. The manual processing of data is replaced by the TAM system. Users signing in via Single Sign-On ensures that user details are correct and can be referenced back to an individual. Furthermore, creating a schedule no longer requires manually assigning students to labs and checking constraints, as this is now done by the automated scheduling system. In addition, the TAM system is completely in-house hosted and GDPR compliant.

While TAM succeeded in solving the core problem, a lot of extensions are possible. The TAM codebase serves as a solid foundation designed to be easy to work with and extend, with extensive documentation and tests in place. We provide a list of future improvements, both to improve the workflow further and add new functionality, which can be found in Section 7.1.

A

Functional Requirements

A.1. Must Have

- All roles
  - Log in using Single Sign-On and determine access rights
- Course Manager
  - Create a course by providing a name, course code, quarter, year, start date and end date
  - For a created course, create a lab by providing timeslots and the desired number of TAs per timeslot
  - Show an overview of the course, including created labs and TAs assigned to the course
  - Show an overview of an individual lab, including assigned TAs
  - Show an overview of interested TAs, with information such as name, student number, other courses the TA is interested in, and past experience as a TA
  - In the overview of interested TAs, select preferences for TAs that the course manager would prefer having in their course
- Teaching Assistant
  - Select courses you want to assist.
  - If it is needed to verify the English test, upload verification (e.g. pdf, png)
  - Indicate that you have passed the TA training
  - Provide preferences for courses, and information such as a maximum amount of lab hours per week (optional), a maximum amount of courses to assist at once (optional), student number, and past experience.
  - For the entire quarter, availability in a week
- Verifier
  - Show an overview of TAs
  - For a TA, confirm they passed the English Test
  - For a TA, confirm they passed the TA Training
  - For a TA, indicate (based on grade) how suitable they are for each course they are interested in
  - For a TA, indicate their payscale
  - For a TA, have the ability to contact them for further screening before accepting
- Scheduler

  - Show an overview of all courses in a specified quarter and year
  - In a quarter, see an overview of courses and show statistics such as the number of TA's interested in courses, number of labs for each course
  - Show a list of interested TAs
  - For a quarter, generate a schedule using a provided algorithm
  - View a generated schedule
    - See TAs per course
    - See TAs per lab in course
    - Show imperfections in a generated schedule (e.g. not enough TA's for a certain lab)
  - Manually edit a generated schedule.
    - Edit which TAs are assigned to which course
    - Edit which TAs are assigned to which lab in a course
    - Highlight conflicts (e.g. TA is not available during the manually selected timeslot)
  - Reject a generated schedule
  - Save the last schedule
  - Accept a schedule
  - Export a schedule to pdf or csv
- Admin
  - View all users, assign/unassign roles
  - Can access all perspectives

A.2. Should Have

- Course Manager
  - Notification once the schedule has been accepted by the scheduler
  - See an overview of a course
  - In a course overview, see a list of TAs
  - In a course overview, see a lab overview
  - See information for a specific lab, such as assigned TAs, date and start and end time
  - Add additional managers to a course
  - For a lab, add/remove (adjust) scheduled TA (indicate TA availability)
  - Get notified of TA absence
  - Leave feedback on TAs
- Teaching Assistant
  - Indicate availability for each individual week
  - Notification when scheduled for a course
  - See an overview of courses you are TA for
  - For your course: see course page
  - For your course: an overview of labs, showing me for which labs I am a TA
  - For your lab: see information (which TAs, time etc.)
  - For your lab: inform course manager that I can't make it
- Verifier
  - Notification settings

- Scheduler
  - When accepting a schedule, notify involved TAs and course managers
- Admin
  - Export an individuals personal data to (to e.g. csv) to comply with the GDPR right to access

A.3. Could Have

- Course Manager
  - Import labs from MyTimetable
  - For a course, add or remove labs after the schedule has been made
  - Export a schedule to pdf or csv
  - Notify all TAs of a course
  - Notify all TAs of a specific lab
  - Export the schedule to the Queue system prior to a lab, by providing an endpoint that the Queue system can use in order to obtain the lab data
  - See an overview of total hours each TA is assigned to lab sessions
  - Give certain TAs course manager functionality (e.g. editing/adding labs)
  - Request extra TAs on Mattermost for a specific lab session (when other TAs can't make it)
- Teaching Assistant
  - Export schedule to a personal calendar stream (getting updates in case the schedule changes)
- Verifier
  -
- Scheduler
  - Customize the parameters of the used scheduling algorithm
- Admin
  - Have the ability to impersonate a specified user and view their perspective(s)

