



Schedule changes

Graduation report

Author: Michela A. Karunakalage

Organization: Topicus B.V.

Education: Stenden University of Applied Sciences

Version: 1.0

Deventer, 23rd June 2019

General Info

STUDENT NUMBER 1475319
COURSE INFORMATION TECHNOLOGY
PLACE EMMEN, NETHERLANDS

Contact

Emond Papegaaij (Company supervisor)	Karoline Heidrich (Client)
emond.papegaaij@topicus.nl	karoline.hedrich@topicus.nl
N.A.	N.A.
Michela A. Karunakalage (Intern)	Niels Doorn (Stenden supervisor)
michela.karunakalage@student.stenden.com	niels.doorn@stenden.com
0650408920	0610402112

Version management

Version	Description
0.1	Layout and writing the Preface and assignment description
0.2	Writing the Research chapter
0.3	Writing the Methodology and approach chapter
0.4	Writing the Realisation chapter
0.5	Writing the Conclusion and recommendation
0.6	Writing the Evaluation and reflection
0.7	Writing the Introduction and summary
0.8	Adding the Glossary, bibliography, figure list and appendices
0.9	Revision of the concept report
1.0	Concept version for check
2.0	Final version of the report

Company supervisor's signature

Approval of the Graduation report.

Emond Papegaaij

A handwritten signature in blue ink, appearing to be 'Emond Papegaaij', is written over a horizontal black line. The signature is stylized and fluid.

Preface

This work placement report is written by me, a fourth year IT student at NHL Stenden University of Applied Sciences, located in Emmen. The report is created as a part of the assignment of my graduation and on behalf of my company, Topicus. The purpose of the report is to explain the assignment that I have been conducting for Somtoday Docent, from the 4th of February 2019 till the 14th of June 2019, to my fellow IT classmates of NHL Stenden and to anyone interested in the analysis and implementation of schedule importers, and the process of schedule changes.

This document contains confidential information and may not be published or distributed without the written permission of Topicus and myself.

I would like to heartily thank Topicus and especially my company supervisor Emond Papegaaij to make this a pleasant journey and a successful assignment. I learned how much driven, enthusiastic and professional people they are.

Very special thanks go to my NHL Stenden supervisor Niels Doorn for his time and effort spent on me during my graduation period. I asked him many times for guidance during this process, which helped me a lot to clearly see my ultimate results.

I hope that you will enjoy reading it as much as I loved writing it.

Michela A. Karunakalage

Deventer, 18th May 2019

Glossary

Acronym/Term	Meaning	Description
API	<i>Application Programming Interface</i>	A set of routines, protocols and functions for building application software (Techopedia, 2019)
ANGULAR		A framework for building single page applications with JavaScript (Angular, 2019)
APA	<i>American Psychological Association</i>	A writing style widely adopted to cite sources
BURNDOWN CHART		A chart showing the evolution of remaining effort against time (Scrum, 2019)
COBRA		Cobra is a collection of modules that provided a set of standard and reusable components to build projects at Topicus Education
CONFIDENCE LEVEL		The certainty that results are statistically relevant
CONFLUENCE		A collaboration tool from Atlassian used for documentation from the consultants of Topicus
ECLIPSE		An integrated development environment used for programming mostly in Java
EXTERNAL LIBRARY		A set of functions which can be used in other code
FRAMEWORK		A platform for developing applications, easier and safer to use (Techopedia, 2016)
GITHUB		A git repository hosting service provided with a web-based graphical interface (GitHub, 2019)
GRAPHQL		A query language for API's and a runtime for fulfilling those queries with existing data (GraphQL, 2019)
HIBERNATE		A Java framework that simplifies the development of a Java application to interact with the database (Hibernate, 2019)
HTTP	<i>Hypertext transfer Protocol</i>	A set of standards that permit users of the World Wide Web to exchange information found on web pages (ComputerHope, 2019)
HTML		A standard mark-up language for creating Web pages (W3schools, 2019)
JAVA EE	<i>Java Enterprise Edition</i>	A platform that offers an API and runtime environment for developing and running large-scale applications
JIRA		An agile board tool used in agile teams to plan as well as to manage projects

JS	<i>JavaScript</i>	Programming language for creation of dynamic websites (W3schools, 2019)
JSON	<i>JavaScript object notation</i>	Minimal and readable format to structure the data
OAUTH		An authentication protocol that allows approving an application's user without giving away the password (Sobers, 2012)
POSTGRESQL		An object-relation database management system
PRODUCT BACKLOG		An ordered list of everything that is known to be needed in the product (Scrum, 2019)
PROTOTYPE		A working example to test a concept or process
REST	<i>Representational State Transfer</i>	An architect style to develop network applications
SCRUM		A framework to support teams in complex product development (Scrum, 2019)
SDLC	<i>Software development lifecycle</i>	A systematic process used to design, develop and test high-quality softwares
SOMTODAY CORE		The parent of all Sometown modules
SOMTODAY DOCENT		A Software module within Sometown focuses on functionalities for teachers
SPRINT		A consistent iteration where User Stories are completed (Scrum, 2019)
STATISTICAL SIGNIFICANCE		The probability that a result is unlikely due to chance (Measuring U, 2018)
UI	<i>User interface</i>	The means in which a person controls a software application or hardware device (Techterms, 2018).
UNIX TIMESTAMP		A timestamp made up of ten-digit numbers to represent time globally (Pinger, 2012)
URL	<i>Uniform Resource Locator</i>	A standardised identification convention for addressing documents accessible over the Internet
USE CASE		Several actions that are (partly) implemented by a system and that lead to a meaningful result for a user
USER STORY		A software feature for developers to know what to develop (Scrum, 2019)
UUID	<i>Universal unique identifier</i>	A 128bit number that recognises unique data
WICKET		Component-based web application framework for Java language (Wicket.apache, 2019)
ZERMELO		A software solution for the school schedules

Dictionary

Dutch term	English term
Actie	Action
ActieRealisatie	Action-realisation
Afspraak	Appointment
Boodschap	Message
Dag	Day
Docent	Teacher
Entiteit	Entity
Inhoud	Content
Instelling	Configuration
Les	Lesson
Lesuur	Class hour
Medewerker	Employee
Nieuw	New
Notificatie	Notification
Onderwerp	Subject
Onderwijs	Education
Organisatie	Organisation
Rooster	Schedule
Uitgevallen	Dropped out
Verplaatst	Moved
VerwerkAfspraak	Processed appointment
Vestiging	Branch
RoosterWijziging	Schedule-change
Zoek	Search
VerzendDatum	Sending date
OudeBeginDatumTijd	Old beginning time
OudeEindDatumTijd	Old ending time
Jaar	Year

Summary

Somtoday is an application for high school institutions to manage their data about students, employees and information. Somtoday Docent, a recently developed module, contains functionalities for teachers. However, a functionality of viewing changes in the schedules of teachers is still missing.

The objective of the assignment was to create a prototype of a schedule change functionality within Somtoday Docent. The following question was central throughout the entire process: *“How can a schedule-change notification system be developed within the platform of Somtoday regarding the requirements of the end users?”*

First, the infrastructure of the Somtoday Docent application was analysed. Somtoday Docent is an Angular application that uses GraphQL to retrieve its data from the Somtoday REST API. It uses the Somtoday OAuth endpoint for authentication. The project is built up from components and retrieves its data through data-services.

After this, the Somtoday core project was analysed. Somtoday is a Java Wicket application that runs on a Wildfly application server and saves its data to a PostgreSQL database. It uses Hibernate to map Java objects to relational database tables and Topicus Onderwijs’ internally developed Cobra framework. This framework contains a generic project structure that separates REST API from web application logic.

Somtoday core contains a functionality that imports appointments from *Zermelo*, a school schedule management application. Dummy appointments were created and updated in an environment available for Topicus developers. These appointments can be imported into Somtoday via the Zermelo webservice. A job can be started that runs the import in the background without locking the UI.

With the information obtained about Somtoday, Somtoday Docent and Zermelo, it was possible to analyse the connection among them. GraphQL is the intermediate layer which calls REST endpoints based on the connections that are defined. A schema needs to be made to create query syntax that calls a certain endpoint. After this, a query can be created to retrieve specific data that can be offered in a data-service to components in Angular.

Also, a questionnaire was sent out to teachers to obtain requirements for the prototype. 25 out of 35 teachers responded with the answer that new lessons, time changes and location changes would be the most relevant schedule changes. Furthermore, the answers state that the notifications should be visible in multiple, separate messages that should be selected by category. They should be retrieved throughout the day, only in Somtoday Docent and there is no need for reviewing them later.

Scrum has been used throughout the realisation of the prototype. Four Sprints have been done, each consisting of two weeks and each ending with a Sprint review and a retrospective. A Product Backlog, Sprint backlogs and burndown charts have been used. The scrum tool Taiga was used for maintaining the scrum process and GitHub for version control of the source code.

An infrastructure design, class definition and wireframes were created for the implementation of the prototype. This defined a REST endpoint that would expose Somtoday appointments with the help of Hibernate. The class would use properties from the existing *Actie* class that contains properties for sending messages.

After the creation of the designs, the prototype of schedule change system was developed. First, dummy schedule changes were saved to the database to ensure the mechanics of saving schedule was functional. Within the import job, the newly created appointments and the update of existing appointments would be detected and schedule changes would be saved. After this, the */roosterwijzigingnotificatieacties* endpoint was made to expose the schedule changes. The Cobra framework was referred when creating this.

With schedule changes being exposed in the REST API, the Somtoday Docent was edited to display them in the schedules. The newly created REST endpoint was added in the connections list, a schema and a query were made and the mapping logic was created in the data-service. These were then translated through the component tree to display newly created and updated appointments in the schedule of teachers.

Table of Contents

1 General Introduction	1
2 Assignment description	2
3 Research	3
3.1 Research methodology.....	3
3.2 What is the current situation of Somtoday Docent?.....	4
3.3 What is the current situation of Somtoday core?	7
3.4 What is Zermelo and how does the Zermelo importer work?	15
3.5 How are Somtoday core and Somtoday Docent connected?	19
3.6 What requirements would end users like to have for the schedule-change notification system?..	21
4 Methodology and Approach.....	27
4.1 What is Scrum?	27
4.2 Taiga	31
4.3 Version control	32
5 Realisation	33
5.1 Design	33
5.2 Somtoday.....	38
5.3 Somtoday Docent	41
5.4 Test Strategy.....	43
6 Conclusion	44
6.1 Recommendations.....	46
7 Evaluation	47
7.1 Reflection.....	48
Bibliography	49
List of Tables	52
List of Figures	53

1 General Introduction

Somtoday is a student administration system for Dutch secondary education that provides various digital tools for teachers and students, with the main purpose of keeping an easy workflow between each other. During the last years it became harder to maintain this workflow, especially for teachers, because of the increasing size of the application. This resulted in a lack of user-friendliness, slow and unclear navigation, and the difficulty to only work with a desktop version of the application (Somtoday portaal, 2018). In order to meet these requirements, a new module has been developed called *Somtoday Docent*. Somtoday Docent is a web application created specifically for teachers where, in a simple and user-friendly interface, it is possible to support students. Teachers can have a clear overview of their own timetable, check the attendance and homework of students and prepare the lessons and exams needed. Although the current system is operational, a need exists to track changes in the schedule and develop a schedule-change notification system. With this, teachers are finally able to be notified of changes by showing whether a specific lesson is deleted, modified or remained untouched. Which will bring huge business value to Somtoday Docent by being a fully functional application.

This report gives an insight into the development done on the Somtoday Docent schedule application. The following research question has been used throughout the entire process: *“How can a schedule-change notification system be developed within the platform of Somtoday regarding the requirements of the end users?”*

This project was carried out for Topicus, a software development company based in Deventer. Topicus offers IT solutions such as software products, hosting and helpdesk for external companies and external clients in the sectors of Finance, Healthcare, Education and Government. According to L. Essink (personal communication, 15 February 2019), co-owner of the company, Topicus’ vision is to emphasise on growth and innovation by continuously searching for new markets and providing these clients with stable, efficient services and security. In order to achieve that, Topicus strives for internal knowledge, trust and mutual connection. The company currently consists of four divisions. A division in Topicus is a company itself that concentrates on one market and it has its own budget. These are: Finance, Education, Government and Healthcare. Each division contains one or more business lines. A business line is a set of multiple teams that work for the same goal within the division. (M. Dashorst, personal communication, 20 February 2019). The ‘Somtoday Docent schedule changes’ project is run under the ‘Somtoday’ business line within Topicus Education. The intern worked closely with the Somtoday Core team (where the schedule importers are developed) and the Somtoday Docent team (where the new Docent-application is developed). An overview of the Topicus hierarchy can be found in Appendix A.

The first chapter defines the assignment in detail. It explains the motive, problem statement and objectives. The next chapter describes the conducted research and the research results. After this, the methodology and approach are defined. This is followed by the realisation results. Finally, the report ends with a conclusion and evaluation of the development process. Lists of resources follow: bibliography, a list of tables and figures, and a list of appendices.

2 Assignment description

This chapter describes the assignment in general. The first section explains the motive for the assignment and the problem statement. In the second section, the assignment objective is described.

2.1 Motive

Scheduling lessons for high schools is a challenging and complex process. This results in an adaptive schedule for teachers and students, with cancellations of classes as well as changes in location and time. Getting these changes to the appropriate subjects is even more challenging. This is mainly due to the complexity of schedule importers and the fact that there is more than one import routine because of different planning software systems.

2.2 Problem statement

Topicus' student information system 'Somtoday' recently developed a new module 'Somtoday Docent', which focuses solely on teachers and their responsibilities. But the teachers' schedule is missing two important requirements: a determination of actual schedule changes and a schedule-change notification system. Currently the importers just delete all the existing data from a specific period and import all new data, making it impossible to determine whether a specific lesson was changed or remained untouched (in terms of date, time and location). Also, it needs to be considered that different changes in schedule might require different notifications (or none at all) and that teachers have different opinions about how and when these notifications are to be received.

2.3 Objectives

The goal of this graduation is to create a prototype application for the Somtoday Docent module where it is possible to detect and track changes in the schedules and consequently notify the teachers.

3 Research

A research has been carried out to investigate the requirements and unclarities of the assignment. The results of the research are described in this chapter. This is required information before outlining the details of the realisation.

The following main research question has been used throughout the entire process:

“How can a schedule-change notification system be developed within the platform of Somtoday regarding the requirements of the end users?”

The main research question is expanded in the following sub-questions:

1. What is the current situation of Somtoday Docent?
2. What is the current situation of Somtoday core?
3. What is Zermelo and how does the Zermelo importer work?
4. How are Somtoday core and Somtoday Docent connected?
5. What requirements would end users like to have for the schedule-change notification system?

These questions have been answered in the listed order above.

3.1 Research methodology

In this study, different research methods have been used to answer the above sub questions. A research method is defined for each sub-question and how the results are considered valid and reliable:

- *Sub question 1:* a desk research has been conducted, with the goal of creating an infrastructure diagram that displays the structure of Somtoday Docent and all its dependencies. The type of data collection used were observation, reports and interviews. The reliability is validated by showing the infrastructure diagram to at least two different Somtoday Docent developers and incorporating any feedback that they give. To improve the validity, two Somtoday Docent developers were selected that have been working for Somtoday Docent since the day it existed;
- *Sub question 2:* this research contained the same methodology as the previous sub question: creating an infrastructure diagram that was reviewed by two Somtoday core developers. The type of data collection used were also the same as the previous sub question: observation, reports and interviews. To improve the validity, two Somtoday core developers were chosen that had at least two years of working experience with Somtoday core;
- *Sub question 3:* a desk research has been done to answer this sub-question. This mainly consisted of experimental activities. The type of data collection used were reports and interviews. There was no observation since there was no Somtoday developer available that actively works with Zermelo. To improve the validity, official Zermelo documents were used.
- *Sub question 4:* a desk research has been done in which a diagram was created to display the connection comp between Somtoday Docent and core. The selected measuring instruments were observation and internal documents. The reliability of this research is validated by showing it to one Somtoday core and one Somtoday Docent developer and including any feedback that they

might have. To improve the validity, these developers must have been different developers than the reviewers of the previous sub questions and they must have had at least two years of working experience at Somtoday;

- *Sub question 5*: a quantitative research was chosen only for this sub question to get the opinion of various Somtoday users. The selected measuring instrument was a questionnaire because of the possibility to generalise the results. To validate, the questionnaire was reviewed by at least two Somtoday employees. These employees must have had at least two years of working experience at Somtoday. According to an internal document (T. Gerritsen, 2018), Somtoday Docent has a list of 35 schools that have been using the application the most since the launch. Each of these schools has one reference teacher that Somtoday is allowed to contact for interviews and/or surveys. Therefore, the sample size for the questionnaire has also followed the available document. To ensure the reliability, at least 20 questionnaires out of the 35 sent needed to be filled in based on a confidence level of 95%, a population of 3000 users and an error margin of 22%. (For more information see section 3.6).

The analysed reports and documents are retrieved from *Topicus Onderwijs B.V.* - the company's intranet. These are written by reliable writers that have working experience with their respective departments.

3.2 What is the current situation of Somtoday Docent?

Somtoday Docent is a software module within the Somtoday software package focused on functionalities for teachers. They are able to view their schedule, assign homework and register absence. Somtoday Docent aims to support teachers in their daily work as much as possible, so that they have to spend less time on administration and more on guiding students in their intellectual and personal growth (Somtoday Docent portaal, 2018).

Before the implementation of a schedule change system, a research is required about the product. Section 3.2.1. describes the infrastructure of Somtoday Docent. It contains information about the connections and the technologies that were used to develop it. In section 3.2.2, the structure of the source code is defined.

3.2.1 Somtoday Docent infrastructure

Somtoday Docent is a web application built with Angular 7 and uses GraphQL 14 as an intermediate technology for retrieving data from the Somtoday REST endpoints. It uses the OAuth2 endpoint from Somtoday to authenticate. Somtoday users are able to log in and can be redirected to the Somtoday Docent application. An internal document of Topicus Onderwijs shows the infrastructure diagram in figure 1:

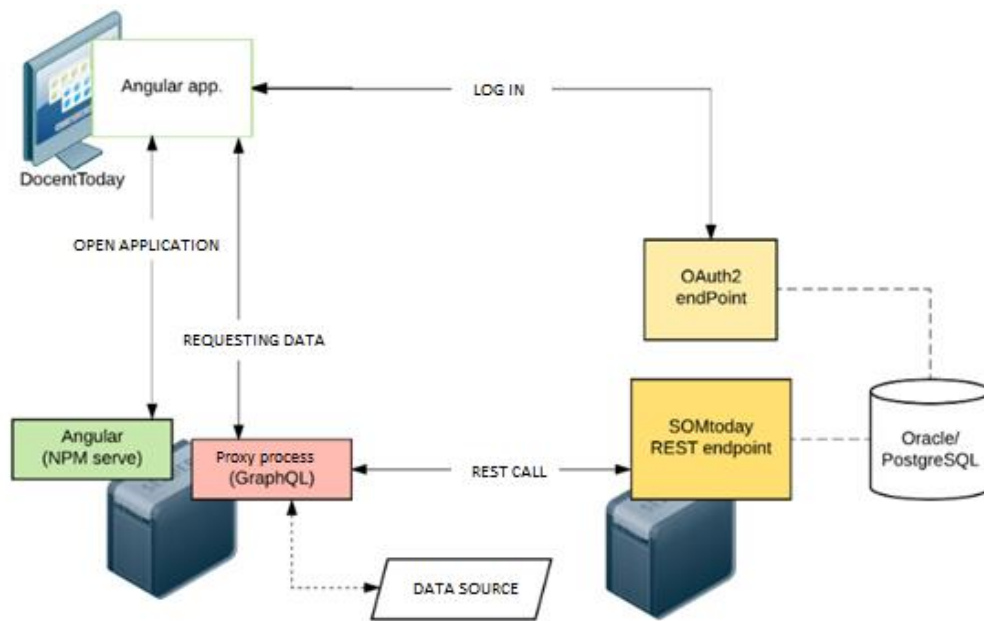


Figure 1: Somtoday Docent infrastructure (Timmerman, 2018)

As shown in the diagram above, Somtoday Docent depends on two connections: the server where the Somtoday Docent Angular application is hosted together with the GraphQL server and the server where the Somtoday core project is running for retrieving data and authenticating. This part of the product is explained in section 3.3.

Before diving into the source code of the Somtoday Docent project, some knowledge about the frameworks it uses is required. This is focused on the two main technologies used: Angular and GraphQL. The focus during this research is what these frameworks can do, what the benefits are and why they were chosen for the development of Somtoday Docent.

Angular

Angular is a framework for building single page applications with JavaScript. Single page applications load the entire content of the application into a single page (usually index.html). This means that when links are clicked within the application, it does not redirect to a different page. It updates sections within the page itself (JsWiz, 2016).

Angular was developed with three key concepts in mind: To make it modular, testable and maintainable. An Angular app consists of pieces called *components*. Components consist of three parts: A *template* which contains HTML that will be rendered onto the page, a *class* which contains properties and methods that are called inside the template and *meta data* which tells Angular which template and class pair become a component. (Angular, 2019). TJ van Toll (2018) describes in a post on the Telerik Developer Network that Angular is an excellent framework for non-trivial applications that involve data. It works well in form-based apps, and it is also well suited for large and complex applications. According to these sources, it is clear that Somtoday Docent structure is a single page application built up from components. This increases the maintainability, since functionalities are developed individually from each other.

GraphQL

GraphQL is an API query language developed by Facebook to improve the performance of interactions between a server and a client, such as a web browser or mobile app. GraphQL clients are able to request specified data from an API instead of retrieving all the data, which the client would need to filter out. Many resources can be retrieved with a single request as well (GraphQL, 2018).

GraphQL offers several features that fit in well with the requirements of Somtoday Docent:

- *Performance*: a maximum of one network request is always made from a page in Somtoday Docent. In addition, only the required data is requested;
- *Limiting*: the knowledge of data sources within Somtoday Docent is limited to GraphQL. This enables a consistent design of the code for data actions;
- *Flexibility*: it is difficult to predict whether, and if so, which functionalities are missing to offer the desired features in Somtoday Docent. By using GraphQL it is possible to simulate data without the application being responsible for it.

With the knowledge of the technologies used in the Somtoday Docent project, it is now possible to explore the structure of the project.

