FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	"Chronosite" – application for the timetable slots assigning
Student:	Yaroslav Borovyk
Supervisor:	Ing. Ondřej Guth, Ph.D.
Study program:	Informatics
Branch / specialization:	Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2023/2024

Instructions

Design and implement a prototype of a tool for assigning teachers to the existing timetable slots. The application has to:
1. Provide a web interface both for teachers and the administrator.
2. Gather constraints from teachers, and use them for computation of the possible assignments of teachers to slots.
3. Display possible assignments to the administrator.

The project should minimize the administrator's work and also allow future integration with university information systems.

Electronically approved by Ing. Michal Valenta, Ph.D. on 6 February 2023 in Prague.

Bachelor's thesis

"CHRONOSITE" – APPLICATION FOR THE TIMETABLE SLOTS ASSIGNING

Yaroslav Borovyk

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Ondřej Guth , Ph.D.
May 11, 2023

Czech Technical University in Prague
Faculty of Information Technology
© 2023 Yaroslav Borovyk. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Borovyk Yaroslav. *"Chronosite" – application for the timetable slots assigning.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

List of Tables

List of code listings

Firstly, I want to express my heartfelt thanks to Ing. Ondřej Guth, Ph.D., for his invaluable assistance and invaluable guidance while I was writing my thesis. I am also grateful to my family for their support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 11, 2023 ....

Abstract

The thesis provides analysis, design, implementation, and testing of a prototype application that assigns teachers to timetable slots based on their preferences. The tool consists of a server written in Java and web interfaces for admins and teachers, implemented using Thymeleaf. The purpose of the system is to receive preferred schedule options from teachers and generate possible schedules satisfying these options. The functional module's core part, a backtracking-type algorithm, performs the parallel and asynchronous computation of timetables.

The thesis begins with a general analysis of the problem, including its primary aspects, functional and non-functional requirements, use cases, investigation of existing solutions, and domain model. The subsequent chapters cover the prototype's design and implementation. Finally, the thesis finishes with concepts of future improvements based on handled user testing of the principal functionality.

Keywords prototype application, teacher scheduling, backtracking algorithm, parallel and asynchronous computation, Java, Spring

Abstrakt

Práce obsahuje analýzu, návrh, implementaci a testování prototypu aplikace, která přiděluje učitelům rozvrhová místa na základě jejích preferencí. Nástroj se skládá ze serveru napsaného v jazyce Java a webového rozhraní pro administrátory a učitele, implementovaného s využitím šablonového enginu Thymeleaf. Účelem systému je přijímat od učitelů preferované možnosti rozvrhu a vytvářet možné rozvrhy, které tyto možnosti splňují. Základní část funkčního modulu, algoritmus typu backtracking, provádí paralelní a asynchronní výpočet rozvrhů.

Práce začíná obecnou analýzou problému, včetně jeho primárních aspektů, funkčních a nefunkčních požadavků, případů užití, zkoumání existujících řešení a doménových modelů. Následující kapitoly se zabývají návrhem a implementací prototypu. Nakonec práce končí koncepty budoucích vylepšení na základě provedeného uživatelského testování hlavní funknosti.

Klíčová slova prototypová aplikace, rozvrhování učitelů, algoritmus backtracking, paralelní a asynchronní výpočet, Java, Spring

List of abbreviations

UI	User Interface
UX	User Experience
NP	Nondeterministic Polynomial Time
WWW	World Wide Web
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
SQL	Structured Query Language
NoSQL	Not Only Structured Query Language
ROP	Role Object Pattern
MVC	Model-View-Controller
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
XHTML	EXtensible HyperText Markup Language
XML	Extensible Markup Language
CSRF	Cross-Site Request Forgery
LDAP	Lightweight Directory Access Protocol
JVM	Java Virtual Machine
JSP	JavaServer Pages
JPA	Java Persistence API
JMS	Java Messaging Service
EJB	Enterprise Java Beans
EE	Enterprise Edition
NPM	Node Package Manager

Chapter 1
Introduction

The following chapter introduces the problem and motivation that inspired the author to choose a proposed topic. Additionally, it outlines the objectives and the structure of the thesis.

1.1 Motivation

In every aspect of our lives, we are often tasked with managing multiple things, but managing people is undoubtedly one of the most challenging responsibilities. It requires time, effort, and understanding of people's thoughts and emotions. What if we could simplify this process and make it less challenging?

The answer is simple and obvious – a solution like this would be beneficial in many daily scenarios and there are certainly tools and solutions that can help make managing people easier. With this motivation, the author decided to create a tool that simplifies the process of creating timetables while considering people's preferences and availability.

The majority of existing solutions to this problem are private applications with complex architecture. Drawing from experience with these applications, the author attempted to create a "Chronosite" – the tool that can gather people's preferences and automate the process of assigning people to timetable slots.

1.2 Thesis objectives

The primary objective is to analyze the problem of assigning people to the time slots, and design and implement an application that simplifies the process of collecting information from teachers regarding their preferences and limitations for scheduling. By using this data, the system will automate the processes of receiving the data from people and assigning them to the timetable slots. In detail, it will generate possible schedules that will be satisfactory for everyone involved, making the process of creating a timetable easier for the person in charge. The additional objectives are testing the prototype and formulating possible enhancements to the system.

1.3 Thesis structure

The thesis has the following structure:

- **“Analysis”** – formulates a common analysis of the problem and identifies its functional and non-functional requirements. This includes describing use cases, and domain model, as well as analysis of existing solutions.
- **“Design”** – focuses on defining a design of a functional prototype that addresses the problem effectively. Describes the author’s architecture solutions for the system, including algorithms, data management, and UI.
- **“Implementation”** – represents the technical realization of the tool, based on the key statements discussed in the previous chapters.
- **“Testing”** – demonstrates the testing process of the developed application and outlines future improvements based on the test results.
- **“Conclusion”** – provides a summary of the achievements in meeting the objectives of the thesis, along with an overview of the resulting prototype.

Chapter 2

Analysis

This chapter will provide a general analysis of the problem by focusing on the common subproblems of the timetable creation process. It will also introduce the functional and non-functional requirements, use cases, domain model, and existing solutions for the "Chronosite" that will be implemented.

2.1 Timatable Creation Process

The author decided to start the following chapter with an analysis of the current process of creating timetables. Nowadays, receiving preferred schedules from teachers and generating timetables with assigned teachers are mostly manual tasks, so it is important to define the roles and the steps of this process.

2.1.1 General roles

At first, the timetable creation process divided the involved people into 2 specific groups. They are either teachers or the person who is in charge of the creation of the timetable (administrator):

- **Administrator** – a person who controls all the processes such as creating the timetable with time slots, receiving the suitable schedules for the teachers and communicating with them, and choosing the most optimal way how to assign teachers to the existing time slots considering their preferences.
- **Teacher** – an individual who reviews the created timetable by the administrator and provides the availability for the time slots, preferable schedules, or limitations on the timetable. Also, they provide feedback on the created schedule by the administrator whether they accept or reject it.

An important point regarding the communication between teachers and administrators is that the teachers usually provide unstructured preferences. The teachers submit sometimes abstract preferences, for example, "I do not want to take more than 3 time slots in the raw on any day of the week". The author decided to state later in this chapter the precise rules for the incoming data from the teachers.

2.1.2 Process steps

The following subsection describe several concrete steps that should be performed:

1. **Gather information** – the order of priority regarding course selection during scheduled timings is then determined based on the gathered information, such as course offerings, teacher availability, classroom availability, and other details that can be useful during the scheduling process.
2. **Create draft schedule** – a preliminary schedule will be created, taking into account all factors mentioned in the previous step. Also, this step can be repeated several times after the review from the teachers, or the draft schedule can be updated.
3. **Review draft schedule** – the draft schedule will be reviewed and revised as needed to ensure that it meets the needs of all parties involved.
4. **Finalize the schedule** – once the schedule has been reviewed and revised, a final version will be created and distributed to students and teachers.

These steps describe the concept of the timetable creation process. Gathering information, and generating and reviewing schedules are tasks that can be sorted out using online tools or services, such as email for communication. However, the concept of automating this nontrivial process should aim to automate each step as much as possible.

2.1.3 Succefully created timetable

So, how the successfully created timetable can be described? The successfully created timetable should follow the listed requirements:

1. The logical amount of teachers should be involved and assigned to the timetable slots.
2. The created timetable should match the preferable schedules of the involved teachers.
3. The created timetable should be approved by the administrator and the teachers that were assigned to the time slots.

If the created timetable adheres to the all stated requirements, then it can be considered as a successfully created timetable. A detailed description of the optimizations or rules of the timetable generation process in the “Choronosite” application is presented in the next chapters of the thesis.

The next sections will focus on the requirements, existing solutions, use cases, and domain model of the ”Chronosite” application, which is related to the process defined in this section.

2.2 Incoming data

Before establishing the requirements and use cases, the author decided to describe the rules for the data that will be received from the teachers.
Such rules are highly sensitive to organizational differences as useful data can vary between institutions. The author has defined basic rules for data that will be submitted by the teachers to the application, including:

1. Time slot preferences – teachers should be able to accept or reject each time slot individually. More complex options, such as preferred colleagues to teach with, can also be added.
2. Range of taken time slots – teachers should be able to specify a minimum and a maximum number of time slots to take per day or week.
3. Various number of the preferable schedules – teachers should be able to create any number of preferred timetables.

While there may be additional rules specific to certain institutions, the author believes that these basic rules cover a large percentage of teachers' preferences. The time slot operations provide flexibility to create any desired combinations of accepted and rejected time slots, and the range of taken time slots allows teachers to generalize their preferences. Finally, the teachers would be able to create a pool of the preferred schedules on different timetables provided by the administrator.

2.3 Chronosite requirements

"Requirements" – a popular noun that people use in different contexts across various areas of life. Firstly, it is vital to define "requirements" in the context of software engineering:

"Requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system."[1]

This definition acknowledges the diverse types of information that collectively are referred to as "the requirements." Requirements encompass both the user's view of the external system behavior and the developer's view of some internal characteristics. They include both the behavior of the system under specific conditions and those properties that make the system suitable—and maybe even enjoyable—for use by its intended operators.[1]

Furthermore, this chapter formulates functional and non-functional requirements based on the objectives of the thesis.

2.3.1 Functional Requirements

Functional requirement – a description of behavior that a system will exhibit under specific conditions.[1] Functional requirements have been established for the "Chronosite", as follows:

- **F1: Admin profile** The system provides an administrator role. Predefined administrators can create teachers' profiles (based on username and password), manipulate schedules with time slots, and generate schedules.
- **F2: Teacher profile** Created teachers can use the application with limitations. Mainly, they can submit their preferred timetables to the system and send feedback about the generated schedule to the administrator.
- **F3: Display schedule** The system displays schedules with their time slots to the users (administrators and teachers).
- **F4: Constraints on timetable** Teachers can choose suitable time slots to pick. Also, it should be possible to manipulate the number ranges of the picked time slots for the day or week. It should be possible to choose any number of time slots in all possible ways. By default, all time slots are selected by the teacher.
- **F5: Variety of the preferable timetables** Teachers can create any number of preferred schedules. All of them should be used in the phase of schedule generation.
- **F6: Validation of the preferable timetables** The suitable schedule should be validated by the creator (teacher). It should be possible to validate the changed preferred timetable after the original timetable changes.

- **F7: Schedule generation** The system should generate all possible timetables with the assigned teachers to the timetable slots, considering validated preferred schedules from the teachers. The generation of the timetable should be available only if each teacher has submitted at least one preferred timetable. Generated timetables should be saved for future usage.

- **F8: Review of the generated schedule** Users can approve or reject generated schedules. The administrator should approve one possible schedule in the first place. All teachers should also approve the same schedule. If any teacher rejects the proposed schedule, then it becomes unapproved automatically. Additionally, teachers can write a comment on the provided schedule regardless of their choice (approve or reject).

- **F9: Notifications** The system should send notifications to the administrators. These notifications should contain information about the teachers' feedback and schedule states.

2.3.2 Non-functional Requirements

Non-functional requirement – a description of a property or characteristic that a system must exhibit or a constraint that it must respect.[1] "Chronosite" is expected to meet the following non-functional requirements:

- **N1: Client** The website is expected to be designed using modern web technologies and front-end frameworks to ensure an effective UI. It should be responsive and easily accessible on various devices, such as laptops, desktop computers, tablets, and smartphones.

- **N2: Server** The server should be running using the Java application using modern programming languages and frameworks. Also, It has to be integrable with other systems or services.

- **N3: Maintainability** It's important to consider the system's scalability to integrate future changes and updates. This means that the system should be able to easily adapt to progressive technologies, new functionalities, and changes to the domain model while maintaining its performance and stability.

- **N4: Performance** Considering the potentially high time complexity of the schedule generation problem, it is necessary to ensure that the schedule generation operations are optimized to execute within a rational timeframe.

- **N5: Security** The system should provide secure authentication and authorization processes. Additionally, it should use technologies and frameworks that will have features like password hashing, encryption, and access control mechanisms that can help maintain the system's security.

2.4 Existing solutions

The investigation of the existing fields in software engineering requires to make research on working applications, implemented concepts, and talented people.

