Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Developing a digital signage solution for office attendance

Christoph Goller

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Jinghua Yu, Cornelius Rusch
Commenced:	17.08.2020
Completed:	16.02.2020

Abstract

In an increasingly connected world, for a growing amount of jobs remote work is a valid alternative to the traditional office work. This can provide new opportunities as well as challenges. While it can give employees the possibility to achieve a better work-life balance, it can also lead to confusion at work as it may be unclear where co-workers are currently located. This was the case at the company evia. However, as the company wanted to maintain its flexible working model with all its benefits, a solution for this problem had to be developed.

It was decided to develop a system that allows employees to enter whether they're working remotely or in the office. The collected information should be displayed on digital signs in the office rooms of the company. Digital signs are centrally managed computer screens which can display dynamic content to their audience. However, in order to increase usability, additional ways to display the data should be explored.

In the context of this thesis this software system was developed. This included defining, designing, developing, validating and verifying the system. The overall goal of this thesis was to produce a working prototype as a proof of concept.

In addition to digital signs the system was able to display data within the Microsoft Teams app and in a browser, after the implementing phase. The user can input his data through a Microsoft Teams conversational bot, e-mail and the web app.

Overall the Project took a good course and produced a working system that got adopted by the company.

Contents

1 Introduction **15**
- 1.1 Introducing evia . . . 15
- 1.2 Context and Problem Description . . . 15

2 Methodology **17**
- 2.1 SDLC Models . . . 17
- 2.2 Waterfall Model . . . 17

3 Requirements **21**
- 3.1 Analysis . . . 21
- 3.2 Functional Requirements . . . 23
- 3.3 Non-Functional Requirements . . . 28
- 3.4 Summary . . . 29

4 Design **31**
- 4.1 Basics . . . 32
- 4.2 Server Application . . . 32
- 4.3 Single-Page Client Application . . . 38
- 4.4 PiSignage . . . 39
- 4.5 MS Teams App . . . 40

5 Implementation **41**
- 5.1 Development Environment . . . 41
- 5.2 Server Application . . . 43
- 5.3 Single-Page Client Application . . . 47
- 5.4 PiSignage . . . 57
- 5.5 MS Teams App . . . 57
- 5.6 Results . . . 58

6 Verification and Validation **61**
- 6.1 Verification . . . 61
- 6.2 Validation . . . 62

7 Conclusion **63**
- 7.1 Challenges . . . 63
- 7.2 Future Work . . . 64

Bibliography **67**

List of Figures

2.1 The traditional waterfall approach. 18

3.1 Survey results of question 1. 23
3.2 Survey results of question 2. 23
3.3 Example of a custom MS Teams tab 27

4.1 High-level architecture diagram. 31
4.2 Modular monolithic structure of the server application. 34
4.3 MongoDB data model. 35

5.1 CI/CD flow. 42
5.2 Sequence chart of the communication with the MS Teams bot. 45
5.3 Design draft of desktop view. 48
5.4 Design draft of mobile view. 48
5.5 Component structure of the AUROA Angular application. 50
5.6 GUI of “settings dialog” component 51
5.7 GUI of “personal attendance table” component 51
5.8 GUI of “personal attendance table mobile” component 52
5.9 GUI of “mobile controls” component. 52
5.10 GUI of “attendance table” component 53
5.11 GUI of “controls bar” component 53
5.12 GUI of “table header” component 53
5.13 OAuth Implicit Flow [Micb]. 55
5.14 Example flow in the frontend after user logged in. 57
5.15 Picture of the finished software on multiple output channels. 58
5.16 An example conversation with the MS Teams bot. 59
5.17 Screenshot of the MS Teams app. 59

List of Tables

3.1 Overview of input and output channels categorized by priority. 30
3.2 Overview of requirements categorized by priority. 30

4.1 Overview of HTTP Endpoints. 37
4.2 Overview of HTTP Parameters. 39

5.1 Overview of injected services. 54

List of Listings

5.1 The docker-compose.yml file. 43
5.2 The angular interceptor responsible for authentication. 56

List of Abbreviations

AAD Azure Active Directory - Identity platform by Microsoft. 39, 54, 55

AJAX Asynchronous JavaScript And XML. 38

API Application Programming Interface. 25, 29, 34, 36, 37, 38, 46, 49

BE Back End (code that runs on the server). 32, 33, 34, 46, 49, 54, 55

CI/CD Continous Integration/ Continous Deployment. 41, 42, 44, 63

CLI Command Line Integration. 56, 57

DAO Data Access Model. 44

FE Front End (code that runs on the client machine). 34, 56

GUI Graphical User Interface. 47, 50, 51, 52, 53, 58, 62, 63

HTML Hyper Text Markup Language. 27, 47, 52

HTTP Hypertext Transfer Protocol. 39, 43, 46, 49, 54, 55

IDE Integrated Development Environment. 38

IMAP Internet Message Access Protocol. 45

ISO International Organization for Standardization. 19, 28

JSON JavaScript Object Notation. 24, 35, 40, 46

JWT JSON Web Token. 32, 55

MPA Multi Page Application. 38

NIST National Institute of Standards and Technology. 29

SDLC Software Development Life Cylce. 17, 63

SPA Single Page Application. 37, 38, 39, 41, 43, 44, 47, 50, 54, 55, 63

SRS Software Requirements Specification - A standard by the ISO to ensure the quality of requirement documentations. 19

SSO Single Sign On. 39

URL Uniform Resource Locator. 49, 55, 56

VM Virtual Maschine. 41, 42, 43, 44, 57, 61

VPN Virtual Private Network. 15, 22

1 Introduction

This thesis documents the development of a digital signage solution for office attendance used by the company evia. The first section introduces the company and describes the problem which is to be solved.

1.1 Introducing evia

The evia solutions GmbH is a company based in Germany. At the time of this thesis evia has offices in four different locations, namely: Stuttgart, Leonberg, Munich and Ulm. Distributed over these locations, evia employs around 200 people. The business model of evia is to help other companies digitalize their processes. This includes project management, system development, consulting and software development. The evia solutions GmbH will from now on be referred to as “customer”.

1.2 Context and Problem Description

The concept of remote work, especially home office, already exists since the invention of telecommunication. Its advantages and disadvantages are controversially discussed until today and depend on many factors like company structure and type of work.

In addition to the traditional office work, at the customer home office is a fully established alternative. Since the core business of the customer mainly consists of software development and consulting, it is possible to do most of the required work from a remote workplace. Remote workplaces are as efficient as never before thanks to modern solutions like Virtual Private Network (VPN) and video calls. As a modern employer, the customer aims to enhance the work-life balance of their employees by offering various workplace models. However, due to the extremely flexible working-model, it is difficult to keep track of where somebody is working from. This can lead to unnecessary confusion and creates collaboration between colleagues even more complicated. Due to the positive feedback from their employees the customer does not want to change their flexible home office approach. Thus a new way of monitoring the presence of colleagues is necessary.

Furthermore, the management of the customer has been considering the use of digital signs in its offices for some time. Digital signs can be a valuable addition to a workplace by providing customized and dynamic content.

As a result of the aforementioned situation, the office presence solution was planned to serve as a pilot project for digital signs at the customer.

2 Methodology

Because this thesis documents the development of a software solution, the methodology and structure of this paper is strongly linked to the management of this software development project. The PMBOK™ Guide defines project management as the "application of knowledge, skills, tools and techniques to project activities to meet the project requirements". A proper project management, while not being the only factor that matters, can still have a huge impact on the success of a project [MB96]. In the following paragraphs the most important tools for this specific project will be introduced.

2.1 SDLC Models

Adel Alshamrani and Abdullah Bahattab define that the Software Development Life Cylce (SDLC) consists of all activities dedicated to designing, building or maintaining software [AB15].

R. Bender describes that in the early 1960s, when commercial software started to take off, there was no system behind developing software [Ben03]. When projects became more and more complex in the late 1960s it became clear that the process of developing software had to become more structured [Ben03].

As a consequence, a variety of different SDLC models were developed. According to Apoorva Mishra and Deepty Dubey the most common SDLC models include the Waterfall Model, the Spiral Model, the RAD Model, the Incremental Model and the V-Shaped Model [MD13]. They compare the different models and conclude that the usage of a specific model heavily depends on the environment they are used in [MD13]. Nayan B. Ruparelia comes to a similar conclusion when comparing even more different models [Rup10].

2.2 Waterfall Model

The waterfall model is one of the oldest SDLC models. It was introduced by Winston W. Royce in his paper "Managing the development of large software systems" [Roy70]. While Winston initially introduced it as a flawed example of software development it quickly became a widely used model. This is because of its small overhead compared to many of the other SDLC models. But the waterfall model only is a good fit for very specific software projects. The most important property of a project that shall be managed by using the waterfall model is that the requirements are well defined. More specific that means that they should be documented, stable and not ambiguous [IDC]. This is so critical because all the later stages of the waterfall model build up on these requirements. If the requirements change during these later stages, it is nearly impossible to implement them.

In the specific case of this project the requirements are well defined and it is not that important to get a working version as fast as possible because the absence of the solution produces no actual costs. This situation made it possible to choose the waterfall model because of its ease of implementation and low costs, rather than a more agile but also more complex model.

The waterfall model proposes a sequential process of software development. Figure 2.1 shows the typical steps of the waterfall model: Requirements, Design, Implementation, Verification, Maintenance.

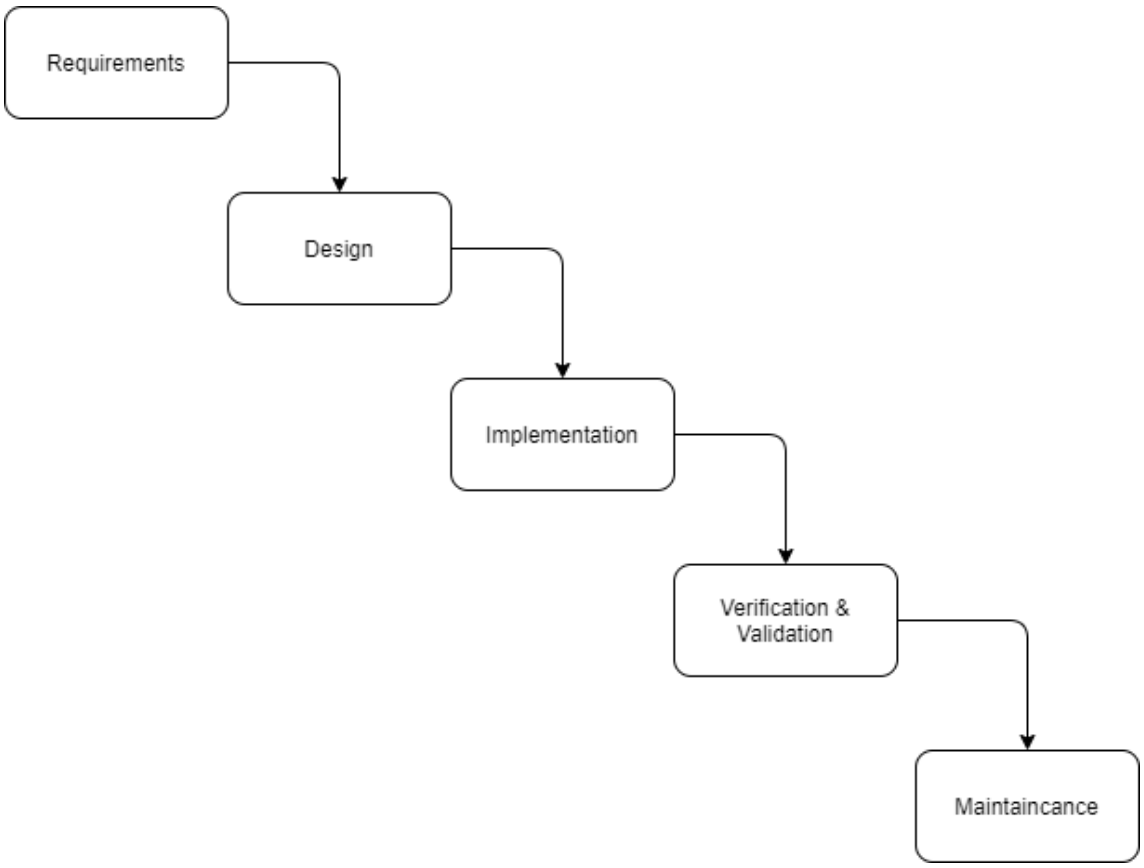


Figure 2.1: The traditional waterfall approach.

2.2.1 Requirements

Requirement Engineering is a critical part of the software development process. A study of the Standish Group in 2010 found out that only 37% of software projects get finished successfully [Sta]. 42% are delivered to late or over the budget and 21% are cancelled [Sta]. As one of the most important factors for projects failing the Standish Group identified requirement engineering with 87% of companies saying that it played a role in their failed projects [Sta]. While the meaningfulness of this report is often contested [EV08][EV06], the importance of well-defined requirements is undisputed.

Requirement Engineering describes the process to ensure that all requirements are known and understood. Furthermore, it also implies that all stakeholders of the software agree on these requirements and that they are documented precisely.

To ensure the quality of the documentation the International Organization for Standardization (ISO) defines the standard Software Requirements Specification - A standard by the ISO to ensure the quality of requirement documentations (SRS). It describes characteristics of good requirement specification like correctness, unambiguousness, completeness and more. Requirements are often divided into two categories: the functional and the non-functional requirements. Functional requirements answer the question "What should the software do" and therefore define a specific behavior. Non-functional requirements on the other hand evaluate the operation of a system. Typical examples for this are availability, maintainability or security.

2.2.2 Design

Software design is the process of planning the implementation of the software solution in a way that it can meet the requirements. It includes all the choices that need to be made before implementing the solution. For example, technologies like the programming language or the operating system, the algorithms or the used patterns. Patterns are reusable blueprints on how to solve problems of a specific kind.

One of the most popular software patterns were introduced by the "Gang of Four", namely: Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides [EV94]. They decided to categorize software patterns into three subtypes [EV94]. Creational patterns are patterns that produce objects in regard to initiating them one by one. Examples for creational patterns are the factory method, the singleton or the prototype [Car]. Structural patterns "provide a manner to define relationships between classes or objects" [Car]. An adapter, a decorator or a facade are examples for this subtype. The third subtype, "behavioural patterns", defines how objects communicate between each other. Specimen of this type are the observer pattern, the state pattern or the strategy pattern [Car].

In addition to software patterns there are also more high-level architectural patterns like Microservices, Event-Driven Architecture or Multitier architecure. There are many other patterns, however mentioning and describing them all would go beyond the scope of this brief overview.

2.2.3 Implementation

Implementation describes the time period in which developers realize the design. In addition to the actual coding, this step also involves testing and installation. Testing during development is important because it can reduce the cost of fixing bugs later on.