3.2.2 Component structure of schedule

Somtoday Docent is able to show a schedule with appointments that are linked to the teacher and are within the selected week. To realise a schedule change system, changes would need to be made in here. Since it is already known that Somtoday Docent is an Angular application (see section 3.2.1.), a research has been conducted about which components and which files are relevant to the realisation. The result is a component structure diagram and is visually displayed the figure 2:

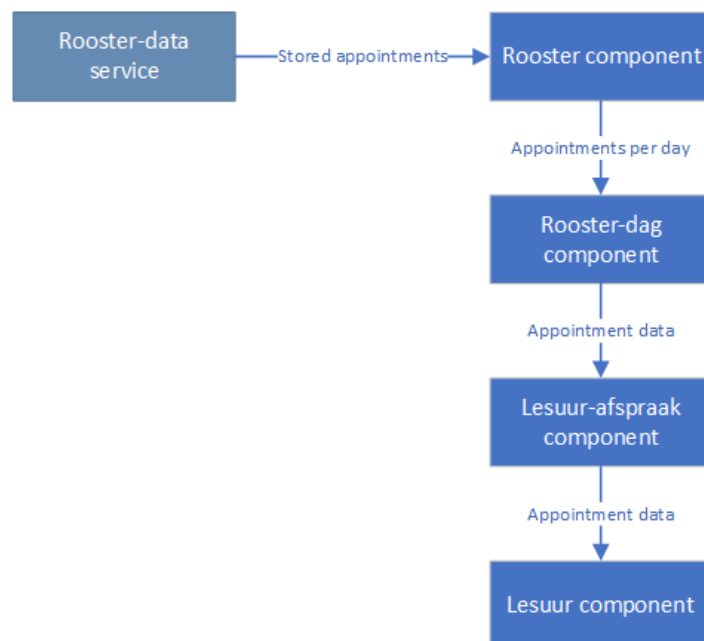


Figure 2: Somtoday Docent components diagram regarding schedules

The diagram shows that the following components are relevant to the schedule page: `rooster`, `rooster-dag`, `lesuur-afspraak` and `lesuur`. The `rooster` component functions as the parent of all the components, the definition of the schedule itself. It retrieves all the appointment items from the `rooster-data` service. In short: The `rooster-data-service` retrieves the appointments that are stored in the Somtoday database through the Somtoday REST API (for more information, see section 3.5.). The `rooster` component creates a list of days which are known as `rooster-dag` components. A `rooster-dag` component is given a list of appointments in the form of *models* by the `rooster` component. For each appointment in the list, the `rooster-dag` component creates a `lesuur-afspraak` component. This is the visual representation of an appointment. The location, time and icons are shown in here. The `lesuur` component is a general component that displays the text and emblem of a schedule item. Finally, the entire structure results in the schedule shown in figure 3:

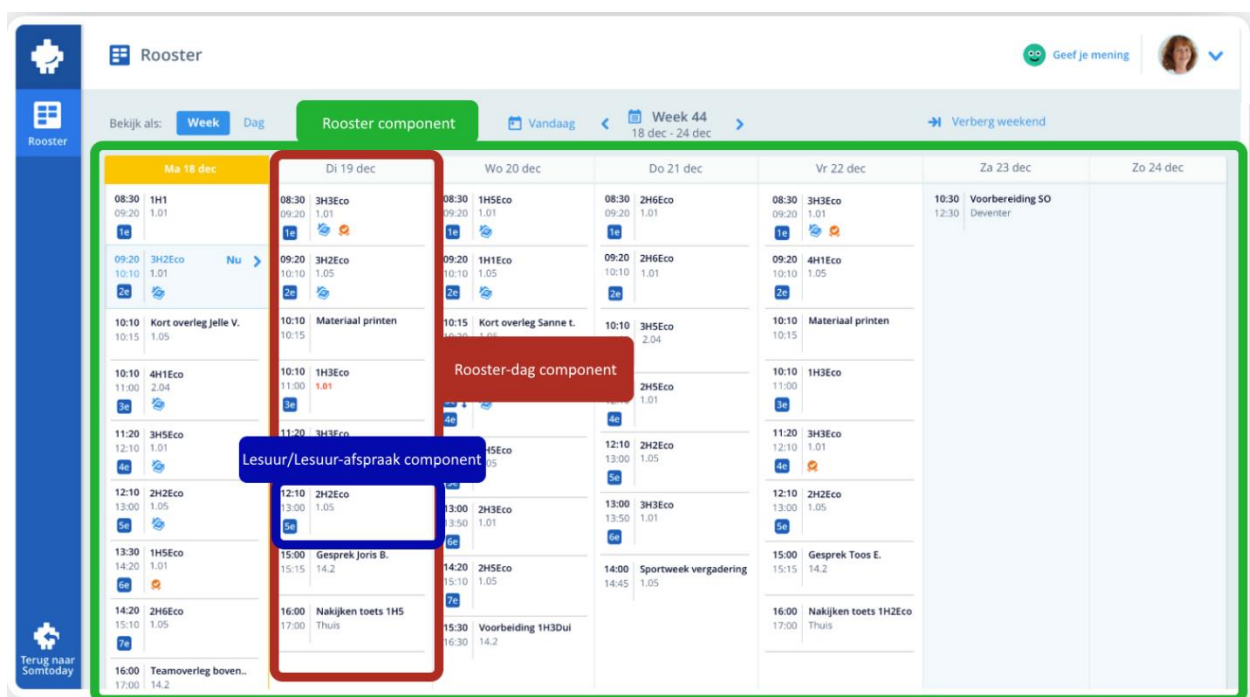


Figure 3: Component structure visually displayed in schedule

3.3 What is the current situation of Somtoday core?

With the Somtoday core application, users are able to administer entities like students, subjects, groups and staff. This is the base application where data is managed. The Readme file on the GitHub page (Idserda, 2016) states that the Somtoday core project uses the following frameworks:

- The *Cobra* framework
- The *Wicket* web framework
- *Hibernate* for object-relational mapping

To translate this in a summary: Somtoday core is a Java Wicket web application that follows the conventions and tools offered by Topicus' Cobra framework. It uses Hibernate to translate objects to relational database to save, update and delete entities. Because the source code would need to be changed for the schedule change notification system to function, research has been conducted about the

fundamental parts of Somtoday core. Section 3.3.1. shows the general infrastructure of Somtoday core, section 3.3.2. explains about the Cobra framework which was used to build Somtoday core. Section 3.3.3. states the most relevant folders for this project. Section 3.3.4. contains information about the database that Somtoday core uses and finally, the section 3.3.5. lists entities that are the most relevant to this project.

3.3.1 Somtoday core infrastructure

The most general aspect to research about is the infrastructure of Somtoday core. By doing so, new technical aspects would be revealed to research about. The total infrastructure that is relevant to this assignment is shown in figure 4:

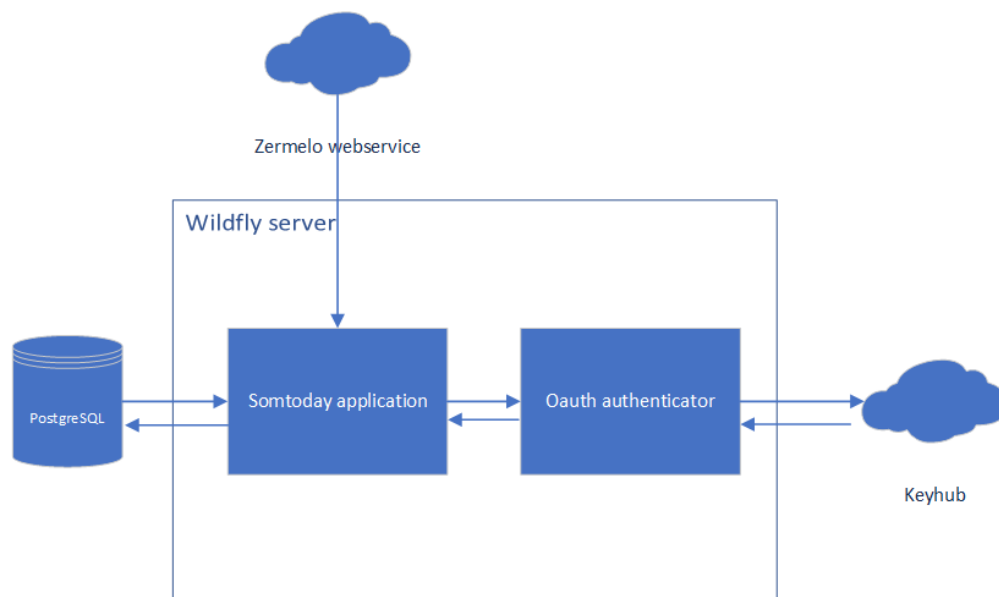


Figure 4: Somtoday core infrastructure

The Somtoday core project runs on a *Wildfly* server. Wildfly (also known as *JBoss*) is an open source application server developed with Java. An application server provides data that can be used in various applications. Wildfly contains various features, like a management console and unified configuration (Wildfly, 2019). In a blog about choosing application servers, Davis M. (2017) explains that Wildfly is used in applications where users need to access the full functionality of the Java Enterprise Edition suite. It provides a vast variety of features to run enterprise applications.

Two separate modules can be found in the Wildfly server: The Somtoday application and the OAuth authenticator. The Somtoday application is the application that contains the main functionalities and is connected to a PostgreSQL database (for more information, see section 3.3.4.). It can also retrieve data from a Zermelo webservice, which can be read more about in section 3.4. The OAuth authenticator handles all kinds of authentication and SSO (Single sign on) flows for many different systems. It also handles 2FA (Two-factor authentication) for teachers. For support-employees, it can federate login to KeyHub.

3.3.2 Cobra framework

Cobra is a set of modules that function as a basis for applications of Topicus Education. It's a collection of reusable components to work with Java EE technologies, such as JPA for accessing database entries in an object-oriented way and JAX-RS for building REST API's. The goal of the Cobra framework is to decrease development time within the Topicus Education department (Topicus Onderwijs B.V., 2019).

The Cobra framework contains various aspects, such as conventions and tools that are reusable for the schedule changes notification system. The following sections contain information from the book *Cobra In Action* (Topicus Onderwijs B.V., 2019) that are relevant to the development process of the schedule-change notification system.

3.3.2.1 Folder structure

Every Cobra application has a common folder structure. The folder structure is show in figure 5:



Figure 5: Cobra folder structure

The *backend* folder contains a project for the backend of the application and consists of the following parts: *entities* for the entity classes, *dao* for database services, *rest* for the REST API logic and *ear* for EAR files that contain published REST API logic. The *contract* folder is the REST API definition (the contract between the server and the clients). The *scripts* folder contains scripts for configuring Wildfly and the *web* folder is for the Wicket web application that will be packaged as a WAR file.

3.3.2.2 Data Access Objects

Data Access Objects (DAO's) are objects that the object relational mapping tool uses to translate data from data tables back and forth. DAO's are accessible via *services*. The purpose is to hide the complexity of creating complex database queries while data could also be accessed through object-oriented programming. An example of a DAO is show in figure 6:

```

@Stateless
@TransactionalAttribute(TransactionAttributeType.MANDATORY)
public class CheeseDAO extends AbstractBoundDAO<PCheese, CheeseZoekFilter>
{
}
  
```

Figure 6: DAO example

As shown in the figure above, a DAO class must extend from the `AbstractBoundDAO` class and must contain two annotations: The `@Stateless` and `@TransactionAttribute` annotations tell if a transaction must already be running throughout the saving session or not. Further the DAO requires two more parameters: The entity class (in this case `PCheese`) and the search filter class (In this case `CheeseZoekFilter`).

3.3.2.3 Rest resources

The Cobra framework offers entities through the REST API in the form of *rest resources*. With this, the name of the entity can be used combined with the HTTP method to execute a function (e.g. GET for retrieving, POST for saving and DELETE for deleting). The naming convention of the rest resource is dependent on the project. For new project: database entities receive a 'P' in front of their name and rest resources not. An example entity would be *Group* where *PGroup* is the name of the database entity and *Group* the name of the rest resource. For old projects: database entities have no prefix in their name and rest resources receive a 'R' in front of their name. In the context of the Group example: The database entity would be called *Group* and the rest resource *RGroup*.

Two types of resources exist: a *rootresource* which is a completely independent resource and a *subresource* which is dependent on a rootresource or another subresource. To create a rootresource, the following files need to be created:

- In the contract folder:
 - o `RGroup`: Contains all the field that can be accessed in the REST API regarding this resource;
 - o `RGroupResource`: Contains the URI path of the resource and all the available HTTP methods;
- In the web folder:
 - o `RGroupResourceClient`: Class to be injected in the Wicket page so it can be used in the Wicket application;
 - o `RGroupFilter`: The client-side search filter (for the Wicket application);
- In the dao folder:
 - o `GroupDAO`: The DAO of this resource so that data regarding this resource can be retrieved from the database;
 - o `GroupZoekFilter`: Defines the server-side search filters. These can be used in the URI to filter out certain resources;
- In the rest folder:
 - o `RGroupResourceService`: The REST service that implements the `RGroupResource` interface. Used to write custom logic.

With the creation of these files, Cobra can translate these files to a REST endpoint that executes logic and responds with data.

3.3.3 Project Structure

The Somtoday project is referred as *Iridium* and can be found in the private Github repository of the Topicus organisation. The project structure of iridium has been examined before the realisation, so that adjustments can be made more easily. The total project structure is show in figure 7:

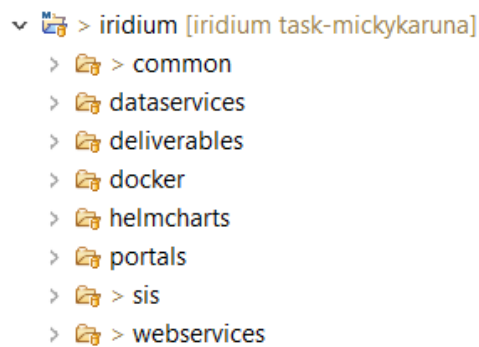


Figure 7: Iridium project structure

Note that the project follows the Cobra framework conventions regarding coding and structure. A complete overview of the Cobra framework conventions can be found in section 3.3.2. Each folder in the project has a specific purpose. The purpose of the most important folders is explained in the section below:

Common

This folder contains various classes that are commonly used throughout the entire project. This in the context of entities, DAO's, REST resources, services and assets. This folder mainly contains classes that represent something. *Entities* represent objects that can be retrieved from and saved to the data and can be found in the *entities* folder. *DAO* stands for Data Access Object and are objects used to separate high end business logic from low end operations. They can be found in the *dao* folder. *REST Resources* are representations of a REST endpoint and contain fields that are retrieved in the endpoint. These can be found in the *service-contract* folder. Services are injectable in other classes so they can be used everywhere. For example: with the *BoodschapService* a method can be executed that creates new messages and links them to specified teachers. Services can be found in the *services* folder. Assets are resources like images, fonts and stylesheets and can be found in the *resources* folder. A complete overview of all the folders found in the common folder can be seen in figure 8:

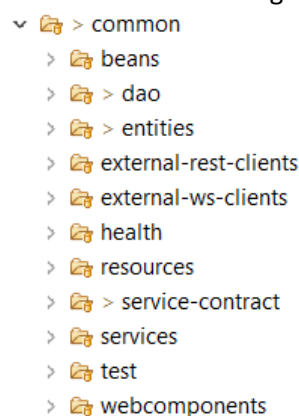


Figure 8: Common folder content

Deliverables

This folder contains all the deliverables that are used in Wildfly. Whenever code gets changed and published, a deliverable in this folder will be overridden. Separate modules like the authenticator and main application are published in here.

Sis

The source code of the complete core application lies in this folder. It contains all the pages and logic of the entire Somtoday core project and consumes classes from the *common* folder. The most important folder in here is the *core* folder, which contains all folders relevant to the Wicket application. An overview of this is shown in figure 9. The folders in here that are relevant to the development process are: *dao*, *jobs* and *web*. The *dao* folder contains all the `DataAccessHelper` classes that are used to retrieve data from the database in the form of an object (see section 3.3.4. for more information). In the *jobs* folder, logic of Somtoday jobs reside. Jobs are functionalities that are executed in the background of the Somtoday application and are not dependant on the user interface. The *web* folder contains the Wicket pages of the Somtoday application. In here, the layouts and business logic are editable.

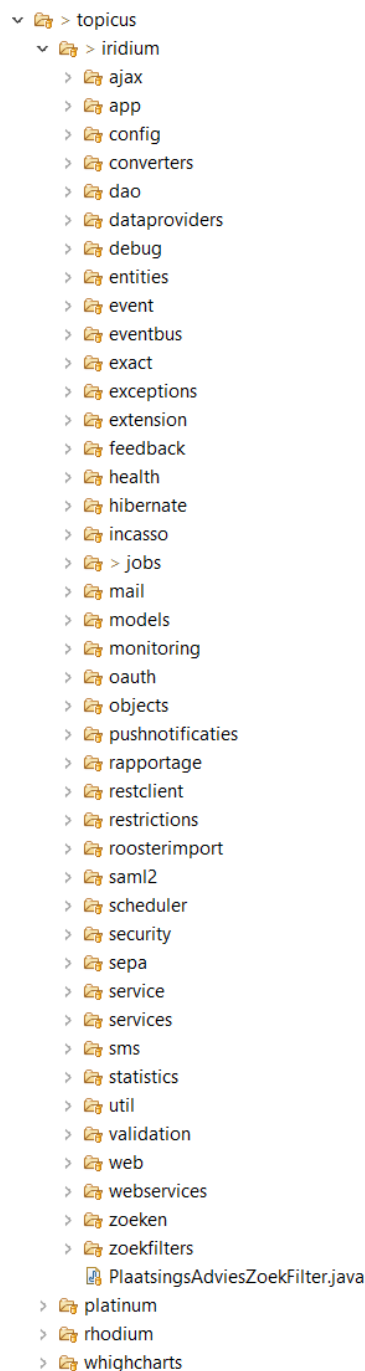


Figure 9: Core folder content

Webservices

The webservises folder contains all the logic of the Somtoday REST service. The authentication route always goes through here, since it contains the *authenticator-rest* folder which holds the OAuth logic. The other REST functionalities are stored in the *rest* folder. This folder contains the implementation of all the REST Resource classes within Somtoday (see section 3.3.2). An overview of the content of the webservises folder is displayed in figure 10:

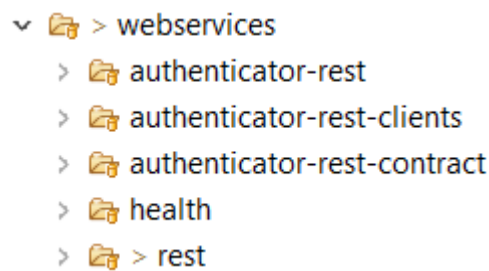


Figure 10: Webservices folder content

3.3.4 Database

According to an internal document (Kupers, 2019), Somtoday uses a PostgreSQL database to store its information. It used to store its data in an Oracle database system, but since Oracle adjusted the pricing of licenses, a migration is currently in session (Oracle, 2019). Therefore, developers in the Somtoday team are only allowed to run a PostgreSQL development database. Somtoday uses Hibernate to communicate from application to database level. Additionally, the Cobra framework contains various helpers using the Hibernate functions. This means that the Somtoday Wicket application uses the Cobra framework functionalities which translate back to Hibernate (Topicus Onderwijs B.V., 2019). Two examples of these are the specialised *DataAccessHelper* classes and the static *Entiteit* class. The *DataAccessHelper* classes contain methods which can retrieve data through Hibernate. An example of this is the *AfspraakDataAccessHelper* which contains a *getAfspraken* method that obtains all the appointments within a certain institution. The *Entities* class is used to modify data. It contains a *save*, *update* and *delete* to which an object can be passed that will be modified. For example: To save a new appointment, the code is used in figure 11:

```
Entities.save(afspraak);
```

Figure 11: Save appointment example

With the knowledge of objects being used to retrieve and manipulate data from the database, the most common entities were analysed.

3.3.5 Common Entities

Before designing the notification system, knowledge would need to be gathered about existing entities. This knowledge would be used when encountering entities during the development process. According to the information retrieved from E. Papegaaij (personal communication, 2019), the most valuable entities to analyse are: *Organisatie*, *Medewerker*, *IridiumAccount*, *Afspraak*, *AfspraakParticipant*, *Actie* and *ActieRealisatie*.

Organisatie

Represents an organisation. This entity is a parent of the entities *Instelling* which represent schools and *Vestiging* which represents branches of schools. Various information is stored, such as the name of the schools, the abbreviation and the UUID required to log in with a user on a specific school. This entity is linked to the *organisatie* table in the database.

Medewerker

Represents an employee in a specific institution. Examples of employees are teachers, support and administration. Unlike the *IridiumAccount* entity, this entity does not contain credentials (see *IridiumAccount*). It is used to contain employee information, such as the name, contact information and function title. This entity is linked to the *medewerker* table in the database.

IridiumAccount

Represents the credentials of a user. The username and hashed password are stored in here, connected to an *Organisatie* entity. The *Medewerker* entity is linked to this through the *account* column. This entity is linked to the *iridiumaccount* table in the database.

Afspraak

Represents an appointment in Somtoday. Appointments are general entities which could mean a lesson hour or an appointment with another teacher. Multiple people are linked to an appointment through the *afspraakparticipant* table (see *AfspraakParticipant*). Appointment contain various information, such as a title, description, start/endtime and a location. This entity is linked to the *afspraak* table in the database.

AfspraakParticipant

Represents a link between an appointment and a person. This person could be an employee or a student or a group. This entity is linked to the *afspraakparticipant* table in the database.

Actie

Represents an action to be taken. This comes in a variety of forms, such as a message or a notification. It contains a list of people that are connected to this action, a publication date and the type of action. Various classes inherit from this entity, such as *Boodschap*, which represents a message. This entity is linked to the *actie* table in the database.

ActieRealisatie

Represents the realisation status of an action. Thus, it contains a link to the *actie* table. It also has a boolean which tells the system whether an action is fulfilled and the time of fulfilment. This entity is linked to the *actierealisatie* table in the database.

3.4 What is Zermelo and how does the Zermelo importer work?

Zermelo is a software solution for the school schedules. Users are able to create, update and publish schedules. According to K. Heidrich and to an internal document of Somtoday Docent, Zermelo is the market leader of school schedule creation software. Because of this, the Somtoday team decided to create a *Zermelo importer*, which retrieves appointments from the Zermelo API and converts it to Somtoday appointments (Topicus Onderwijs B.V., 2018). This is possible by connecting Somtoday to the *Zermelo webservice*. To test the functionalities of Zermelo and to create dummy data, Topicus Education has a development environment running of the Zermelo portal. Topicus users are able to log in and use various debug features. This portal has also been used for the creation of dummy appointments for the schedule change system. The most important functionalities were the creation and mutation of appointments.

3.4.1 Creating appointments

Because the test Zermelo portal does not have a functionality to create appointments, this must be done by sending a POST request to the Zermelo REST API. The official Zermelo API documentation (n.d.) states that this is done through the `/api/v3/appointments` endpoint. An access token would need to be retrieved first, which can be obtained from the Zermelo test portal itself. Once taken, a GET parameter would need to be added, so the URI link would have the form: `<zermeloURI>/api/v3/appointments?access_token=<token>`. The body of the request contains the parameters that are relevant to the appointment. Figure 12 contains a screenshot of the parameters for a new appointment:

```

1 {
2   "start": "1560243600",
3   "end": "1560247200",
4   "subjects": ["bi"],
5   "locations": ["1"],
6   "teachers": ["vlen"],
7   "groupsInDepartments": [501],
8   "type": "lesson",
9   "valid": true,
10  "branch": "bmb"
11 }
```

Figure 12: Parameters for new appointment

Important aspects to notice about these parameters are:

- The start and end dates are *Unix timestamps*. These are the total amount of seconds since January 1st, 1970 at UTC (Dan's tools, n.d.);
- Every value in a square bracket is a list. This means that it is possible to create an appointment with multiple subjects, locations and teachers.

If the appointment is created successfully, The Zermelo API gives a response message with status 200. A response message as given by Zermelo can be seen in figure 13:

```

1 {
2   "response": {
3     "status": 200,
4     "message": "",
5     "details": "",
6     "eventId": 0,
7     "startRow": 0,
8     "endRow": 1,
9     "totalRows": 1,
10    "data": [
11      {
12        "id": 97906,
13        "appointmentInstance": 97831,
14        "start": 1560243600,
15        "end": 1560247200,
16        "startTimeSlot": null,
17        "endTimeSlot": null,
18        "branch": "bmb",
19        "branchOfSchool": null,
20        "type": "lesson",
21        "teachers": [
22          "vlen"
23        ],
24        "groups": [
25          "n3bwi"
26        ],
27        "groupsInDepartments": [
28          501
29        ],
30        "locations": [
31          "1"

```

Figure 13: Response from Zermelo after creating new appointment

This also contains two noticeable values: The *id* and the *appointmentInstance*. These are both ID's generated by Zermelo, but for two different purposes: The *appointmentInstance* is an ID for one certain appointment. Whenever this appointment gets updated, the *appointmentInstance* stays the same, but the *id* will change. The *id* in Zermelo identifies a specific version of that appointment instance in Zermelo. For more information about this, see section 3.4.2.