For the study purposes of the existing solutions that solve the same problem, the author investigated the WWW using different search engines.
Unfortunately, there do not exist many public programs related to the problem itself. To be more precise, only **Doodle**, as a public website, can be compared to "Chronosite" in terms of existing solutions.

2.4.1 Doodle

“Doodle is an online calendar tool for time management and coordinating meetings. Users are asked to determine the best time and date to meet. The organizer then chooses the time that suits everyone and the meeting is booked in the user’s calendar. Meeting coordinators (administrators) receive e-mail alerts for votes and comments.”[2]

Key features:

- **Poll creation:** The users can easily create polls to find the most suitable date and time for a meeting or event. The poll creator has the flexibility to propose multiple options for dates and times, and your participants can vote on their availability. The same logic contains the requirements F1, F2, and F4. The administrator will create a timetable with time slots and the teachers will choose their preferred time slots.
- **Real-time updates:** When participants respond to the poll, the poll creator can view the results in real time. As stated in the F4, teachers will submit their preferable schedules to the system and the administrator can track submissions during the timetable generation process.
- **Suitable date and time detection:** The poll creator can manually determine the most suitable date and time for the meeting using real-time updates which is similar to requirement F7, but the difference will be described later in this chapter.

Overall, **Doodle** application has a user-friendly and intuitive interface and efficient communication options. Moreover, it solves the problem of agreeing on the date and time for the events.
Distinctions:

- **Participant preferences:** The participants can only submit one suitable option for the specific event in the poll. “Chronosite” provides the functionality to create multiple preferred options for one schedule.
- **Possible poll generation:** The Doodle provides the ability for the poll creator to manually assign participants to a specific event. “Chronosite” can collect preferred schedules and automatically generate all possible timetables.
- **Approvement by participants:** The Doodle does not provide the functionality to submit feedback to the poll creator or reject the finished poll with assigned people for the events. However, “Chronosite” offers the possibility to accept or reject the generated schedule and submit feedback.

2.5 Use cases

The upcoming chapter describes the use cases of the application, but first, the meaning of the term *“use case”* in the software engineering context must be defined:

*“A **use case** describes a sequence of interactions between a system and an external actor that results in the actor being able to achieve some outcome of value.”*[1]

The following use cases have been defined for ”Chronosite”. They are divided into 2 categories: admin and teacher use cases.

2.5.1 Admin use cases

This section primarily centers on the administrator’s use cases regarding manipulations of the teachers, schedules, and time slots and they are represented in Figure 2.1, but the role of the administrator should be generally described first.

ADMIN – has access to operations such as creating, updating, and deleting teachers, timetables, and timetable slots. It also provides permission to generate possible schedules, decide which generated schedule will be approved, and used it as the blueprint for assigning teachers to the original timetable slots. Additionally, the administrator receives notifications regarding the timetable creation process.

- **UC1: Create teacher** The administrator should be able to register teachers using usernames and passwords created by the administrator.
- **UC2: Update teacher** The administrator should be able to update teachers' credentials using their existing username.
- **UC3: Delete teacher** The administrator should be able to delete registered teachers using their existing username.
- **UC4: Create timetable** The administrator should be able to create new timetables by submitting their future names.
- **UC5: Edit timetable** The administrator should be able to edit timetables by performing various operations on the time slots of the selected timetable.
- **UC6: Delete timetable** The administrator should be able to delete timetables through manual interaction with the system.
- **UC7: Create time slot** The administrator should be able to create time slots by submitting their future start and end time.
- **UC8: Update time slot** The administrator should have the ability to manually edit the time slots of selected schedules. This means that the administrator should be able to change the start or end time of the selected time slot.
- **UC9: Delete time slot** The administrator should have the option of manually removing the time slots from chosen schedules.
- **UC10: Receive notification** The administrator should be able to receive notifications from the system regarding a specific schedule state or notifications regarding the teacher's feedback.
- **UC11: Read the message of notification** The administrator should be able to read the more detailed message of the notification if it exists.
- **UC12: Delete notification** The option to manually remove notification should be available to the administrator.
- **UC13: Browse schedules** The administrator should be able to browse through created timetbles.
- **UC14: Browse time slots** The administrator should be able to browse through created time slots in the selected schedule.
- **UC15: Browse notifications** The administrator should be able to browse through received notifications.
- **UC16: Generate schedule** The administrator should be able to generate all possible schedules. It is possible if all teachers have submitted at least 1 verified preference of the schedule.

- **UC17: Approve generated schedule** The administrator should be able to approve generated schedule. Doing so, all other generated schedules were deleted, and chosen generated schedule is available for approval by the teachers.

2.5.2 Teacher use casaes

This subsection focuses on the teacher's use cases involving the interaction with schedules, preferred schedules, and time slots. They are described in Figure 2.2, also the description of the teacher role is stated as follows:

TEACHER – has access to the timetables created by the administrator and can create the preferable schedules that will the most suit the availability of the teacher. Also, the teacher can perform the operations on the created preferable schedule as edition and validation. Moreover, the teacher can send feedback to the administrator regarding the approved generated schedule by the administrator.

- **UC18: Edit profile** The teacher should be able to edit the profile and credentials which were created by the administrator.
- **UC19: Create preferable schedule** The teacher should be able to create many preferable schedules on the chosen timetable created by the administrator by passing the name of the future suitable schedule.
- **UC20: Validate preferable schedule** The teacher should be able to confirm the preferred schedule they have created.
- **UC21: Edit preferable schedule** The teacher should be able to edit the preferred schedule that they have created. Moreover, they should have the option to update their preferences for specific time slots and ranges for each day and week.
- **UC22: Delete preferable schedule** The teacher should be able to delete their preferred schedules from the selected schedules that were created by the administrator, by manually interacting with the system.
- **UC23: Browse schedules** The teacher should be able to browse through the schedules that were created by the administrator.
- **UC24: Browse preferable schedules** The teacher should be able to browse through the preferred schedules they have created.
- **UC25: Review generated schedule** The teacher should be able to review the schedule that has already been generated and approved by the administrator before deciding to accept or reject the presented schedule.
- **UC26: Approve generated schedule** The teacher should be able to approve the schedule that has already been generated and approved by the administrator. By doing so, the teacher can provide feedback to the administrator.
- **UC27: Reject generated schedule** The teacher should be able to reject the already approved generated schedule by the administrator. By doing so, the teacher can send feedback to the administrator, and the generated schedule is no longer approved by the administrator and unavailable to the other teachers.

2.5.3 Summary

To summarize how the use cases related to the functional requirements, the author decided to provide a short description of the correlation between the use cases and functional requirements and the Table 2.1:

- The **admin profile** requirement is covered by the use cases related to the operations of the teachers, timetables, and time slots.
- The **teacher profile** requirement is correlated to the use cases that describe the manipulations on the preferable schedules and the approval or rejection of the generated schedules by the administrator.
- The **display schedule** requirement is covered by the use cases that are linked to the browsing of the timetables by users.
- The **constraints on timetable**, **validation of the preferable timetables**, and **variety of the preferable timetables** requirements are correlated to the use cases that are related to the operations on the teachers' preferable schedules.
- The **schedule generation** requirement is covered by the use cases that describe the schedule generation process performed by the administrator.
- The **review of the generated schedule** requirement is correlated to the use cases that are related to approval or rejection by users.
- The **notifications** requirement is related to the use cases that describe the connection between the administrator and the notifications.

Table 2.1 Functional requirements covered by use cases

Functional requirements	Use cases
F1	UC1-UC9, UC16
F2	UC19, UC21, UC22, UC26-27
F3	UC13, UC23, UC25
F4	UC19, UC20, UC22
F5	UC19
F6	UC20,UC21
F7	UC16
F8	UC17, UC25-UC27
F9	UC10-UC12

Figure 2.1 Admin use cases

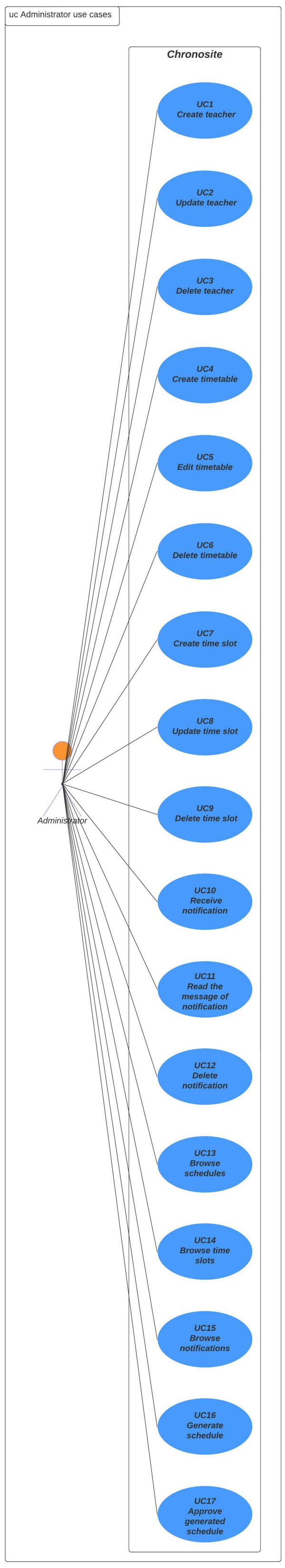
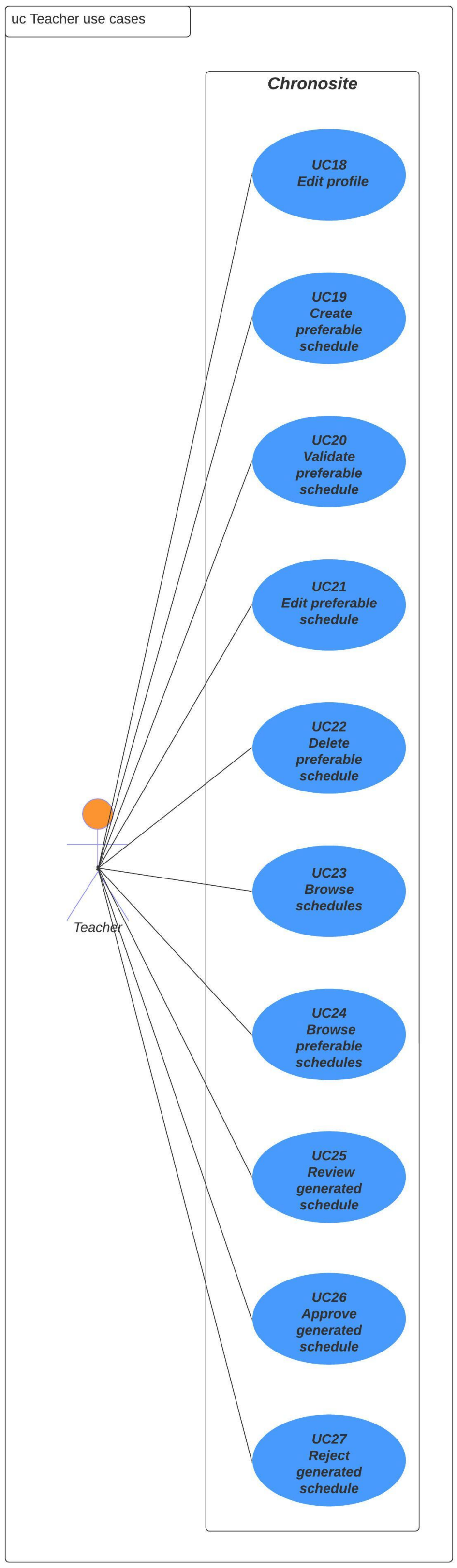


Figure 2.2 Teacher use cases

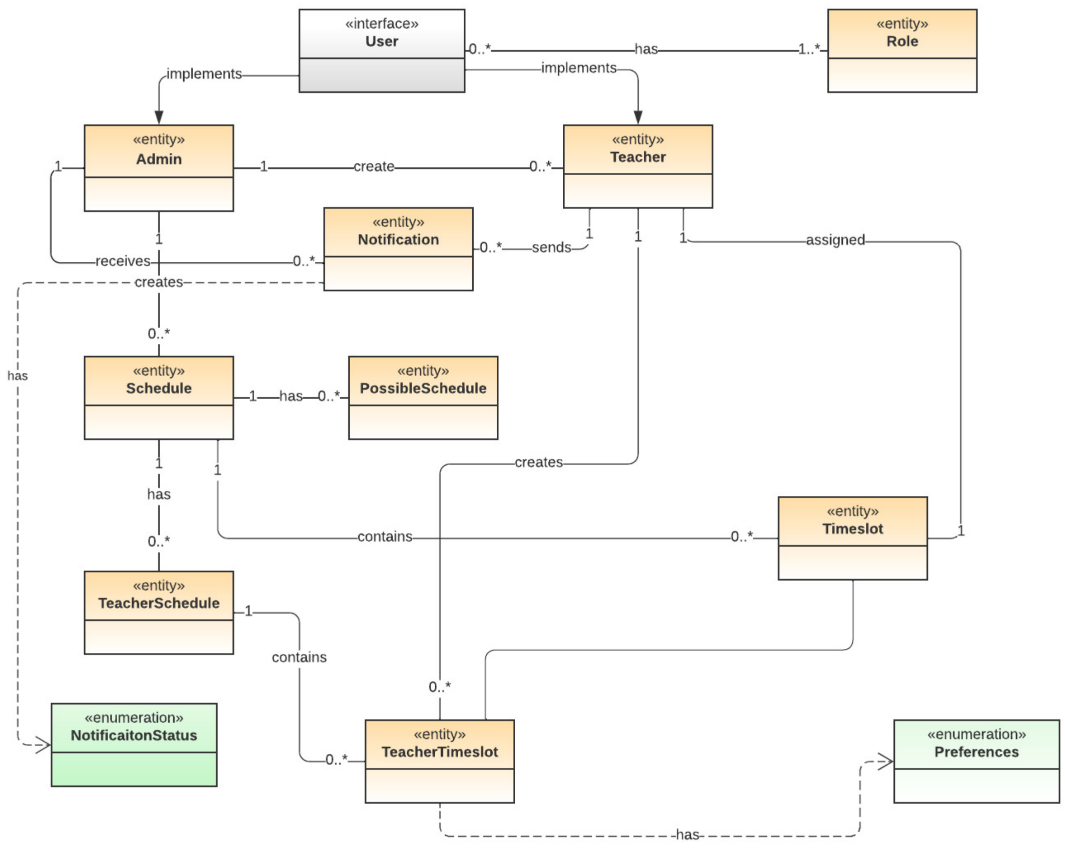

uc Teacher use cases
Chronosite
UC18 Edit profile
UC19 Create preferable schedule
UC20 Validate preferable schedule
UC21 Edit preferable schedule
UC22 Delete preferable schedule
UC23 Browse schedules
UC24 Browse preferable schedules
UC25 Review generated schedule
UC26 Approve generated schedule
UC27 Reject generated schedule
Teacher

2.6 Domain model

The analysis in this chapter, which includes functional and non-functional requirements, problem analysis, and use cases, provides an idea of how the domain model could be constructed. This model is then implemented and shown in 2.3.
Additionally, the entities and enumerations in the model have to be described.