A typical challenge of this period is the communication between different developers and teams working on the same codebase. However in this project this is not a problem because there only is one developer.

One popular way of differencing between different test types are glass-box tests and black-box tests. Glass-box tests are tests that are developed on base of the actual code and are often done by the developers themselves. This can lead to false negatives however, because misunderstood requirements also lead to wrong tests. A popular example of this are "Unit Tests". Conversely,

black box tests are developed without knowledge of the codebase and are usually performed by an additional test group. To make testing more efficient it can make sense to develop automated tests. This can include both glass-box tests and black-box tests. It is important to measure the cost to develop automated tests compared to manual testing. Testing in general is very important to find errors in the code before the software gets launched. Sometimes the tests are even written before the code. This is called test-driven development, as explained by Kent Beck [Bec03].

2.2.4 Verification and Validation

While sometimes mistakenly getting treated as synonyms, verification and validation actually describe different processes. In the book "Software Engineering Economics" Boehm explains that you're doing verification if you ask yourself "Are we building the product right?" [Boe81]. However if you ask yourself "Are we building the right product?" you're doing validation [Boe81]. Verification therefore means looking back to the requirements and check if the built product actually meets them. A typical way of doing this is a meeting with the customer. Validation on the other hand checks if the product is what the end user wanted.

In agile software development both of these questions get asked regularly during the development. But when using the waterfall model they are not relevant before the implementation step is finished. Thats why it is so important to do them carefully, as already mentioned in sections 2.2.1 and 2.2.2.

They are still important though, because valuable lessons for future projects can be drawn from them.

2.2.5 Maintenance

Maintenance describes the agreed time period after the software was implemented in which the software will get patches in order to fix bugs. Depending on the agreement between customer and developer, enhancements can also be part of the maintenance phase.

In the paper "Characteristics of Application Software Maintenance" Lientz, Swanson and Tompkins come to the conclusion that maintenance and enhancements consume much of the total resources used on a software project [BP 78]. They also say that some estimations for the percentage of the resources that go into maintenance range from 75% to 80% [BP 78].

It is also worth mentioning that the effort to maintain software can be influenced by the design and the implementation. Clean code and a well documented architecture can reduce the amount of time that needs to be put into maintaining it.

However, due to the limited time of six months, the maintenance period is not part of this thesis.

3 Requirements

In this chapter, an overview of the elicited requirements is given. The first step in requirement engineering is to investigate which stakeholders are involved. After that, use cases can be developed for each stakeholder. For this project three stakeholder groups were identified: The employees, the team leaders and the administrators.

- As an employee I want to be able to put my status into the system and also be able to see the status of my colleagues.
- As a team lead I want to have an overview of where my team members are working from.
- As an administrator I am responsible for the maintenance of the software environment.

In this phase it was also decided to give the project a name: AURORA. This is an acronym for "Avoiding Uncertainty Regarding Office and Remote Attendance".

3.1 Analysis

The first step is to think about already existing solutions. In the past there were attempts to use a shared calendar to solve the problem regarding the attendance of colleagues. While this was very simple to set up, it quickly proved to be a flawed solution and never established itself. When trying to figure out why it failed the customer came to the following main conclusions:

- A lack of integration in existing workflows
- Too much additional work (e.g. vacation data needs to be inserted in multiple systems)
- Not universal - every team has its own solution

While discussing this it became clear that additional output and input channels would be necessary to achieve acceptance of the employees. There are some time management softwares available which allow the storage of the current location, but none of them seemed to provide the utilities to achieve what the calendar could not. For this reason a survey was conducted to help define requirements for a new software.

3.1.1 Survey

To best meet the requirements of the employees, a survey was conducted to compare different input and output channels. It consisted of two questions: one regarding the input channel and one regarding the output channel. Both questions had pre-defined choices that could be selected with multiple choice. Additionally, it was possible to give own suggestions.

The survey was accessible to every employee of the customer. The vast majority of the employees have a background in managing or developing software projects. In the end the survey had around 60 participants.

Question 1: Which of the following data input options would you prefer?

- *Conversational MS Teams bot:* MS Teams is part of the Office365 package by Microsoft and is a software solution to help teams collaborate. The customer has been using it for several years. In addition to provide the ability to chat, video call and share data, it also allows the addition of user-specific apps. These apps can include a conversational bot and additional tabs. Conversational bots are programs that are trying to mimic the conversation with a real person. The goal of this is to make the interaction with the program as intuitive as possible.
- *E-mail:* The idea here is to input your working status by writing an e-mail to a specific e-mail address. In order to be able to parse the e-mail content and to extract the correct information the message needs to be in a specific format.
- *Responsive intranet site:* A responsive intranet site is a website that can be accessed by a browser. However it can only be accessed if you're connected to the companies network (by being on site or via VPN). Responsive means that the website adjusts itself according to the size of the viewport it is displayed on.

Question 2: Which of the following data output options would you prefer?

- *Responsive intranet site:* See description in question 1.
- *Responsive website with login credentials:* The only difference between this and the responsive intranet site is that this website is not restricted to the company's network. It can be accessed from every place with an internet connection. In this case it is necessary that the user logs himself in so that only employees of the customer can access this data.
- *Conversational MS Teams bot:* See description in question 1.
- *Dynamic Status in MS Teams:* In Microsoft Teams (see description in question 1 - conversational MS Teams bot) every user has a status. This status can be "available", "busy", "do not disturb", "be right back", "appear away" or "out of office". In addition it is possible to write a custom text that gets displayed when you hover over the status icon. The idea is to automatically set these values according to the status the user puts into the system.

3.1.1.1 Results

The result of this survey can be seen in figure 3.1 and figure 3.2. As for the input options the most voted option was the conversational MS Teams Bot with 22 votes. With just two votes less, the responsive intranet site was the second most popular choice. The third choice “E-mail“ was only voted for nine times. The preferred output options were the dynamic MS Teams Status and the responsive intranet site with 20 votes each, followed by the website with 10 votes and the conversational MS Teams bot with 9 votes. Based on this result and the customer needs, the requirements were defined.

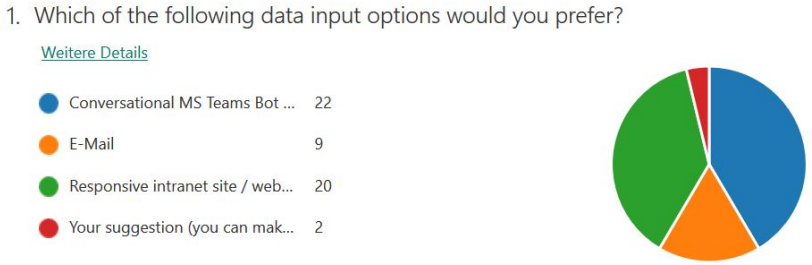


Figure 3.1: Survey results of question 1.

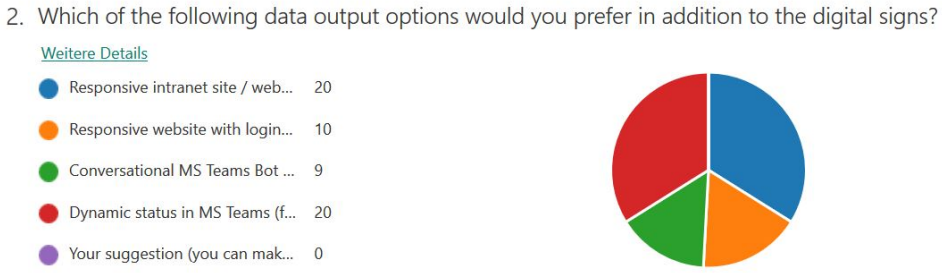


Figure 3.2: Survey results of question 2.

3.2 Functional Requirements

This section lists and explains all functional requirements for the system. It was decided to give each requirement a priority between 1 and 3 with 1 being the most important priority. Each requirement with priority 1 is a “minimum requirement” and each requirement with a priority beyond is considered an “additional requirement”. A requirement is part of the minimum requirements if it is absolutely critical to get it realized. Additional requirements are requirements for which an attempt is made to implement them. However if they are not implemented at the end of the project the solution is still considered to be usable and the project therefore can be finished successfully. More colloquially spoken: These features are “nice to have“.

3.2.1 Data

This section summarizes the requirements that define which data needs to be maintained in the system.

3.2.1.1 [R01] Employees - *Priority 1*

The system should store a record of each employee in a persistent way. The minimum requirement is that these records can be created by deploying a JavaScript Object Notation (JSON) file with the required information onto the systems server.

3.2.1.2 [R02] Status - *Priority 1*

The system should store a status for each employee and workday in a persistent way. The status of an employee shall always be one of the three options “office”, “remote” or “absent”. The status is defined unspecifically because of data privacy-reasons, meaning more detailed information such as where someone is working from or why someone is not working at all should not be published. Another reason is simplicity in order to make the system more usable. The default status is “office”. Furthermore only regular workdays from monday to friday are relevant to the system. This is because at the customer working during the weekend is undesired in order to preserve the work-life balance of their employees.

3.2.1.3 [R03] Teams - *Priority 1*

The system should store teams in a persistent way. Every team consists of zero or more employees. The minimum requirement is that these teams can be created by deploying a json file with the required information on the systems server.

3.2.1.4 [R04] Duration of Data Persistance - *Priority 1*

The system should store the status for at least two months. Because of the COVID-19 crisis, it was decided that the system should store the attendance data of the past. This way, if an infected person was working in the office, all potential contacts can be determined. To prevent an unnecessary big database, obsolete data can be deleted after the decided time period.

3.2.2 Data Input

Data input summarizes all requirements that have something to do with how data is stored in the system.

3.2.2.1 [R05] User Input Medium: E-mail - *Priority 1*

The system should be able to process data sent by e-mail. As a minimum requirement, it was decided to use an e-mail server as an instrument for employees to put their status into the system. The reason for choosing the e-mail server, despite its low popularity in the survey, was the predicted ease of implementation and the resulting assurance that it could be finished on time. The simplicity in this case occurs because it is easy to identify the employee. The employee e-mail addresses have a specified format and by parsing the e-mail address you can easily update the status of the employee.

3.2.2.2 [R06] User Input Medium: Public Web App with Microsoft Login - *Priority 2*

The system may provide a web app as a medium to input data. By using the Microsoft credentials, the additional work for the end user can be reduced compared to a custom login process. That is because he already has a Microsoft account which can prove his identity and affiliation with the company. The web app can be considered as an extension to the intranet site (section 3.2.3.2). When authenticating the user it is possible to provide additional functionality. The most important feature is the possibility to enable the user to input his data through the website because he can be identified. Also it would make it possible to allow users outside of the companies network to access the site as the data is secured behind a login.

3.2.2.3 [R07] User Input Medium - MS Teams App: *Priority 2*

The system may be able to process data sent to a MS Teams conversational bot. The user could input his data in a defined text form and get a response from the system verifying his input.

3.2.2.4 [R08] Include Projectfacts Data - *Priority 3*

The system may be able to process data pulled from the Application Programming Interface (API) of the third-party software “projectfacts“, which the customer already uses to store the working hours of their employees. This data would consist of the registered vacations and sick leaves of employees. This way the end-user does not have to do work twice and is more likely to actively use the solution.

3.2.2.5 [R09] Include Microsoft Data - *Priority 2*

The system may automatically pull and update both users and teams from the Microsoft Graph API. This would increase the maintainability by a huge margin because teams and users would not need to be created and updated manually.

3.2.2.6 [R10] Regular Schedule - *Priority 2*

The system should be able to store the regular schedule regarding home office of an employee. This is because many employees have one workday per week when they work from home. Thus this would make the system more user-friendly because the end user only has to update his data if there is an exception to his regular schedule.

3.2.2.7 [R11] Holidays - *Priority 3*

The system may be able to recognize days that are official holidays in Baden-Württemberg and automatically set the status of every employee to "absent".

3.2.3 Data Output

This subsection summarizes all requirements regarding the way the data of the system gets displayed.

3.2.3.1 [R12] Output Medium: Digital Sign - *Priority 1*

The system should display data on digital signs. Because the introduction of digital signs has already been planned at the customer, it was decided to use them as primary output channel. Digital signs are basically computer screens that show dynamic content like presentations or web sites. The advantage of digital signs compared to "non-digital signs" is the possibility to switch the displayed information depending on daytime, currentness of the information, a regular schedule and many more. Probably the most common use of digital signs is advertising. The customer wants to use them in multiple ways in the future, for example to monitor meeting rooms occupancy or display company news. But in this pilot project the signs should hang in the meeting rooms of the individual teams. This way, when doing daily meetings for example, all members present can get a quick overview of the attendance of colleagues. The digital signs are defined as a minimum by the customer.

3.2.3.2 [R13] Output Medium: Intranet Site - *Priority 1*

The system should display data on a website that is accessible when connected to the customers network. The main advantage of using the intranet site is the possibility to access the site from every supported device and interact. Because of the interaction the user can access all teams instead of only a predefined-one. The digital sign on the other hand lacks this interaction and therefore needs to be pre-configured to a specific team. Because the intranet site makes the system so much more flexible it was decided to also make this a priority-1 requirement.

3.2.3.3 [R14] Output Medium: MS Teams App - *Priority 2*

The system may display data in a MS Teams tab. A MS Teams tab provides the possibility to include an iframe with a custom website inside the MS Teams app. An iframe is an Hyper Text Markup Language (HTML)-tag to embed a website within another website. An example of a tab in MS Teams can be seen in figure 3.3. This website is then accessible through a tab in the chat section of every team. This fact could be used to navigate to the correct team depending on from which team it is viewed from. The main advantage of the MS Teams app is the fact that MS Teams is already in use at the customer, which means that the employees would not need to use an additional solution but rather can use the software already in use.