3.4.2 Updating appointments

Internal documents state that the Zermelo test portal contains a functionality to update appointments without the necessity for using the REST API. Unfortunately, this was not available and consequently the REST API still had to be used. According to the official Zermelo API documentation (n.d.), updates are also done through POST requests to the same URI as creating appointments, but with the *appointmentInstance* of a specific appointment in the parameter. An example of the preparation of such a POST request can be seen in figure 14.

```

1 {
2   "appointmentInstance": 97831,
3   "start": "1560247200",
4   "end": "1560250800",
5   "subjects": ["bi"],
6   "locations": ["1"],
7   "teachers": ["vlen"],
8   "groupsInDepartments": [501],
9   "type": "lesson",
10  "valid": true,
11  "branch": "bmb"
12 }

```

Figure 14: Parameters for updating existing appointment

As seen in the figure above, the appointmentInstance is given in the body of the POST request. With this, Zermelo is able to link the values that have changed to the appointment with the already existing appointmentInstance (e.g.: If the start time changed, it is changed in the appointment and if the locations changed, it is changed in the appointment as well). Like creating a new appointment, updating an appointment gives a similar response. An example response can be seen in figure 15:

```

1 {
2   "response": {
3     "status": 200,
4     "message": "",
5     "details": "",
6     "eventId": 0,
7     "startRow": 0,
8     "endRow": 1,
9     "totalRows": 1,
10    "data": [
11      {
12        "id": 97907,
13        "appointmentInstance": 97831,
14        "start": 1560247200,
15        "end": 1560250800,
16        "startTimeSlot": null,
17        "endTimeSlot": null,
18        "branch": "bmb",
19        "branchOfSchool": null,
20        "type": "lesson",
21        "teachers": [
22          "vlen"
23        ],
24        "groups": [
25          "n3bwi"
26        ],
27        "groupsInDepartments": [
28          501
29        ],
30        "locations": [
31          "1"
32        ],
33        "locationsOfBranch": [],
34        "optional": false,
35        "valid": true,

```

Figure 15: Response from Zermelo after updating an existing appointment

As the response of creating appointments, this response gives an id and an appointmentInstance as well. The only difference is that the id changed and the appointmentInstance not. This is because Zermelo has created a new appointment, but with a higher version of the same appointment instance. Zermelo does not update appointments itself directly.

3.4.3 Zermelo importer in Somtoday

As already mentioned in section 3.4. Somtoday contains an importer that converts appointments retrieved from Zermelo to Somtoday appointments. This is possible by retrieving appointments from the Zermelo webservice. This can only be done in the admin panel of Somtoday.

3.4.3.1 Importing through webservice

Every Zermelo has a JSON REST API available that can be accessed through with an access token. Various entities can be retrieved, such as announcements, appointments and parent-teacher nights. It is possible to link Somtoday with this Zermelo webservice to obtain appointments remotely. Figure 16 shows the Zermelo webservice page, currently available in Somtoday:

Beheer

Instelling Onderwijs Plaatsing Afwezigheid Resultaten Rapportage Begeleiding Leermiddelen Controles Import & Export Info

Rooster Zermelo webservice

Zermelo instellingen

Omgeving: MichelaTest

Vestiging: Bouwmeester bovenbouw

Rooster importeren vanaf: 12-06-2019

Rooster importeren t/m: 12-06-2019

Somtoday instellingen

Vestiging: Duinwetering

Basisrooster: Basisrooster 2018-2019

Afspraaktype voor Zermelo les: Les rooster

Participanatype: Groepen

Afspraaktype voor Zermelo toets: Niet importeren

Afspraaktype voor Zermelo activiteit: Niet importeren

Koppel docenten aan lesgroep: ☒

Omschrijving	Datum/tijd opgestart	Aanvullende info	Status	Progressie
Taken				
Begin	Einde	Gestart door	Samenvatting	

Geplande taken Geplande taak toevoegen Starten

Figure 16: Zermelo import screen – Webservice

The webservice import page contains parameters to be set before starting the import job. On the left top side of the page, settings regarding Zermelo can be set. An *environment* must be set selected first, this is a set up done in another page. A *Zermelo branch* can be selected and a *time range* can be specified. On the right top side, settings regarding Somtoday can be set. The *Somtoday branch* can be selected along with other specifications, such as the schedule *category*. Once all the parameters are set, the user can click on the ‘Start’ button to execute the import *job* (see section 3.3.3. for more information about jobs). In the background, appointments are retrieved from the Zermelo webservice and are converted to Somtoday appointments. Once completed, a log entry will be shown in the middle table that explains the results of the job.

3.4.3.2 Internal structure

With the knowledge of Zermelo import functionality, the research was moved to the next stage: observing the source code of the importer. This would be used as a reference throughout the realisation phase. The import functionality is presented in the form of a background job. This limited the search to the *jobs* folder in the project structure (see section 3.3.3.). This led to the `ZermeloRoosterImportJob` class that resides in the *roosters* folder. The `ZermeloRoosterImportJob` starts with the `doExecute` method that takes the parameters set in the form of an object. The logic executed is done by the `ZermeloRoosterImportJobService` that contains a method called `getAppointments` that retrieves the Zermelo appointments and a `verwerkAfspraken` method that processes these appointments. In the `verwerkAfspraken` method, the Zermelo appointments are converted to Somtoday appointments with the `zermeloAsAfspraakFunction` object. For each of these, the `verwerkAfspraak` method makes a certain decision: creating a new appointment if it does not exist yet, updating an existing appointment or deleting an appointment that does not exist anymore in Zermelo. This logic is being executed in the class that the `ZermeloRoosterImportJobService` extends from: the `RoosterImportJobService` class. A complete overview of this process can be seen in Appendix B.

3.5 How are Somtoday core and Somtoday Docent connected?

Now that both Somtoday and Somtoday Docent projects are clear, the opportunity exists to research about the communication between the two. As already stated in the Somtoday Docent infrastructure (see section 3.2.1.), data is retrieved from the Somtoday REST endpoints. Although this information is available, the communication flow and relevant files are still unclear. Section 3.5.1. shows the communication flow between the two projects, with the emphasis on how GraphQL handles the communication. Section 3.5.2. states what files in the Somtoday Docent project are relevant for retrieving data.

3.5.1 Communication flow

As already stated in section 3.2.1. GraphQL is responsible for the communication of Somtoday Docent to Somtoday, without the need for acquiring knowledge about the REST endpoints. Although this is true, it is relevant to know what GraphQL does in its communication flow, since this explains what the responsibility of each server is. Figure 17 shows a sequence diagram of a request to save data in Somtoday:

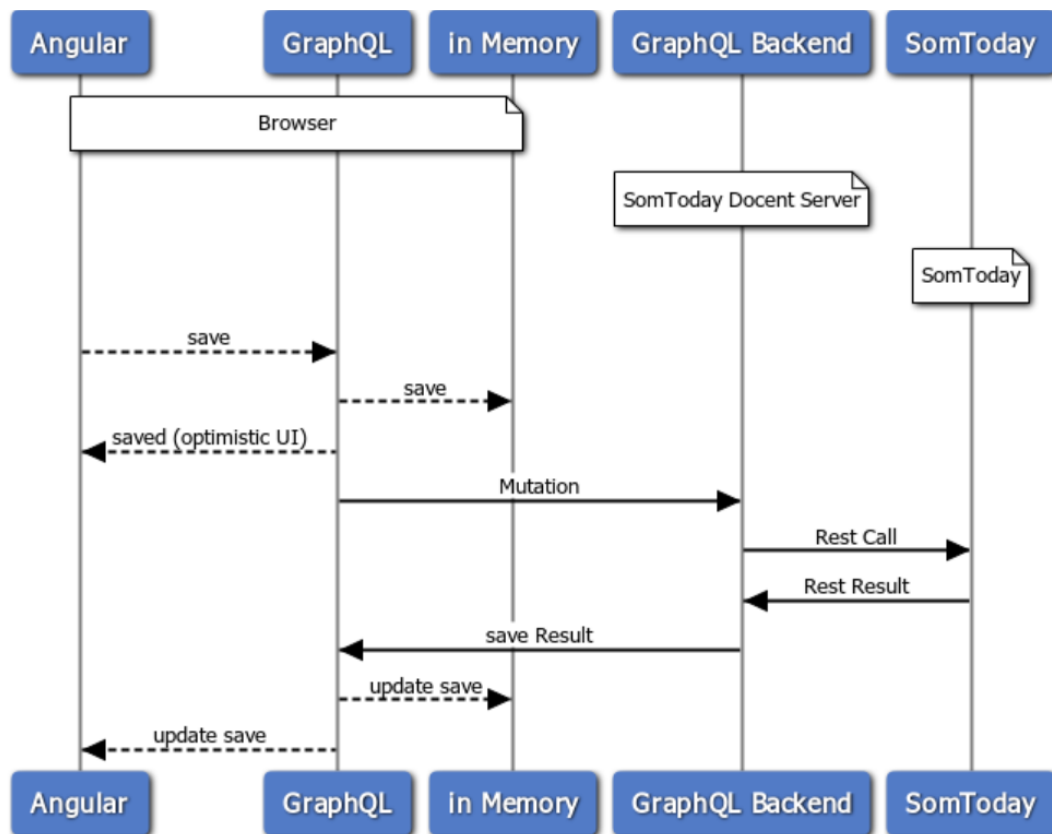


Figure 17: Communication flow between Somtoday Docent and Somtoday (Timmerman, 2018)

In the diagram above it can be seen that GraphQL consists of three parts: The GraphQL client (also known as Apollo: The GraphQL client for Angular projects) to which the Angular application communicates to, the memory of the browser and the GraphQL backend that communicates with the Somtoday REST endpoints. Whenever the Angular application wants to save data, the request is given to the GraphQL client. This first saves it in the browser memory and tells the Angular application that the data is saved,

although it is going to try afterwards. Even though it is not real data, the Angular application displays what it received from the GraphQL client (this is known as *optimistic UI*). Then, the GraphQL client passes a *mutation* object to the GraphQL backend, which resides within the Somtoday Docent server. The GraphQL backend makes the actual REST call to the Somtoday REST endpoint and gives its result back to the GraphQL client. Finally, the GraphQL client updates the memory because it retrieved a response and passes it back to the Angular application, which contains the actual data now.

3.5.2 Relevant files

With the communication flow being clear, it is now time to view the source code of Somtoday Docent and figure out its relevant files that are related to the communication with GraphQL. Inside the folder structure resides a special folder for GraphQL functionalities called *graphql*. When opening this folder, the files and folders are revealed as shown in figure 18:

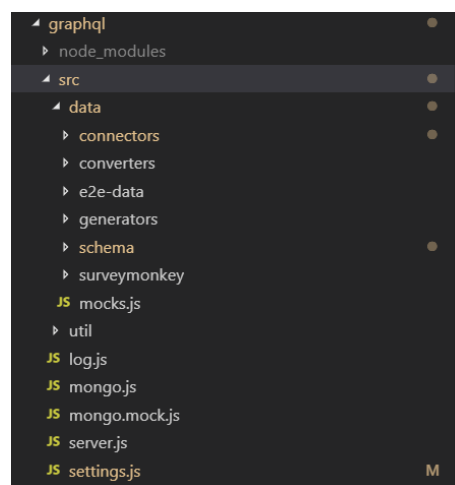


Figure 18: GraphQL folder content

The most relevant folders here are *connectors* and *schema*. The connectors folder contains connector files which contain functions that are linked to REST endpoints of Somtoday. These connector files are used in the schema class that hold GraphQL *schemas*. According to the official GraphQL website (GraphQL, n.d.), schemas are definitions of how the GraphQL queries are made. In the end, GraphQL queries are validated against schemas. In this context, connector files are used to define to which REST endpoints certain schemas are connected to. In the end, data is retrieved through GraphQL *queries* that request certain properties from an endpoint. These GraphQL queries are made in *data-services*. Data-services belong to components themselves and hence reside in the same folder. They cannot be found in the *graphql* folder, but in the *app* folder where all the components are. In the data-service files, the data retrieved from the GraphQL queries are converted to *models* which can be used in Angular components to display data. An example of relevant files of retrieve appointments can be seen in figure 19:

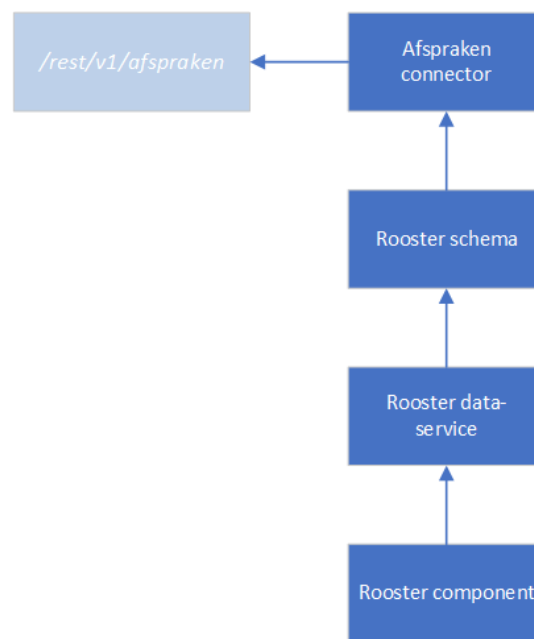


Figure 19: Relevant files for retrieving appointments in Somtoday Docent

To explain the above diagram: An `afspraken.connector.js` file exists that contains a function that retrieves appointments from the `/rest/v1/afspraken` endpoint, along with its filters given in the parameters. The `rooster.schema.js` file uses this function in a schema, so that GraphQL queries can be written to retrieve data from the `/rest/v1/afspraken` endpoint. This is exactly what happens in the `rooster-data.service.ts` file. It contains a GraphQL query that uses the schema defined for appointments and maps those to models. The `rooster.component.ts` file uses these models to display the data.

3.6 What requirements would end users like to have for the schedule-change notification system?

In order to comprehend the end users' requirements about the new system, a questionnaire has been made. Based on A. Bryman's book (2015) a good questionnaire has the following features:

- Questions should not be ambiguous;
- Questions must be easily understood;
- Questions should not require rigorous calculations;
- The questionnaire should not be too long;
- The questionnaire should cover the exact object of the inquiry.

Keeping in mind these aforementioned requirements the questions were created.

3.6.1 Questionnaire structure

The questionnaire comprised of six open and closed questions. The followings were:

1. What are the most important schedule changes that you wish to be notified about? (Rank them from 1 = most important to 7 = least important)

In this question users had to sort out from seven possible options which schedule changes are the most relevant for them.

2. How would you like to manage your notifications?

This question had three choices that the user could have selected: no options, global on/off switch or selecting the notifications per category.

3. Would you like to review notifications at a later date?

In this question users had two choices: yes or no.

4. Where would you like to receive your notifications?

This question had also two options only: only in Somtoday Docent or desktop notifications.

5. In which way would you like to see your schedule changes?

Users had two choices again: either one big notification view or multiple separate notifications.

6. When do you wish to get the notifications?

There were three selections possible: in the morning before lessons start, after the school day or during the day – whenever there is a change happening-.

This questionnaire was sent to 35 high school teachers that use daily Somtoday Docent. ‘SurveyMonkey’ was used to create the questionnaire because of the following advantages:

- It offers customisable questions with regards to the different types such as customer satisfaction or employee feedback;
- It provides an analysis tool that transfers the obtained data in presentable results;
- It has a free version that offers all the tools needed to write the questions, and analyse and show the gathered data;

The book *Statistics for dummies* (Rumsey, 2010) describes that a valid and reliable questionnaire must have a 95% confidence level. With the survey tool *SurveyMonkey* it is possible to calculate the error-margin that defines how much the survey can be generalised. The smaller the error-margin, the more trustworthy the survey is. SurveyMonkey calculates this considering the population, confidence level and sample size. Based on 3000 teachers, a 95% confidence level and sample size of 20 users that complete the questionnaire, the error margin will be of 22%.

3.6.2 Results

From the 35 submitted questionnaires, 25 teachers have filled it in within the two weeks' time of that running Sprint. As mentioned in the Research methodology (see section 3.1.), at least 20 respondents were needed to make the research results reliable and statistically significant. Based on the outcome, the goal has been accomplished and the error margin decreased from 22% to 20%.

The analysis of the questions' outcome has been assessed in the same sequential order as defined in the Questionnaire structure (see section 3.6.1.) The results are used to choose the Design of the implementation of the schedule-change notification system. A complete overview of the Questionnaire's results can be found in Appendix C.

Question 1 displays that the top three changes that teachers want to see in their schedule are: a new lesson that has been added and a change in time and/or location of a lesson. Figure 20 shows the results:

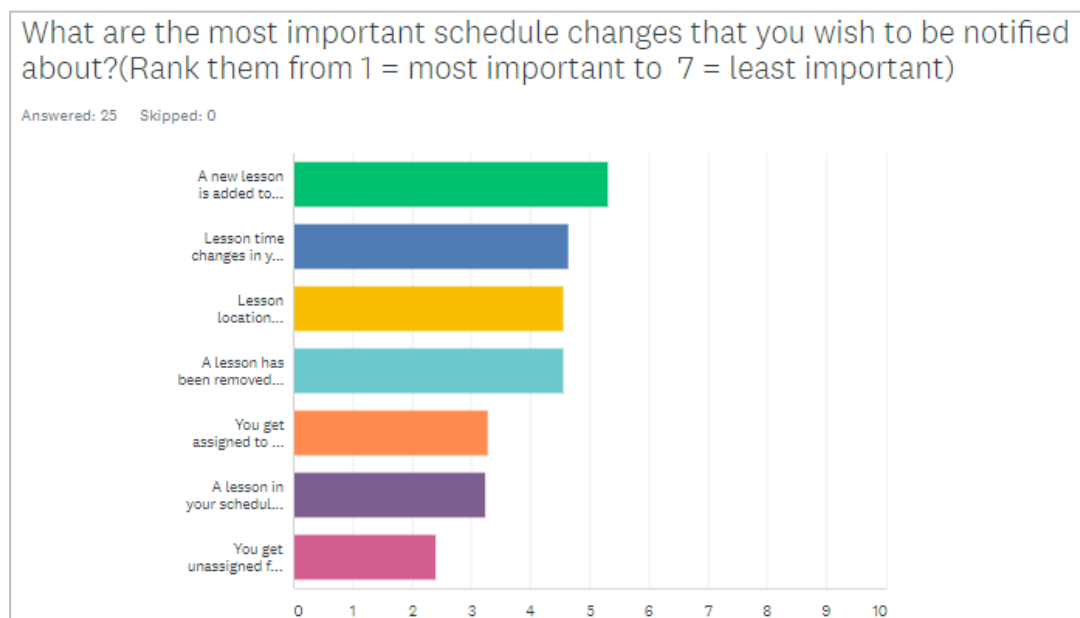


Figure 20: Most important schedule changes

Question 2 displays that 18 teachers out of 25 want to manage notifications per category. Figure 21 shows the results:

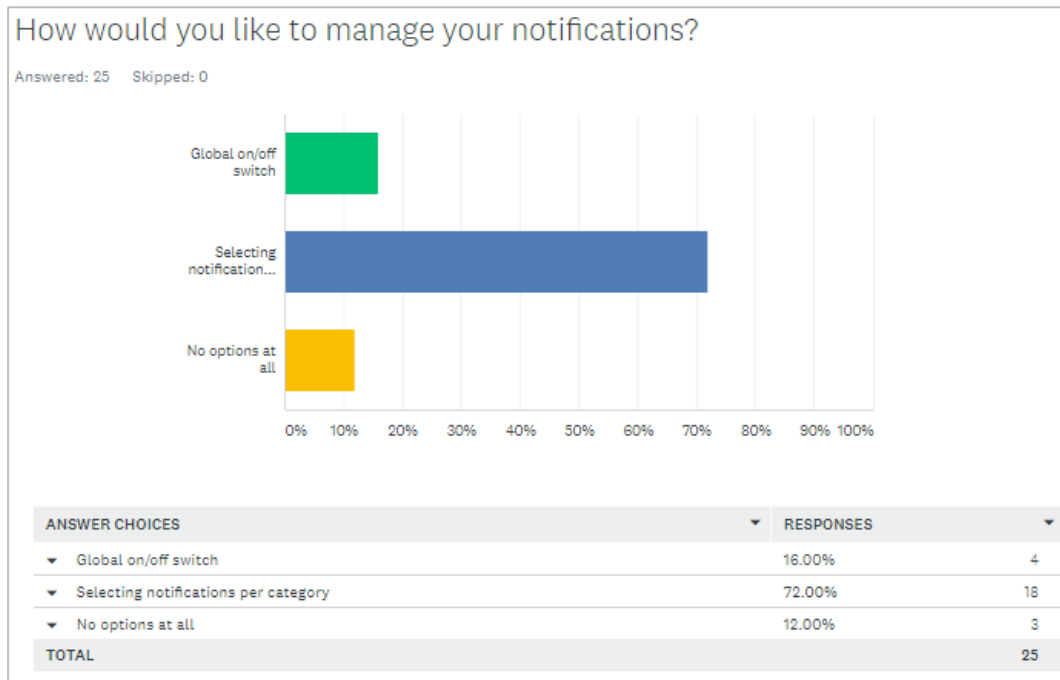


Figure 21: Notifications selection

Question 3 displays that 18 teachers out of 25 do not have interests to review notification at a later time. Figure 22 shows the results:

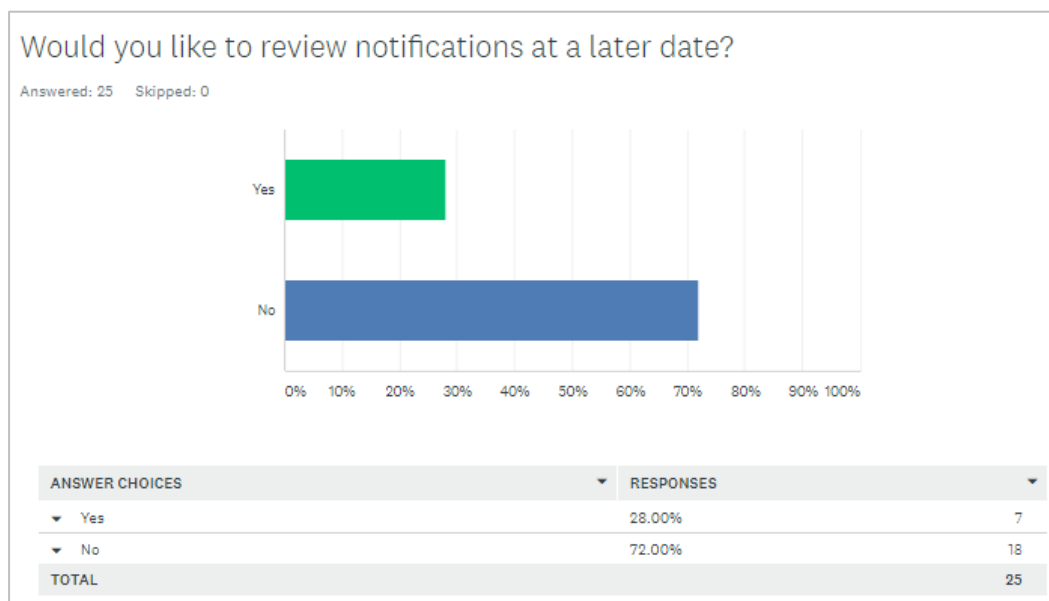


Figure 22: Review of notifications later in the day

Question 4 shows that 14 teachers out of 25 would like to see the notifications in Somtoday Docent. Figure 23 shows the results:

