

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Integrating FelSight into FEL.HUB

Tomáš Hauser

Thesis supervisor: **Ing. Miroslav Balík, Ph.D.**

Department of Computer Science

MAY 2024

I. Personal and study details

Student's name: **Hauser Tomáš** Personal ID number: **492159**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Computer Science**
Study program: **Open Informatics**
Specialisation: **Software Engineering**

II. Master's thesis details

Master's thesis title in English:

Integrating FelSight into FEL.HUB

Master's thesis title in Czech:

Integrace aplikace FelSight do FEL.HUB

Guidelines:

The aim of this thesis is to integrate the core functionalities of FelSight into the FEL.HUB integration platform. In particular, the implementation is expected to include timetable viewing, tools for timetable creation, and searching for courses. The backend will be based on the work from the bachelor's thesis of Ladislav Svoboda who split the FelSight backend into microservices. The UI designs are provided by Lucie Baronová, who created them based on UX research in her bachelor's thesis.

1. Describe the motivations behind the project.
2. Conduct a software design analysis.
3. Implement the front-end interface based on the provided UI designs.
4. Adapt and fit the provided microservices into the FEL.HUB ecosystem.
5. Develop and analyze an algorithm to help students pick an optimal set of time slots in a timetable.
6. Describe what improvements were made over the existing system.
7. Demonstrate the resulting algorithm for timetable optimization on a few practical examples of real timetables.
8. Compare the actual frontend implementation with the UI designs from Lucie Baronová.

Bibliography / sources:

1. Ladislav Svoboda. Migration of felsight application to microservice architecture. Bachelor's thesis, Czech Technical University in Prague, 2023. URL <https://dspace.cvut.cz/handle/10467/109283>.
2. Gardner, Micah. Frontend Architecture for Design Systems: A Modern Blueprint for Scalable and Sustainable Websites. O'Reilly Media, 2016. ISBN 978-1491926783.
3. Blazewicz, Jacek, Klaus Ecker, Erwin Pesch, Guenter Schmidt, a Jan W glarz. 2001. Scheduling Computer and Manufacturing Processes. <https://doi.org/10.1007/978-3-662-04363-9>.

Name and workplace of master's thesis supervisor:

Ing. Miroslav Balík, Ph.D. Department of Theoretical Computer Science FIT

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **07.02.2024** Deadline for master's thesis submission: _____

Assignment valid until: **21.09.2025**

Ing. Miroslav Balík, Ph.D.
Supervisor's signature

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature



I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date
.....



Acknowledgements

I am deeply grateful to my family for their unwavering support during my academic journey. I would also like to extend my sincere thanks to my advisor, Ing. Miroslav Balík, Ph.D., for his guidance and support throughout the writing of this thesis.

Abstract

This thesis addresses the migration of essential functionalities from the web application FelSight to the modern integration platform FEL.HUB, leveraging the backend and user interface designs produced in prior works. FelSight is introduced, highlighting the components targeted for migration and discussing its limitations along with the corresponding metrics. Introduction of FEL.HUB follows with a description of its purpose and technologies, concluding with a list of the motivations behind this project. The analytical part dissects the problem domain into use cases that are subsequently elaborated in detail. A comprehensive overview of the backend implementation requirements is also presented. The frontend implementation is showcased via annotated screenshots, which are then compared to the original designs. Additionally, a dedicated chapter addresses the development and evaluation of an algorithm designed to help students in creating their timetables. This includes a review of the algorithm's complexity and performance metrics, concluding with a practical application of the algorithm on an actual timetable.

Keywords: FelSight, FEL.HUB, timetable optimization

Abstrakt

Tato práce se zabývá migrací základních funkcionalit z aplikace FelSight na moderní integrační platformu FEL.HUB za využití backendové implementace a návrhů uživatelského rozhraní z předcházejících prací. Aplikace FelSight je představena společně s identifikací komponent určených k migraci a diskuzí o jeho nedostatecích podloženou příslušnými metrikami. Představení platformy FEL.HUB následuje popisem jeho záměru a technologií a je zakončeno seznamem motivací za tímto projektem. Analytická část je věnována rozčlenění problému do případů užití, které jsou následně rozebrány do detailu. Spolu s tím je také prodiskutováno, co je potřeba implementovat na backendu. Samotná implementace je představena seznamem komentovaných snímků obrazovky, které jsou nakonec porovnány s původním návrhem. Jedna kapitola je věnována návrhu algoritmu, který má studentům pomoci s plánováním rozvrhu, což zahrnuje analýzu jeho složitosti a výkonu. Výsledný algoritmus je demonstrován na reálném rozvrhu.

Klíčová slova: FelSight, FEL.HUB, optimalizace rozvrhů

Contents

List of abbreviations	vii
1 Introduction	1
1.1 Goals	1
1.2 Outline of the Thesis	2
2 FelSight	3
2.1 Content	4
2.1.1 Timetable	4
2.1.2 Timetable Planner	5
2.1.3 Building Plans	6
2.1.4 Food Menus	6
2.1.5 Study Rooms	7
2.1.6 Semester Overview	7
2.1.7 Summary	7
2.2 Technologies	8
2.2.1 Preface	8
2.2.2 Example	8
2.2.3 Architecture	11
2.3 Issues	12
2.3.1 Complexity	12
2.3.2 Size	13
2.3.3 Performance	14
2.3.4 Personal	15

3	FEL.HUB	17
3.1	General Idea	17
3.2	Technologies	19
3.2.1	SPA and PWA	19
3.2.2	React	20
3.2.3	GraphQL	23
3.2.4	Java, SpringBoot, DGS	25
3.3	Architecture	25
3.4	Motivations Behind The Project	26
4	Analysis	27
4.1	Use Cases	28
4.1.1	Timetable Views	29
4.1.2	Timetable controls	29
4.1.3	Searching	29
4.1.4	User Events	30
4.1.5	Timetable Irregularity	30
4.1.6	Details	30
4.2	Backend	31
4.2.1	Timetable Service	32
4.2.2	Course Semester Service	34
4.2.3	Room Service	36
4.2.4	User Service	36
5	Frontend Implementation	37
5.1	Timetable Views	40
5.1.1	Weekly Timetable View	40
5.1.2	Collision Management	42
5.1.3	Monthly Timetable View	43
5.1.4	Timetable Views On Phones	44
5.2	Timetable Controls	45
5.2.1	Switching Time	45
5.2.2	Timetable Page Controls	46

5.2.3	Timetable Planner Controls	47
5.2.4	Timetable Controls On Phones	48
5.3	Search	49
5.3.1	Grouped Search	49
5.3.2	Advanced Search	49
5.4	User Events	52
5.5	Timetable Irregularity	53
5.6	Details	53
5.6.1	Event Card Detail	53
5.6.2	Person Detail	54
5.6.3	Room Detail	55
5.6.4	Course Detail	56
5.7	Comparison With Designs	57
6	Timetable Optimization	59
6.1	Initial Solution	59
6.2	Complexity	62
6.3	Implementation	63
6.4	Example	66
7	Conclusion	67
7.1	Improvements	69

List of abbreviations

Table 1 contains abbreviations used in this thesis.

Abbreviation	Meaning
CZM	Center for Knowledge Management
JSF	Java Server Faces
EL	Expression Language
TTFB	Time To First Byte
UI	User Interface
UX	User Experience
HTML	Hypertext Markup Language
XHTML	Extensible HTML
CSS	Cascading Style Sheets
CI/CD	Continuous Integration, Continuous Development
API	Application Programming Interface
SPA	Single Page Application
MPA	Multi-Page Application
PWA	Progressive Web Application
JSX	JavaScript Syntax eXtension
GQL	GraphQL
REST	Representational State Transfer
DB	Database
AMQP	Advanced Message Queuing Protocol
<i>TO</i>	Timetable Optimization algorithm

Table 1: List of abbreviations

Chapter 1

Introduction

Contents

1.1	Goals	1
1.2	Outline of the Thesis	2

1.1 Goals

FelSight¹ is a web application developed by Center for Knowledge Management². It provides a suite of tools, including timetable viewing, planning, room reservation, and searching for rooms, people, and courses. Although FelSight has served well throughout the years, it is in need of renovation for the reasons that will be elaborated upon.

FEL.HUB is an integration platform whose goal is to integrate and unify a broad range of faculty systems under one roof. It offers tools for teachers such as managing doctoral theses, semester projects, employee evaluation, as well as for students – room navigation and room reservation. It is a relatively young project that employs the latest technologies and is designed with scalability and long-term sustainability at its core.

The objective of this thesis is to transfer certain essential components of FelSight into the FEL.HUB integration platform. The user interface (UI) should be developed following Lucie Baronová’s designs as outlined in her thesis [1]. The backend will be based on the work by Ladislav Svoboda who has split some of the core functionalities of FelSight into microservices [2].

Another objective is to develop an optimization algorithm that will help students with creating their timetables.

¹Available at <https://felsight.fel.cvut.cz/>

²More information at <https://czm.fel.cvut.cz/cs/>

1.2 Outline of the Thesis

In chapter 2, FelSight is introduced. To give a comprehensive overview of its functionalities and facilitate comparisons with subsequent implementations, the chapter includes screenshots accompanied by concise descriptions. A table 2.1 then summarizes the migration status of the individual sections. Technologies with which FelSight was built are presented on simple examples with a commentary on their downsides. The entire architecture is depicted in the diagram 2.9. A list of issues is presented, accompanied by relevant metrics.

Chapter 3 introduces FEL.HUB. Its technologies and architecture are explained in a manner similar to the previous chapter. The chapter concludes with a list of motivations behind the project, building upon the knowledge established thus far.

Since the frontend implementation takes basis on the provided designs, the analysis chapter 4 focuses on identifying the use cases – actions that user can take in the system. This provides a comprehensive checklist of features, allowing for each feature to be ticked off as it is implemented, ensuring that no detail is overlooked. Backend implementation, on the other hand, requires creating a GraphQL schema and implementing logic for its queries. The backend analysis, therefore, consists of describing the relevant services and specifying queries that need to be implemented.

Since the implementation of the backend simply entails linking the queries outlined in the analysis stage with the service layer, there are no developments or complications to highlight or discuss. In contrast, the frontend implementation warrants its own dedicated chapter (5), where screenshots of the frontend are showcased, and both implementation details and complications are thoroughly explored. One section (5.7) is dedicated to outlining the deviations from the original designs.

The design of the timetable optimization algorithm is detailed in a separate chapter (6) that includes its own introduction, analysis, and implementation sections. This chapter revisits the previous solution, highlighting its shortcomings. The algorithm itself is described and placed within a complexity class. Initially, a general brute force algorithm is implemented and later optimized for enhanced performance. The chapter concludes with a demonstration of the algorithm on a real timetable.

Finally, the conclusion chapter (7) summarizes the results of this project and highlights the improvements that have been made (7.1).

Chapter 2

FelSight

Contents

2.1	Content	4
2.2	Technologies	8
2.3	Issues	12

As previously stated in the introduction, FelSight is a web application designed to assist students with their academic pursuits. It is focused on providing useful information via gathering, transforming and presenting data from various faculty APIs. The objective of this chapter is to provide a brief overview and demonstration of the key features of FelSight, identifying those that will be the focus of this thesis. In order to understand the reasons behind this project, a brief and simple example will serve to describe the nuance of the technologies in use. Subsequently, the architecture is described. Following the provision of all essential information, the challenges confronting FelSight are addressed.

2.1 Content

2.1.1 Timetable

The view of the weekly timetable occupies the main page of FelSight.

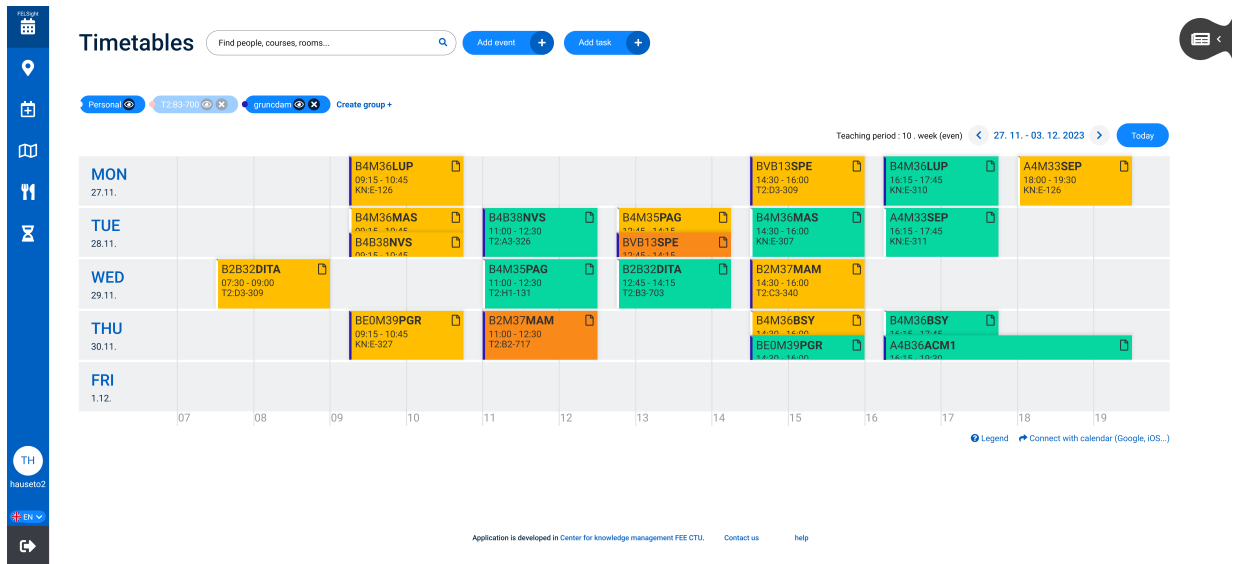


Figure 2.1: Timetable page on FelSight

The cards in the timetable represent events such as lectures, tutorials and user events. The arrows located above the timetable are utilized for navigating between different weeks. The rectangles with the room code and username above the timetable represent the owners of the individual timetables. Each group is marked by a color that is also visible in the strip on the cards. Using the search at the top (2.2, 2.3), new owners can be added.

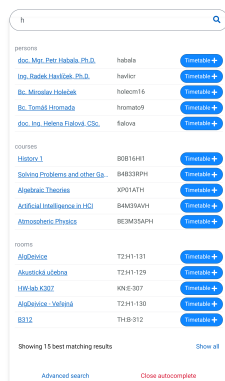


Figure 2.2: Search component

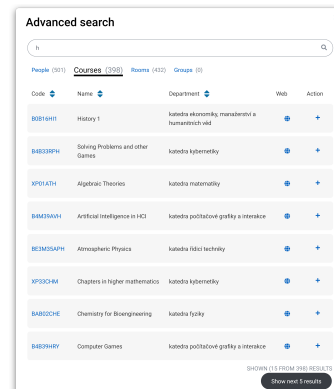


Figure 2.3: Advanced search component

2.1.2 Timetable Planner

The timetable planner offers a convenient method to easily experiment with possible new timetables.

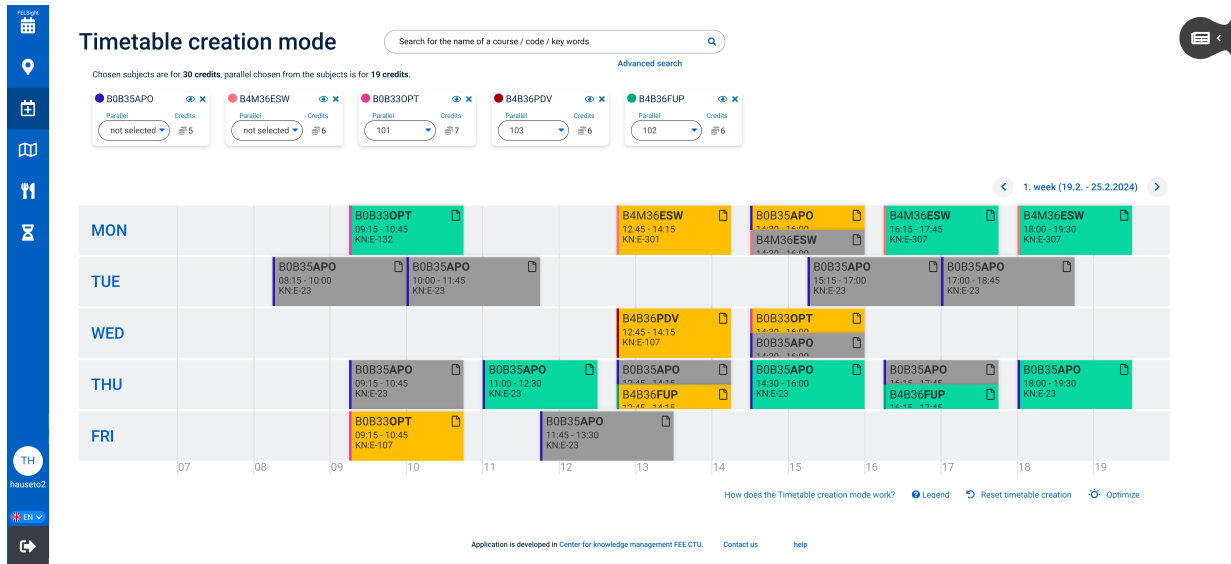


Figure 2.4: Timetable planner page on FelSight

For a student, planning a timetable for the next semester consists of choosing courses to meet their credit, time and interest demands. Subsequently, the student has to choose a parallel for each course. If a course has non-standard time slots, a notification will appear (2.5), and either a week parity switch (5.26) or a full week switch (2.6) will be displayed accordingly.

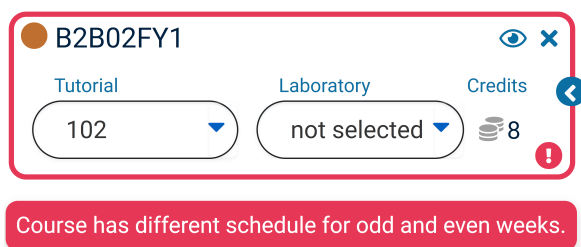


Figure 2.5: Course card with warning

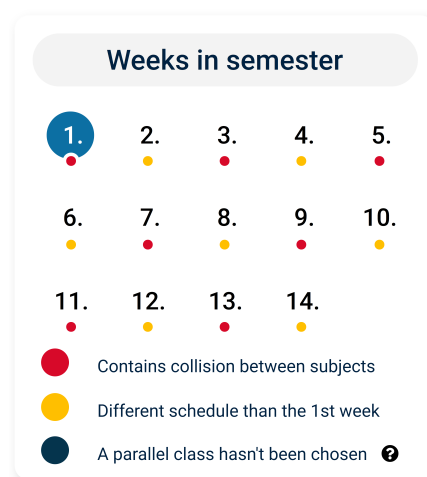


Figure 2.6: Week switch

2.1.3 Building Plans

In order to navigate through the different CTU buildings, students can use the building plans section.



Figure 2.7: Building plans on FelSight

It was built as a plugin that can be easily imported into any website on CTU network [3]. For that reason, it is already available at FEL.HUB as well. The main difference from the official building plans is the interactivity. The user selects an origin and a destination, and the system will show the most efficient route between the two locations. The application is also running on a stationary device near the entrance to Dejvice kampus and Karlovo Náměstí.

2.1.4 Food Menus

This section provides users with a convenient way to access information about the various food choices available at all the relevant canteens in one place. There is a dedicated service¹ in the FelSight stack that fetches data about the food menus from an official Agata² API. Incorporating food menus into FEL.HUB is currently in the planning phase. The UI designs for this feature were aimed primarily at smartphones. FelSight did not use to support displaying the food menus on desktop, and later simply scaled the mobile version up.

¹<https://gitlab.fel.cvut.cz/czm/hub/jidelnický/food-menu-service>

²<https://agata.suz.cvut.cz/jidelnický/>

2.1.5 Study Rooms

Every room at CTU FEL follows a schedule. While some rooms can be booked for studying during specific times, others cannot. The study rooms section provides features that help in identifying available rooms and making reservations. Unfortunately, there is no dedicated API for this, therefore, a raw HTML lookup of a live table on the faculty intranet³ is done.

2.1.6 Semester Overview

One of the less utilized functions is the semester overview. It is presented as a timeline featuring significant events at the faculty level, such as semester applications and graduation ceremonies. This functionality has been deprecated and will not be transferred.

2.1.7 Summary

The following table provides a summary of the migration status for each section.







Feature	Not included	Considered	Implemented now	Done
 Timetable			✓	
 Study rooms		✓		
 Timetable Planner			✓	
 Building plans				✓
 Food menu		✓		
 Semester overview	✓			

Table 2.1: Overview of the state of the FelSight migration into FEL.HUB

Implementing the scheduling features—timetable and timetable planner—is the primary goal of this thesis. As discussed in section 2.1.3, building plans have been successfully integrated into FEL.HUB thanks to their adaptability. The semester overview will not be implemented, reflecting its low usage statistics. Food menus are under consideration and primarily require new UI designs since the backend can be reused. Study rooms are also being considered; however, this feature will likely require significant redevelopment.

³<https://intranet.fel.cvut.cz/cz/education/studovny-samostudium.html>

2.2 Technologies

2.2.1 Preface

FelSight utilizes the Java Server Faces (JSF) framework in conjunction with a Java Enterprise backend. To begin with, JSF is a web framework that streamlines development through its component-based methodology [4]. The components are defined within *.xhtml* documents and their state is managed by dedicated Java classes on the server side. These documents act as a template and are converted into HTML on the server upon client's request.