- **Notification status** The notifications have the 3 statuses: **"Accept"**, **"Reject"**, and **"Complete"**. These statuses are the main, short messages of the notifications. Specifically, **"Accept"** indicates that a teacher has accepted the proposed schedule, while **"Reject"** indicates that the schedule has been rejected. **"Complete"** is the schedule's status once everyone has approved it and teachers are assigned to the time slots.
- **Preferences** The teacher time slots have 2 statutes: **"Accept"**, **"Reject"**. Teachers can pick one option when they are creating a preferable schedule. Prospectively, **"Accept"** is when a teacher wants to be assigned for the time slot, if not – **"Reject"**.
- **User** The **user** interface provides the ability to implement the **administrators** and **teachers** with the predefined **roles**. The goal of distinguishing the administrators and teachers instead of leaving the only **user** is to have different relationships with other entities.
- **Schedules** The **Schedule**, **TeacherSchedule**, and **PossibleSchedule** are generally the timetables with the concrete roles. **Schedule** is a original timetable created by the **administrators** and has the **timeslot** entities. The **TeacherSchedule** and **PossibleSchedule** are linked to the original timetable. So **TeacherSchedule** is the preferable timetable of the teacher and **PossibleSchedule** is the generated schedule using the teachers' preferable schedules.
- **Timeslots** The **Timeslot** is an entity that describes the start and end time and the specific day of the week of the lecture, seminar, or any other event. A **TeacherTimeslot** is the entity that describes the choice of the teacher to teach or not the original time slot.

Figure 2.3 Class diagram



Chapter 3

Design

This chapter will present the design of the "Chronosite" application, which includes the system architecture and its functional components. The server-side design and UI layer, along with the wireframes, will be discussed in detail.

3.1 Wireframes

Whether designing websites or web applications today creating purposeful graphics is not enough if there is not sufficient understanding of how best to lay out web pages themselves for optimal effectiveness. A good page structure must embody features like easy-to-use navigation systems that integrate seamlessly with excellent user interfaces (UIs) which embody great attention to detail information designs. A visually presented diagrammatic schematic or **wireframe** is an essential tool in achieving this since it provides a complete overview of the web page's constituent parts and layout allowing the design team to plan and execute their designs with greater precision.[3]
The author decided to present the wireframes that will introduce the UI of the "Chronosite" application. They were created based on the analysis in the previous chapter, especially considering the functional requirements and use cases of the system.

- The Figure 3.1 shows the administrator's screen that is related to the manipulations with teachers. It describes the use cases UC3, UC4, and UC5. The input fields as "Username" and "Password" are provided by the design of the wireframe because these data are required to perform operations on the teachers.

- The Figure 3.2 demonstrates the screens of the administrator's timetable list with available operations on them. The wireframes describe the use cases UC6, UC8, and UC15. The design of this wireframe provides the ability to choose different operations on the timetables in one place. Also, the input field "Name" is gain the required data to create a new timetable.

- The Figure 3.3 shows the screen with timetable. This wireframe provides the ability to review all the time slots of the timetable and differentiate them by day, start and end time, location, and type of lesson (lecture, seminar). Overall, the timetable would be displayed in the same way for all use cases that are related to timetable management or preferable schedule creation. The only difference would be the active buttons for the teachers' preferences and time slot creation. So, it covers the use cases UC7, UC9, UC21, UC23, and UC27.

- The Figure 3.4 demonstrates a screen with a notification list and operations on it. The wireframe is related to use cases UC12, UC13, UC14, and UC17, and includes functionality such as buttons for “Message” and “Delete” to perform operations on the notifications, as well as a brief description of the notifications.

- The Figure 3.5 demonstrates the use cases related to the operations on the time slots. It especially covers use cases UC10, UC11, and UC21 and provides input fields such as “Start time”, “End time”, “Location” and “Type” to collect data that is needed to create or update the time slot. Also, the wireframe includes functionality such as buttons for ”Message” and ”Delete” to receive a teacher preference regarding a specific time slot.

- The Figure 3.6 shows the process of the profile of the teacher editing. The wireframes are related to UC18. The input field “New password” and the button “Submit” are designed to receive a new password from the teacher and perform the profile edition process.

- The browsing of the preferable schedules of the teacher and operations on them are shown in Figure 3.7. The wireframes describe the use cases UC19, UC20, UC22, UC23, and UC24 and provide the ability to perform operations on the preferable schedules via buttons labeled “Validate/Edit” and “Delete”. In addition, teachers can edit the preferable schedule by clicking on the displayed schedule and using the “Week Range” button to change the number of time slots per week. This design allows teachers to easily track and perform changes as everything is in one place.

- The Figure 3.8 describes the process of the timetable reviewing. The wireframes cover the use cases UC26, UC27, UC28, and UC29, providing intuitively located buttons “Approve” and “Reject” that make it easy for teachers to decide whether to accept or reject the proposed schedule, which will be displayed on the same page. In addition, an input field labeled “Message” is provided to teachers to submit more precise feedback.

- Finally, Figure 3.9 shows the screen of the administrator for the timetable generation and it is related to the use case UC25. The wireframe contains a list of teachers along with their statuses (submitted or not submitted), which helps the administrator understand when it will be possible to generate the schedules. Additionally, the “Approve” button will be available after the schedule generation, allowing the administrator to easily review all of the generated schedules.

Figure 3.1 Operations on teachers by admin

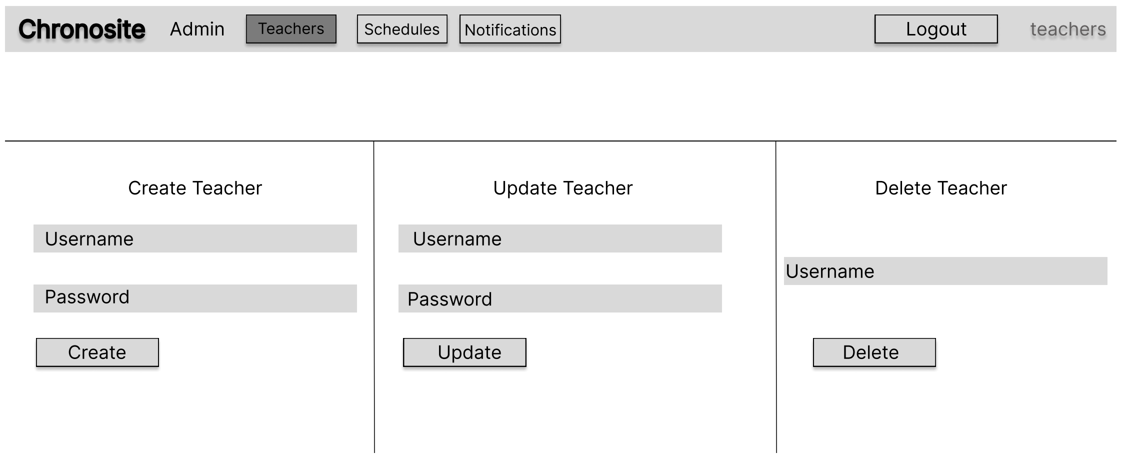
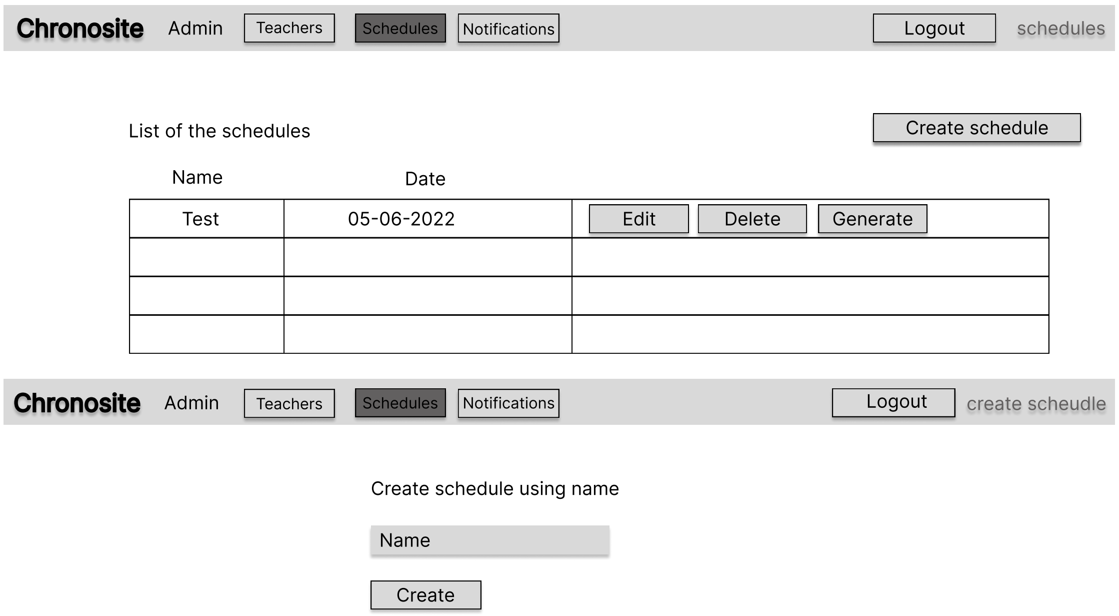


