Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Design and implementation of an employee shift scheduling system for restaurants

Diploma thesis

Bc. Artem Hurbych

Faculty: Faculty of Electrical Engineering
Study programme: Otevřená informatika
Supervisor: RNDr. Ladislav Serédi

Prague, May 2024

Thesis Supervisor:
RNDr. Ladislav Serédi
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2
160 00 Prague 6
Czech Republic

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Hurbych Artem** Personal ID number: **477691**

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:

Design and implementation of an employee shift scheduling system for restaurants

Master’s thesis title in Czech:

Návrh a implementace systému pro plánování směn v provozovnách stravovacích služeb

Guidelines:

Bibliography / sources:

Models and algorithms for a staff scheduling problem, Alberto Caprara, Michele Monaci & Paolo Toth, dostupné on-line:
https://link.springer.com/article/10.1007/s10107-003-0413-7#Abs1
The general employee scheduling problem. An integration of MS and AI, Fred Glover, Claude McMillan, dostupné on-line:
https://www.sciencedirect.com/science/article/pii/030505488690050X
Gurobi Optimizer Reference Manual, Java API Overview, dostupné on-line:
https://www.gurobi.com/documentation/current/refman/java_api_overview.html

Name and workplace of master’s thesis supervisor:

RNDr. Ladislav Serédi Center for Software Training FEE

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: **24.05.2024**

Assignment valid until: **21.09.2025**

RNDr. Ladislav Serédi 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

Declaration

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 for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, May 2024

.....
Bc. Artem Hurbych

Abstract

This thesis presents the development of an interactive shift scheduling system for restaurants, where the backend is implemented using Java and the Gurobi optimization solver. The primary aim is to enhance operational efficiency through automated shift scheduling, integrating linear programming to handle various constraints such as employee availability, shift overlaps, and cost minimization. The system features a user-friendly frontend using the Angular framework, designed to simplify the management of shift allocations and real-time adjustments. The document details the system's architecture, discusses the chosen algorithms for shift scheduling optimization, and evaluates system performance with a focus on usability and efficiency in real-world scenarios.

Keywords: Employee Shift Scheduling, Java Implementation, Calendar solutions, User-friendly frontend design, Springboot framework, Cost Efficiency, Gurobi solver.

Abstrakt

Tato diplomová práce prezentuje vývoj interaktivního systému pro plánování směn v restauracích, kde backend je implementován pomocí Javy a optimalizačního řešitele Gurobi. Hlavním cílem je zvýšit operační efektivitu prostřednictvím automatizovaného plánování směn, integrující lineární programování pro zvládání složitých omezení, jako jsou dostupnost zaměstnanců, překrývání směn a minimalizace nákladů. Systém obsahuje uživatelsky přívětivé frontendové řešení s využitím frameworku Angular, které je navrženo tak, aby zjednodušilo správu přidělování směn a úprav v reálném čase. Dokument podrobně popisuje architekturu systému, diskutuje o vybraných algoritmech pro optimalizaci plánování směn a hodnotí výkon systému s důrazem na použitelnost a efektivitu v reálných scénářích.

Klíčová slova: Plánování směn zaměstnanců, Implementace v Javě, Řešení kalendáře, Uživatelsky přívětivý design frontendu, Framework Springboot, Efektivita nákladů, Solver Gurobi.

Acknowledgements

I would like to thank RNDr. Ladislav Serédi for mentoring my diploma thesis. His advises were useful and helped me a lot.

In addition, I am grateful to people who were involved in testing the implementation part of the developed application. Their testing results opened new topics to discuss and bring new features to develop.

List of Tables

List of Figures

Contents

Chapter 1

Introduction

The primary ambition of this project, and a pivotal component of my Diploma thesis, revolves around the development of a web application designed explicitly for scheduling employee shifts within the restaurant industry. I intend to leverage cutting-edge web development technologies and contemporary coding methodologies to bring this vision to life. Throughout the implementation phase, I plan to employ the most effective design patterns and best practices that I have acquired over time as a software developer. A comparative analysis of various technologies will also form a crucial segment of this endeavor, offering insights into the effectiveness and efficiency of different web development tools and frameworks in solving real-world problems.

Moreover, an integral objective of both this project and the broader scope of my Diploma Thesis is to engage directly with individuals working in the restaurant sector through a questionnaire. This initiative aims to gather firsthand experiences and insights, thereby ensuring the finished prototype will meet the demands and preferences of its end users.

Furthermore, this project intends to explore and identify the most suitable algorithmic solutions for addressing the specific challenges associated with shift scheduling in the restaurant industry.

In essence, the project represents a confluence of technical innovation, industry-specific problem-solving, and user-centered design philosophy.

Chapter 2

Research

2.1 Existing Solutions Comparison

In this section, I will attempt to compare existing scheduling systems according to various aspects. In the domain of shift scheduling for restaurants, a myriad of solutions offer various features, pricing models, integration capabilities, user experiences, and support levels to meet the diverse needs of the industry. This chapter delves into the comparison of notable platforms, namely Connecteam[1], ADP Workforce Now[2], Homebase[3], Calamari[4], HotSchedules[5], Jolt[6], Sling[7], Zoho People[8], and When I Work[9], to provide a comprehensive overview that can assist restaurant managers and owners in selecting the optimal solution for their operational requirements.
Here you can see an example of what a scheduling system looks like.

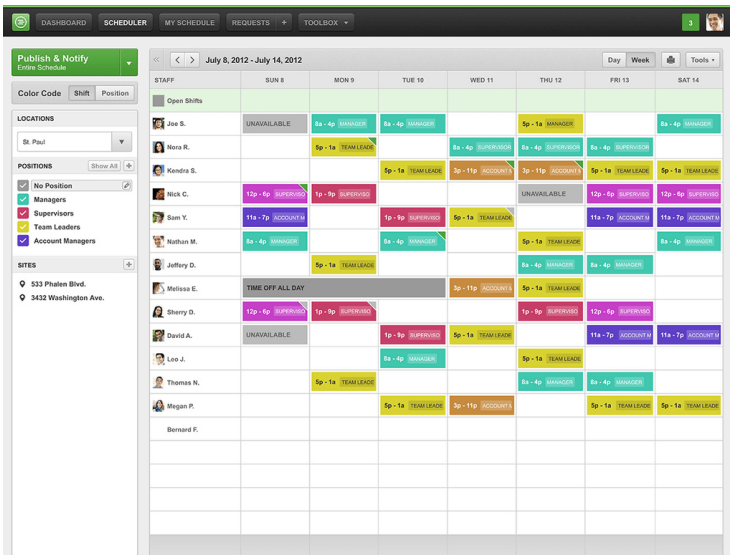

Figure 2.1: Research: Shift scheduling system example

2.1.1 Pricing

Pricing is a critical factor for many businesses, especially those in the competitive restaurant industry. Connecteam offers an affordable starting price of $29/month for 30 users, along with a free plan, catering to small to medium-sized businesses. In contrast, ADP Workforce Now targets a slightly different segment with its pricing starting at $160/month for up to 49 employees, positioning itself as a solution for midsize businesses. Homebase presents a range of plans from a free option to more advanced subscriptions that go up to $80 per month per location, accommodating businesses of various sizes and needs. Calamari and When I Work provide individual user pricing, making them scalable options for growing businesses. In summary, each service offers a unique pricing structure to fit different budgetary requirements and business scales.

2.1.2 User Experience

Ease of use significantly influences the adoption and effectiveness of shift scheduling software. Connecteam and Homebase are renowned for their user-friendly interfaces, simplifying the scheduling process and enhancing usability for managers and employees alike. When I Work is also favored for its intuitive design, promoting efficient shift management and employee engagement through its platform. These aspects are crucial for ensuring high adoption rates and operational efficiency within the restaurant environment.

2.1.3 Features

The core functionality across the reviewed platforms encompasses basic scheduling, shift swapping, and time-off request handling. However, some solutions offer distinctive features that set them apart. For instance, Homebase integrates HR functionalities and financial services like cash-out options, broadening its utility beyond scheduling. Connecteam focuses on comprehensive communication and management tools for workers, including training and payroll integration, which is beneficial for businesses with a mobile workforce. When I Work and Sling extends their offerings to include labor cost forecasting and control, addressing the critical aspect of budget management in labor-intensive industries like hospitality.

Integrations

Integration capability enhances the value of shift scheduling software by allowing businesses to connect various systems, such as payroll and HR, creating a seamless operational flow. ADP Workforce Now boasts a wide range of integrations with platforms like

QuickBooks, Deputy, and Slack, facilitating diverse business processes. Connecteam's integrations include Paychex, Xero, Gusto, and QuickBooks Payroll, among others, demonstrating its versatility in fitting into the existing technology ecosystem of a business.

2.1.4 Support

Customer support is pivotal for ensuring smooth implementation and ongoing use of any software solution. Connecteam, Homebase, and When I Work are highlighted for their exceptional support services, offering resources such as 24/7 assistance, comprehensive online knowledge bases, and quick response times to inquiries. This level of support is indispensable for businesses that rely on their scheduling software for daily operations, ensuring that any issues can be promptly addressed without significant impact on the business.

2.1.5 Existing Solutions research conclusion

In conclusion, the choice of shift scheduling software for restaurants depends on a blend of factors including pricing, user experience, specific features, integration capabilities, and the level of customer support. Each platform reviewed in this chapter brings its unique strengths to the table, catering to different segments of the restaurant industry. Managers and owners are advised to consider their specific operational needs, budget constraints, and desired functionalities when selecting a scheduling solution to ensure that it aligns with their business objectives and enhances operational efficiency.

The next table represents features every analyzed application has:

Feature	**Connecteam**	**ADP**	**Homebase**	**Calamari**
Pricing (Free Plan)	+		+	
Open Source				
Mobile App	+	+	+	+
Customizable Schedules	+	+	+	+
Shift Swapping	+	+	+	
Time Tracking	+	+	+	+
Integration Capabilities	+	+	+	
Advanced Analytics		+		
Labor Cost Management	+	+	+	
User Support	+	+	+	+

Table 2.1: Research: Comparison of Shift Scheduling Applications, Part 1

Feature	Jolt	Sling	Zoho People	When I Work	Hot Schedules
Pricing (Free Plan)		+		+	
Open Source					
Mobile App	+	+	+	+	+
Customizable Schedules	+	+	+	+	+
Shift Swapping	+	+		+	+
Time Tracking	+	+	+	+	+
Integration Capabilities	+	+	+	+	+
Advanced Analytics	+		+		
Labor Cost Management		+		+	+
User Support	+	+	+	+	+

Table 2.2: Research: Comparison of Shift Scheduling Applications, Part 2

In the previous section, I reviewed various shift scheduling systems, highlighting their features. Common to most solutions is the inclusion of a scheduling calendar, a feature my solution will also support. However, a significant distinction of my approach is to be open-source and free which is not present in reviewed solutions. Additionally, while some systems offer payroll integration, my solution will not initially include this to maintain focus on core scheduling functionalities. This decision is informed by the potential complexity and resource requirements of integrating payroll systems, which may not align with the primary needs of target users.

To have a better view of customer needs I did some interviewing of people from food businesses. The results will be provided in the next section.