A.4. Would Have

- Course Manager
  - Register students automatically at FlexDelft after confirming a schedule
- Teaching Assistant
  -
- Verifier
  -
- Scheduler
  - Select a scheduling algorithm out of available alternatives
- Admin
  -

B

Model

B.1. Definitions

- Set of students S. An element of S will be indexed using the character i (e.g. S_i).
- Set of courses C. An element of C will be indexed using the character j (e.g. C_j).
- Set of timeslots T. An element of T will be indexed using the character k (e.g. T_k).
- Set of labs L. An element of L will be indexed using the character m (e.g. L_m).

B.2. Input Data

- $SP_{i,j}$: Binary $S \times C$ matrix that indicates whether student S_i wants to TA for course C_j.
- $TP_{i,j}$: Integer ($\{1, 2, 3\}$) $S \times C$ matrix that indicates course C_j's preference for student S_i.
- $CAS_{i,j}$: Binary $S \times C$ matrix that indicates whether student S_i is allowed to TA for course C_j.
- $CRS_{i,j}$: Binary $S \times C$ matrix that indicates whether student S_i must TA for course C_j.
- ST_i, k: Binary $S \times T$ matrix that indicates whether student S_i is available during timeslot T_k.
- $CL_{j,m}$: Integer $C \times L$ matrix that indicates how many TAs of course C_j are needed during lab L_m.
- $LT_{m,k}$: Binary $T \times L$ matrix that indicates that lab L_m takes place in timeslot T_k.

B.3. Decision Variables

- $TC_{i,j} = 1$ iff S_i is selected to TA for C_j.
- $TL_{i,m} = 1$ iff S_i is selected to TA during L_m.

B.4. Hard Constraints

1. A student cannot be scheduled for multiple labs during one timeslot.

$$\forall S_i \in S : \forall T_k \in T : sum(LT_{k,m} \Rightarrow TL_{i,m}) \leq 1$$

2. A student can only be assigned as a TA to a course if they want to TA for that course.

$$TC_{i,j} \Rightarrow SP_{i,j}$$

3. A student can only be assigned as a TA to a course if the course allows the student to be a TA.

$$TC_{i,j} \Rightarrow CAS_{i,j}$$

4. A student must be assigned as a TA to a course if the course has indicated so.

$$TC_{i,j} \Rightarrow CRS_{i,j}$$

5. A student can only be assigned as a TA at a timeslot if they are available during that timeslot.

$$TL_{i,k} \Rightarrow (LT_{k,m} \Rightarrow ST_{i,k})$$

6. A student can only be assigned as a TA to a lab if they are assigned to the course that the lab belongs to.

$$TL_{i,m} \Rightarrow \exists TC_{i,j} : CL_{j,m} > 0$$

B.5. Soft Constraints

1. It is better for a course to have fewer TAs.

$$\forall C_j \in C : \text{ Minimise } \sum_{p=0}^{|S|} TC_{i,j}$$

2. It is better for a student to be assigned fewer courses.

$$\forall S_i \in S : \text{ Minimise } \sum_{p=0}^{|C|} TC_{i,j}$$

3. It is better for a course to have TAs which it prefers to have.

$$\forall C_j \in C : \text{ Maximize } \sum_{i \in I} TP_{i,j} \text{ where } I = \{i \mid TC_{i,j}\}$$

4. It is better to have the same amount of TAs assigned to a lab be the same as the amount required for the lab.

$$\forall L_m \in L : \forall C_k \in C : CL_{k,m} > 0 \Rightarrow \sum_{p=0}^{|S|} TL_{i,m} = CL_{k,m}$$

C

API Specification

This appendix contains the full list of endpoints available in the TAM API. This is part of the full API description included in the project.