Figure 3.2 Operations on schedules by admin



List of the schedules

Name	Date	
Test	05-06-2022	Edit Delete Generate

Create schedule using name

Figure 3.3 Displayed schedule

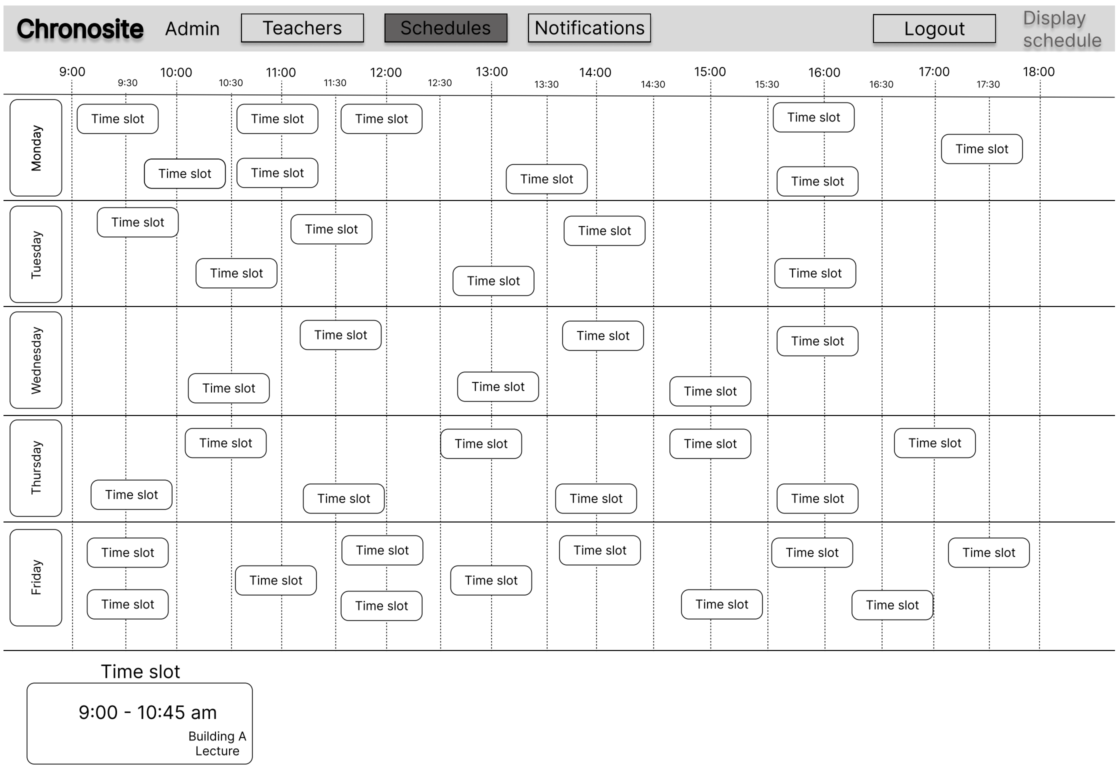


Figure 3.4 Notifications

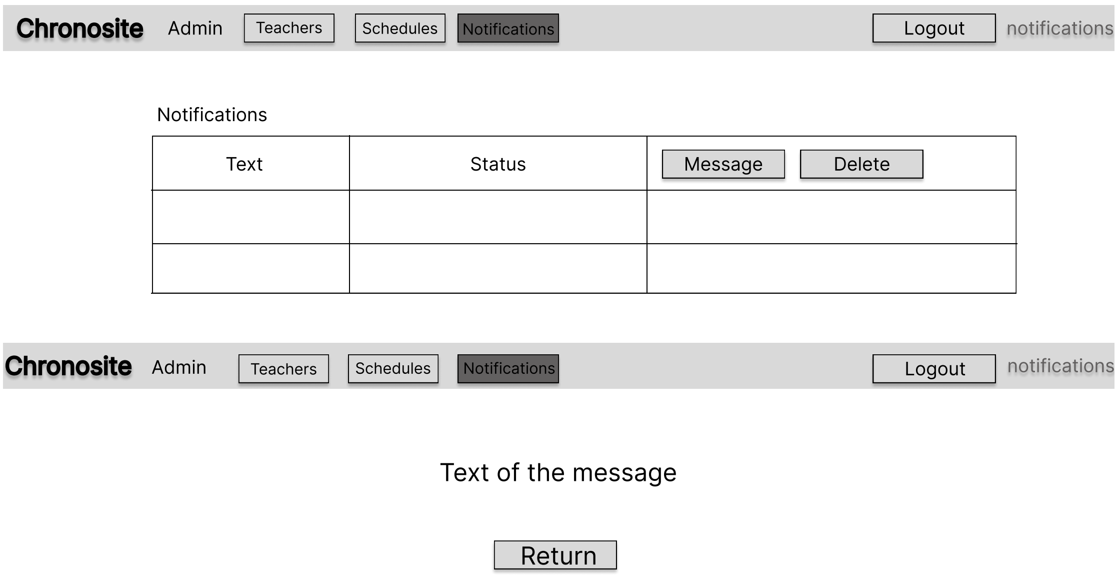
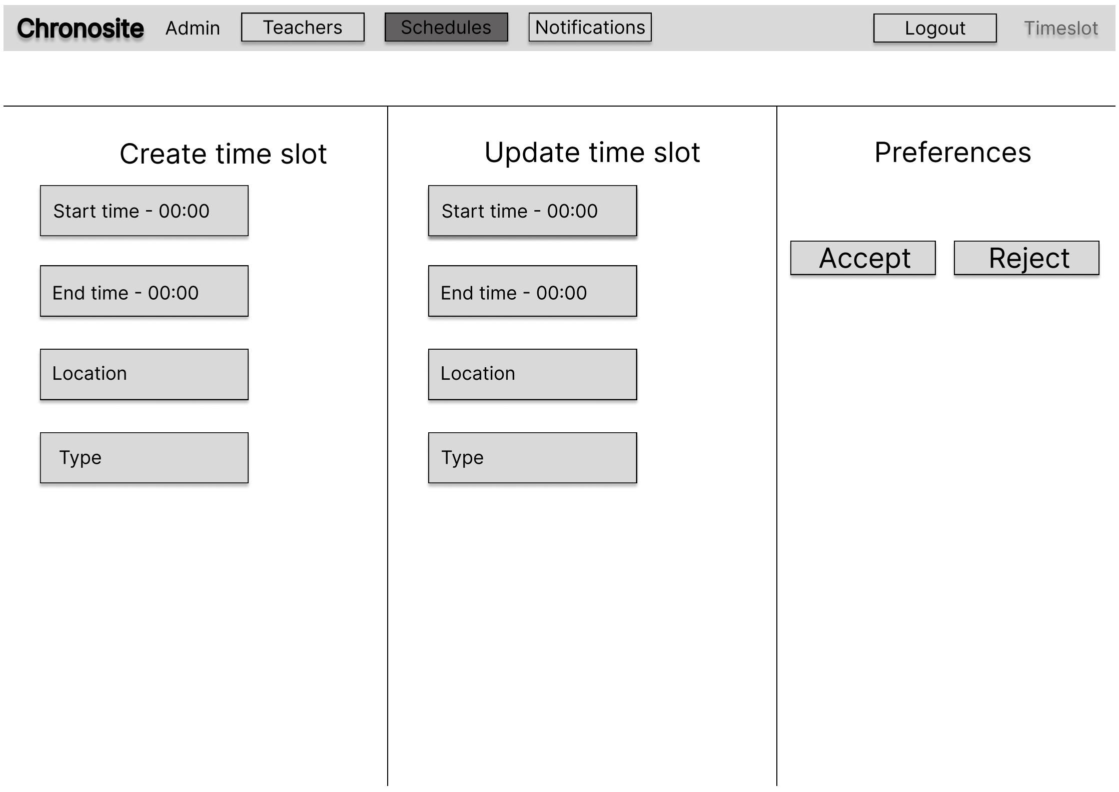


Figure 3.5 Operation on the timetable slots



3.2 System architecture

In this section, the system architecture will be discussed with its types, and functionality of the different components, but firstly the author decided to state several key points and the definition

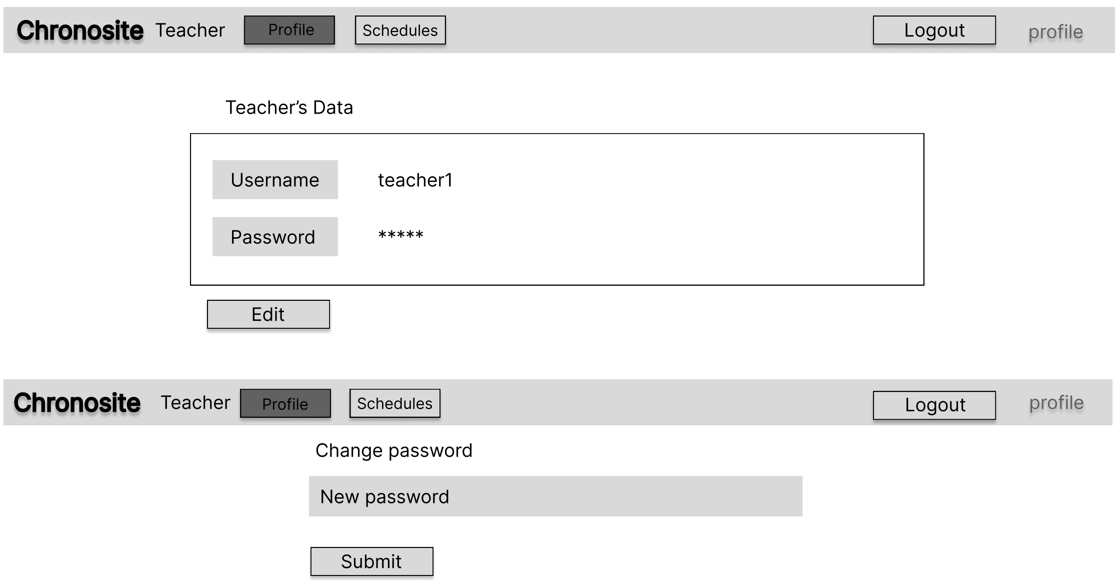


Figure 3.6 Profile edit process

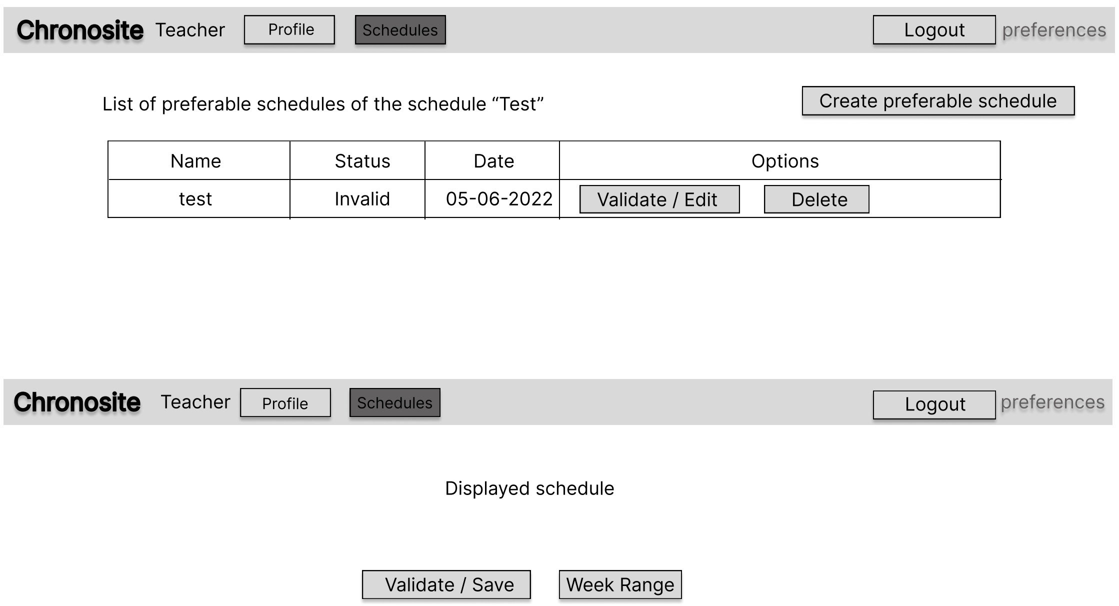


Figure 3.7 Preferences edit process

Chronosite
Teacher
Profile
Schedules
Logout
preferences
List of preferable schedules of the schedule “Test”
Create preferable schedule

Name	Status	Date	Options
test	Invalid	05-06-2022	Validate / Edit Delete

of the *“system architecture”*.

The **system architecture** is the representation of a system that maps out the functionalities of hardware and software components, as well as takes into account human interaction with these components. The system architecture covers the entire system, including hardware, software, and humans.[4]

Figure 3.8 Review timetable process

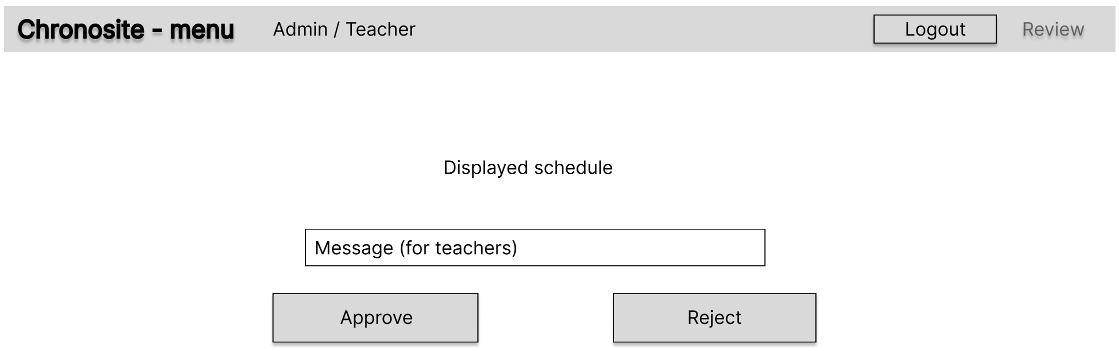
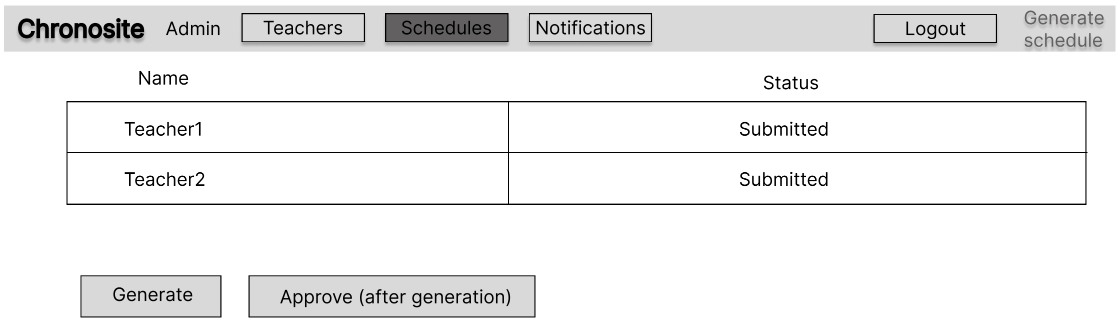


Figure 3.9 Timetable generation process



Name	Status
Teacher1	Submitted
Teacher2	Submitted

The thesis describes the concept of the web application. The system architecture of a web application can vary depending on the specific requirements and technologies used, but in general, it can be described as a client-server architecture. However, the web application can have different types of system architectures, such as microservices, service-oriented, and event-driven architectures, and the most suitable for the "Chronosite" prototype – monolithic architecture.