2.2 Interviewing

As it was discovered shift schedules for the food business are mostly created by managers in messenger chats. It is ineffective and might create time and money losses for those businesses. The proposed solution aims to solve these issues. To collect opinions and ways of creating schedules in different restaurants some workers or ex-workers were interviewed. In the chapters, I will provide their experience.

2.2.1 Respondents

Respondent 1

What kind of food business did you work in?

Coffee shop.

What was your role?

Barista.

What messenger did you use?
WhatsApp.

How many workers who worked on shifts did you have?
4-5 people. 1 person at a time on shift.

When was the schedule created in relation to its scheduled start time?
Now it is 3-5 days before shifts.

How did you organize the schedule in the restaurant where you worked?
Previously it was done just in group chat. Workers were writing when they could work and deciding between themselves. Now we have an Excel table where we put our shifts.

How often did you have problems with your schedule? Why?
Quite often. For example, an Excel document with a schedule was sent to our chat. After some time it was changed without any notification. As a result, I unexpectedly needed to go to work the next day as I decided to double-check the Excel sheet.

Another example was when some workers exchanged their shifts without notifying others. As a result, we had issues with salary calculations.

Respondent 2

What kind of food business did you work in?
Coffee shop.

What was your role?
Barista.

What messenger did you use?
Facebook messanger.

How many workers who worked on shifts did you have?
Usually 2-3 people worked simultaneously.

When was the schedule created in relation to its scheduled start time?

1-2 days before the start of a schedule.

How did you organize the schedule in the restaurant where you worked?

Excel sheet where were placeholders for shifts and you need to fill in what days you can work. Usually, it was two shifts during the day.

How often did you have problems with your schedule? Why?

Sometimes. It was quite often a situation when the manager kept asking you to work on some shift when you could not.

Respondent 3

What kind of food business did you work in?

Bar.

What was your role?

Barman and waiter.

What messenger did you use?

Telegram.

How many workers who worked on shifts did you have?

It was 2-3 waiters and 2 barmen.

When was the schedule created in relation to its scheduled start time?

Always on Sunday for the next week.

How did you organize the schedule in the restaurant where you worked?

Between ourselves in Telegram. After some discussion, we showed the manager our agreed work schedule.

How often did you have problems with your schedule? Why?

Not so often. In cases when we had no workers wanted a shift manager needed to work on it. In the worst cases, business owner took a shift.

Respondent 4

What kind of food business did you work in?

Coffee shop.

What was your role?

Manager and barista.

What messenger did you use?

Facebook messanger.

How many workers who worked on shifts did you have?

Up to 10 people were taking shifts.

When was the schedule created in relation to its scheduled start time?

Week schedules were created in our coffee shop. It took 3-5 days a week before the schedule started to create a timetable. Usually finalized schedules were sent in 1-2 days before its start.

How did you organize the schedule in the restaurant where you worked?

Usually, I did some questionnaires in our chat and then I assigned people to every shift. If more people wanted one place I was trying to choose the one who replied first. Also sometimes we had requirements from our owners to cut our expenses, so I prioritized people with smaller wages.

How often did you have problems with your schedule? Why?

Sometimes. It is quite difficult to create a big schedule only from a chat questionnaire. Sometimes people change their response and I don't notice it. Also, it is time-consuming to create the cheapest schedule as I need to check everyone's salary and choose shifts where they fit the best.

Respondent 5

What kind of food business did you work in?

Italian restaurant.

What was your role?

Scheduler and accounter.

What messenger did you use?
WhatsApp.

How many workers who worked on shifts did you have?
Around 15 people working in shifts.

When was the schedule created in relation to its scheduled start time?
We usually prepare our schedules two weeks in advance. I aimed to finalize and circulate the schedules at least 5 days before the start of the week.

How did you organize the schedule in the restaurant where you worked?
I collected availability through a WhatsApp group every month. Based on the responses, I filled in the shifts, giving priority to those who responded earliest. If a shift was particularly popular, I had to consider seniority and skill level to decide. During busy seasons, I also factored in the need to balance hours to manage labor costs effectively.

How often did you have problems with your schedule? Why?
Frequently. Balancing everyone's availability, preferences, and the restaurant's needs was challenging. Often, workers would update their availability last minute, leading to conflicts and the need for quick rearrangements. Also, creating a cost-effective schedule required constant adjustment to ensure efficient staffing without overspending.

2.2.2 Interviewing conclusion

After the interviews, it is evident that despite the availability of many good solutions, they are often not used in small and medium businesses. I assume that these businesses attempt to save money by using free tools such as messengers. This underscores another reason to develop a new and free solution.

The interviews conducted provided some insights into the practical needs and challenges faced by workers and managers in the restaurant industry. Based on this research and the most valuable features from existing solutions, I will design an application that offers shift scheduling functionalities tailored to the specific needs identified. This includes a user-friendly interface for managing shift preferences and an automated system for creating schedules. Also, there should be an option to create a cost-effective schedule.

Chapter 3

Analysis

3.1 Requirements

3.1.1 Functional requirements

1. **User Authentication and Authorization:**

   - The system should support secure user authentication for both employees and administrators.
   - Different levels of access should be granted based on user roles, allowing employees to input their availability and view schedules, while administrators have access to the complete scheduling functionality.

2. **Employee Availability Management:**

   - Employees should be able to input their availability.
   - The system should validate and handle conflicts arising from overlapping availability requests.

3. **Shift Creation and Assignment:**

   - Administrators should be able to create and define shifts based on business needs.
   - The system should provide functionality to assign employees to shifts, considering their availability.
   - Shift assignments should be communicated to employees promptly.

4. **Schedule Viewing and Notifications:**

- Employees should have access to their schedules, including upcoming shifts and any changes made.
- Notifications should be sent to employees and administrators for new possible shifts.

5. **Schedule Optimization:**

   - The system should provide optimization algorithms to automate the shift scheduling process, taking into account employee queues.
   - The system should provide optimization algorithms to automate the shift scheduling process, taking into account employee salaries.

6. **Data Storage:**

   - The application should securely store all scheduling data in a database.

3.1.2 Non-Functional requirements

1. **Security:**

   - The system should comply with industry standards for data security, including encryption of sensitive information and secure transmission of data.
   - Access controls should be implemented to prevent unauthorized access to scheduling data.

2. **Scalability:**

   - The architecture should be scalable to accommodate an increasing number of users and shifts without compromising performance.
   - The system should handle a large volume of simultaneous user interactions during peak times.

3. **Reliability:**

   - The system should have a high level of availability, with minimal downtime for maintenance.
   - Data integrity should be maintained, and the system should be resilient to potential failures.

4. **Usability:**

- The user interface should be intuitive and user-friendly for both employees and administrators.
- The system should provide clear instructions and error messages to assist users in navigating and using the application effectively.

5. **Compatibility:**

   - The application should be compatible with commonly used web browsers and devices.
   - Responsive design principles should be applied to ensure usability across different screen sizes.

6. **Performance:**

   - The system should respond to user interactions promptly, providing real-time updates when applicable.
   - The backend algorithms for shift scheduling should execute efficiently, even with a large dataset of employees and shifts.

3.2 Use case diagram

After identifying the functional requirements of the system, a use case diagram was created that captures the activities that actors can perform within the system according to their roles.

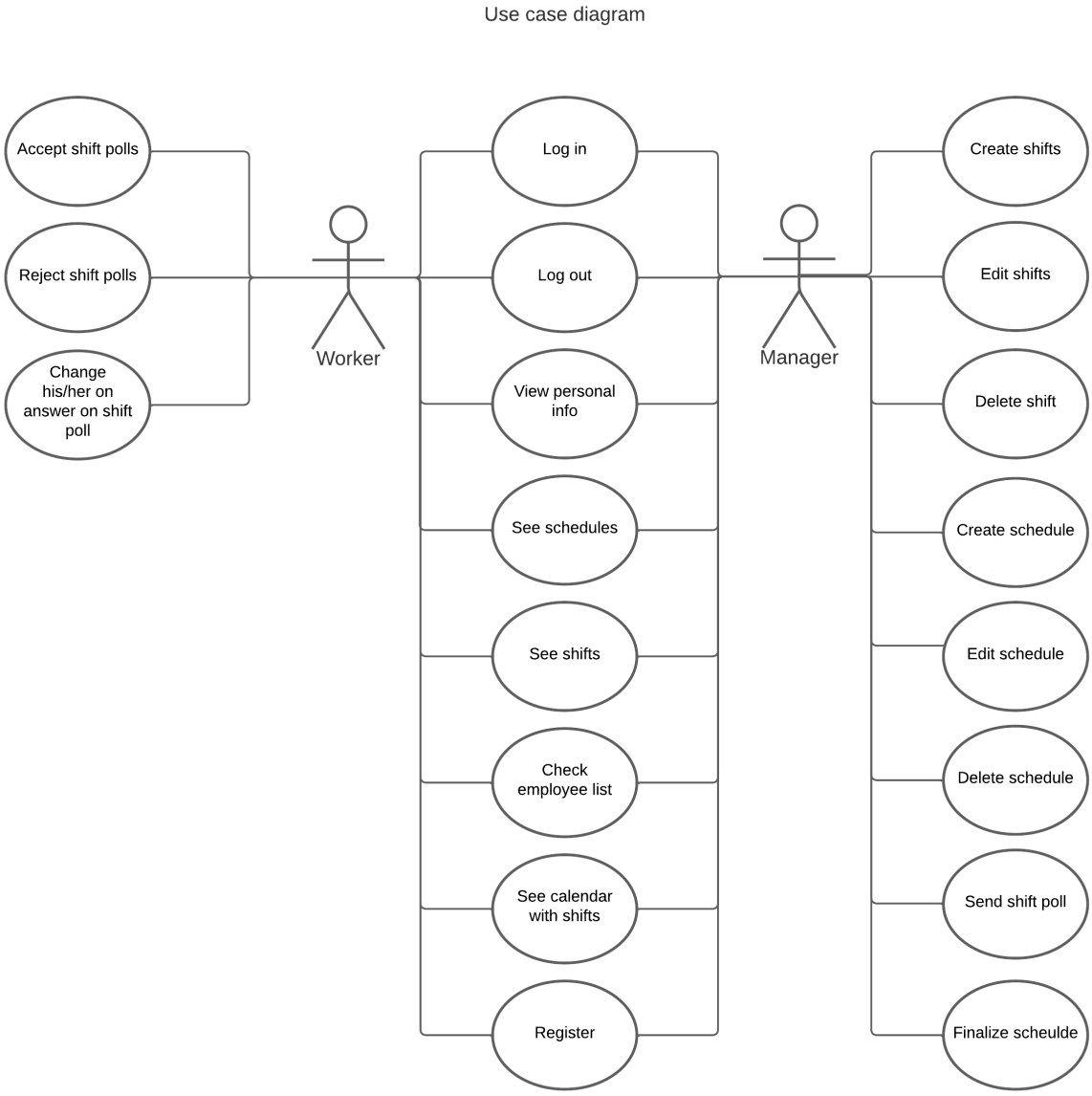

Figure 3.1: Analysis: Use case diagram

Chapter 4

User interface design

In this chapter, I explore the concept of mental models and their role in shaping user interface design. Additionally, I will outline the usability principles as proposed by Steve Krug[11]. Utilizing these mental models and principles, I will develop a wireframe for a web application and map out the overall interaction of users with the interface.