2.2.2 Example

To demonstrate the internal mechanisms and limitations of the technology in use, consider a basic component designed to switch between even and odd weeks (2.8). Its implementation using XHTML is shown in the listing 2.1. Below is a brief description of the operations performed, organized by the range of lines in the code.

- 4–9: Declaration of used libraries.
- 10–15: Specification of the parameters necessary for this component.
- 17–46: Declaration of the component itself. It consists of two buttons labeled *Even* and *Odd*.
- 18, 30: Declaration of a button from the Primefaces library.
- 19: Server-side action to be taken upon clicking. A `changeParityToEven` method is called.
- 20: Client-side action to be taken upon clicking. A timetable loading icon is displayed.
- 21: Ids of elements that need to be updated after the action takes place.
- 22: Client-side action to be taken at the end of JSF processing cycle.
- 23–29: References to CSS classes to be used to style the button. The ternary operator expression within the curly braces utilizes EL⁴ for branching.



Figure 2.8: Week switch component

⁴EL (Expression Language) is a mechanism that facilitates communication between the presentation and application layer. EL expressions are enclosed in curly braces and prefixed with a hash symbol. [5]

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5     xmlns:cc="http://java.sun.com/jsf/composite"
6     xmlns:p="http://primefaces.org/ui"
7     xmlns:h="http://xmlns.jcp.org/jsf/html"
8     xmlns:a="http://xmlns.jcp.org/jsf/passthrough"
9 >
10 <cc:interface>
11     <cc:attribute name="controller"
12         type="felsight.beans.interfaces.WeekswitchController"
13         required="true"/>
14 </cc:interface>
15
16 <cc:implementation>
17     <h:form styleClass="timetable-controls-week-switch time-controls">
18         <p:commandLink value="#{lang['timetable.create.value.EvenWeek']}"
19             action="#{cc.attrs.controller.changeParityToEven}"
20             onclick="$(' .timetable-loader').show();"
21             update="@form :timetable:timetable-days"
22             oncomplete="refreshTimetable();"
23             class="timetable-controls-parity-button
24                 timetable-controls-even
25                 #{
26                     cc.attrs.controller.weekEven ?
27                     'timetable-controls-selected' :
28                     'timetable-controls-not-selected'
29                 }"/>
30         <p:commandLink value="#{lang['timetable.create.value.OddWeek']}"
31             action="#{cc.attrs.controller.changeParityToOdd}"
32             update="@form :timetable:timetable-days "
33             onclick="$(' .timetable-loader').show();"
34             oncomplete="refreshTimetable();"
35             class="timetable-controls-parity-button
36                 timetable-controls-odd
37                 #{
38                     cc.attrs.controller.weekEven ?
39                     'timetable-controls-not-selected' :
40                     'timetable-controls-selected'
41                 }"/>
42     </h:form>
43 </cc:implementation>
44 </html>
```

Listing 2.1: JSF component example

Listing 2.2 shows the mentioned controller. It is in a form of a simple Java interface that defines the necessary methods.

```
1 public interface WeekSwitchController {
2     boolean isWeekEven();
3     void changeParityToEven();
4     void changeParityToOdd();
5 }
```

Listing 2.2: Controller for a JSF component

Controllers of this type are subsequently implemented using Java beans, which are specialized classes that adhere to specific conventions. The crucial aspect is that they manage the state of the user interface elements and execute the necessary logic within them. Because the server maintains the state and the user interface operates on the client machine, there needs to be ongoing communication between them. This delay in reacting to state changes slows down the response time, as the message needs to travel to the server and back for the UI to respond accordingly. This ultimately hampers responsiveness, user experience, and leads to higher data throughput.

```
1 @Named
2 @ViewScoped
3 public class TimetableBean implements WeekSwitchController {
4     // ... some code
5     private WeekParity weekParity;
6     public boolean isWeekEven() {
7         return this.weekParity.equals(WeekParity.EVEN);
8     }
9     public void changeParityToEven() {
10        // Some logic goes here
11        this.weekParity = WeekParity.EVEN;
12    }
13    public void changeParityToOdd() {
14        // Some logic goes here
15        this.weekParity = WeekParity.ODD;
16    }
17 }
```

Listing 2.3: Java Bean

Styles and JavaScript functions are stored in separate files and, upon request, are bundled into a single file that is sent to the client.

2.2.3 Architecture

FelSight gathers data from several external APIs. The figure 2.9 illustrates the architecture in detail. The arrows represent a “uses” relationship. The text adjacent to the arrows explains the purpose of using the service, while the text enclosed in parentheses specifies the communication protocol or type. The services are organized into two categories: the external category on the right, which includes services outside the management of CZM, and the internal on the left.

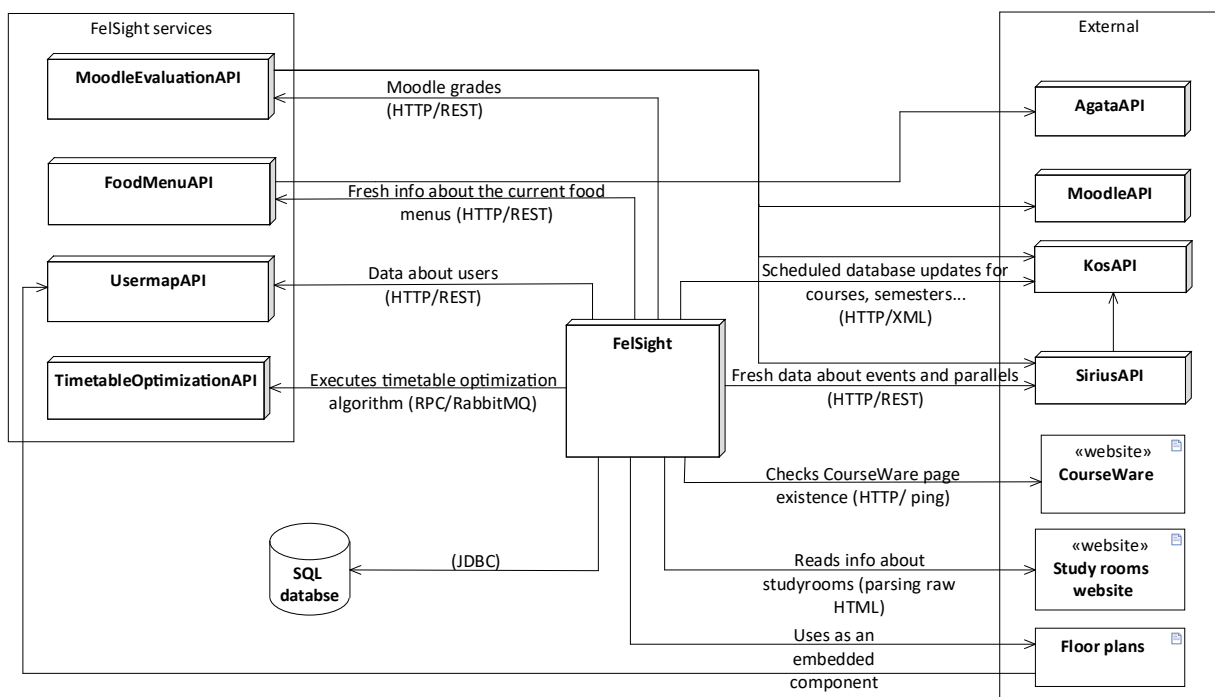


Figure 2.9: Detailed view on FelSight’s architecture

Although the architecture seems rather modular, the internal services simply fetch data from external APIs, combine them, and expose them at convenient endpoints⁵. They do not store any data and they cover only a small portion of the FelSight’s domain.

Most of the logic is contained inside the monolithic Java backend and all data are stored in a single database. The stored data are periodically refetched to remain up to date, whereas data that need to be fresh, such as parallel availability, are fetched directly from the data source. The database currently consists of almost 50 tables and often requires manual maintenance due to frequent migration issues.

⁵This is referred to as *API composition pattern* [6].

2.3 Issues

Firstly, the complexity of JSF is addressed with an in-depth exploration of its intricate processing lifecycle. Following that, the size of the project is discussed, accompanied by measurements including the number of files, lines, and other relevant statistics. Subsequently, the project's performance is measured and the challenges associated with hiring new developers are discussed.

2.3.1 Complexity

Building a complex reactive frontend using JSF comes at a cost. For example, in the code 2.1, notice the `update` property – it is required to manually specify the ids of elements that require updating. Moreover, when an action takes place, the request processing lifecycle comes into play. The process is quite intricate and involves several stages that commence when JSF needs to handle a client request. That in and of itself would not necessarily be an issue, however, the programmer has to keep the individual stages in mind during development. They need to know in what order will the individual actions take place. For example, on the 18th line in the code example 2.1, the order of the actions will be `onclick`, `action`, `update`, `oncomplete`. There are also other attributes that can be fit in between these actions. To elaborate on this, the diagram in the figure 2.10 demonstrates the specific stages as outlined by [4].

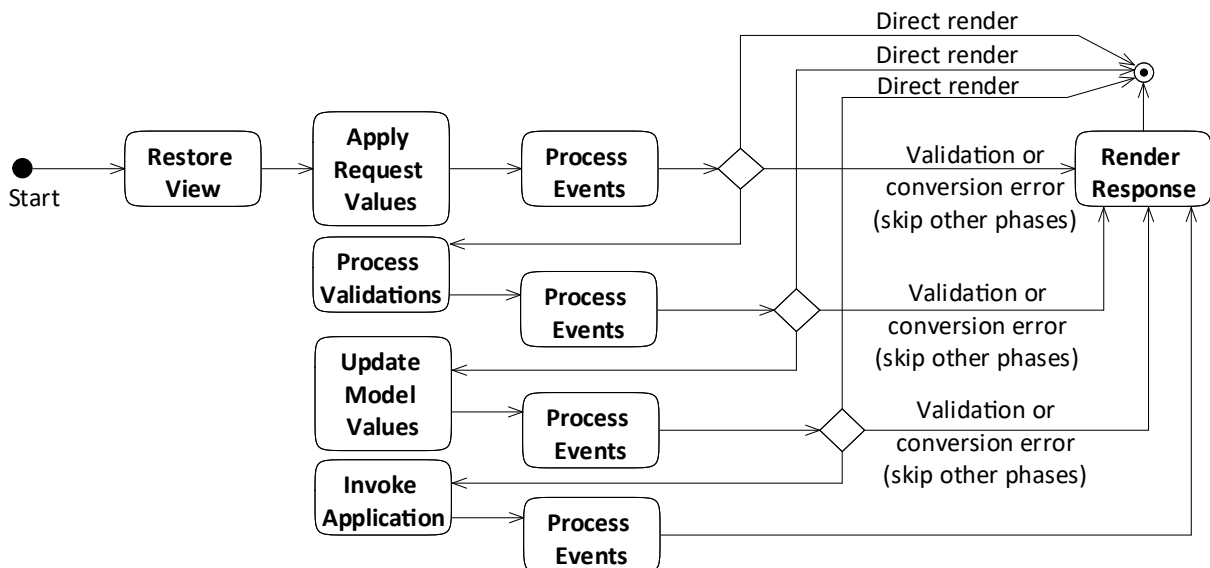


Figure 2.10: JSF request processing lifecycle

2.3.2 Size

This point is closely related to complexity, readability, and also plays part in the architectural issues. FelSight has been in development for almost a decade now. As the codebase grows, it becomes more and more difficult to develop.

To get the idea of how large FelSight really is, the following charts show actual numbers. The bubble chart on the left (2.11) shows the numbers of files by file extensions. The bar chart on the right (2.12) shows the numbers of lines⁶ categorized by languages.

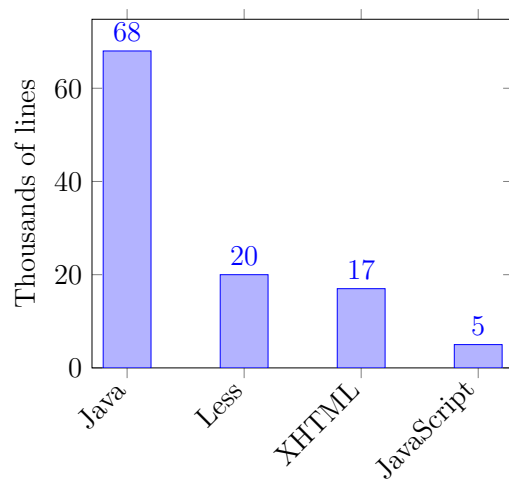
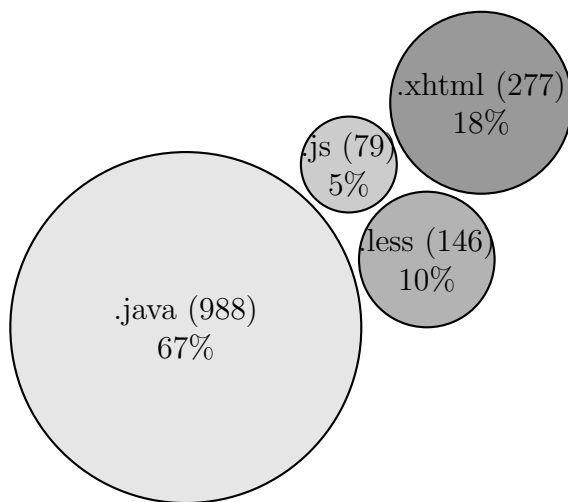


Figure 2.11: Numbers of files by extension

Figure 2.12: Number of lines by languages

In total, the project comprises 1,490 files, which together account for 110 thousand lines of code, predominantly written in Java. The fact that XHTML files outnumber LESS files while containing fewer lines of code suggests that there is greater modularity in the JSF components compared to the styling files.

Using SourceMonitor⁷ an overview of the Java backend has been compiled. Some interesting points are:

- 16% of the code consists of comments
- There are 1002 classes
- On average, there are 7.36 calls per method
- There is a total of 55 thousand statements
- There are over 23 thousand method calls

⁶Counted by an open source cloc app, excluding comments and blank lines.

⁷<https://www.derpaul.net/SourceMonitor/>

The following is an overview of the challenges that were observed as the project size increased.

- **Slower development and deployment** – the build time got longer and the memory and CPU demands increased⁸.
- **Increased risk of failure** – issues in a single module frequently affect the rest of the application.
- **Difficulties in CI/CD** – the build, test, and deployment in CI/CD pipelines became more time-consuming and very demanding on the runners⁹.
- **Version control** – merging conflicts became more common, larger, and more difficult to resolve.
- **Reduced agility** – the ability to implement new features or adapt to changing demands quickly was hindered.

2.3.3 Performance

One notable problem with FelSight is its performance, with the main issue being the loading times. The figure 2.13 displays the loading times that have been measured on 23rd March 2024 at 4:00 AM on a clean install of Google Chrome browser with cleared cache.

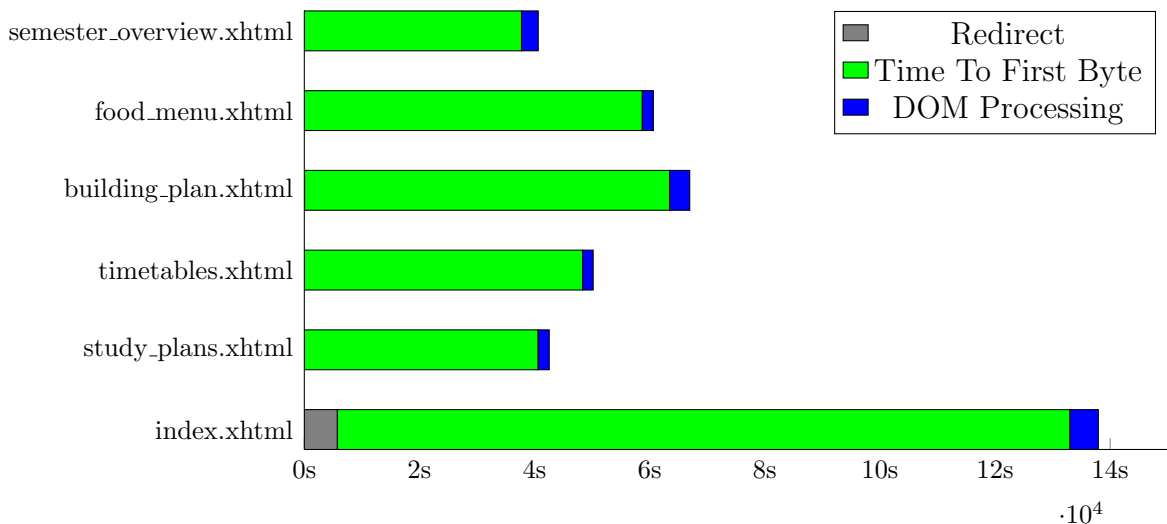


Figure 2.13: Measured times

⁸The build and execution typically require around 90 seconds, while refreshing it to observe modifications takes about 30 seconds, although this duration can vary significantly depending on the machine.

⁹Depending on the stages run, pipeline may take up to 10 minutes to finish.

The Time To First Byte (TTFB) measures the delay between sending a page request and receiving the first byte of the response [7]. DOM Processing refers to the steps a web browser takes to construct the UI and thus the additional time user has to wait before they can use the website. The time required for *index.xhtml* to load includes the login process, so the duration to redirect to FelSight from the login page was factored in.

Although JSF is not the sole reason for the performance issues, it contributes greatly. Performance of JSF can be influenced by subtle and not readily apparent factors that were not taken into consideration when developing FelSight [8]. Selecting the appropriate component library can have a significant influence as well [9]. In spite of this, JSF is limited by its inherent nature as it prioritizes full server-side rendering, going against the current trend in web development. It needs to maintain the state of the UI component tree between requests and also needs to regularly interact with the server, leading to a notable increase in network traffic.

2.3.4 Personal

Apart from issues with the technologies and architecture, there are also issues involving hiring new interns and retaining the current ones. Below is a list of problems related to this.

- **Steep learning curve** – Several articles mention JSF’s steep learning curve [10], [11]. It is a common experience in CZM that junior interns take a comparatively long time to become proficient in the technology.
 - **Career Development Concerns** – JSF’s market share in Web Application Frameworks is in 2024 below 0.1% [12], therefore, interns may feel that their skills are becoming less marketable as they spend their time with it. In a rapidly evolving tech landscape, there is a strong desire to work with cutting-edge technologies that will enhance a developer’s resume and future job prospects. Persisting with JSF might lead to concerns about a lack of advancement and decreased competitiveness in the job market.
 - **Lack of mentorship** – over time, CZM has been facing a shortage of mentors capable of introducing incoming interns to JSF. Moreover, the ecosystem around JSF, including third-party libraries, tools, and community support, is diminishing as its first release has been well over 20 years ago.
 - **Need for Full-Stack Expertise** – JSF, being a server-side framework that integrates closely with the Java ecosystem for web applications, necessitates developers to have full-stack expertise. Every intern needs to know Java, JavaScript, CSS and XHTML.
-

Chapter 3

FEL.HUB

Contents

3.1	General Idea	17
3.2	Technologies	19
3.3	Architecture	25
3.4	Motivations Behind The Project	26

FEL.HUB is an integration platform with the aim of integrating various faculty systems into a single unified system. This chapter presents an overview of FEL.HUB, focusing on its architecture, technologies, and concept.

3.1 General Idea

FEL.HUB was developed as a solution to the problem of a growing number of different faculty systems and the complexity within. Users, i.e. students and teachers, had to orient themselves in a broader and broader range of different systems that were developed independently of one another. FEL.HUB's aim is to integrate all of those systems under one roof and offer their functionalities in terms of modules called agendas. It also wants to offer a modern, accessible, scalable, and long-lasting system built on solid foundations. [13]

The distinction between the current system and the vision of a new one from the perspective of FEL.HUB is illustrated in the figure 3.1.

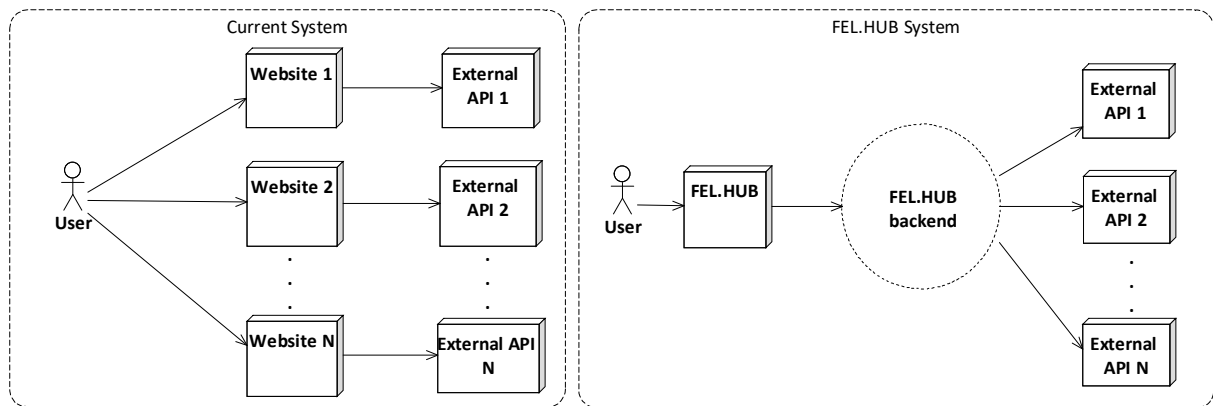


Figure 3.1: Comparison between the current and the FEL.HUB system

When a visitor accesses FEL.HUB, they can view all the agendas that they have authorization to access. There are also links to external systems below the main agendas.

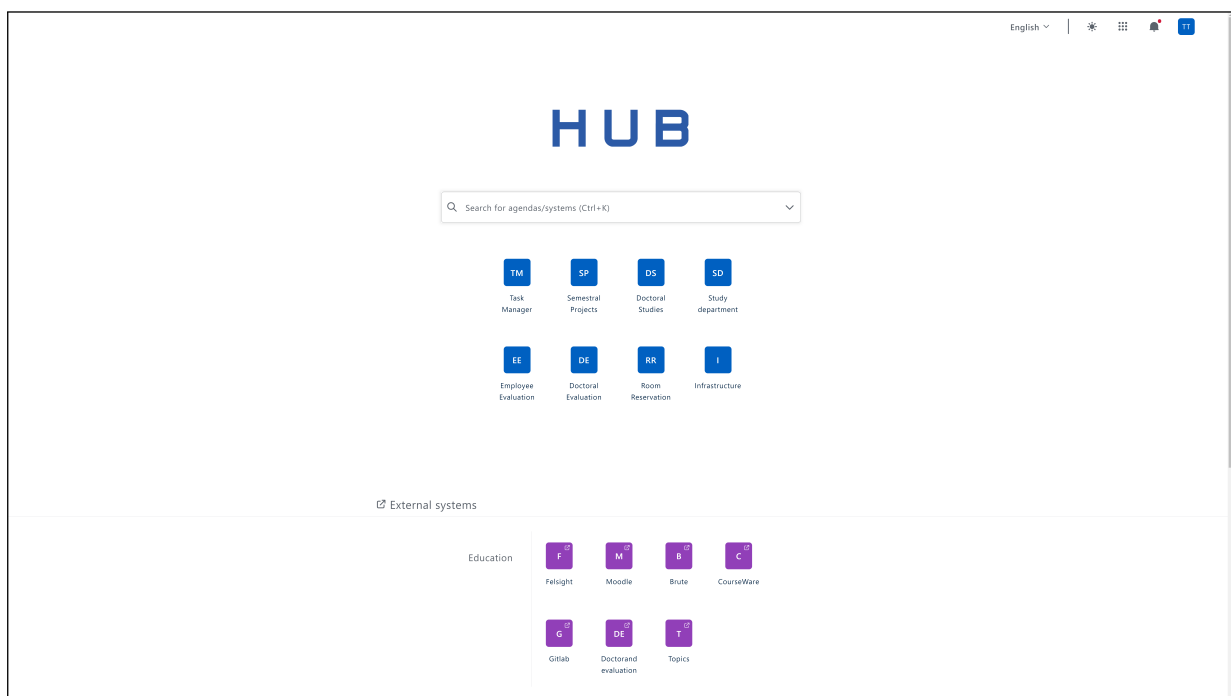


Figure 3.2: Main page of FEL.HUB

3.2 Technologies

This section introduces the technologies used in FEL.HUB and puts them in contrast with those of FelSight.

3.2.1 SPA and PWA

The frontend of FEL.HUB implements a so-called Single Page Application (SPA). SPAs are JavaScript-driven web applications requiring only a single page load. When the user enters the site, all essential files are loaded, and JavaScript takes control of the navigation from that point forward [14]. Thanks to the lack of abrupt page reloads, the UI feels smoother and resembles a native application¹. FEL.HUB leverages that by meeting the necessary standards of PWAs², enabling the website to be downloaded and used as a native application on a phone.

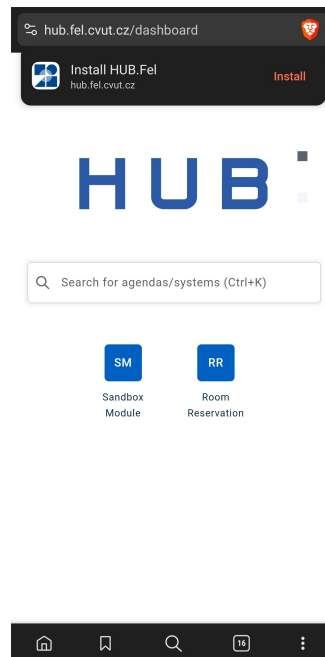


Figure 3.3: Browser prompting to download FEL.HUB

Although FelSight includes support for PWAs as well, it is an MPA (Multi Page Application) and consequently lacks the native-like experience that HUB provides. FelSight does not meet the browser requirements to provide it as a PWA out-of-the-box; instead, it is available for download on Google Play and the App Store.

¹Native applications are built for use on a particular platform or device [15].

²PWAs (Progressive Web Applications) are browser based web applications. They are typically run in a separate browser window without the address bar and are therefore visually and functionally near-indistinguishable from native applications [16]

3.2.2 React

The frontend is developed using React. It is a JavaScript library that, similar to JSF, is designed to build the user interface through reusable components [17]. Let us revisit a specific example to highlight the distinctions between the two technologies. The listing 3.1 shows a code for the week parity switch component that was used to introduce JSF (2.1).