Also, the system should store user profiles and all types of schedules described in the domain model section. Additionally, it has to resolve the problem of timetable generation, so the algorithm and problem should be described. The following section will cover these precise topics with various architectures.

3.2.1 Architecture types

This subsection provides general information about system architecture types and describes their advantages and disadvantages.

In a **monolithic architecture**, which is described in [5], the application is developed and deployed as a single unit, resulting in a single executable. While this approach is straightforward to implement and manage, it may pose challenges in scaling and maintaining the system as it becomes larger and more complex.

Advantages:

1. Simple to develop and deploy, as the application is built and deployed as a single unit.
2. Easier to test, as the entire application can be tested in one go.
3. Fewer network calls as all components of the application reside on the same server, which results in faster performance.

Disadvantages:

1. Monolithic architecture can be difficult to scale, as the application has to be scaled up or down, besides the specific component that requires more resources.
2. Lack of modularity makes it challenging to make changes to specific components of the application, requiring redeployment of the entire application.
3. Risk of the system failing if any part of the application fails, leading to a single point of failure.

A **client-server architecture** is a type of distributed architecture that comprises two distinct parts: the client, which handles user interaction, and the server, which processes requests and stores data. This type of architecture is usually employed in web applications and offers benefits such as scalability, flexibility, and ease of maintenance. It explained in [6]

Advantages:

1. Allows the distribution of the workload between the server and clients, which makes it easier to expand the system as needed.
2. Allows for centralized management, making it possible to monitor and control access to resources.
3. Can be more cost-effective than other architectures since clients can be less expensive and less powerful, while the server can be more powerful and expensive.

Disadvantages:

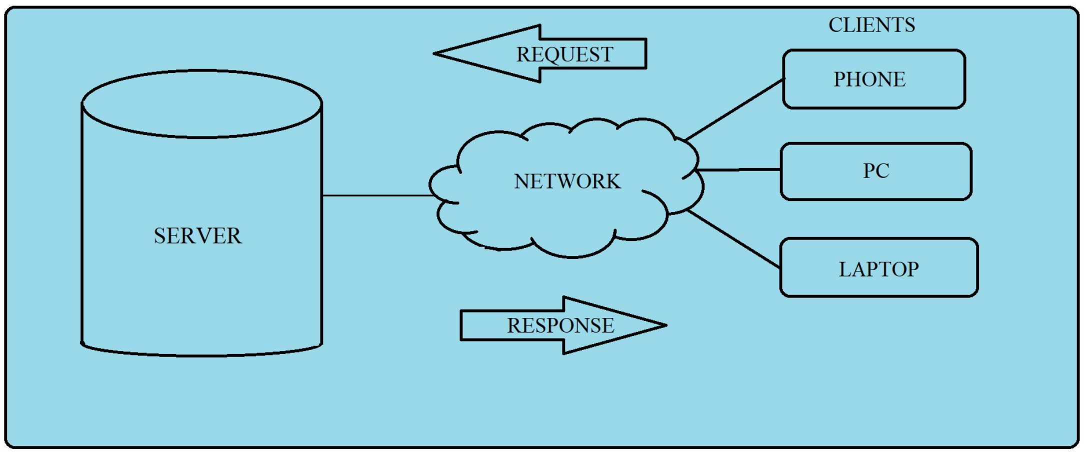
1. The risk of a single point of failure. If the server fails, it can bring the entire system down.
2. The architecture is also heavily reliant on network communication, which can result in delays or interruptions if network problems arise.
3. The server's workload can become a bottleneck, impacting system performance and responsiveness, especially when there are many clients.

A **microservices architecture**, which is outlined in the [7] involves building an application using small, autonomous services that interact with each other through APIs. This approach offers benefits such as flexibility, scalability, and ease of maintenance. However, it can also add complexity in terms of managing and deploying the services, which can be challenging.

Advantages:

1. Microservices architecture enables scalability, flexibility, and fault isolation.
2. Each component can be scaled independently, allowing the system to handle higher loads as it grows.

Figure 3.10 Client-Server architecture

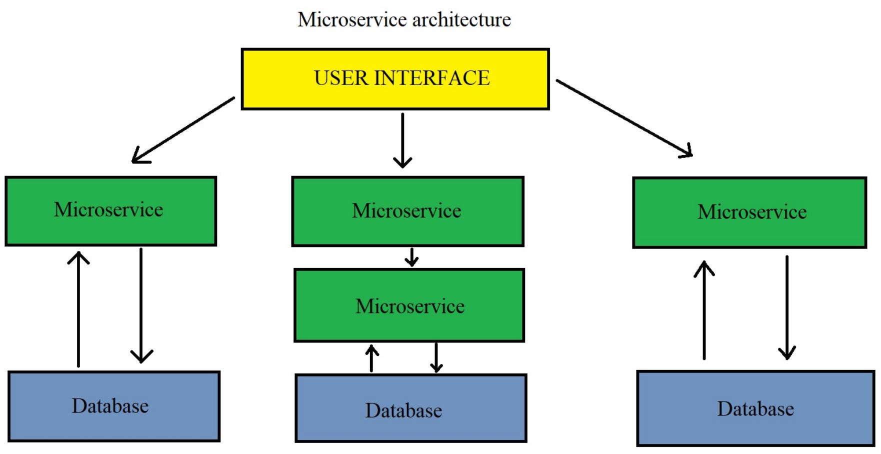


3. Faster development and deployment of new features, making it easier to respond to customer needs and market changes.

Disadvantages:

1. Microservices architecture can be more complex to manage, especially when dealing with a large number of services.
2. Requires additional management and monitoring to ensure consistency, reliability, and security across all services.
3. The overhead involved in managing and coordinating multiple services can also lead to increased costs and complexity, particularly for smaller projects or teams.

Figure 3.11 Microservices architecture



Considering the advantages and disadvantages of the several architecture types, which should be used for the prototyping? Using a monolithic architecture for prototyping a "Chonosite" can be a good option due to its simplicity and ease of setup, which allows focusing on building and testing the application's core features without worrying about the complexities of a microservices architecture.

3.2.2 Roles

The system requirements, F1 and F2, mandate that the system must offer roles for registered users. One approach to handling authorization and access control in the system is to use the **Role Object Pattern** (ROP), as described in [8]. ROP involves creating a **Role** object that contains information about a specific role in the system, such as its name and permissions. Users are then assigned to these roles, allowing them to access only the parts of the system that are relevant to their role. This simplifies permission management, as changes can be made at the role level, instead of for each user. Furthermore, this pattern promotes consistency and helps ensure that users have the appropriate level of access for their role, while also reducing the risk of unauthorized access.

3.2.3 Data management

The system requirements F1, F2, and N5 specify that user profiles must be stored securely to prevent potential data leaks. Additionally, requirements related to timetables call for optimized data structures and database schema. Therefore, an SQL database with a security framework like Spring Security would be preferable. However, a NoSQL database could be used for data storage. A detailed description of the technologies will be provided in the next chapter, but first, a comparison between SQL and NoSQL databases will be presented.
One of the primary differences between SQL and NoSQL databases is their approach to data modeling. SQL databases rely on a rigid schema that defines the database structure, which can make it challenging to modify the schema when new requirements arise. In contrast, NoSQL databases employ a flexible schema that allows developers to add new data elements without modifying the existing schema. However, NoSQL databases are relatively new and lack a well-established security framework compared to SQL databases. Typically, NoSQL databases use access control mechanisms like IP-based access restrictions, encryption, and firewalls to secure the data.
In summary, based on the system requirements, SQL databases are a better fit than NoSQL databases. Therefore, the choice is clear.

3.2.4 Problem of the timetable generation

Requirement F7 states that the prototype should generate all possible schedules, considering validated preferred schedules from the teachers. To create a design of the solution that automates the process described in the analysis section 2.1, the problem should first be described in detail. Additionally, each step of the timetable generation process should be described along with a solution for automation.

3.2.4.1 Description

The described problem itself is similar to the **employee scheduling problem** in computer science that is explained in [9] with possible solutions. The employee scheduling problem involves assigning employees to work shifts based on various constraints, such as availability, skills, preferences, labor laws, and union rules and is also considered an NP-hard problem and is characterized by several types of constraints, including:

- **Employee availability** Each employee has certain hours or days of availability during which they can work, and the schedule must take these constraints into account.

- **Skill requirements** Certain tasks may require specific skills or qualifications that only some employees possess. The schedule must assign the appropriate employees to these tasks.
- **Time constraints** Some tasks may have strict deadlines or specific start times, and the schedule must ensure that these constraints are met.

As stated in the [10], polynomial-time reductions provide a formal means for showing that one problem is at least as hard as another, to within a polynomial-time factor. That is, if $L_1 \leq_p L_2$, then L_1 is not more than a polynomial factor harder than L_2, which is why the "less than or equal to" notation for reduction is mnemonic. We can now define the set of NP-complete languages, which are the hardest problems in NP.

A language $L \subseteq \{0,1\}^*$ is NP-complete if

1. $L \in$ NP, and
2. $L' \leq_p L$ for every $L' \in$ NP

If a language L satisfies property 2, but not necessarily property 1, we say that L is **NP-hard**. We also define NPC to be the class of NP-complete languages. As the following theorem shows, NP-completeness is at the crux of deciding whether P is, in fact, equal to NP.

Based on the definition of **NP-hard** and an analysis of the problem, the author decided to demonstrate a possible polynomial-time reduction from the employee scheduling problem to show that the teacher scheduling problem is an **NP-hard** problem:

1. Create an employee for each teacher and create a task for each course.
2. Assign a set of available time slots for each employee, based on the times that they are available to teach.
3. For each time slot, create a set of required qualifications and assign a set of required employees to teach the time slot, based on their qualifications.
4. Run the employee scheduling algorithm on the resulting problem instance.

Doing so, the teacher scheduling problem can be solved by transforming it into an instance of the employee scheduling problem, which is known to be **NP-hard** and it can be stated that the teacher scheduling problem is also **NP-hard**.

3.2.4.2 Create draft schedule

To create a draft schedule, an optimized algorithm is needed to solve the teacher scheduling problem. The crucial step is to choose the optimal algorithm type that best addresses the problem. The algorithm flowchart presented in Figure 3.12 and described within five steps as follows:

1. The algorithm should take each time slot one after another.
2. It should assign one teacher to each time slot using only one suitable schedule submitted by the following teacher. If the teacher can be assigned again, the algorithm should use the same preferable schedule until the end of the computational process.
3. If it is impossible to perform the previous step, the algorithm cannot create a possible schedule.
4. If it is possible to assign a teacher, the algorithm should move on to the next time slot.
5. If all time slots have been assigned to teachers, the algorithm should create a possible schedule.

Considering the steps outlined, several algorithm types can be used for the teacher scheduling problem, such as backtracking, genetic, and simulated annealing algorithms. Genetic and simulated annealing algorithms are typically used in machine learning or robotics areas and are complex algorithms for large systems. In the case of the prototype, the **backtracking algorithm** would be the most suitable option.

Backtracking is an algorithmic approach described in [10] that involves starting with an incomplete solution and gradually adding new elements to it until a complete solution is reached. At each step, the algorithm generates all possible extensions of the current partial solution and checks if each one is valid. If a valid extension is found, the algorithm proceeds to the next step with the extended solution. However, if no valid extensions are found, the algorithm backtracks to the previous step and tries a different extension. It's important to note that backtracking is a brute-force approach, so the resulting schedules should have a reasonable number of assigned teachers and optimization to decrease time complexity.

The list of rules to prevent an illogical number of teachers from being assigned to time slots are as follows:

1. If the number of teachers is equal to the number of time slots, then each teacher must be assigned only once.

2. If the number of teachers is greater than the number of time slots, then the number of assigned teachers must be equal to the number of time slots in the timetable.

3. If the number of teachers is less than the number of time slots, then each teacher must be assigned at least once, and some of them can be assigned again.

The list of optimizations that will cut off the failure schedules and improve the assigning process:

1. The assignment should start with unassigned teachers so that the possible schedules with unique teachers can be generated first.