4.1 Mental model

Mental models are the user-generated assumptions about the operation of a specific product, like a web application. These assumptions are based on prior experiences with analogous products. Factors such as education, age, and cultural background also play a role in the formation of these models, leading each user to develop a unique perspective.

The design of a product that aligns with user knowledge and anticipations of similar products facilitates effortless interactions, quick familiarity, and enhanced usability. Hence, the design of the user interface will be aligned with the requirements identified in Chapter 3 and the interview results from Section 2.2.

4.2 Three Laws of Usability

Steve Krug's definition of usability[11] is based on the ease with which a person can comprehend and use a product to achieve desired outcomes. He articulates three key usability principles necessary for optimal user experience:

1. **Don't Make Me Think** This pivotal principle ensures that the design is intuitive. Users should instantly understand the purpose and usage of a webpage without needing to ponder. Questions that provoke user deliberation distract from the intended interaction.

2. **Users like mindless choices** The number of clicks involved is irrelevant as long as each one is intuitive and clear.

3. **Omit unnecessary words** This principle advises the reduction of superfluous text to prevent user distraction, thereby highlighting essential content and reducing the need for excessive scrolling.

4.3 Wireframe design

The design process begins with creating wireframes, which are visual representations of the layout of application pages. I opt for wireframes with low to medium fidelity to ensure that the designs are detailed enough for visualization without being overly complex at the outset, allowing for easy adaptations based on user feedback.[12]

4.3.1 Balsamiq

For creating low-fidelity wireframes of the user interface, I chose Balsamiq Wireframes. This tool is selected for its capability to produce simple yet detailed designs, encouraging focus on structural and content elements over aesthetic details, which are to be refined later in the design process.[13]

4.3.2 Interaction map

In the screenshots below you can see the interaction map proposal for Shift Scheduler application.

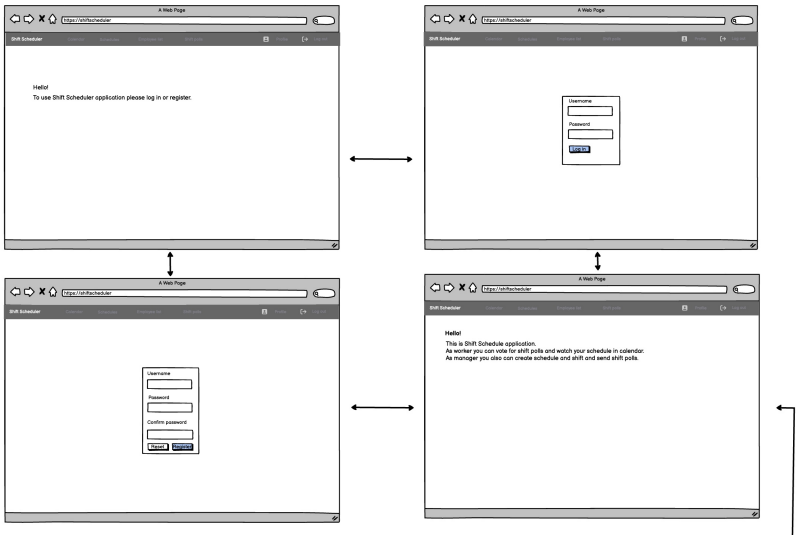

Figure 4.1: User interface design, Part 1

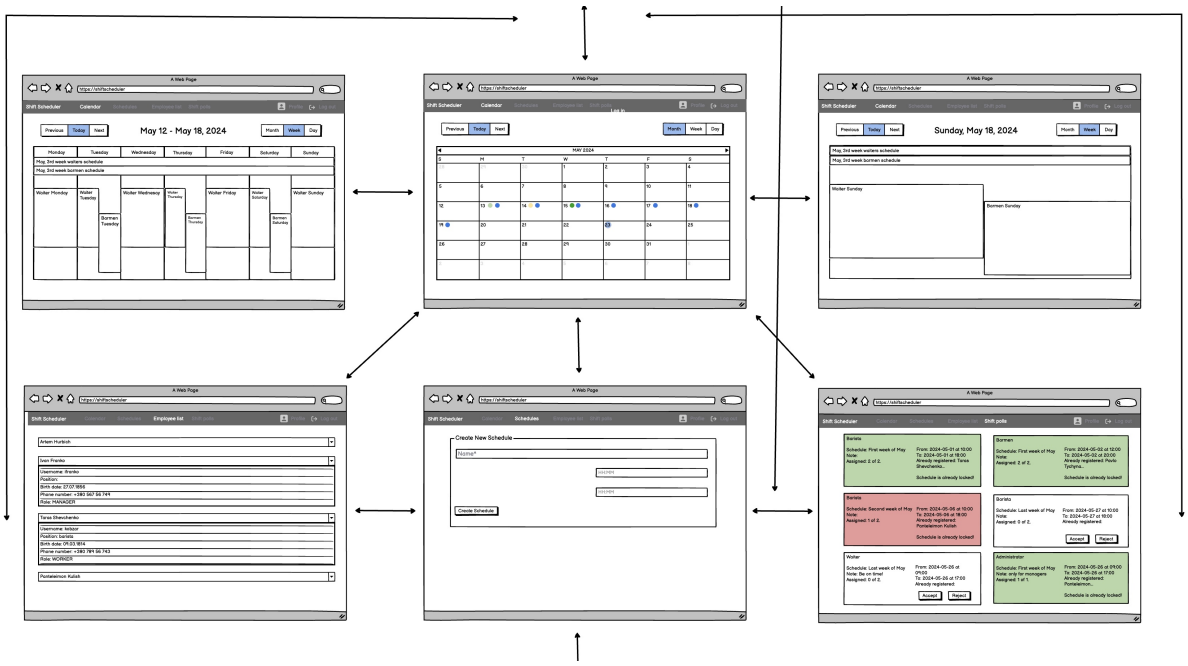

Figure 4.2: User interface design, Part 2

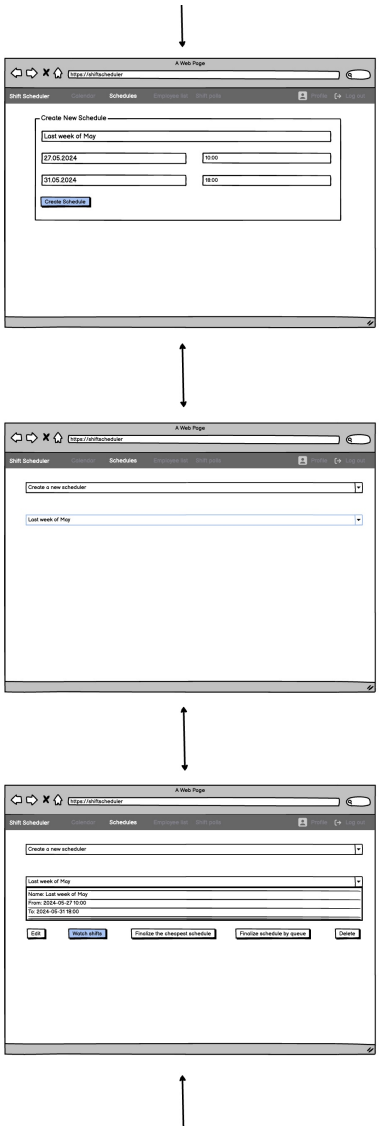

Figure 4.3: User interface design, Part 3

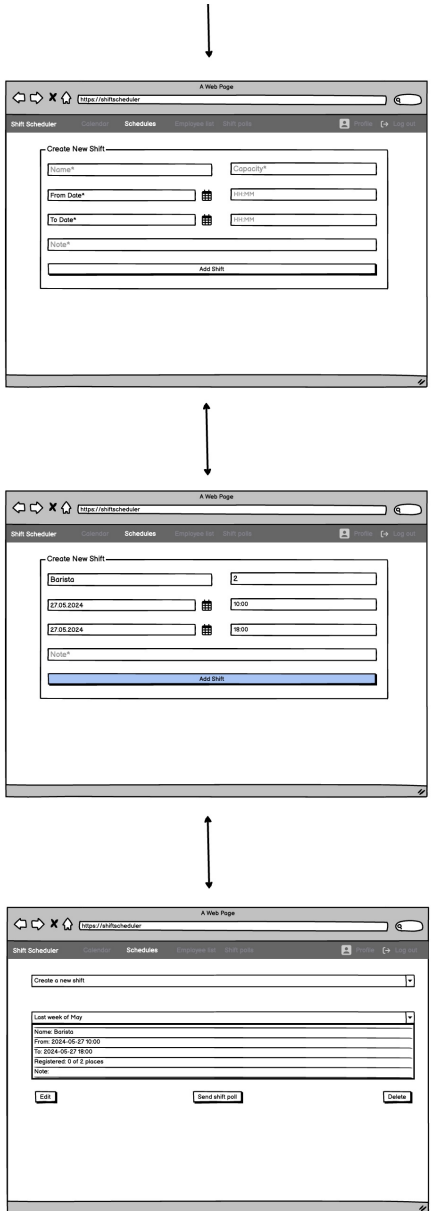

Figure 4.4: User interface design, Part 4

Chapter 5

Solution proposal

5.1 Database

5.1.1 PostgreSQL

PostgreSQL[14], often referred to as Postgres, is a powerful and open-source RDBMS. It has been in development for over three decades and has gained a reputation for its robustness, extensibility, and adherence to SQL standards. Here are some key aspects of PostgreSQL:

Data Model: PostgreSQL follows a relational data model, which means it stores data in tables with rows and columns. Each table has a well-defined schema, and data integrity is maintained through constraints and relationships.

Schema: Postgres uses a fixed schema approach, requiring a user to define the structure of his data by specifying tables and their columns in advance. Any changes to the schema often involve performing migrations to ensure data consistency.

Query Language: PostgreSQL uses SQL, which is a standardized language for querying and manipulating relational data. SQL is well-known and widely used, making it easy to find resources and expertise for working with PostgreSQL.

ACID Compliance: PostgreSQL is known for its strict adherence to ACID properties, ensuring data consistency, reliability, and strong transaction support. It's suitable for applications that require robust data integrity.

Scalability: PostgreSQL provides options for scaling through replication and partitioning. While it can handle large datasets and high workloads, scaling can be more complex and often requires manual configuration.

Use Cases: PostgreSQL is commonly used in applications that demand complex querying, data consistency, and structured data. Typical use cases include e-commerce platforms, financial systems, content management systems, and data warehousing.

Ecosystem: PostgreSQL has a rich ecosystem of extensions and libraries, enabling developers to extend its capabilities. It offers support for various programming languages and is available on various operating systems.

5.1.2 MongoDB

MongoDB[15] is a popular open-source NoSQL database that falls under the document-oriented database category. It's designed for flexibility and scalability, making it suitable for applications with dynamic data requirements. Here's an overview of MongoDB:

Data Model: MongoDB uses a document data model. It stores data in flexible, JSON-like documents known as BSON. Each document can have a different structure within the same collection, allowing for semi-structured or unstructured data storage.

Schema: MongoDB employs a dynamic schema approach. This means you can insert documents without predefining their structure, and you can add fields to documents on the fly. This flexibility is well-suited for applications with evolving data needs.

Query Language: MongoDB uses a JSON-like query language for retrieving documents from collections. This language is more aligned with the document-based structure of the data it stores.

ACID Compliance: MongoDB can be configured for ACID compliance at the document level using multi-document transactions in recent versions. However, it is often used in scenarios that can tolerate eventual consistency due to its distributed nature.