Teaching Assistant Management Platform 1.0.0

[Base URL: ta.ewi.tudelft.nl/api]

This page documents the REST API for the Teaching Assistant Management (TAM) platform.

Contact the developer

Schemes

HTTPS

me Endpoint to request information about the logged in user

GET	/me	Get user information for the logged in user

user User endpoint

GET	/user/{netid}	Get user information for the specified user
DELETE	/user/{netid}	Delete user by ID
POST	/user/{netid}/gdpr	Update the GDPR compliance from the user.
POST	/user/{netid}/tshirt	Update user tshirt properties by ID
GET	/user/{netid}/prerequisites	Get the prerequisites of a specified user
POST	/user/{netid}/prerequisites	Update the prerequisites of a specified user
GET	/user/{netid}/preferences	Find preferences for given courses
POST	/user/{netid}/preferences	Update or create preferences

DELETE	`/user/{netid}/preferences`	Delete preferences for the specified courses and user
POST	`/user/{netid}/vrank`	Update list of verifier ranks for the specified user
GET	`/user/{netid}/note`	Get notes for the specified user
POST	`/user/{netid}/note`	Update list of notes by the specified user
DELETE	`/user/{netid}/note`	Delete list of notes by the specified user
GET	`/user/{netid}/availability`	Get the availability of the user
POST	`/user/{netid}/availability`	Update the availability of the user
GET	`/user/{netid}/limits`	Get the limits of the user at the time periods which allow users to post availability
POST	`/user/{netid}/limits`	Update the limits of the user at the specified year and quarter
GET	`/user/{netid}/manages`	Get the courses which the user manages.
POST	`/user/{netid}/upload/englishcert`	[FUTURE] Upload the English test certification

course Courses endpoint

POST	`/course`	Add course to the database
GET	`/course/active`	Find courses which are currently open for preferences
GET	`/course/verifiable`	Find courses which allow verifier ranks to be updated
GET	`/course/{courseId}`	Get course information of the course with the given id
POST	`/course/{courseId}`	Update course information of the course with the given id
DELETE	`/course/{courseId}`	Delete the course with the given id

GET	`/course/{courseId}/assigned`	Get assigned students to the given course.
GET	`/course/{courseId}/labs`	Get the labs of the course with the given id
POST	`/course/{courseId}/labs`	Create a new lab for the given course
GET	`/course/{courseId}/labs/{labId}`	Get the lab for the given course with the given id
POST	`/course/{courseId}/labs/{labId}`	Update the lab for the given course
DELETE	`/course/{courseId}/labs/{labId}`	Delete the lab of the given course
GET	`/course/{courseId}/labs/{labId}/assigned`	Get assigned students to the given lab for the given course.

timeslot Timeslots endpoint

GET	`/timeslot/available`	Get timeslots for which availability can be given

student Student endpoint

GET	`/student`	Get all students, together with their prerequisites and preferences

schedule Schedule endpoint

POST	`/schedule`	Create a new schedule for the given year and quarter
POST	`/schedule/{scheduleId}/print`	Print the assignments of a schedule, including statistics

D

SIG Code Quality Evaluation

D.1. Initial Feedback

Beste,
Hierbij ontvang je onze evaluatie van de door jou opgestuurde code. De evaluatie bevat een aantal aanbevelingen die meegenomen kunnen worden in de laatste fase van het project.
Deze evaluatie heeft als doel om studenten bewuster te maken van de onderhoudbaarheid van hun code en dient niet gebruikt te worden voor andere doeleinden.
Mochten er nog vragen of opmerkingen zijn dan hoor ik dat graag.
Met vriendelijke groet,
Dennis Bijlsma

Feedback

De code van het systeem scoort 3.9 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Interfacing vanwege de lagere deelscore als mogelijk verbeterpunt.
Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden.
In jullie project is de constructor van CourseEdition een voorbeeld van een behoorlijk aantal parameters, waarbij het ook opvalt dat de meeste van die parameters strings zijn. Dat lijkt niet altijd logisch, zo is "teacher" nu een string, terwijl je daar een object zou verwachten. Je vergroot zo de onderhoudbaarheid, want je kunt later dan vrij makkelijk nieuwe velden aan het Teacher-object toevoegen.
De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.
Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