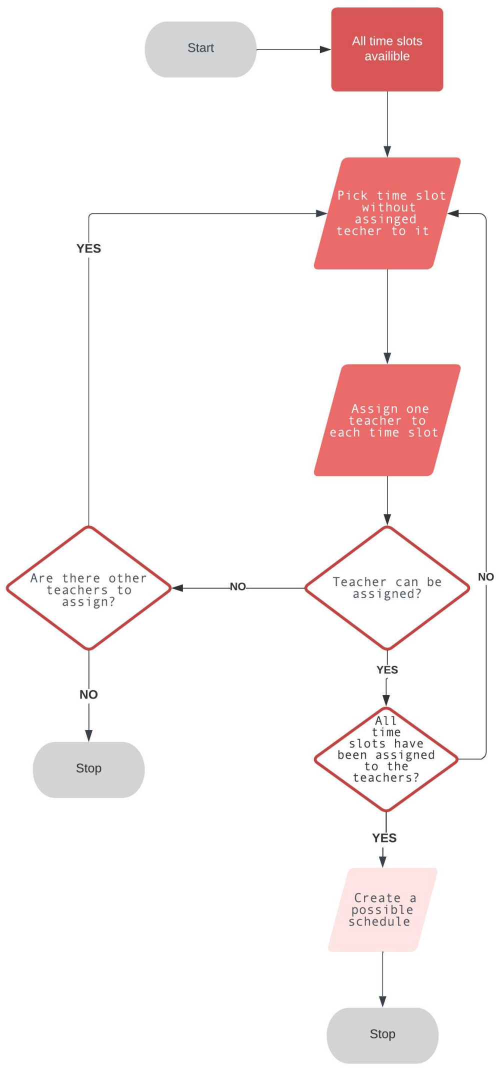
2. The difference between the assigned and unassigned teachers should be tracked. It would be used to follow the rules that prevent an illogical number of teachers that being assigned to time slots.

3. The ranges of the taken time slots per week or day of the chosen preferable schedule of the teacher and the number of assigned teachers should be reviewed by the system during the assignment process. This optimization will decrease the amount of generated schedules and eliminate potentially unsuccessful generated schedules.

Additionally, to improve the performance of the algorithm a great option is to use parallel computation and asynchronous programming techniques. The parallel computation approach involves breaking down a problem into smaller tasks, such as the process of the time slot, that can be processed in parallel. Multiple processors can work together to solve the problem faster. Moreover, asynchronous programming allows multiple tasks, such as reviewing the ranges per day or week of the chosen preferable schedule of the teacher or the number of assigned teachers, to be executed simultaneously without waiting for each other to finish. This allows the program to continue running while it waits for some long-running task to complete.

To summarize, the backtracking algorithm should be used to solve the teacher scheduling problem. It should have the listed optimizations, which will create the most suitable algorithm for the problem and increase computational power.

Figure 3.12 Algorithm flowchart diagram


Start
All time slots availible
Pick time slot without assinged techer to it
Assign one teacher to each time slot
Teacher can be assigned?
NO
Are there other teachers to assign?
YES
NO
Stop
YES
All time slots have been assigned to the teachers?
NO
YES
Create a possible schedule
Stop

3.2.4.3 Review draft schedule

The reviewing process is split into two main parts: review by the administrator and review by all involved teachers.

1. The administrator should choose one schedule from the list of generated schedules that, in his opinion, would be acceptable for the involved teachers.

2. The teachers can then accept or reject the schedule selected by the administrator. They can also leave a message for the administrator along with their choice.

3.2.4.4 Finalize the schedule

The complete timetable would be a timetable with assigned teachers to all time slots. If all the involved teachers accept the generated and approved schedule by the administrator, then teachers would be assigned to the time slots in the original timetable created by the administrator. However, if at least one teacher rejects the proposed schedule by the administrator, then the process of schedule generation starts from scratch.

3.3 Server-side

To meet the functional requirements, use cases, and system architecture of the prototype, the languages, frameworks, and libraries used for developing the server layer should satisfy the following requirements:

1. Simplicity, scalability, robustness.
2. High computing, performance speed.
3. Security.

Many languages meet enumerated requirements such as **Java**, **Node.js**, **Ruby**, and **Python**. Also, the programming should be platform-independent, robust language with security libraries to satisfy all required criteria.

As stated before, the monolithic architecture allows for building a single unit. Packaging in such an architecture is simple because all components are packaged together, so complex packaging structures are not required.

3.3.1 Architecture

The architecture of the server should be structured into three layers, each with a distinct set of responsibilities:

- **Presentation layer** is responsible for exposing the functionality of the server to clients via a RESTful API. It consists of REST controllers that manage HTTP communication and handle requests and responses.
- **Domain layer** contains the core business logic of the server. It defines the data structures, business rules, and algorithms that govern the behavior of the application.
- **Data layer** is responsible for data persistence. It is connected to an SQL database and handles database queries and updates.

To ensure a high degree of modularity and maintainability, the Presentation layer depends only on the Domain layer, while the Domain layer depends only on the Data layer. This design ensures that there are no inverse dependencies between layers, thereby avoiding tight coupling and making it easier to modify individual layers without affecting others.

3.4 UI layer

This UI layer should be compatible with the monolithic architecture, front-end frameworks, and modern programming languages. Also, it should follow the Model-View-Controller (MVC) pattern[11] that has the following key points:

- **Separation of Concerns** The MVC pattern aims to separate an application's data model, user interface, and control logic into 3 distinct components: Model, View, and Controller.
- **Model** The Model is responsible for managing the data and the business logic of the application. It encapsulates the data and provides methods to access and manipulate it.

- **View** The View is responsible for presenting data to the user and receiving user input. It represents the user interface of the application.
- **Controller** The Controller acts as an intermediary between the Model and the View. It receives input from the View, manipulates the Model, and updates the View accordingly.
- **Loose Coupling** The MVC pattern promotes loose coupling between the Model, View, and Controller. This means that each component is independent of the others, making it easier to maintain, test, and modify the code.

Chapter 4

Implementation

This chapter aims to demonstrate the implementation of the "Chronosite" prototype using the design discussed in the previous chapter.

4.1 Programming language

The chosen programming language should be the most suitable for the "Chronosite" considering the statements from the design chapter. Because the author is only able to implement a prototype using Java or JavaScript programming languages, these 2 languages can be used. A runtime environment such as Node.js can be utilized for JavaScript. However, a comparison between Java and Javascript would be pointless, because Javascript is commonly used to create dynamic and interactive web pages[12]. So, the description and key features will be provided for Java and Node.js with a comparison of them.

4.1.1 Java

Java is a popular programming language that can be run on any operating system that has a Java Virtual Machine (JVM) installed. It is an object-oriented language, meaning it is based on the idea of objects interacting with each other to carry out tasks. Java is strongly typed, which means all variables must be explicitly declared with a specific data type. Additionally, Java supports automatic garbage collection to help manage memory usage.[13]

Key features:

- Portability – Java code can be compiled into bytecode, which can be run on any system that has a JVM installed, making it highly portable.
- Security – Java includes built-in security features, such as sandboxing, that help protect against malicious code.
- Multithreading – Java supports multithreading, which allows multiple threads to run concurrently within a single program.
- Exception handling – Java includes a robust exception-handling mechanism that helps developers write more reliable and robust code.
- Rich APIs – Java includes a large set of standard libraries and APIs, including networking, and database access which can help developers build complex applications more quickly and easily.

4.1.1.1 Collections Framework

Based on the [14], the Collection Framework is a set of tools for managing collections of objects in Java. It includes several key interfaces, like List, Set, and Map, and their associated implementations. These tools help simplify code and improve performance by providing efficient ways to sort, search, filter, and manipulate collections of data. Some examples of the common data structures supported by the framework are ArrayList, LinkedList, HashSet, and TreeMap.

4.1.1.2 Stream API

The Stream API is a library in Java. It was introduced in Java 8 as a part of the Java Collections Framework and is a powerful tool for processing collections of data. The Stream API provides a functional programming approach to working with data collections, allowing developers to perform operations such as filtering, mapping, and reducing a collection of objects in a concise and readable way. It is a part of the java.util.stream package and provides several classes and interfaces that can be used to work with data streams in Java.[15]

4.1.1.3 Concurrent

It is a library in Java that provides a set of utilities for concurrent programming in Java. It includes classes and interfaces for managing threads, locks, atomic variables, synchronizers, and executor services. The library was introduced in Java 5 to make it easier to write multithreaded applications in Java. It provides higher-level abstractions for managing concurrency, making it easier to write correct and efficient concurrent code.[16]

4.1.2 Node.js

Node.js is a runtime environment that lets developers use JavaScript on the server side, which was previously limited to client-side use in web browsers. It is open-source and cross-platform, meaning it can be used on different operating systems. Node.js is based on the V8 JavaScript engine from Google, which enables fast and efficient execution of JavaScript code. It comes with built-in modules for tasks like file system access, networking, and handling HTTP requests. Node.js also has a vast collection of third-party packages available through the Node Package Manager (NPM), making it easy for developers to incorporate existing tools and libraries into their projects.[17]

Key features:

- Asynchronous and Event-driven Programming Model – Node.js is designed to handle multiple concurrent connections efficiently without slowing down the server, thanks to its event-driven, non-blocking I/O model

- Built-in Modules and Libraries – Node.js includes built-in modules and libraries for tasks such as file system access, networking, and HTTP request handling, making application development faster and easier

- Cross-platform Compatibility – Node.js is a cross-platform runtime environment that can be run on Windows, macOS, and Linux.

- Scalability and Performance – Node.js is highly scalable and performant, allowing for the creation of applications that can handle many simultaneous connections due to its asynchronous I/O model.

4.1.3 Summary

Before choosing the programming language, the author decided to provide a comparison between Java and Node.js:

- Node.js applications are easy to deploy, especially when compared to Java, as they require fewer resources and can be run on smaller servers.
- As described in the [18], Node.js is not as suitable for CPU-intensive tasks as Java. It may not be the best choice for applications that require a lot of computational power.
- Node.js has limited support for multithreading, which can be a drawback for applications requiring parallel processing or high scalability.

To summarize, Node.js is a good choice for prototyping, but Java has significantly better computational power compared to Node.js because as stated before Node.js is not suitable for CPU-intensive tasks and has limited support for multithreading. The problem of scheduling teachers is an NP-Hard problem and has a high time complexity and requires high computational power. So, based on the comparison, Java has been chosen as the programming language for the application.

4.1.4 Libraries

Several libraries of Java programming language were utilized to streamline the development process. Below are the most noteworthy of these.

4.1.4.1 Guava

Guava is a Java library created by Google that offers a range of utilities and helper classes to improve Java programming by making it simpler and more effective. The library enhances the functionality provided by the Java standard library with features such as collections, caching, concurrency, functional programming, and I/O.[19]

4.1.4.2 Lombok

Based on [20], Lombok is a library for Java that aims to simplify the process of writing Java classes by reducing the amount of boilerplate code that developers need to write. It provides a set of annotations that are processed at compile time to generate code that would otherwise need to be written manually. Although some people refer to it as a “compile-time framework,” it does not have a runtime component or a formal architecture for building applications. Instead, it is a tool that can be used to improve the efficiency and readability of Java code.

4.2 Server implementation

4.2.1 Framework

The backend framework needs to be in sync with both the Java language and the server-side design and should offer a broad range of libraries, such as security and dependency injection. Based on the author’s research, the well-known Spring framework, Jakarta EE, and Google Guice can be potential candidates to use.

4.2.2 Spring

The Spring Framework is an open-source framework that provides extensive infrastructure support for building enterprise-level applications. It offers a wide range of features such as dependency injection, inversion of control, aspect-oriented programming, and data access.[21]

Advantages:

- Dependency injection simplifies object dependencies, promotes loose coupling
- Integrates well with 3rd-party libraries and frameworks
- Large, active community with extensive documentation and tools

Disadvantages:

- Runtime performance overhead compared to lighter frameworks.
- Potential complexity in large projects, requiring good planning and adherence to best practices.

4.2.3 Jakarta EE

Jakarta EE is a platform that offers a collection of technologies and specifications for building enterprise-level Java applications. This platform provides a standard framework and a shared set of APIs, making it easier to construct secure, scalable, and reliable enterprise applications that can be deployed on various servers and platforms. Jakarta EE includes a set of well-defined technologies such as servlets, JavaServer Pages (JSP), Java Persistence API (JPA), Java Messaging Service (JMS), Enterprise Java Beans (EJB), and more. These standardized components help to build high-performance, distributed, and transactional applications with minimal effort.[22]

Advantages:

- Standardized platform and APIs for enterprise Java development
- Large community support and wide range of compatible servers
- Mature and well-established technology stack

Disadvantages:

- Complex and heavyweight, may not be suitable for small applications
- Limited innovation and slow pace of updates compared to other frameworks.

4.2.4 Google Guice

Google Guice is a framework that helps Java developers write more flexible and maintainable code by separating different parts of an application and reducing dependencies between them. By defining bindings between interfaces and their implementations, Guice automatically injects dependencies at runtime, making code more modular, testable, and easier to maintain.[23]

Advantages:

- Dependency injection simplifies object dependencies, promotes loose coupling
- Integrates well with 3rd-party libraries and frameworks

- Large, active community with extensive documentation and tools

Disadvantages:

- Smaller Community and Resources
- Configuration Complexity for Large Projects

4.2.5 Final decision

The named frameworks have their benefits and cons, however, the Google Guice does not provide any security library. Also, the Spring framework and Jakarta EE are more mature products than Google Guice, so the author decided to use them for the implementation of the "Chronosite".

4.2.6 Libraries

Certain external libraries were utilized in the development process, and they are outlined below.

4.2.6.1 Spring Boot

Spring Boot is a framework for building standalone applications in the Java ecosystem. It aims to simplify the development process by providing pre-configured settings that allow developers to quickly set up and deploy applications with minimal configuration. Spring Boot includes a range of features such as embedded servers, health checks, metrics, and security, among others.