Scalability: MongoDB is built for horizontal scalability and can easily distribute data across multiple servers using sharding. This makes it well-suited for applications with large datasets and high write and read throughput requirements.

Use Cases: MongoDB is commonly used in applications where data structures are dynamic or semi-structured, such as content management systems, catalogs, real-time analytics, and Internet of Things data storage.

Ecosystem: MongoDB has a robust ecosystem that includes services like MongoDB Atlas (a cloud-based database service), a variety of drivers for different programming languages, and a thriving community that contributes to its development.

5.1.3 Database choice

MongoDB is more interesting from a technological perspective. It offers a unique and flexible approach to data management. Its document-oriented, NoSQL approach allows for a dynamic and adaptable data model, making it well-suited for applications with ever-changing data requirements. The ability to store and query data in a JSON-like format, without the constraints of a fixed schema, can be particularly appealing for developers

who explore new and unconventional approaches to data management.

MongoDB's focus on horizontal scalability and its support for distributed, high-throughput workloads also make it an interesting choice for applications that need to handle large volumes of data and traffic.

So I choose it as DBMS for our project.

5.2 Backend

I choose Java and Springboot for the backend of our application. There are some advantages of these technologies which affected our decision:

Robust and Mature Ecosystem: Java is a widely used, mature, and well-established programming language with a vast ecosystem of libraries and frameworks. Spring Boot, built on top of Java, is a robust and highly popular framework for building web applications. The mature ecosystem provides a wide range of tools and resources that can be beneficial for development.

Reliability and Stability: Java is known for its stability and reliability. This is crucial for shift scheduling applications, as any downtime or data inconsistencies can lead to significant disruptions. Java's strong type system and error-checking mechanisms help in producing dependable code.

Strong Database Support: Java has strong support for working with databases. Spring Boot simplifies database access through the use of the Spring Data JPA and Hibernate, which are powerful tools for handling data. You can connect to a variety of databases, such as PostgreSQL or MongoDB, making it easy to manage shift schedules and user data.

Cross-Platform Compatibility: Java is platform-independent, which means you can develop Shift Scheduling application to run on different operating systems and platforms. This can be crucial for ensuring that our application is accessible to a wide audience.

Scalability: Java and Spring Boot allow to building of scalable applications. Shift scheduling systems often need to handle a varying number of users and data over time. With proper design and architecture, a developer can easily scale the application to accommodate growth.

Security: Shift scheduling applications frequently deal with sensitive data, such as employee information and schedules. Java and Spring Boot provide robust security mechanisms, including authentication and authorization features, to protect sensitive data from unauthorized access.

Community Support: Java and Spring Boot have large and active communities.

This means it is possible to find plenty of resources, tutorials, and forums for help and support when building your application. It's easier to get answers to your questions and find solutions to any challenges you may encounter.

RESTful API Development: Spring Boot excels in building RESTful APIs, which can be useful for creating a modern and responsive shift scheduling webpage. It is easy to expose data through RESTful endpoints for consumption by web and mobile clients.

Integration Capabilities: Java and Spring Boot support integration with various third-party systems and services. This is important if Shift Scheduling application needs to interact with other tools, such as payroll systems, time-tracking software, or HR databases.

Maintainability and Extensibility: Java's strong object-oriented principles and design patterns, along with Spring Boot's modular architecture, make it easier to maintain and extend our application as business requirements change or new features are added.

In summary, Java and Spring Boot are a strong choice for building a shift scheduling webpage with a database due to their reliability, database support, scalability, security features, and the extensive resources available within their communities. The combination of these technologies can help you create a robust and maintainable solution for your shift scheduling needs.

To correspond to new technologies I chose Java 21 LTS[16] and Springboot 3[17].

5.3 Frontend

This section will describe the most popular frontend frameworks and find the most suitable for my solution.

5.3.1 React

React[18] is a JavaScript library developed and maintained by Facebook. It is widely used for building user interfaces, particularly for single-page applications where the content is dynamically updated without requiring a full page reload. React follows a component-based architecture, allowing developers to create reusable and modular components that can be composed to build complex UIs. React's virtual DOM efficiently updates only the parts of the actual DOM that have changed, improving performance. It has a large and active community, making it easy to find resources and support.

5.3.2 Angular

Angular[19] is a comprehensive front-end framework developed and maintained by Google. It provides a robust and opinionated structure for building dynamic web applications. Angular uses TypeScript, a superset of JavaScript, which adds static typing and other features to enhance code quality and maintainability. Angular follows a component-based architecture similar to React but includes a broader set of tools and features out of the box. It includes a powerful dependency injection system, and a declarative template syntax, and supports two-way data binding, making it suitable for large-scale applications with complex requirements. Angular has a steep learning curve but offers a full-fledged solution for enterprise-level projects.

5.3.3 Vue.js

Vue.js[20] is a progressive JavaScript framework for building user interfaces, designed to be incrementally adoptable. Developed by Evan You, Vue is known for its simplicity and flexibility. Vue's core library focuses on the view layer only, but it can be easily integrated with other libraries or existing projects. Vue provides a reactive data-binding system and a component-based architecture similar to React and Angular. It has a gentle learning curve, making it accessible for developers with varying levels of experience. Vue's simplicity and ease of integration make it a popular choice for smaller projects and startups.

5.3.4 Frontend conclusion

While React, Angular, and Vue each have their strengths, Angular stands out as the most comprehensive and feature-rich framework. Angular provides a complete solution with a powerful set of tools, a well-defined structure, and a strong emphasis on scalability. Its use of TypeScript enhances code quality and maintainability, and the built-in features like dependency injection and two-way data binding simplify development. So I decided to use Angular for frontend development.

5.4 Problem and Algorithm

The Linear Programming approach can be used to solve shift scheduling problems efficiently. In this case, I want to create a schedule for different employees to get to specific constraints and objectives. Gurobi[21] is a powerful optimization solver that provides high-performance mathematical programming solvers for linear programming, mixed-integer programming, quadratic programming, and other optimization problems.

Gurobi supports various programming languages, including Java. Gurobi provides a Java API that allows you to interact with the Gurobi optimization engine from your Java applications.
To introduce readers to Gurobi and Linear Programming here is an example of a solution for the basic Shift Scheduling problem in Java with the Gurobi library.

5.4.1 Basic algorithm

To introduce readers to Gurobi and Linear Programming here is an example of a solution for the basic Shift Scheduling problem in Java with the Gurobi library. The mathematical model for solving the Shift Scheduling Problem is composed of the following elements:

Variables

- Define x_j as a binary variable where $x_j = 1$ if shift j is selected and $x_j = 0$ otherwise.

Parameters

- c_j - Represents the cost associated with each shift j.
- S_j - The set of time periods that shift j covers.
- T - The complete set of time periods that require coverage.

Constraints Ensure each time period is covered by at least one shift:

$$\sum_{j:i \in S_j} x_j \geq 1, \quad \forall i \in T$$

Objective Function The objective is to minimize the total cost of employed shifts:

$$\text{Minimize} \quad \sum_j c_j x_j$$

Below you can see part of the implementation for the basic problem.

```
// Create variables for each shift with associated costs
GRBVar[] x = new GRBVar[shifts.length];
for (int j = 0; j < shifts.length; j++) {
    x[j] = model.addVar(0, 1, costs[j], GRB.BINARY,
    "Shift_" + (j + 1));
}

// Each time period must be covered by at least one shift
for (int i = 0; i < numPeriods; i++) {
    GRBLinExpr expr = new GRBLinExpr();
    for (int j = 0; j < shifts.length; j++) {
        for (int k = 0; k < shifts[j].length; k++) {
            if (shifts[j][k] == i) {
                expr.addTerm(1, x[j]);
                break;
            }
        }
    }
    model.addConstr(expr,
        GRB.GREATER_EQUAL, 1, "Coverage_" + i);
}

// Objective: minimize the total cost of the shifts used
model.setObjective(model.getObjective(), GRB.MINIMIZE);

// Optimize the model
model.optimize();
}
```

Table 5.1: Solution proposal: Java solver for Basic Scheduling Problem

5.4.2 Modifications from the basic Shift Scheduling Problem

The basic Shift Scheduling Problem primarily focuses on minimizing the number of staff required to cover all shifts, with each shift covering a specific set of hours without considering individual worker preferences or costs. In contrast, our implementation incorporates several key modifications to address real-world complexities specific to a restaurant environment, which include handling costs, worker availability, and shift overlaps. These modifications enhance the model's applicability and effectiveness in practical settings.

Introduction to Modifications The traditional basic model is extended to include additional constraints and cost considerations, adapting the problem to a more realistic scenario where both worker schedules and operational costs are critical.

Incorporation of Costs Unlike the basic problem which does not consider the cost associated with each shift, our implementation integrates hourly wages into the optimization process. The objective function is modified to minimize the total labor cost, calculated as:

$$\text{Objective: Minimize} \quad \sum_{i,j} \text{hourlyWages}_i \cdot \text{shiftHours}_j \cdot x_{ij},$$

where x_{ij} is a binary decision variable indicating whether worker i is assigned to shift j.

Worker Availability Constraints To accommodate individual worker availability, a binary matrix $\text{workerAvailability}_{ij}$ is used, where each element is 1 if worker i is available for shift j and 0 otherwise. This matrix is fundamental in defining the feasible set of decision variables, ensuring the model only assigns workers to shifts they can actually work.

Shift Capacity Constraints The model ensures that each shift has exactly the number of workers required, using the constraint:

$$\sum_i x_{ij} = \text{shiftCapacities}_j \quad \forall j,$$

which enforces that the total number of workers assigned to each shift matches its required capacity exactly.

Non-Overlap Constraints for Workers Our model adds constraints to prevent a worker from being assigned to overlapping shifts. If shifts j and k overlap in time, then:

$$x_{ij} + x_{ik} \leq 1 \quad \forall i,$$

ensuring no worker is scheduled for more than one shift at any time.

Modified problem implementation

Below you can see part of the Java implantation of the solution introduced before.

```
// Decision variables
GRBVar[][] x = new GRBVar[numWorkers][numShifts];
for (int i = 0; i < numWorkers; i++) {
    for (int j = 0; j < numShifts; j++) {
        if (workerAvailability[i][j]) {
            x[i][j] = model.addVar(0, 1,
            hourlyWages[i] * shiftHours[j],
                GRB.BINARY, "x_" + i + "_" + j);
        }
    }
}

// Objective: Minimize total cost
GRBLinExpr objective = new GRBLinExpr();
for (int i = 0; i < numWorkers; i++) {
    for (int j = 0; j < numShifts; j++) {
        if (workerAvailability[i][j]) {
            objective.addTerm(hourlyWages[i] * shiftHours[j], x[i][j]);
        }
    }
}
model.setObjective(objective, GRB.MINIMIZE);
```

Table 5.2: Solution proposal: Java solver of our Shift Scheduling problem, Part 1

```
// Constraints: Shift capacity - Ensure exactly the capacity
for (int j = 0; j < numShifts; j++) {
    GRBLinExpr capacityConstraint = new GRBLinExpr();
    for (int i = 0; i < numWorkers; i++) {
        if (workerAvailability[i][j]) {
            capacityConstraint.addTerm(1, x[i][j]);
        }
    }
    model.addConstr(capacityConstraint, GRB.EQUAL,
        shiftCapacities[j], "cap_" + j);
}

// Constraint: No overlapping shifts for any worker
for (int i = 0; i < numWorkers; i++) {
    for (int j = 0; j < numShifts; j++) {
        for (int k = j + 1; k < numShifts; k++) {
            if ((shiftStarts[j] < shiftEnds[k]
                && shiftStarts[k] < shiftEnds[j])
                && workerAvailability[i][j]
                && workerAvailability[i][k]) {
                    GRBLinExpr overlapExpr = new GRBLinExpr();
                    overlapExpr.addTerm(1.0, x[i][j]);
                    overlapExpr.addTerm(1.0, x[i][k]);
                    model.addConstr(overlapExpr, GRB.LESS_EQUAL,
                    1, "no_overlap_" + i + "_" + j + "_" + k);
            }
        }
    }
}
```

Table 5.3: Solution proposal: Java solver of our Shift Scheduling problem, Part 2

5.4.3 Conclusion

These modifications from the basic Shift Scheduling Problem not only tailor the model to specific operational needs but also enhance its utility by addressing cost efficiency and employee work-life balance.