```
1 export type WeekParitySwitchType = WeekParity.EVEN | WeekParity.ODD;
2
3 type Props = {
4   onParityChange: (newParity: WeekParitySwitchType) => void;
5   currentParity: WeekParitySwitchType;
6 };
7
8 export const WeekParitySwitch = (props: Props) => (
9   <div>
10    <StyledParityButton
11      parityType={WeekParity.EVEN}
12      isSelected={props.currentParity === WeekParity.EVEN}
13      onClick={() => props.onParityChange(WeekParity.EVEN)}>
14      {t'Even week'}
15    </StyledParityButton>
16    <StyledParityButton
17      parityType={WeekParity.ODD}
18      isSelected={props.currentParity === WeekParity.ODD}
19      onClick={() => props.onParityChange(WeekParity.ODD)}>
20      {t'Odd week'}
21    </StyledParityButton>
22  </div>
23 );
```

Listing 3.1: React component example

As can be seen by the type definitions above the component declaration, the project uses React in conjunction with TypeScript³. The component accepts the value of current week parity and a function to be run upon clicking a button that changes it. These arguments are used in the returned JSX⁴. The `StyledParityButton` comes from the use of a `Styled Components` library⁵ which allows the use of the CSS-in-JS styling technique. The definition of button styles can be seen in the listing 3.2.

³TypeScript is an open-source, high-level programming language that enhances JavaScript by introducing static typing through optional type annotations [18].

⁴JSX is a JavaScript extension that enables the creation of HTML elements using the well-known XML syntax [19].

⁵<https://styled-components.com/>


```
1 const StyledParityButton = styled.button<{
2   parityType: WeekParitySwitchType;
3   isSelected: boolean;
4 }>`
5   background: ${({ isSelected }) => (isSelected ? '#a6a2b2' : '#f2f5fc')};
6   color: ${({ isSelected }) => (isSelected ? '#fafafa' : 'unset')};
7   border: none;
8   padding: 0.35em 0.75em;
9   font-size: 1.8rem;
10  cursor: pointer;
11
12  &:hover {
13    background: '#d9dce1';
14  }
15
16  --border-radius: 0.25em;
17  ${({ parityType }) =>
18    parityType === 'ODD'
19    ? css`border-bottom-right-radius: var(--border-radius);
20      border-top-right-radius: var(--border-radius);`
21    : css`border-bottom-left-radius: var(--border-radius);
22      border-top-left-radius: var(--border-radius);`
23  }
24 `;
```

Listing 3.2: Styled component example

Styled Components offer a way to modularize the CSS and bring it closer to JavaScript, thus increasing cohesion. In addition, it also provides a robust method for directly manipulating styles using JavaScript.

In the first line, it is specified that the styled component is a button HTML element. The following two lines define the parameters which can then be worked with using the notation in the 3rd line. The code inside the dollar with curly braces is JavaScript. It is usually used for branching and working with the component's arguments.

In line 12, a standard CSS selector is used to specify a hover color. The rest of the code features an example of using a standard CSS variable within the library to define a border radius.

The following list summarizes the key advantages of the frontend technologies of FEL.HUB.

- **Resource Efficiency** – The client-side houses the entirety of the UI logic, while the backend is tasked mainly with data delivery and executing operations unsuitable for JavaScript. This approach significantly reduces bandwidth and data transfer requirements.
- **Technology Choice** – The project employs React, currently the leading framework in popularity [20]. React’s prominence in the job market makes it easier to recruit both experienced developers and enthusiastic interns eager to acquire this sought-after skill.
- **Community Support** – React benefits from unparalleled community support, distinguishing itself from JSF with a vast number of developers actively enhancing its ecosystem [21].
- **Rich Ecosystem and Libraries** – With access to an extensive range of third-party libraries, tools, and frameworks, React enables seamless expansion and integration of additional functionalities into applications [22].
- **Optimized Performance** – Through the use of virtual DOM and sophisticated diff algorithms, React significantly optimizes performance by minimizing unnecessary DOM manipulations [23].
- **Enhanced Developer Experience** – FEL.HUB emphasizes high cohesion⁶, addressing a notable challenge within the FelSight codebase. By integrating HTML, CSS, JavaScript, and core logic within the same component, it simplifies navigation and maintenance of the codebase as it expands, in contrast to FelSight’s segregated approach.

⁶Cohesion refers to how closely related the components within a module are, regarding their functionality [24].

3.2.3 GraphQL

The frontend communicates with the backend using GraphQL. It is a data query and manipulation language that also ships with its runtime engine. It enables for a declarative way of fetching data in which a client specifies exactly what data they need. The server exposes a single endpoint to which the client sends an HTTP POST request with the list of the required data. The runtime engine decodes the message and uses the available services to obtain the data, and returns it in JSON format. [25]

The diagram 3.4 captures this process.

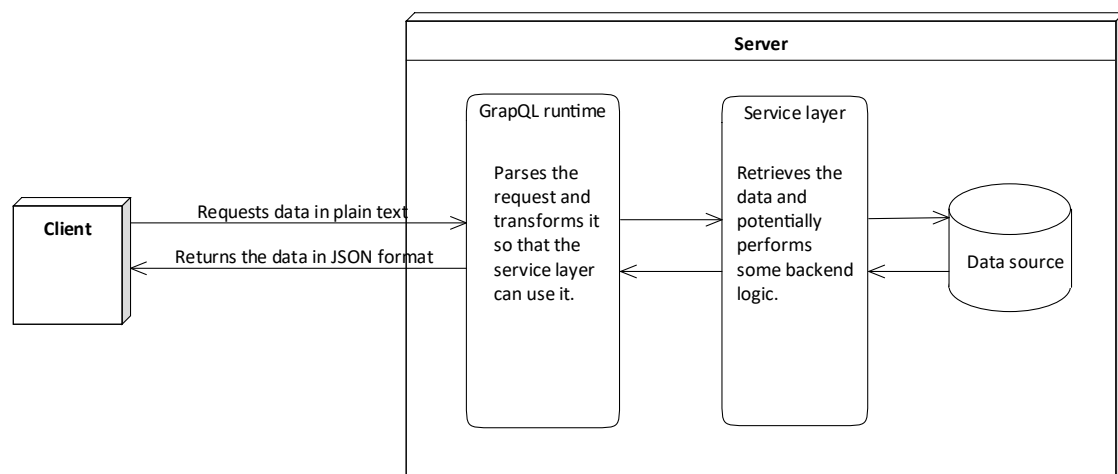


Figure 3.4: GraphQL communication path

Instead of individual endpoints, GraphQL APIs use a simple schema language to express the possible queries and types which can be fetched from them. The listing 3.3 shows what the schema may look like.

```
1 type Query {
2   getCourseInfo(code: String!): Course
3 }
4 type Course {
5   name: String!
6   credits: Int!
7   students: [Person!]!
8 }
9 type Person {
10  username: String!
11  email: String
12 }
```

Listing 3.3: GraphQL schema example

There are two types defined – `Course` and `Person`. The `Query` types define a query to retrieve information about a course given its code. If the type ends with an exclamation mark, it means that it is not nullable.

The listing 3.4 shows the plain text that would be sent in the POST request to specifically retrieve the name of the course and usernames of its students.