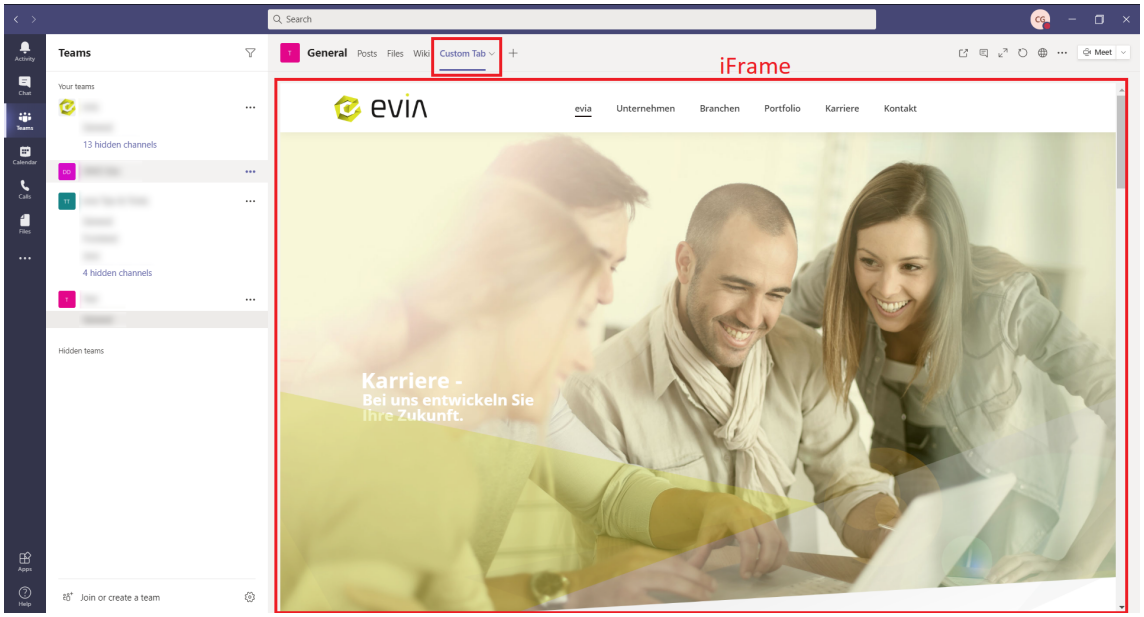


Figure 3.3: Example of a custom MS Teams tab

3.2.4 Other

Here are all requirements listed that do not fit any of the above categories.

3.2.4.1 [R15] Team Memberships (Depends on 3.2.2.2) - *Priority 2*

The system may provide the user with a pre-selection of teams he is a member of. One of the big advantages of the website with login is that it can be personalized. If the public web app with login is implemented the user could see a list of all his current teams and choose one. This should be implemented as a dropdown.

3.2.4.2 [R16] Search Colleague (Depends on 3.2.3.2) - *Priority 3*

The system may provide a way to search for a colleague through an autocomplete input on the intranet site. The autocomplete is necessary because otherwise the user would need to know the exact name of the colleague he is searching for which is not very user-friendly. A dropdown menu, which also would solve this problem, would be too big to be useful. As this use case is supposed to be very uncommon, it was decided to categorize it as priority-3 requirement.

3.2.4.3 [R17] Favorite Team (Depends on 3.2.2.2) - *Priority 3*

The system may provide a way to save a favorite team when using the web app. If the public web app with login is implemented there could be an option where the user can select his current "main" team. This information should be saved and available for future logins. However this is a pure comfort function that does not provide that much functionality, which is why it only got rated as priority 3.

3.2.4.4 [R18] Themes (Depends on 3.2.3.2) - *Priority 3*

The system may provide different themes when using the web app. There could be the option to choose between a dark and a light theme. While this improves the user experience it is also a comfort function, which is why it is also rated priority 3.

3.3 Non-Functional Requirements

Since this is a non-critical system, typical non-functional requirements like availability and reliability are not that important. In this context, "non-critical system" means that no cost will arise if the system is not available. The worst-case scenario could be that an employee falsely assumes that his colleague is in the home office when he is actually on site.

3.3.1 [R19] Maintainability - *Priority 1*

The ISO defines maintainability as the "degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers" [ISO]. The customer defined this as a priority-1 requirement because he may wants to keep developing this application after this thesis is finished. In the context of this thesis this project is finished after six months with a proof of concept application. Everything that gets developed after this time period counts as maintenance. However for the customer this project could be further developed in an incremental way. This means that the solution should be built in a way that allows later additions to the system. This is because it is not clear at the time of designing the system if the additional requirements can be implemented within the time period of six months. Other examples, even beyond the additional requirements, are the dynamic MS Teams status or additional functions for the digital signs that could potentially be added after this thesis. This is why maintainability is such an important requirement.

3.3.2 [R20] Security - *Priority 1*

Because the system stores confident data, security is also a very important non-functional requirement. The National Institute of Standards and Technology (NIST) describes that the main efforts of security are to ensure “confidentiality”, “integrity” and “availability”. [LB10] They define confidentiality as “Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information”. [LB10] Integrity is defined as the effort to avoid unauthorized modification or destruction of information. [LB10] Availability means to ensure the “timely and reliable access to and use of information”. [LB10] Out of these three, confidentiality is the most important one for this system, since the customer first and foremost wants to avoid data leaks. This does not mean that the other two parts of security can be completely neglected though. Thats why security is also a number-1 priority.

3.3.3 [R21] Usability - *Priority 2*

Usability means that the software should be intuitive and simple to use. This is a requirement that should always be considered when developing software. However neither maintainability nor the security should be sacrificed in order to achieve better usability because that could lead to unexpected cost for the customer. But otherwise it should always be the goal to make the software as usable as possible. For that reason it is a number-2 priority.

3.4 Summary

The dynamic MS Teams status, which was very popular in the survey, had to be ignored because the MS Teams API does not support this feature at the time of this thesis. The decision to implement many of the additional requirements can be reduced to the decision of whether to use the Microsoft ecosystem in this project or not. If so, it would be possible to use existing systems such as the identification of the user and the teams to which he belongs to. This is because the customer already is part of the Microsoft ecosystem and maintains the relevant data there. Otherwise it would be necessary to create and maintain a new database for this to be independent from the Microsoft ecosystem. But because the effort of connecting the system to the Microsoft ecosystem is hard to estimate at the time of the requirements engineering, the requirements were divided into minimum and additional requirements. In addition to that, it is not clear if the management of the customer will allow the access of a “third party application” (the solution developed in this thesis) to connect to Microsoft because of security concerns. That’s why the system needs to be designed in a way that easily allows to implement or respectively not to implement the additional requirements. An overview of all requirements is given in table 3.2. Overall there are 21 requirements. Out of these, nine were rated as priority 1. Respectively eight were rated as priority 2 and four as priority 3.

For a better understanding, of the system that needs to be designed, table 3.1 additionally gives an overview of the different input and output channels categorized by priority.

Priority	User Input Channel	Output Channel
1	E-mail	Digital Sign, Intranet Site
2	MS Teams App (bot), Web App	MS Teams App (tab)
3	-	-

Table 3.1: Overview of input and output channels categorized by priority.

ID	Requirement	Type
Priority 1 - Minimum Requirements		
R01	Employees	f
R02	Status	f
R03	Teams	f
R04	Persistance Duration	f
R05	Input - E-mail	f
R12	Output - Digital Sign	f
R13	Output - Intranet Site	f
R19	Maintainability	nf
R21	Security	nf
Priority 2 - Additional Requirements		
R06	Input - Web-App	f
R07	Input - MS Teams-App	f
R09	Include Microsoft Data	f
R10	Regular Schedule	f
R14	Output - MS Teams	f
R15	Team Memberships	f
R16	Search colleague	f
R21	Usability	nf
Priority 3 - Additional Requirements		
R08	Include Projectfacts	f
R11	Holidays	f
R17	Main Team	f
R18	Theme	f

Table 3.2: Overview of requirements categorized by priority.

4 Design

This section explains all major design choices. A simplified diagram of the final architecture is given in figure 4.1. On a high level, the architecture is based on the client-server model. In the client-server model the application is split up into the client software and the server software. One server can serve multiple client applications. These clients do not necessarily need to be instances of the same application. In a web app the client software does not need to be installed on the end users machine, but rather gets delivered by the server and is executed in a browser. This is a good fit for a system with the requirements of AURORA. All of the output mediums offer the possibility to display a website (browser, digital signs, MS Teams tab). While this is obvious for the browser and the MS Teams tab (iframe), a digital sign can be implemented in any technology that enables the developer to display graphical content. This makes it possible to develop one client-side application rather than three, and thus potentially reduce the development time by a significant margin.

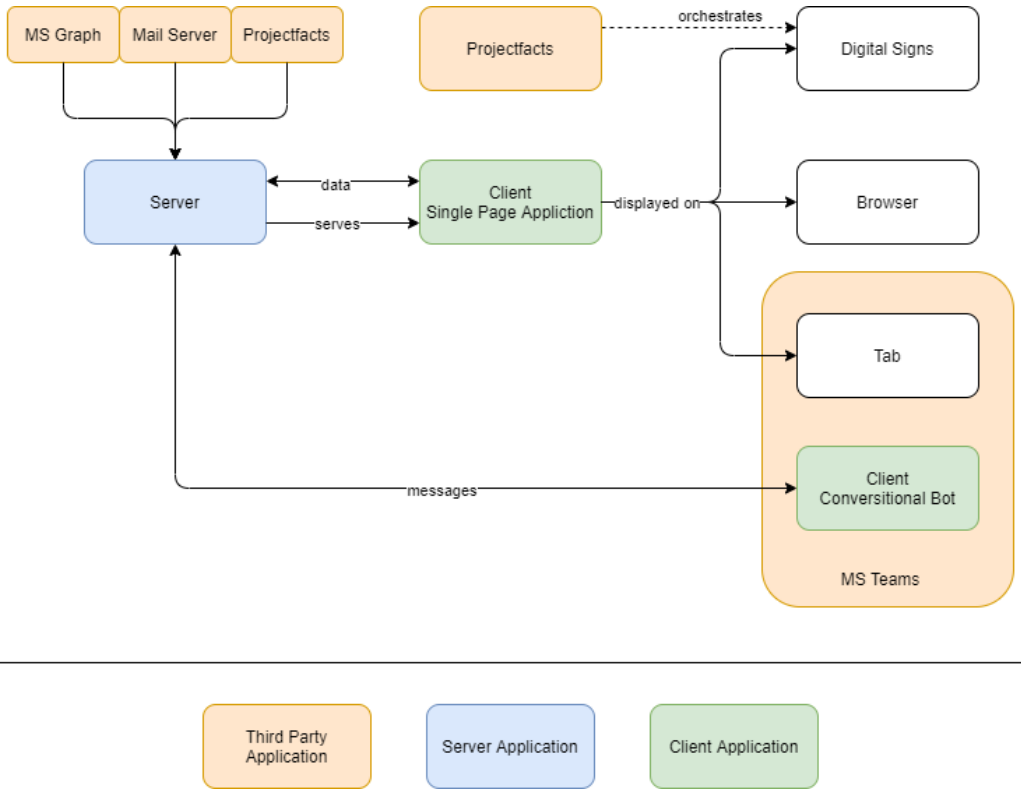


Figure 4.1: High-level architecture diagram.

One by one all of the instances are described and explained in the following paragraphs. It is important to note that the design should include all requirements (minimum and additional), regardless of whether they are actually implemented as part of this thesis. This way they can potentially be added later.

4.1 Basics

In this section basic technologies, which will be used throughout the design of the system, are explained.

4.1.1 Docker

Docker is an open source software developed and maintained by the Docker Inc. It helps isolating systems by using so-called containers. Docker describes a container as an "unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another" [Doc]. It ensures that a piece of software can be executed on every machine that has docker installed by providing the same environment every time the container runs. This includes the runtime, the code, system tools, system libraries and settings. The way docker works is by writing a dockerfile first. Based on this dockerfile a docker image can be build. This image can then be executed on the docker engine to run one or multiple containers. According to Docker Inc. one of the biggest advantages of containers over virtual machines is that containers are much more lightweight [Doc]. For this reason docker containers are also a valuable tool in scaling applications vertically. This is often done in combination with a software for clustering, like for example kubernetes.

4.1.2 OAuth 2.0

OAuth 2.0 is an authorization standard developed by the IETF OAuth Working Group. It provides different authorization flows for web apps, desktop apps and other applications. It uses the JSON Web Token (JWT) as a Bearer Token to verify users or servers. JWT itself is another standard that is used as a way to encode claims in a secure way. OAuth defines different flows based on whether the token exchange happens via a back channel or via a front channel. A back channel describes a trusted source like a backend. The front channel is a distrusted source like a browser.

4.2 Server Application

The Server application is the heart of the system because every input and output channel ends up there. To achieve the required modularity the software will be split up into different modules. The goal is that all modules can work independently from the other modules. However, the Back End (code that runs on the server) (BE) itself is designed as a monolith architecture instead of the more modern microservice architecture, which is very modular by nature. In short, a monolith

architecture consists of a single codebase that is deployed as a whole. In the microservice pattern the software is split up into different services. Each service consists of its own codebase and is developed, tested and deployed independently from the other services.

In software development there exists a principle called KISS. KISS is an acronym that has different dissolutions (originally “keep it simple, stupid”) but they all appeal to keeping a system as simple as possible without complicating it unnecessarily [Pri]. It is a widely known and accepted principle within the software development community [Pri] [Smi] and will be applied on all decisions made during designing the system. It was decided to not use microservices just because it is currently very popular. This has a couple of reasons.

In the paper “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud” M. Villamizar et al. compare the two architectures and present their results [VGC+15]. They come to the conclusion that for applications with a small number of users (hundreds or thousands) the monolith approach can be more practical and a faster way to start [VGC+15]. This is because a microservice architecture introduces new problems while developing and deploying the software [VGC+15]. With the current number of employees at the customer it is not that important to think of scalability for now. What is important however, is to get the system up and running in the time period of this thesis. A big advandtage of microservices is that different teams can work on different services and develop them in their own language and their own time schedule. This advantage is not relevant for this project though, because only one person actively develops the software. According to M. Villamizar et al. microservices make most sense when the software has “thousands or millions of users” and the company developing it has to manage a large code base [VGC+15].

It is also worth mentioning that the application will run in docker containers. This allows easy deploys and can reduce the maintaining cost because it reduces the risk of problems originating in application dependencies. The database and the rest of the BE will run in separate containers. This way the setup of the database will be easier because a pre-build image from the official docker registry can be used. The resulting architecture is shown in figure 4.2. The center of the application is the “data controller” module. This module will provide basic functionality to interact with the the database. These functions should at least consist of:

- Creating a user
- Creating a team
- Deleting a user
- Deleting a team
- Updating a user
- Updating a team
- Reading a specific user from the database
- Reading a specific team from the database
- Reading a list of users from the database, based on query parameters
- Reading a list of teams from the database, based on query parameters

This covers the four basic operations for persistent storage: create, read, update, delete (CRUD). If needed there can be additional functions.