4.2.6.2 Spring Security

Spring Security is a security framework for Java applications that provides authentication, authorization, and access control features. It is built on top of the Spring Framework and integrates seamlessly with it. Spring Security offers a flexible and customizable architecture that supports a wide range of authentication and authorization mechanisms, including form-based, HTTP basic, and OAuth2. It also provides advanced features such as CSRF protection, session management, and LDAP integration.

4.2.7 Java-based Templating Engine

A Java-based server-side templating engine is a software tool that generates dynamic web pages by combining HTML templates with data from a back-end system. They provide powerful features such as conditional statements, and variable substitution, and integrate easily with Java-based web frameworks like Spring. The popular and intuitive in practice Java-based server-side templating engine is Thymeleaf.

"Thymeleaf is a Java XML/XHTML/HTML5 template engine that can work both in web (servlet-based) and non-web environments. It is better suited for serving XHTML/HTML5 at the view layer of MVC-based web applications, but it can process any XML file even in offline environments. It provides full Spring Framework integration".[24]

The main benefits of the usage of the Thymeleaf are:

- Fully supports HTML5 and integrates with popular Java-based web frameworks.
- Enables reusability through the creation of template fragments, which can save development time and effort.

- Provides advanced features such as conditional statements, loops, and variable substitution, which can help create complex and dynamic web pages.

Overall, Thymeleaf is compatible with monolithic architecture and Java language. Also, it provides various benefits which make the described engine a good option to choose.

4.3 Algorithm implementation

The following section describes the backtracking algorithm implementation with the optimizations established in the design chapter.

4.3.1 Backtracking algorithm

The method `generateSchedule` shown in the listing 1 is a core part of the computation of the possible schedule based on the backtracking algorithm. Mainly, it is divided into the 3 parts: preparation, computation, and result.

4.3.1.1 Preparation

In the preparation stage, the method initializes and prepares the necessary data structures and variables for generating possible schedules. The steps of the preparation are described as follows:

1. The method `generateSchedules` takes two input parameters: `scheduleId` of type `Long` and `adminUsername` of type `String`.
2. The method initializes three lists: `possibleScheduleDtos`, `timeslotDtos`, and `teachers`.
3. The method also retrieves a list of `PossibleSchedule` entities from a repository using `scheduleId`.
4. The `futureShuffle` boolean variable is set to true if the number of possible combinations of teachers and timeslots is less than or equal to the number of already generated schedules.
5. If `futureShuffle` is false and there is only one schedule and it's not approved, then the method returns the list of PossibleSchedule entities for that `scheduleId`.
6. Otherwise, the method deletes all the previously generated `PossibleSchedule` entities for the given `scheduleId`.

4.3.1.2 Computation

In the computation stage, the method generates all possible schedules by using a backtracking algorithm that is implemented with the similarity to the Depth-First Search algorithm[10] by the following steps:

1. The method initializes a `Stack` of shown in the listing 2 `BackTrackState` objects with one initial object representing an empty schedule.
2. The algorithm runs in a loop until the `Stack` is empty.
3. Each iteration of the loop collects in parallel a limited number of `BackTrackState` objects from the `Stack` to the `statesToProcess` list. The limit number is up to the available processes of the system.

4. The processState method takes a BackTrackState object from the statesToProcess list and tries to generate a possible schedule based on the current state and the remaining timeslotDtos and teachers.
5. The processState method adds the newly created BackTrackState objects to the Stack and removes the processed ones.
6. The algorithm terminates either when the Stack is empty or when the maximum number of schedules (maxSchedules) is reached.

4.3.1.3 Result

In the result stage, the method returns a list of PossibleSchedule objects by converting each PossibleScheduleDto object in the following steps:

1. After the backtracking algorithm finishes, the method generateSchedules checks if any valid PossibleScheduleDto objects were generated. If not, it throws a custom exception.
2. Otherwise, it converts each PossibleScheduleDto to a PossibleSchedule entity and adds it to a list of PossibleSchedule entities.
3. Finally, the method returns the list of PossibleSchedule entities.

```
@Override
@Transactional
public List<PossibleSchedule> generateSchedules(Long scheduleId, String
 ↪  adminUsername) {

List<PossibleScheduleDto> possibleScheduleDtos = new ArrayList<>();
List<TimeslotAssignmentDto> timeslotDtos =
 ↪  timeslotGenerationService.getAssignmentDtos(scheduleId);
List<TeacherAssignmentDto> teachers =
 ↪  teacherService.getAllByAdminUsername(adminUsername)
        .stream()
        .map(teacher ->
        ↪  ScheduleGenerationMapper.fromEntityToTeacherAssignmentDto(teacher,
        ↪  scheduleId))
        .collect(Collectors.toList());

List<PossibleSchedule> existedSchedules =
 ↪  possibleScheduleRepository.getAllByScheduleId(scheduleId);
boolean futureShuffle = ((int)Math.pow(teachers.size(), timeslotDtos.size())
 ↪  <= existedSchedules.size());

if(!futureShuffle && (existedSchedules.size() == 1 &&
 ↪  !existedSchedules.get(0).getApprovedByAdmin())  ){
    return  possibleScheduleRepository.getAllByScheduleId(scheduleId);
}else {
    possibleScheduleRepository.deleteAllByScheduleId(scheduleId);
}

Stack<BackTrackState> stack = new Stack<>();
stack.push(new BackTrackState(new PossibleScheduleDto(scheduleId,
 ↪  timeslotDtos.size()), timeslotDtos, teachers, futureShuffle));
```

```java
while (!stack.isEmpty()) {
    List<BackTrackState> statesToProcess = stack
            .stream()
            .parallel()
            .limit(Math.min(stack.size(),
            ↪  Runtime.getRuntime().availableProcessors()))
            .collect(Collectors.toList());

    stack.removeAll(statesToProcess);

    int maxSchedules = 50;
    List<PossibleScheduleDto> processedSchedules = statesToProcess
            .parallelStream()
            .map(state -> processState(state, stack))
            .filter(Objects::nonNull)
            .limit(maxSchedules - possibleScheduleDtos.size())
            .collect(Collectors.toList());

    possibleScheduleDtos.addAll(processedSchedules);
}

if (possibleScheduleDtos.isEmpty() ||
↪  possibleScheduleDtos.get(0).getTimeslots().isEmpty()) {
    throw new
    ↪  CustomException("Impossible to generate schedule with teachers' preferences or schedu
}

return possibleScheduleDtos
        .stream()
        .parallel()
        .map(this::toEntity)
        .collect(Collectors.toList());
}
```

■ **1** The `generateSchedules` method in the `ScheduleGenerationService` class

```java
public record BackTrackState(PossibleScheduleDto schedule,
                             List<TimeslotAssignmentDto> timeslots,
                             List<TeacherAssignmentDto> teachers,
                             boolean shuffle) {
}
```

■ **Code listing 2** The `BackTrackState` record

4.3.2 Additional methods

This subsection describes the additional methods that were used in the backtracking algorithm implementation.

4.3.2.1 `processState`

The `processState` method shown in the listing 3 is a part of the backtracking algorithm that performs the assignment of the teachers to available timeslots in a schedule. It takes a state object and a stack of previous states as inputs and returns a `PossibleScheduleDto` object representing the completed schedule if one is found.

The method first checks if the schedule is already complete. If not, it retrieves the available timeslots and teacher assignments from the state object, and checks if the current state is valid. If the state is valid, it attempts to assign a teacher to the first available timeslot.

If there are unassigned teachers left and there are not enough available timeslots to assign them all, the method will backtrack to a previous state and try a different assignment. If no solution is found, the method returns `null`.

4.3.2.2 `validateRanges`

The `validateRanges` method that is presented in the listing 4 receives a multimap of teacher timeslots and a map of used teacher preferences. It validates the assigned teacher ranges for each teacher. If a teacher's ranges are invalid, it throws a `RuntimeException`. If all teacher ranges are valid, it returns `true`.

4.3.2.3 `teachersLogic`

As shown in the listing 5, the `teachersLogic` method receives a list of available teachers, a timeslot, and a boolean indicating whether any teachers are unassigned. It shuffles the list of teachers if required, and then iterates through them to check whether they can be assigned to the timeslot. If a teacher is already assigned to the timeslot, it creates a new schedule with the same teacher and adds it to the stack. Otherwise, it calls the `iterateThroughTeacherPreferences` method for the teacher.

4.3.2.4 `iterateThroughTeacherPreferences`

The `iterateThroughTeacherPreferences` method that is shown in the listing 6 iterates through a teacher's schedule to check whether they are available to teach during a given timeslot. If a teacher is available, it creates a new schedule with the teacher assigned to the timeslot and adds it to the stack.

4.3.3 Optimizations implementation

All described optimizations in the design chapter are related to the `processState` method 3 and some of them are implemented in the additional methods.

- The optimization of starting the assignment with unassigned teachers has been implemented in a way that the `unassignedTeachers` list receives teachers who were not assigned to any time slot during the generation of the schedule. Depending on the value of the `logicState` variable, this list can be used in future computational processes.

- The optimization of tracking the difference between the assigned and unassigned teachers to prevent an illogical number of teachers being assigned to time slots was implemented by creating a boolean variable called `logicState`. This variable dictates which list of teachers (`unassignedTeachers` or `teachers`) will be passed to the method `teachersLogic`.

```
     public PossibleScheduleDto processState(BackTrackState state,
      ↪  Stack<BackTrackState> stack) {
        PossibleScheduleDto schedule = state.schedule();
        List<TimeslotAssignmentDto> allTimeslots = state.timeslots();
        List<TeacherAssignmentDto> teachers = state.teachers();

        Multimap<String, TimeslotAssignmentDto> teacherTimeslotMap =
        ↪  schedule.getTeachersTimeslots();
        Map<String, Long> usedPreferences = schedule.getTeacherPreferences();

        // Check for the completed schedule
        if (schedule.isComplete()) {
            return schedule;
        }

        // Receive available timeslots and already used teachers with
        ↪  preferences
        List<TimeslotAssignmentDto> availableTimeslots =
        ↪  getAvailableTimeSlots(allTimeslots, schedule.getTimeslots());

        if(!validateRanges(teacherTimeslotMap, usedPreferences)) {
            return null;
        }

        // Go through all available timeslots and try to assign teacher
        if (!availableTimeslots.isEmpty()) {
            try {
                TimeslotAssignmentDto availableTimeslot =
                ↪  availableTimeslots.get(0);
                List<CompletableFuture<Void>> futures = new ArrayList<>();
                List<TeacherAssignmentDto> unassignedTeachers =
                ↪  unassignedTeachers(schedule, teachers);

                boolean logicState = !unassignedTeachers.isEmpty() &&
                ↪  unassignedTeachers.size() - availableTimeslots.size() ==
                ↪  0;
                teachersLogic(
                        stack,
                        schedule,
                        usedPreferences,
                        allTimeslots,
                        logicState ? unassignedTeachers : teachers,
                        availableTimeslot,
                        futures,
                        logicState,
                        state.shuffle()
                );

                CompletableFuture.allOf(futures.toArray(new
                ↪  CompletableFuture[0])).get();
            } catch (InterruptedException | ExecutionException e) {
                e.printStackTrace();
            }
        }

        return null;
    }
```

Code listing 3 The `processState` method in the `ScheduleGenerationService`

```
    private Boolean validateRanges(Multimap<String, TimeslotAssignmentDto>
    ↪  teacherTimeslotMap, Map<String, Long> usedPreferences) {
    try {
        List<CompletableFuture<Void>> futures = new ArrayList<>();

        for (Map.Entry<String, Collection<TimeslotAssignmentDto>> entry :
        ↪  teacherTimeslotMap.asMap().entrySet()) {
            CompletableFuture<Void> future = CompletableFuture.runAsync(()
            ↪  -> {
                if (!teacherGenerationService.validateRanges(
                    new ArrayList<>(entry.getValue()),
                    entry.getKey(),
                    usedPreferences.get(entry.getKey())
                )) {
                    throw new
                    ↪  RuntimeException("Validation failed for teacher "
                    ↪  + entry.getKey());
                }
            });
            futures.add(future);
        }

        CompletableFuture.allOf(futures.toArray(new
        ↪  CompletableFuture[0])).get();
    } catch (InterruptedException | ExecutionException e) {
        return false;
    }
    return true;
}
```

Code listing 4 The `validateRanges` method in the `ScheduleGenerationService` class

```
    private void teachersLogic(Stack<BackTrackState> stack,
                               PossibleScheduleDto schedule,
                               Map<String, Long> usedPreferences,
                               List<TimeslotAssignmentDto> timeslots,
                               List<TeacherAssignmentDto> teachers,
                               TimeslotAssignmentDto timeslot,
                               List<CompletableFuture<Void>> futures,
                               boolean unassigned,
                               boolean shuffle){
        if(shuffle)
            Collections.shuffle(teachers);

        for (TeacherAssignmentDto teacher : teachers) {
            // if teacher's already assigned
            if (!unassigned &&
                usedPreferences.containsKey(teacher.getUsername())) {
                CompletableFuture<Void> future = CompletableFuture.runAsync(()
                ↪  -> {
                if (wantTeach(usedPreferences, timeslot, teacher)) {
                    PossibleScheduleDto newSchedule = createNewSchedule(
                        schedule,
                        timeslot,
                        teacher,
                        usedPreferences.get(teacher.getUsername())
                    );
                    stack.push(new BackTrackState(newSchedule, timeslots,
                    ↪  teachers, shuffle));
                }});
                futures.add(future);
                continue;
            }

            iterateThroughTeacherPreferences(stack, schedule, timeslots,
            ↪  teachers, timeslot, futures, teacher, shuffle);
        }
    }
```

■ **Code listing 5** The `teachersLogic` method in the `ScheduleGenerationService`

```
private void iterateThroughTeacherPreferences(
                Stack<BackTrackState> stack,
                PossibleScheduleDto schedule,
                List<TimeslotAssignmentDto> timeslots,
                List<TeacherAssignmentDto> teachers,
                TimeslotAssignmentDto timeslot,
                List<CompletableFuture<Void>> futures,
                TeacherAssignmentDto teacher,
                boolean shuffle
    ) {
    teacher.getTeacherSchedules().forEach(teacherSchedule -> {
        if (teacherSchedule.getValid() &&
        ↪   teacherGenerationService.wantTeach(timeslot, teacherSchedule))
        ↪   {
            CompletableFuture<Void> future = CompletableFuture.runAsync(()
            ↪   -> {
                PossibleScheduleDto newSchedule = createNewSchedule(
                    schedule,
                    timeslot,
                    teacher,
                    teacherSchedule.getId()
                );
                stack.push(new BackTrackState(newSchedule, timeslots,
                ↪   teachers, shuffle));
            });
            futures.add(future);
        }
    });
}
```

Code listing 6 The `iterateThroughTeacherPreferences` method in the `ScheduleGenerationService` class

- The optimization of reviewing the ranges of taken time slots per week or day of the preferred schedule of the teacher is implemented in the `validateRanges` method. This method is used before the process of assigning the teacher to a new time slot begins in the `processState` method.