```

1 query {
2   getCourseInfo(code: "BOB01LAG") {
3     name
4     students {
5       username
6     }
7   }
8 }

```

Listing 3.4: GraphQL query example

To fully leverage the potential of GraphQL with its out-of-the-box schema definitions, the Apollo Federation is used. It is an architecture that enables combining schemas from multiple GraphQL APIs into one through which the frontend can interact using a single request [26]. This insulates the frontend from needing to know about the individual backend services.

Instead of directly interacting with individual services, the frontend communicates exclusively with a GraphQL gateway. This service dynamically orchestrates a unified super-schema by aggregating GraphQL schemas from the individual services. Incoming requests are routed to the corresponding services. This process is illustrated in the diagram 3.5.

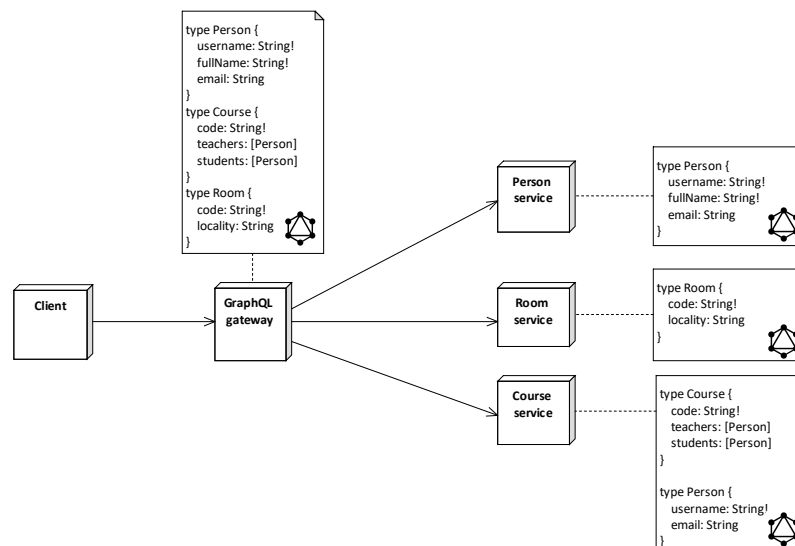


Figure 3.5: GraphQL federation example

3.2.4 Java, SpringBoot, DGS

When it comes to backend, most⁷ of the services at the moment use Java with Spring-Boot. It is a development platform within the Java ecosystem designed to streamline the configuration and deployment of web applications by abstracting and automating boilerplate setup processes. Fundamentally, Spring provides a container that is responsible for the creation and management of application components. These components, also known as beans, are interconnected within the Spring application context to form a comprehensive application [27].

The services use Netflix's DGS library to manage the GraphQL workflow by providing annotations for data fetchers, error handling, and testing [28].

3.3 Architecture

From a broad perspective, the architecture of FEL.HUB adheres to the straightforward diagram shown in the schema 3.1.

Microservices within the FEL.HUB ecosystem follow a standard three-layer architecture. It is a software architecture pattern in which the user interface (presentation), business logic, and data storage and access are developed as autonomous components [29]. The diagram 3.6 shows an architecture that a FEL.HUB microservice generally follows.

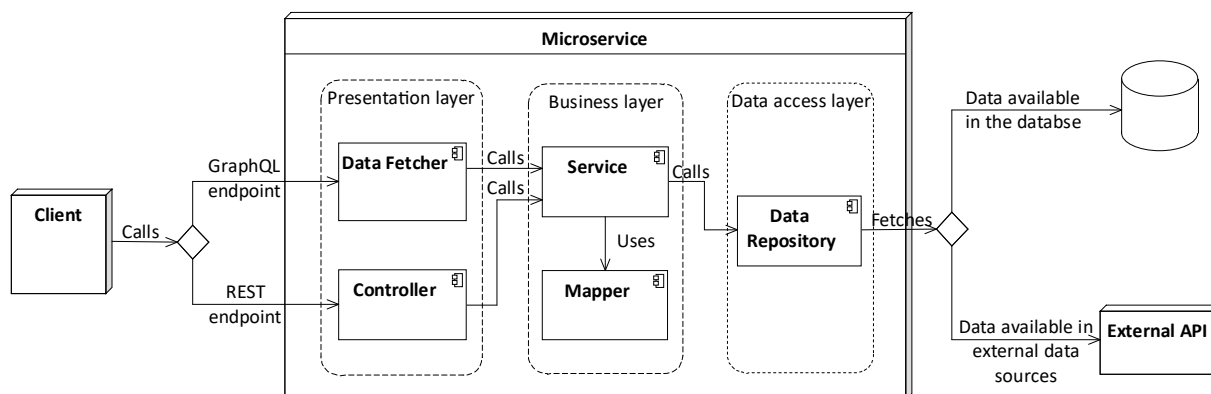


Figure 3.6: Architecture of a FEL.HUB microservice

While the frontend and backend exclusively communicate via GraphQL, interservice communication continues to utilize REST protocols. When a request targets the GraphQL endpoint, it is handled by a component known as *data fetcher* within the DGS framework. Conversely, REST requests are managed by controllers maintained by Spring. Both component types delegate to specialized services within the business layer.

⁷There are several services whose developers chose to use Kotlin instead of Java.

These services are tasked with executing business logic. In addition to services, there could be other elements like mappers or certain utility classes.

The data access layer is made up of repositories responsible for retrieving data, either from a database or an external API, if available. These repositories are tasked with formatting requests or queries, incorporating elements such as pagination, limits, and offsets to ensure efficient data handling and retrieval.

3.4 Motivations Behind The Project

Let us summarize the motivations behind the project given what has been said in the chapters 2 and 3.

- **Increase scalability** – FelSight has scalability issues that would require extensive remodeling of the whole architecture. FEL.HUB is built with scalability in mind and is thus suitable to cover the demands for the features that FelSight provides.
 - **Improve performance** – FelSight is becoming increasingly slower due to the technological and architectural decisions made at its inception. FEL.HUB leverages the most up-to-date technologies that are suitable for building highly performant applications.
 - **Reduce development cost** – Given the size of the project and the implications of its monolithic architecture, adding new features, fixing bugs, and maintaining the application overall has become more time consuming. FEL.HUB effectively mitigates these challenges by modularizing its functionality and logic, maintaining a nearly constant development cost per feature despite the project's growth.
 - **Ease developer hiring** – FelSight's technology stack has a steep learning curve resulting in substantial amount of time of the incoming developers being delegated to mastering it. Moreover, motivation for learning the technology is lacking due to its obsolescence. FEL.HUB employs a technology stack that is both profitable to learn and highly sought after, ensuring that interns' considerable time investment is not wasted.
 - **Improve functionality** – Lastly, the functionality of FelSight has been revised and improved during the creation of the UI styles [1]. Features that were not used will not be reimplemented, while those that were will be refined and prioritized.
-

Chapter 4

Analysis

Contents

4.1	Use Cases	28
4.2	Backend	31

This chapter attempts to provide a comprehensive analysis of the project, outlining the essential elements that need to be implemented.

Initially, a series of use cases are delineated and categorized based on their association with the individual pages. Due to the large number of these use cases, they were further divided into six distinct categories. For each of these categories, a list of associated use cases is provided together with a detailed description.

The backend is presented in great detail with an overall description of what needs to be done. The functionalities of each service are elaborated thereupon, detailing the specific work required. An outline¹ of the GraphQL schema is presented in each of the services.

The frontend component is not covered in this analysis, as it primarily focuses on creating a user interface that adheres closely to the specified designs. The design from Lucie Baronová's thesis [1] did not encompass all the use cases discussed in this chapter. Consequently, ongoing collaboration with the UI/UX team of CZM has been initiated to ensure that the remaining functionalities are accurately captured and implemented.

¹Presenting the whole schema including the specific types and enums would be impractical. Moreover, the diagrams of the databases that will be presented shortly closely resemble the GQL types.

4.1 Use Cases

The diagram 4.1 presents use cases that were created with the goal of capturing the functionalities of FelSight in the timetable and the planner page together with the new ideas from the UI designs of Lucie Baronová.

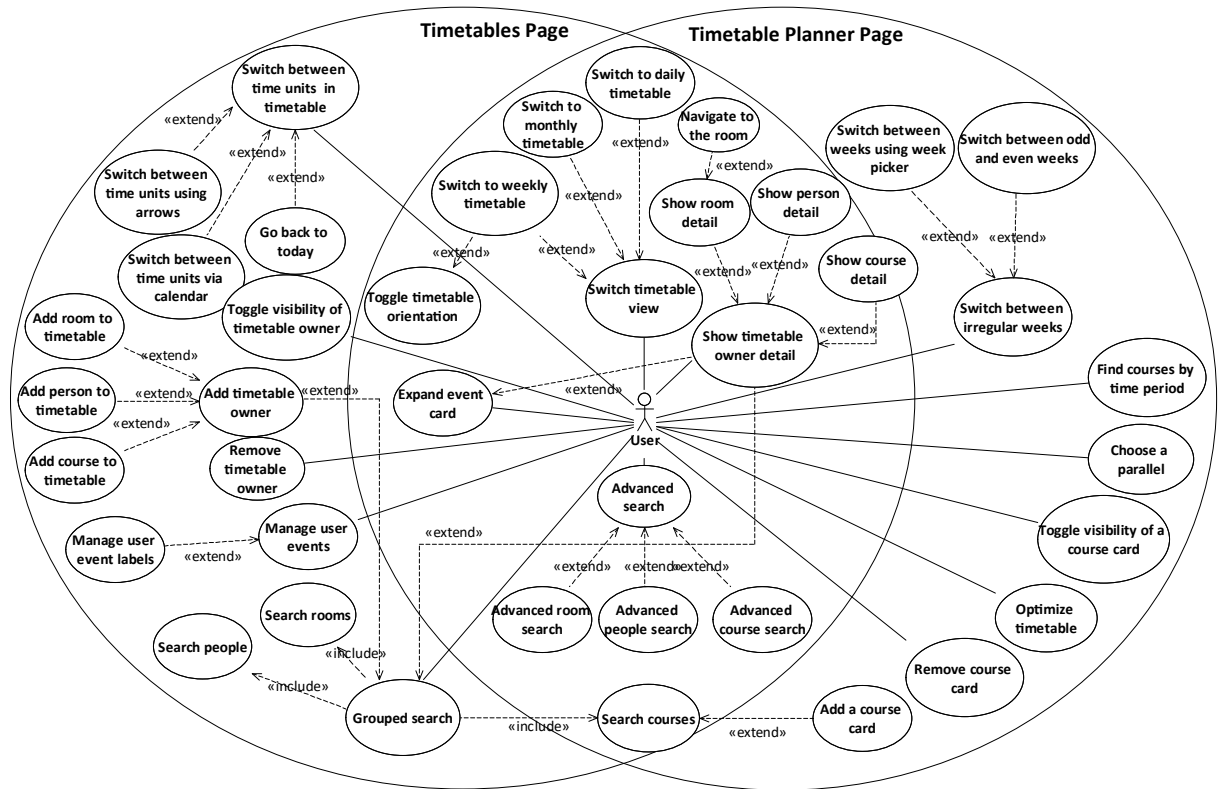


Figure 4.1: Use case diagram

The left circle represents the boundary of the timetable page and the right circle contains the use cases that belong to the timetable planner page. The use cases at the intersection of the two boundaries are related to both pages. The core component of both pages is a timetable that contains items called events.

Later, in the implementation stage, these use cases are linked to individual components that fulfill their scope. Because there are quite a lot of use cases, they were divided into six categories and further specified thereafter.

4.1.1 Timetable Views

Use cases: **Switch timetable view**, **Switch to weekly/monthly/daily timetable**, **Toggle timetable orientation**

There are several views of the timetable that are based on time range or orientation. On desktop, users have the option to view the timetable horizontally, with time displayed along the x-axis, or vertically, with time along the y-axis. They can also switch between a weekly and a monthly view of the timetable. The monthly view is similar to a google calendar and helps users to plan further ahead. There is also a daily view which is a default view on phones.

4.1.2 Timetable controls

Use cases: **Switch between time units (calendar / arrows)**, **Add/remove timetable owner (person / room / course)**, **Add/remove course card**, **Choose a parallel**

Timetable presents events within a specified time range, be it a day, week or a month. In order to switch between these time ranges, there are arrows that move by a corresponding time unit forward or backwards, as well as a calendar that allows the user to select a specific date.

Regarding the choice and visibility of the events in the timetable, there is a side panel with cards and a search bar through which they can be searched. In the timetable page, these cards are referred to as *owner cards*. They represent a timetable of a room, person or a course. Timetable planner's cards, on the other hand, are simply called course cards and they allow the user to select parallels.

4.1.3 Searching

Use cases: **Grouped search (people, rooms, courses)**, **Advanced search (people, rooms, courses)**

As mentioned in the previous section, the cards representing events in the timetable can be searched in a dedicated search bar. In the timetable page, the search is grouped. In the timetable planner only courses are searched.

Both search bars also contain an option of an “advanced search”. Clicking on the option reveals a modal with a more thorough search allowing advanced filtering, sorting, etc.

4.1.4 User Events

Use cases: **Manage user events**

In the timetable page, users can manage their *user events*. This feature is aimed at creating a more personalized experience for the user as they can merge their own affairs with their school schedule. The user events will have the same parameters as in FelSight – there are *timed* and *whole-day* events, as well as *one-time* and *repeated* ones. Additionally, users can organize these events under customizable labels. These labels function similarly to timetable owners, meaning they can be collectively viewed, hidden, or removed.

4.1.5 Timetable Irregularity

Use cases: **Switch between irregular weeks (using week picker / odd, even weeks)**

In timetable planner, a situation can occur in which some courses have irregular parallels. For example, physics labs may be taught every other week, or there might be a safety training microcourse that consists only of a few lectures in the first two weeks of the semester. Users need to be properly informed about these irregularities in their future timetable.

There are two components to tackle this. The first is a week-parity switch (2.8), which is used to alternate between even and odd weeks. In case of higher-order irregularities, there is a week picker (2.6) in which users can choose a specific week to look at.

4.1.6 Details

Use cases: **Show timetable owner detail, Expand event card**

Upon clicking on a search result, a detail modal is shown. It is also possible to get into the owner detail by clicking on a teacher's name or the room code in the expanded event card.

In the case of rooms, the detail contains a navigation button that is linked to the building plans page (2.7) with a prefilled destination. The floor projection is also displayed with a dynamically generated sentence describing the room.

Person detail displays a photo of the user (if available and public) and additional information such as the user's department and a list of emails. What is unique here are the translated roles – UserMap roles are fetched in a unified code string like *B-13000-SUMA-STUDENT-MAGISTR* which is difficult to present in a user-friendly way. FelSight has made an attempt to translate about fifty most used and important roles such as *Student*, *full-time master's degree* or *Vice dean for foreign relations*. These translations will be copied over to FEL.HUB.

Course detail just shows more information about a given course such as the required literature or a full description.

4.2 Backend

The backend (initially crafted to modularize FelSight's architecture) from Ladislav Svoboda consists of four microservices as depicted in the diagram 4.2.

Timetable service provides fresh data for timetable viewing and planning using Sirius API. It also enables creating, updating, and fetching data for user events. Finally, it serves as storage for saved timetable planner selections.

Course semester service provides highly filterable data that can be used for searching courses and parallels. It also stores and exposes data on semesters and departments, as well as detailed information about courses. Its database is the largest and is periodically updated via scheduled jobs using KOS service.

Room service also periodically updates its database using KOS service, however, the data are only concerned with rooms.

The KOS service was created to translate the XML responses of KOS API into REST.

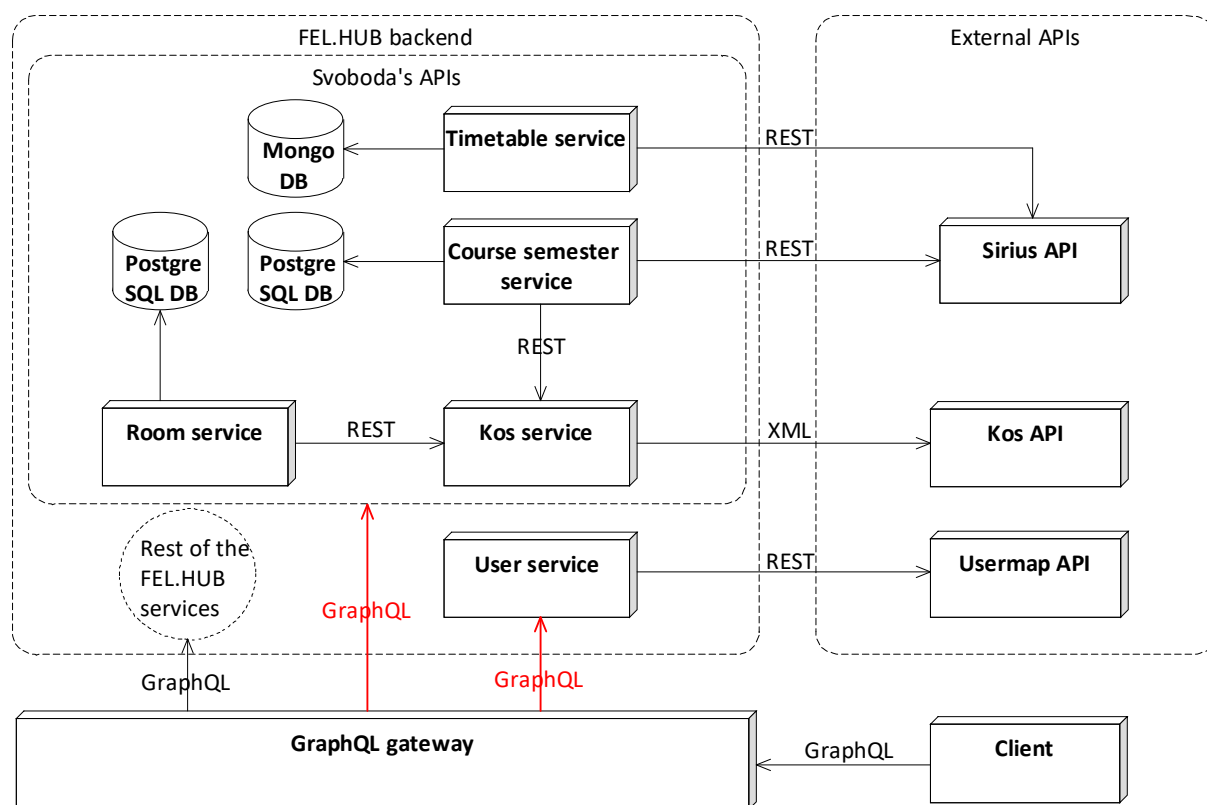


Figure 4.2: Diagram of the FEL.HUB backend

The Svoboda's services as well as user service only contain REST endpoints. As indicated by the red arrows, they need to expose a GraphQL schema to GraphQL gateway in order for the backend to be complete.

4.2.1 Timetable Service

The timetable service ensures that users have access to the latest information about events and sessions in both the timetable viewer and planner. This is essential because both events and sessions are subject to frequent updates, and users need to stay informed about these changes.

Moreover, together with the current events for a given time range, it needs to package possible user-defined events with the response. Since the user events also need to be linked with labels, there need to be options to create, delete and retrieve them as well.

In order to save the courses and parallels user has selected in timetable planner, there needs to be a way of saving the current selection as well as retrieving it if available.

In GraphQL, mutations are used to modify data on the server, such as creating, updating, or deleting data. The listing 4.1 shows the queries and data mutations that need to be implemented in the Timetable service.

```
1 type Query {
2   eventMany(from: Date, to: Date,
3             usernames: [String],
4             courseCodes: [String],
5             roomCodes: [String]): [TimetableSlot]
6   timetableForCourses(semesterCode: String,
7                       courseCodes: [String]): [PlannerTimetableCourse]
8
9   getUsersPlannerSelection: PlannerSelection!
10
11  userEventOneById(id: String!): UserEvent
12  userEventMany: [UserEvent]
13
14  userEventLabelMany: [UserEventLabel]
15 }
16
17 type Mutation {
18   savePlannerSelection(selection: PlannerInput): PlannerSelection!
19
20   saveUserEvent(userEventInput: UserEventInput): UserEvent
21   deleteUserEvent(id: String): String
22
23   saveUserEventLabel(userEventLabel: UserEventLabelInput): String
24   deleteUserEventLabel(labelName: String): String
25 }
```

Listing 4.1: Timetable service queries

The `eventMany` fetches the events for the given timetable owners (rooms, people, courses). In case of users, it also appends any user events of that user.

The `timetableForCourses` query is designed to gather the essential data needed for constructing the timetable planner component. This includes information on the number of credits, course parallels, the times at which these parallels are taught, and any irregularities, among other details.

There are also queries for managing user events and user-event labels as well as saving and retrieving the user's planner selection.

Regarding the database, the entities for planner selection saving are already present; however, there are no user events. The diagram 4.3 outlines a design of the completed MongoDB data model.

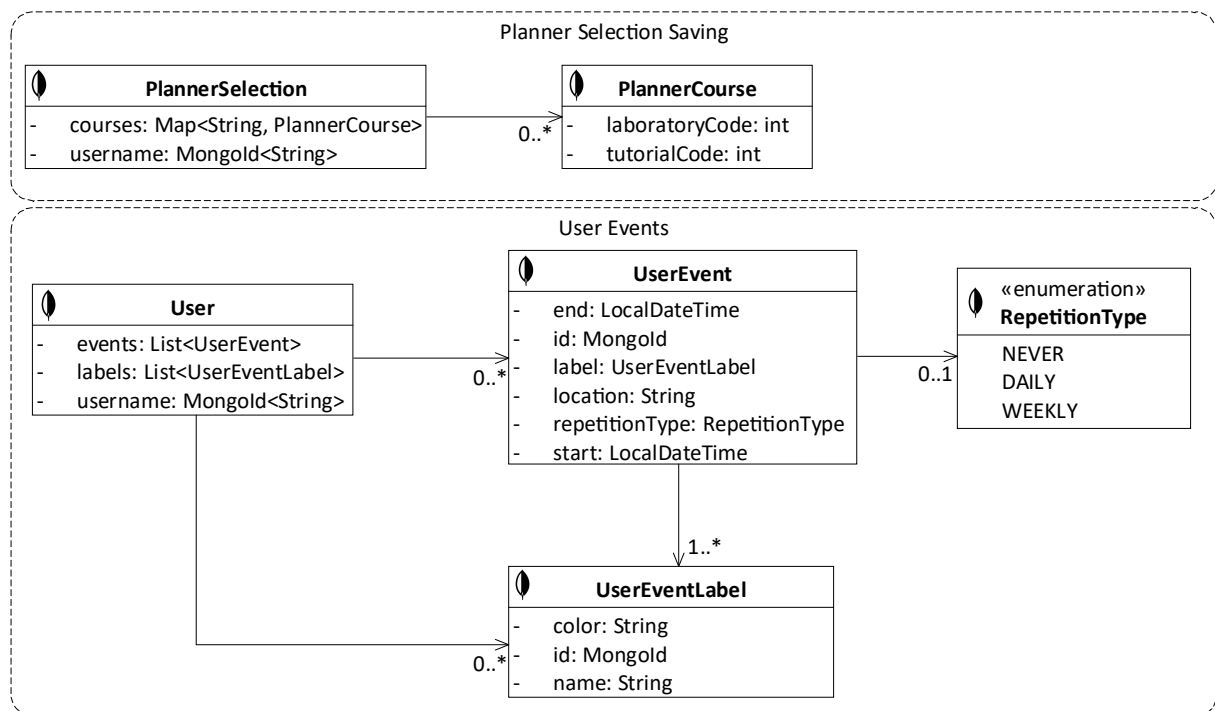


Figure 4.3: Data model for timetable service database

The `User` entity was created in order to make the queries for user events more effective as well as to make the database itself clearer. When verifying the feasibility of creating a user event, it is essential to have access to all of the user's labels and events. This makes it easy to check whether an event with the same name already exists, or if there is a label sharing the same color. The existence of `User` alleviates the need to join or having to perform additional queries.

4.2.2 Course Semester Service

In order to be able to offer highly specific and advanced data filtering, Course semester service stores data about Courses in an SQL database and periodically updates it through KOS service. Thanks to the fact that the data is inside an SQL database, it can be efficiently filtered through using Spring's query-building APIs.

Given the relatively invariant nature of semester data, it is also stored within the database. Updates are required only on a bimonthly basis, facilitating rapid query execution without the necessity of invoking the KOS service.

The entire data model is depicted in the diagram 4.4. Several simple modifications have been implemented in the model relative to its original configuration.

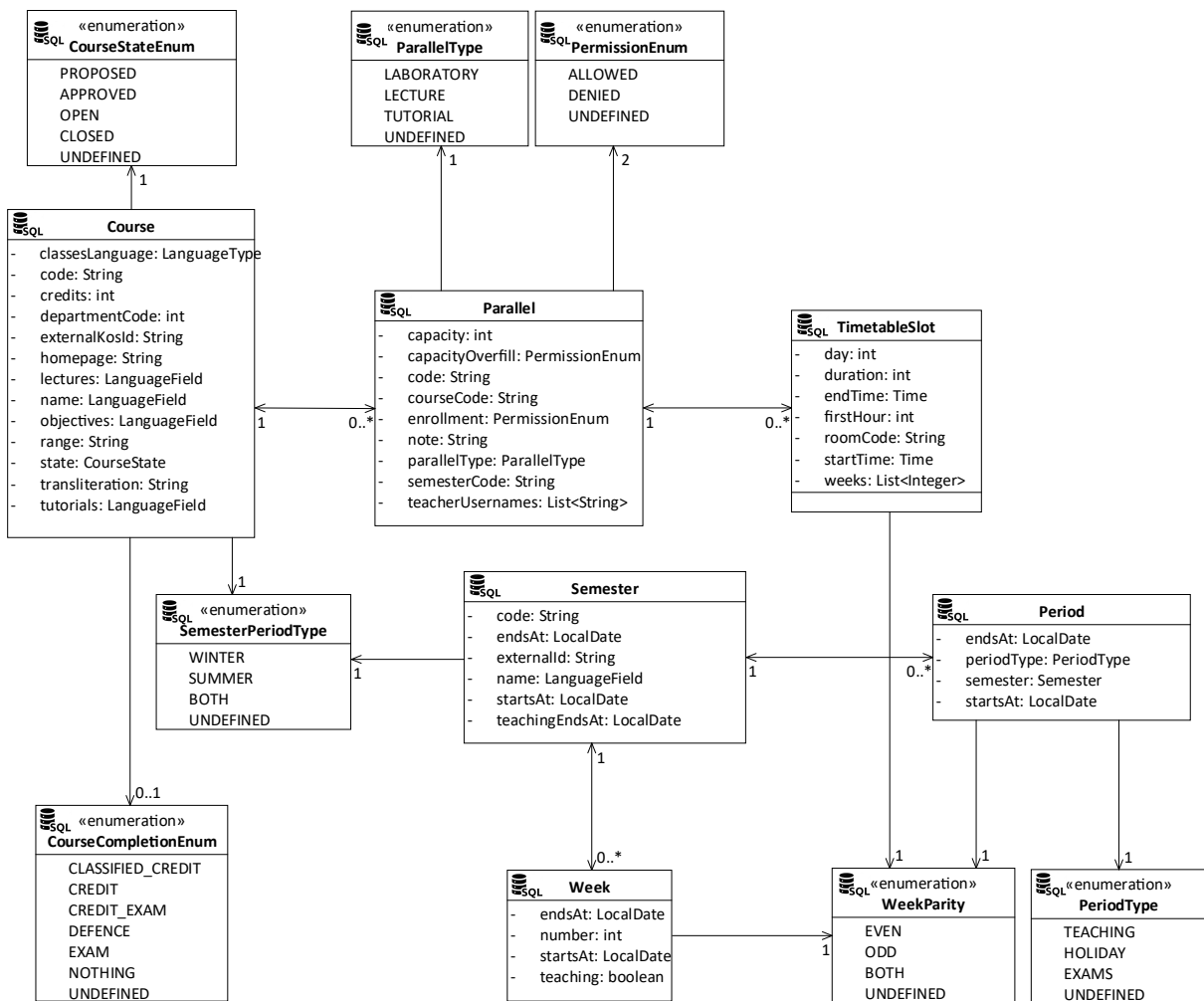


Figure 4.4: Data model for course semester service database

The listing 4.2 shows the GraphQL queries that need to be implemented.

```
1 type Query {  
2   courseOne(code: String, language: LanguageCode): Course  
3  
4   semesterOneByDate(date: Date): Semester  
5  
6   courseMany(language: LanguageCode,  
7             pagination: PagedCourseRequestInput,  
8             search: String): [Course]  
9  
10  advancedCourseMany(language: LanguageCode,  
11                    pagination: PagedCourseRequestInput,  
12                    filter: AdvancedCourseInput,  
13                    search: String): AdvancedCourseManyResult  
14 }
```

Listing 4.2: Course semester service queries

The list below details the utilization of the queries within the system.

- `courseOne` will be used to fetch data needed to display the course detail. It will simply look up the course in the database and return it.
- `semesterOne` will return information needed to display the correct semester code in the timetable given the currently selected date.
- `courseMany` will retrieve data from the courses database table. It will be called in advanced grouped search as shown in 5.19.
- `advancedCourseMany` will provide a highly filterable API for advanced timetable searching.

While the first three queries primarily involve straightforward lookups into the database, the `advancedCourseMany` requires a sophisticated use of query-building APIs. This is necessary to construct highly complex queries that can accommodate the detailed filtering options.

4.2.3 Room Service

Room service also stores the data in the database and frequently updates them through KOS service.

Similarly to `courseOne` and `courseMany`, the Room service requires the implementation of `roomOne` and `roomMany` queries. The `roomOne` query is designed to retrieve the necessary data to display detailed information about a specific room, while `roomMany` facilitates the retrieval of room data for the room tab within the advanced grouped search interface.

```
1 type Query {
2   roomMany(pagination: PagedRoomRequestInput, search: String): [Room]
3   roomOne(code: String): Room
4 }
```

Listing 4.3: Room service queries

Although the Room service currently encompasses only a small fraction of the backend logic, it is anticipated to expand in the future to include the logic for room reservations.

4.2.4 User Service

Similarly to the Room service, the User service also requires two key implementations: one query to fetch detailed information for individual user profiles, and another query for populating the *people* tab in the advanced grouped search. However, a distinct aspect of the User service is that it does not store user data locally in a database. Instead, it retrieves this information dynamically from the UserMap API.