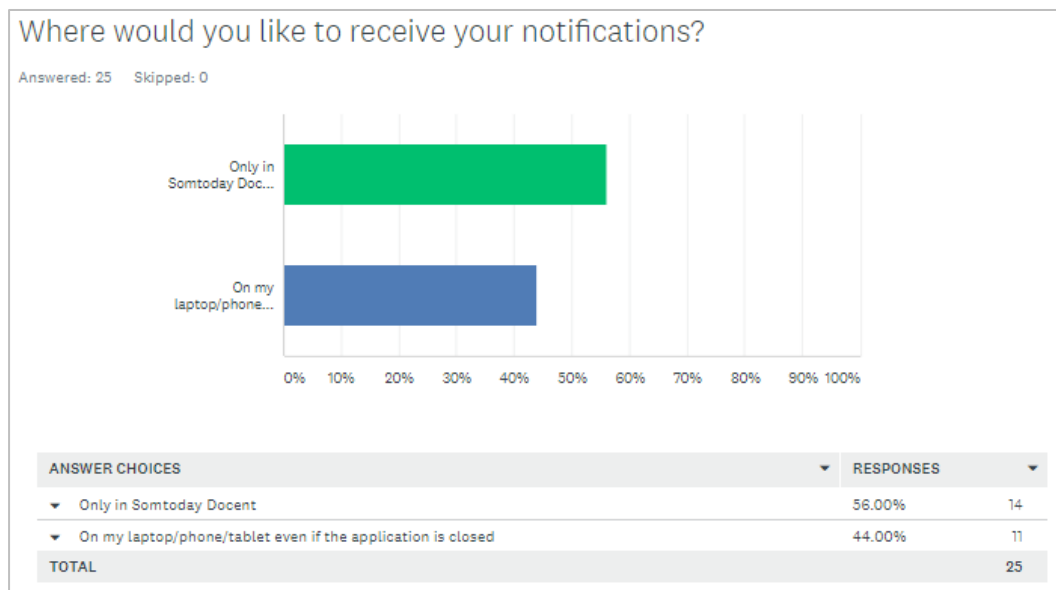


Figure 23: Notification display

Question 5 represents that 17 teachers out of 25 would like to have separate multiple notifications. Figure 24 shows the results:

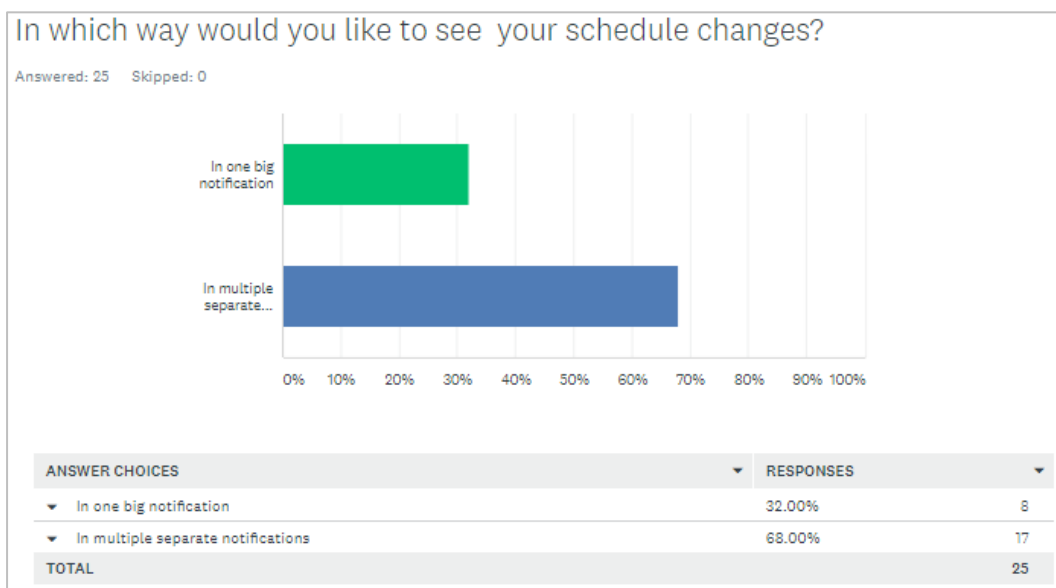


Figure 24: Notification receiver

Question 6 shows that 17 teachers out of 25 would like to receive these notifications during the day. Figure 25 displays the outcome:

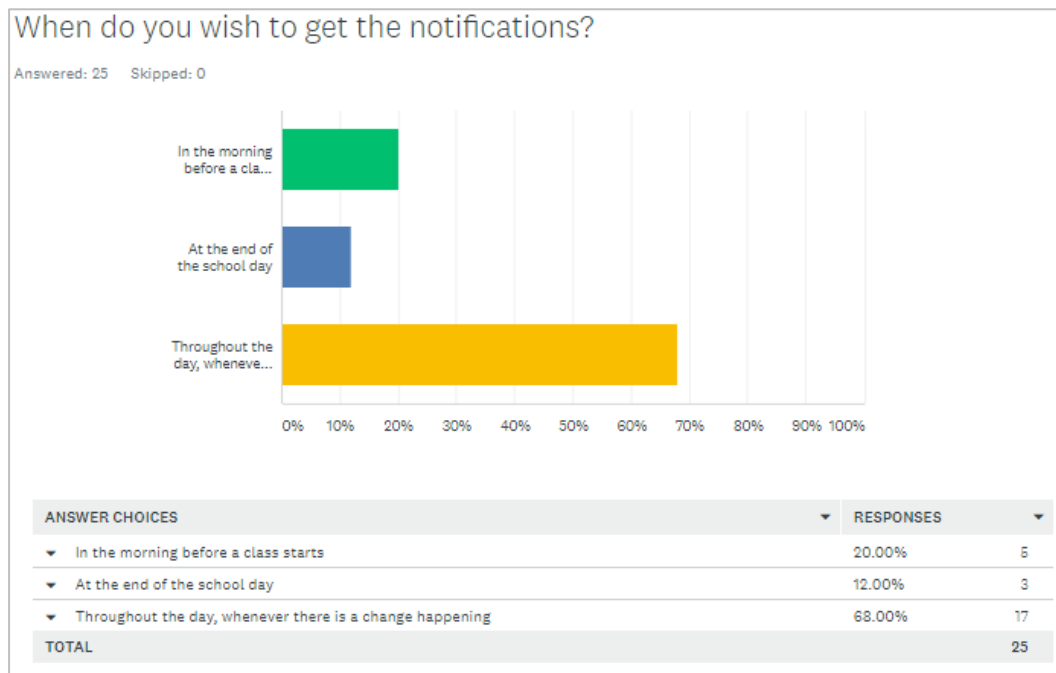


Figure 25: Notifications time

If the same research would be replicated, identical results will be retrieved because all the questions, other than question 4, are statistically significant. For this last question, considered the client's requirement, it is recommended to display notifications in Somtoday Docent.

4 Methodology and Approach

This chapter describes the methodology and approach used in the realisation process. The first section includes a description and a substantiation of the chosen methodology. The second section explains the tools used during the realisation phase. The university guideline allows for the graduation to choose among different methodologies that follow the SDLC process, which are Scrum, Kanban and the waterfall model. Within the Education department, it is possible to only work with Scrum and Kanban. Hence, it was necessary to choose between them to find the most suitable one for this project. The following activities and tools would favour the process:

- Iterated results;
- A list of tasks to organize activities;
- Prioritising tasks per iteration;
- A time estimation per task;
- A chart to keep track of the progress and to guard the time estimation;

Scrum contains all the previous aspects, whereas Kanban does not include a mean to estimate time and prioritise tasks. The other aspects are optional (LeanKit, 2014). Therefore, the chosen framework was Scrum. Note that the research and the realisation were not synchronous phases, but they were executed in parallel using the Scrum methodology.

4.1 What is Scrum?

“Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value” (Schwaber & Sutherland, 2017, page 3). In figure 26 is illustrated an example of the Scrum process:

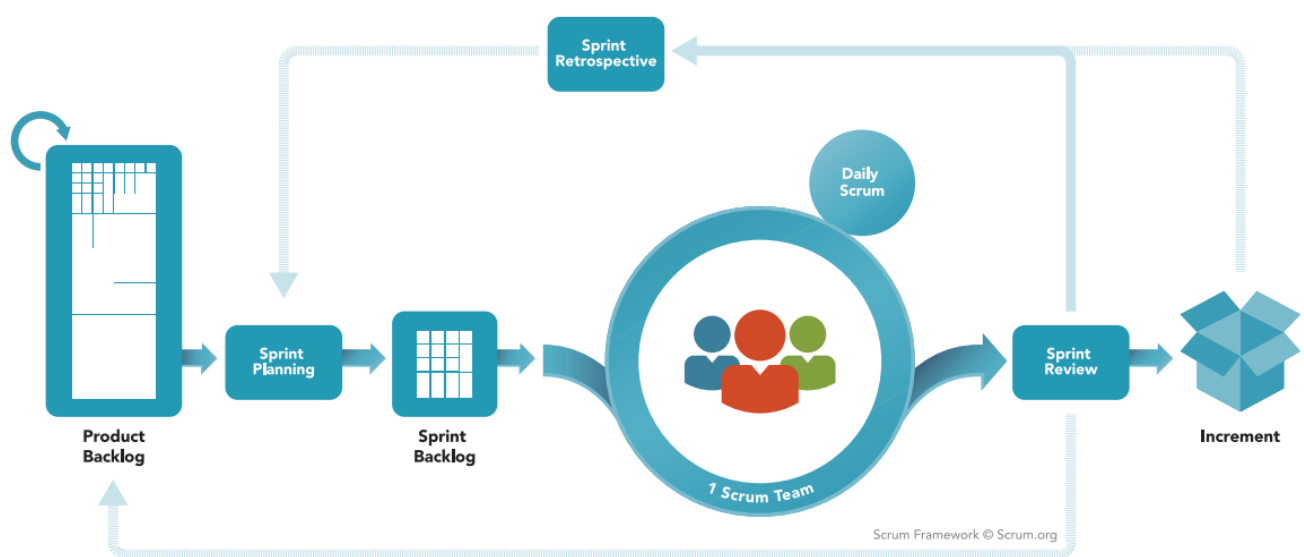


Figure 26: Scrum Framework (Schwaber & Sutherland, 2019)

4.1.1 Roles

According to Schwaber & Sutherland's Scrum guidebook (2017) a general Scrum team consist of a Scrum Master, a Product Owner and a Development Team.

The development team consisted of only M. Karunakalage. Because of the limited size of the team, the roles of Scrum Master and Product owner were not assigned. For learning purposes, the Product Owner's responsibilities were carried out by the Development team.

4.1.2 Events

Scrum consists of *events* and *artifacts*. Scrum events are time-boxed iterations, which means each iteration repeats these events and each event has its own time limit. The events used during the realisation process were:

The Sprint

A Sprint is a consistent iteration where User Stories are completed. It is never longer than a month and is usually about one or two weeks. At the start of every Sprint, there is a Sprint meeting in which some of the User Stories that contain defined points and a priority are moved from the Product Backlog (see section 4.1.3) to the newly created Sprint. For the development of the assignment, there were a total of four Sprints where each Sprint lasted for two weeks. Doing so, the client and the stakeholders were updated about the work progress. Each Sprint had an average of 21 points;

The Sprint review

A Sprint review is a periodic meeting that happens at the end of every Sprint to show the client and the stakeholders the progress of the assignment so that they can either approve it or ask for changes. Also, the Product Owner checks if the Sprint has been completed and implemented properly. For this assignment, there were a total of four Sprint reviews, each taking 45 minutes maximum;

The Sprint retrospective

A Sprint retrospective is a meeting that happens after the Sprint review to detect and improve a project's obstacles so that the same mistakes will not be made in future Sprints. The retrospective meeting contains the following questions:

1. What did we want to accomplish in this Sprint?
2. What actually happened?
3. Why did it happen?
4. What are we going to do next time?

For this assignment, there were a total of four retrospective meetings, each taking 30 minutes to one hour.

4.1.3 Artifacts

“Scrum’s artifacts represent work or value to provide transparency and opportunities for inspection and adaptation. Artifacts defined by Scrum are specifically designed to maximize transparency of key information so that everybody has the same understanding of the artifact.” (Schwaber & Sutherland, 2017, page 14). The following artifacts were used in the realisation process:

Product Backlog

The Product Backlog is an ordered list of elements that are desired for the end product. It lists all features, bugs and other tasks. The list consists of elements that fit in the project scope and those that are not feasible at the moment but can be completed in the future (Scrum guide, 2017). The complete Product Backlog of this assignment can be found in Appendix D. In figure 27 is delineated an example of Product Backlog:

	Votes	User Stories	Status	Points ▾
<input type="checkbox"/>	▲ 0	#86 Example	Ready ▾	12
<input type="checkbox"/>	▲ 0	#87 Example 2	Ready ▾	6
<input type="checkbox"/>	▲ 0	#88 Example 3	New ▾	9

Figure 27: Product Backlog

Every item in the Product Backlog has his order (or vote as illustrated in the example above), id, description, status and value. These items are called User Stories. A User Story is a description of a software feature for developers to have a clear vision of what is needed for the project and why. A User Story has the following structure:

As a <persona>, I want to be able to <user-requirement> so that <reason>

Sprint Backlog

“The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality into a done increment” (Schwaber & Sutherland, 2017, page 16). A Sprint Backlog is a collection of selected User Stories from the Product Backlog chosen to be implemented by the Development Team during the current Sprint. It must have a realistic total of User Stories that can be handled by the Development Team. The amount may vary based on how many points the Development Team can handle. The final Sprint Backlogs can be found in Appendix E. In figure 28 is illustrated an example of Sprint Backlog:

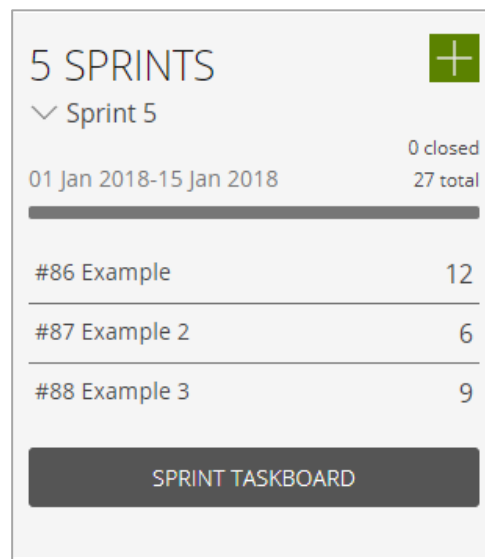


Figure 28: Sprint Backlog

Burndown Chart

The Burndown Chart is a graph that tracks the total work remaining, (on the vertical axis), and the time elapsed, (on the horizontal axis) since the start of a project. In this way, the team can view the progress of the total project. Having the same concept of the Burndown Chart, there is the Sprint Burndown Chart which is also used to track the work progress but with the difference that it displays the process of a single Sprint. The final Burndown Chart of the web application can be found in Appendix F. In figure 29 an example of a Burndown Chart can be viewed:

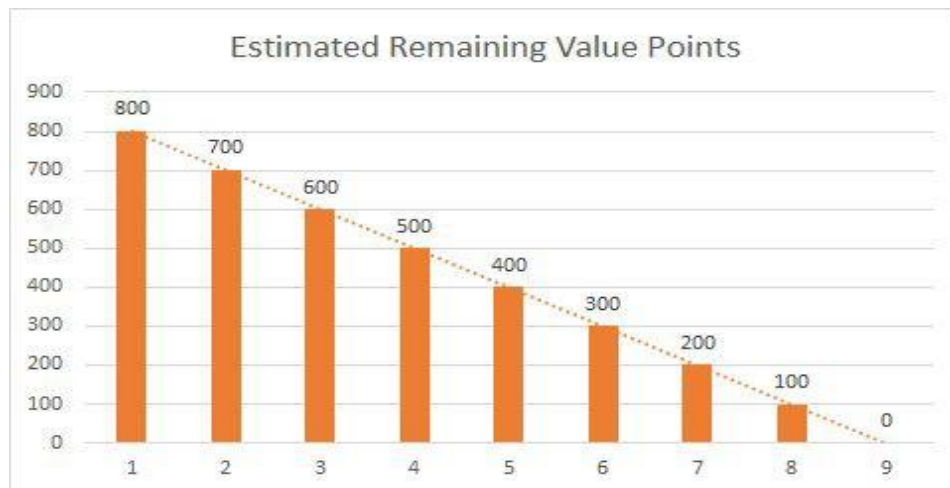


Figure 29: Burndown Chart

4.1.4 Definition of 'done'

A User Story can be considered *done* when it has passed these stages:

1. Code is produced;
2. Code is commented, checked and run against current version in source control;
3. Builds without errors;
4. Manual testing;
5. Remaining hours for task set to zero and task closed;
6. Project committed to GitHub.

4.2 Taiga

To keep track of the activities conducted during Scrum a tool named Taiga was used. Taiga is a web-based project management application that is used both for Scrum and Kanban frameworks. It was chosen over Jira because the intern was not going to have any personal space and it was not available outside of the office. Taiga uses boards that contain lists. The lists contain cards. These cards, which contain the tasks of the User Story, progress from one list to the other until they reach the list 'ready for test'. Only after the Sprint review and if there are no changes needed, they can move to last list: 'closed'. Users can be assigned to cards, such that everyone in the team knows who is responsible for which card. In figure 30 is shown an example of the Taiga dashboard:

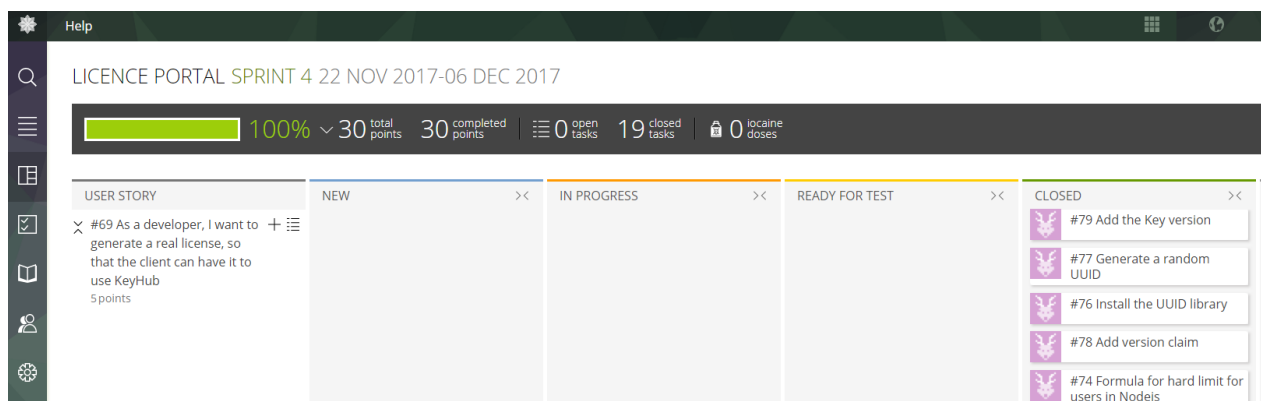


Figure 30: Taiga dashboard

4.3 Version control

Version control is a system that records the changes to a project, allowing, therefore, to track what the team has worked on so that versions are being managed. Git was the version control protocol used during this assignment (Cobra, 2019). Git comprehends the following terms:

- *Repository*: a project's folder that stores all the project's files, the documentation, the revision of that documentation and most importantly the full history of all commits;
- *Commit*: a recorded change of the source code;
- *Clone*: a copy function of a repository that lets developers download the project;
- *Branch*: it is a parallel version of a repository that is found inside the repository. In Topicus there are two kinds of branches: the master branch, which is the permanent branch that reflects a production ready-state, and the release branch, that is a temporary supporting branch for bug fixing and preparation of a new production release;
- *Fetch*: a feature that let users see the latest changes from an online repository and compare them, without merging, to the local branches on the desktop;
- *Fork*: a copy of a repository that allows users to make changes to the project without interfering with the original one;
- *Push*: it sends the committed changes to a remote repository so that others can access them;
- *Pull*: it fetches from the online repository and brings the local branch up to date;
- *Merge*: it synchronizes two branches.

GitHub was used for repository hosting. The files were hosted in a private source. Here all commits and revisions were placed for the product.

5 Realisation

This chapter describes the complete process of the realisation phase. Important findings, choices and results are declared. This in order to state what has been realised in total. Certain decisions were adjusted throughout this stage. The chapter is split into three parts: Design, Somtoday and Somtoday Docent. The design subchapter contains information about the total design of the notification system. Secondly, the Somtoday subchapter states what has been adjusted in the Somtoday core project in order to create the notifications. Finally, the Somtoday Docent subchapter describes how the notifications are retrieved from the Somtoday core project and how these are displayed. Note that the implementation of Somtoday Docent and Somtoday core happened in parallel.

5.1 Design

Before implementing the notification system, different concept designs were created to include all the possible questionnaire's outcomes.

5.1.1 Infrastructure

First, an infrastructure diagram was set up, since the other designs are dependent on this. Because the answers of the questionnaire were not yet retrieved, there was no certainty about the requirements of the notification system. Therefore, multiple infrastructural designs have been created and can be found in Appendix G. The final selected infrastructure is displayed in figure 31:

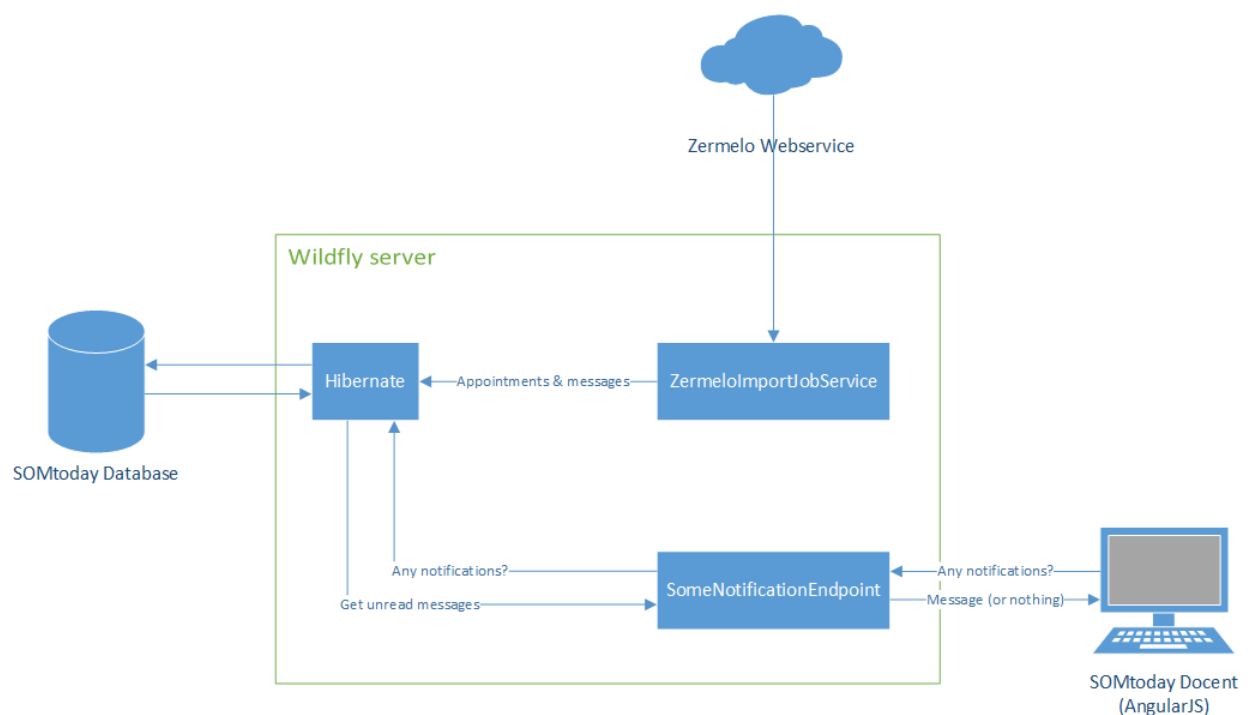


Figure 31: Final selected infrastructure

This option is the simplest to implement and to maintain. No need for real-time data synchronisation was required. The implementation would only happen on the application level, not database level.