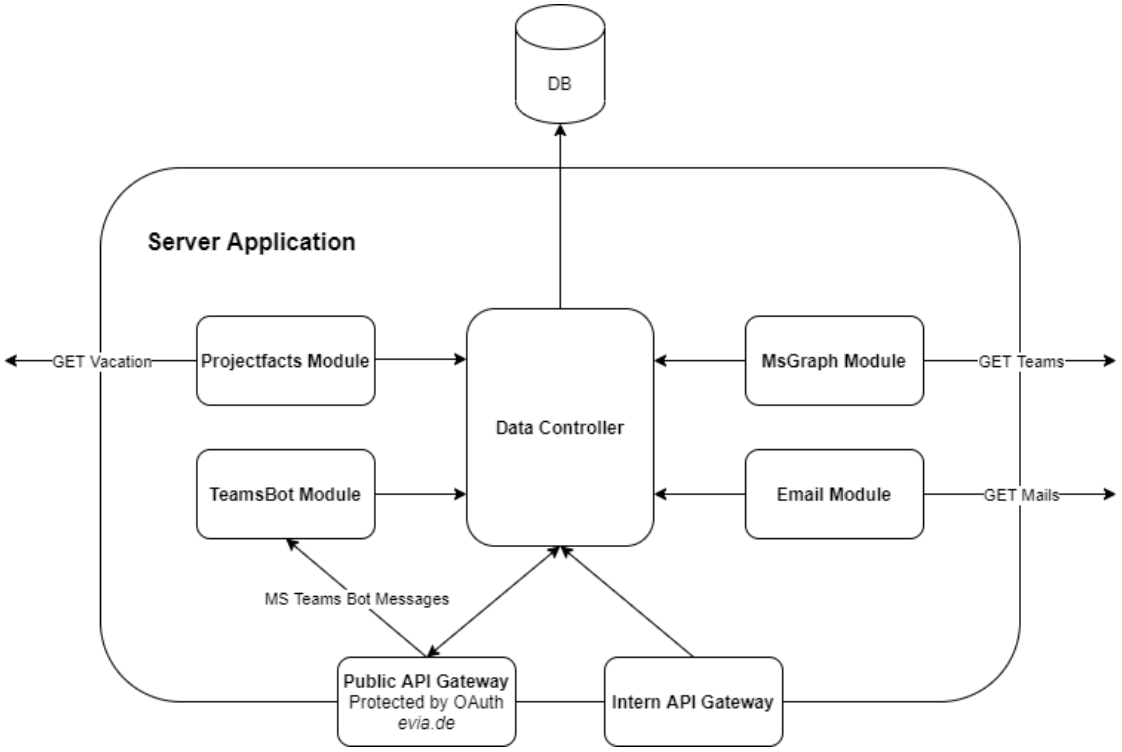


Figure 4.2: Modular monolithic structure of the server application.

4.2.1 Technologies

When evaluating what technologies to use it was decided to try and use as many technologies as possible which the developer already knows about. This only applies though, if this technology does not have huge drawbacks to alternative solutions.

Because the server application is the server of a web app the technology needs to fit this use case. Among the most popular choices for such a case are Java, PHP, .Net or NodeJs. The choice fell on NodeJs. NodeJs is a JavaScript runtime built on Chrome's V8 JavaScript engine. It is open-source and managed by the OpenJs Foundation. This was decided as it allows the same programming language to be used in the BE and the Front End (code that runs on the client machine) (FE). Since the developer has a background in FE development this is a big advantage over the other technologies.

All modules will be written in Typescript. Typescript is a superset of JavaScript, which itself is standardized by EcmaScript. Typescript is also open-source and managed by Microsoft. It extends JavaScript by introducing static typing to the language.

The endpoints will be implemented with the help of "express". This is a popular framework to create API's and host websites on NodeJs. Currently (2020/11) it has around 15 million weekly downloads [npm].

The chosen database is “MongoDB”. This decision was made because MongoDB works well with NodeJs. The reason for this is that this is a document-based database rather than a SQL database. These documents are very similar to the JSON syntax which means that JavaScript objects can be stored in a very convenient way. While there are also other document-based databases, MongoDB is amongst the most popular choices. In addition to this the developer already made good experiences with it in the past.

4.2.2 Database

Figure 4.3 shows the chosen data model. The database consists of two collections, the users collection and the teams collection. The users collection contains user documents which have the “UserId” as primary key. The teams collection contains team documents which have the “TeamId” as primary key. A user has a “regular” property which contains an array of regular schedules. A regular schedule contains a status for every workday and a date on which day the schedule was created. It is an array because a change to the regular schedule should not change the calculated attendance retroactively. Possible values for the status are: “absent”, “remote” and “office”. The default value is “office” for each weekday. The “exceptions” property is an array with exception objects. These objects consist of a date property, which is a JavaScript date object, and a status property, which can receive the same three values as the weekdays of the “regular” object.

The entries of the property “members” from the team document are foreign keys of which each references one user document. This is called a normalized data model. It is used here because the users and teams are in a “many-to-many” relationship, since every user can be (and generally is) in many different teams and every team has multiple members. The mongoDB documentation suggests to use the normalized data model in this case [Mon].

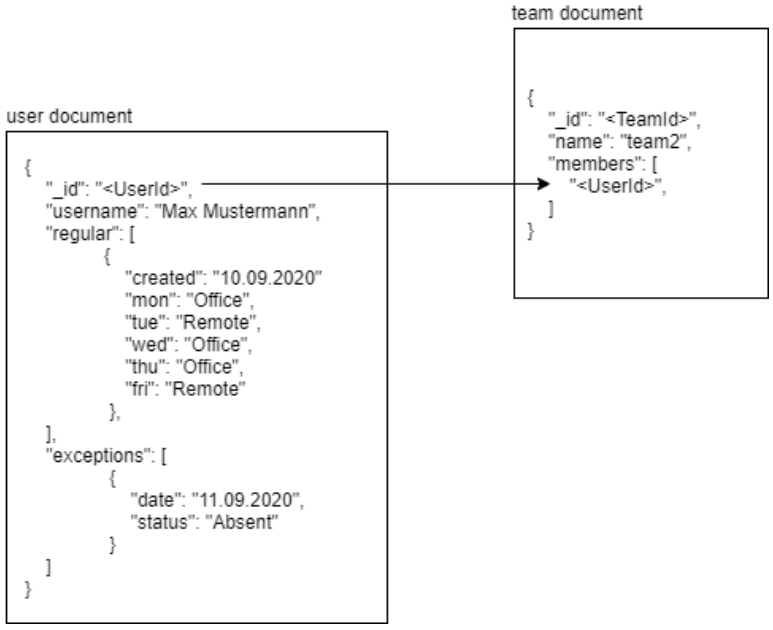


Figure 4.3: MongoDB data model.

The “exceptions” objects however are in a “one-to-many-relationship”. In this case the mongoDB documentation suggests the use of the embedded data model [Mon].

In conclusion the chosen data model is a hybrid between the normalized data model and the embedded data model.

The goal of this particular model is to reduce the size of the database by only storing the exceptions instead of saving a record each regular day. A disadvantage of this is that time the regular schedule changes, the previous record still needs to be stored in the database in order to calculate the correct status. However, it is expected that the average regular schedule will not be changed often enough for this to become a problem.

4.2.3 MSGraph Module

The MSGraph module will be responsible for pulling the existing MS Teams teams and employees from the MSGraph API and storing the data in the database by creating new or updating existing entries. It should do so twice a day automatically. In addition there should be an endpoint to force a new pull request. The module authenticates itself against the MSGraph API via Oauth 2.0 (section 4.1.2) by storing a secret securely in the backend.

4.2.4 Projectfacts Module

The Projectfacts module pulls the existing vacation data from the third-party software called “Projectfacts”. This should also happen twice a day and additionally, when required, by calling a specific endpoint. It authenticates itself with the help of a key that can be provided by the module team of the projectfacts company.

4.2.5 TeamsBot Module

The TeamsBot module provides the functionality required to operate an MS Teams bot. This module receives the message object from MS Teams, extracts the data with a regex from the message, and stores it in the database.

4.2.6 E-mail Module

The E-mail module pulls the e-mail from an e-mail server. Afterwards a regex scans all new e-mails and extracts the attendance status. This information is then updated in the database. The user will be identified by the senders e-mail address.

4.2.7 Endpoints

The application should serve all endpoints on two different ports. One for serving the publicly accessible domain and one for serving requests from within the customers network.

The publicly available API endpoints that provide sensitive data are all protected by OAuth 2.0 (section 4.1.2) to only be accessible after a successful login with the customers Microsoft account.

The endpoints available in the intranet are unsecured when pulling data. This ensures that the digital signs can access the data. The reason for this will be explained in section 4.3.2.

The endpoints, the server application should provide, are shown in table 4.1.

Endpoint	**Explanation**
GET: \	This endpoint serves the Single Page Application (SPA) (4.3) and must be unprotected because the SPA provides the login form.
GET: *teams-config*	This endpoint points to the config file which is explained in 4.5. It is not secured for the same reasons that applied to the SPA.
POST: *api**teams-bot**messages*	Forwards the requests to the TeamsBot module described in section 4.2.5. This endpoint must be secured when exposed on a public port.
POST: *api**updateTeams*	Triggers the pull of teams described in section 4.2.3.
POST: *api**updateVacation*	Triggers the pull of vacation described in section 4.2.4.
POST/GET/PUT/DELETE: *api**teams*\...	Executes the basic database operations for teams, provided by the controller which is explained in section 4.2. These endpoints must be secured when exposed publicly.
POST/GET/PUT/DELETE: *api**user*\...	Same as the endpoint above, except for users instead of teams.

Table 4.1: Overview of HTTP Endpoints.

4.3 Single-Page Client Application

An SPA is a website that is loaded once and then updates its content via data from API's. This concept is called Asynchronous JavaScript And XML (AJAX). It can boost performance once the site is initially loaded and can provide a more dynamic behavior [Moz]. The SPA stands in contrast to the more traditional Multi Page Application (MPA) which loads a new document from a server if new content is to be displayed.

Both concepts have advantages and disadvantages. However for this project it was decided to use an SPA solely because the developer has much more expertise in this area. The limited time window for this project makes it necessary to use known technologies rather than learning new ones, if possible.

4.3.1 Technology

There are many frameworks that simplify the building of a SPA. The biggest ones are Angular (Google), React (Facebook) and VueJs [Moz]. Technically both, React and VueJs, are libraries and not real frameworks but because of their size they are often referred to as such. They share many similarities, for example all of them use components as basic building blocks. To choose the appropriate solution the same arguments that were applied for using SPA can be considered here. Because the developer already is skilled in Angular, it was decided that it is unnecessary to learn a new framework, especially because the resulting web app does not have very specific functionalities. This way more time can be spent actually developing the software.

Together with the technologies from the application server the whole technology stack of the application is called "MEAN" stack (MongoDB, Express, Angular, NodeJs). These technologies are known for working together nicely, which is why this stack made a name for itself. According to IBM the MEAN stack is mostly "valued for its flexibility, consistent language, and scalability" [IBM]. Angular is an open-source framework written in Type Script and maintained by Google. It is licensed under the MIT Licence [Gooa]. It comes with its own command-line integration and there are add-ons for many Integrated Development Environments (IDEs).

4.3.2 Parameters

As seen in figure 4.1 the website should be displayed in a browser, inside the MS Teams app, and on digital signs. This is done to reduce the overall development time but also provides a new challenge: authorization.

The digital signs can not authorize themselves because this would require user interaction every time the side is loaded. In this case the SPA should access the API via an endpoint that is only accessible in the intranet of the company. This works because the digital signs will always be located in the customers offices and therefore have access to the intranet. As a result, the data is safe and at the same time quite easy to access, although somewhat limited. The same arguments apply for only implementing the intranet website. As a consequence, these mediums can only be used as output channels because there is no identified user which would make it impossible to map input data in the backend.

The website with login credentials can work as an input channel though. In this case the user can authorize himself by logging in with his Microsoft account. For such use-case, Microsoft provides the Azure Active Directory - Identity platform by Microsoft (AAD) [Mica]. It supports features like Single Sign On (SSO) and conditional access. Conditional access is important because users do not have to give an application access to their entire profile. Instead the application asks for specific permissions. In this case, for example, the permission to access the server application. The AAD platform works based on the OAuth 2.0 protocol described in section 4.1.2.

The third display medium is MS Teams. In this case the website can also use the AAD platform to authorize the user. To make the usage more convenient for the user, the application should use the fact that the user is already logged in when using MS Teams. This is possible with help of the "Microsoft Teams JavaScript SDK" [Micc]. Which authorization to use is controlled by an Hypertext Transfer Protocol (HTTP) parameter called: "displayMedium".

The appearance also differs between the different mediums. The browser version and the MS Teams app version should be responsive, meaning they should optimize their interface on mobile devices. In addition the SPA serves as input and output channel when displayed in the browser (if not only the intranet site is implemented). Within the MS Team app or a digital sign, however, it only serves as an output channel. This means that the input controls on the site need to be optional. The input controls are opted in or out according to the HTTP parameter "displayMedium".

Because digital signs are not necessarily interactive the displayed team needs to be predefined. This can also be used when displayed inside the MS Teams app because the website gets opened in the context of a specific team. Through an HTTP parameter called "teamId" a team can be preselected.

In order to optimize the user experience when using the SPA it should include the possibility to choose between a dark and a light theme. The theme can be selected through an HTTP parameter called "theme".

Figure 4.2 gives an overview of the possible HTTP parameters and their allowed values.

Name	**Allowed values**
displayMedium	"standaloneBrowser", "teamsTab", "digitalSign"
teamId	<an existing team id>
theme	"dark", "light"

Table 4.2: Overview of HTTP Parameters.

4.4 PiSignage

PiSignage is an open-source solution to operate and host digital signs. Compared to just displaying the SPA through a browser on a monitor, the advantage of using a digital-sign software is extensibility. PiSignage offers additional features like playlists for different contents, schedules, monitoring of all

digital signs and many more [PiS]. So it is easy to add content to the digital signs e.g. company news, meeting-room occupancy or meal plans. This is important because, as mentioned in section 1.2, this project only is a pilot project for digital signs.

While there are different available solutions for this, PiSignage convinces with the possibility of hosting the server locally and one-time payment for licensing the individual players. There were no other solutions found that offer this payment method, instead they were all using a subscription based model.

PiSignage also will run inside a docker container to make it independent from the environment. This way the restart will be easier if the machine fails for some reason. As mentioned in section 3.3, this is an important non-functional requirement.

4.5 MS Teams App

The MS teams app is the second client of the application. It is defined by a JSON file that can be created with the help of the MS Teams app “App Studio”. Two different endpoints have to be defined in this file. One is the endpoint for the MS Teams bot which is discussed in section 4.2.5. The other endpoint needs to point to a specific configuration file, in which the endpoint to the actual web app is listed.

5 Implementation

This chapter describes the implementation process. For each section there is an explanation on why different components are implemented as they are. Additionally there are some code snippets to help understand these choices. At the end there is a summary of the final result.

5.1 Development Environment

For the development process a new GitLab server was deployed on a Virtual Maschine (VM) on one of the customers internal servers. A VM is a software-based abstraction of a computer system within another computer system. GitLab is an open-source development platform based on git, which is a popular version-controlling software. This VM is named “aurora” and runs on Ubuntu 20.04 which is a linux distribution. Linux is an open-source operating system and an alternative to the popular systems windows and macOS. It is open-source and was initially published by Linus Torvalds. The usage of such software while developing has many benefits. The biggest one is the versioning. Earlier versions of a software can easily be restored if a new feature broke something.