D.2. Final Feedback

Beste,
Hierbij ontvang je onze evaluatie van de door jou opgestuurde code. De evaluatie bevat een aantal aanbevelingen die meegenomen kunnen worden in de laatste fase van het project.
Deze evaluatie heeft als doel om studenten bewuster te maken van de onderhoudbaarheid van hun code en dient niet gebruikt te worden voor andere doeleinden.
Mochten er nog vragen of opmerkingen zijn dan hoor ik dat graag.
Met vriendelijke groet, Dennis Bijlsma

Hermeting

In de tweede upload zien we dat het project een stuk groter is geworden. De score voor onderhoudbaarheid is in vergelijking met de eerste upload gestegen.
Bij Unit Interfacing, dat in de feedback op de eerste upload nog als verbeterpunt werd genoemd, zien we een duidelijke stijging. Het is goed om te zien dat jullie naast het refactoren van de genoemde voorbeelden ook andere gevallen hebben verbeterd. Daarnaast zien we dat de nieuwe code op dit punt een stuk beter scoort.
Naast de toename in de hoeveelheid productiecode is het goed om te zien dat jullie ook nieuwe testcode hebben toegevoegd. De hoeveelheid tests ziet er dan ook nog steeds goed uit.
Uit deze observaties kunnen we concluderen dat de aanbevelingen uit de feedback op de eerste upload zijn meegenomen tijdens het ontwikkeltraject.

E

Table of Technologies

Table E.1: Table of Technologies

Technology	Used in the final version	**Link**
Angular	no	https://angularjs.org/
Bootstrap	yes	https://getbootstrap.com/
Bootstrap-Vue	yes	https://bootstrap-vue.js.org/
Checkstyle	yes	http://checkstyle.sourceforge.net/
Cobertura	no	http://cobertura.github.io/cobertura/
Django	no	https://www.djangoproject.com/
Elm	no	http://elm-lang.org/
Eslint	yes	https://eslint.org/
Findbugs	no	http://findbugs.sourceforge.net/
Flow	no	https://flow.org/en/
GitLab	yes	https://about.gitlab.com/
GitLab CI	yes	https://about.gitlab.com/features/gitlab-ci-cd/
Gradle	yes	https://gradle.org/
IntelliJ IDEA	yes	https://www.jetbrains.com/idea/
Jacoco	yes	https://www.eclemma.org/jacoco/
JDBC	yes	http://www.oracle.com/technetwork/java/javase/jdbc/index.html
Jest	yes	https://facebook.github.io/jest/
JUnit	yes	https://junit.org/
Mockito	yes	http://site.mockito.org/
MySQL	yes	https://www.mysql.com/
NPM	yes	https://www.npmjs.com/
PMD	yes	https://pmd.github.io/
PostgreSQL	no	https://www.postgresql.org/
React	no	https://reactjs.org/
SonarJS	no	https://www.sonarsource.com/products/codeanalyzers/sonarjs.html
SonarQube	no	https://www.sonarqube.org/
Spotbugs	yes	https://spotbugs.github.io/
Spring	yes	https://spring.io/
SQLite	no	https://www.sqlite.org/
Swagger IO	yes	https://swagger.io/
Thymeleaf	no	https://www.thymeleaf.org/
Travis-CI	no	https://travis-ci.org/
Vertabelo	yes	https://www.vertabelo.com/
VSCode	yes	https://code.visualstudio.com/
Vue	yes	https://vuejs.org/
Webpack	yes	https://webpack.js.org/