5.1.2 Endpoint design

After the selection of the infrastructure, the `/roosterwijzigingnotificatieacties` endpoint was designed. This defined which data the endpoint was going to offer and what possible filters it would contain. The cobra framework (see section 3.3.2) explains that an endpoint is created according to a resource class. Therefore, a resource class has been designed that contains fields that are offered in the endpoint. Complete information about this class can be found in Appendix H.

5.1.2.1 Data fields

Throughout the design process, knowledge has been obtained about an existing messaging system in the Somtoday application. This project uses the properties of the `Actie` entity to send a message. Various classes exist that inherit from this `Actie` class, such as the `Boodschap` and `NotificatieActie` classes. In the end, the decision has led to create a class inherited from the `NotificatieActie` class since the Somtoday system recognises subclasses of this as a notification message. This enhances maintainability because the new class is categorised. The newly created entity is called `RoosterWijzigingNotificatieActie`. The most important properties are stated in the table 1:

Table 1: `RoosterWijzigingNotificatieActie` properties

Property name	Datatype	Exists in class	Description
<code>inhoud</code>	String	<code>Boodschap</code>	The content of the notification (e.g. defines what has been updated in the appointment).
<code>onderwerp</code>	String	<code>Boodschap</code>	The title of the notification
<code>actieType</code>	enum	<code>Actie</code>	Defines which action has been performed to create this notification. The options are: <code>CREATIE</code> = appointment has been created <code>WIJZIGING</code> = appointment has been updated <code>VERWIJDERING</code> = appointment has been deleted
<code>instelling</code>	<code>Instelling</code>	<code>InstellingsEntiteit</code>	The instance of a school institution. The notification will be sent to a teacher within this institution.
<code>verzendDatum</code>	Date	<code>Boodschap</code>	The date when the notification has been sent.

afspraak	Afspraak	RoosterWijzigingNotificatieActie	A link to the created or updated appointment. This way, the latest data can be retrieved. Is NULL if the appointment has been deleted.
medewerker	Medewerker	RoosterWijzigingNotificatieActie	A link to the teacher that the notification is sent to.
oudeBeginDatumTijd	Date	RoosterWijzigingNotificatieActie	The old start time and date of the appointment (if the time or date is updated, else NULL)
oudeEindDatumTijd	Date	RoosterWijzigingNotificatieActie	The old end time and date of the appointment (if the time or date is updated, else NULL)
zermeloId	Long	RoosterWijzigingNotificatieActie	A reference to the version ID that is given by Zermelo. This way, the latest change can be retrieved.

5.1.2.2 Filters

Because of performance and confidentiality reason, filters were designed. The most important filters and their reason for existence is stated in the table 2:

Table 2: Endpoint filters

Filter name	Parameter	Description
medewerker	The ID of the teacher.	By implementing this filter, the schedule page in Somtoday Docent only retrieves notifications of the current logged in teacher. This means that the current logged in teacher cannot view the notifications of other teachers.
jaarWeek	A year and week number divided with a ~ symbol.	By implementing this filter, the schedule page in Somtoday Docent only retrieves notifications of the currently selected year and week. This way, not all the notifications are retrieved but only from a certain week which decreased the number of notifications retrieved and therefore enhancing the performance.

5.1.3 Wireframes

In the final stage of the design process, wireframes were made to function as visual blueprints for the Somtoday Docent schedule page. The wireframe of a schedule with changes is displayed in figure 32:

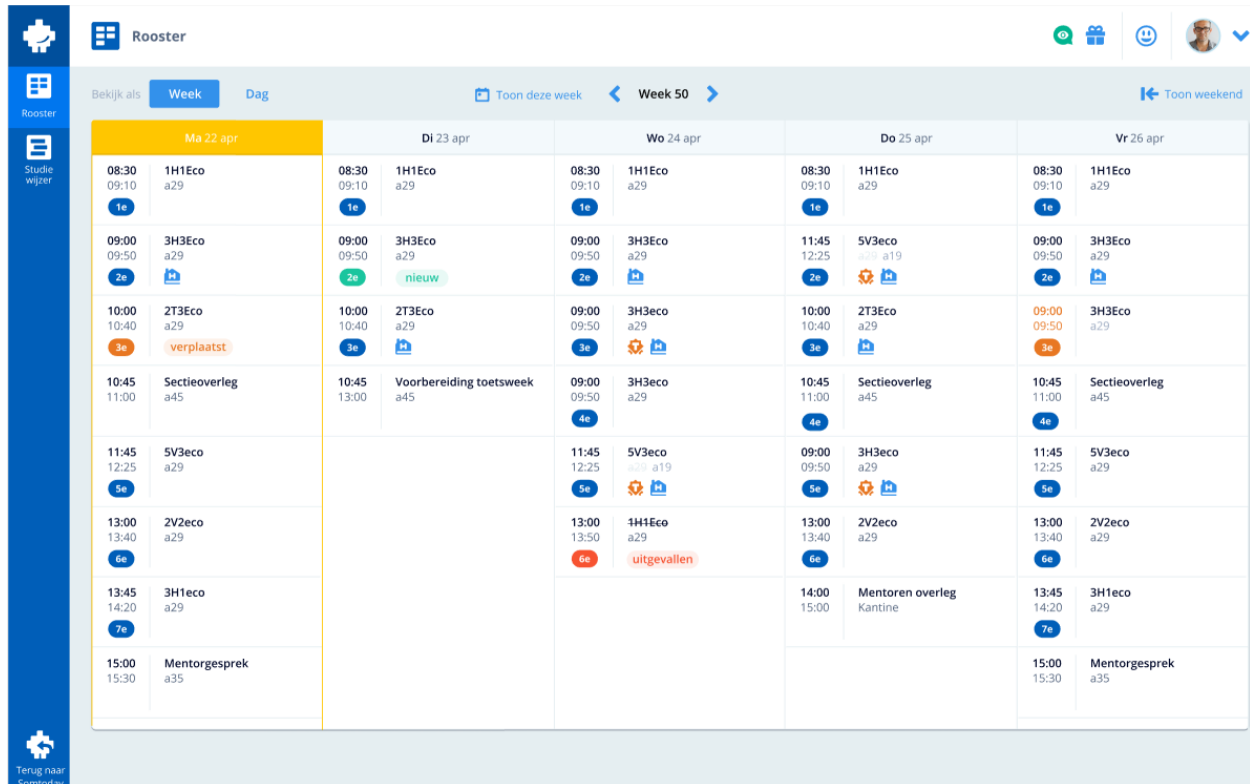


Figure 32: Wireframe schedule changes

In here, four different schedule changes can be seen:

- **New appointments:** Contains a green emblem on the left and a green icon with the text 'Nieuw' on the right;
- **Time changed appointments:** Contains an orange emblem on the left and the time is coloured orange as well;
- **Location changed appointment:** Contains an orange emblem on the left and an orange icon with the text 'Verplaatst' on it;
- **Deleted appointment:** Contains a red emblem on the left and a red icon with the text 'Uitgevallen' on it.

The completed wireframe can be found in Appendix I. After the creation of these designs, the realisation phase moved to the next phase: Implementing.

5.2 Somtoday

The development of every feature started with the Somtoday core project. In order to display notifications, they would first need to be saved to the database. Three different functionalities would save notifications: Creating, updating and deleting appointments. In order to save notifications in the first place, a setup procedure had to be done.

5.2.1 Prototype

The prototype phase consisted of three parts: Setting up the development environment, saving the notifications to the Somtoday database and exposing the notifications via a rest endpoint. Before these, a test environment was set up on the servers of Topicus which could be accessed remotely. One more step was required to complete the setup of the development environment: creating a local test environment that connects to the database that was just created. The first activity was to pull the source code from the GitHub repository. The Somtoday source code resides in the *Iridium* repository. The master branch was pulled since this branch contains the latest, stable changes available. The second activity was to install Wildfly, which consisted of downloading the Wildfly installation files and the installation of the Eclipse plugin called *JBoss Tools* to run a Wildfly server. The final activity was connecting the Wildfly server to the development database. This was done through the *Wildfly administrator panel*.

A development environment of Zermelo existed before the start of the realisation phase. This is a remotely accessible portal where developers can create data to import in the Somtoday core project. The only required aspect is to have a link to the Zermelo development environment in the *oauthclient* table. Once the database setup was completed and logging in was possible, the first step of the setup was complete. This created the possibility to save notifications to the database that was just set up. First, a class was required in order to save a notification, since Hibernate uses classes and maps them to the database. This led to the creation of the `RoosterWijzigingNotificatieActie` class, which inherits from the `Actie` class. Because of the inheritance, Hibernate would store the data in the *actie* table. The *actie* table did not contain all the required columns that were required for the newly created `RoosterWijzigingNotificatieActie` class (e.g. *oudeBeginDatumTijd*, *oudeEindDatumTijd* and *zermeloid*), so these were created during the process. With the new class and the missing columns created a test notification could be saved. The code from figure 33 was added to the method `vulAfspraakAan` in the `RoosterImportJobService` class.

```
// CREATE ROOSTERWIJZIGING NOTIFICATION
Medewerker teacher = IridiumAfspraakService.getUniekeDocent(afspraak);
RoosterWijzigingNotificatieActie notification = new RoosterWijzigingNotificatieActie(
    afspraak, teacher, instelling, ActieType.CREATIE);

notification.setOnderwerp("Create afspraak test");
notification.setInhoud("Wow, we actually imported a new afspraak!");
notification.setPrioriteit(Prioriteit.NORMAAL);

Entities.save(notification);

// Create Actierealisatie with teacher linked to it
Actierealisatie actierealisatieTest = new Actierealisatie(notification);
actierealisatieTest.setIridiumAccount(teacher.getAccount());
Entities.save(actierealisatieTest);
```

Figure 33: Saving test notification to database

This piece of code takes the following steps: First it retrieves the teacher that is linked to the appointment that is just created. This is used in the constructor of the notification in order to link it to the teacher that is also linked to the appointment with the static `IridiumAfspraakService` class. After this, it sets some test data to the notification which will be adjusted later and tells Hibernate to save this entity to the database. Finally, an `ActieRealisatie` object is created with a link to the created notification. With this, later features are possible, such as marking the notification as read. The result of importing appointments from Zermelo can be seen in figure 34:

select * from actie Enter a SQL expression to filter results (use Ctrl+Space)									
	abc dtype	123 id	123 version	✓ draft	abc inhoud	abc mimetype	abc onderwerp	abc pri	
1	RoosterWijzigingNotificatie	1,845,582,210	1	false	[NULL]	[NULL]	[NULL]	[NULL]	
2	RoosterWijzigingNotificatie	1,851,582,210	1	false	[NULL]	[NULL]	[NULL]	[NULL]	
3	RoosterWijzigingNotificatie	1,852,582,211	1	false	[NULL]	[NULL]	[NULL]	[NULL]	
4	RoosterWijzigingNotificatie	1,852,582,212	1	false	[NULL]	[NULL]	[NULL]	[NULL]	
5	RoosterWijzigingNotificatie	1,858,082,410	0	false	[NULL]	[NULL]	[NULL]	[NULL]	
6	RoosterWijzigingNotificatie	1,858,082,411	0	false	[NULL]	[NULL]	[NULL]	[NULL]	
7	RoosterWijzigingNotificatie	1,858,082,412	0	false	[NULL]	[NULL]	[NULL]	[NULL]	
8	RoosterWijzigingNotificatie	1,859,582,510	0	false	Wow, we actually imported a new afspraak!	[NULL]	Create afspraak test	[NULL]	

Figure 34: Saved test notifications in database

Now that notifications can be saved in the database, it is possible to present them through an endpoint. The classes required for the endpoint have been created using the Cobra framework conventions, as mentioned in the Cobra framework research (see section 3.3.2.) The following files were created:

- `RRoosterWijzigingNotificatieActie`: The REST representation of the notification. It contains the fields that are presented in the response. All the fields offered in the `RoosterWijzigingNotificatieActie` class are available in this class. It inherits from the `RBoodschap` class to present the properties of that class and the `RActie` class as well.
- `RRoosterWijzigingNotificatieActieResource`: Contains the definition of the endpoint (e.g. the path of the endpoint. In this case `/roosterwijzigingnotificatieacties`). It inherits from the `CrudResource` class so that the Cobra framework converts this class to an endpoint according to the properties in the `RRoosterWijzigingNotificatieActie` class.
- `RoosterWijzigingNotificatieActieZoekFilter`: Contains the logic of the search filters of the endpoint.
- `RoosterWijzigingNotificatieActieDAO`: A class that is normally used to access the data of the entity from the database, but in this case solely used for containing the search filter definition.
- `RRoosterWijzigingNotificatieActieResourceImpl`: The final link to complete the endpoint. It inherits from the `AbstractPlatinumCrudRootRestService` class which links the normal entity and the resource class together. It contains a method that returns a `RoosterWijzigingNotificatieActieDAO` to retrieve the search filters.

After the implementation of these files and running the Wildfly server, the `/roosterwijzigingnotificatieacties` endpoint could be accessed. A response of the endpoint is shown in figure 35:

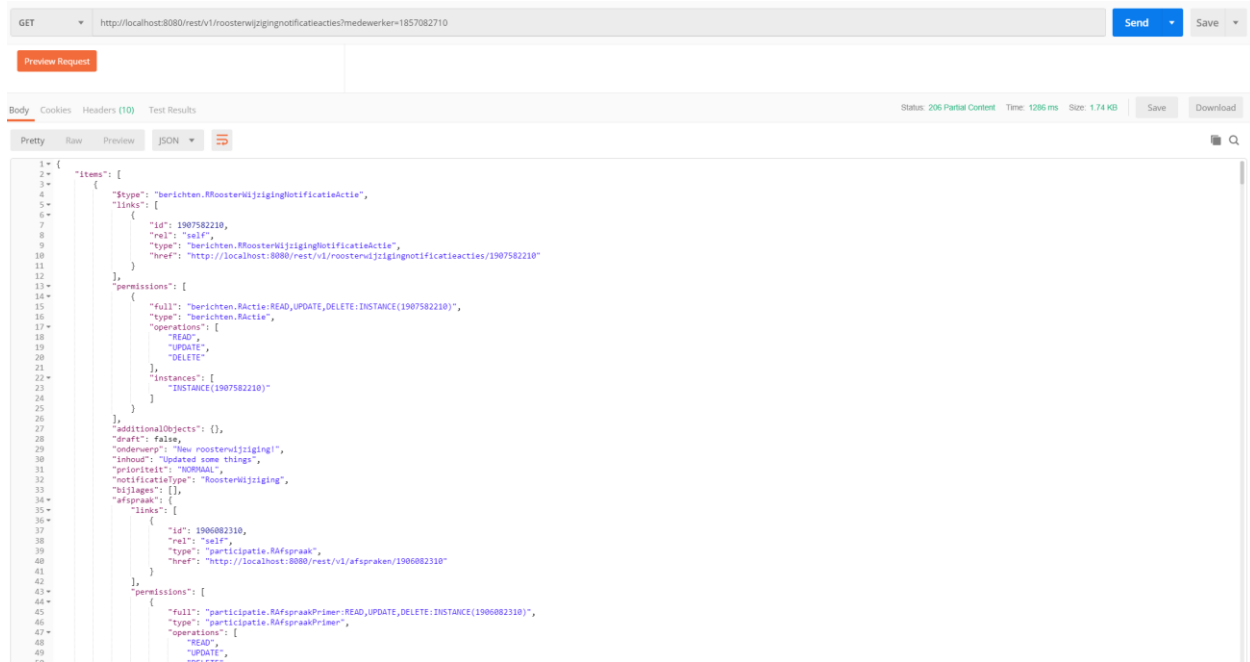


Figure 35: notification endpoint response

With this setup complete, the possibility existed to develop the product further.

5.2.2 Create new appointment

Saving notifications of newly created appointments has already been developed during the setup phase. The only point to note is to make sure that the `ActieType` is set to `ActieType.CREATIE`. With this setting, Somtoday Docent will see this notification as a newly created appointment. The total process of creating a notification of a newly created appointment in Zermelo can be found in Appendix J.

5.2.3 Update existing appointment

Compared to the process of creating appointments, updating appointments is more complex. In the Zermelo research (see section 3.4.2.) it is described that when an appointment gets updated in Zermelo, it creates a new appointment with a higher version number. Before, this version number was not saved in the database of Somtoday, so there was no possibility to determine the latest change. As for this, the `ZermeloRoosterImportJobService` only checked on the external ID (zermelo appointment instance) to see if the appointment already existed in the database. If so, it would update the appointment in the order that the Zermelo web service provided the appointments, which resulted in a wrong output at times. Therefore, a new column was given to the `afspraak` table: the `zermelodid`. This contains the version number retrieved from Zermelo to determine the latest change. After this, an adjustment has been made to the `ZermeloRoosterImportJobService`: it now checks the latest zermelo ID before updating the appointment. If the given appointment from Zermelo has a higher ID than the current Zermelo ID saved, that means that a new update has taken place. The appointment is then updated, the Zermelo ID is

overwritten and a notification with the type `ActieType.WIJZIGING` is saved. The complete update process can be found in Appendix K.

5.3 Somtoday Docent

After the setup of the endpoint that displays notifications, a setup could be created to retrieve them in the Somtoday Docent application. Next, certain components were changed to display the various types of schedule changes.

5.3.1 Prototype

The prototype would be completed if the right notifications of the current logged in user could be shown in the console. To make that possible, the following files were changed:

- `afpraken.connector.js`: A new function has been created called `getRoosterWijzigingen()`. This function connects to the endpoint and passes the filter parameters (e.g. teacher ID, year, week);
- `rooster.schema.js`: A new type and query definition have been made for `roosterwijzigingen`. A new method has been added to the resolver: `roosterWijzigingen()`, which makes use of the newly created method in the `afpraken.connector.js`;
- `roosterwijziging.model.ts`: A new model has been created to represent notifications and to use them as objects. This gets used in the `rooster-data.service.ts`;
- `rooster-data.service.ts`: A new GraphQL query has been created to retrieve the right fields. The private method `roosterWijzigingenQuery` executes that query which is followed by `getRoosterWijzigingen` that converts them to a model;
- `rooster.component.ts`: Finally, the list of notifications is outputted in this file. In the private function `applyDate`, which is activated when a new week has been selected. The code in figure 36 outputs the notifications to the console:

```
this.roosterDataService.getRoosterWijzigingen()
  .subscribe((wijziging) => {
    console.log(wijziging);
  });
```

Figure 36: notifications shown in console

This resulted in the output shown in figure 37:

```
▼ (18) [{"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}, {"-"}] 1
  ► 0: {onderwerp: null, inhoud: null, __typename: "RoosterWijziging"}
  ► 1: {onderwerp: null, inhoud: null, __typename: "RoosterWijziging"}
  ► 2: {onderwerp: null, inhoud: null, __typename: "RoosterWijziging"}
  ► 3: {onderwerp: null, inhoud: null, __typename: "RoosterWijziging"}
  ► 4: {onderwerp: null, inhoud: null, __typename: "RoosterWijziging"}
  ► 5: {onderwerp: null, inhoud: null, __typename: "RoosterWijziging"}
  ► 6: {onderwerp: null, inhoud: null, __typename: "RoosterWijziging"}
  ► 7: {onderwerp: "Create afspraak test", inhoud: "Wow, we actually imported a new afspraak!", __typename: "RoosterWijziging"}
  ► 8: {onderwerp: "Create afspraak test", inhoud: "Wow, we actually imported a new afspraak!", __typename: "RoosterWijziging"}
  ► 9: {onderwerp: "Create afspraak test", inhoud: "Wow, we actually imported a new afspraak!", __typename: "RoosterWijziging"}
  ► 10: {onderwerp: "Create afspraak test", inhoud: "Wow, we actually imported a new afspraak!", __typename: "RoosterWijziging"}
  ► 11: {onderwerp: "Create afspraak test", inhoud: "Wow, we actually imported a new afspraak!", __typename: "RoosterWijziging"}
  ► 12: {onderwerp: "Create afspraak test", inhoud: "Wow, we actually imported a new afspraak!", __typename: "RoosterWijziging"}
  ► 13: {onderwerp: "Create afspraak test", inhoud: "Wow, we actually imported a new afspraak!", __typename: "RoosterWijziging"}
  ► 14: {onderwerp: "Create afspraak test", inhoud: "Sprint demo afspraak ready :)", __typename: "RoosterWijziging"}
  ► 15: {onderwerp: "Create afspraak test", inhoud: "Sprint demo afspraak ready :)", __typename: "RoosterWijziging"}
  ► 16: {onderwerp: "Create afspraak test", inhoud: "Sprint demo afspraak ready :)", __typename: "RoosterWijziging"}
  ► 17: {onderwerp: "Create afspraak test", inhoud: "Sprint demo afspraak ready :)", __typename: "RoosterWijziging"}
  length: 18
  ► proto : Array(0)
```

Figure 37: notifications shown in console

After this connection prototype, the notifications could be displayed in the schedule itself.

5.3.2 Create notifications

For creation notifications, two sections would be changed: The colour of the emblem on the left and a new icon on the right. In the wireframe, a newly created appointment looks like figure 38:



Figure 38: Wireframe new appointment

These items can both be found in the `lesuur-afspraak` component. This component displays the elements based on the `afspraak` parameter it retrieves from the `rooster-dag` component. Therefore, a new field has been added to the `afspraak.model.ts` file: `roosterWijzigingType`. This is a property that, when set, describes what type of schedule change exists for an appointment. When this property is set to 'Creatie', a CSS class will be added which turns the left emblem green and the additional icon will be displayed. After the display implementation, the notifications needed to be linked to an appointment in the schedule.

A new function has been created in the `rooster.component.ts` file, called `showRoosterWijzigingen`. This function first sorts the notifications based on the ascending Zermelo ID and loops through them. This way, the appointments are ensured of the latest changes. The loop finds the appointments in the schedule through the appointment ID's in the notification. Once found, the type of notification is determined: Creation, update or deletion. A screenshot of a schedule with new appointments can be seen in figure 39:

Ma 10 jun		Di 11 jun		Wo 12 jun	
9 uur		6 uur		3 uur	
17:00 18:00 les	4 - vlen 4	14:00 15:00 les	1 - vlen 1 nieuw	11:00 12:00 les	1 - vlen 1 nieuw
17:00 18:00 les	4 - vlen 4				

Figure 39: New appointments in schedule

5.3.3 Update notifications

Like the created appointments, the updated appointments would have two elements changed: the emblem on the left and the time colour would change. In the wireframe, an updated appointment looks like figure 40:

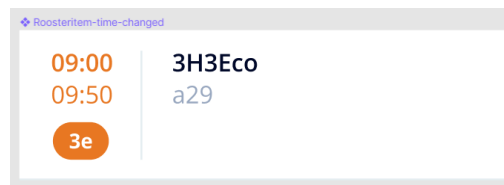


Figure 40: Wireframe updated appointment

The result was achieved through the same development process as the created appointments: a CSS class name is added based on the `roosterWijzigingType` enum in the `afspraak` model. The `showRoosterWijzigingen` function would map the notification to the appointments in the schedule and would give them the type: `RoosterWijzigingType.Wijziging`. The result of an updated appointment can be seen in figure 41:

Ma 10 jun		Di 11 jun		Wo 12 jun	
9 uur		7 uur		3 uur	
17:00 18:00 les	4 - vlen 4	15:00 16:00 les	7 - vlen 7	11:00 12:00 les	1 - vlen 1 nieuw
17:00 18:00 les	4 - vlen 4				

Figure 41: Updated appointments in schedule

5.4 Test Strategy

A test strategy has been created to improve the quality of the product. This strategy was established with the help of the *testing pyramid*. Various tests, derived from the layers in this pyramid, could be implemented into the product. For the complete test strategy and the reasoning of this, see Appendix L.

6 Conclusion

This chapter is describing the conclusion of the entire project. A number of recommendations are also made that can serve as a follow-up to the project.