```
1 type Query {
2   personMany(search: String,
3             searchBy: PersonTableColumn,
4             pagination: PagedPersonRequestInput!): [UmPerson!]!
5   personOne(username: String): UmPerson
6 }
```

Listing 4.4: User service queries

The User service falls outside the scope of this thesis and beyond the control of its author. The changes made to it will be tested locally and subsequently submitted for review to its administrators.

Chapter 5

Frontend Implementation

Contents

5.1	Timetable Views	40
5.2	Timetable Controls	45
5.3	Search	49
5.4	User Events	52
5.5	Timetable Irregularity	53
5.6	Details	53
5.7	Comparison With Designs	57

This chapter details the frontend implementation through annotated screenshots of the components, organized according to the categories used in the discussion of the use cases (4.1).

Figure 5.1 provides a comprehensive overview of all React components included in the implementation, excluding common FEL.HUB components like buttons. The arrangement of the packaged components mirrors the directory structure within the project. Arrows indicating a connection from component A to component B signify that component A uses component B.

The components are delineated by dashed rectangles representing distinct boundaries; these boundaries correspond to the timetable page and the planner page. Components within the hatched region are shared between both pages.

The implementation consists of approximately 50 components and to maintain clarity and readability, efforts were made to limit their size to under 200 lines, excluding import statements. The core logic for each component resides in its primary file where it is defined. Styles are specified below the component definition using the styled-components library (see

3.2). Minor parts of a component's logic are incorporated inline, while more substantial sections are organized into separate functions and placed in a corresponding file within the same folder, from which they are then imported.

Styling constants that were often referenced, such as width of the timetable row, were aggregated into a configuration object and moved into a separate file from which they can be exported. This gave a better overview of which styles are applied where.

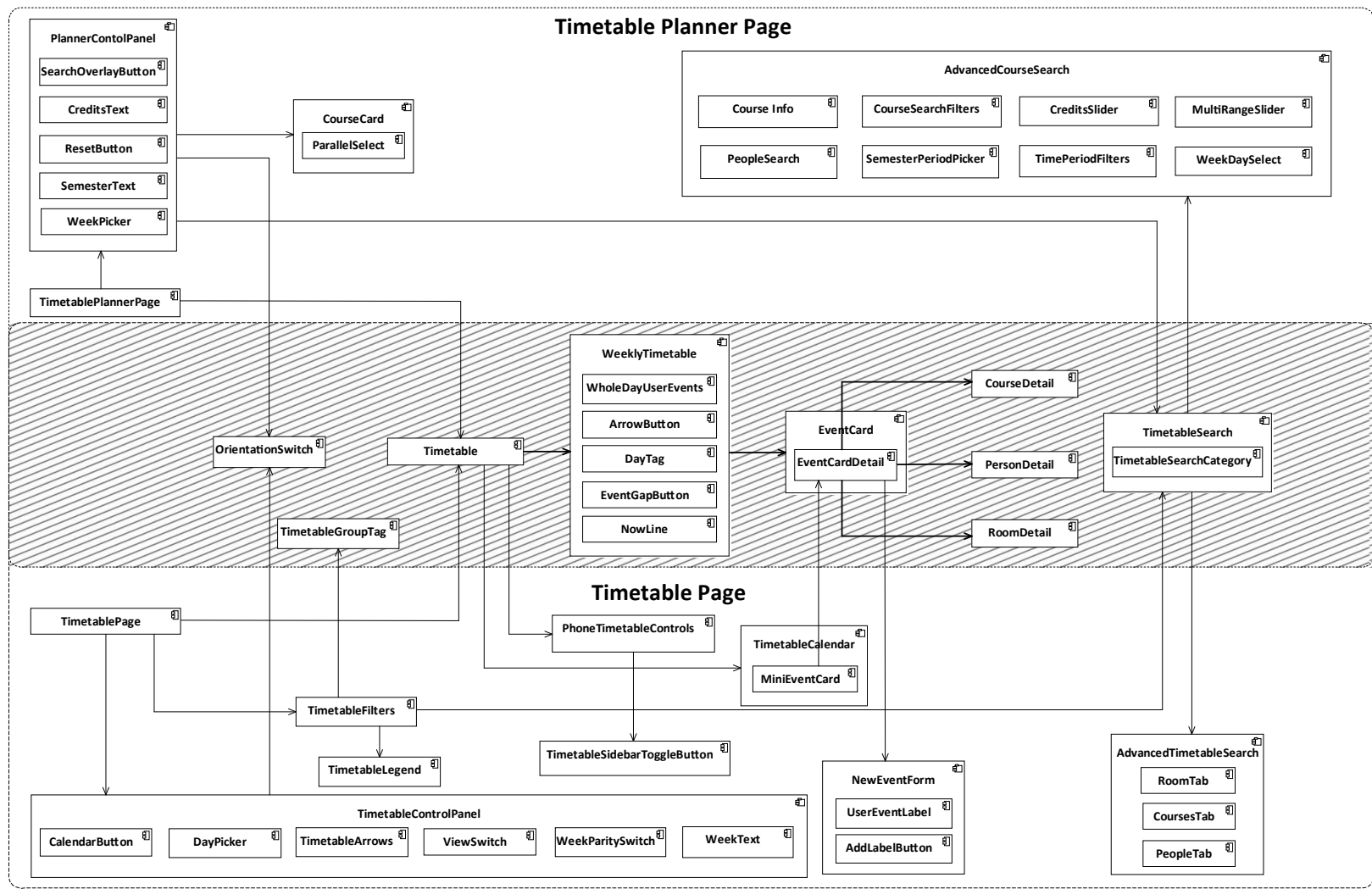


Figure 5.1: React components

5.1 Timetable Views

5.1.1 Weekly Timetable View

Figure 5.2 depicts the adaptation of the traditional vertical weekly timetable format, familiar to users from the FelSight application. Contrary to the final designs by Lucie Baronová, this specific layout was developed based on initial designs during the Software Project course. Given that students are accustomed to this horizontal view, retaining it in the project was deemed beneficial. This inclusion not only preserves continuity and familiarity for the users but also enhances user choice, offering them the flexibility to select between layout presentations according to their personal preferences.



Figure 5.2: Horizontal weekly timetable

Figure 5.3 presents a vertical view of the weekly timetable, which more closely resembles a conventional calendar layout. Despite the apparent complexities stemming from the two distinct orientations, both views are integrated within the same component. The variations between these orientations are facilitated through modifications in the styles and functions responsible for accurately calculating the event positions.

For the positioning of events, it is essential to employ units that are relative to the container's size. This approach ensures that when the dimensions of the container are adjusted, the events maintain their appropriate placement along the time axis. Specifically, in the horizontal view, the horizontal offset of events is calculated using percentage units, which scale dynamically with changes in width. Conversely, in the vertical view, the vertical offset is calculated using `vh` (viewport height)¹ units, ensuring that the events align correctly as the container's height varies.

¹The 'vh' unit in CSS represents 1% of the viewport's height [30].

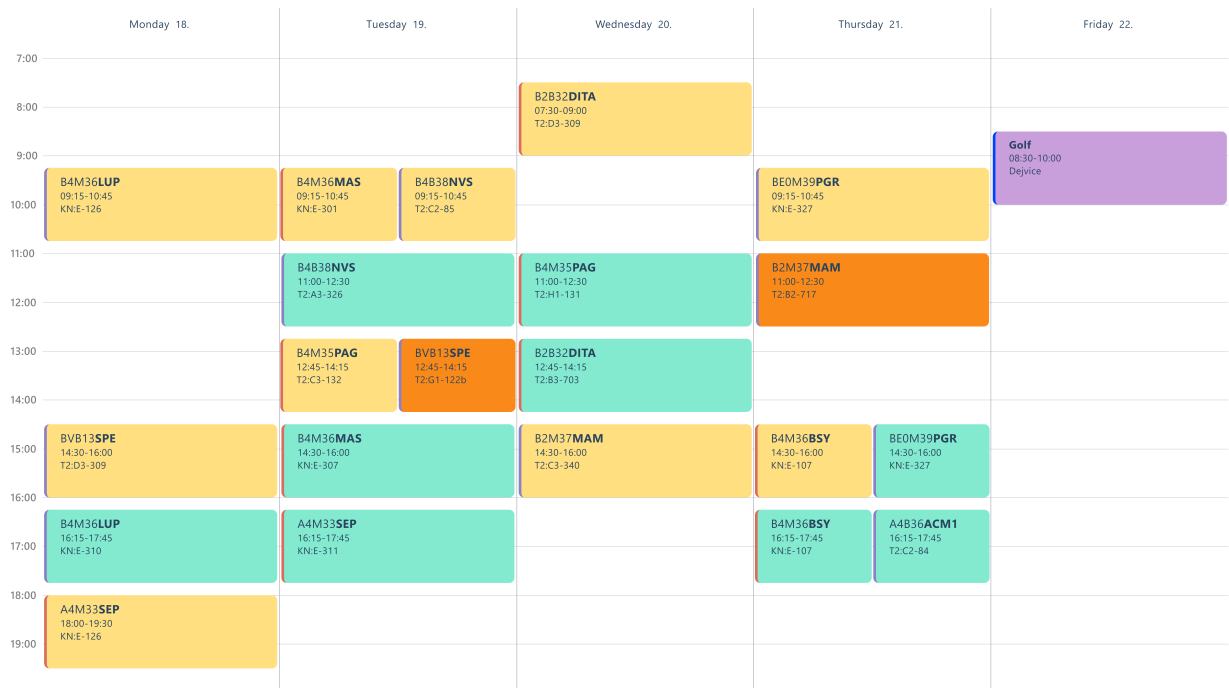


Figure 5.3: Vertical weekly timetable

5.1.2 Collision Management

To properly style overlapping events and ensure clear visualization, they must be grouped. This is achieved using a 3-dimensional array. The first dimension of this array represents the days of the week, while the second dimension contains arrays of elements that overlap, referred to as *collision slots*. If an event does not overlap with others, it is the sole member of its *collision slot*. For instance, in the figure 5.3, the array for Tuesday would be [[MAS, NVS], [NVS], [PAG, SPE], [MAS], [SEP]].

As illustrated in Figures 5.4 and 5.5, the management of collisions involves adjusting the width or height of an event's display proportionally. The determination of appropriate styles for handling these collisions is governed by two parameters: *collision cardinality* and *collision offset*. The *collision cardinality* denotes the count of event cards that overlap with the current one, including the card itself. In the examples provided, the cardinalities are 3, 1, and 2. *Collision offset*, on the other hand, defines the sequence in which the cards are arranged, which is dictated by the starting time of the events; for instance, the **Golf** event takes place earliest and is therefore positioned as first.

The event card component computes the offset using these parameters to ensure precise alignment with other cards, especially since these elements are absolutely positioned rather than relying on flexbox or similar tools. This calculation is intricate due to minimal spacing between events.

Efforts have been made to preserve readability up to the third level of collision. Since collisions involving more than three cards are exceedingly rare, adaptations are made to accommodate up to three simultaneous events. For instance, in the horizontal timetable, padding is minimized when three events collide. Conversely, in the vertical timetable, readability is maintained even with three overlapping events by abbreviating the course code to its essential form.

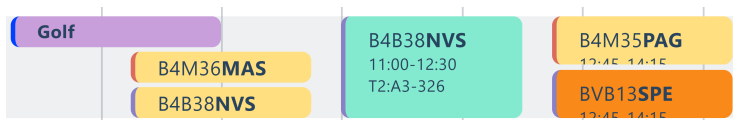


Figure 5.4: Collisions in horizontal timetable



Figure 5.5: Collisions in vertical timetable

5.1.3 Monthly Timetable View

Figure 5.6 features a monthly view of the timetable, which aligns with the familiar format of popular calendar applications like Google Calendar. This layout is designed to provide an overarching perspective of scheduled events over the course of a month. In instances where the volume of events exceeds the display capacity of a single cell, a button marked with three dots is introduced (as observed in the cell labeled with nine). Interaction with this button triggers an expansion of the cell, revealing all events scheduled for that particular day.

To improve navigation between timetable views, clicking on a cell in the monthly view automatically transitions the user to the corresponding week's view.

The monthly view's date range is determined by identifying the first Monday of the month and extending the range based on the calendar's dimensions, while excluding week-ends. Calendar applications typically align the calendar this way to maintain a consistent and predictable grid layout across all months.

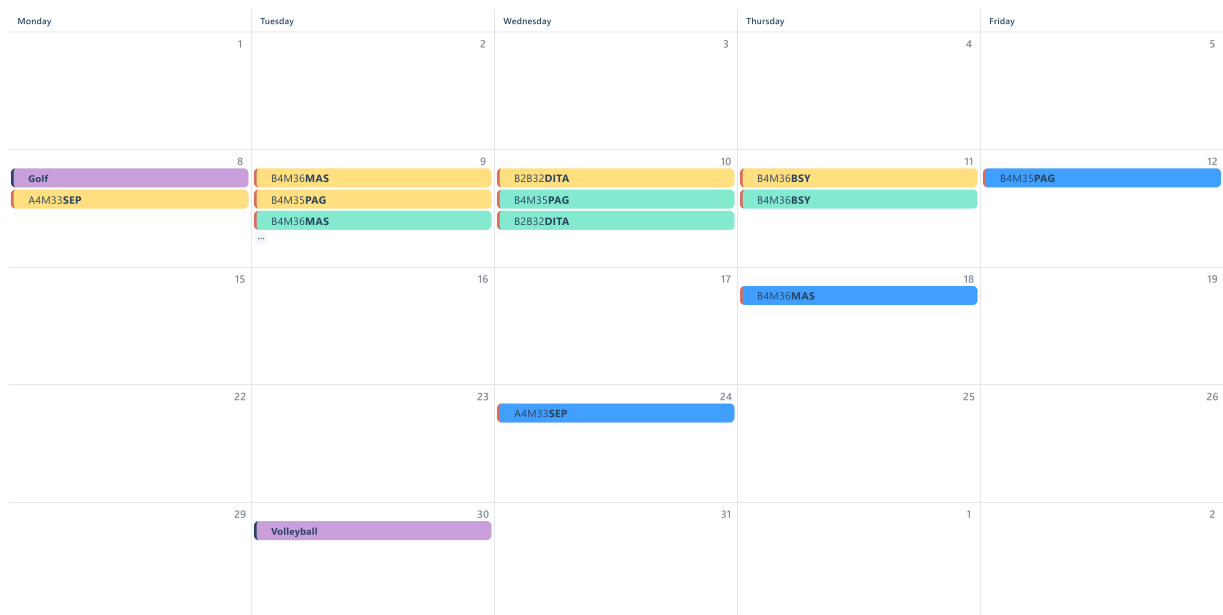


Figure 5.6: Monthly timetable view

5.1.4 Timetable Views On Phones

The timetable views for phones are presented in the figures 5.8, 5.9, 5.7. Apart from the discussed weekly and monthly views, which were adapted to smaller screens using CSS, there is also a daily view which is chosen on default. Thanks to that, user will immediately, upon arriving to the website, see their schedule for the day.

Contrary to what one might expect, the daily view is not a standalone component. It is in fact just a zoomed-in weekly view component. Thanks to that, it was possible to implement a smooth animation for transitioning between days.

Apart from using the day picker above the component, it is also possible to navigate between days by swiping left or right. A custom hook was built that detects a left or right swipe motion and triggers given functions. This interaction leverages the inherent interaction patterns of mobile device users to improve user experience.

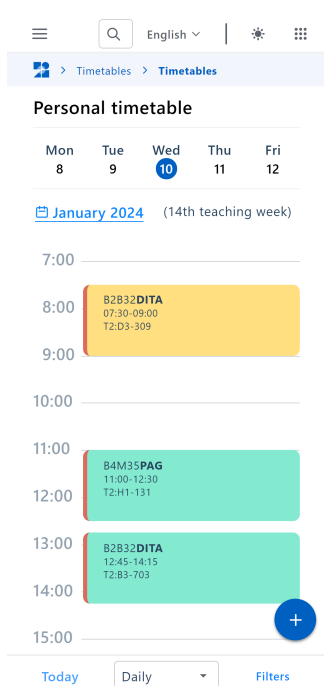


Figure 5.7: Daily timetable view for phones



Figure 5.8: Weekly timetable view for phones

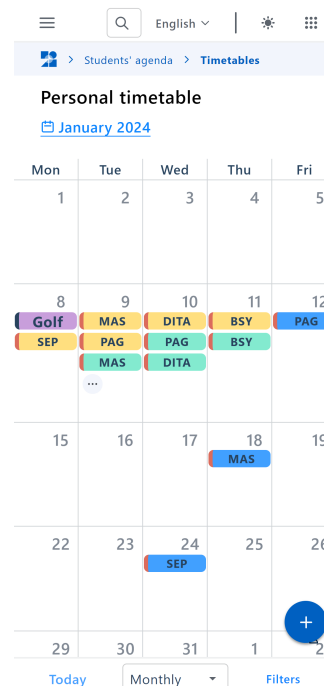


Figure 5.9: Monthly timetable view for phones

5.2 Timetable Controls

5.2.1 Switching Time

Adjusting the time of the timetable can be accomplished using various methods. In the daily timetable view, a menu displays the days of the week, as illustrated in the screenshot (5.7). To navigate the timetable by weeks or months, users can utilize the forward and back arrow buttons (5.10). To select a specific date, particularly one that is not immediately forthcoming, there is a calendar button (5.11) that opens a native calendar interface, allowing users to easily choose any date.



Figure 5.10: Arrow buttons

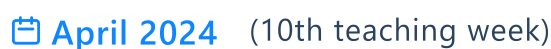


Figure 5.11: Calendar button with week information

HTML lacks built-in support for displaying a standalone calendar interface. Instead, when using an `<input type="date">` element, a box with the selected date is displayed. Since the design is custom, this default box has been hidden using CSS and upon clicking on the “fake” calendar button, the calendar interface is displayed programmatically by calling a `showPicker()` function. The appearance of the calendar interface is unfortunately inconsistent across different browsers, necessitating the use of browser-specific prefixes in CSS for proper styling and adjustments.

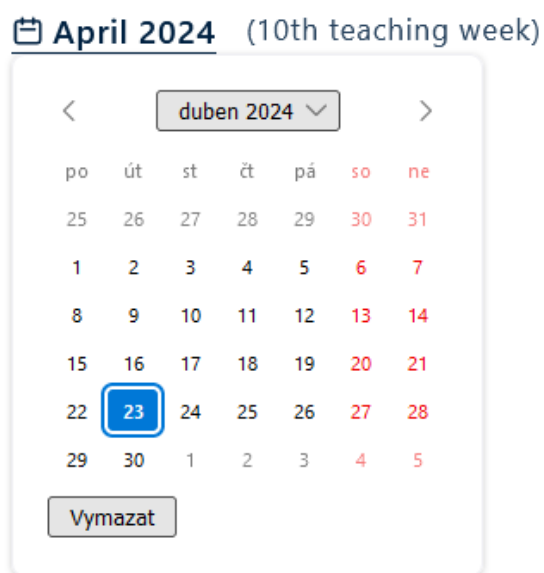


Figure 5.12: Calendar interface in Mozilla

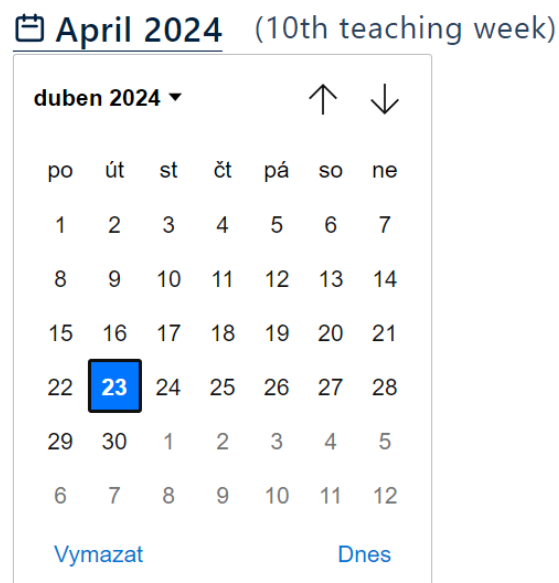


Figure 5.13: Calendar interface in Chrome

5.2.2 Timetable Page Controls

Tools for managing what events are displayed in the timetable are located in the retractable sidebar called **Filters**. Owners (people, rooms, or courses) can be added to the timetable through the search component (5.18) at the top. Once selected, owners are displayed in the sidebar, each accompanied by a checkbox and a remove button. If the checkbox is unchecked, the events for that owner are hidden. Hovering over an owner's name highlights their events, aiding in quick identification.

For easy identification, each owner is assigned a unique color. This color appears as a stripe on the event cards and in the corresponding checkbox. The UI/UX team provided a palette of 12 visually distinct colors to be used for the event owners. These colors are currently chosen based on the owners order.

In the screenshot 5.14, three owners are selected: the timetable user labeled as **personal**, a user with the username **gruncdam**, and a room **T2:C3-56**, which is unchecked and therefore its events are not displayed in the timetable. In addition, there are two categories of user events: **Sports** and **Errands**. Since **personal** is being hovered over, all other events are temporarily dimmed.

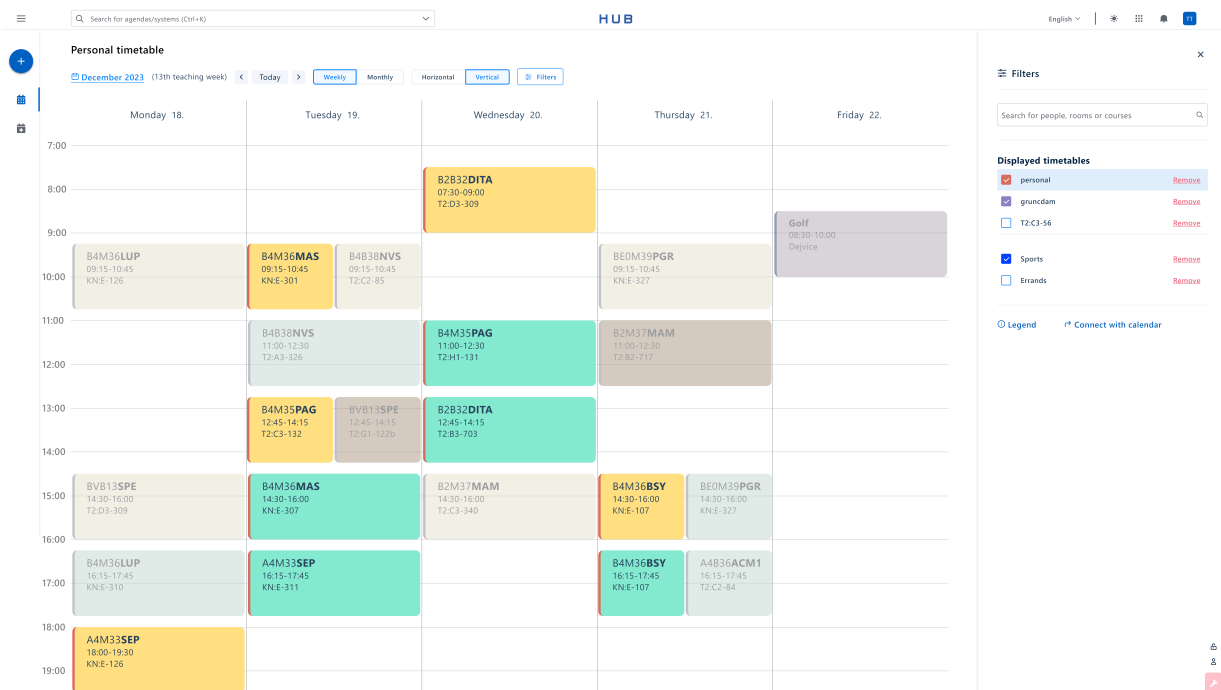


Figure 5.14: Timetable filters with a hovered owner

5.2.3 Timetable Planner Controls

The sidebar for timetable planner consists of a series of course cards, each containing a select menu with available parallels to choose from. A search bar is also available there under the plus icon; however, this time only courses can be searched and added. Similar to the timetable page, hovering on a course card highlights the corresponding events. Hovering on a specific parallel in the select menu highlights that parallel in the timetable. The cards can be retracted or expanded by clicking on the arrow in the bottom right corner.

The timetable only shows the selected parallels. If no parallels are chosen, all available options are displayed. Parallel numbers are indicated at the bottom of each card, except for lectures. The parallels 109, 101, and 108 of the course BOB01LGR are faded, indicating that they are fully occupied.

At the bottom, there is information about the total number of credits. While the total number of credits for the courses sums up to 20, only courses for 15 credits have a selected parallel. In case the course has both tutorials and laboratories, both have to have a parallel selected in order to count as being selected.

Under the credits text is a button that restarts the planner. In the default state, the planner displays courses that the user is enrolled in for the upcoming semester. After the user alters that configuration and leaves the page, the selection is saved. By clicking on the restart button, the planner will reset to the default state.

The screenshot displays the 'Timetable Planner' interface for the summer semester 24/25. The main area is a grid showing courses for Monday 22nd to Friday 26th. The vertical axis represents time from 7:00 to 19:00. Courses are represented by colored blocks with their respective course codes, times, and parallel numbers. For example, BOB01LGR is scheduled on Tuesday 10:00-10:45 (parallel 109) and Thursday 09:15-10:45 (parallel 112). Other courses include BE4B38PSIA, BEB01PST, and BE5B35APO.

On the right side, there is a 'Courses' sidebar. It lists the selected courses with their details:

- BOB01LGR** (Logika a grafy): 5 credits, tutorial: Not chosen.
- BEB01PST** (Pravdepodobnost a statistika): 4 credits, tutorial: 101.
- BE4B38PSIA** (Computer Networks): 5 credits, tutorial: 101, laboratory: 1011.
- BEB01PST** (Computer Architectures): 6 credits, tutorial: 101.

At the bottom of the sidebar, it shows 'Total credits: 20' and 'Chosen credits: 15'. A 'Reset timetable planner' button is located at the bottom right of the sidebar.

Figure 5.15: Timetable planner page

5.2.4 Timetable Controls On Phones

The sidebar's inherent horizontal layout made it unsuitable for phone views in its original form. Instead, a design similar to a bottom sheet² was chosen in the UI designs. This transformation of the sidebar was achieved solely by modifying the component's CSS.