Screenshot of TAM

This appendix contains a screenshot of the TAM platform

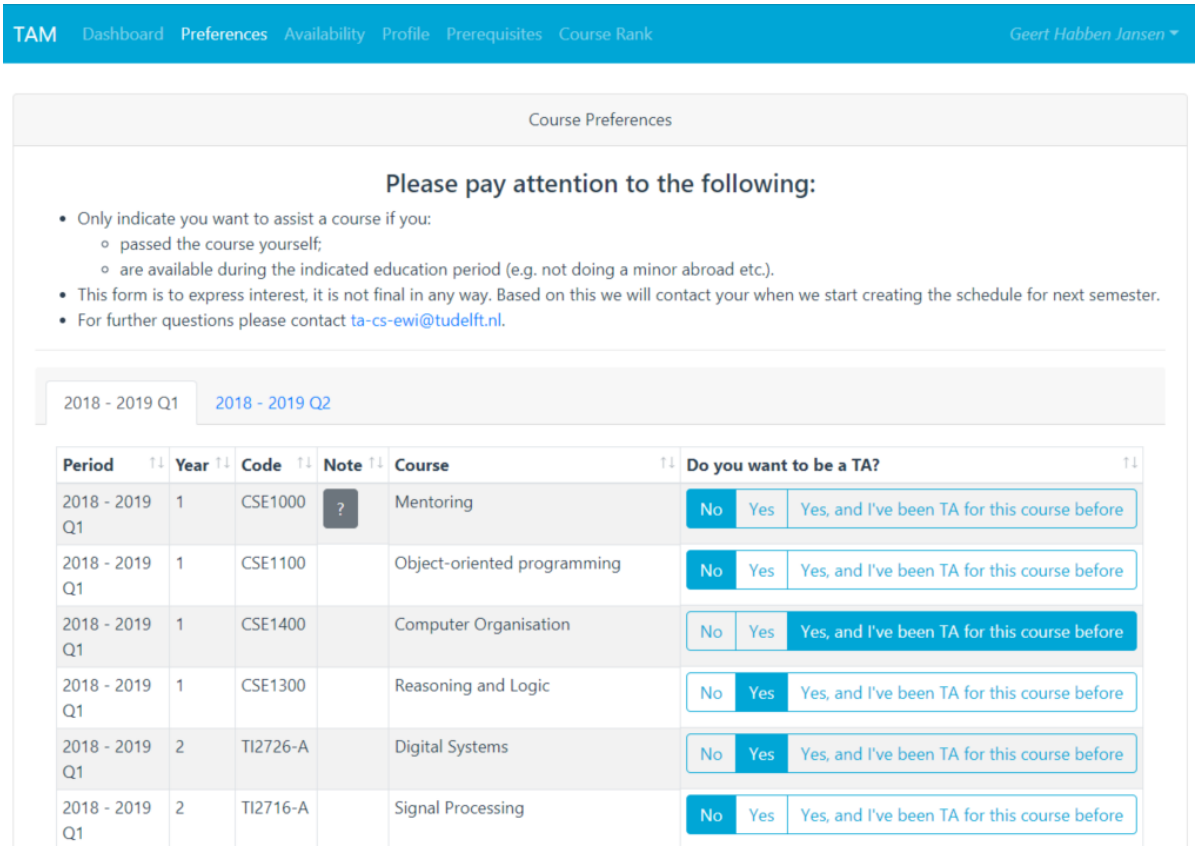


Figure F.1: A screenshot of the course preference page of the live application.

G

Original Project Description

This appendix contains the original project description

With the large influx of students, the number of teaching assistants required in the bachelor has risen over the last few years. In Q1 of 2017-2018 alone over 75 teaching assistants have been employed for the bachelor Computer Science & Engineering alone.

In an effort to automate the process that starts with gathering interest from potential TA's to actually hiring the TA's we are looking for a "TA database" that will not only keep track of all relevant information about TA's, but also help us in automatically creating schedules, informing TA's etc.

One of our developers has already created a prototype application for this is in Python, but we are now hoping to bring this prototype to a production ready environment.

H

Infosheet

This appendix contains the A4 infosheet of the project.

Teaching Assistant Management Platform

Description

The majority of the courses in the Computer Science Bachelor use *lab sessions* to provide an opportunity for students to ask questions. Teaching assistants are needed to assist during these labs. With the number of students in the Bachelor quickly growing, the process of manually recruiting students to become a TA and assigning the TAs to lab sessions is becoming infeasible.
The goal of this project is to ease the process of gathering and scheduling TAs by automating the workflow and increasing efficiency. TAM envelops the entire process. Lecturers can register their courses on TAM, students are able to indicate their interest and representatives from Education and Student Affairs can validate the applications of the students. Using this input, TAM is able to automatically create a schedule by assigning TAs to lab sessions.