5.5 Diagrams

5.5.1 Architecture diagram

The architectural diagram is based on the solution described in the Solution Technologies section.

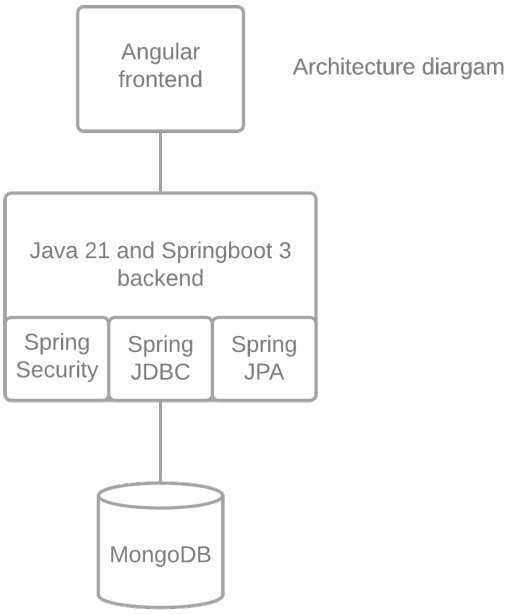

Figure 5.1: Solution proposal: Architecture diagram

Chapter 6

Implementation

In this chapter, I will describe the process of application implementation including personal challenges, interesting technologies used, application workflow, and others.

6.1 Personal experience

The whole application including frontend and backend I wrote on MacOS created some challenges. For instance, I need to install MongoDB on my personal laptop for further connection to the backend. To install the database I needed to update MacOS and due to some bug update was not possible. Fortunately, this issue was fixed by Apple in some time.

Also, I did not have much experience with frontend as my previous projects were mostly backend, so I needed to use some new techniques to use Angular. In the end, I gained the needed experience and knowledge from available sources and got some advice from my colleagues. So as a result minimalistic but good-looking and functional frontend was created.

6.2 Design patterns

To implement both frontend and backend easy-to-read and corresponding SOLID principles[22] I used different design patterns. It is important to follow such principles as they provide a robust foundation for designing maintainable and scalable software systems. By adhering to these principles, developers can ensure their applications are easier to understand, extend, and modify, thereby enhancing code quality and reducing the risk of introducing bugs. This results in a more flexible architecture that can adapt to changing requirements and facilitates collaboration among development teams. Usage of some of them will be described in this section.

6.2.1 DTOs

To transfer data in frontend and backend DTOs were used for all possible information. In the screenshot below, you can see the usage of Shift DTO.

```
export interface Shift {
  id: string;
  name: string;
  fromDate: string;
  fromTime: string;
  toDate: string;
  toTime: string;
  note: string;
  capacity: string;
  registered: Employee[];
  scheduleId: string;
  isEditing: boolean;
}
```

Table 6.1: Implementation: Angular Class Definition for Shift

And next one shows the use of Shift DTO in the backend.

```
@AllArgsConstructor
@Builder
@Data
public class ShiftDto {
    String id;
    String name;
    LocalDateTime from;
    LocalDateTime to;
    List<AccountDto> registered;
    Integer capacity;
    String note;
    String scheduleId;
}
```

Table 6.2: Implementation: Java Class Definition for ShiftDto

6.2.2 Interceptor

On both application parts Interceptor design pattern was used to separate authentication with JWT token from business logic. It allows to check if a token is present for every

incoming request on the backend side and to set a token for every outgoing from a frontend token.

Here is an example from the application code for the frontend in Angular.

```
const TOKEN_HEADER_KEY = 'Authorization';

@Injectable()
export class HttpRequestInterceptor implements HttpInterceptor {
  constructor(private storageService: StorageService) {
  }

  intercept(req: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {
    let authReq = req;
    const token = this.storageService.getUser().token;
    if (token != null) {
      authReq = this.addTokenHeader(req, token);
    }
    return next.handle(authReq);
  }

  private addTokenHeader(request: HttpRequest<any>, token: string) {
    return request.clone({headers: request.headers.set(TOKEN_HEADER_KEY,
    'Bearer ' + token)});
  }
}

export const httpInterceptorProviders = [
  {
      provide: HTTP_INTERCEPTORS,
      useClass: HttpRequestInterceptor,
      multi: true
  },
];
```

Table 6.3: Implementation: Angular Class Definition for HttpRequestInterceptor

And the next one is implementation in Java with Springboot on the backend side.

```
protected void doFilterInternal(@NonNull HttpServletRequest request,
                                @NonNull HttpServletResponse response,
                                @NonNull FilterChain filterChain)
        throws ServletException, IOException {
    final String authHeader = request.getHeader("Authorization");
    final String jwt;
    final String userEmail;
    if (StringUtils.isEmpty(authHeader) ||
        !StringUtils.startsWithIgnoreCase(authHeader, "Bearer ")) {
        filterChain.doFilter(request, response);
        return;
    }
    jwt = authHeader.substring(7);
    userEmail = jwtService.extractUserName(jwt);
    if (!StringUtils.isEmpty(userEmail)
            && SecurityContextHolder.getContext().getAuthentication()
                == null) {
        UserDetails userDetails = accountService.userDetailsService()
                .loadUserByUsername(userEmail);
        if (jwtService.isTokenValid(jwt, userDetails)) {
            SecurityContext context =
                SecurityContextHolder.createEmptyContext();
            UsernamePasswordAuthenticationToken authToken =
                new UsernamePasswordAuthenticationToken(
                    userDetails, null, userDetails.getAuthorities());
            authToken.setDetails(new WebAuthenticationDetailsSource()
                .buildDetails(request));
            context.setAuthentication(authToken);
            SecurityContextHolder.setContext(context);
        }
    }
    filterChain.doFilter(request, response);
}
```

Table 6.4: Implementation: Java Class Definition for JwtAuthenticationFilter

6.2.3 Builder

Also to convert DTO objects to objects used for the ORM Builder pattern was useful. In the screenshot below you can see the usage of this pattern for Schedule business entity.

```
public ScheduleDto schedule(Schedule schedule){
    return ScheduleDto.builder()
            .id(schedule.getId())
            .name(schedule.getName())
            .from(schedule.getFrom())
            .to(schedule.getTo())
            .restaurant(restaurant(schedule.getRestaurant()))
            .isLocked(schedule.getIsLocked())
            .build();
}
```

Table 6.5: Implementation: Java Class Definition for ScheduleDto

6.2.4 Springboot design patterns

Also just by using Springboot, we involve some design patterns. For example Singleton as default bean creation strategy[24] or Dependency Injection for autowiring of components to other components[25].

6.3 Class diagram

To illustrate the code structure, a class diagram is provided below.

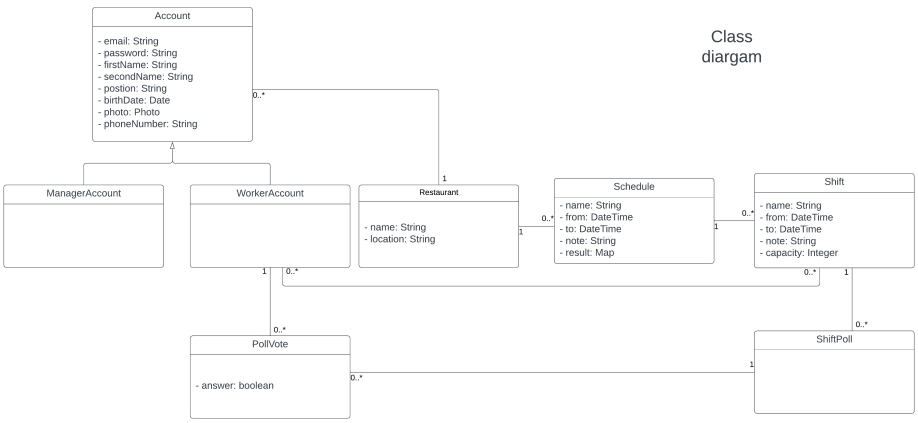

Figure 6.1: Implementation: Class diagram

6.4 Application workflow

6.4.1 Default pages

The first step for each user is to register in the application. After registration with default information, they can log in. After logging in user can see two general pages. One explains the default features and behavior of the application. The second one shows profile information.

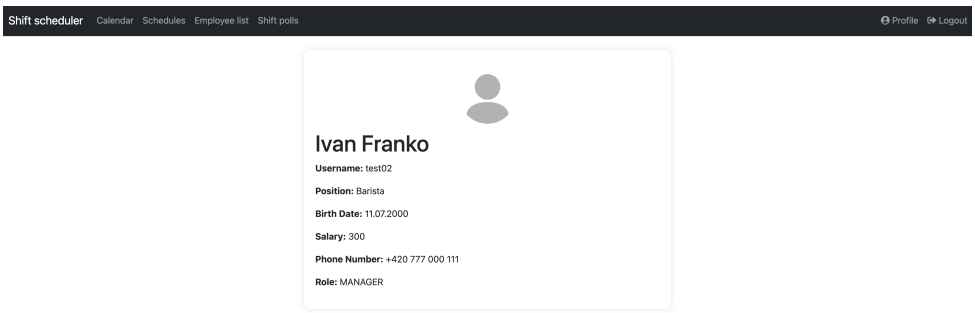

Figure 6.2: Implementation: Profile page

Also, the user can access the page with employees of his restaurant information. Here is an example of such page.

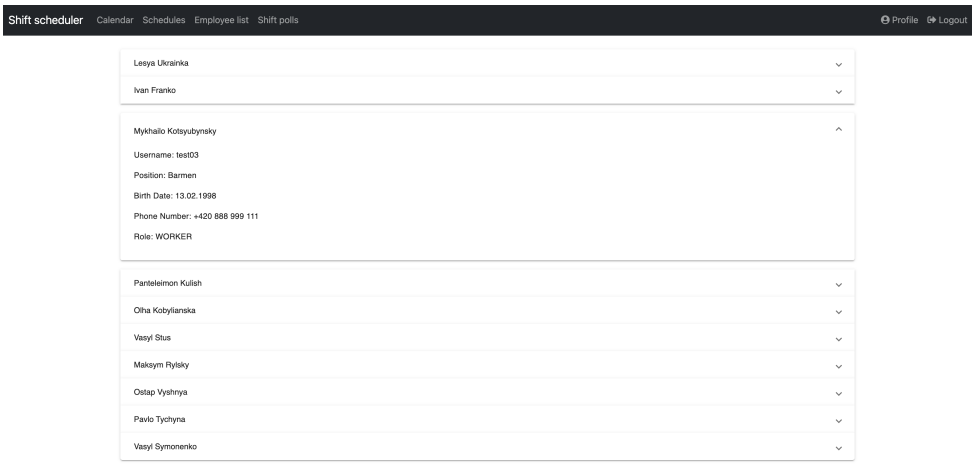

Figure 6.3: Implementation: Employee list page

6.4.2 Schedules and shifts

Users can access Schedule page where schedule viewing, creation, and editing are possible. Creation and editing are available only for users with MANAGER role.