An additional advantage arises when a so called “remote repository” is used. A repository is a directory which is monitored by git. A remote git repository can be easily established by using a software like GitLab. This way the progress can be stored on a centralized server. The most obvious advantage of this is that the software is stored on different hardware and therefore the risk of loosing progress by hardware failure is reduced. But it also considerably improves the ability to collaborate on a software project because one “remote repository” can be connected to many “local repositories”. Git also allows to merge different software versions together. These different versions of a software are stored in so called “branches”.

GitLab also provides additional features like a Continous Integration/ Continous Deployment (CI/CD) system, a wiki, a user management and many more. The community edition is free to use and allows the deployment of a GitLab instance on private servers.

Two git projects were set up for this project. “aurora-frontend” contains the code for the SPA. “aurora-backend” contains the code of the server application.

To allow easier additions after the software is deployed, which is an important non-functional requirement, a CI/CD system was set up. For that, two additional VM’s were created on the customers servers, to host the actual application. These VM’s also run on Ubuntu 20.04 and will be addressed as “application servers” from now on. One VM is called “aurora-01” and is designed to run a development version of the application. This makes it easy to test out new features before actually delivering it to the end users. The other VM is called “aurora-02” and runs the production version of the software. This is the version that actually gets used by the employees of the customer.

As soon as a new version gets committed to the branch “dev” of the git project “aurora-backend” or “aurora-frontend”, the application is tested, built and deployed on “aurora-01”. The same happens on “aurora-02” as soon as something is committed to the branch “master” on either of these projects.

This is done with the help of a tool called “GitLab-runner”. This software is installed on the two application servers and connected to the GitLab instance on the aurora VM. Then a file called “GitLab-ci.yml” is added to the git repositories which defines what runner should be selected and what commands they have to execute to build and start the application.

To further improve the quality of the code in the repositories, a hook was implemented that lints the code before it gets pushed to the remote repository. Linting is a process in which code gets analyzed regarding style violations and potential bugs. This hook is implemented with help of a library called “husky”. Together with the regression testing every time a new new version gets deployed by the “GitLab-runner”, it helps preventing bugs and bad coding styles.

An overview of the development environment is given in figure 5.1. It is important to mention that such a CI/CD is not typical for a software project that is developed with the help of the waterfall model. However in this case it is done to increase the extensibility. After this thesis is finished the CI/CD makes it easy to add new features while the application is already in use.

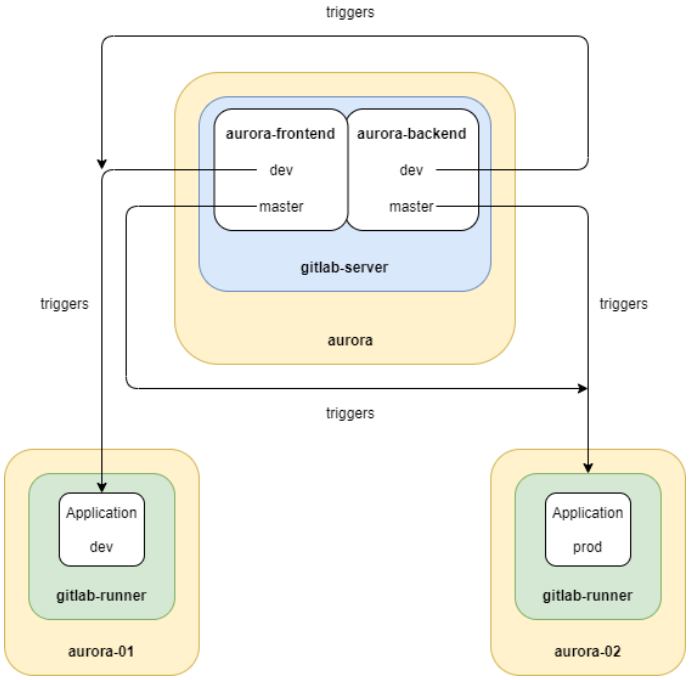


Figure 5.1: CI/CD flow.

5.2 Server Application

The “aurora-backend” project contains a “docker-compose.yml” file. This file can be seen in listing 5.1. It configures a custom virtual network for the application. The only ports opened to the outside are 8080, 8081 (standard HTTP ports) and 27017 (standard mongoDB port). The configuration contains the two services “web” and “mongo”.

“web” starts a container based on an image which compiles the server application. This image also pulls and builds the SPA and copies it into the public folder of the server application. When the container is executed it starts the node process to serve the application.

“mongo” is a pre-build image which is downloaded from the internet and provides the database. The “volumes” parameter allows the container to mount a directory on the machine. This makes the data being stored persistently on the machine even the container stops for some reason. This is important for when the VM shuts down unexpectedly the container also gets stopped and the data would be lost.

This file makes serving the entire application as easy as running one command. When running the command “docker-compose up” in the directory of the “aurora-backend” the network and both services are created.

Listing 5.1 The docker-compose.yml file.

```
version: "3.8"
networks:
  net:
    ipam:
      config:
        - subnet: 172.61.1.0/24
services:
  web:
    networks:
      net:
    build: .
    ports:
      - "8080:3000"
      - "8081:3001"
    depends_on:
    - mongo
mongo:
  networks:
    net:
  image: mongo
  volumes:
    - ~mongo/data:/data/db
  ports:
    - "27017:27017"
```

5.2.1 Mock

The server application also provides a mock database. The goal of the mock is to provide an easier start for new developers because they do not have to install a database to serve the application locally. For this reason there are multiple "data controllers", which each implement a specific interface that defines the possible data operations. This method is called the Data Access Model (DAO) pattern [Ora]. The goal is it to "abstract and encapsulate all access to the data source" [Ora]. This way the data source can be exchanged. Technically the mock database is a simple json file that gets read and written to by the application. While this has many drawbacks compared to dedicated databases, it is sufficient enough for this use case and does not need additional software to be installed. The biggest drawback probably is the performance.

To figure out what DAO should be used there are different environment files that assign values to specific environment variables. When starting the server, one file can be chosen by adding "–env=<environment>" to the command. During the runtime these variables can be accessed through process.env.<nameOfVariable>. The software currently has five environment files: production, development, development-local, mock and test. The development-local file is for using a local instance of MongoDB, while production and development are for using the respective instances on "aurora-01" and "aurora-02". Mock and test are both using the mock database.

5.2.2 Testing

The tests for the server application were written with Jasmine. Jasmine is an open-source behavior-driven testing framework for JavaScript [Jas]. All tests are executed by the CI/CD before a new version of the application gets deployed. This is called regression testing and helps to ensure that a modification in the codebase does not lead to an unexpected behavior somewhere in the application.

5.2.3 Ports

The application exposes two ports: 8080 and 8081. 8080 provides all endpoints without authentication. 8081 secures all relevant endpoints as described in table 4.1. Both of these endpoints are addressable by the ip addresses of the machine but only if the accessing machine is in the same network. In addition port 8080 is publicly available through a proxy which resolves a specific domain and redirects all requests to the respective VM. This way the SPA can access the data when displayed on a digital sign located in the office of the customer but also by employees outside of the customers network (e.g. home office, public transportation, customer site, ...).

5.2.4 TeamsBot Module

The TeamsBot module receives the text message from the MS Teams app. The "TextParser" extracts the intent of the message with the help of different regular expressions. Figure 5.2 shows a sequence diagram that visualizes this process. The text in blue shows an example message while it gets processed by the system.

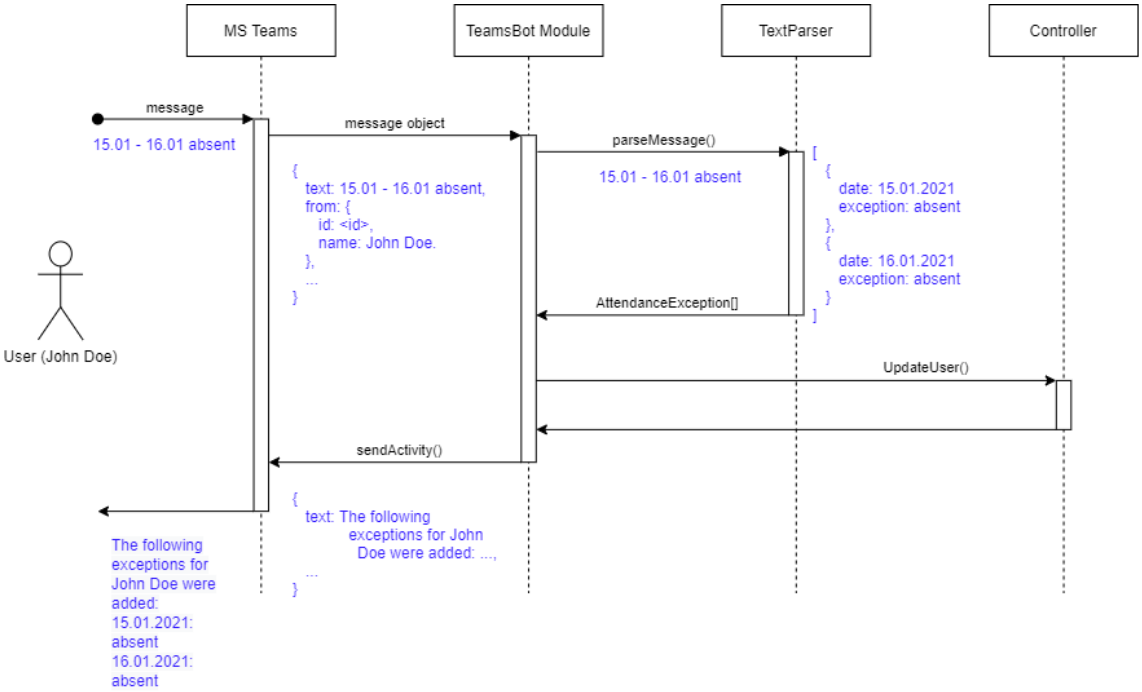


Figure 5.2: Sequence chart of the communication with the MS Teams bot.

The regular expressions recognize dates in the form dd.mm.yyyy (df1) and dd.mm (df2) in which case it assumes that the current year is meant. It also recognizes time spans which consist of two dates (df1 or df2) separated by a hyphen. After it finds a time span or a single date it scans the text until the next date occurrence for the presence of keywords. These keywords resemble the attendance status. “parseMessage()” returns an array of “AttendanceExceptions”. This is an interface consisting of a date, which is a JavaScript Date object, and a exception, which is either “Office”, “Absent” or “Remote”.

After the text is parsed and the entries are updated the user gets a response from the bot. This response message consists of all updated days to give the user feedback on how his input was parsed.

5.2.5 E-mail Module

The first requirement to implement the E-mail module was to set up a new mailbox in Microsoft Outlook. The E-mail module pulls the mails with the Internet Message Access Protocol (IMAP). Because Outlook also provides IMAP IDLE, the module can react in realtime to new messages rather than pulling in predefined intervals. The mail is converted into a string and is processed in the same way as a message from the MS Teams bot (see figure 5.2).

5.2.6 MsGraph Module

The MSGraph module synchronizes the AUROA database with the Graph API automatically every 6 hours.

When a new database synchronization is requested (by the timer or a call to the associated endpoint) the module first removes the team memberships of every user in the AUROA database. At the same time it requests a token to attach to every request by sending a predefined secret to a specific endpoint.

After this the module requests the teams of the Graph API. However the Graph API only ever returns the first one hundred entities. If this number is exceeded it attaches a link that, when requested, automatically sends the next hundred entries. For this reason a function calls itself recursively until no link is send back by the Graph API and merges all teams into one array.

In the next step, for every team, another request gets send to retrieve its members. This happens asynchronously which improves the performance because if the request would block the execution of the software the response time would quickly accumulate. After all requests are finished it summarizes all teams and the associated members.

Then it synchronously updates every team and its members in the AURORA database. This is important because otherwise “lost update” errors can occur. A “lost update” error can occur when multiple read and write operations to a single resource happen in a short time window. More precisely it is a read-write conflict in this case that causes this error. If two processes read a value from a database, that they both want to update, it can happen that the read operations happen so fast after each other that the first process can not write the updated version in time. In this case the second process overwrites the update of the first process. Thats why its called a “lost update” error. Because the read and write operations to a database on the same machine are comparatively fast this is not a noticeable performance issue compared to asynchronous approach.

5.2.7 Projectfacts Module

The projectfacts module functions in a similar way to the MsGraph module. To access the data a technical user was created in the projectfacts database. With the users login data the BE can authenticate itself through “Basic Auth” which is part of the HTTP.

In a defined time interval the module requests the data for the next ten days. This time period is a compromise between workload for the projectfacts server and enough foresight for the AURORA users. The way the projectfacts API is build there is an individual entry for every day an employee submits some information like working hours, vacation or sick leave. With the help of a query parameter the application can send one request for each day, parse the JSON response, and update the entries.

5.3 Single-Page Client Application

To simplify the development of the Graphical User Interface (GUI) it was decided to use the Angular Material library. Angular Material provides different components that were developed with the Material designing system by Google. There are components for inputs, buttons, tables, tabs and many more. This leads to the fact that a good-looking app can be developed with comparatively little effort. Furthermore the components follow common best practices for designing user interfaces and therefore support the development of an application with good usability.

The application consists of multiple services and components, which is typical for Angular. Components are reusable building blocks for an Angular application. They consist of a JavaScript class that contains data and logic which is associated to an HTML template. This template can contain regular HTML and other components. Services are a way to encapsulate logic that is not related to a specific component. They can be injected to each component where the logic is needed. This process is called “Dependency Injection”.

This section explains the gui design and the code structure of the SPA. It also provides some code snippet as examples for important patterns used in the code.

5.3.1 GUI - Design

Before developing the actual software the GUI was designed. However this design is not meant to be an exact specification but rather an approximate draft.

Figure 5.3 shows the draft in a desktop view displayed within the MS Teams app. On the top there are two inputs to search for teams or individual colleagues. Beneath that there is the team section. This is a table with a column for each workday of a week and one column for the names. Two arrow icons on top of the table enable the user to navigate between weeks. At the bottom is the personal section. Here the logged-in user can change his current attendance throughout the week. Alternatively he can change his regular schedule with a click on the button “master settings”. The changing of an attendance works by clicking on the respective table cell. This cell then rotates between the different possible attendance states.

Figure 5.4 shows the different views on a phone. To deal with the smaller display space the two sections were split up into two different views which can be navigated through a navigation bar. The displayed time period is reduced to one day. However on the phone the user has the additional navigation possibility of swipes. With a swipe to the right the user can navigate one day into the future and with a swipe to left one day into the past. By swiping down the two inputs for teams and individuals appear beneath the navigation bar. This is indicated by the “arrow down”.