TAM consists of three major components: a MySQL database, a backend written using Spring, containing the business logic and the scheduler, and a frontend website created with Vue. The frontend and the backend are connected using a well documented REST API. Both the frontend and backend are unit tested and well documented.

The research focused on the scheduling algorithm. Initially a model based on the minimum-cost max flow problem was created. Due to complications with the implementation, the team switched to a linear solver (Gurobi). By modelling the constrains and objectives of the scheduling, Gurobi is used to process the input into a schedule.
TAM was designed based on the existing workflow. It was first deployed in week 4, and subsequent versions deployed iteratively. Both the project repository and this report contain future improvements, such as integration with Mattermost for notifications and MyTimeTable for importing course data.
During the development, multiple problems have been encountered. The team underestimated the time required to learn the new technologies, as well as the time needed to maintain a system in production. Furthermore, configuring Single Sign-On required more time than expected.

Members of the project team

Name	Interests	Contributions
Max van Deursen	Algorithm Design, Reasoning and Logic	Backend setup & development, deployment, database maintenance
Geert Habben Jansen	Algorithm Design, cybersecurity	Frontend setup & development
Ruben Keulemans	User Interface design, application development	Frontend development, SSO, public relations
Max Pigmans	Algorithm Design, Robotics	Backend setup, project structure setup, scheduling, SSO, deployment

All team members contributed to API and database design, all reports and the final presentation, and meetings/communication with client and coach.

Key Information

- **Client organization:** Education Innovation Projects, Delft University of Technology
- **Presentation:** Monday July 2nd, 10:00 AM - 11:00 AM, lecture room Boole
- **Client:** MSc. Stefan Hugtenburg, Education Innovation Projects, Delft University of Technology
- **Coach:** Dr. Xavier Devroey, Software Engineering Research Group, Delft University of Technology
- **Contact Person:** Team member Max Pigmans, **max.pigmans@gmail.com**
- The final report for this project can be found at http://repository.tudelft.nl

Bibliography

[1] C. European Parliament, *Regulation (eu) 2016/679,(general data protection regulation),* https://eur-lex.europa.eu/eli/reg/2016/679/oj (2016).

[2] M. W. Carter and G. Laporte, *Recent developments in practical course timetabling,* in *PATAT*, Lecture Notes in Computer Science, Vol. 1408 (Springer, 1997) pp. 3–19.

[3] T. H. Hultberg and D. M. Cardoso, *The teacher assignment problem: A special case of the fixed charge transportation problem,* European Journal of Operational Research **101**, 463 (1997).

[4] B. Domenech and A. Lusa, *A MILP model for the teacher assignment problem considering teachers' preferences,* European Journal of Operational Research **249**, 1153 (2016).

[5] A. Gunawan, K. M. Ng, and K. Poh, *A hybridized lagrangian relaxation and simulated annealing method for the course timetabling problem,* Computers & OR **39**, 3074 (2012).

[6] D. Parkes and S. Seuken, *Economics and Computation* (unpublished, 2017).

[7] J. M. Kleinberg and É. Tardos, *Algorithm design* (Addison-Wesley, 2006).

[8] H. Mittelmann, *Benchmark of simplex lp solvers,* (2018).

[9] P. Chahal, *Bootstrap 3: An Ever Lasting Legacy for Responsive Designs,* http://templatetoaster.com/tutorials/bootstrap-3/ (2018).

[10] M. Petrosyan, *Angular 5 vs. React vs. Vue,* https://itnext.io/angular-5-vs-react-vs-vue-6b976a3f9172 (2018).

[11] J. Neuhaus, *Angular vs. React vs. Vue: A 2017 comparison,* https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176 (2017).

[12] S. Burge, *Bootstrap's Popularity Grew 1000% in 3 Years,* https://www.ostraining.com/blog/coding/bootstrap-popularity/ (2016).

[13] P. Chahal, *Developer Survey Results,* https://insights.stackoverflow.com/survey/2018/#technology-databases (2018).