Somtoday is a software solution developed by Topicus Onderwijs B.V. that helps high school instructions managing their information, students and staff. The Somtoday team recently developed a new module: *Somtoday Docent*, which solely supports the activities of teachers. One of the main functionalities is that teachers can view their schedule. This functionality is missing two important requirements: Changes being applied in schedules are not tracked and therefore teachers cannot see what changes were made in their schedule.

The goal of this graduation was to create a prototype application for the Somtoday Docent module where it is possible to detect and track changes in the schedules and consequently notify the teachers. The following main research question was used during the entire process of the graduation: *“How can a schedule-change notification system be developed within the platform of Somtoday regarding the requirements of the end users?”* To answer the main question, research has been conducted about Somtoday, Somtoday Docent, Zermelo and how all of these are connected. Along the analysis of the technical aspects, a questionnaire was sent out to teachers to obtain the requirements of the prototype to develop.

First, the Somtoday Docent module was analysed. This would benefit the realisation of the schedule change system because the application where the schedules are displayed would be examined. Two aspects have been examined: the infrastructure of Somtoday Docent and the project structure regarding the visualisation of the schedules. What has been found is that Somtoday Docent is an *Angular* web application that uses *GraphQL* to connect to the Somtoday REST API to exchange data. It uses the OAuth functionality of Somtoday to authenticate. This way, teachers are able to log into Somtoday and can be redirected to the Somtoday Docent application. Because Angular was used for building Somtoday Docent, its structure is divided into components. Within this project, data is retrieved from a `data-service` file, which is then translated to a schedule with appointments through a component tree structure.

After the clarification of the Somtoday Docent module, relevant aspects of the *Somtoday core* project were analysed since this is the origin of the appointments that are loaded in Somtoday Docent. What has been found is that Somtoday core is a Java Wicket web application that is running on a *Wildfly* application server and saves its data to a *PostgreSQL* database. It uses *Hibernate* to map Java objects to relational database tables and Topicus Onderwijs’ internally developed *Cobra* framework. Out of these technologies, the Cobra framework was the most relevant to research about, since it defines the project structure of the entire Somtoday application.

Somtoday core is built with Topicus Onderwijs’ Cobra framework with the goal of decreasing development time. Cobra contains a set of modules that can be reused. Cobra applications contain a generic project structure that separate REST API and web application related files. With *Data Access Objects (DAO’s)*, developers are able to retrieve and manipulate data in the form of objects. These can then be used in combination with a REST resource object to create a *REST endpoint*. This is a URI that exposes entities for external applications to consume. The project structure of Somtoday core contains the generic Cobra project structure as well, REST API logic is separated from the web application logic. It contains this and more internally developed technologies, such as the `Entities` class, which saves and manipulates

objects in the form of entities. The most common used entities are: `Organisatie` that represents organisational data, `Medewerker` that represents data of employees, `IridiumAccount` which contains credentials, `Afspraak` that represents appointments, `AfpraakParticipant` that functions as a link between appointments, teachers and students, `Actie` that represents a message and `ActieRealisatie` that represents the fulfilment of a message.

Somtoday core contains a functionality that imports appointments from *Zermelo*, a school schedule management application. A development environment exists wherein Somtoday developers can create dummy appointments to test their functionalities. Both creating and updating appointments are possible by sending a POST request to the Zermelo REST API. Zermelo assigns an *ID* used for versioning and an *appointmentInstance* used to reference to one specific appointment.

Importing appointments from Zermelo is possible via the *Zermelo webservice*. This is done through a *background job*, which executes independently from the UI. The job lists the appointment and compares them to the appointments that are currently stored in the Somtoday database. It creates a new appointment, updates existing and deletes appointments that are not available in Zermelo anymore.

With the information obtained about Somtoday, Somtoday Docent and Zermelo, it was possible to analyse the connection between them. GraphQL is responsible for retrieving appointments from the Somtoday REST API and passing them to the Somtoday Docent application. It is the intermediate layer which requests specific data fields and uses the in-browser memory to keep the UI status updates. GraphQL defines *connectors* that describe which REST endpoints the application is connected to, *schemas* that describes query syntax where these connections can be used in, *queries* that specify the data to be retrieved and *data-services* that execute queries and map them to objects that can further be processed by the Angular components.

Although the technical aspects were examined, it was still unclear what kind of notifications the teachers would like to see on their schedule. A questionnaire with six questions was sent out. 25 out of 35 teachers responded with the answer that new lessons, time changes and location changes would be the most relevant schedule changes. Furthermore, the answers state that the notifications should be visible in multiple, separate messages that should be selected by category. They should be retrieved throughout the day, only in Somtoday Docent and there is no need for reviewing them later.

Scrum has been used during the realisation of the prototype. The main reason for this decision was that Scrum and Kanban were the only methodologies used with the Education department. Also, Scrum would favour the realisation process because it creates iterated results, organizes activities and guards time estimation, whereas Kanban does not contain these benefits. Four Sprints have been done, each consisting of two weeks and each ending with a Sprint review and a retrospective. The artifacts Product Backlog, Sprint backlogs and Burndown charts have been used. The scrum tool Taiga was used for maintaining the scrum process and GitHub for version control of the source code.

Because the research phase was executed parallel to the realisation phase, multiple infrastructure designs were made to cover the multiple outcomes of the questionnaire. After the results were collected, the most suitable was chosen. This defined a REST endpoint that would expose Somtoday appointments with the help of Hibernate. It would use properties from the existing `Actie` class that contains properties for sending messages. After this, the fields that would be exposed were listed as a REST resource design to fit

within the Cobra framework. In the final stage of the design process, wireframes were made that visualised how these exposed fields would look like within the schedule shown in Somtoday Docent.

After the creation of the designs, the prototype schedule change system was developed. This started with the setup of the local development environment of Somtoday. After this, the fields defined in the design were implemented in the form of a class, so that the process of saving schedule changes could be saved. Within the import job, the newly created appointments and the update of existing appointments would be detected and schedule changes would be saved. After this, the */roosterwijzigingnotificatieacties* endpoint was made to expose the schedule changes. The Cobra framework was referred when creating this.

With schedule changes being exposed in the REST API, Somtoday Docent would be able to retrieve and display them. The newly created REST endpoint was added in the connections list, a schema and a query were made and the mapping logic was created in the data-service. These were then translated through the component tree to display newly created and updated appointments in the schedule of teachers.

6.1 Recommendations

This section is dedicated for recommendations regarding the project.

‘Seen’ button for notifications

In the wireframe it was created an icon that teachers can ideally click to set the new or changed appointments to an existing blue-coloured appointment. A suggestion would be to implement this feature to improve the user experience of teachers.

Selecting notifications per category

One of the questions from the questionnaire outlined the fact that teachers would like to manage the notifications per category. This is the only requirement from the questionnaire that has not been realised. Hence, it is a possible next step for improvement of the module.

Implementing tests

The intern made a test strategy according to the test pyramid model (see Appendix L). Because of time constraints this was not implemented. Hence, it is suggested to create these tests to ensure the quality of the functionalities with every adjustment made.

Implementing the new module in the current Somtoday system

The intern created a proof of concept that meets all the requirements from the stakeholders and the client. It is advisable because of its business value to implement the proof of concept in the Somtoday system.

Knowledge sharing

Gathering data about the Zermelo importer and its implementation in Somtoday was a big challenge. This because of the limited availability of up-to-date documentation. Also, because the Somtoday team did not know how an important part of their system works. According to Hansen (1999), there are two strategies for sharing knowledge: codification and personalisation. It is evident that first strategy is applied but within this project is not sufficient. There is the urge to use the second strategy which is sharing knowledge between two or more employees.

7 Evaluation

This chapter is used to evaluate the graduation assignment and to reflect the intern's performance throughout the entire process.

At the beginning of the graduation two objectives were defined by the University and by the company supervisor. The first one was a fast integration of the intern in a large company environment within the graduation period. The second was the intern completing the MVP (Minimum Viable Product) defined by the company supervisor. Regarding the first objective, I effectively managed to adapt myself within the company, especially in the new team. This is because I always try to keep an open mind and a proactive attitude. So that the transition from one environment to another can be as smooth as possible. Also, Topicus has an informal atmosphere which allows freedom within certain boundaries and that fits very well with my personality. Regarding the second objective I successfully completed the MVP within the given time. The realised product is in the end, a proof of concept that can detect and track changes in the schedule. Extra features were implemented such as the design and implementation for the visualisation of the required notifications and the integration with corporate design. This has a high value to Topicus Education because it is finally possible to track schedule changes, without having the importer delete all the existing data and import all new ones in the schedule. Furthermore, this can be used as basis for developing other import routines.

The assignment had a high level of complexity, which made it very suitable for the graduation level. This was because of the many number of frameworks and tools used, the infrastructure analysis that has been done, the complex problems solved and the design of a software architecture consisting of both new and existing systems and ultimately the creation of a test strategy that would improve the quality of the delivered product. Additionally, there was the need to keep an up to date documentation throughout the process, since almost none was available upfront. Therefore, the required level of competences demanded from the software engineering course at NHL Stenden were achieved.

7.1 Reflection

Before starting the graduation, I was so excited to go back to Topicus and particularly to have the same supervisor who guided and followed me so well during my first internship: Martijn Maatman. And because of that, I was certain that my graduation was going to be one of the most challenging, fun and yet joyful experience of my student life. But this blissful situation changed when Martijn had to move to Germany because of personal reasons and therefore my assignment was cancelled since he was the one having the knowledge to guide me through it. Losing the supervisor and the assignment at the same time, right before the start of the graduation, made me very concerned. But thankfully Martijn took the effort to find me a good replacement assignment in another division and another supervisor that would meet the NHL Stenden requirements. I was very happy with the choices he made. Especially about the guidance offered by Emond Papegaaij, who used to be my second supervisor at my time in KeyHub. He never put pressure with my work and he always checked that I enjoyed what I was doing. He also ensured that I had everything that I needed to work in a good environment to perform the best results. I can definitely say that Emond was an excellent supervisor, just like Martijn.




Regarding the assignment, I did not fully understand the complexity at first. But the first meeting made me comprehend that I had to start from week one with the assignment because there were many things that needed to be investigated and many others that I had to learn before I could start. I conducted a research about the infrastructure of Somtoday core and Somtoday Docent and I read the documentation that was available. I also investigated the different schedule importers and how they generally worked. This created a clear picture of the state of the two application and I could make decisions such as the main choice of schedule importer to work with. A big challenge that existed was researching whether detecting schedule changes was feasible and how the importers were implemented within the core application. There was no up to date documentation available and the Somtoday people did not know how these importers worked since the implementation was created ten years ago by one employee that left the company. Patience and perseverance were required in order to succeed this.











Throughout this project I have learned that I am capable of learning new skills quickly, independent of technical complexity, that I am a perfectionist, especially about code quality and that I am able to focus on completing intricate and complex projects perfectly. On an organisational level, I have learned that I prefer an informal environment that appreciates a high level of frankness and critical thinking.













If I could do this assignment again, one thing I would change is the collection of information. I would execute my research in a more academic way. Although the obtained results are valid, more informal research methods could have been used, such as interviews. I would start with assessing the knowledge of people and check if there is relevant documentation, since that would give an overview of the available information.

Overall, I am very satisfied and very proud with my work done here. According to K. Heidrich, many employees before me failed to work with these importers and just detect the schedule changes (February 7, 2019). I have experienced this project as very instructive. There have been many learning moments and a great deal of knowledge has been gained. I am also glad for the opportunity that I was given to work on a real working product used in the Dutch high school sector, instead of doing the usual graduation assignment designed specifically for students.

Bibliography

-  Anderson D., (n.d.). *What is Kanban?* Retrieved on June 12, 2019, from <http://www.everydaykanban.com/what-is-kanban/>
-  Angeli, E., Wagner, J., Lawrick, E., Moore, K., Anderson, M., Soderlund, L., & Brizee, A. (2010, May 5). *General format.* Retrieved on May 9, 2019, from <http://owl.english.purdue.edu/owl/resource/560/01/>
-  Angular. (n.d.). Retrieved June 12, 2019, from <https://docs.angularjs.org/guide/introduction>
-  Dan's tools. (n.d.). *Unix timestamp.* Retrieved on June 12, 2019, from <https://www.unixtimestamp.com/>
-  Davis, M. (2017, February 13). *JBoss vs. Tomcat: Choosing A Java Application Server.* Retrieved on June 11, 2019, from <https://www.futurehosting.com/blog/jboss-vs-tomcat-choosing-a-java-application-server/>
-  D.J. Rumsey. (2010). *Statistics for dummies.* Retrieved on June 12, 2019. Indianapolis Wiley Publishing, Inc.
-  ECollege, T. F. (n.d.). What Is REST? Retrieved on June 11, 2019, from <http://www.restapitutorial.com/lessons/whatisrest.html>
-  GitHub Help. (n.d.). *GitHub glossary.* Retrieved on May 6, 2019, from <https://help.github.com/articles/github-glossary/>
-  GraphQL. (n.d.). *GraphQL: A query language for APIs.* Retrieved on May 6, 2019, from <https://graphql.org/>
-  Gupta, K. (2016, November 9). *Introduction to Wildfly.* Retrieved on Retrieved on June 11, 2019, from www.tothenew.com/blog/introduction-to-wildfly/.
-  Hibernate ORM. (n.d.). *What is Object relational mapping?* Retrieved on May 6, 2019 from <https://hibernate.org/orm/what-is-an-orm/>
-  Idserda, J. (2016, April 7). *Iridium - README.md.* Retrieved on June 10, 2019, from <https://github.com/topicusonderwijs/iridium>

-  Institute of Business Management. (2013, August 09). *Knowledge Management framework of Hansen, Earl and Alvesson*. Retrieved on May 6, 2019 from <https://www.slideshare.net/GaurangaMohanta/knowledge-m-framework-hansen-earlalvessoncm>
-  Java EE (n.d.). *Java™ EE at a Glance*. Retrieved on June 11, 2019, from <http://www.oracle.com/technetwork/java/javaee/overview/index.html>
-  JsWiz. (2016, June 22). *What is Angular*. Retrieved on June 11, 2019, from <https://www.youtube.com/watch?v=WAZTZUgeLhQ>
-  Kupers, T. (2019, April 11). *Dingen die je moet weten als techneut*. Retrieved on June 10, 2019, from <https://confluence.topicus.nl/display/SOM/Dingen+die+je+moet+weten+als+techneut>
-  MuleSoft Videos. (2015, June 19). *What is an API?* Retrieved on June 10, 2019, from https://www.youtube.com/watch?time_continue=59&v=s7wmiS2mSXY
-  Openshift tutorials. (2019, January 11). *What is WildFly?* Retrieved on June 10, 2019, from <http://www.mastertheboss.com/other/faqs/what-is-wildfly>
-  Oracle. (2019, June 4). *Oracle Technology Global Price List*. Retrieved on June 10, 2019, from <https://www.oracle.com/assets/technology-price-list-070617.pdf>
-  Pingler. (2012). *What are UNIX Timestamps and Why are They Useful?* Retrieved on May 6, 2019, from <https://pingler.com/blog/what-are-unix-timestamps-and-why-are-they-useful/>
-  Research Methodology. (n.d.). *Data Collection Methods - Research-Methodology* Retrieved on May 6, 2019 from <https://research-methodology.net/research-methods/data-collection/>
-  Schwaber K., Sutherland J. (2019, April 01). *Scrum Glossary*. Retrieved on May 9, 2019, from <https://www.scrum.org/scrum-glossary>
-  Schwaber K., Sutherland J. (2019, April 01). *The Scrum Guide*. Retrieved on May 9, 2019, from <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>
-  Sobers, R. (2012). *Introduction to OAuth (in Plain English)*. Retrieved on May 6, 2019 from <https://blog.varonis.com/introduction-to-oauth/>

-  Survey Monkey. (n.d.) *SurveyMonkey Reviews: Overview, Pricing and Features*. Retrieved on June 11, 2019, from <https://reviews.financesonline.com/p/surveymonkey/>
-  Taiga. (n.d.). *Taiga Blog*. Retrieved on June 10, 2019, from <https://blog.taiga.io/>
-  Techopedia, (2011, July 31). *What is a software framework?* Retrieved on June 10, 2019, from <https://www.techopedia.com/definition/14384/software-framework>
-  Techopedia, (2017, June 07). *What is an Application Programming Interface (API)? - Definition from Techopedia*. Retrieved on June 10, 2019, from <https://www.techopedia.com/definition/24407/application-programming-interface-api>
-  Timmerman, M. (2018, May 16). *Architectuur*. Retrieved on June 11, 2019, from <https://confluence.topicus.nl/display/SDT/Architectuur>
-  Toll, T. V. (2018, April 01). *What is Angular?* Retrieved on June 11, 2019, from <https://developer.telerik.com/topics/web-development/what-is-angular/>
-  Topicus Onderwijs B.V. (2019, June 2). *Cobra in Action*. Retrieved on June 10, 2019, from <https://cobra-in-action.topicusonderwijs.nl>
-  Vue Mastery. (2017, May 11). *What is Angular?* Retrieved on June 11, 2019, from <https://www.youtube.com/watch?v=8sC55WKzJW4>
-  W3org. (n.d). Hypertext Transfer Protocol -- HTTP/1.1. Retrieved on May 9, 2019, from <https://www.w3schools.com/css/>
-  Wicket. (n.d). *Learn guide of Wicket*. Retrieved on June 12, 2019, from <https://wicket.apache.org/learn/#guide>
-  Wildfly. (n.d). *Wildfly docs*. Retrieved on June 10, 2019, from <https://docs.wildfly.org/16/>
-  Zermelo. (n.d). *Appointment - Developers*. Retrieved on June 12, 2019, from <https://confluence.zermelo.nl/pages/viewpage.action?pageId=15860217>

List of Tables

Table 1: RoosterWijzigingNotificatieActie properties.....	34
Table 2: Endpoint filters.....	36

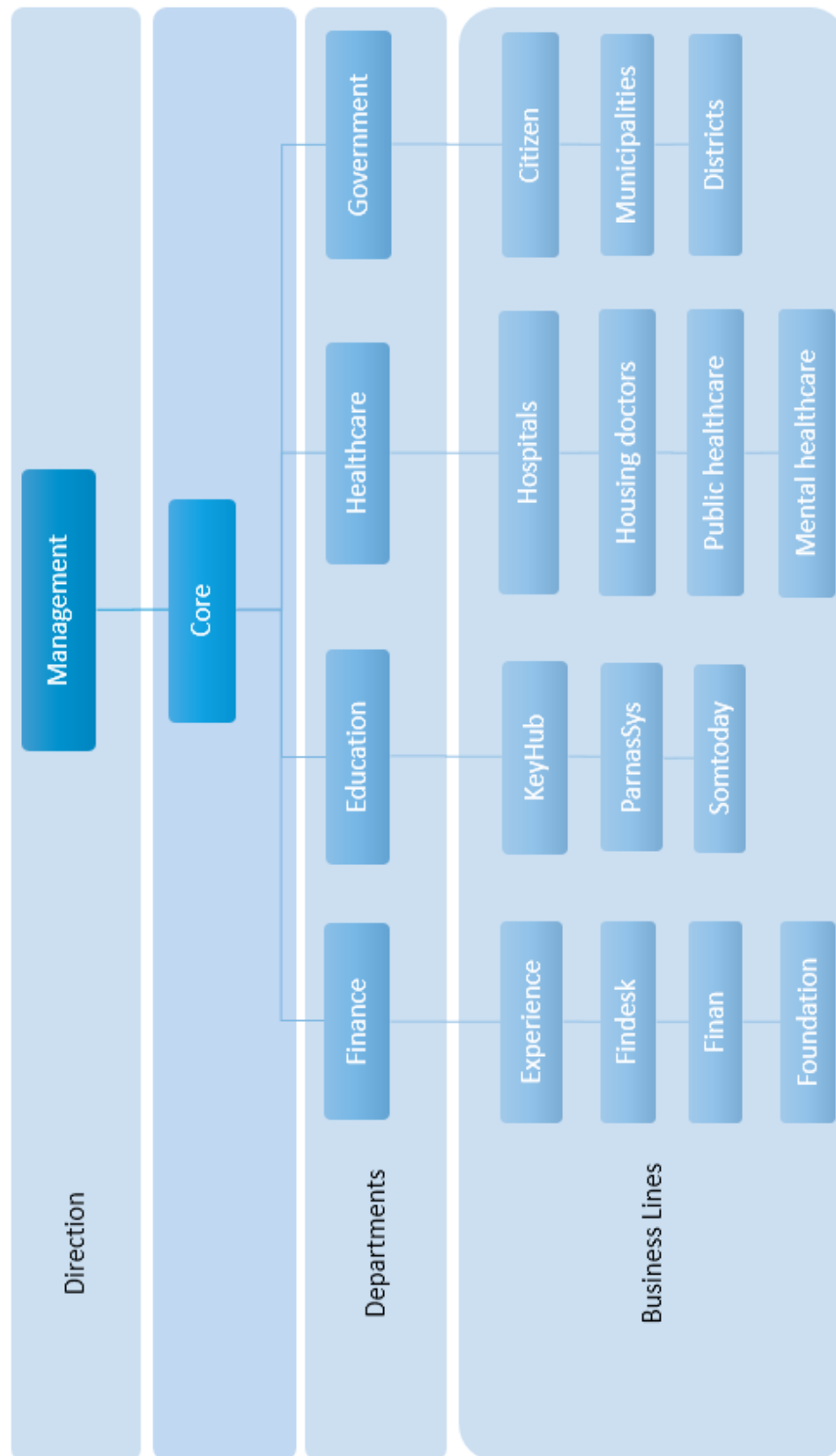
List of Figures

Figure 1: Somtoday Docent infrastructure (Timmerman, 2018)	5
Figure 2: Somtoday Docent components diagram regarding schedules	6
Figure 3: Component structure visually displayed in schedule	7
Figure 4: Somtoday core infrastructure	8
Figure 5: Cobra folder structure	9
Figure 6: DAO example	9
Figure 7: Iridium project structure	11
Figure 8: Common folder content	11
Figure 9: Core folder content	12
Figure 10: Webservices folder content	13
Figure 11: Save appointment example	13
Figure 12: Parameters for new appointment	15
Figure 13: Response from Zermelo after creating new appointment	16
Figure 14: Parameters for updating existing appointment	16
Figure 15: Response from Zermelo after updating an existing appointment	17
Figure 16: Zermelo import screen – Webservice	18
Figure 17: Communication flow between Somtoday Docent and Somtoday (Timmerman, 2018)	19
Figure 18: GraphQL folder content	20
Figure 19: Relevant files for retrieving appointments in Somtoday Docent	21
Figure 20: Most important schedule changes	23
Figure 21: Notifications selection	24
Figure 22: Review of notifications later in the day	24
Figure 23: Notification display	25
Figure 24: Notification receiver	25
Figure 25: Notifications time	26
Figure 26: Scrum Framework (Schwaber & Sutherland, 2019)	27
Figure 27: Product Backlog	29
Figure 28: Sprint Backlog	30
Figure 29: Burndown Chart	30
Figure 30: Taiga dashboard	31
Figure 31: Final selected infrastructure	33
Figure 32: Wireframe schedule changes	37
Figure 33: Saving test notification to database	38
Figure 34: Saved test notifications in database	39
Figure 35: notification endpoint response	40
Figure 36: notifications shown in console	41
Figure 37: notifications shown in console	42
Figure 38: Wireframe new appointment	42
Figure 39: New appointments in schedule	43
Figure 40: Wireframe updated appointment	43
Figure 41: Updated appointments in schedule	43