The personal view is very different on mobile devices compared to the desktop version. This is can be seen in figure 5.4a. To make proper use of the space each attendance state gets its own button. Bigger buttons on mobile versions can be very helpful because the input device on the phone (the thumb) is much more imprecise than the one on the desktop (mouse pointer). To switch between the regular status and the actual status a button switch is beneath the navigation bar.

Figure 5.4b shows the team section on a mobile device.

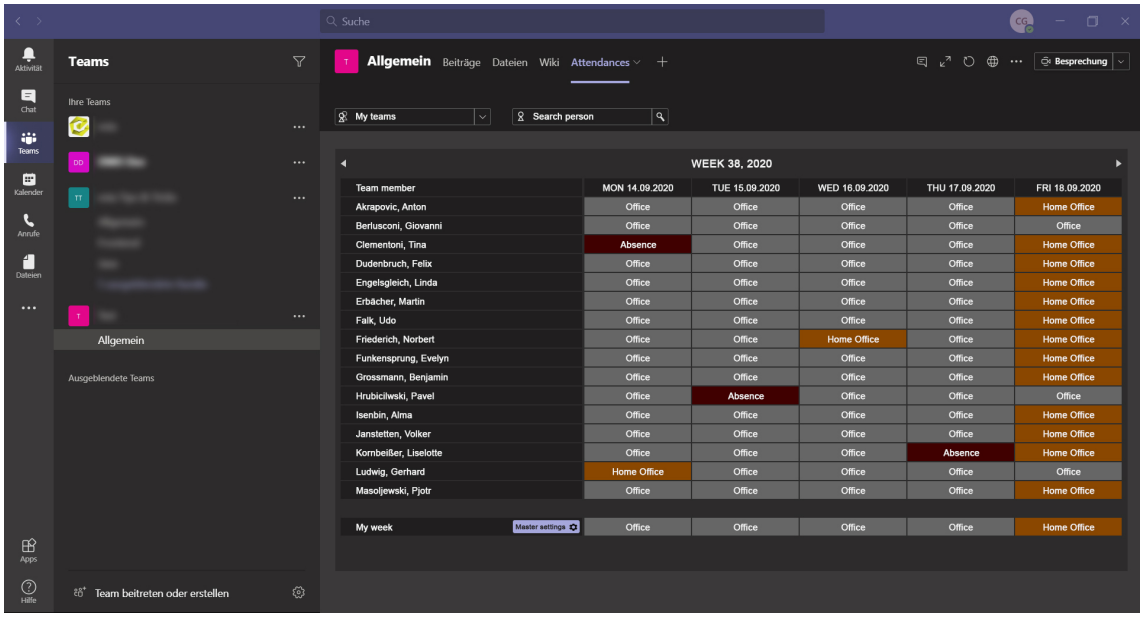
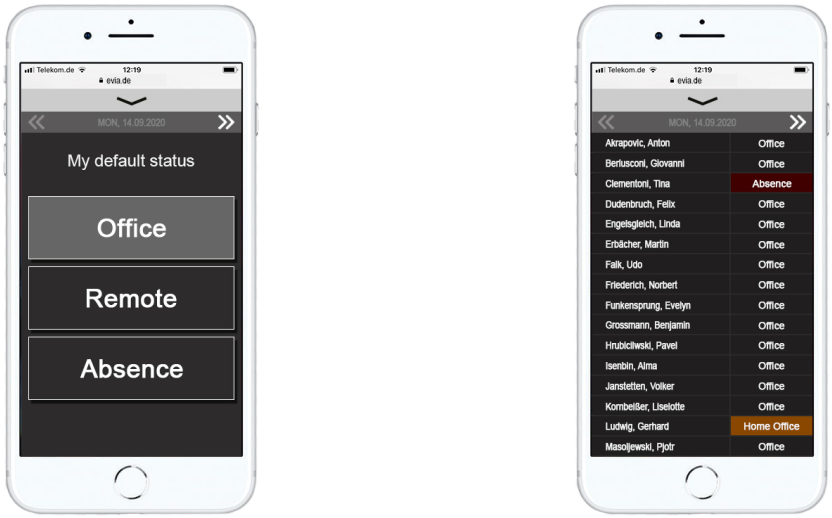


Figure 5.3: Design draft of desktop view.



(a) Mobile personal view.

(b) Mobile team view.

Figure 5.4: Design draft of mobile view.

5.3.2 Services

The AURORA application currently consists of seven services.

- Attendance-Data-Service: This service provides logic to calculate the actual attendance of employees or teams based on the data that gets provided by the BE. This means it takes the regular schedule, the exceptions, and the holidays to calculate an array with objects consisting of a date and a attendance status.
- Auth Service: This service handles everything concerning the authentication. Depending on the display medium it chooses the correct way to authenticate the application against the BE. This is described in detail in section 5.3.4. The service also provides a subject that emits "true" when the application is authenticated. This signalizes that API calls to the protected endpoints can now be executed. In Angular subjects are the implementation of the Observer software-design pattern [Goob]. The official Angular documentation describes the Observer pattern as "a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of state changes" [Goob]. In Angular a subject can emit a value and all observers that subscribed to that subject earlier get notified with that value. By adding a "pipe" before subscribing this value can be altered by subsequently executed "operator functions". In this project all subjects are named with an "$" at the end, which is a popular approach in the Angular community to increase readability.
- Date Service: The date service holds helpful data like the currently displayed day, the current day and all relevant holidays. The current display date is stored in a subject which allows all subscribers to automatically react to changes regarding the display date. It also provides functions to help dealing with dates like calculating the calendar week of a specific date.
- Display-Context Service: This service extracts information from the Uniform Resource Locator (URL), which are theme, display medium and display team. It also figures out if the application is displayed on a smartphone and thus needs to adjust its views. It provides a subject called "displayContext" that emits every time one of the parameter changes.
- Employee Service: This service handles all HTTP requests regarding the employees of the customer.
- Team Service: This service handles all HTTP requests regarding the teams of the customer.
- User Service: This service handles all HTTP requests regarding the current user of the application. It also provides a "user$" subject that emits every time the user gets updated.

5.3.3 Components

Every Angular application has a root component. The root component in this application renders one of the four main components: "desktop", "mobile-login", "mobile-general" and "mobile-personal". Internally these components are called "view components" because they each cover the entire viewport and are swapped out as needed. This is done by a so called "router" that navigates between these components depending on different factors. The default route '/' links to the desktop

component. If the SPA is displayed on a smartphone, the router (notified by the display-context service) navigates to the mobile-login component ('/mobile-login). After a successful login the router navigates to the mobile-general component (/mobile-general).

On the other hand these view components can consist of smaller, reusable components each of which provides one GUI building block. Figure 5.5 gives an overview of this structure. These smaller components are subdivided into desktop, mobile and shared. Desktop components are only used in the desktop views while mobile components are only used in mobile views. Shared components can be used in both desktop and mobile views because they can adapt to different viewports.

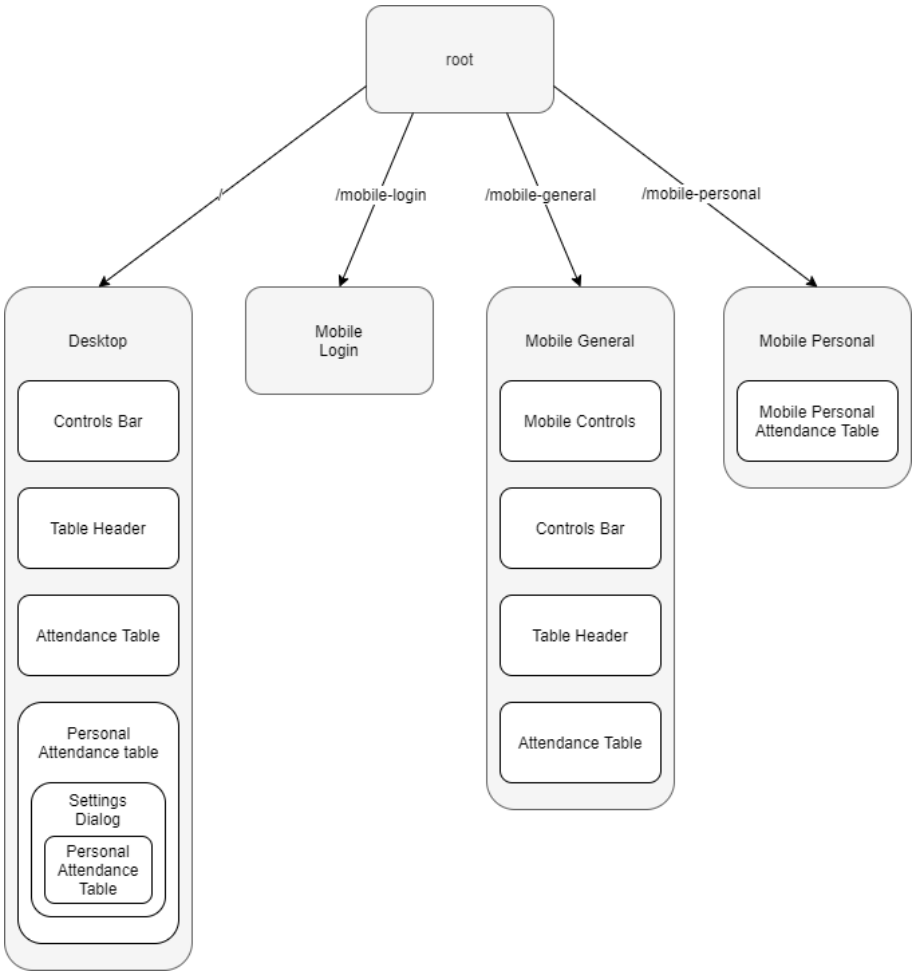


Figure 5.5: Component structure of the AUROA Angular application.

Currently there are two desktop-only components:

- Settings Dialog:
  This component holds all settings the user currently can configure.

  Figure 5.6 shows the GUI of this component. The “settings dialog” is a “mat dialog” which is basically a popup that hovers over the actual interface, which is provided by the Material Angular library. It consist of two dropdown fields for choosing the theme and the main team, and a “personal attendance table” component which is set to ’regular mode’.

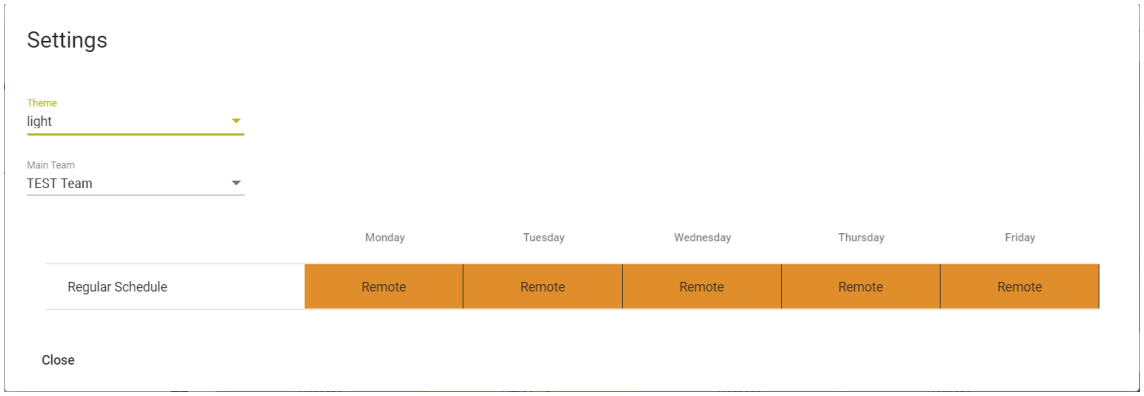


Figure 5.6: GUI of “settings dialog” component

- Personal Attendance Table:
  The “personal attendance table” shows the attendance date of the user that is logged in and lets him change it. From there the user can also open the “settings dialog”. When the user is authorized the component pulls the user data automatically.

  It has an input decorator that can switch between the “actual mode“ and the “regular mode”. The Angular input decorator is a way to share data between a parent and a child component. The actual mode allows the user to see his actual status (calculated from exceptions, regular schedule and holidays). This mode also enables the functionality to open the settings dialog. The regular mode, in contrary, allows the user to see his current regular schedule.

  Figure 5.7 shows the GUI of this component. In this case the input is set to ’actual mode’, which is why the settings symbol is visible. The settings button is located in this specific place because the section of the “personal attendance table” is supposed to hold all data and functionality that relate to the user. This is different to the initial GUI design because tests showed that a button to switch modes was not perceived as intuitive. To further clarify the displayed data all days that have an exception to the regular schedule are marked with an exclamation mark.

Figure 5.7: GUI of “personal attendance table” component

The mobile folder also consists of two components:

- Mobile Personal Attendance Table:
  This is the counterpart to the “personal attendance table”. It provides the same basic functions but has a GUI that is better suited for mobile viewports.

  Figure 5.8 shows the GUI of this component. It consists of three vertically aligned buttons to change a status.

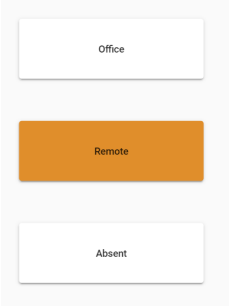
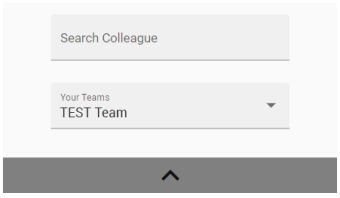


Figure 5.8: GUI of “personal attendance table mobile” component

- Mobile Controls:
  This component can show and hide any HTML content and thus helps to make better use of the limited space on mobile viewports.

  The GUI of this component is shown in figure 5.9. In figure 5.9b it can be seen in its closed state. Figure 5.9a shows how it displays the content, in this case the “controls bar” component.

(a) “mobile controls” component closed.



(b) “mobile controls” component opened.

Figure 5.9: GUI of “mobile controls” component.

Finally the shared folder contains three components:

- Attendance Table:
  The “attendance table” component shows the attendance status for a list of employees. It automatically adapts to different changes in the viewport by changing the amount of displayed days.

  It has an input decorator to tell which team or employee to show. This input is an object with an id and a mode which is either “team” or “employee”.

Figure 5.10 shows the GUI of this component. In the desktop version there is a column for the status of each workday of the current week and one for the names. On a mobile viewport only one weekday would be displayed.

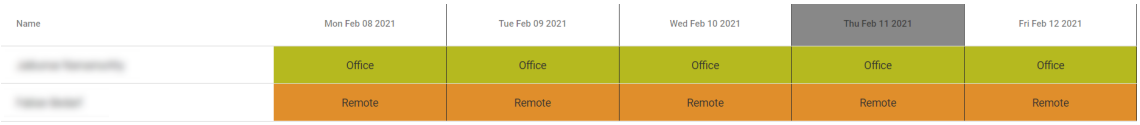


Figure 5.10: GUI of "attendance table" component

- Controls Bar:
  The "controls bar" provides user inputs for choosing a team or an employee and can trigger the log-in process.