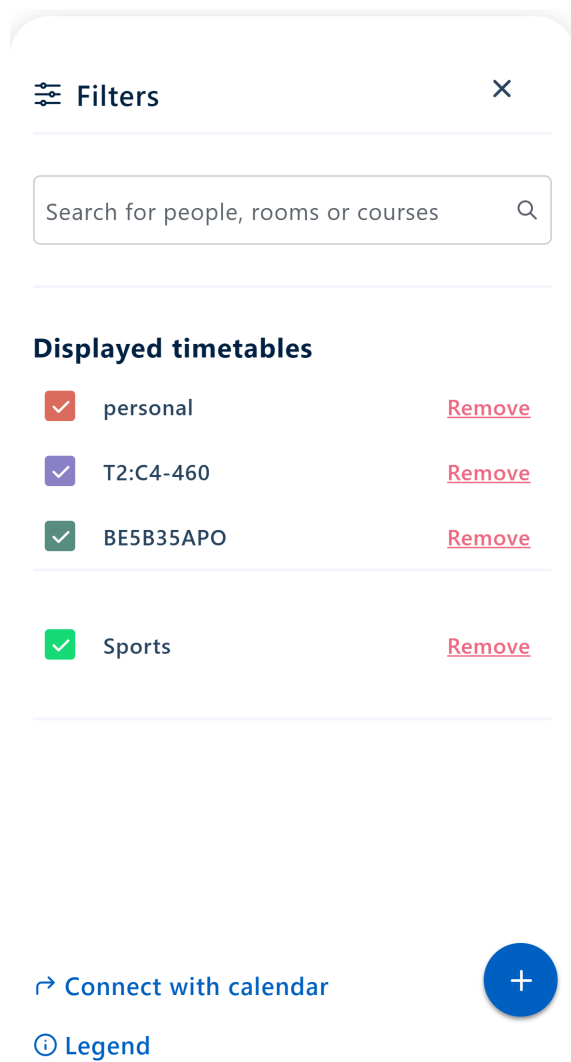


Figure 5.16: Timetable filters on a phone

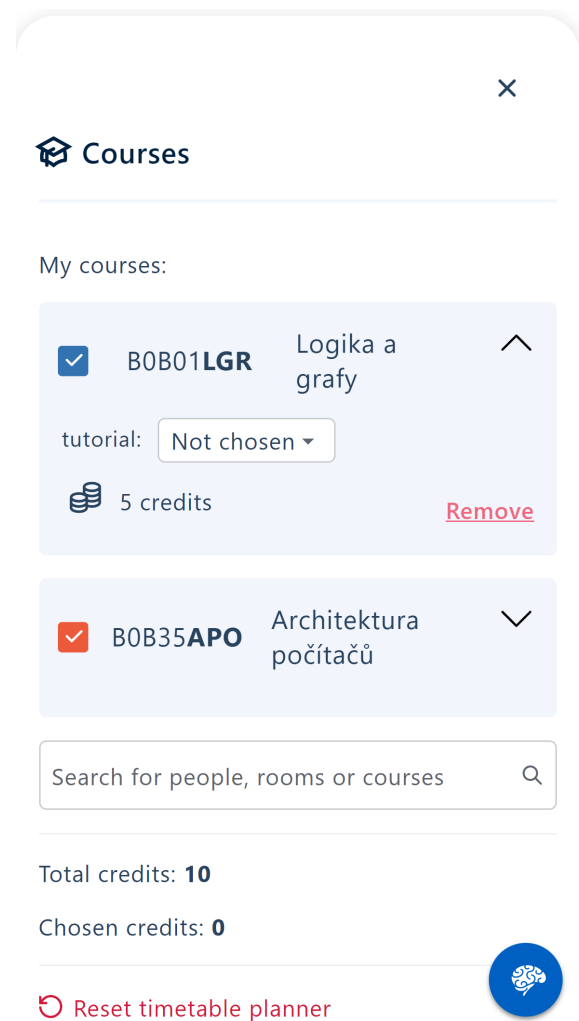


Figure 5.17: Planner filters on a phone

²A bottom sheet is a sliding panel that emerges from the bottom of the screen, providing additional content, options, or controls without leaving the current view [31].

5.3 Search

5.3.1 Grouped Search

The grouped search component depicted in figure 5.18 is designed to facilitate the addition of new timetable owners. This component retrieves individual owner types (courses, people and rooms) using distinct asynchronous queries. In the event of a failure, an error message is displayed, but successfully retrieved owner types continue to be shown. The same component is utilized in the timetable planner pages, although there it exclusively displays the course category.

Typing into the search will result in a grouped search across all three owner types.

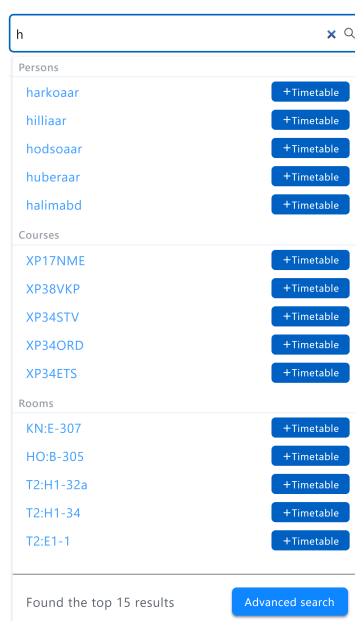
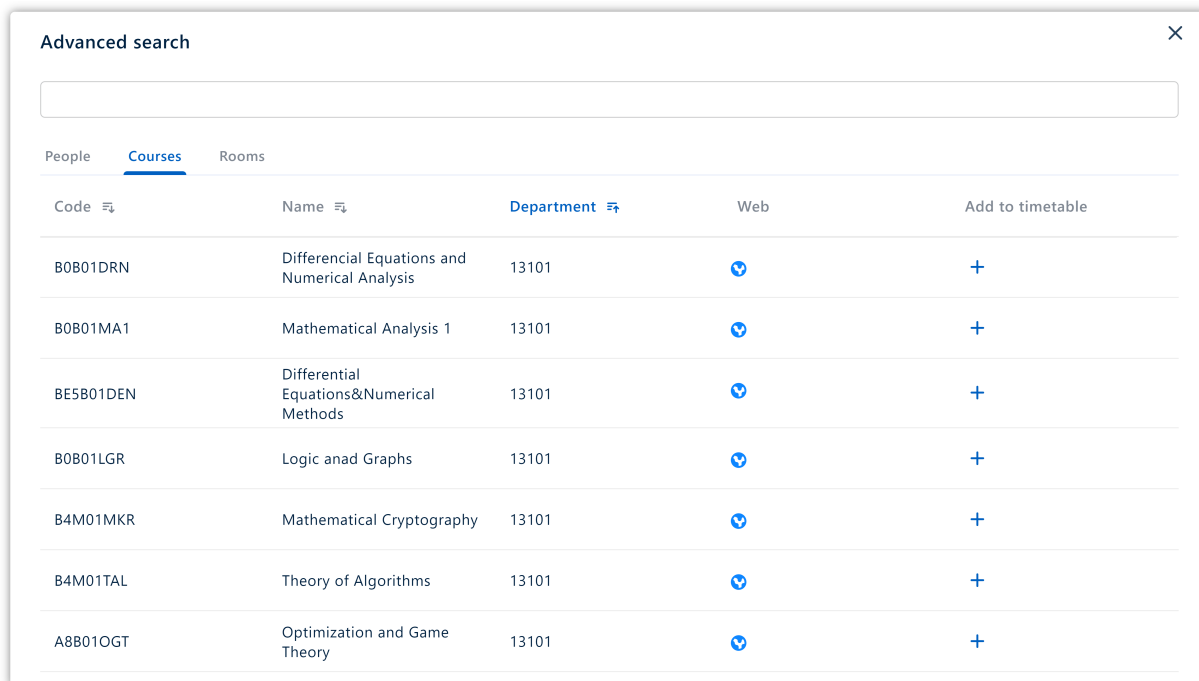


Figure 5.18: Grouped search component

5.3.2 Advanced Search

Clicking on the *Advanced search* button on the timetables page opens a modal with three tabs, each representing a different owner type, as shown in 5.19. Similar to the standard grouped search component, typing in the search bar initiates a search across all three tables³. This search is multirange, using the entered text to search both the codes and the names of the owners. The tables provide additional information about the timetable owners. Users can sort the data in ascending or descending order by clicking on the column titles. Each entry also features an Add to timetable button.

³The tables load lazily, reducing the number of queries required to be refetched on each keystroke.



Advanced search

People Courses Rooms

Code	Name	Department	Web	Add to timetable
B0B01DRN	Differential Equations and Numerical Analysis	13101		+
B0B01MA1	Mathematical Analysis 1	13101		+
BE5B01DEN	Differential Equations&Numerical Methods	13101		+
B0B01LGR	Logic and Graphs	13101		+
B4M01MKR	Mathematical Cryptography	13101		+
B4M01TAL	Theory of Algorithms	13101		+
A8B01OGT	Optimization and Game Theory	13101		+

Figure 5.19: Advanced grouped search modal

The *Advanced search* button in the planner page opens a different modal that specializes in searching courses. In the figure 5.20, there are advanced filters on the right side which control the content displayed inside the table on the left. In addition to the functionalities provided by FelSight, students can pick the exact times that tutorials and lectures take place separately. Furthermore, users can conveniently pick a credits range using the custom dual range slider. Searching a teacher is also more convenient because of an autocomplete feature which helps in finding the teacher even if the student only knows a part of their name for example.

In terms of implementation, the entire right side with filters is treated as a form with the *Apply* button serving as its submit button. The search above the table is also in the query. The response to submitting the form is presented as a table on the left. The input field for days of the week is a custom component, as well as the dual range slider. Since there is no native HTML element for the dual range slider, a custom one was created taking inspiration from [32]. The slider was created by appending two native single-range HTML sliders and using CSS to hide the fact that they are separate, and JavaScript to override their default behavior.

Code	Name	Semester	Credits	Action
XP17NME	Numerical Methods in Electromagnetic Field	SUMMER	4	+
XP38VKP	Selected Parts of Instrumentation	BOTH	4	+
XP34MSY	Microsystems	BOTH	4	+
XP34STV	VLSI Structures and Technologies	WINTER	4	+
XP34ORD	Optical Radiation Detection and Detectors	SUMMER	4	+
XP34APD	Advanced Power Semiconductor Devices and ICs	BOTH	4	+
XP34ETS	Electrical Transport in Semiconductors	WINTER	4	+
ASM99PR1	Project 1	SUMMER	6	+
XP135JD	Quality Control Systems	SUMMER	4	+
BIO101	Biology 101		0	+
CHEM101	Chemistry 101		0	+
XP34SRS	Semiconductor Radiation Sources	BOTH	4	+
XP02VNP	Plasma Waves and Instabilities	WINTER	4	+
AE2B32PRI	Programming	WINTER	5	+

Figure 5.20: Advanced course search

An alternative way of accessing the advanced course search is by clicking on gaps between events in the timetable planner. These gaps, represented as buttons, only appear if the interval between consecutive events exceeds 90 minutes⁴. During the rendering process, events are analyzed to identify these gaps, which are then stored in an array. For each identified gap, an `EventGapButton` component is generated. The placement of these buttons varies depending on the timetable's orientation, as illustrated in figures 5.21 and 5.22. Clicking on the button opens the advanced course search modal with preselected values for start and end times.

Figure 5.21: Horizontal event gap button

Figure 5.22: Vertical event gap button

⁴This duration was selected because it matches the typical length of a lecture or tutorial.

5.4 User Events

Clicking the plus button on the left panel of the timetable page opens a modal for creating new user events, as depicted in figure 5.23. Users are prompted to select a specific date and time. If the “Whole day” option is checked, as shown in Figure 5.24, users must select a date range. There are options to set the event to repeat daily, weekly, or monthly. Each event must be associated with a label, and new labels can be created by clicking the plus button next to the label list. Deleting a label and all associated events using the cross icon triggers a confirmation pop-up that informs the user about the consequences of this action.

Figure 5.23: User event form

Figure 5.24: Whole-day user event form

There is dedicated row in the timetable for whole-day user events while time-bound user events are treated as regular events, having their own event card with a detail. In the detail that is open for the **Golf** user event in the figure 5.25, user can remove a specific event or update it. Updating an event is implemented using the same form as for the event creation; however, the values are pre-filled there.

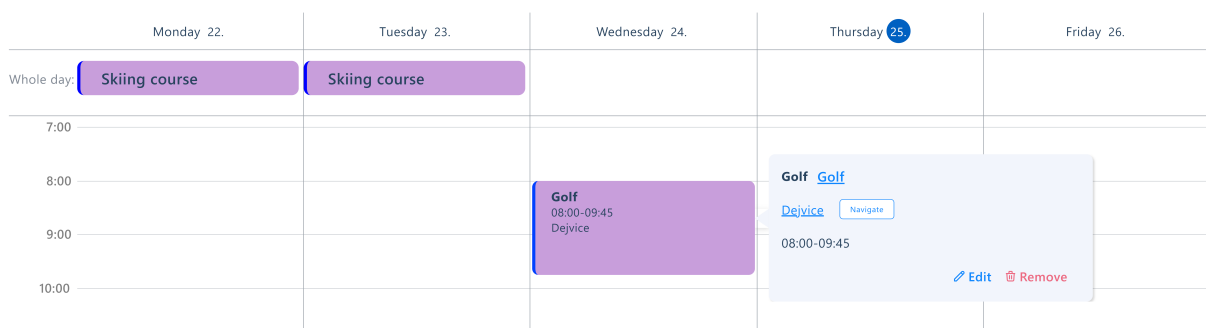


Figure 5.25: User events in timetable

5.5 Timetable Irregularity

The presence of courses with irregular parallels is detectable by simply checking whether they have a defined `weeks` field. In case an irregularity is detected, it is further inquired whether it is of parity type. That is, if the timetable is characterized by alternating between two configurations each week. In that case a week parity switch is displayed (5.26). If the irregularity is not characterizable by week parity, a switch with the irregular weeks is displayed (figure 5.27).



Figure 5.26: Week parity switch



Figure 5.27: Week switch

5.6 Details

5.6.1 Event Card Detail

When a user clicks on an event card in a timetable, the details of the event appear, providing specific information based on the user's current page. Figures 5.28 and 5.29 illustrate the details displayed on the timetable and planner pages, respectively. Each detail view includes the full course name, room code with a navigation option, time, and a list of teachers. The planner page detail additionally presents the occupancy of the parallel, which is particularly useful during timetable planning. It also features options to remove a course, cancel a parallel selection, and add a parallel to the planner. All titles in blue are interactive; clicking on the course name, room code, or teacher's name opens a detail modal for the corresponding entity.

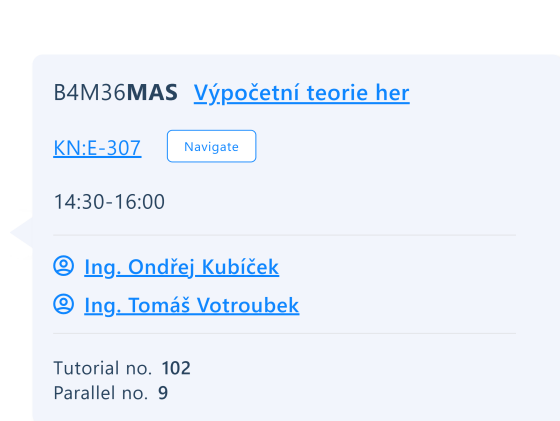


Figure 5.28: Event card detail

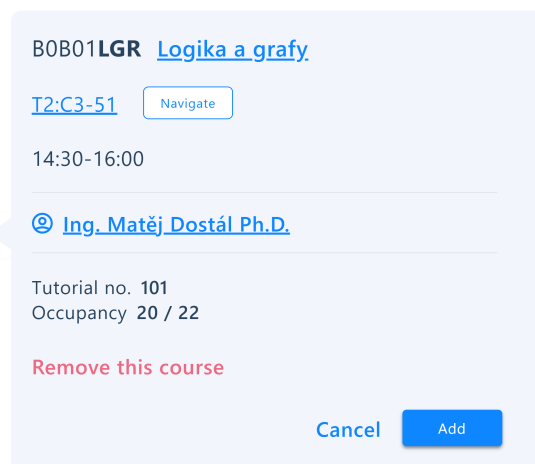


Figure 5.29: Planner event card detail

The event card details, which are designed to appear adjacent to the absolutely positioned event cards, extend beyond the original card when opened. The detail features a left-pointing triangle created using a CSS border trick⁵. As this triangle is a distinct shape, it cannot seamlessly share the box shadow with its neighboring rectangle without creating a visible separation. Therefore, the shadow cast by the entire detail is intentionally directed to the right to maintain a cohesive appearance.

5.6.2 Person Detail

Initially, it was planned to display a photograph of individuals in the person detail view if certain conditions were met; however, this approach was later adjusted to show only a placeholder image due to privacy concerns. The detail⁶ (5.30) includes the person’s full name along with their academic titles. As outlined in the analysis (4.1.6), the codes representing academic positions are parsed, and the most prevalent ones are translated. The departments and rooms associated with the person are listed if applicable. Email addresses are also displayed, with the preferred email highlighted. Clicking on an email address initiates the *mailto* link⁷, launching an email client (if available) to facilitate direct communication.

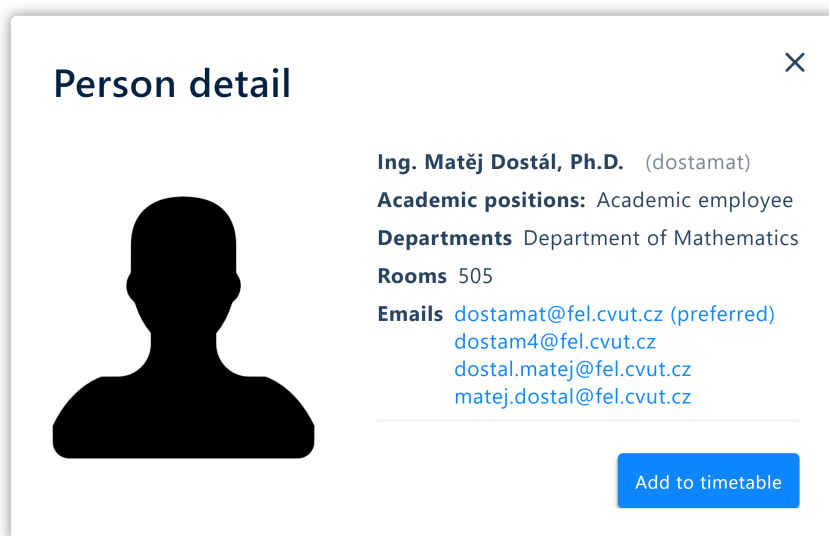


Figure 5.30: Person detail

⁵The CSS border trick involves using the border property to create triangles or other shapes by manipulating the width and transparency of different sides of an element’s border [33].

⁶Ing. Matěj Dostál, Ph.D has kindly agreed with presenting the person detail using his information.

⁷A “mailto” link is a type of hyperlink used on websites to automatically open the user’s email client with a new message window, pre-filled with information such as the recipient’s email address [34].

5.6.3 Room Detail

The room detail includes a top-down view of the floor where the room is situated, with the appropriate image selected by parsing the room code. If the floor is not identifiable, a placeholder image is used. Currently, these images are raster screenshots, but there are plans to replace them with high-quality SVGs in the future.

In addition, the details feature a dynamically generated sentence that describes the room based on its floor, building, address, and block. This description is crafted by parsing the room code and analyzing its components according to established naming conventions.

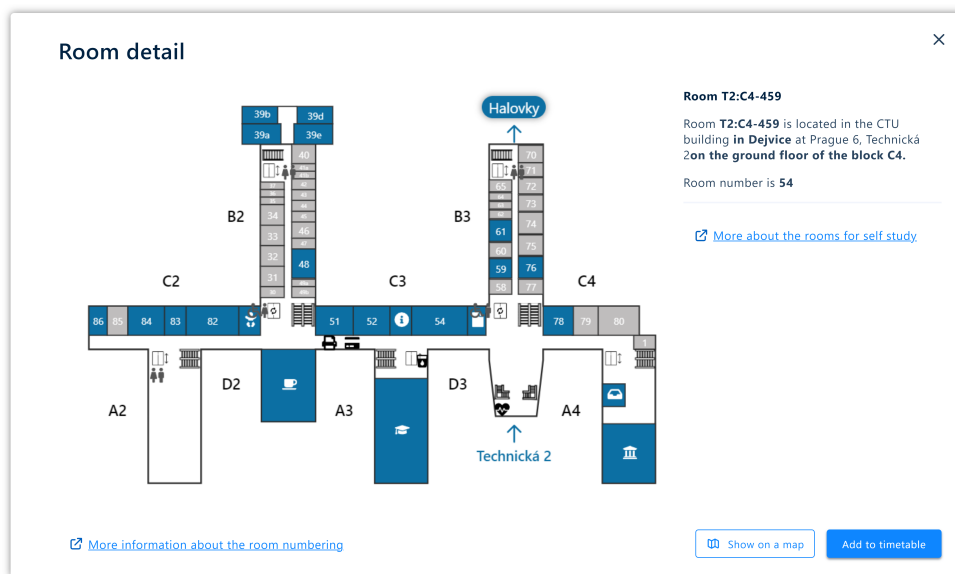


Figure 5.31: Room detail

5.6.4 Course Detail

The course detail provides additional information about the course, including recommended literature, a description, and the requirements. However, the individual fields within this detail are not standardized, making it challenging to reformat the content into a more structured presentation, such as bullet points.



The screenshot shows a window titled "Course detail" with a close button (X) in the top right corner. The course name is "Computer Architectures" with the code "BESB35APO". The language is "English", completion is "Assessment with exam", and the range is "2P+2L". The description states: "Subject provides overview of basic building blocks of computer systems. Explanation starts from hardware side where it extends knowledge presented in the previous lectures of Structures of computer systems. Topics cover building blocks description, CPU structure, multiple processors interconnections, input/output subsystem and basic overview of network and buses topologies. Emphasis is placed on clarification of interconnection of hardware components with software support, mainly lower levels of operating systems, device drivers and virtualization techniques. General principles are more elaborated during presentation of examples of multiple standard CPU architectures. Exercises are more focused on the software view to the contrary. Students are lead from basic programming on CPU level to the interaction with raw hardware." The requirements section says: "Basic knowledge of C language and area of combinatorial and sequential logic circuits. Basic knowledge of command line and compilers use in POSIX standard conformant environment (i.e. Linux) is invited." The literature section lists several references: [1] Hennessy, J. L., and D. A. Patterson. Computer Architecture: A Quantitative Approach, 3rd ed. San Mateo, CA: Morgan Kaufman, 2002. ISBN: 1558605967. [2] Patterson, D. A., and J. L. Hennessy. Computer Organization and Design: The Hardware/Software Interface, 3rd ed. San Mateo, CA: Morgan Kaufman, 2004. ISBN: 1558606041. [3] Andrew S. Tanenbaum: Structured Computer Organization, Prentice Hall, 2006. ISBN-10:0131485210. [4] Andrew S. Tanenbaum: Computer Networks, Prentice Hall 2003. ISBN-10:0-13-066102-3. [5] Andrew S. Tanenbaum: Modern Operating Systems, Prentice Hall 2001 [6] Hyde, R.; The Art of Assembly Language, 2003, 928 pp. ISBN-10 1-886411-97-2 ISBN-13 978-1-886411-97 http://webster.cs.ucr.edu/AoA/ [7] Bach, M., J.: The Design of the UNIX Operating System, Prentice Hall, 1986 [8] Bayko., J.: Great Microprocessors of the Past and Present http://www.cpushack.com/CPU/cpu.html. There is a blue button labeled "Add to timetable" in the bottom right corner.

Figure 5.32: Course detail

5.7 Comparison With Designs

Although the general goal of the frontend implementation was to remain as close to the UI designs as possible, some changes were made. All the modifications listed below were reviewed and approved in consultation with the UI/UX team and the thesis supervisor.

- Timetable Views
 - The new designs did not incorporate the horizontal timetable at all. For that reason a timetable orientation switch was added.
 - The designs use a select menu to switch between timetable views. This is in contradiction with the design system guidelines which state that for 2-4 items, radio buttons are to be used.
 - Unlike in designs, the timetable components do not display weekends. This is to significantly decrease the implementation complexity as well as increase readability on small devices (without including weekends, the event cards are almost 30% wider, which increases readability significantly).
 - Collision management was not defined in the UI designs. The padding adjustments, course code shortening, etc. all went beyond the provided designs.