Figure 6.4: Implementation: Schedule page, creation

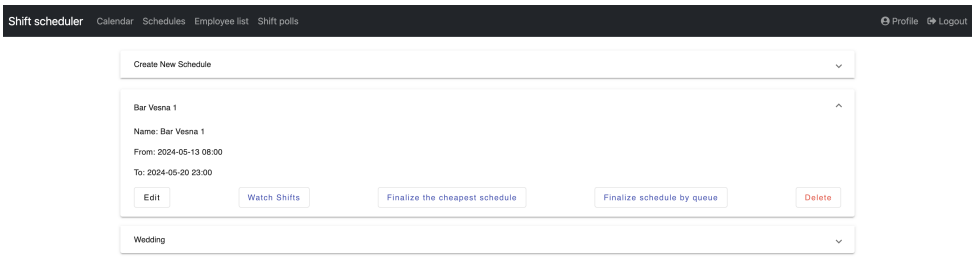

Figure 6.5: Implementation: Schedule page, viewing

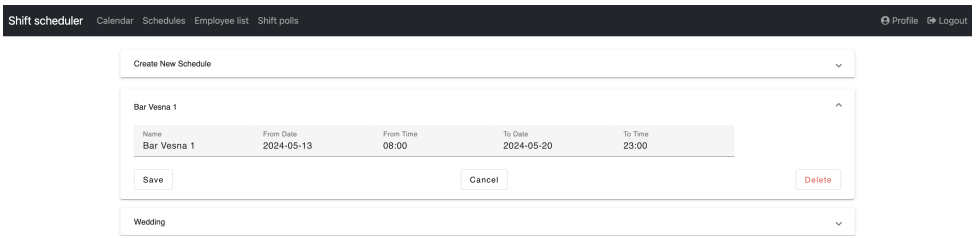

Figure 6.6: Implementation: Schedule page, editing

Also, there are some buttons. All of them except "Watch shifts" are available only for managers. By clicking the "Finalize" button we can create our schedule. It is done when shifts for this schedule are created and votes for shift polls are gotten. From interviewing I got the request to have two options for finalizing. One is by queue and the second is to create the cheapest cost when a business has a small amount of money. Finalizing by cost is possible only when we have at least the same amount as capacity voted for all shifts in the calculated schedule. Otherwise "by queue" option will be used even if the "by cost" button is selected. A message with information will appear after using the button. By clicking on the "Watch shifts" button we go to the shifts page.

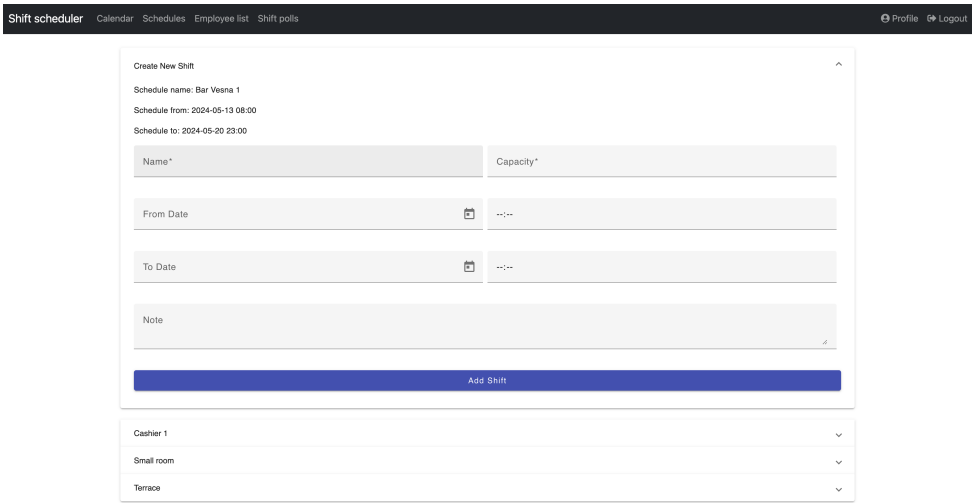

Figure 6.7: Implementation: Shift page, creation

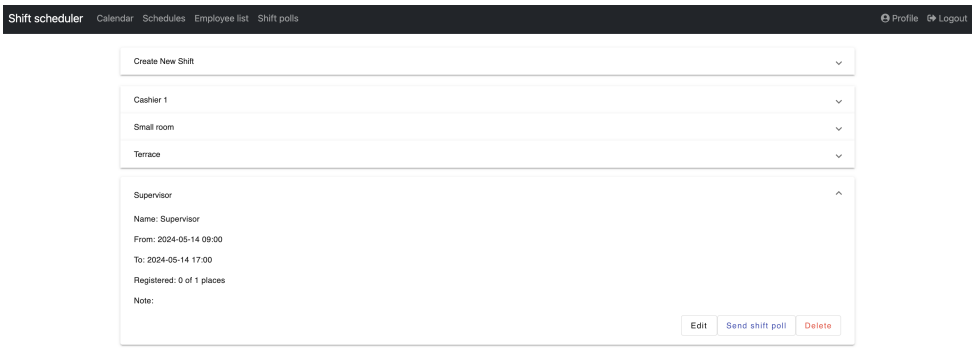

Figure 6.8: Implementation: Shift page, shift viewing

As for schedules for shifts, it is possible to edit and delete them. Also "Send shift poll" button is present. All these features are available only for managers. By clicking on the "Send shift poll" button an invitation for every employee of this restaurant will be sent.

6.4.3 Shift polls

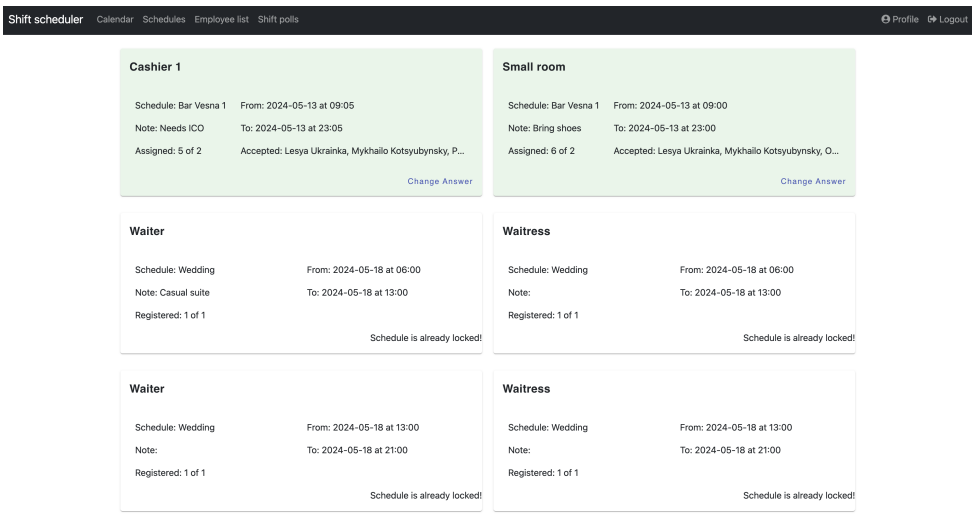

Figure 6.9: Implementation: Shift polls page, viewing

On the shift polls page, every user gets a notification for available shifts. He can accept, reject, and change answer later if the schedule is not finalized yet.

6.4.4 Calendar

Every user can see events planned for the restaurant where he/she works. There are 3 views: month, week, and day. Blue events represent schedules. Yellow events represent the shift where the shift poll was sent, however, there is no response from the current user yet. Transparent green events represent accepted shifts but for shifts where the schedule is not finalized yet. Green events represent events where the shift is accepted and the schedule is finalized. Finally, rejected shifts are not present in the calendar. Here you can see a month view where dots represent events.

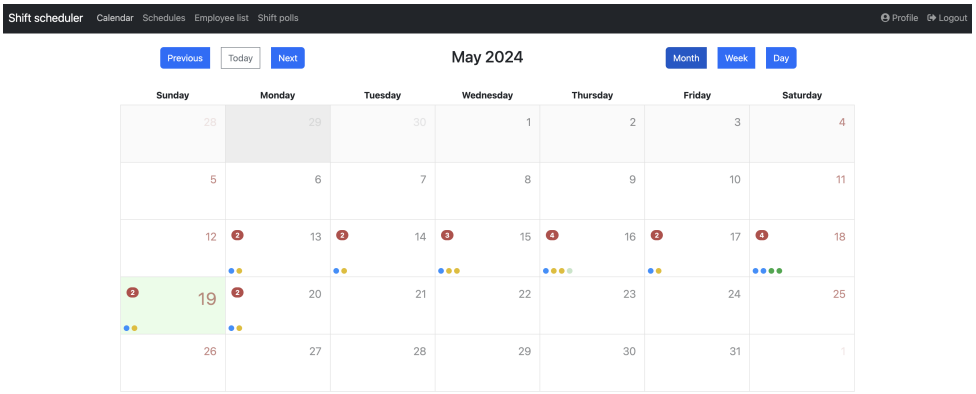

Figure 6.10: Implementation: Calendar page, month view

The next one is the week view.