  It provides two output directives to notify its parent component of changes in the respective record.

  Figure 5.11 shows the GUI of this component. Depending on its viewport the orientation of the inputs are either vertical (on mobile viewport) or horizontal (on desktop viewport) to take advantage of the shape of the displaying device. In the desktop version there is also the customers logo displayed in the top left

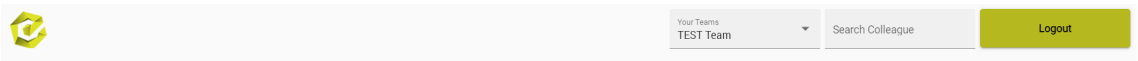


Figure 5.11: GUI of "controls bar" component

- Table Header:
  The "table header" modifies the globally available and currently selected date of the "date service".

  Figure 5.12 shows the GUI of this component. It displays the date and provides three user inputs to change it. One of these inputs is the "Angular Material Datepicker" which allows a fast navigation to dates that are far in the future or past. The other two inputs are arrows that navigate a specific number of days into the future or back the past. This number once again depends on the viewport.

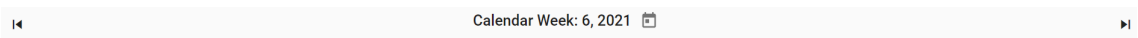


Figure 5.12: GUI of "table header" component

Table 5.1 provides an additional overview of which services get injected into the individual components.

Component	Injected Services
Desktop	Display Context, Team, Auth
Mobile Login	Auth
Mobile General	-
Mobile Personal	-
Personal Attendance Table	User, Date
Settings Dialog	User, Display Context
Mobile Controls	-
Mobile Personal Attendance Table	User, Auth, Date, Attendance Data
Attendance Table	Display Context, Date, Attendance Data
Controls Bar	Auth, Employee, User, Display Context
Table Header	Display Context, Date

Table 5.1: Overview of injected services.

5.3.4 Authentication

To implement the login process, the npm package “msal-angular” is used. “msal-angular” is a wrapper of “msal” specifically for Angular. Msal stands for “Microsoft authentication library” and is developed by Microsoft. It simplifies the usage of the OAuth standard to authenticate a user by using his Microsoft account.

The way the application authenticates the user is shown in figure 5.13. This is called the implicit flow. In step one the user gets redirected to “Authorization Server” in this case the AAD. Here the user can securely input his confidential data like his username and password because it is given directly to Microsoft and not to a third party. The authorization server is configured to only allow employees of the customer to login. If the login was successful the authorization server redirects back to the SPA with an identity token (step two). In step three the application tries to get an access token for the desired scope from the authorization server. A scope is a specific permission that the user gives the SPA. This is important because in most cases the user does not want the SPA to access and change all the data managed in his microsoft account. For this reason he specifically has to consent to every scope the application wants to get authorized for. In the case of AURORA the scope is defined in the AAD by registering the BE as an application. The SPA also gets registered and is configured to be authorized to request an token with the scope to access the BE. If the user consents to the scope (this is often done once trough an additional popup, which in this case is displayed after the first login) the authorization server returns the access token. Step four and five shows an example request to a server with protected endpoints. The application sends the access token in the HTTP “authorization” header and the backend tries to verify it. If it does so successfully it responds with the requested data.

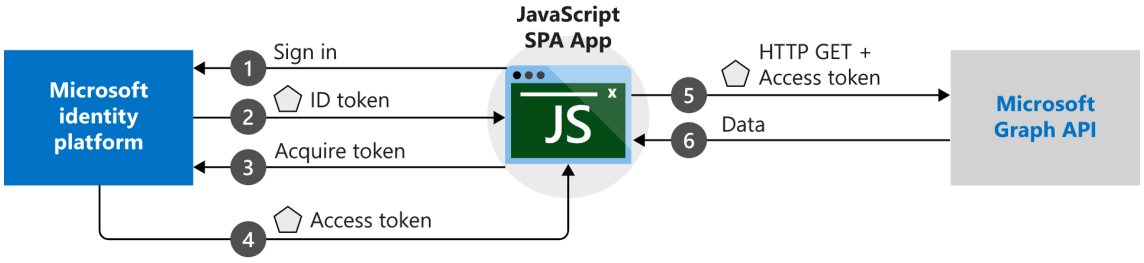


Figure 5.13: OAuth Implicit Flow [Micb].

While the Oauth 2.0 standard strongly recommends using the newer "authorization code flow" instead of the "implicit flow", "msal-angular" is not updated at the time of this thesis. In a discussion on github Jake McKennon, a software engineer at Microsoft, suggests to use "msal-angular" nonetheless and wait for the package to get updated [Jak].

However, in order for the authentication to work automatically, when display within MS Teams, an additional package is required. As already mentioned this is the package "'Microsoft Teams JavaScript SDK". This enables the application to call the method "microsoftTeams.authentication.getAuthToken()". When called within a MS Teams context this returns a JWT token. However, this token does not authorize the application to access the data in the BE because, by default, it does not include the required scope. To get a JWT with the required scope this token needs to get exchanged against a new token. This is done at the unprotected endpoint "api/token" because it can get pre-authorized in the AAD to get the required token and does not need additional approval. For this the BE gets a secret token. This is possible because the backend code (in contrast to the frontend code) is not accessible from the outside.

Internally the SPA manages these different authentication variants with the help of an "interceptor". The interceptor is an Angular tool which allows the interception and modification of every http call the application sends. Depending on the http parameter "displayMedium" this interceptor decides how to modify the request. In case of "standaloneBrowser" and "teamsTab" it checks for the required JWT in the browsers application storage. In case of "digitalSign" it swaps the public URL with the internal URL without protected endpoints. The TypeScript code responsible for this can be seen in figure 5.2. The parameter "req" is the intercepted HTTP request. "next" is an object that provides the function *handle()* which takes the modified request and continues the call. The big advantage of using an interceptor is that the entire authentication happens centralized in one place and does not have to be considered every time an HTTP call is sent.

Listing 5.2 The angular interceptor responsible for authentication.

```
intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    switch (this.contextService.getContext().displayMedium) {
        case DisplayMedium.DIGITAL_SIGN:
            return next.handle(req.clone({ \gls{url}: environment.internApi }));

        case DisplayMedium.STANDALONE_BROWSER:
            return this.msalInterceptor.intercept(req, next);

        case DisplayMedium.TEAMS_TAB:
            const accessToken = this.getRequiredAccessTokenFromStorageInTeams(req)
            return next.handle(req.clone({ setHeaders: { Authorization: `Bearer \${accessToken}`
                 } }));

        default:
            return next.handle(req);
    }
}
```

5.3.5 Workflow

After having introduced the main FE elements (components, services, interceptor), this chapter gives an example of the event-driven way of how those elements work together. Figure 5.14 shows an illustration of all events that result in a team being displayed inside the “attendance table” after the user logs in. It starts with the “user” service pulling the data of the logged in user from the server application. The “control bar” component is directly connected to this service because it always needs a logged-in user to provide its functionality. After receiving the user it selects a team (first in list or main team) and notifies its parent component. This component then provides a new “DisplayObject” to the “attendance table” component which contains the id. The “attendance table” component is not directly connected to the “user” service because it could (and does) also receive instruction from other sources. For example if the user searches for a single co-worker or if the team id gets provided by an URL Parameter. This way the component can be used more flexible. In the next steps the “attendance table” requests the attendance for the team and displays the results.

While this workflow can seem complex at first, it was developed with simplicity in mind. For example, any component can just subscribe to data provided by services and request data from services without worrying about authentication. Components that are designed to work with one specific data source are connected directly to the respective service (by injecting it into the component). But components that can receive data from multiple sources use inputs instead. In this case the parent component can control what the child component should display.

5.3.6 Testing

Because the FE was built with the angular Command Line Integration (CLI) the framework Karma was used for unit testing. Karma is the standard unit testing framework for angular. The unit tests can be execute with the command “ng test”.

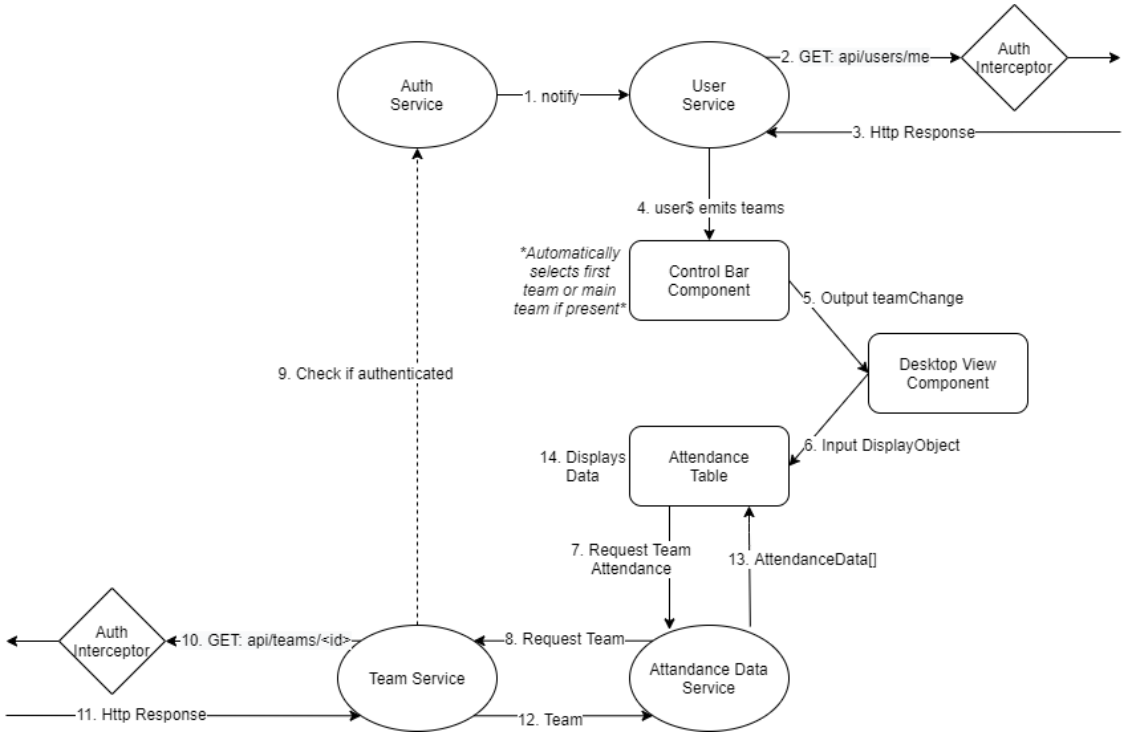


Figure 5.14: Example flow in the frontend after user logged in.

Due to timely limitations no end-to-end tests were written. End-to-end tests are tests that mimic the behavior of the end user. Therefore they belong to the “black-box” test group. However the angular CLI also sets up the end-to-end testing framework Protractor. This means that it would be easy to add such tests later if necessary.

5.4 PiSignage

The PiSignage Server was set up alongside the Gitlab server on the “aurora” VM. It also runs in a docker container. Team leads can create their own schedules for digital signs by navigating to a web surface which is available under the ip address of the “aurora” VM.

5.5 MS Teams App

The MS Teams app consists of two json files, one for the development environnement and one for the production environment. There are also two config files in the public folder of the server application.

5.6 Results

After the implementation phase the entire design was successfully implemented. Figure 5.15 shows the resulting system on different output devices. In the background is a comparatively big screen that is connected to a raspberry pi and functions as a digital sign. If you look closely you can see that all GUI blocks that require interaction are not displayed. In the foreground the standalone version can be seen on the different viewports of a laptop, a tablet and a smartphone.

Figure 5.17 shows the custom MS Teams tab. In this case it was decided to not display the “personal attendance table” component and the “controls bar” component. This way there is more space to display team members. Also it was suspected that in most cases, if an employee wants to change his settings, he would probably do it through the browser app anyway.

Figure 5.16 shows an example for a conversation with the MS Teams bot.

Figure 5.15: Picture of the finished software on multiple output channels.

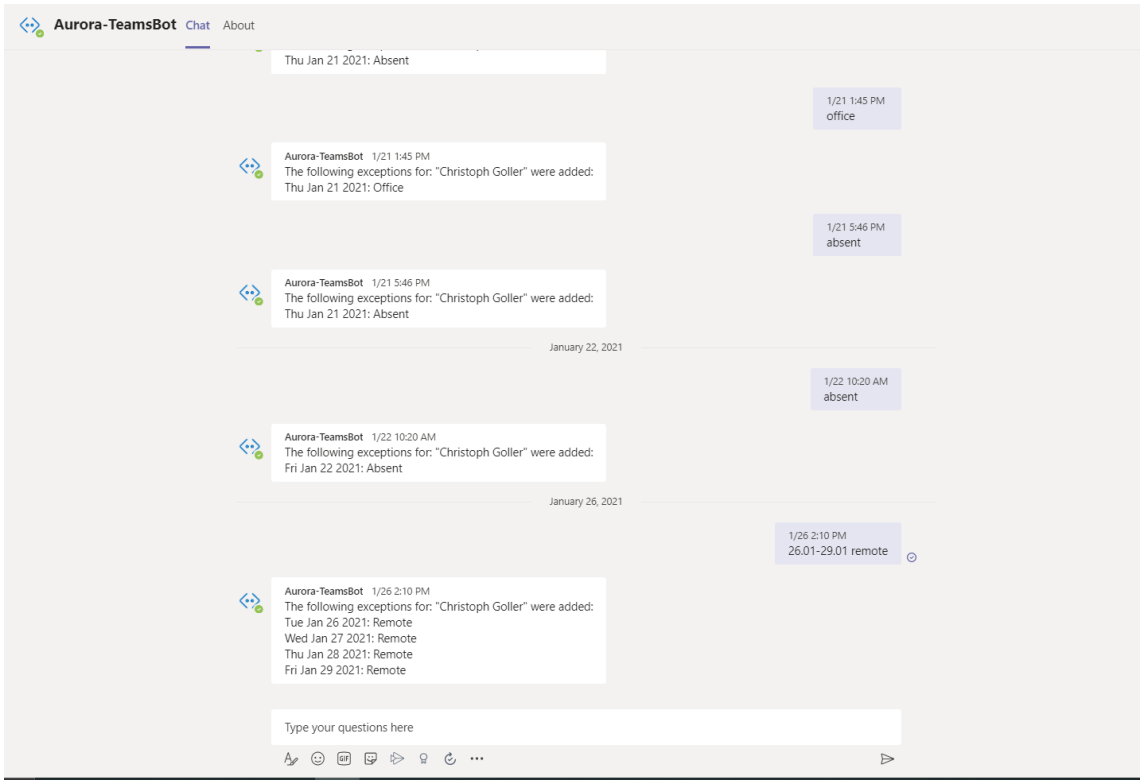


Figure 5.16: An example conversation with the MS Teams bot.