 - In case of vertical overflow, the scrollbar is bound to the timetable component itself in the designs. This setup fails because some actions such as opening an event detail may also trigger an overflow. In addition, there have been numerous responsiveness issues with the component, necessitating a comprehensive redesign of the entire implementation.
 - Timetable Controls
 - While the designs contain a custom implementation of a calendar widget, a native HTML component has been used instead. This is to increase accessibility and decrease implementation complexity.
 - In the UI designs, the cards in *Filters* sidebar (5.14) are divided into categories based on the event type – lecture, tutorial, laboratory, etc. This was changed to be the same as in FelSight where the events are filtered by their owner – a person, course or room.
 - In the monthly timetable view (5.6), some interactions were added. For example, clicking on a day will redirect user to a weekly view of a particular week this day is in. Clicking on a button with three dots results in showing all of the events of that day by making the cell taller. Both of these interactions were improvised and may be subject to change in the future.
-

- Search
 - Designs for the regular search component (5.18) were not delivered and were therefore improvised to look similar to FelSight’s search components while remaining within FEL.HUB design guidelines.
 - A column with a button to redirect user to a course’s homepage was added to the tab with courses of the advanced grouped search (5.19). This column was present in FelSight’s grouped search (2.3) and was thus included in order to not deviate from the original data presentation.
 - The `EventGapButton` (5.21, 5.22) was not a part of the UI design. It is based on the same feature that is available in FelSight’s planner.
 - User Events
 - The way labels are added was not included in the designs. It was implemented as a miniature form with one text box and a native color input so that the user can enter any color.
 - Timetable Irregularity
 - Because UI/UX team did not deliver the designs for the necessary components, they were improvised. As a result, their functionality is limited because the direction of the final designs remains unclear.
 - Details
 - Unbeknownst to UI/UX team, some items in the data types of the detail subjects are lists. For example, a person can have multiple rooms, multiple emails and be associated with multiple departments. These fields were therefore pluralized and the items therein listed vertically (5.30).
 - Since the data type of a person includes their preferred email, it was highlighted in the listing of emails (5.30).
-

Chapter 6

Timetable Optimization

Contents

6.1	Initial Solution	59
6.2	Complexity	62
6.3	Implementation	63
6.4	Example	66

6.1 Initial Solution

FelSight gives users a helping hand when creating a timetable for the next semester by providing a functionality that aims to find the “ideal” timetable algorithmically.

In timetable planner, users choose a set of courses they wish to attend for the next semester. Then, for each of the courses, they have to choose a parallel. Timetable optimization refers to finding a set of parallels that together form a timetable with the highest score.

In FelSight, when the user clicks on “Optimize timetable” button, a specialized service is called. This service runs a genetic algorithm¹ on the given set of courses and parallels using Optaplanner’s² planning engine. The criteria for finding the optimal timetable are defined using constraints. These constraints use advanced data structures provided by Optaplanner to access and quickly iterate over the intermediate solution and give it a score

¹Evolutionary algorithms are a class of optimization algorithms that simulate the process of natural selection by using techniques like mutation, crossover, and selection to generate solutions to optimization and search problems [35].

²OptaPlanner is an open-source optimization engine primarily aimed at tackling scheduling optimization problems [36].

based on the function's purpose. For example, there might be a function that subtracts 100 from the total score for each day in the timetable that has events in it. This makes the timetables with fewer days in the week have a higher score and thus are more likely to be picked as optimal. The following list showcases all of the constraints.

- `penalizeEventGaps` – penalizes each gap between events by the length of the gap in seconds.
- `penalizeDays` – penalizes each day with at least one event
- `penalizeParallelsOverlaps` – penalizes each occurrence of two events overlapping.
- `penalizeMultipleBuildingsPerDay` – penalizes for each day having at least two events in different buildings.
- `penalizeInsufficientTimeToCommute` – penalizes each day having at least two events in different buildings with a longer duration between them than `COMMUTE_TIME`.
- `penalizeLunchTimeIntersection` – penalizes each occurrence of an event intersecting with preselected lunch time.

`FelSight` interacts with the timetable optimization service using `RabbitMQ`, a message broker that adheres to the `AMQP`³ specification [38]. This setup provides a standardized method for executing potentially time-consuming algorithms without the constraints and timeout issues associated with the `HTTP` protocol. By leveraging `RabbitMQ`, requests for timetable optimization are efficiently queued and managed, ensuring that they are processed systematically and without delay, thereby enhancing the system's performance and reliability.

The constraints presuppose certain preferences regarding student timetables. For instance, it is assumed that students would favor a configuration of parallels that minimizes the number of days spent on campus. Although this seems like a plausible assumption, preferences may vary; some students might prefer attending two parallels over three days, rather than three across two days. These constraints do not adequately account for the diverse range of personal preferences that students may have. To address this issue, it is important to provide students with the flexibility to select which constraints they wish to apply to their timetables. This approach would allow for a more personalized scheduling experience that accommodates individual preferences and requirements.

In the preliminary phase of the analysis, prior to defining the scope of this thesis, it was hypothesized that the deployment of `OptaPlanner` may be superfluous. This stems from the fact that the dimensions of realistic timetable problem instances seem to be sufficiently modest to permit efficient resolution via brute force methods on modern computers. To

³`AMQP` stands for Advanced Message Queuing Protocol. It is an application layer protocol designed for sending messages between systems in a reliable and standardized way [37].

be specific, consider a scenario where the instance size is restricted to eight courses, each with ten parallels, an atypically large configuration, but feasible in theory. In such a case, a brute force algorithm, devoid of any optimization or pruning techniques, would be required to evaluate 10^8 permutations. This quantity, while substantial, appears to be manageable within computational limits.

Should the reliance on a specialized optimization solver be successfully eliminated, there would be no need for the feature to be implemented on backend. Consequently, a naive solution could be implemented directly in JavaScript. This approach would not only simplify the architecture, but would also reduce the operational complexities and associated overheads.

In conclusion, the objective is to develop a brute force algorithm in JavaScript that enables users to select the constraints they wish to apply. If its performance meets expectations, it would eliminate the need for the timetable optimization service, thereby reducing the strain on infrastructure. Furthermore, the new algorithm should provide users with a more personalized experience by letting them have more control over the optimization process.

6.2 Complexity

For this section, let us denote the discussed timetable optimization problem as \mathcal{TO} . To get an idea of how fast \mathcal{TO} can be, it is good practice to try to fit it into a corresponding complexity class. In this section, it will be shown that \mathcal{TO} belongs to a class of \mathcal{NP} -complete problems. This can be done by constructing a polynomial reduction from a problem that is already known to be \mathcal{NP} -complete [39].

A scheduling problem $1 | r_j, \tilde{d}_j | C_{\max}$ was chosen for the reduction. There are n tasks such that the task j has a release time r_j , a deadline \tilde{d}_j and a processing time p_j . The goal is to find a schedule such that the the last task ends before C_{\max} . Specifically, its decision version, which is also \mathcal{NP} -complete [40], will be used. It asks whether a timetable of size at most C_{\max} exists.

The $1 | r_j, \tilde{d}_j | C_{\max}$ problem has been proven to be strongly \mathcal{NP} -complete [41]. This means that it remains \mathcal{NP} -complete even when all numerical parameters are limited to values that are polynomially bounded by the length of the input [42]. The conversion will leverage this fact and work with the assumption that the parameters r_j , \tilde{d}_j and p_j are integers bounded by $O(n)$. Let us show that $1 | r_j, \tilde{d}_j | C_{\max}$ can be converted into \mathcal{TO} in polynomial time.

For each task j create a new course with $\tilde{d}_j - r_j - p_j + 1$ parallels such that the i -th parallel starts at the time $r_j + i$ and ends at $r_j + i + p_j$.

For example, for a task with $r = 4$, $\tilde{d} = 10$, and $p = 3$ the start and end times of the corresponding parallels would be $(4, 7)$, $(5, 8)$, $(6, 9)$, $(7, 10)$. This symbolizes all the possible choices of placing the task within its release time and deadline.

Set the \mathcal{TO} parameters in the following way:

- Let all parallels take place on the same day in order to establish a single timeline.
- Let all parallels take place in the same building so that the penalization for different buildings does not interfere.
- Set the penalization for colliding parallels to ∞ to simulate having only one processing element.
- Set lunchtime intersection penalization to ∞ and lunchtime to occur at the interval $[C_{\max}, \infty)$ so that the C_{\max} bound is respected.

Run the \mathcal{TO} algorithm on the new problem. The timetable of size at most C_{\max} exists if and only if \mathcal{TO} algorithm finds a timetable with a finite score.

There are n courses, each with the number of parallels that is polynomially bounded as assumed, which makes size of the newly created \mathcal{TO} instance polynomial. \mathcal{TO} is therefore \mathcal{NP} -complete.

6.3 Implementation

The implementation employs a brute force approach, characterized by the generation of permutations of potential parallel choices. Upon generation of each permutation, its corresponding score is evaluated by a scoring function. The best score and permutations are then updated accordingly. To facilitate the management of the best score and its associated permutations, the function described in algorithm 1 is incorporated within a class, where these elements are maintained as class attributes. The function is designed to track not only the highest score, but also the top N scores (lines 8-10). This approach provides users with several choices, accommodating the possibility that the optimal permutation might not always meet their specific requirements. Also, in the beginning, the user is prompted to select the constraints they wish to be applied. That is reflected in the line 35 where it is checked whether the current constraint is allowed.

An important aspect of the score calculation is its impact on the addition of parallel elements. Except for `penalizeEventGaps`, all components of the score calculation are designed to ensure that adding another parallel will invariably **decrease** the score. This is because inserting a new parallel between two existing ones decreases the sum of the gaps between them, consequently increasing the score. By adjusting this constraint to penalize based on the length of the timetable rather than by summing all the gaps, we guarantee the preservation of the score decreasing invariant. This adjustment allows for significant pruning of the decision tree by halting the process if the interim score is worse than the best score found (lines 14 to 16). This mechanism operates on the assurance that once the score becomes worse than the best score, it cannot improve further. Extensive testing has confirmed that the algorithm yields the same timetable regardless of whether pruning is applied.

In order to measure the performance of the algorithm, let us define its parameters and, subsequently, derive the time complexity.

Let m denote the number of courses such that the i -th course has n_i parallels. Let $n = \max\{n_1, \dots, n_m\}$. The upper bound for the number of permutations the `perm` function would have to go through would therefore be $O(n^m)$. The `getScore` function adheres to the constraints outlined in section 6.1. Initially, it maps parallels from the current permutation to appropriate days in $O(mn)$. It then organizes the events for each day by start time, requiring $O(mn \log(mn))$ steps. The function proceeds by iterating over each day and its events, applying the relevant penalizations to the permutation score. The penalizations take at most $O(mn)$ time, so the overall time complexity of evaluating the score is $O(mn \log(mn))$. The total time complexity of the algorithm is $O(n^m mn \log(mn))$.

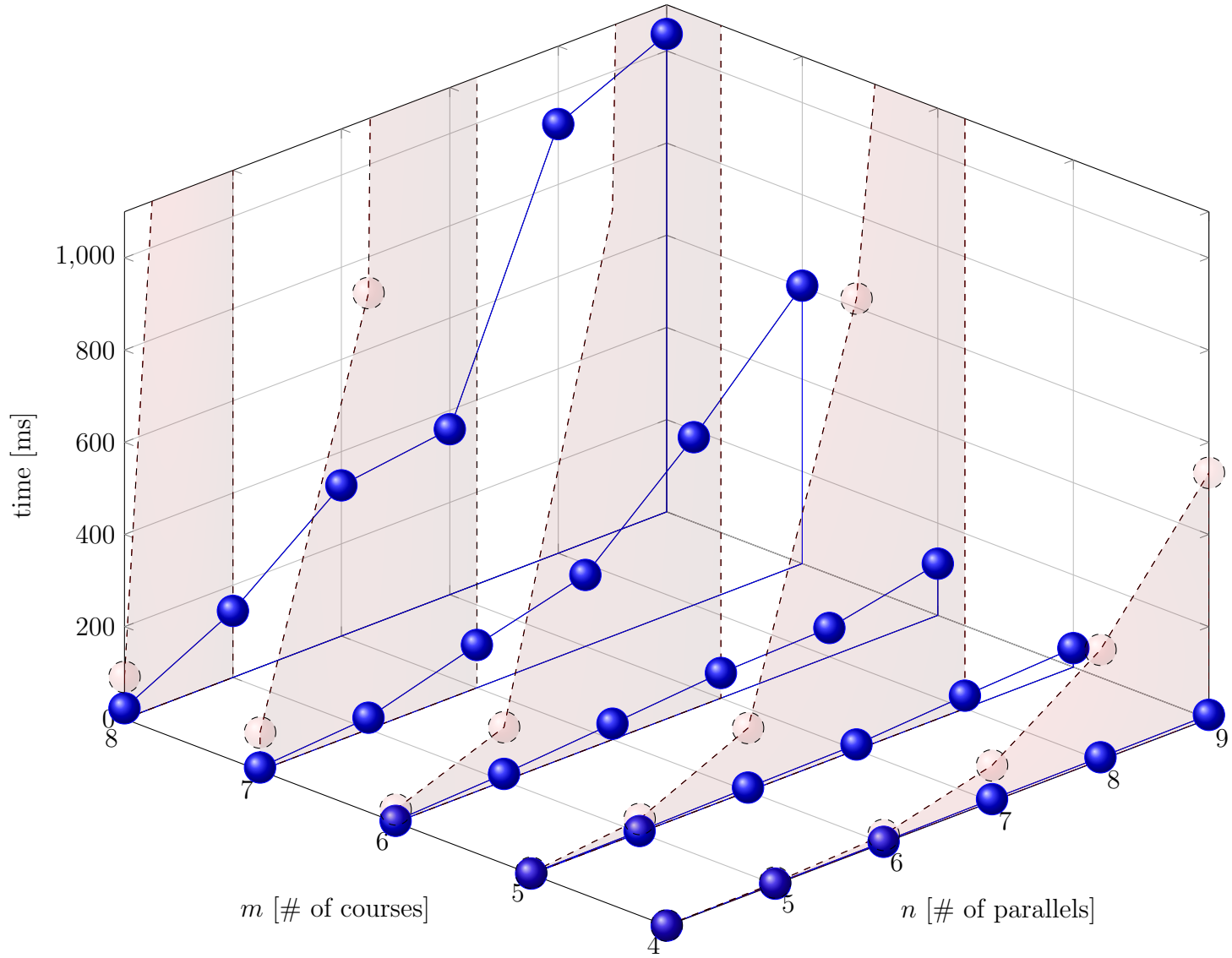
While the initial algorithm exceeded one second at $m = n = 6$, the pruned version handles up to $m = 8$, $n = 9$ instances until it reached a second which is more than satisfactory for the real life use cases. The graph 6.1 shows the comparison between the original (red) and pruned (blue) version of the algorithm. Due to significant variations in individual measurement times, the graph displays the average times over 100 runs.

Algorithm 1 Pseudocode for the \mathcal{TO} algorithm