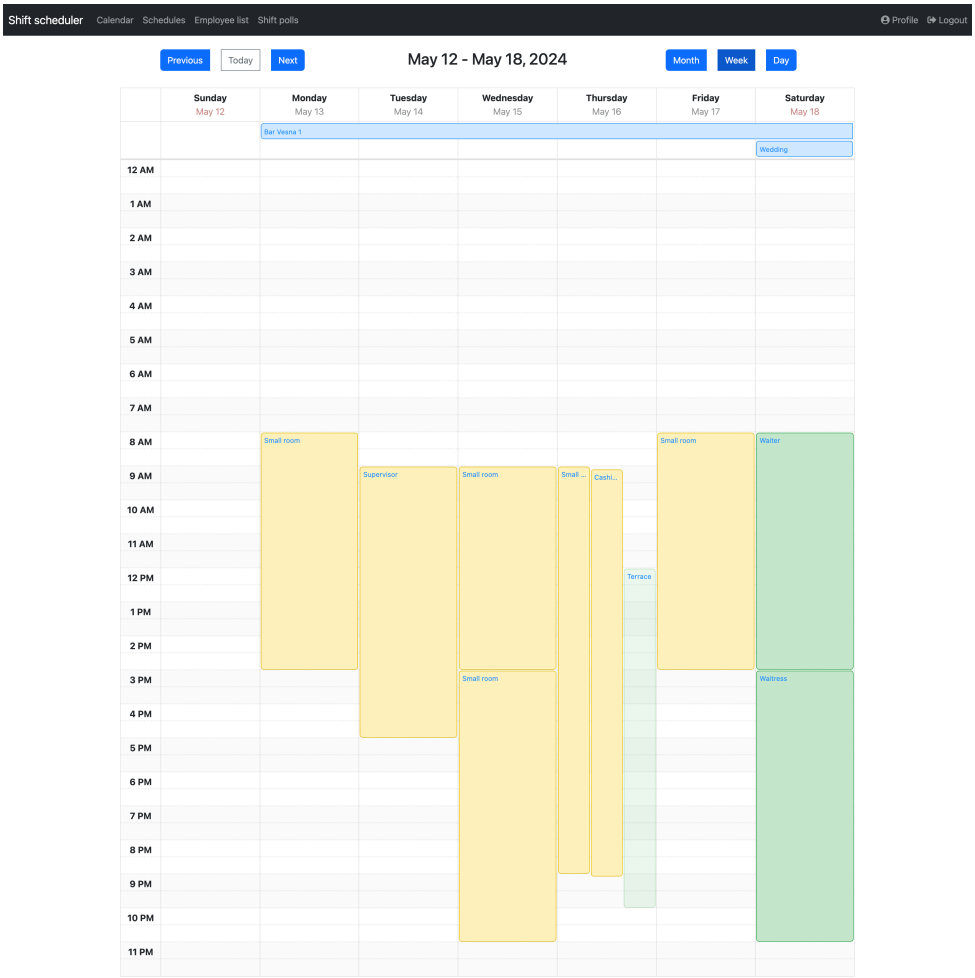

Figure 6.11: Implementation: Calendar page, week view

In the end here is the day view.

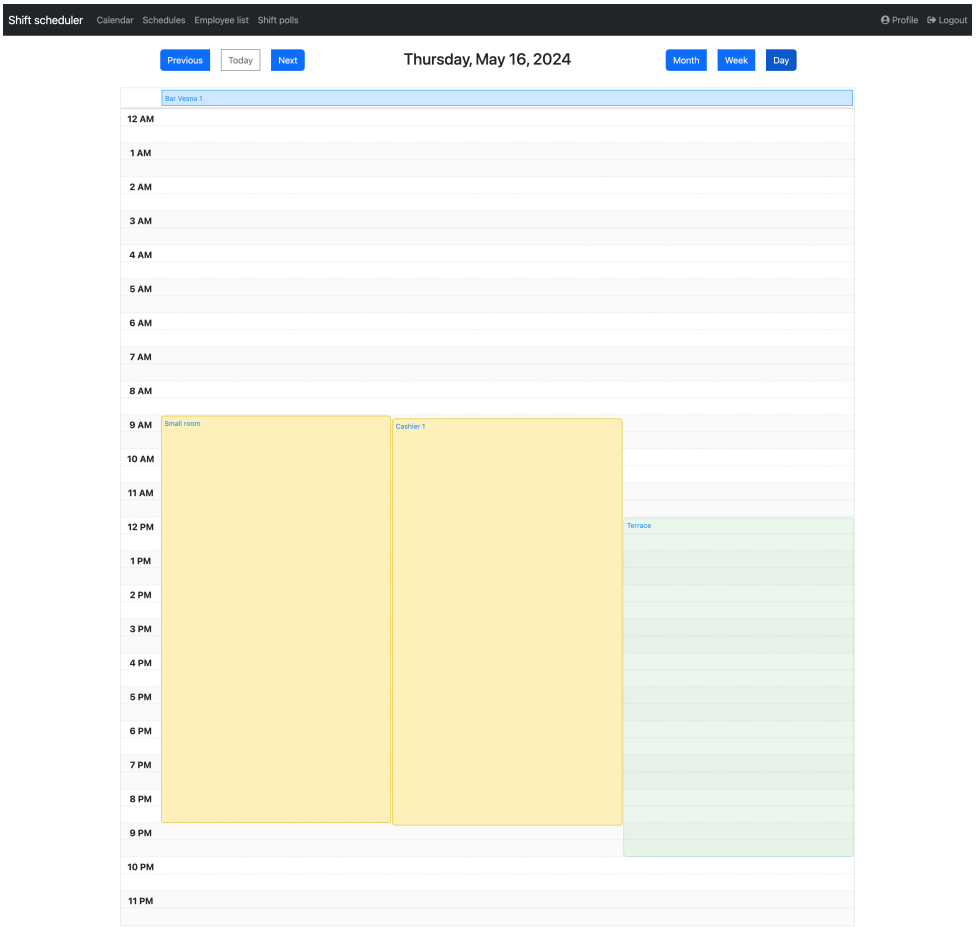

Figure 6.12: Implementation: Calendar page, day view

6.5 Implementation conclusion

Due to a lack of development resources the project was developed as MVP. The application has a minimalist design and offers all requested functionality to the final user. All needed scenarios are possible to achieve. Also, I used new and interesting technologies, so the solution corresponds to modern trends.

Simultaneously ideas for further improvements were collected during development. They will be present together with the testers' suggestions in conclusion.

Chapter 7

Testing

7.1 User Testing Approach

For testing, we used the grey box testing strategy[27] when the tester has limited knowledge of the internal workings of an application and also has the knowledge of fundamental aspects of the system.

7.2 Selection of User Testers

The decision to limit the number of testers to five for the usability testing phase of this application is informed by established usability research, particularly the findings reported by Nielsen and Landauer in their seminal work on the efficiency of user testing[26].

Jakob Nielsen's research demonstrates that the majority of usability problems in a design are identified when testing with just five participants. With five users, it is possible to uncover approximately 85% of usability issues in the system, making this approach both cost-effective and time-efficient. The curve of diminishing returns kicks in sharply after the fifth tester, as each subsequent tester is less likely to reveal new information. Therefore, testing with five participants ensures a balanced approach, maximizing insight while minimizing redundancy and resource expenditure.
In alignment with these principles, the current study will involve five testers who represent the target demographic of the application. This strategy ensures that the testing phase is optimized to gather actionable insights rapidly, which is crucial for iterative design processes within the tight timelines typical of thesis projects.

7.3 Expected Outcomes

By integrating Nielsen's methodology into the usability testing framework of this thesis, it is anticipated that the core usability issues will be efficiently identified, allowing for targeted improvements to the application's design. This approach not only aligns with best practices in UX research but also enhances the feasibility of conducting rigorous usability testing within academic constraints.

7.4 Tests

7.4.1 Test scenario

Prerequisites For testing 10 accounts in one restaurant were created. They are named test00, test01, ... test10. The first two are managers and all others are workers. All accounts are assigned to the same restaurant.

Test steps

1. **Log in as Manager 1:**
    - Navigate to the login page.
    - Enter valid credentials for Manager 1 and log in.
2. **Check Profile Information:**
    - Go to the profile section.
    - Verify that all profile information is displayed correctly.
3. **Check Employee List:**
    - Navigate to the employee list section.
    - Verify that the list of employees is displayed correctly.
4. **Create a Schedule:**
    - Navigate to the scheduling section.
    - Create a new schedule with relevant details (e.g., date range, schedule name).
5. **Create Shifts in the Schedule:**
    - Add shifts to the newly created schedule.

- Ensure shifts are correctly configured with appropriate times and roles.

6. **Send Shift Polls:**
   - Send shift polls to the relevant employees for the created shifts.
   - Verify that notifications are sent out successfully.
7. **Employee Response to Shift Polls:**
   - Log in from different employee accounts.
   - Accept or reject the shift polls received.
   - Verify that the responses are correctly recorded.
8. **Create Schedule by Queue:**
   - Use the queue method to create a new schedule.
   - Ensure the schedule is created correctly.
9. **Check Calendar:**
   - Navigate to the calendar section.
   - Verify that the scheduled shifts are displayed correctly.
10. **Log in as Manager 2:**
    - Log out from Manager 1's account.
    - Log in with Manager 2's credentials.
11. **Create Another Schedule:**
    - Navigate to the scheduling section.
    - Create a new schedule with relevant details for Manager 2.
12. **Create Shifts in the Schedule:**
    - Add shifts to the newly created schedule.
    - Ensure shifts are correctly configured with appropriate times and roles.
13. **Send Shift Polls:**
    - Send shift polls to the relevant employees for the created shifts.
    - Verify that notifications are sent out successfully.

14. **Employee Response to Shift Polls:**
    - Log in from different employee accounts.
    - Accept or reject the shift polls received.
    - Verify that the responses are correctly recorded.
15. **Create Schedule by Cost:**
    - Use the cost method to create a new schedule.
    - Ensure the schedule is created correctly.
16. **Check Calendar:**
    - Navigate to the calendar section.
    - Verify that the scheduled shifts are displayed correctly.

Expected results

- All profile information and employee lists are displayed correctly.
- Schedules and shifts are created and displayed accurately.
- Shift polls are sent and received successfully.
- Employee responses are correctly recorded and reflected in the system.
- Calendar accurately reflects the scheduled shifts.

7.4.2 User test in the middle of development

In the middle of development, user acceptance testing was performed to see all the current issues of the application. The following sections provide feedback, suggestions and complaints from the participants of this testing.

Tester 1

User has no shift polls: For empty shift polls, the message "User has no shift polls" should be displayed.

Edit Schedule/Shift: No pop-up calendar is available. No time validation is performed. Unable to change the "To date".

Adding Shift: Schedule data is not shown when adding a shift.

Creating Repeating Shifts*: Suggestion to add functionality for creating repeating shifts.

Shift Polls: The message "Already registered" is displayed. Missing information indicated by "—". Accept and Reject buttons are missing for both manager and worker.

Prolong Token: Tokens expire after approximately 30 minutes; recommendation to prolong token validity.

Calendar: Events span only 30 minutes instead of the expected couple of hours.

Pop-up in Calendar about Shift Capacity and Note*: Suggestion to add a pop-up in the calendar displaying shift capacity and notes.

It wasn't possible to finish the scenario as functionality was broken.

Points marked with * were proposed as good ideas but were not possible to achieve due to lack of resources and time.

7.4.3 Testing in the End of Development

Tester 2

Age: 25
Position: Barmen in restaurant

Prolongation of Pop-up Message Duration: It is recommended to extend the duration of pop-up messages related to schedule creation. The current short display time may not be sufficient for users to fully read and comprehend the information presented. A longer duration would enhance user awareness and interaction with the system.

Navigation Arrows in Calendar: Introducing navigation arrows on the left and right of the calendar date would significantly improve usability. Currently, the navigation buttons are positioned on the left, which has been found to be non-intuitive by some users. Placing arrows directly next to the calendar date would provide a more intuitive and user-friendly navigation experience.

Informative Field Tips: Enhancing the informativeness of field tips within the system is strongly recommended. More detailed and explanatory tips would aid users in understanding the functions and options available to them, thereby improving overall user experience and reducing the likelihood of errors.

Positive Aspects

Intuitiveness: The system has been praised for its intuitiveness, allowing users to navigate and utilize its features with ease. The straightforward design and clear layout contribute to a positive user experience.

Reliable Operation: The system operates reliably, with no significant disruptions or failures reported. This stability is crucial for maintaining user trust and ensuring consistent performance.

Limited Access for Workers: The implementation of limited access for workers has been positively received. This restriction ensures that only authorized personnel can access sensitive information and functionalities, thereby enhancing security and data protection.