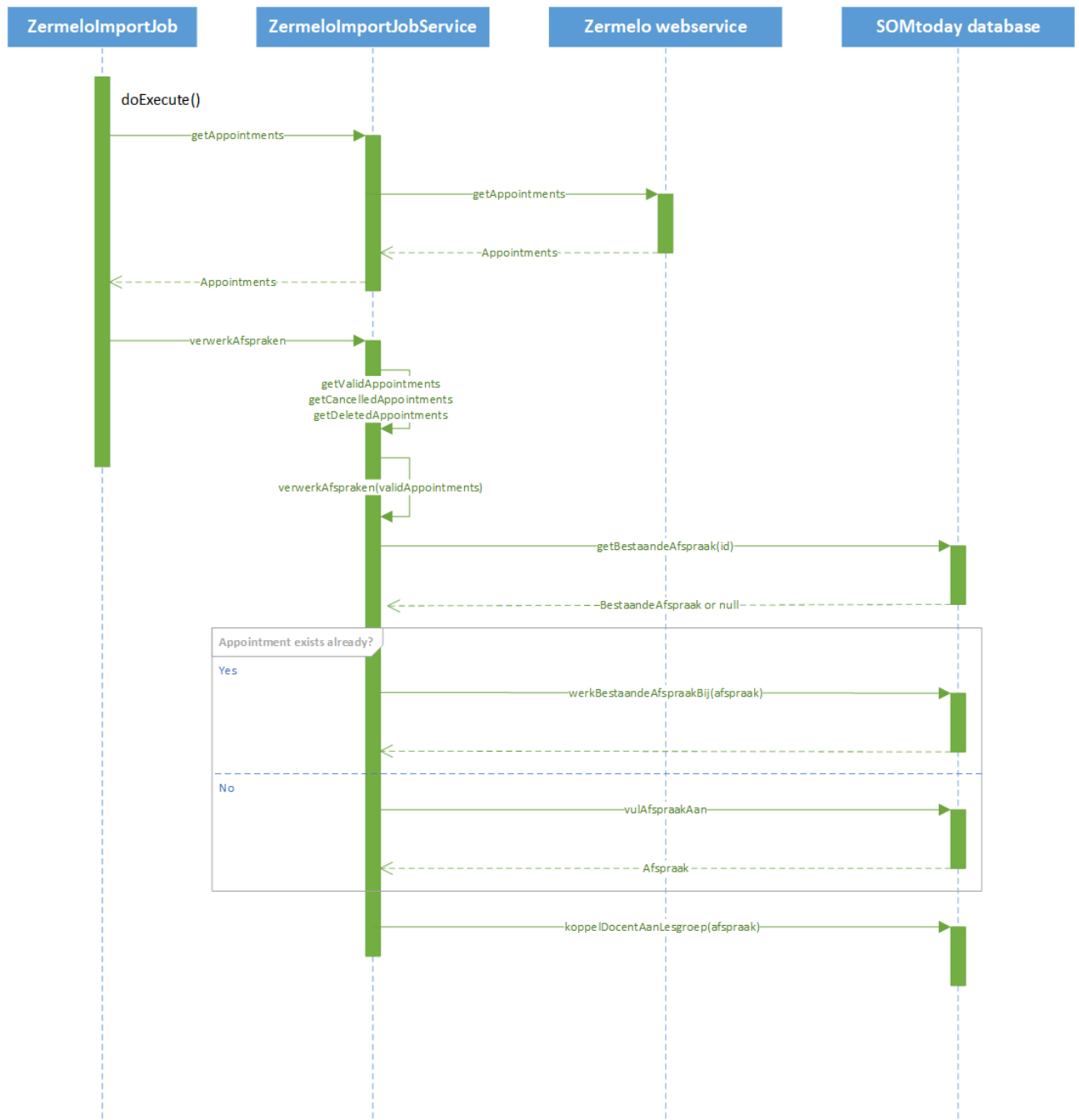
Appendices Table of Contents

Appendix A: Topicus structure.....	1
Appendix B: Zermelo Importer sequence diagram.....	2
Appendix C: Questionnaire results.....	3
Appendix D: Product Backlog.....	6
Appendix E: Sprint Backlogs and relative Burndown charts.....	7
Appendix F: Burndown chart.....	9
Appendix G: Infrastructure ideas.....	10
Appendix H: Class design.....	12
Appendix I: Wireframes of Schedule changes.....	14
Appendix J: Notification creation process.....	16
Appendix K: Notification update process.....	17
Appendix L: Test Strategy.....	18
Appendix M: Graduation assignment description.....	19
Appendix N: Action Plan.....	20
Appendix O: Original Planning.....	21
Appendix P: System requirements specification.....	22
Appendix P: Research set up.....	22

Appendix A: Topicus Structure



Appendix B: Zermelo Importer sequence diagram

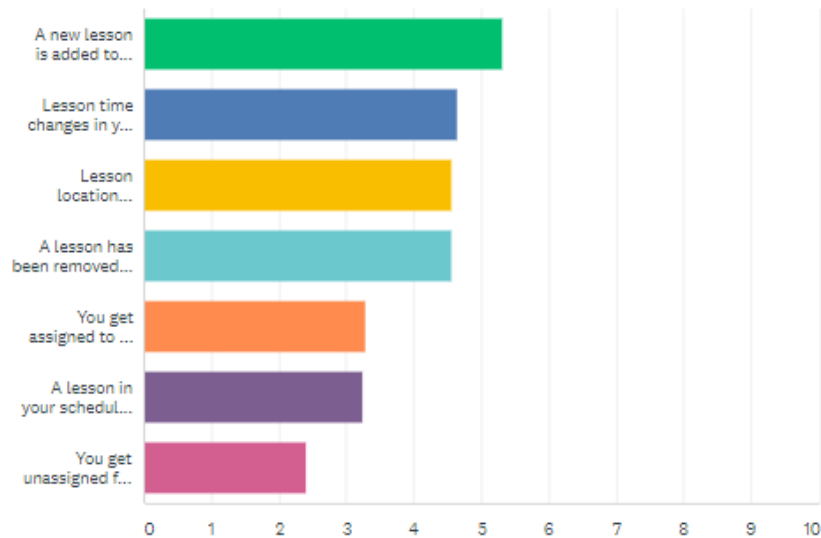


Appendix C: Questionnaire results

Question 1

What are the most important schedule changes that you wish to be notified about?(Rank them from 1 = most important to 7 = least important)

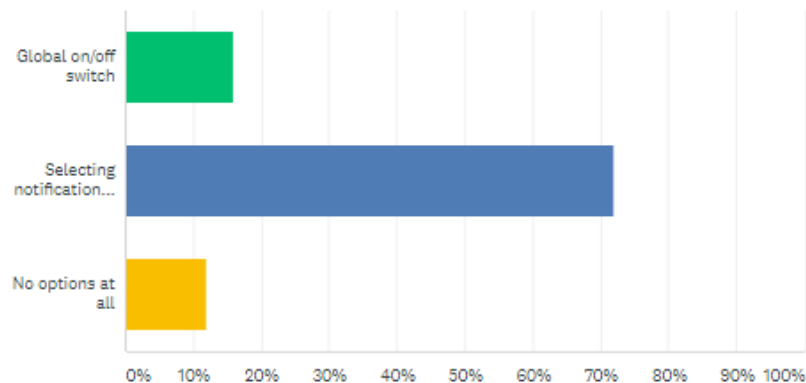
Answered: 25 Skipped: 0



Question 2

How would you like to manage your notifications?

Answered: 25 Skipped: 0



ANSWER CHOICES

Global on/off switch

Selecting notifications per category

No options at all

TOTAL

RESPONSES

16.00%

72.00%

12.00%

4

18

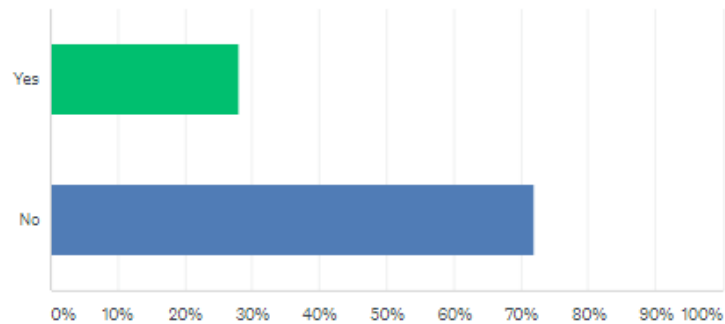
3

25

Question 3

Would you like to review notifications at a later date?

Answered: 25 Skipped: 0

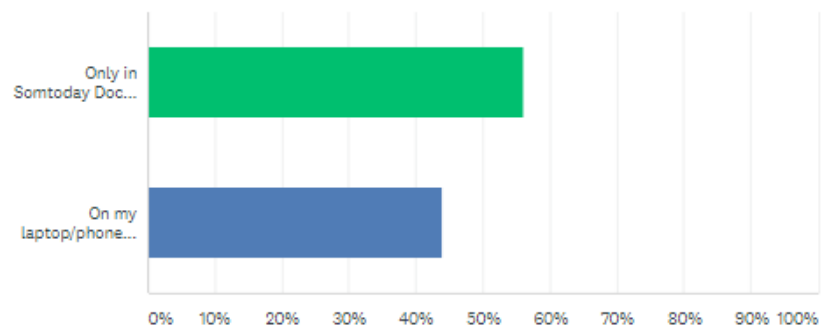


ANSWER CHOICES	RESPONSES	
▼ Yes	28.00%	7
▼ No	72.00%	18
TOTAL		25

Question 4

Where would you like to receive your notifications?

Answered: 25 Skipped: 0

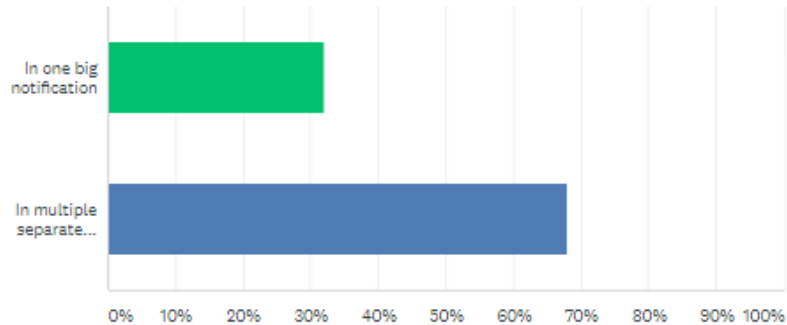


ANSWER CHOICES	RESPONSES	
▼ Only in Somtoday Docent	56.00%	14
▼ On my laptop/phone/tablet even if the application is closed	44.00%	11
TOTAL		25

Question 5

In which way would you like to see your schedule changes?

Answered: 25 Skipped: 0

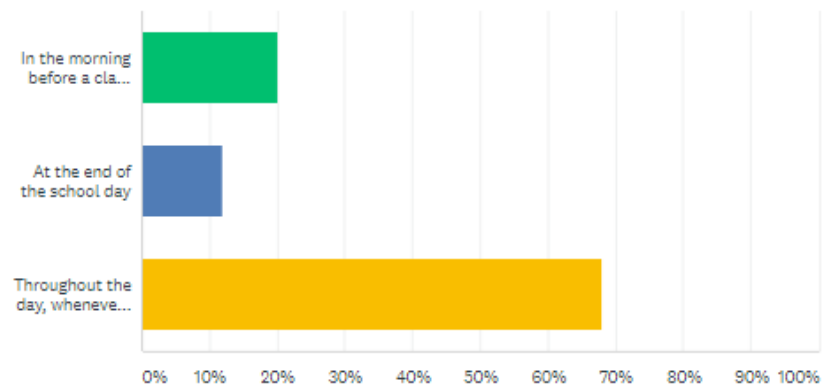


ANSWER CHOICES	RESPONSES	
▼ In one big notification	32.00%	8
▼ In multiple separate notifications	68.00%	17
TOTAL		25

Question 6

When do you wish to get the notifications?

Answered: 25 Skipped: 0



ANSWER CHOICES	RESPONSES	
▼ In the morning before a class starts	20.00%	5
▼ At the end of the school day	12.00%	3
▼ Throughout the day, whenever there is a change happening	68.00%	17
TOTAL		25

Appendix D: Product Backlog

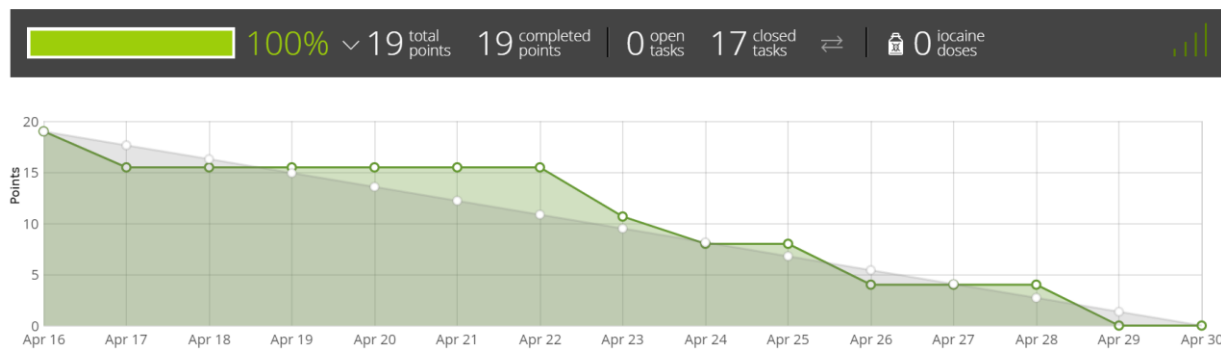
ACTIVITY NUMBER	USER STORY	POINTS
1	As a developer I want to design a general architecture of the notification system, so that the stakeholders have documentation of the system	3
2	As a developer I want to track schedule changes, so that I can know what has been updated in my schedule	8
3	As a developer I want to track which properties of an appointment have been modified, so that I can notify the teachers	8
4	As a developer, I want to save a test notification in the DB so that I can check if it works as designed in the architecture	8
5	As a developer, I want to send a test notification to Somtoday Docent, so that a teacher can see the change in the portal	13
6	As a developer, I want to create the wireframes of changes in the schedule, so that I can have an overview of what the program is going to look like	3
7	As a teacher, I want to receive a notification whenever a class gets added to my schedule, so that I know which groups I have to prepare for	5
8	As a developer, I want to send a test notification to Somtoday Docent, so that a teacher can see the change in the portal - Part 2	13
9	As a teacher, I want to receive a notification whenever the time of a class in my schedule changes, so that I can come on time	8
10	As a teacher, I want to recognise the changes in the schedule, so that I can easily see the differences	13
11	As a teacher, I want to receive a notification whenever the location of a class in my schedule changes, so that I know where I have to be present	n.d.
12	As a teacher, I want to receive a notification whenever a class gets deleted from my schedule, so that I know which groups and classes I do not have to prepare for	n.d.
13	As a developer, I want to deploy my project to a test server, so that I can check the online availability via web browser	n.d.

Appendix E: Sprint Backlogs and relative Burndown charts

Sprint 1

ACTIVITY NUMBER	USER STORY	POINTS
1	As a developer I want to design a general architecture of the notification system, so that the stakeholders have documentation of the system	3
2	As a developer I want to track schedule changes, so that I can know what has been updated in my schedule	8
3	As a developer I want to track which properties of an appointment have been modified, so that I can notify the teachers	8

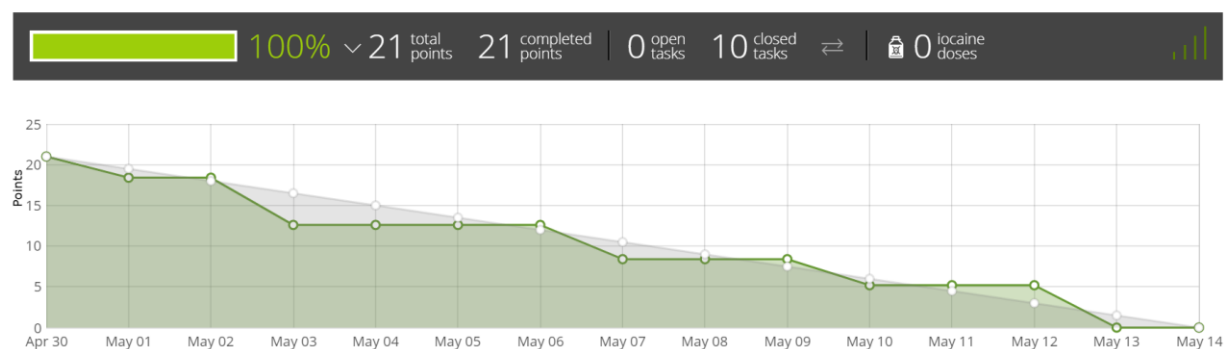
SOMTODAY DOCENT SCH... SPRINT 1 16 APR 2019-30 APR 2019



Sprint 2

ACTIVITY NUMBER	USER STORY	POINTS
4	As a developer, I want to save a test notification in the DB so that I can check if it works as designed in the architecture	8
5	As a developer, I want to send a test notification to Somtoday Docent, so that a teacher can see the change in the portal	13

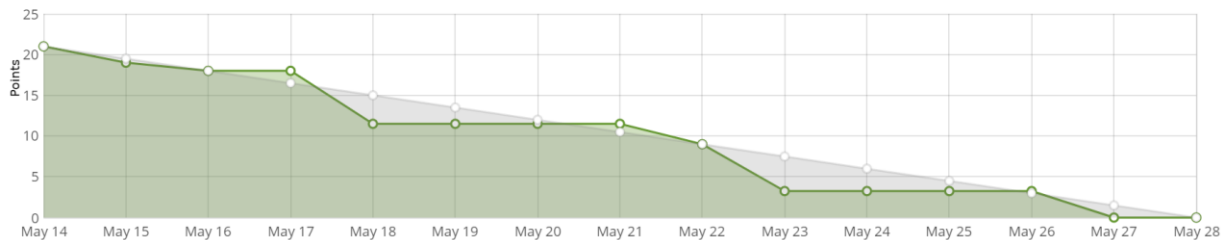
SOMTODAY DOCENT SCH... SPRINT 2 30 APR 2019-14 MAY 2019



Sprint 3

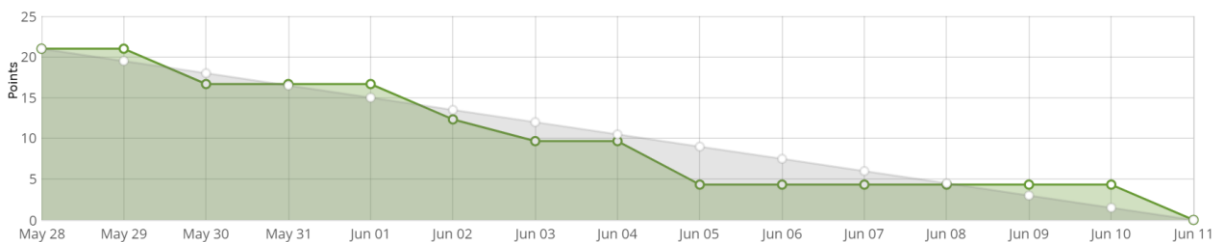
ACTIVITY NUMBER	USER STORY	POINTS
6	As a developer, I want to create the wireframes of changes in the schedule, so that I can have an overview of what the program is going to look like	3
7	As a teacher, I want to receive a notification whenever a class gets added to my schedule, so that I know which groups I have to prepare for	8
8	As a developer, I want to send a test notification to Somtoday Docent, so that a teacher can see the change in the portal - Part 2	8

SOMTODAY DOCENT SCH... SPRINT 3 14 MAY 2019-28 MAY 2019

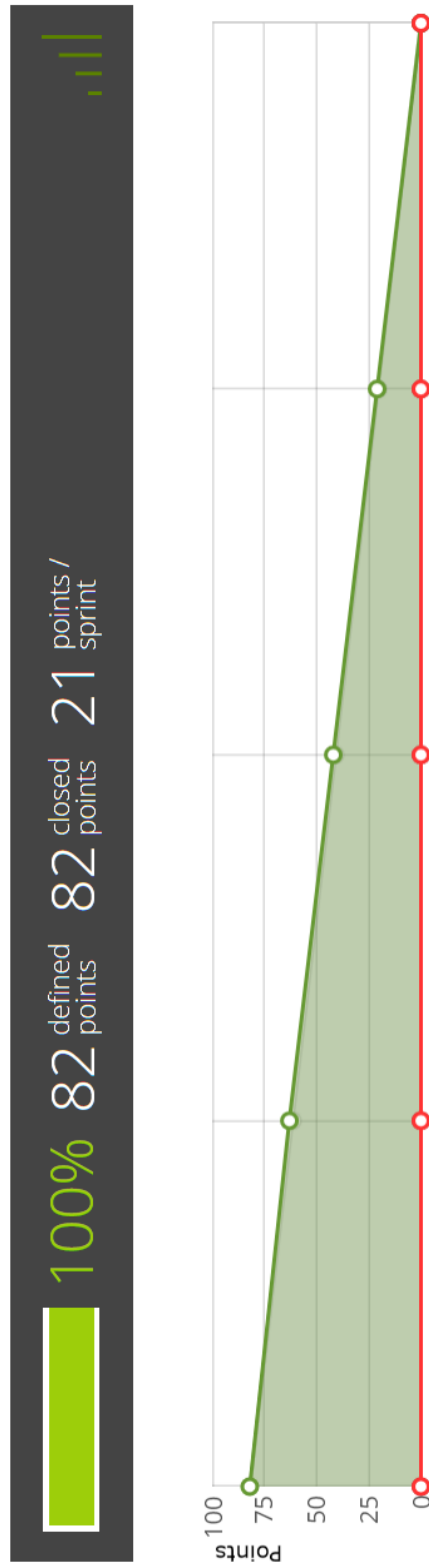
**Sprint 4**

ACTIVITY NUMBER	USER STORY	POINTS
9	As a teacher, I want to receive a notification whenever the time of a class in my schedule changes, so that I can come on time	8
10	As a teacher, I want to recognise the changes in the schedule, so that I can easily see the differences	13