```

1: procedure PERM(courses, indices = [], depth = 0)
2:   if depth = length of courses then
3:     score  $\leftarrow$  GETSCORE(courses, indices)
4:     if score > this.bestScore then
5:       this.bestScore  $\leftarrow$  score
6:       this.currentBestPerm  $\leftarrow$  indices
7:       this.bestPerms.push(indices)
8:       if length of this.bestPerms > this.numberOfBest then
9:         this.bestPerms  $\leftarrow$  this.bestPerms[1 : this.numberOfBest]
10:      end if
11:    end if
12:    return
13:  end if
14:  if GETSCORE(courses, indices) > this.bestScore then
15:    return
16:  end if
17:  for i  $\leftarrow$  0 to length of courseTimetables[depth].parallels - 1 do
18:    PERM(courses, indices + [i], depth + 1)
19:  end for
20: end procedure
21: procedure GETSCORE(courses: array of CourseTimetable, indices: array of indices)
22:   Initialize slotsByDays to array of 5 empty arrays
23:   for i = 0 to min(length of courses, length of indices) - 1 do
24:     parallelSlots  $\leftarrow$  courses[i].parallels[indices[i]].timetableSlots
25:     for each timetableSlot in parallelSlots do
26:       Append timetableSlot to slotsByDays[timetableSlot.weekDay - 1]
27:     end for
28:   end for
29:   for each day in slotsByDays do
30:     Sort day by start time
31:   end for
32:   penalization  $\leftarrow$  0
33:   for each day in slotsByDays do
34:     for each constraint in constraints do
35:       if constraint is allowed then
36:         penalization  $\leftarrow$  penalization + CONSTRAINT(day)
37:       end if
38:     end for
39:   end for
40:   return -penalization
41: end procedure

```

Figure 6.1: Performance of \mathcal{TO} with and without pruning

6.4 Example

To illustrate the way \mathcal{TO} can be used, let us consider an example⁴ timetable (6.2). The green cards are the tutorials that the student is expected to choose from.

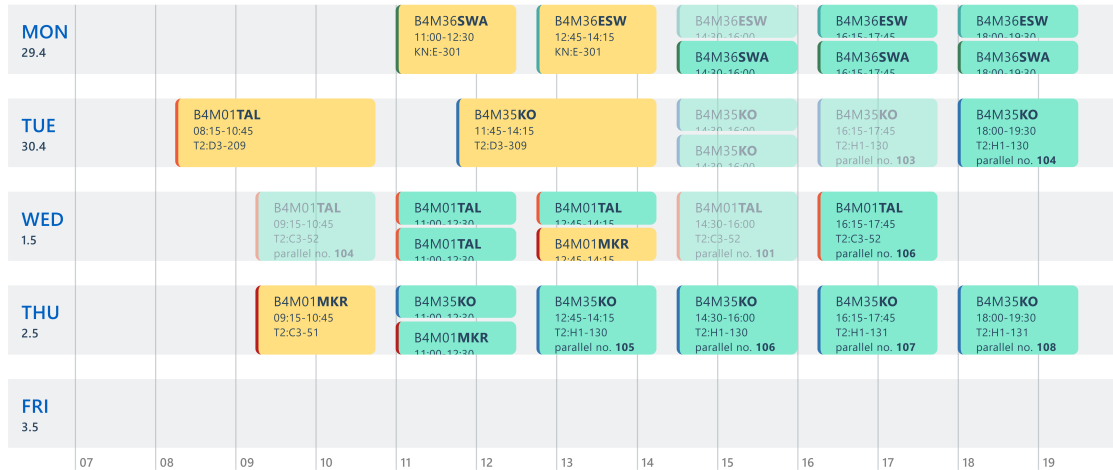


Figure 6.2: Timetable with parallels to choose

For SWA and ESW \mathcal{TO} has selected the only permutation of parallels that minimizes gaps while accommodating capacity. For KO on Tuesday, the only sufficiently large parallel is impractically distant from the lecture. Conversely, scheduling KO with MKR on Tuesday results in a more favorable configuration. For TAL, the only viable option is the parallel before MKR, as all others are either too distant or conflicting with MKR.

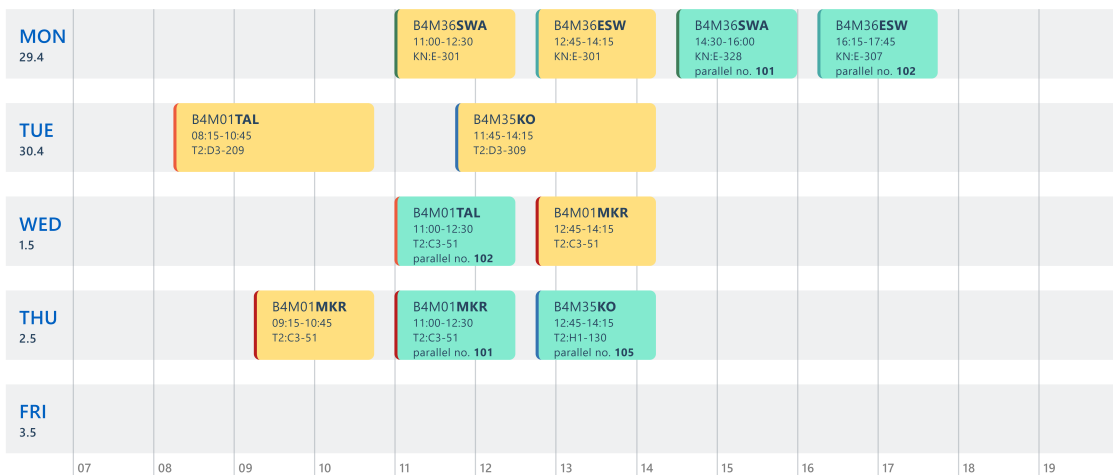


Figure 6.3: Chosen timetable

⁴The figure depicts a real timetable a student may face in the second semester of Open Informatics with a specialization in software engineering.

Chapter 7

Conclusion

Contents

7.1 Improvements	69
-----------------------------------	-----------

Following the description of the general goals of the project (1.1) and outlining its structure (1.2), FelSight application was introduced (2). The individual pages of FelSight were presented with a brief description. In the table 2.1, the features of FelSight were classified according to their migration status. The description of technologies was thorough and included actual code examples (2.1, 2.2, 2.3). This was followed by a detailed examination of FelSight’s challenges (2.3), such as complexity (6.2), size (2.3.2), and performance (2.3.3), supported by relevant metrics.

Subsequently, the concept of FEL.HUB was introduced (3), detailing its technologies (3.2) and architectural design (3.3). The key advantages of FEL.HUB’s technologies are presented in the list 3.2.2. Finally, a comprehensive list of motivations behind the project was presented, discussing how migration of FelSight to FEL.HUB solves the previously mentioned issues (3.4).

Since the frontend implementation is based on UI designs, the analysis chapter (4) focuses mainly on the backend part (4.2). The individual services are presented with a brief description of their purpose. Importantly, for each service, a listing of the GraphQL queries that need to be implemented is provided. An extensive use case diagram (4.1) outlines the general requirements, with use cases grouped and described by categories.

The backend implementation builds on previous work by Ladislav Svoboda ([2]), originally intended for FelSight and REST communication. The work that needed to be done on the backend services is discussed in the chapter 4.2. Details of the backend implementation are omitted as they involve standard practices and do not add substantial scholarly value.

The frontend implementation (5) is organized by categories identified during the analysis of use cases (4.1). Screenshots of components fulfilling these use cases are shown under each category, with detailed discussions of the components' behaviors and key implementation decisions. Where the implementation deviates from the original designs, these changes are cataloged separately (5.7).

The algorithm for timetable optimization, denoted as \mathcal{TO} , is comprehensively covered in its own chapter (6). Firstly, the initial solution is presented (6.1). \mathcal{TO} is then classified as \mathcal{NP} -complete (6.2) by constructing a polynomial reduction from $1 | r_j, \tilde{d}_j | C_{\max}$. Implementation details are discussed thereupon, introducing the solution in a pseudocode (1). A powerful pruning technique was discovered, significantly accelerating the algorithm. This enhancement enables it to handle instances far larger than those typically considered practical. The chapter concludes with an example of \mathcal{TO} applied to an actual timetable (6.4).

The source code for the frontend went through a thorough code review and is now in the stage of manual testing in the development environment. Locations of the codebases of the frontend¹, Timetable service², Course semester service³, and Room service⁴ are listed in the corresponding footnotes.

¹<https://gitlab.fel.cvut.cz/czm/hub/frontend/frontend-base>

²<https://gitlab.fel.cvut.cz/czm/hub/rozvrhy/timetable-service>

³<https://gitlab.fel.cvut.cz/czm/hub/rozvrhy/course-semester-service>

⁴<https://gitlab.fel.cvut.cz/czm/hub/rozvrhy/room-service>

7.1 Improvements

Let us get back to the limitations of FelSight discussed in the chapter 2. Namely, let us compare the complexity (2.3.1), size (2.3.2) and performance (2.3.3) between the two projects.

During the development process, considerable emphasis was placed on minimizing the code **complexity**. Efforts were concentrated on ensuring the cohesion and modularity of the codebase. As illustrated in the component diagram 5.1, the system is comprised of over fifty distinct components. Each component is designed to fulfill a specific function, closely aligned with a corresponding use case as shown in the use-case diagram 4.1. To facilitate readability and maintain proper modularity, each component was intentionally restricted to no more than 200 lines of code. Moreover, the timetable optimization algorithm was kept as simple as possible while still achieving its practical purpose.





Size of the project is difficult to compare – the exact number of lines of code used to develop timetable and planner pages is unclear. Moreover, the code is now split between frontend and backend. There are approximately 10000 lines of code on the frontend (counting only lines from this project) and 4000-6000 lines on the backend. That is approximately 15% of the size of FelSight while capturing its most essential functionalities. That seems to be a significant improvement, although it is challenging to fairly compare the two systems in this regard.

The **performance** of FEL.HUB significantly surpasses that of FelSight. Logging in itself has improved dramatically from nearly 14 seconds (2.13), to 400 milliseconds. The initial loading time of the site is approximately 100 milliseconds, during which, owing to its architecture as a single page application, all essential frontend components are loaded. Subsequent interactions involve only data retrieval. Since the loading times are much smaller now, the measured values are greatly influenced by the user's geographic location and the development environment in which FEL.HUB runs. In the development environment, queries are completed within a maximum of 100 milliseconds. In the production environment, additional security measures can extend response times to up to 500 milliseconds. This proves to be a significant improvement.

Other improvements such as reduced development cost and increased scalability are inherent properties of the new system and are detailed in section 3.4.

AI Acknowledgment

I acknowledge the use of artificial intelligence during the development and writing of this thesis. This usage was in accordance with the *Methodological guideline No. 5/2024*⁵. The software used and the purposes for its application are detailed below.

-  **ChatGPT**⁶ aided in text stylization, rephrasing and also assisted with data transformations and provided answers to simple coding questions.
-  **Writefull**⁷ assisted with spell checking and offered valuable grammatical and stylistic suggestions.
-  **GitHub Copilot**⁸ provided autocompletion for repetitive and straightforward code snippets.
-  **JetBrains AI**⁹ assistant delivered answers to more specialized and code-heavy questions.

In the figure 7.1, the relative usage of the mentioned AI tools is estimated.

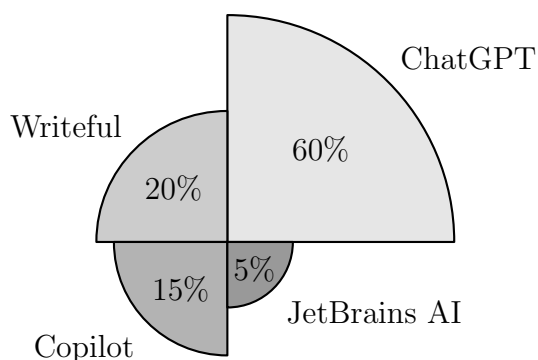


Figure 7.1: AI tool usage distribution

⁵<https://intranet.fel.cvut.cz/cz/rozvoj/MP-pouzivani-ui.pdf>

⁶<https://chatgpt.com/>

⁷<https://www.writefull.com/>

⁸<https://github.com/features/copilot>

⁹<https://www.jetbrains.com/ai/>

Bibliography

- [1] L. Baronová, “Analysis and design of an application to support students during their studies at university,” Bachelor’s thesis, Czech Technical University in Prague, 2024. [Online]. Available: <https://dspace.cvut.cz/handle/10467/113410>
 - [2] L. Svoboda, “Migration of FELsight application to microservice architecture,” Bachelor’s thesis, Czech Technical University in Prague, 2023. [Online]. Available: <https://dspace.cvut.cz/handle/10467/109283>
 - [3] A. Kohout, “The architecture transformation of FelSight faculty application,” Master’s thesis, Czech Technical University in Prague, 2022. [Online]. Available: <https://dspace.cvut.cz/handle/10467/101696>
 - [4] K. D. Mann, *JavaServer Faces in Action*. Manning Publications, 2005, p. 58.
 - [5] A. Leonard. (2014) Mastering JavaServer Faces 2.2.
 - [6] C. Richardson, *Microservices Patterns: With examples in Java*. Manning, 2018, p. 225.
 - [7] K. T. Jeffrey Hassan, *Performance Tuning and Optimizing ASP.NET Applications*. Apress, 2008, p. 255.
 - [8] R. Frohn, “JSF components optimization,” Bachelor’s thesis, Masaryk University, 2014. [Online]. Available: <https://is.muni.cz/th/ubrc9/?predmet=674645>
 - [9] D. Palacios, “Análisis del rendimiento de librerías de componentes Java Server Faces en el desarrollo de aplicaciones web,” *Revista Digital Novasinerгия*, vol. 1, pp. 54 – 59, 2018. [Online]. Available: http://scielo.senescyt.gob.ec/scielo.php?script=sci_arttext&pid=S2631-26542018000200054&nrm=iso
 - [10] K. Srinivasan. (2008) Advantages and disadvantages – JSF. [Online]. Available: <https://javabeat.net/advantages-and-disadvantages-jsf/>
 - [11] A. Majhi. (2023) Overview of JSF. [Online]. Available: <https://www.codingninjas.com/studio/library/overview-of-jsf>
-

-
- [12] (2024) Technology usage statistics. [Online]. Available: <https://webtechsurvey.com/technology/javaserver-faces>
- [13] A. Kovář, “Integration platform FEL Hub,” Master’s thesis, Czech Technical University in Prague, 2023. [Online]. Available: <https://dspace.cvut.cz/handle/10467/109462>
- [14] D. Flanagan, *JavaScript: The Definitive Guide, 7th Edition*, 5th ed. O’Reilly Media, 2006, p. 497.
- [15] A. S. Gillis. (2022) What is a native app? [Online]. Available: <https://www.techtarget.com/searchsoftwarequality/definition/native-application-native-app>
- [16] B. Hilchenbach. (2023) Definition of progressive web app (PWA). [Online]. Available: <https://techradar.softwareag.com/technology/progressive-web-apps/>
- [17] C. Minnick, *Beginning ReactJS Foundations Building User Interfaces with ReactJS: An Approachable Guide*. John Wiley & Sons, Inc, 2022, ch. 2, p. 11.
- [18] P. Bright. (2012) Microsoft TypeScript: the JavaScript we need, or a solution looking for a problem? [Online]. Available: <https://arstechnica.com/information-technology/2012/10/microsoft-typescript-the-javascript-we-need-or-a-solution-looking-for-a-problem/>
- [19] Facebook. (2022) JSX. [Online]. Available: <https://facebook.github.io/jsx/>
- [20] StackOverflow. (2023) 2023 developer survey. [Online]. Available: <https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe-prof>
- [21] S. Surve. (2021) Why you should use React.js for web development. [Online]. Available: <https://www.freecodecamp.org/news/why-use-react-for-web-development#react-has-broader-community-support-too>
- [22] A. Olawale. (2023) What is React? [Online]. Available: <https://www.freecodecamp.org/news/front-end-javascript-development-react-angular-vue-compared#what-is-react>
- [23] A. Ioffe. (2023) Deep dive into React’s virtual DOM and reconciliation. [Online]. Available: <https://borstch.com/blog/deep-dive-into-reacts-virtual-dom-and-reconciliation>
- [24] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, 1st ed. Pearson, 2021, ch. 8, p. 95.
- [25] S. Buna, *GraphQL in Action*. Manning, 2021, ch. 1, p. 3.
- [26] (2024) Introduction to Apollo Federation. [Online]. Available: <https://www.apollographql.com/docs/react/>
-

-
- [27] C. Walls, *Spring in Action*, 6th ed. Manning, 2021, ch. 1.1, p. 4.
- [28] Netflix. (2024) Getting started. [Online]. Available: <https://netflix.github.io/dgs/getting-started/>
- [29] W. W. Eckerson, “Three tier client/server architecture: Achieving scalability, performance, and efficiency in client server applications,” *Open Information Systems*, vol. 10, no. 1, pp. 3–20, January 1995.
- [30] T. A. Powell, *HTML & CSS: The Complete Reference*, 5th ed. McGraw-Hill, 2010, ch. 6, p. 621.
- [31] P. Laubheimer. (2023) Bottom sheets: Definition and UX guidelines. [Online]. Available: <https://www.nngroup.com/articles/bottom-sheet/>
- [32] P. Davidovic. (2022) Native dual range slider. [Online]. Available: <https://medium.com/@predragdavidovic10/native-dual-range-slider-html-css-javascript-91e778134816>
- [33] A. Montoro. (2023) Drawing a triangle with CSS. [Online]. Available: <https://alvaromontoro.com/blog/67970/drawing-a-triangle-with-css>
- [34] C. Coyier. (2010) Mailto links. [Online]. Available: <https://css-tricks.com/snippets/html/mailto-links/>
- [35] J. S. A.E. Eiben, *Introduction to Evolutionary Computing*, 2nd ed. Springer, 2015, ch. 3.1, p. 25.
- [36] I. Red Hat. (2024) OptaPlanner. [Online]. Available: <https://www.optaplanner.org/>
- [37] J. O’Hara, “Toward a commodity enterprise middleware,” *Queue*, vol. 5, no. 4, p. 48–55, 2007. [Online]. Available: <https://doi.org/10.1145/1255421.1255424>
- [38] G. Roy, *RabbitMQ in Depth*. Manning, 2018, ch. 1.1, p. 4.
- [39] D. S. J. Michael R. Garey, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979, ch. 1.5, p. 13.
- [40] M. Demlová, “Theorem 4.1.8,” Course Materials, p. 32, 2024. [Online]. Available: <https://math.fel.cvut.cz/en/people/demlova/tal/tal-doh.pdf>
- [41] J. Błażewicz, *Scheduling Computer and Manufacturing Processes*, 2nd ed. Springer-Verlag Berlin Heidelberg, 2001, p. 74.
- [42] M. R. Garey and D. S. Johnson, “Strong NP-completeness results: Motivation, examples, and implications,” *J. ACM*, vol. 25, no. 3, p. 499–508, 1978. [Online]. Available: <https://doi.org/10.1145/322077.322090>
-