4.3.4 Improvement of performance

The design chapter states that parallel computation and asynchronous programming techniques could be used to improve the performance of the algorithm.

These techniques were used during the implementation of the prototype using the `Stream API` and `CompletableFuture` class.

As described in the [13], the `Stream API` provides a way to achieve parallelism through the `parallel()` method, which can be invoked on a stream to enable parallel processing. With parallel processing, the stream is split into multiple sub-streams that are processed on separate threads, thus improving processing efficiency. To be more precise, the method `parallel()` was used several times in the `generateSchedules` during the iterations in the `while` loop.

`CompletableFuture` is a class in the Java programming language that provides a way to perform asynchronous and concurrent programming. It is a part of the Java Concurrency API and was introduced in Java 8.[13] The `runAsync()` method allows you to execute a task asynchronously using a `ForkJoinPool` or a specific executor. The task is specified by passing a lambda expression as an argument to the `runAsync()` method. When `runAsync()` is called, it returns a `CompletableFuture<Void>` object right away, which represents the asynchronous computation that will eventually be complete when the task is finished. This approach was used in the functions `processState`, `iterateThroughTeacherPreferences`, and `teachersLogic`.

4.4 UI implementation

4.4.1 Front-end framework

A front-end framework is essentially a set of pre-built tools and resources that developers can use to create the user interface and user experience of a web application. It is designed to simplify and speed up the development process by providing pre-written code, libraries, and templates that can be easily reused. The suitable easy-to-use framework would be Bootstrap.

4.4.1.1 Boostrap

“**Bootstrap** is an open-source CSS framework directed at responsive, mobile-first front-end web development. It contains HTML, CSS, and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.”[25]

The advantages of the Boostrap are as follows:

- Provides a flexible and adaptable grid system for responsive design.
- Offers consistency in styles and components for improved user experience.
- Highly customizable with pre-built components and templates.

4.4.2 Summary

Why the Bootstrap and Thymeleaf should be used together? Using Bootstrap with Thymeleaf in web application development offers several advantages. Bootstrap provides a set of pre-built UI components that can be quickly integrated with Thymeleaf templates, allowing developers to create web pages rapidly. It also ensures that web pages are optimized for mobile devices and provides a consistent and professional design across different browsers and devices.

Chapter 5

Testing

In this chapter, the implemented system is discussed in terms of testing, as well as outlining possible improvements for future development.

5.1 User tests

To understand the current problems of the prototype and identify areas for future improvement, user tests should be conducted. Teachers with different academic degrees from universities and schools were selected to test the functionality of ”Chronosite”. The implemented prototype ran on one laptop to perform the tests. Firstly, the author played the role of the administrator, while the teachers played the role of teachers. Secondly, the author became a teacher and the teachers took turns being the administrator. Each individual tested the prototype without communicating in real time.

The author established the scenarios for each role to test the common functionality that covers the use cases:
Admin:

1. Try to create teachers and update or delete one of them.
2. Try to create a timetable.
3. Try to edit the created timetable in step 2 and add some time slots.
4. Try to edit the generated possible schedules of the created timetable in step 2. Choose the one that is most logical for you and approve it.
5. Check the notifications and try to read the messages of several notifications.
6. Wait for feedback from all the involved teachers. If all the teachers have approved the proposed schedule, check the created timetable in step 2 with the assigned teachers for the time slots.

Teacher:

1. Try editing the provided profile from the administrator, for example, by changing the password.
2. Try to browse through the timetables created by the administrator.
3. Try to create a preferred schedule for a specific timetable.

4. Try to edit and validate the preferred schedule created in step 3. For example, change the range of taken time slots for all days.
5. Wait until the administrator generates possible schedules and approves one of them. Then, try accepting or rejecting the schedule chosen by the administrator, and don't forget to provide a message.

The important questions were discussed with selected teachers after successfully passed scenarios:

1. Was the location of the UI elements or the flow of the actions intuitive or sometimes additional instructions were needed?
2. What were the things or features that you like the most?
3. Do you feel like several features or some functionality were missing?

The primary objective of the provided questions was to assess the implemented design, identify any potential improvements, and determine what aspects users appreciate the most in the created prototype as well as any issues they perceive.

The responses provided by the users were as follows:

User 1

1. The location of the UI elements and the flow of the actions were intuitive. However, I did need additional instructions in a few cases, particularly when trying to validate the created preferably schedule in the teacher role scenario.
2. I appreciated the search functionality that allowed me to quickly create a suitable schedule in the teacher's scenario. I also liked the generation of the possible schedule process in the admin's scenario with the feedback from the teacher.
3. The prototype seemed to have all the necessary features for assigning teachers to the timetable slots. However, the potential missing thing is the lack of information about the provided time slots by the administrator. They contain only start and the end time, I suggest adding at least one location to the time slot information.

User 2

1. Overall, everything was quite intuitive, and I don't have any negative feedback regarding the location of the UI elements and the flow of actions.
2. The prototype has great functionality and useful features. I extremely appreciate the ability to create multiple suitable schedules in the teacher's scenario, it gives the flexibility to isolate and submit all of my preferences.
3. While the prototype covered most of the necessary functionality for assigning time slots, I did feel like there could have been more options for specifying more complex teacher preferences.

5.2 Future improvements

Considering the results of the testing and the feedback from the teachers who were involved in the testing process, the author defined the following future improvements:

- **User Experience (UX)** Some parts of the UI were intuitive for the users, however, more user tests could be performed to identify which UI and UX elements could make ”Chonosite” more user-friendly.

- **Additional constraits** The teacher should have more options to specify their preferences. This improvement would increase the number of strategies available to create preferable schedules. One possible approach is to customize the constraints according to the organization. However, this improvement is sensitive to the chosen organization and may not be useful to other organizations.
- **The nearest possible schedule** Perhaps the most necessary improvement is to provide functionality for generating possible schedules when it is impossible to satisfy all teachers' preferences. These schedules would satisfy the maximum possible percentage of teachers' preferences but not all of them. This functionality could be provided if teachers' preferences were assigned levels of importance, and preferences with lower priority would be omitted first.

Chapter 6
Conclusion

The objective of this thesis was to develop a system that can assign teachers to timetable slots based on their preferences. Once all the necessary processes, such as gathering preferable schedules from teachers, generating possible scheduling, and reviewing the schedule, are completed, the application creates a completed timetable with assigned teachers to the time slots. The thesis was structured into several logical parts, with each part beginning with an explanation of fundamental concepts and relevant technologies that are pertinent to the work. This approach was adopted to aid in achieving the overall objective of the thesis.

The first step was to analyze the timetable creation process, including its key steps and processes. Additionally, this part formulated detailed functional and non-functional requirements and use cases, as well as analyzed existing solutions.

Using this information, the application's design was created, including the system architecture, server, and UI layer and functional modules such as the key algorithm with optimizations, and data management. These architectures were designed based on the previous analysis and modern best practices.

After the system was designed, it was implemented using suitable libraries. All the analyzed requirements and use cases, such as user authentication, an algorithm for timetable generation, and receiving preferable schedules from teachers, were successfully implemented. Advanced optimizations were also made to the backtracking algorithm to improve the performance and stability of the schedule generation process.

Finally, the system's correctness and usability were tested via user testing. As expected, the current state of the prototype does not allow it to be released due to its limited functionality. However, in the course of working on this thesis, a good foundation was created that makes the prototype easily extensible.

Bibliography

1. KARL, Wiegers; JOY, Beatty. *Software requirements.* Pearson Education, 2013. ISBN 978-0-7356-7966-5.
2. *Doodle (website)* [online]. Wikimedia Foundation, 2023 [visited on 2023-03-14]. Available from: `https://en.wikipedia.org/wiki/Doodle_(website)`.
3. GARRETT, Jesse James. *The Elements of User Experience.* Peachpit Pr, 2002. ISBN 0-321-68368-4.
4. BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. *Software architecture in practice.* Addison-Wesley Professional, 2003. ISBN 978-0321154958.
5. NEWMAN, Sam. *Monolith to microservices: evolutionary patterns to transform your monolith.* O'Reilly Media, 2019. ISBN 978-1492047841.
6. VAN STEEN, Maarten; TANENBAUM, A. Distributed systems principles and paradigms. *Network.* 2002, vol. 2, no. 28, p. 1.
7. NEWMAN, Sam. *Building microservices.* O'Reilly Media, Inc., 2021. ISBN 978-1492034025.
8. FOWLER, Martin. *Patterns of Enterprise Application Architecture.* Addison-Wesley, 2012. ISBN 978-0321127426.
9. FRED, Glover; CLAUDE, McMillan. The general employee scheduling problem. An integration of MS and AI. *Computers & operations research.* 1986, vol. 13, no. 5, pp. 563–573.
10. THOMAS, H.Cormen; CHARLES, E.Leiserson; RONALD, L.Rivest; CLIFFORD, Stein. *Introduction to algorithms.* MIT press, 2022. ISBN 978-0-262-03384-8.
11. GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; JOHNSON, Ralph E; VLISSIDES, John. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley, 1995. ISBN 978-0201633610.
12. FLANAGAN, David; NOVAK, Gregor M. *Java-Script: The Definitive Guide.* O'Reilly Media, 1998. ISBN 978-1565923928.
13. SCHILDT, Herbert; COWARD, Danny. *Java: the complete reference.* McGraw-Hill Education New York, 2014. ISBN 978-0071808552.
14. *Collections Framework* [online]. 2023. [visited on 2023-05-09]. Available from: `https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html`.
15. *Stream (java platform SE 8)* [online]. 2023. [visited on 2023-05-07]. Available from: `https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html`.

16. *Package java.util.concurrent* [online]. 2023. [visited on 2023-05-09]. Available from: `https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html`.
17. CANTELON, Mike; HARTER, Marc; HOLOWAYCHUK, TJ; RAJLICH, Nathan. *Node.js in Action*. Manning Greenwich, 2014. ISBN 978-1617290572.
18. *The good and the bad of Node.js Web App Programming* [online]. AltexSoft, 2023 [visited on 2023-05-10]. Available from: `https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/`.
19. *Google Guava* [online]. Wikimedia Foundation, 2023 [visited on 2023-05-09]. Available from: `https://en.wikipedia.org/wiki/Google_Guava`.
20. *Lombok* [online]. Project Lombok, 2023 [visited on 2023-05-10]. Available from: `https://projectlombok.org/`.
21. WALLS, Craig. *Spring in action*. Simon and Schuster, 2022. ISBN 978-1617297571.
22. *Jakarta EE* [online]. Wikimedia Foundation, 2023 [visited on 2023-05-08]. Available from: `https://en.wikipedia.org/wiki/Jakarta_EE`.
23. *Google Guice* [online]. Wikimedia Foundation, 2023 [visited on 2023-05-09]. Available from: `https://en.wikipedia.org/wiki/Google_Guice`.
24. *Thymeleaf*. Wikimedia Foundation, 2023. Available also from: `https://en.wikipedia.org/wiki/Thymeleaf`.
25. *Bootstrap (front-end framework)*. Wikimedia Foundation, 2023. Available also from: `https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)`.

Contents of enclosed media

```
readme.txt ........................ a brief description of the media content
src ....................................... the source codes of the work
├── impl ............................ the directory of implementation source codes
└── thesis ........................ the directory of LaTeX source codes of the work
text .............................................. the text of the work
└── thesis.pdf ........................ the text of the work in the PDF format
```