Tester 3

Age: 27
Position: Manager at a restaurant

Lack of Reusable Shift Templates: There is a notable absence of reusable shift templates, which are frequently needed for routine scheduling. For instance, as a manager, I need to schedule two people at the bar from 10 AM to 6 PM, Monday to Friday, but each day requires individual setup instead of using a template that could span from May 18th to June 18th for weekdays only.

Post-Shift Addition Feedback: After adding a shift, there is no confirmation window such as "Shift successfully added" nor is there an option to "Create next shift" which would enhance the scheduling flow.

Non-Clickable Shifts in Calendar: Shifts displayed in the calendar are not clickable, which limits functionality and ease of use.

Managerial Calendar View Limitations: It is problematic that managers cannot view shifts in the calendar to see if they are filled or not and who is registered for them.

Quick Disappearing Error Messages: Error messages, especially those explaining why the calendar could not be finalized, disappear too quickly, making it hard to understand what went wrong.

Unavailable Cheaper Calendar Option: If it is impossible to implement a cheaper calendar option, the button for this feature should be disabled to prevent confusion.

Inconvenient Calendar Interaction: When accessing the monthly calendar and clicking on a day with a scheduled shift, only the shift title appears without the time, which is inconvenient for quick reference.

Positive Aspects

User-Friendly and Intuitive Navigation: The system is user-friendly with intuitive navigation and easy clickability without glitches. As a worker, it's very easy to select

available shifts. Conveniently, all necessary features are integrated within the calendar without any superfluous elements.

Tester 4

Age: 24
Position: Waiter in restaurant

Positive Aspects

Aesthetic Minimalist Design: The design of the application is commendable for its good-looking, minimalist approach, which enhances usability without overwhelming the user with unnecessary elements.

Smooth Application Flow: The application maintains a smooth flow, allowing for seamless navigation and operation, which significantly improves the user experience.

Areas for Improvement

Insufficient Validation Mechanisms: There is a noticeable lack of validation in several key areas of the application where it might be critical, potentially leading to user errors or data integrity issues.

Enhanced Shift Template Functionality: The absence of reusable shift templates hampers efficient scheduling. This functionality would help in routine task management, particularly for recurring shifts.

Tester 5

Age: 30 Position: Software developer

Timezone Discrepancies: The application currently displays a wrong timezone offset (+2 hours) on shift polls. This can lead to confusion and mismanagement of shift timings, adversely affecting operational efficiency.

Calendar Integration Flaws: After accepting shifts, particularly when non-managers exceed the shift capacity, these shifts do not appear in the calendar.

Inability to Create Cost-Efficient Schedules: The system lacks the functionality to create the cheapest schedule possible.

7.4.4 Test conclusion

The recent testing phase has yielded critical insights into the current state of my project, highlighting both strengths and areas in need of improvement. I have successfully addressed and resolved most of identified issues. These corrections have already been integrated into the development cycle, resulting in a better product offering.

Additionally, I have collected a set of findings that require further analysis and development. They will be shown in the Conclusion chapter.

Chapter 8

Conclusion

8.1 Results

The development and implementation of the employee shift scheduling system as described in this thesis represent a significant advancement in the use of technology for operational management within the restaurant industry. By employing Java for backend processes and the Gurobi solver for optimization, the system effectively addresses the complexities associated with employee shift scheduling, including handling overlapping shifts, employee availability, and minimizing operational costs.

The system not only automates the scheduling process but also introduces a high degree of flexibility and efficiency, proving its potential to significantly reduce the time managers spend on scheduling tasks. The Angular-based user-friendly frontend enhances user engagement and simplifies interactions, partially allowing for real-time updates and modifications. This aspect of the system is particularly crucial in the dynamic environment of restaurants where shift changes are frequent and sometimes urgent.

Furthermore, the architectural design of the system and the selection of technology have been proposed to ensure scalability and security, making the system suitable for both small eateries and larger restaurant chains. The use of the Springboot framework complements this by providing a robust and extensible platform for further development.

The evaluation of the system demonstrated notable improvements in scheduling efficiency and operational cost reductions, as highlighted by the user feedback and system performance metrics gathered during the testing phase.

Looking forward, there are several avenues for further research and development. Integration with other systems such as payroll and HR software, development of mobile applications for greater accessibility, and the incorporation of artificial intelligence to predict staffing needs based on historical data are all feasible enhancements that could increase the system's usefulness.

8.2 Future development

During the implementation and testing phases, ideas for future development were collected. They were not possible to achieve during the current development due to limited developer capacity. Implementation of these ideas would bring more benefits and features to existing solution.

1. **Add templates for shifts and schedules**
   The templates would make the manager's work easier and faster. It is quite a common situation when schedules and shifts start and finish same time and have a similar name.

2. **Add repeating shifts**
   Again it would make managers work more efficiently. The idea behind this proposal is to fill in default information for a shift and then choose to repeat this shift every day/week/month.

3. **Add more information to shifts and schedules in calendar**
   As was noticed during testing adding more information to shifts and schedules as capacity, notes and registered users would be beneficial for both workers and managers.

4. **Add additional validation and more messages with information**
   These steps would make processes even more intuitive and would prevent possible mistakes.

5. **Small front-end improvements**
   It is always possible to make better graphical interfaces to make customers more satisfied. However, sometimes seeing the best front-end solution is subjective.

Appendix A

Application source code

Figure A.1: Apendix: QR code for project repository

To check the application source code visit the project repository. To visit the project repository use the QR code above(it is clickable).

Appendix B

How to run the application

1. To run the application, you need to install Java 21 or higher. You can download it from `https://www.oracle.com/java/technologies/java-se-glance.html`.
2. You need Node.js and the npm package manager to run the application. You can download them from `https://nodejs.org/en/`.
3. To run the application, you need to install MongoDB. You can install it from `https://www.mongodb.com/try/download/community`.
4. Create your credentials for the database, then create the database itself. Fill in the credentials in `shift-scheduler/src/main/resources/application.yaml`.
5. To start the backend, run `shift-scheduler/src/main/java/cz/cvut/fel/shiftscheduler/ShiftSchedulerApplication.java`.
6. Run `npm install -q @angular/cli` in the frontend folder.
7. Run `ng build` to start the frontend.

Appendix C

Acronyms

- **SQL** - Structured query language
- **DBMS** - Database management system
- **RDBMS** - Relational database management system
- **JPA** - Java persistence API
- **API** - Application Programming Interface
- **ACID** - Atomic, Consistent, Isolated & Durable
- **JSON** - JavaScript Object Notation
- **BSON** - Binary JSON
- **LP** - Linear programming
- **JWT** - JSON Web Token
- **MVP** - Minimum viable product
- **LTS** - Long-term support
- **ORM** - Object–relational mapping

Bibliography

[1] Connecteam. Connecteam: The All-In-One App to Manage Your Team [online]. Connecteam, 2024. Available from: https://connecteam.com/

[2] ADP. ADP Official Site — Payroll, HR and Tax Services [online]. ADP, 2024. Available from: https://www.adp.com/

[3] Homebase. Homebase: Free Employee Scheduling, Time Clocks, Timesheets, & More [online]. Homebase, 2024. Available from: https://joinhomebase.com/

[4] Calamari. Calamari: Leave Management & Clocking in System [online]. Calamari, 2024. Available from: https://www.calamari.io/

[5] Fourth. Fourth: End-to-End Restaurant and Hospitality Management Technology Solutions [online]. Fourth, 2024. Available from: https://www.fourth.com/

[6] Jolt. Employee Scheduling [online]. Jolt, 2024. Available from: https://www.jolt.com/products/employee-scheduling/

[7] Sling. Sling: Employee Scheduling Made Easy [online]. Sling, 2024. Available from: https://getsling.com/

[8] Zoho People. Zoho People: HR Software for Growing Businesses [online]. Zoho, 2024. Available from: https://www.zoho.com/people/

[9] When I Work. Employee Scheduling Software [online]. When I Work, 2024. Available from: https://wheniwork.com/

[10] BENSON, S. Leveraging Mental Models in UX Design. In: Toptal.com [online]. 29. 8. 2019. Available from: https://www.toptal.com/designers/ux/mental-models-ux-design

[11] KRUG, S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd Edition. San Francisco: New Riders, 2013. ISBN 9780133597271.

[12] OSMAN, M. How to Create a Wireframe Map. In: Blog.hubspot.com [online]. HubSpot, 22.5.2020. Available from: https://blog.hubspot.com/website/wireframe-map

[13] Balsamiq Wireframes. (n.d.). Industry Standard Low-Fidelity Wireframing Software. Balsamiq. Rapid, Effective and Fun Wireframing Software — Balsamiq [online]. Available from: https://balsamiq.com/wireframes/

[14] PostgreSQL. (n.d.). A powerful, open source object-relational database system [online]. Retrieved from https://www.postgresql.org/

[15] MongoDB. (n.d.). A general purpose, document-based, distributed database built for modern application developers and for the cloud era [online]. Retrieved from https://www.mongodb.com/

[16] Java® Platform, Standard Edition & Java Development Kit Version 21 API Specification [online] © 1993, 2024, Oracle and/or its affiliates, 500 Oracle Parkway, Redwood Shores, CA 94065 USA. https://docs.oracle.com/en/java/javase/21/docs/api/index.html

[17] Spring Boot © 2005 - 2024 Broadcom [online]. https://docs.spring.io/spring-boot/documentation.html

[18] React. (n.d.). A JavaScript library for building user interfaces, maintained by Facebook and a community of individual developers and companies [online]. Retrieved May 23, 2024 from https://react.dev/

[19] Angular. (n.d.). A platform for building mobile and desktop web applications using TypeScript/JavaScript and other programming languages [online]. Retrieved May 23, 2024 from https://angular.io/

[20] Vue.js. (n.d.). An approachable, performant, and versatile framework for building web user interfaces [online]. Retrieved May 23, 2024 from https://vuejs.org/

[21] Gurobi Optimization, LLC. (n.d.). Java API overview. [online] © Gurobi. Retrieved 20.05.2024 from https://www.gurobi.com/documentation/current/refman/java_api_overview.html

[22] SOLID Principles in Software Architecture and Introduction to RESM Concept in OOP [online]. © 01.03.2015 Vamsi Madasu, Trinadh Venna, Tarik Eltaeib. Available from: https://www.researchgate.net/publication/273451390_SOLID_Principles_in_Software_Architecture_and_Introduction_to_RESM_Concept_in_OOP

[23] User interface design of a planning calendar for tutoring lessons shared between students and teachers. [online] © Sabina Balaeva 2023 https://dspace.cvut.cz/handle/10467/108791

[24] Spring Framework. (n.d.). @Bean (Spring Framework 6.0.9 API). Spring [online]. Retrieved May 18, 2024 from https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Bean.html

[25] Spring Framework. (n.d.). Dependencies and Configuration in Spring (Spring Framework Reference Documentation) [online]. Spring. Retrieved May 18, 2024 from https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html

[26] Why You Only Need to Test with 5 Users. [online] © 2000 Jakob Nielsen https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

[27] Khan, Mohd Ehmer, and Farmeena Khan. "A comparative study of white box, black box and grey box testing techniques." International Journal of Advanced Computer Science and Applications 3.6 (2012) [online].

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ab0ed151c010e03e2d34284ae5f89756372e7690#page=22