SOMTODAY DOCENT SCH... SPRINT 4 28 MAY 2019-11 JUN 2019

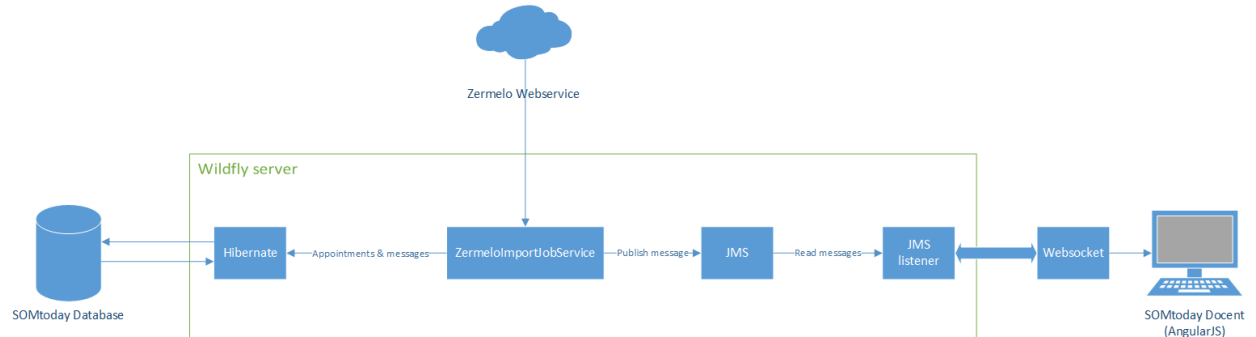


Appendix F: Burndown chart

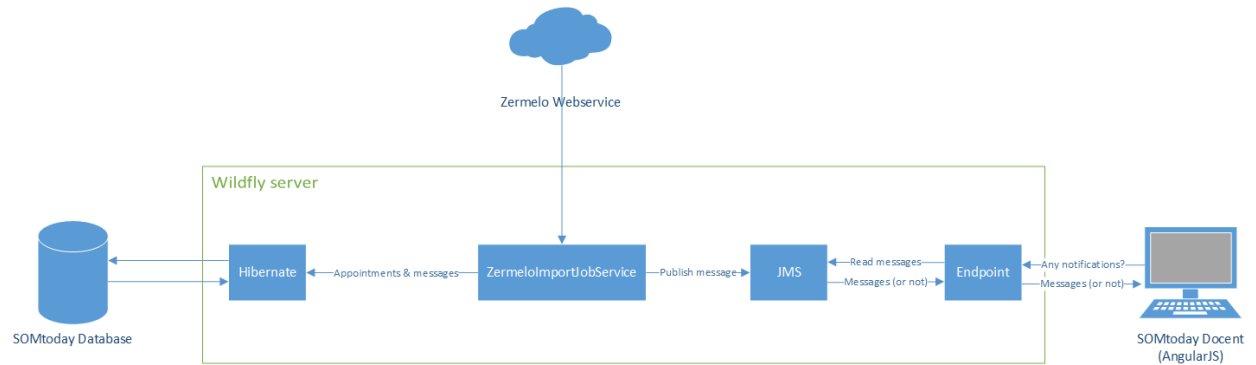


Appendix G: Infrastructure ideas

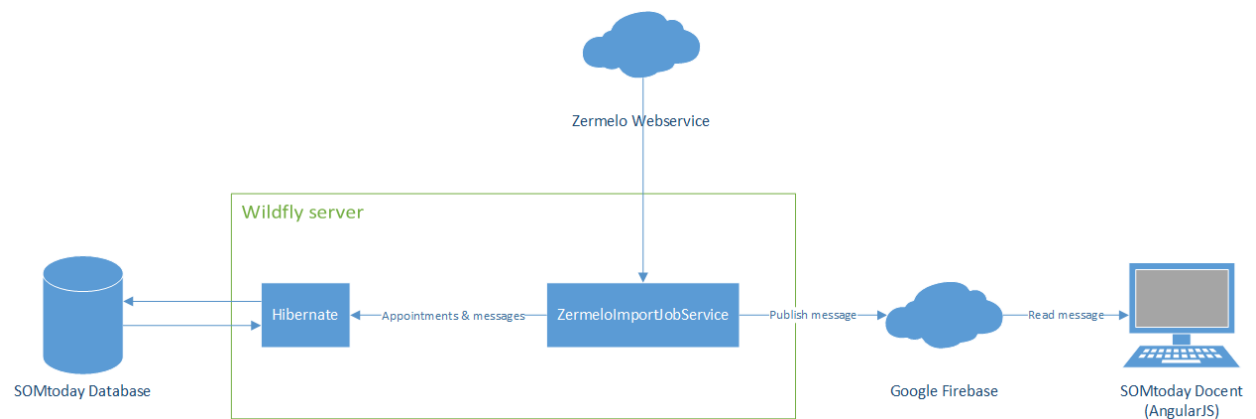
Option 1a



Option 1b



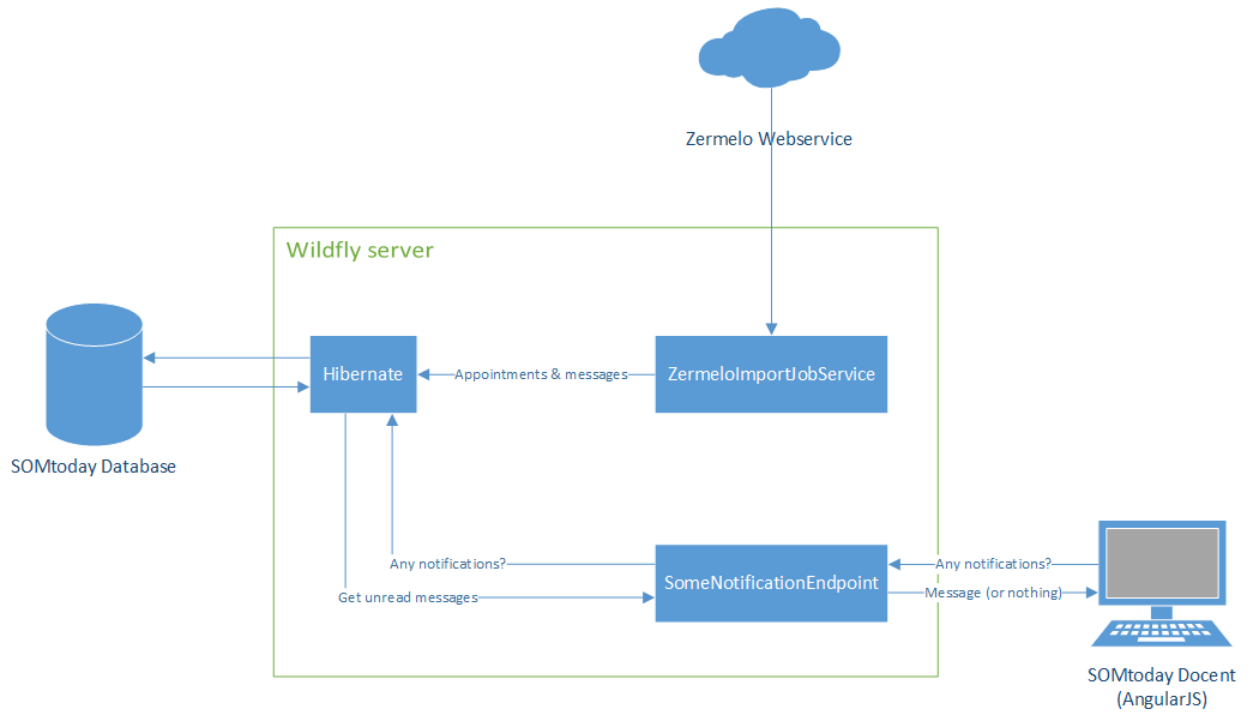
Option 2



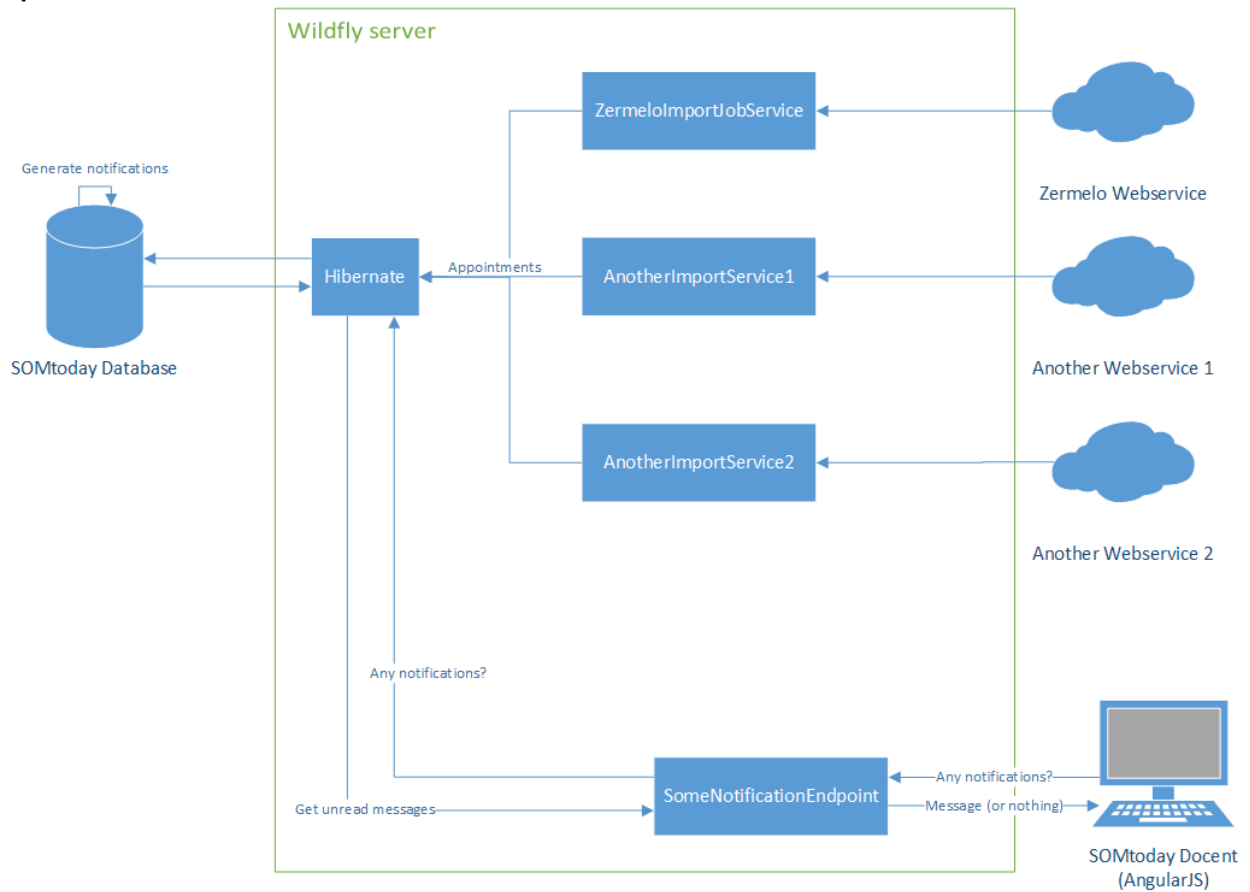
Michela A. Karunakalage

1475319

Option 3



Option 4



Appendix H: Class design

RoosterWijzigingNotificatieActie

Extends from the abstract class `NotificatieActie`. This subclass is meant for notifications about schedule changes. It contains its own fields regarding notification messaging. The `RoosterWijzigingNotificatieActie` class itself will contain fields regarding the schedule changes only.

Existing fields

Before creating brand new fields, I investigated which already existing fields I can use for the notification. These are the inherited fields from the superclasses. Every field contains a description that tells for what kind of purpose they are going to be used for:

Entiteit.java

Name property	Type	Description
<code>dtype</code>	String	The discriminator value that is required to set (in this case it will be: <code>@DiscriminatorValue(value = "RoosterWijzigingNotificatieActie")</code>)
<code>instelling</code>	Instelling	Contains the object representation of the institution. Will be set because it is required.

Actie.java

Name property	Type	Description
<code>actieType</code>	enum	Defined which action has been performed to create this notification. The options are: CREATIE = appointment has been created WIJZIGING = appointment has been updated VERWIJDERING = appointment has been deleted

Boodschap.java

Name property	Type	Description
<code>verzendDateum</code>	ZonedDateTime	Used to save the date when the notification was sent.
<code>onderwerp</code>	String	The subject of the notification (title)
<code>inhoud</code>	String	The content of the notification (body)

Michela A. Karunakalage

















1475319

New fields

Name property	Type	Description
medewerker	Medewerker	A link to the teacher that the notification is sent to
oudeBeginDatumTijd	Date	The old start time and date of the appointment (if the time or date is updated, else NULL)
oudeEindDatumTijd	Date	The old end time and date of the appointment (if the time or date is updated, else NULL)
zermeloId	Long	A reference to the version ID that is given by Zermelo. This way, the latest change can be retrieved.

Appendix I: Wireframes of Schedule changes

Total overview schedule changes

Rooster                

Michela A. Karunakalage

1475319

New appointment

❖ Roosteritem-created

09:00
09:50
2e

3H3Eco
a29
nieuw

Time changed appointment

❖ Roosteritem-time-changed

09:00
09:50
3e

3H3Eco
a29

Location changed appointment

❖ Roosteritem-verplaatst

10:00
10:40
3e

2T3Eco
a29
verplaatst

Deleted appointment

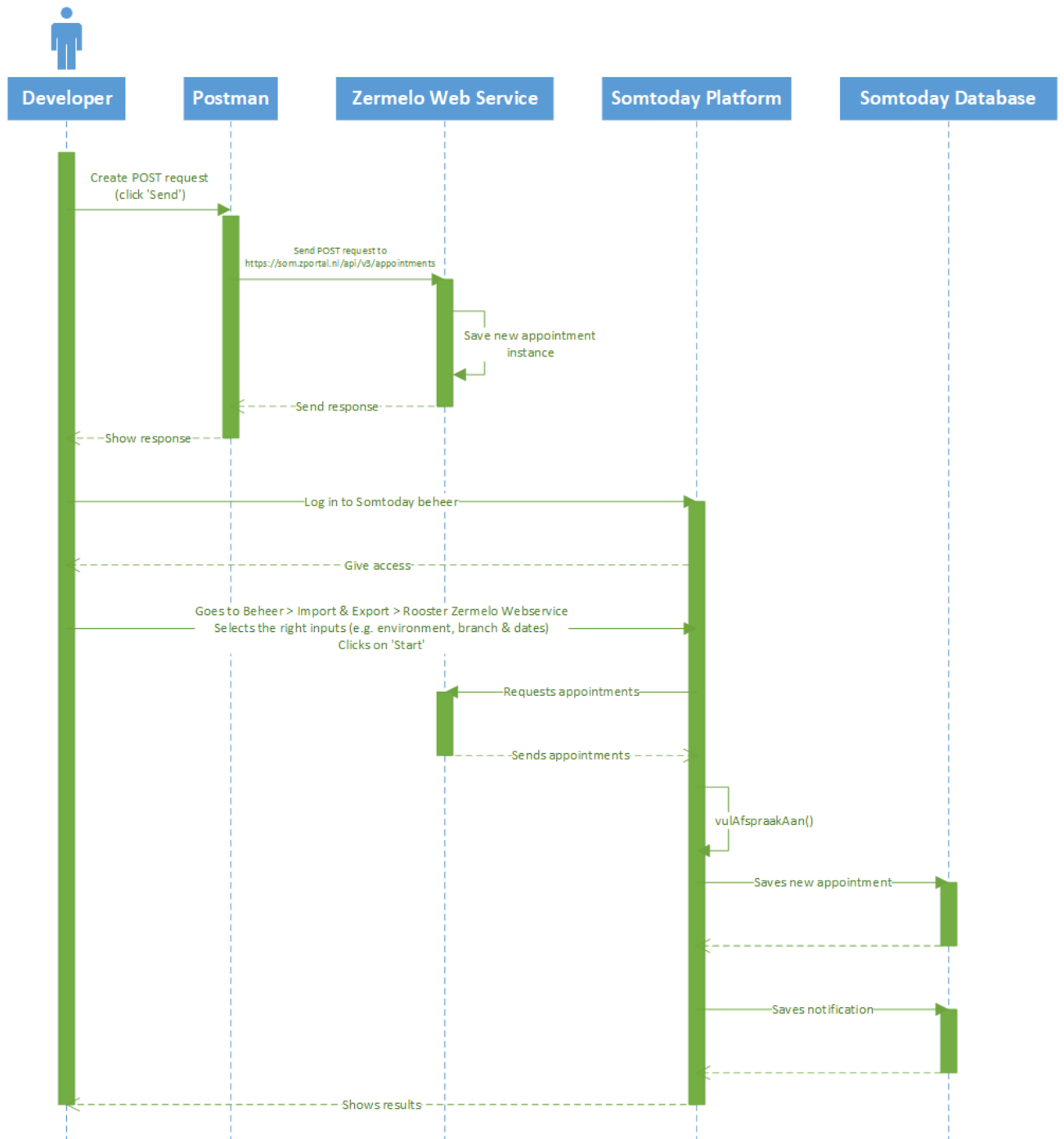
❖ Roosteritem-deleted

13:00
13:50
6e

1H1Eco
a29
uitgevallen

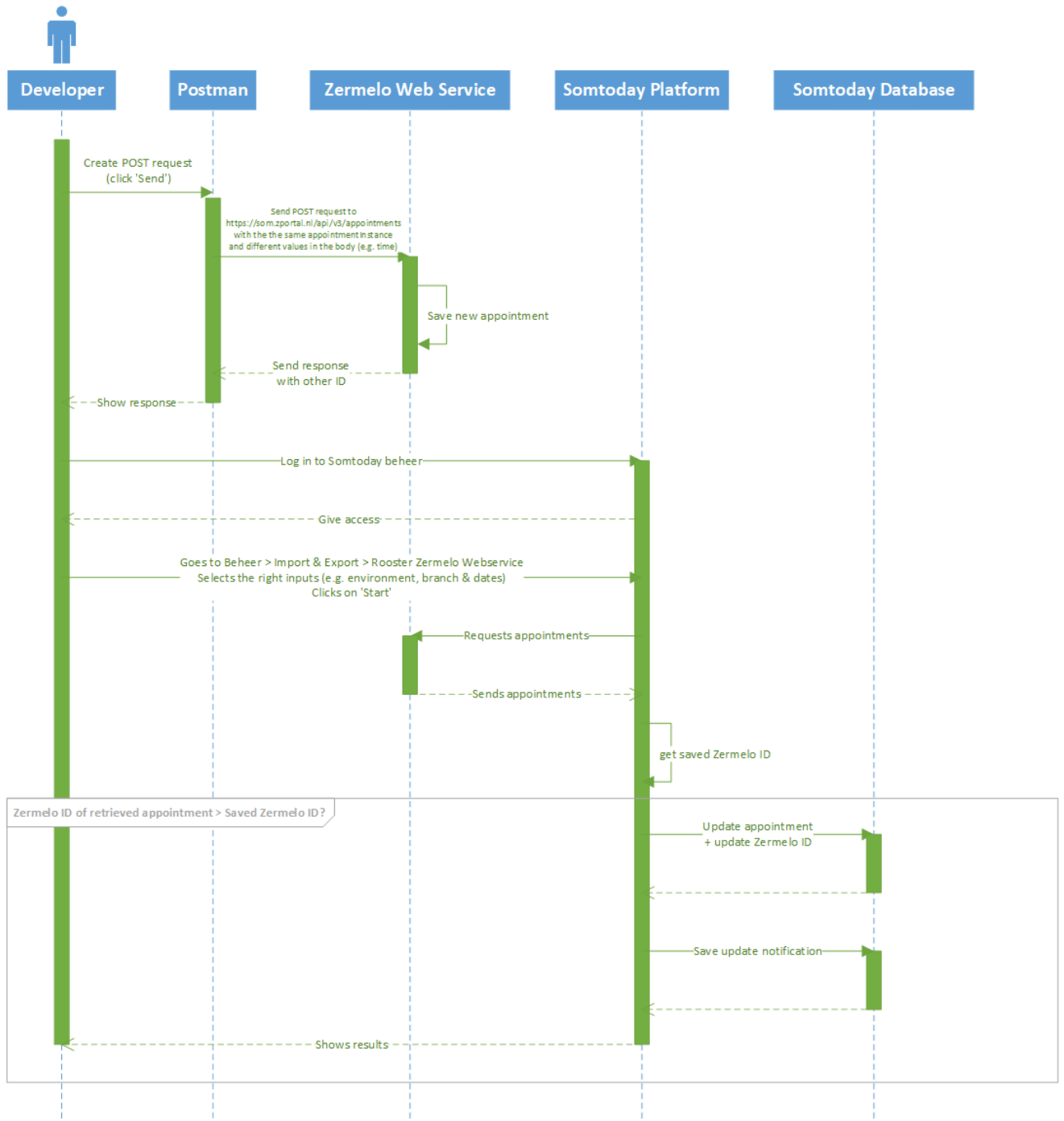
Appendix j: Notification creation process

Creating new appointment



Appendix K: Notification update process

Updating existing appointment



Michela A. Karunakalage

1475319

Appendix L: Test Strategy

See document 'Test strategy'

Michela A. Karunakalage

1475319

Appendix M: Graduation assignment description

See document 'Graduation Assignment description'

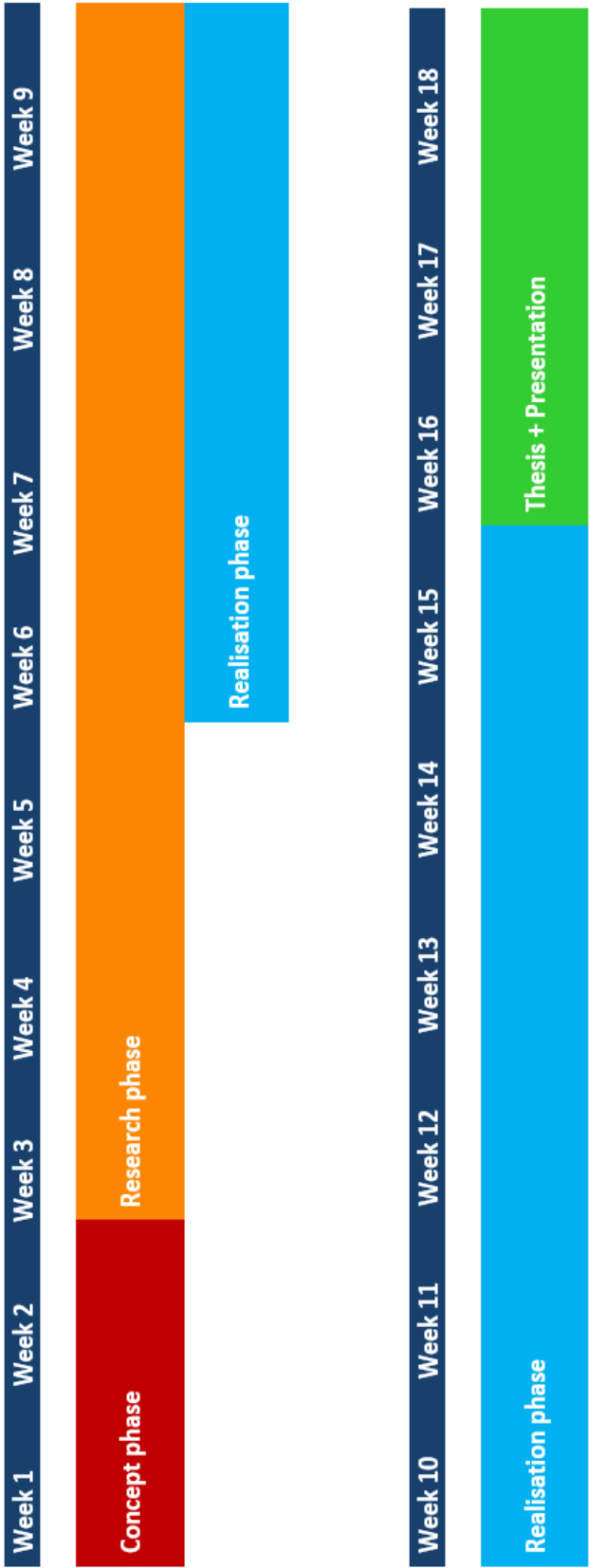
Michela A. Karunakalage

1475319

Appendix N: Action Plan

See document 'Action Plan'

Appendix O: Original Planning



Concept phase: 80 hrs
Research phase: 120 hrs
Realisation phase: 320 hrs
Thesis + presentation: 200 hrs
Total: 720 hrs = 90 days (30 EC)

Appendix P: System requirements specification

This document provides an overview of all the functionalities required from the client, stakeholders and the end users. The purpose of this SRS is to explain which functionalities are classified as essential to complete the Minimum viable product (MVP) and which are categorised as 'extra features' based on the importance for the client and the end users, the feasibility of the tasks and the time constraints.

MVP

The Minimum viable product was realised when creating the Action Plan. It contains all the features that are needed to make this Project successful. These features are:

- Analysis of the current scheduling importers;
- Defining a routine to determine changes between an old and a new schedule;
- Set up of a questionnaire on the different ways' teachers want to receive notifications and which possible settings are required;
- Design of a prototype of the notifications.

Extra features

The following features are all improvements for the new built module:

- Proof of concept on the determination of actual schedule changes;
- Corporate design;
- Visualisation of new lessons added in the schedule of Somtoday Docent;
- Visualisation of time changed lessons in the schedule of Somtoday Docent;
- Visualisation of location changed lessons in the schedule of Somtoday Docent;
- Visualisation of deleted lesson in the schedule of Somtoday Docent;
- Implementation into the Somtoday Docent live environment;
- 'Seen' button for visualised notifications;
- Selection of notifications based on the category;

Michela A. Karunakalage

1475319

Appendix Q: Research set up

See document 'Research set up'