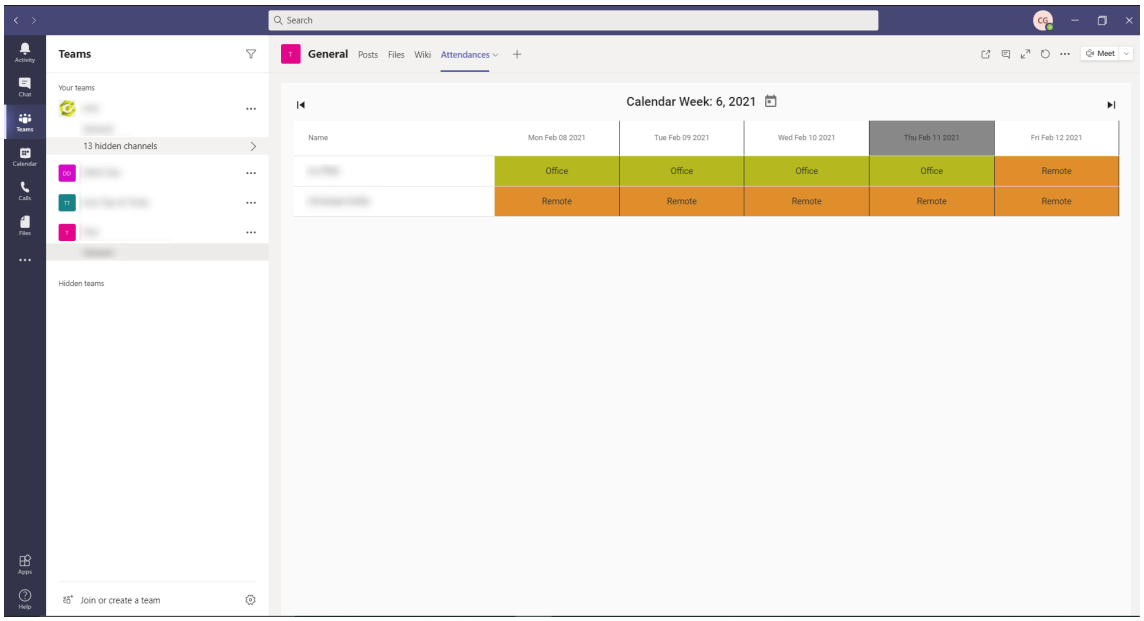


Figure 5.17: Screenshot of the MS Teams app.

6 Verification and Validation

In this chapter the software will be validated and verified.

6.1 Verification

To verify the software the question is asked: “Was the software built right”. This means to check if built software meets the requirements that were defined in the beginning.

The customer confirmed that each functional requirement was implemented. Thereby the software was successfully verified. In the beginning the functional requirements were divided into different priorities because it was expected that not all of them could be implemented. Thus the initial expectations were even exceeded.

6.1.1 Maintainability

This is a very hard metric to validate since there are not many meaningful indexes that evaluate maintainability. It can also be a subjective opinion from time to time. For this reason this non-functional requirement can not be verified precisely. However a big effort was put into keeping the system maintainable during the development and the design of the system. Primarily this was done by designing the server application as modular as possible. Modularity is an important characteristic of maintainability. All the modules were designed to work independently from each other. However it is worth noting that, because the monolithic approach was chosen, the modules still are executed on the same VM, docker container and NodeJs process. This means that one module crashing leads to all modules crashing.

Another attempt to keep the system maintainable consisted of writing intuitive and easy-to-read code. For this reason there were occasional code reviews from independent developers. But while there are generally accepted patterns for good code, on some points this can be very subjective.

6.1.2 Security

Another important non-functional requirement was security. To better assess the achieved degree of security an employee at evia conducted a penetration test. A penetration test tests an application regarding its resistance against attacks that aim to steal data.

The tester actually found a way to update the entries of users other than himself. This was possible because the authentication library can extract the token from the authorization header and the body. However the custom function, which was written to extract the id from the token, always used the

token from the authorization header and did not additionally verify the signature of this token. This circumstance made it possible to “ bypass the route authorization by using a valid jwt token inside the request body and simultanously providing a malicious jwt token inside the Authorization request header”. This bug was fixed as part of a maintainance update.

However this could only be done by a customers employee because a valid token was still needed. From outside no security risks could be found. Because the main goal was to prevent a data leak, the customer decided that this non-functional requirement was successfully implemented. Since every employee can see the data of every co-worker anyway, an internal data leak is not a true data leak in this case.

6.1.3 Usability

Again, this is a hard metric to validate. In an effort to increase usability the GUI was tested by independent testers even during the implementation phase. The feedback obtained by these testers lead to some adjustments in the GUI.

One of this adjustments is the addition of the settings dialog after testers found a button to switch between the two modes confusing. Another minor adjustment was the shift of the team selection and the single person search from the left to right because it felt more natural to the users.

6.2 Validation

To validate the software the question is asked: “Was the right software built”. This means to check if the software meets the end users needs and expectations.

In fact this proofed to be difficult because during the validation period nearly 100% of the employees were working from home. Therefore an application to monitor office attendance currently has only limited use. Nonetheless there are some teams at evia where the responsible people have already introduced AURORA into their teams. This means that, at least from the team leads, there is the expectation that AURORA will help them solve a real problem.

After showing the final result to the CEO of the customer he wrote the following: “I especially like the seamless integration with our project planning software projectfacts and our communication platform MS Teams. This ensures that the new tool is really accepted by our employees.” (freely translated).

7 Conclusion

Because the software is already validated and verified, this section is solely used to discuss possible future work, challenges and the overall course of the project.

Overall the project has taken a very good course. There were no major setbacks and progress was continuously made. All steps of the waterfall model were finished. The superior goal to develop a proof of concept application was achieved.

According to the chosen SDLC model the first step was to precisely define the requirements. From the beginning it was clear that digital signs should be used as an output medium. However after a short analysis of earlier solutions it became clear that additional channels would be required. Additionally the way to enter data into the system needed to be defined. For this reason a survey among the employees of the customer was conducted. After analyzing the result it was decided to split the requirements into different priorities.

In the designing phase it was decided to develop a single page web application. The intent behind this decision was to decrease development time because of its portability. The most important technologies chosen were: NodeJs, Angular, Typescript and MongoDB.

During the development phase, the final GUI of the SPA was designed and the code was written. To increase the maintainability a CI/CD system was set up. This system consists of two different environments (dev and prod). If there will be any bug fixes or new features in the maintainance phase, this allows easy testing and deployment. Simultaneous to the functional code, unit tests were written for a better quality control.

After the development the system was successfully verified and validated. This was mostly done in meetings with the customer to ensure that all implemented requirements were understood correctly. Software validation in particular proofed to be difficult because the current circumstances (COVID-19) were limiting the usability of the system.

7.1 Challenges

Even though the project went well overall, there were a few minor challenges that had to be overcome. One challenge during this thesis was the COVID-19 pandemic. For this reason the office work had to be reduced by nearly 100%. However this was not such a challenge because most work could actually be done from home.

The project covered many different areas of software development including: frontend development, backend development, security, GUI design, OAuth and more. The biggest challenge probably was that, as an result, many new technologies had to be learned during the development. The two biggest new technologies probably were NodeJs and OAuth. But at the same time this was also one of the most profitable circumstances in the end.

Another challenge was that the scope of the requirements was much bigger than initially anticipated. It was clear that the requirements will only be defined during the duration of this thesis and thus the resulting estimate of effort can vary. However initially it was not expected to cover that many input and output channels. For this reason the priority system was set in place to create a bigger buffer for development. But in the end all requirements could be implemented nonetheless.

7.2 Future Work

This software has a huge potential for add-ons. On one hand the digital signs infrastructure has great potential for additional content. They could be used to display occupancy of meeting rooms, company news, lunch menus, to just name a few.

On the other side the actual software also holds potential for improvements. There could potentially be additional views to give more insight about individual co-workers like their main office or a telephone number for example. Another possible example is the dynamic MS Teams status discussed in section 3.1.1. But these are only some possible ideas. There are already ideas from the customer to expand the AURORA system so that it can be used to monitor parking space.

Acknowledgment

First of all I would like to thank Prof. Dr. Wagner for the opportunity to write my bachelor thesis at the Institute of Software Engineering at the University of Stuttgart.

Secondly, I would also like to thank my supervisor Jinghua Yu, who always found time for my questions and whose constructive comments I appreciated very much.

I would also like to thank the company evia and, in particular, my supervisors Cornelius Rusch and Ivo Pfeil for their relentless support throughout all stages of this project.

But most of all I would like to thank my parents who made this study even possible in the first place.

Bibliography

[AB15] A. Alshamrani, A. Bahattab. "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model". In: *IJCSI International Journal of Computer Science Issues* 12.1 (1 2015) (cit. on p. 17).

[Bec03] K. Beck. *Test-Driven Development*. Addison-Wesley, 2003 (cit. on p. 20).

[Ben03] Bender RPT Inc. *Systems Development Life Cycle: Objectives and Requirements*. 2003. URL: http://www.benderrbt.com/Bender-SDLC.pdf (cit. on p. 17).

[Boe81] B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981, pp. 6–7 (cit. on p. 20).

[BP 78] G. T. B.P. Lientz E.B. Swanson. "Characteristics of Application Software Maintenance". In: *Communications of the ACM* 21.6 (1978) (cit. on p. 20).

[Car] R. Carr. *Gang of Four Design Patterns*. URL: http://www.blackwasp.co.uk/gofpatterns.aspx (visited on 10/06/2020) (cit. on p. 19).

[Doc] Docker Incorporated. *Docker Website - Containers*. URL: https://docker.com/resources/what-container (visited on 09/30/2020) (cit. on p. 32).

[EV06] J. Eveleens, C. Verhoef. "The Standish Report: Does It Really Describe a Software Crisis?" In: *Communications of the ACM* 49.8 (2006), pp. 15–16 (cit. on p. 18).

[EV08] J. Eveleens, C. Verhoef. "The rise and fall of the Chaos report figures". In: *IEEE Software* (2008), pp. 30–36 (cit. on p. 18).

[EV94] R. J. Erich Gamma Richard Helm, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994 (cit. on p. 19).

[Gooa] Google LLC. *Angular Website - Features*. URL: https://angular.io/features (visited on 10/26/2020) (cit. on p. 38).

[Goob] Google LLC. *Angular Website - Observables*. URL: https://angular.io/guide/observables (visited on 01/26/2021) (cit. on p. 49).

[IBM] IBM. *The Mean Stack*. URL: https://www.ibm.com/cloud/learn/mean-stack-explained (visited on 01/08/2021) (cit. on p. 38).

[IDC] IDC Technologies. *SDLC Waterfall Model*. URL: http://www.idc-online.com/technical_references/pdfs/information_technology/Sdlc_Waterfall_Model.pdf (cit. on p. 17).

[ISO] ISO/IEC. *ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. URL: https://www.iso.org/standard/35733.html (cit. on p. 28).

[Jak] Jake McKennon. *GitHub discussion about angular msal*. URL: https://github.com/AzureAD/microsoft-authentication-library-for-js/issues/2329 (visited on 01/08/2021) (cit. on p. 55).

[Jas] Jasmine. *Jasmine Webpage*. URL: https://jasmine.github.io/ (visited on 01/08/2021) (cit. on p. 44).

[LB10] A. Lee, T. Brewer. "Guidelines for smart grid cyber security: Vol. 1, smart grid cyber security strategy, architecture, and high-level requirements". In: *NISTIR* 1 (2010), pp. 4, 25 (cit. on p. 29).

[MB96] A. Munns, B. F. Bjeirmi. "The role of project management in achieving project success". In: *International Journal of Project Management* 14.2 (1996), pp. 81–87 (cit. on p. 17).

[MD13] A. Mishra, D. Dubey. "A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios". In: *International Journal of Advance Research in Computer Science and Management Studies* 1 (5 2013) (cit. on p. 17).

[Mica] Microsoft. *Azure Active Directory - Features*. URL: https://azure.microsoft.com/de-de/services/active-directory/#features (visited on 10/27/2020) (cit. on p. 39).

[Micb] Microsoft. *Microsoft Docs - Implicit Flow Picture*. URL: https://docs.microsoft.com/en-us/azure/active-directory/develop/media/tutorial-v2-angular/diagram-auth-flow-spa-angular.svg (visited on 02/09/2021) (cit. on p. 55).

[Micc] Microsoft. *MS Teams SDK - Authentication*. URL: https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/authentication/authentication?view=msteams-client-js-latest (visited on 10/27/2020) (cit. on p. 39).

[Mon] MongoDB. *MongoDB Docs - Data Model*. URL: https://docs.mongodb.com/manual/core/data-model-design/ (visited on 10/27/2020) (cit. on pp. 35, 36).

[Moz] Mozilla Foundation. *Mozilla Web Glossary: SPA*. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (visited on 10/26/2020) (cit. on p. 38).

[npm] npm. *npm - express*. URL: https://www.npmjs.com/package/express (visited on 11/10/2020) (cit. on p. 34).

[Ora] Oracle. *Data Access object*. URL: https://www.oracle.com/java/technologies/dataaccessobject.html (visited on 01/08/2021) (cit. on p. 44).

[PiS] PiSignage. *Pisignage Docs*. URL: https://pisignage.com/homepage/documentation.html (visited on 10/27/2020) (cit. on p. 40).

[Pri] Principles Wiki. *KISS 2*. URL: http://principles-wiki.net/principles:keep_it_simple_stupid (visited on 01/07/2021) (cit. on p. 33).

[Roy70] W. W. Royce. "Managing the development of large software systems". In: *Proceedings of IEEE WESCON26* (1970), pp. 1–9 (cit. on p. 17).

[Rup10] N. B. Ruparelia. "Software Development Lifecycle Models". In: *ACM SIGSOFT Software Engineering Notes* 35.3 (2010) (cit. on p. 17).

[Smi] B. Smith. *KISS 1*. URL: http://www.xdoubledot.com/documents/How%20to%20KISS.pdf (visited on 01/07/2021) (cit. on p. 33).

[Sta] Standish Group. *CHAOS REPORT 2011*. URL: https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/GENREF/ChaosManifest_2011.pdf (cit. on p. 18).

[VGC+15] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, S. Gil. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud". In: *2015 10th Computing Colombian Conference (10CCC)*. IEEE. 2015, pp. 583–590 (cit. on p. 33).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature