Web application for team work organization

Webová aplikace pro organizaci týmové práce

Bachelor's Degree Project

Author:	**Kirill Borodinskiy**
Supervisor:	**doc. Ing. Miroslav Virius, CSc.**
Academic year:	2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Borodinskiy** Jméno: **Kirill** Osobní číslo: **518594**

Fakulta/ústav: **Fakulta jaderná a fyzikálně inženýrská**

Zadávající katedra/ústav: **Katedra matematiky**

Studijní program: **Aplikovaná informatika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Webová aplikace pro organizaci týmové práce

Název bakalářské práce anglicky:

Web application for team work organization

Pokyny pro vypracování:

1. Seznamte se s knihovnami pro tvorbu webových aplikací založenými na programovacím jazyce Java.
2. Sestavte seznam požadavků na aplikaci pro organizaci týmové práce.
3. Na základě analýzy těchto požadavků navrhněte aplikaci.
4. Navrženou aplikaci implementujte a otestujte.

Seznam doporučené literatury:

[1] Nick Williams: Professional Java for Web Applications. John Wiley & Sons 2014. ISBN 9781118656464
[2] David A. Chappell, Tyler Jewell, Michael Wooten: Java Web Services. O'Reilly Media, 2002.
[3] Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Sampaleanu: Professional Java Development with the Spring Framework. Wrox, 2005.
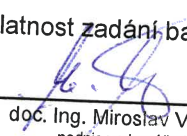
Jméno a pracoviště vedoucí(ho) bakalářské práce:

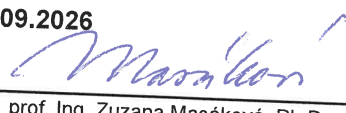**doc. Ing. Miroslav Virius, CSc.**   **katedra softwarového inženýrství   FJFI**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **25.10.2024**   Termín odevzdání bakalářské práce: **04.08.2025**

Platnost zadání bakalářské práce: **30.09.2026**

_____
doc. Ing. Miroslav Virius, CSc.
podpis vedoucí(ho) práce

_____
prof. Ing. Zuzana Masáková, Ph.D.
podpis vedoucí(ho) ústavu/katedry

_____
doc. Ing. Václav Čuba, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

31. 10. 2024
_____
Datum převzetí zadání

_____
Podpis studenta

FAKULTA JADERNÁ A FYZIKÁLN INŽENÝRSKÁ
FACULTY OF NUCLEAR SCIENCES AND PHYSICAL ENGINEERING
B ehová 7
115 19 Praha 1

DECLARATION

I, the undersigned

Student's surname, given name(s): Borodinskiy Kirill
Personal number: 518594
Programme name: Applied Informatics

declare that I have elaborated the bachelor's thesis entitled

Web application for team work organization

independently, and have cited all information sources used in accordance with the Methodological Instruction on the Observance of Ethical Principles in the Preparation of University Theses and with the Framework Rules for the Use of Artificial Intelligence at CTU for Academic and Pedagogical Purposes in Bachelor's and Continuing Master's Programmes.

I declare that I used artificial intelligence tools during the preparation and writing of this thesis. I verified the generated content. I hereby confirm that I am aware of the fact that I am fully responsible for the contents of the thesis.

In Prague on 29.06.2025

Kirill Borodinskiy
....
student's signature

Název práce:

Webová aplikace pro organizaci týmové práce

Autor: Kirill Borodinskiy

Studijní program: Aplikovaná informatika

Druh práce: Bakaláská práce

*Vedoucí práce:*doc. Ing. Miroslav Virius, CSc., DrSc., VUT,Fakulta jaderná a fyzikáln inenýrská, Katedra softwarového inenýrství

Abstrakt: Tato práce pedstavuje návrh a implementaci TeamJob, samostatn hostovaného open-source systému pro plánování týmové práce ureného pro organizace, které upednostují soukromí, flexibilitu a kontrolu nad svými plánovacími daty. Aplikace je postavena pomocí moderních Java webových technologií vetn Spring Boot, Thymeleaf a PostgreSQL s implementací modulární MVC architektury a JWT autentifikací pro zajitní bezpených bezstavových uivatelských relací. Systém poskytuje komplexní monosti správy událostí vetn detekce konflikt, pokroilého filtrování podle uivatel, místností a títk a flexibilního systému títkování pro organizaní úely. Implementace zahrnuje ízení pístupu na základ rolí, auditní stopy a funkce optimalizace zdroj prostednictvím funkce "Find Available". Rozsáhlé testování na úrovni slueb, kontroler a repozitá potvrdilo spolehlivost systému s 100% úspností test napí 47 testovacími pípady. Zatímco backendová logika pro opakující se události je pln implementována, frontendové zobrazení vyaduje dalí vývoj. Projekt demonstruje proveditelnost vytvoení pizpsobitelné, soukromí zamené plánovací platformy, která vyvauje funkcionalitu s bezpeností a poskytuje solidní základ pro budoucí vylepení v kolaborativních plánovacích systémech.

Klíová slova: kalendáový systém, JWT autentifikace, Java, PostgreSQL, Spring Boot, správa událostí, týmová spolupráce, webová aplikace

Title:

Web application for team work organization

Author: Kirill Borodinskiy

Abstract: This thesis explains the design and implementation of TeamJob, a self-hosted, open-source team scheduling system developed for organizations that prioritize privacy, flexibility, and control over their scheduling data. The application is built using modern Java web technologies, including Spring Boot, Thymeleaf, and PostgreSQL. It features a modular MVC architecture with JWT-based authentication to ensure secure, stateless user sessions. The system provides comprehensive event management capabilities, such as conflict detection and advanced filtering by users, rooms, and tags, along with a versatile tagging system tailored for organizational purposes. The implementation includes role-based access control, audit trails, and resource optimization functionalities executed through the "Find Available" feature. Extensive testing conducted at the service, controller, and repository levels verified the system's reliability, yielding a test pass rate of 100% across 47 test cases. The back-end logic for recurring events has been entirely implemented, but further enhancement of the front-end display is required. This project demonstrates the feasibility of developing a customizable, privacy-centric scheduling platform that proficiently integrates functionality with security, thus providing a solid foundation for future advancements in collaborative scheduling systems.

Contents

Chapter 1

Introduction

1.1 Motivation

Efficient team schedule planning is a complex challenge, particularly in organizations that require real-time coordination and resource management. Existing scheduling services often have significant limitations, such as proprietary nature, lack of customization, and dependence on third-party infrastructure. This highlights the need for tailored solutions that can address specific organizational requirements more effectively. This project aims to develop a self-hosted open-source booking system designed for organizations that need a private, adaptable scheduling solution. The system will provide a web-based interface where users can:

- Make and manage reservations

- Check real-time room availability

- Filter bookings by person or room

- View all reservations on a centralized calendar

The backend will be built using the Spring Framework, ensuring scalability, security, and ease of integration with existing infrastructure. Unlike cloud-based alternatives, this system will store all data locally, giving organizations complete privacy and control over scheduling information. By combining flexibility, transparency, and data privacy, this project can provide a practical alternative to commercial scheduling tools, empowering organizations with greater autonomy and customization options.

1.2 The current solution
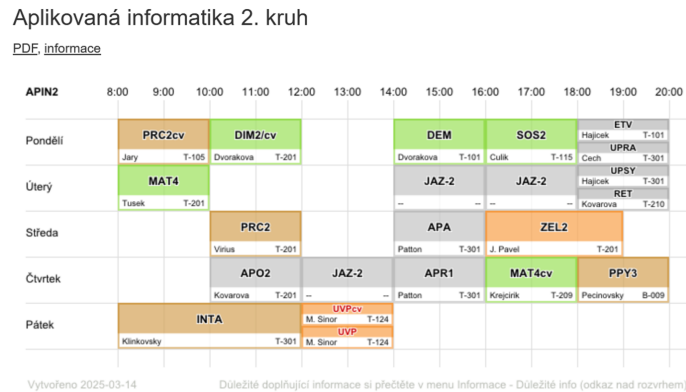
Figure 1.1: Screenshot of the current solution

Initially, it is needed to examine the existing solution employed by my institution, CTU. The website Rozvrh.fjfi.cvut.cz serves as a platform where students can view their academic schedules.

It can be seen in fig. 1.1 that while the website completes its main purpose, it is not customizable, making it difficult to use. The current implementation presents several significant limitations that impact the user experience and efficiency.

Limited Customization: The system lacks personalization features, forcing all users to view the same static schedule format regardless of their specific needs or preferences.

Cross-Program Scheduling Complexity: Students enrolled in classes over different years or programs face particular challenges. They must manually navigate between multiple schedule views and compare information from different sources. This process is not only time-consuming, but is also prone to errors.

Manual Workarounds: The limitations of the current system have led to widespread adoption of inefficient workarounds. Many students resort to taking screenshots of their schedules and manually marking or crossing out irrelevant classes. This practice, while common, introduces several problems: Increased risk of scheduling conflicts due to manual tracking, difficulty in keeping track of schedule updates, time wasted on manual schedule management, potential for missed classes due to human error.

These limitations highlight the urgent need for a more sophisticated scheduling solution. A modern calendar system should address these issues by providing:

Personalization: Allow users to customize their view based on their specific needs and preferences.

Cross-Program Integration: Enable seamless viewing of classes across different programs and years.

Digital-First Approach: Eliminate the need for manual workarounds by providing a comprehensive digital solution

This project aims to develop such a solution, focusing on creating a calendar system that is not only functional, but also user-friendly and customizable.

1.3   Thesis Aims and Objectives

The primary objective of this thesis is to explore and evaluate Java-based web development libraries while designing and implementing a practical team-work organization application. This project aims to demonstrate the application of modern Java web technologies in creating a functional and user-friendly system for team collaboration.

1.3.1   Primary Objectives

The development process is guided by several key objectives that form the foundation of this project. These objectives are designed to ensure a systematic and comprehensive approach to the development of the team-work organization system.

The first step focuses on the *exploration and evaluation of technology*. In this phase, we will conduct comprehensive research into available Java web development libraries and frameworks. This exploration will encompass various aspects including web application development, database integration, user interface implementation, and security measures. Based on this research, we will carefully select and document the most appropriate technologies for the project's specific requirements.

The second objective involves a thorough *requirement analysis*. This phase will identify and document all the functional requirements necessary for effective team-work organization. Additionally, we will establish non-functional requirements covering performance expectations, security needs, usability standards, scalability considerations, and maintainability requirements. This comprehensive analysis will ensure that the system can meet both immediate and long-term organizational needs.

The system design is the third stage, where we will develop a complete system architecture incorporating the selected technologies. This includes designing an efficient database schema for the team work organization, creating detailed user interface mockups, defining system components and their interactions, and establishing a robust security architecture. The design phase will focus on creating a scalable and maintainable system structure.

Implementation forms the fourth phase, where we will develop core functionalities essential for the system's operation. This includes comprehensive user management capabilities, room creation and management features, event assignment and tracking systems, and advanced event displaying and filtering options. The implementation will also incorporate robust security measures, a responsive user interface, and ensure complete data integrity throughout the system.

The final stage focuses on *testing and validation*. This phase will involve developing and executing comprehensive test cases at multiple levels, including unit testing, integration testing, and user acceptance testing. We will verify all system functionality against the established requirements, document test results and any issues discovered, and implement necessary improvements based on testing outcomes. This thorough testing approach will ensure the reliability and effectiveness of the system in real-world applications.

1.3.2 Expected Outcomes

The successful completion of these objectives will result in:

- **A comprehensive understanding** of Java web development libraries and their practical applications

- **A functional application** for team work organization that: Facilitates efficient team collaboration, Provides intuitive event management, Ensures secure interactions with the system, Offers responsive and user-friendly interface.

- **Detailed documentation** covering: technology evaluation and selection process, system architecture and design decisions, implementation details, testing methodology, and results.

- **Practical experience** in: Java web application development, database design and implementation, user interface development, system testing and validation.

Chapter 2

Tools

2.1 Tool Selection and Technology Stack

This chapter examines the decision-making process behind selecting the technologies and tools for the team-work organization system. The selection criteria were based on several key factors: project requirements, scalability needs, security considerations, and long-term maintainability.

2.1.1 Framework Selection

Given the project's requirement to develop a Java application, the framework selection was focused on Java-based solutions. The primary options considered were *The Spring Framework*, *Spring Boot*, *Jakarta EE* (formerly Java EE), and *Micronaut*. Although all of these frameworks are capable of building robust web applications, they differ significantly in their approach and complexity.

Jakarta EE, the enterprise edition of Java, provides a comprehensive set of specifications to build enterprise applications. However, it requires significant boilerplate code and configuration, making it less suitable for this project. **Micronaut**, while promising with its ahead-of-time compilation and low memory footprint, is relatively new and has a smaller community compared to Spring Boot.

The Spring Framework, the foundation of Spring Boot, is a comprehensive programming and configuration model for modern Java-based enterprise applications. It provides a wide range of features including dependency injection, aspect-oriented programming, and transaction management. However, the Spring Framework requires extensive configuration and setup, which can be time consuming and complex.

Spring Boot, built on top of the Spring Framework, was selected as it addresses these configuration challenges while maintaining all the benefits of the Spring Framework. It should perform on par or even better than Django [12] and other frameworks. It provides the following:

- Auto-configuration of Spring Framework components

- Embedded servers for simplified deployment

- Production-ready features like metrics and health checks

- Reduced boilerplate code and configuration

This combination of Spring Framework's robust features and Spring Boot's simplified development approach makes it ideal for this project. The framework's extensive documentation, large community support, and proven track record in enterprise applications further reinforce this choice. Spring

Boot's microservice architecture capabilities have been demonstrated in various applications[11], and its cloud application development features provide significant advantages for scalable systems[7].

2.1.2 Frontend Technology Selection

The selection of front-end technology required careful consideration of integration capabilities with Spring Boot. **Thymeleaf**, as a server-side template rendering engine, was identified as the preferred option due to several factors. It utilizes server-side page rendering to decrease the computational load on client systems. Its integration of Java with HTML makes it relatively straightforward to use. Furthermore, it is frequently used in conjunction with the Spring Boot. Although modern alternatives like **React** and other JavaScript frameworks are prevalent, they would require significant additional research and development time.

The decision to use Thymeleaf was influenced by the simplicity of combining HTML and Java code in a single file. However, modern web development requirements necessitated the incorporation of some JavaScript for specific client-side tasks, particularly for handling asynchronous requests and dynamic content updates.

2.1.3 Database Selection

PostgreSQL was selected as the primary database system after evaluating various database options, including NoSQL alternatives. This choice was driven by PostgreSQL's prevalence in the market and its robust ACID compliance, which ensures data integrity and reliability. The ACID properties provide essential guarantees for data management:

- **Atomicity**: Ensures transactions are completed entirely or not at all.

- **Consistency**: Maintains database validity through all transactions.

- **Isolation**: Prevents concurrent transactions from interfering with each other.

- **Durability**: Guarantees committed transactions remain permanent.

These properties are crucial for maintaining data integrity in a multi-user environment where concurrent access and modifications are common.

2.1.4 Security Architecture Decisions

The security implementation strategy was developed after careful analysis of the security requirements of modern web applications. The decision to use **JWT-based authentication** was made after considering several factors, including the benefits of a stateless architecture that eliminates server-side session storage and improves scalability. The approach also offers excellent cross-platform compatibility, enabling seamless integration with various clients while maintaining compliance with industry best practices for web security. Alternative approaches like **session-based authentication** were evaluated but rejected due to their increased complexity with server-side session storage, increased server resource requirements, and complex session management.

The main security framework used is **Spring Security**, which provides a comprehensive security solution for Spring-based applications. Although it is a powerful framework, it is not without its own challenges, such as the need to configure multiple security components and the potential for increased complexity in the codebase. Still, the benefits of using a well-established security framework benefits the project in the long run.

2.1.5   Development Tool Selection

The choice of development tools was guided by the need to improve code quality and development efficiency. **Lombok** was selected because it effectively reduces boilerplate code without runtime overhead, integrates seamlessly with existing IDEs, and maintains code readability while reducing verbosity. The IDE used is **IntelliJ IDEA**, which has excellent support for Java.

2.1.6   Containerization Strategy

Containerization was implemented using **Docker** to ensure consistent deployment across different environments and simplify the development workflow. The containerization strategy employs a multi-stage build process that optimizes both build time and final image size. The build process utilizes **Amazon Corretto JDK 21** to compile the application and run it in a separate dockerfile stage. The system includes integrated health checks to monitor container status, ensuring reliable operation and quick detection of potential problems.

This containerization approach delivers significant benefits to the project. It ensures a consistent runtime environment across development, testing, and production stages, eliminating environment-specific issues. The deployment process is simplified, reducing the complexity of moving the application between different environments. Container isolation provides an additional layer of security, while the standardized container format enables easy scalability and orchestration capabilities. Furthermore, this approach effectively eliminates the common "works on my machine" problem, as all environments run the same containerized application. The basics of this approach were learned from the video resource [5].

2.1.7   Technology Stack Integration

The selected technologies were tested for their ability to work together. The integration strategy focused on ensuring compatibility between all components, maintaining optimal system performance across all layers, and creating a system that can be easily updated and modified. This careful selection and integration of technologies has resulted in a robust, scalable, and maintainable system that meets project requirements while providing a solid foundation for future enhancements.

Chapter 3

Methods

3.1 Requirements Specification

3.1.1 Functional Requirements

The system provides comprehensive functionality in three main areas: **Basic Calendar Features**, **Additional Calendar Features**, **Secure Interactions**.

Under the **Basic Calendar Features** category, the system offers weekly and daily views, with filtering capabilities by room and user. The system allows managers to create, edit, and delete events, while users can only view the calendar and search for resource availability.

In the **Additional Calendar Features** category, the system offers a day view, which displays events for a specific day organized by room. Furthermore, the system offers a `FindAvailable` feature, which allows users to search for available resources (rooms, users, or events) based on various criteria.

Under the **Secure Interactions** category, the system offers a login system, which allows users to log into the system using a username and password. The system stores all passwords in a hashed format using the BCrypt algorithm. Instead of traditional session-based authentication, the system uses JWT tokens to authenticate users stateless-way. This method was learned from the video resource [1].

3.1.2 Non-Functional Requirements

The application prioritizes function over design by providing a simple and intuitive user interface. Navigation and interaction patterns remain consistent throughout the system, with clear error messages and feedback mechanisms to guide users.

Security is implemented through JWT-based authentication, with all communication restricted to HTTPS. The system incorporates protection against common web vulnerabilities and maintains strict access controls.

Reliability is ensured through robust error handling and data consistency mechanisms.

Maintainability is supported by comprehensive documentation, a modular architecture, and extensive test coverage. This ensures that the system can be easily updated and maintained over time.

3.2 System Architecture

3.2.1 Overall Architecture

The application follows a four-tier architecture pattern, separating concerns across `Models`, `Views`, `Controllers`, and the `Database`. This separation enables better maintainability and scalability while keeping the system modular and efficient. (See fig. 3.1 for the system architecture diagram.)
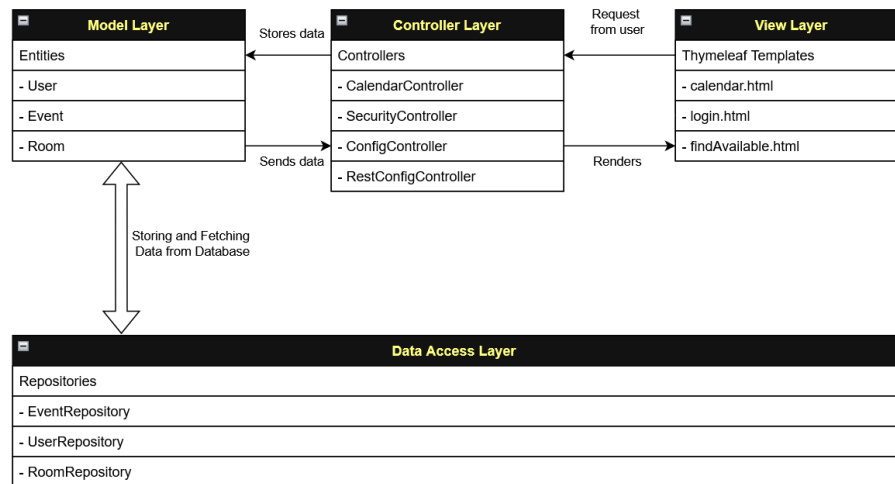

Figure 3.1: System Architecture Diagram

3.2.2 Key Components

The front-end components utilize Thymeleaf templates for server-side rendering, enhanced with JavaScript for client-side interactivity and CSS for design. This combination provides a modern, dynamic user experience while maintaining performance, as the Thymeleaf templates are rendered on the server side.

The backend architecture consists of controllers handling HTTP requests and responses, services implementing business logic, repositories managing data access, and a comprehensive security layer handling authentication and authorization.

3.2.3 API Design

The application provides both REST API endpoints and view-based endpoints to support various client needs. The REST API follows standard conventions and provides endpoints for authentication, resource management, and system queries.

Authentication endpoints handle user sign-in, registration, and logout operations. Resource endpoints manage rooms and events, while query endpoints provide functionality to check resource availability and validate authentication tokens.

View endpoints serve the user interface, with dedicated routes for calendar views and system configuration. Calendar views support weekly and daily perspectives, while configuration views provide interfaces for managing rooms, events, and users.

The basics of this design were adopted following guidance from the video resource [9].

17

3.3 Database Design

3.3.1 Database Schema

The database schema (see fig. 3.2) is designed around three main entities: users, rooms, and events.

After that, the `roles` table is used to manage the user roles.

In addition, 3 tables of `tags` are used to manage the tags for rooms, events, and users to simplify the search for resources.
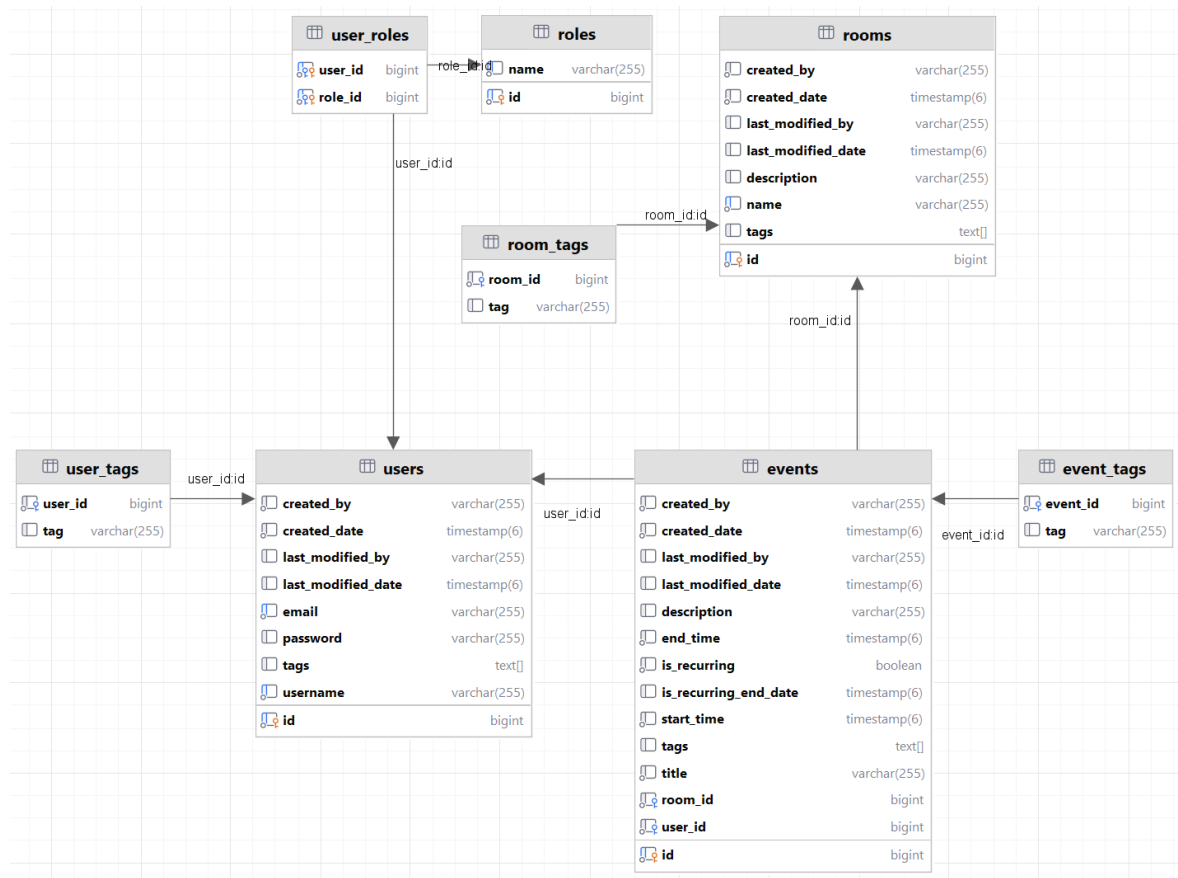
Figure 3.2: Database Schema Diagram

3.3.2 Key Tables and Relationships

The table "users" stores essential user information, including username, email, and securely hashed passwords. Maintains a many-to-many relationship with the role table, enabling flexible role assignment and management. This solution is not the most elegant one, as it requires additional queries to get the roles of a user. However, it was chosen to simplify the implementation of authentication(which validates permissions) and authorization(which validates access to resources).

The roles table defines the various user roles within the system, with a connection to the users table using another table called `user_roles`. This design allows for the easy addition of new roles and modification of existing role permissions.

18

The room table contains information about spaces, including names, descriptions, and categorization tags. Creating a one-to-many relationship with the events table, the rooms table can have multiple events, but not at the same time.

The event table serves as the central scheduling entity, storing event details such as title, description, timing information, and recurrence settings. Events maintain relationships with both users and rooms, tracking who is involved and where the event takes place. Recurrence is implemented following the RFC 5545 standard[2].

3.3.3   Spring Data JPA Implementation

The application relies on Spring Data JPA for efficient data persistence. Entity classes are annotated with @Entity and @Table, while repositories extend JpaRepository to provide standard CRUD operations. Relationships between entities are managed through JPA annotations and Spring handles transaction management automatically. The method of implementing the relationships was learned from the spring.io wiki [10].

3.4   User Interface Design

3.4.1   Design Philosophy

The design of the user interface emphasizes simplicity, consistency, responsiveness, and accessibility.

The main colors are

- **–color-gold** (46.8%, 92.9%, 61.6%): Gold

- **–color-primary** (0%, 0%, 100%): White

- **–color-secondary** (336%, 25%, 60%): Dusty Rose

- **–color-tertiary** (314%, 53%, 21%): Deep Magenta

- **–color-accent** (339%, 27%, 17%): Plum

- **–color-additional** (345%, 18%, 4%): Charcoal

3.4.2   Key Interface Components

The calendar view presents a weekly grid layout with color-coded events.

Event management features include forms to create and edit events, quick filters for different event types, and clear conflict warnings. The interface guides users through the event creation process while preventing scheduling conflicts.

Navigation is streamlined through a consistent menu structure.

Chapter 4

Results

4.1 Calendar Interface and Rendering

The calendar is the primary interface of the application, implemented through an integration of Thymeleaf templates and JavaScript to enhance interactivity. The underlying backend functionality is administered by the CalendarController and CalendarServiceImpl service. Upon users' access to the calendar, the controller delegates service methods such as setupModelForWeekCalendar and setupModelForDayCalendar to the responsibility of preparing the required data for the rendering of weekly or daily views. These methods retrieve all pertinent events from the database, filter them according to user, room, and tag parameters, and structure them into entities such as WeekDay and EventInADay. The filtration process is thorough, accommodating combinations of user IDs, room IDs, and tags, and is implemented in the service layer through the use of helper methods that parse and apply these filters. Subsequently, the calendar view is rendered by Thymeleaf, which receives all data as model attributes, while JavaScript is employed to augment navigation and interactivity, easing shifts between weeks or days.

4.2 Event Creation, Validation, and Conflict Detection

The creation of events is managed through REST endpoints within the RestConfigController. Upon submission of a new event by the user, the backend initiates a check for scheduling conflicts employing the findOverlappingEventsInRoom method of the EventRepository. This mechanism ensures that no two events simultaneously occupy the same room. In instances where a scheduling conflict is identified, the system issues a conflict response, requiring the user to select an alternative time or resource. The event creation process accommodates both singular and recurring events. For recurring events, the system constructs an RRULE string that complies with RFC 5545 standards, including support for intervals, end dates, and specific weekdays in the case of weekly recurrences. The event entity is responsible for maintaining all recurrence-related information, including exception dates (exdate) and supplementary dates (rdate). While the backend logic for recurring events is fully implemented and functional, the frontend display of recurring events in the calendar interface has not been completed. The system successfully stores and processes recurring event data, but the visual representation of these events in the calendar views requires further development. In addition, the DefaultValueService has been implemented to facilitate the demonstration of the project. It generates test events, users, and rooms to verify the system's functionality under conditions of load.

4.3 Filtering by User, Room, and Tags

The capability to filter is a fundamental aspect of the calendar and event views. The backend enables the filtration of events based on user IDs, room IDs, and tags associated with users, rooms, or events. This functionality is executed in the CalendarServiceImpl service, where methods such as convertToDayEvents and generateAvailableTimeRequest implement filters on the event list prior to rendering to the frontend. The filtering mechanism interprets comma-delimited strings of IDs or tags, converts these into sets, and applies them to the event data using Java Streams and supporting techniques. Consequently, events are excluded if their associated user, room, or tags do not correspond to the designated criteria. This enables users to focus on the most relevant events, even in scenarios marked by a substantial number of planned activities.

4.4 Authentication and Role-Based Access Control

Authentication is achieved through the use of JWT (JSON Web Tokens), with the SecurityController overseeing the workflows of sign-in, sign-up, and sign-out. The user credentials are subjected to secure hashing and storage, and the JWT tokens are distributed upon successful completion of authentication. The TokenFilter plays a critical role in intercepting requests, extracting and validating JWT tokens, and establishing the security context for the authenticated user.

Role-based access control is enforced throughout the application. Users are assigned specific roles such as ROLE_USER, ROLE_ADMIN, or ROLE_CONFIG, which are stored within the database and linked to each user. The SecurityConfigurator utilizes Spring Security to impose access restrictions on sensitive endpoints that depend on these roles. For instance, only users possessing roles of ROLE_CONFIG or ROLE_ADMIN are authorized to create, edit, or delete events and rooms, whereas regular users are restricted to activities like viewing and searching. CSRF protection is systematically enabled for all forms.

The auditing part of the Spring Data framework[6] serves as a mechanism to monitor alterations to entities. It records the timestamp of creation, identifies the creator, and documents the identity and timing of modifications within the database. This framework facilitates the verification of the authorship of any modifications made within the system.

4.5 Tagging System

The application incorporates a robust tagging system applicable to users, rooms, and events. Tags are maintained as collections of strings within each entity (User, Room, Event) and are stored in separate tables utilizing JPA's @ElementCollection. Tags are employed extensively throughout the interface for the purposes of filtering and searching, and are also presented in both list and detailed views. The backend includes methods to retrieve all unique tags from the database, which are subsequently displayed in dropdown menus and filter panels within the user interface. Tag-based filtering is executed within the service layer, enabling users to efficiently access pertinent information by selecting one or more tags. Furthermore, the system supports the allocation of random tags to entities during the generation of test data, thus ensuring a diverse array of tags for both demonstration and testing purposes.

4.6   Error Handling and Custom Error Pages

The management of errors is facilitated by an error controller, referred to as the CustomErrorController, which intercepts errors and generates a comprehensive error page `error-debug.html`. This controller is responsible for extracting error statuses, exception details, messages, and the pathways of requests, and it incorporates stack traces whenever they are available. This methodology affords developers and administrators exhaustive information necessary for debugging and system maintenance, while concurrently supplying end users with lucid feedback regarding any issues encountered.

4.7   Room and User Management

The management of rooms and users is integrated into both backend and frontend systems. The ConfigController offers endpoints for the enumeration, addition, and removal of rooms and users. In the backend, there are strict verifications to ensure that only users with designated roles are authorized to carry out these operations. The system performs checks to ensure uniqueness in identifiers such as room name, username, and email.

The frontend displays comprehensive lists of rooms and users, inclusive of their tags and roles, and provides forms for the entry of new data. The deletion process is secured by the inclusion of confirmation prompts and backend verifications to avert the inadvertent removal of critical resources. Additionally, the system accommodates the bulk management of events, rooms, and users, thereby enhancing the efficiency of administrative functions.

4.8   Find Available Functionality

The "Find Available" functionality is executed within the CalendarController and CalendarServiceImpl. Users are enabled to query for available rooms or users by specifying relevant tags, date, start, and end times, as well as the desired duration. The backend systematically processes these inputs, computes unoccupied time slots using the generateAvailableTimeRequest method, and subsequently provides a list of available resources. The returned results include direct links to observe availability within the calendar, thus significantly streamlining the scheduling of meetings or events in environments characterized by high demand for bookings. This particular functionality is notably absent from the existing scheduling system implemented at my University.

4.9 Installation and Deployment tutorial

The installation of this project was made simple and straightforward.

Firstly, install docker. This can be done either from the official website or from the package manager of your operating system. For example, on Ubuntu, you can install docker using the following commands (From the official website [3]):

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
↪   /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo \
 "deb [arch=$(dpkg -print-architecture)
 ↪   signed-by=/etc/apt/keyrings/docker.asc]
 ↪   https://download.docker.com/linux/ubuntu \
 $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}")
 ↪   stable" | \
 sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io
↪   docker-buildx-plugin docker-compose-plugin git
```

This will install git, docker and docker-compose.

Note: If you are using a different operating system, you can find the installation instructions on the official website.

Secondly, clone the repository. This will be done using git that was installed in the previous step.

```
git clone https://github.com/KirillBorodinskiy/TeamJob.git
```

Thirdly, build and run the project. This will be done using docker-compose that was pre-configured to handle all tasks and dependencies automatically.

```
cd TeamJob
docker-compose up
```

The application will be accessible through a web browser at `http://localhost:8080` upon successful deployment. The default port configuration can be modified within the `docker-compose.yml` file to accommodate specific network requirements or port conflicts. This containerized deployment approach ensures consistent execution environments across different operating systems and eliminates the need for manual dependency management.

For development and customization purposes, the application can be executed without Docker containerization, though this approach necessitates additional configuration and setup procedures. This alternative deployment method requires the installation of Java Development Kit (JDK) version 21, which can be obtained from the official Oracle website or through package managers such as Homebrew on macOS or apt on Ubuntu systems. The local execution process involves navigating to the project directory and utilizing the Gradle wrapper to compile and run the application:

```
cd TeamJob
./gradlew build
```

This non-containerized approach requires manual configuration of the PostgreSQL database, including the establishment of database credentials, creation of the database instance, and configuration of connection parameters within the application properties. Additionally, users must ensure that all required environment variables are properly set, including database connection strings, authentication secrets, and application-specific configurations. This deployment method is particularly suitable for developers who require direct access to the application source code for customization, debugging, or integration with existing development workflows.

4.10   Usage tutorial

The application provides an intuitive and comprehensive user interface designed to facilitate efficient scheduling and resource management. The system implements a multi-layered authentication mechanism and offers extensive functionality for calendar management, event creation, and administrative operations.

The authentication system utilizes JSON Web Tokens (JWT) to facilitate secure session management throughout the application. User registration is allowed through multiple endpoints (`/signup` and `/register`), with authentication processes available via `/login`, `/signin`, and the primary endpoint `/`. This redundant routing approach ensures ease of access regardless of the user's point of entry. Following successful authentication, the system issues a cryptographically secure JWT token, which is retained in the browser's cookie storage, thereby providing persistent session management while upholding security through token expiration and validation. The implementation includes comprehensive error handling mechanisms for invalid credentials and session expiration.

The primary calendar interface at `/calendar` functions as the central operational hub, allowing users to visualize, create, and manage events through an intuitive interface. This interface accommodates both weekly and daily perspectives, allowing users to traverse different temporal views. Detailed event information is accessible via the `/event/{id}` endpoint, which provides comprehensive data, including participant details, room allocations, and related metadata. Information specific to rooms can be obtained through the `/room/{id}` endpoint, which provides extensive room specifications, availability patterns and organizational tags.

The centralization of administrative functionality is achieved via the hierarchy of the `/config` endpoint. The `/config/rooms` endpoint provides capabilities for room management, facilitating administrators to view, create, and modify room configurations that include capacity specifications, equipment details, as well as organizational tags. The `/config/events` endpoint provides event management functionality, including reviewing scheduled events, modifying event details, and managing event-specific metadata. In addition, user management through `/config/users` grants administrative access to user account information, role assignments, and user-specific configurations. This administrative interface supports role-based access control, allowing system administrators to allocate appropriate permissions and organizational roles. The system employs comprehensive audit trails for all administrative operations, ensuring accountability and offering detailed logs for purposes related to security and compliance.

The `/calendar/findAvailable` endpoint enables the identification of available resources by allowing users to apply specific criteria such as temporal constraints, resource tags, and duration requirements. This capability is particularly advantageous in contexts characterized by high resource contention, as it allows users to optimally identify appropriate time slots and resources. In addition, the system offers extensive filtering capabilities, allowing users to concentrate on designated subsets of events and resources. Users have the ability to filter calendar views based on user identifiers, room specifications, and organizational tags, creating personalized views that improve the salience of pertinent information while mitigating cognitive load. The realization of this filtering functionality is facilitated through the integration of front-end interface controls alongside back-end query optimization processes; thus, it should maintain performance efficiency even while handling large datasets.

Chapter 5

Discussion

The development of TeamJob represents a significant undertaking in modern web application development, but the project's true value lies not merely in its technical implementation, but in the insights it provides about software engineering practices, architectural decision-making, and the challenges of creating organizational tools. This discussion critically examines the project's successes, limitations, and broader implications.

5.1 Architectural Decisions and Their Implications

Spring Boot selected (Section 5.7); MVC structure clarified, service layer introduced (Section 5.6); JWT authentication implemented (Section 5.2); calendar system designed for RFC 5545 (Section 5.3); PostgreSQL and H2 configured (Section 5.4); Thymeleaf-based UI chosen (Section 5.5); comprehensive testing applied (Section 5.6); technology stack evaluated (Section 5.7).

5.2 Security Implementation

The implementation of JWT-based authentication represents a modern, state-less approach to authentication. This was further intensified by the reluctance to manually manage sessions or establish a dedicated system for this purpose. The issue of improper configuration of the filter chain has frequently emerged; however, with the current setup, such complications should no longer arise during application usage.

Another potential issue is the implementation of distinct account storage and logging systems. This approach may not be compatible with organizations that employ a single account per individual across all their systems. This challenge could be addressed by extensively modifying the source code, given that the program is entirely open to modification and redistribution.

5.3 Calendar Functionality

The calendar system represents the project's most complex component, and its implementation reveals both the power and limitations of the chosen approach. The RFC 5545 compliance for recurring events, while technically correct in the backend implementation, introduced significant complexity that affected the frontend display development. The availability calculation algorithms, while efficient for moderate datasets, may not scale to enterprise-level deployments with thousands of users and events.

The powerful filtering system offers users the ability to identify the most suitable schedule. However, the challenge lies in the extensive filtering required. An idea emerged later in the project to establish predefined tag groups tailored for specific student demographics, such as first-year students in the XYZ program. Students would have the opportunity to adjust these tags to meet their specific requirements. The implementation of this feature has been canceled due to time constraints.

5.4 Database Design

Employing a PostgreSQL database within a Docker container was a strategically beneficial decision; however, it introduced a certain level of complexity when working within a local environment utilizing an H2 database. Considerable time was devoted to the issues of `application.properties` and `application-test.properties`. Eventually, the configuration was adapted to utilize the `.env` file for the deployed environment, while locally-selected variables were employed for test cases.

The current method for implementing the `user_roles` table and "tags" tables is functional. However, it may ultimately complicate the comprehension of the entire system and its operational mechanics.

5.5 User Experience / User Design

The user interface design prioritizes functionality over aesthetics, which is appropriate for an organizational tool, but may limit adoption in environments where user experience is of the utmost importance. The use of a Thymeleaf-based methodology ensures robust performance and security; however, it restricts the level of dynamic interactivity anticipated by modern users. This limitation could be addressed by transitioning to the currently prevalent JavaScript frameworks. However, the author lacked the requisite expertise in such frameworks, and temporal constraints precluded this transition.
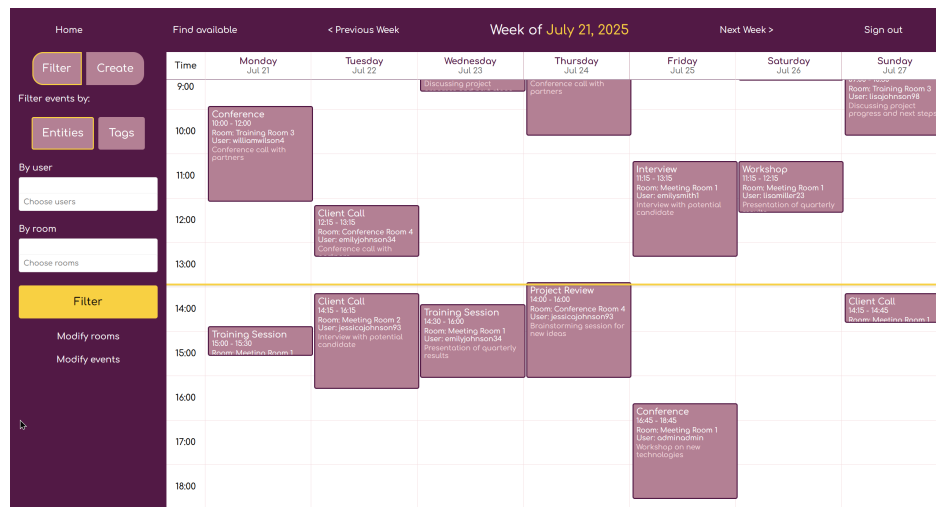

Figure 5.1: Weekly calendar view interface of the TeamJob application displaying scheduled events in a grid format.

The creation of an effective calendar interface presents the challenge of displaying scheduling data in a way that is both clear and user-friendly. Feedback from an expert within the design domain has been positively received regarding the UI elements 5.1 of the application, particularly applauding the selection of color schemes and the intuitive nature of the navigation system.

5.6 Testing Strategy

The testing methodology employed within TeamJob was characterized by a comprehensive and detailed examination that included the service, controller, and repository layers.

Test Data Management: Each test class independently configures and dismantles its data to ensure both isolation and repeatability. Helper methods are used to create realistic test entities, and the database is meticulously cleaned before and after each test to prevent cross-test contamination.

Service Layer Testing: The `CalendarServiceImplTest` class employs JUnit 5 and Mockito to thoroughly evaluate the business logic inherent in the calendar service. The service logic is effectively isolated through the injection of mock repositories, and comprehensive test data is generated for users, rooms, and events. The testing protocol covers various scenarios, such as event conversion, recurrence rules, filtering, and overlapping events, to verify that the core scheduling logic functions correctly.

Controller Layer Testing: The `RestConfigControllerTest` class employs Spring Boot's `@SpringBootTest` and `@AutoConfigureMockMvc` to execute integration testing on REST endpoints. These assessments simulate authentic HTTP requests through the utilization of MockMvc,

validating authentication, authorization, and accurate handling of event creation, including the back-end processing of recurring events. JWT tokens are systematically generated for a variety of user roles to evaluate access control mechanisms. The validation procedures ensure that actions are executable exclusively by users with the necessary roles and that the API delivers precise responses to both legitimate and illegitimate requests.

Repository Layer Testing: The `EventRepositoryTest` and `RoomRepositoryTest` classes utilize `@DataJpaTest` to assess the persistence layer through an in-memory database. Validates custom query methods designed to locate events based on time, room, title, and tags, as well as to detect overlapping events. These tests verify the precision of the repository logic and the ability of the database schema to facilitate the necessary queries.

Table 5.1: Test Suite Results Summary

Test Category	Pass Rate
Application Context Tests	100% (1/1)
Controller Tests	100% (9/9)
Repository Tests	100% (12/12)
Service Tests	100% (25/25)
Total	**100% (47/47)**

5.7 Technology Choices

The technology stack represents a solid choice for the project requirements, but each component introduces both benefits and trade-offs. Spring Boot eases developer productivity and offers a wide array of extensions along with support for various features. Despite its extensive customization capabilities, alternative back-end frameworks may present configuration options that are more straightforward. PostgreSQL is the highly popular database for modern applications. It is often used for both small-scale applications and enterprise applications.[4, 8]

The use of Thymeleaf as a templating engine guarantees substantial security and performance. However, it limits the dynamic user experience that is often required by modern applications. In contrast, contemporary JavaScript frameworks enable the development of more dynamic pages, facilitating easier reuse of design patterns and additional features.

Chapter 6

Conclusion

This thesis aimed to design and implement a self-hosted, open-source team scheduling system specifically suited for organizations that prioritize privacy, flexibility, and control over their scheduling data. Through an analysis of existing solutions and their associated limitations, the project defined clear objectives: design a user-friendly, customizable, and secure calendar application that employs modern Java web technologies and has necessary features for a team scheduling system.

The selection of Spring Boot, Thymeleaf, and PostgreSQL provided a robust foundation, effectively balancing ease of development, security, and scalability. The implementation adhered to best practices in software architecture, using a modular MVC pattern and integrating JWT-based authentication to ensure secure stateless user sessions. The resultant system offers a comprehensive suite of features, including advanced event management, conflict detection, filtering, and a flexible tagging system, all accessible via an intuitive Web interface.

Extensive testing at the service, controller and repository levels confirmed the reliability and correctness of the system. The project's discussion underscored both the strengths and trade-offs of the selected technologies, as well as the challenges encountered in areas such as:

- UI interactivity,

- Database configuration,

- Recurring event backend logic (frontend display incomplete).

The successful achievement of these objectives yielded several significant outcomes. Primarily, the project facilitated a comprehensive understanding of Java web development libraries and their practical applications, realized through the evaluation, selection, and integration of technologies including Spring Boot, Thymeleaf, and PostgreSQL.

The outcome is a functional application designed for team work organization, enabling efficient team collaboration, providing intuitive event management, ensuring secure system interactions, and offering a responsive and user-friendly interface.

Furthermore, the project created detailed documentation covering the technology evaluation and selection process, system architecture and design decisions, implementation specifics, testing methodology, and results, which serves as an essential reference for subsequent development and analogous endeavors.

Finally, the project provided practical experience in Java web application development, database design and implementation, user interface development, and system testing and validation, enhancing skills in both technical expertise and project management.

Although the application achieves its primary objectives, certain limitations persist. The user interface, although functional, could benefit from enhanced dynamism and modern design patterns. Scalability for very large organizations and further automation in tag management present opportunities for future enhancement. Additionally, easy integration with external authentication systems can enhance usability and promote broader adoption.

In conclusion, this work provides a solid technical foundation and practical insights for future development, contributing both a functional solution and a reference point for similar ventures in the domain of collaborative scheduling systems.

Bibliography

[1] Amigoscode. *Spring Boot 3 + Spring Security 6 - JWT Authentication and Authorisation [NEW] [2023]*. Accessed: 2024-05-09. 2023. URL: https://www.youtube.com/watch?v=KxqlJblhzfI.

[2] B. Desruisseaux. *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*. Tech. rep. RFC 5545. Accessed: 2024-05-01. Internet Engineering Task Force, 2009. URL: https://www.rfc-editor.org/rfc/rfc5545.html.

[3] Docker Inc. *Install Docker Engine on Ubuntu*. Accessed: 2025-07-30. 2025. URL: https://docs.docker.com/engine/install/ubuntu/.

[4] Justin Ellingwood. *The benefits of PostgreSQL*. Accessed: 2024-05-01. URL: https://www.prisma.io/dataguide/postgresql/benefits-of-postgresql.

[5] Fireship. *Learn Docker in 7 Easy Steps - Full Beginner's Tutorial*. Accessed: 2024-05-09. 2021. URL: https://www.youtube.com/watch?v=gAkwW2tuIqE.

[6] Spring framework. *Auditing :: Spring Data JPA*. Accessed: 2024-05-01. URL: https://docs.spring.io/spring-data/jpa/reference/auditing.html.

[7] Mateusz Kozak. "Analysis of the Spring Boot and Spring Cloud in developing Java cloud applications." In: *Journal of Computer Sciences Institute* 27 (2023). Accessed: 2024-05-01. DOI: 10.35784/jcsi.3130.

[8] Gerrit Meyer. *Why PostgreSQL is One of the Best Databases for Modern Applications and Why We Use It*. Accessed: 2024-04-01. Feb. 2025. URL: https://heisenware.com/en/blog/why-postgresql-is-one-of-the-best-databases-for-modern-applications.

[9] Visual Computer Science. *What is Spring-Boot Framework? (explained from scratch)*. Accessed: 2024-05-01. 2023. URL: https://www.youtube.com/watch?v=LSEYdU8Dp9Y.

[10] Spring Team. *Accessing Data with JPA*. Accessed: 2024-05-01. 2024. URL: https://spring.io/guides/gs/accessing-data-jpa.

[11] Hatma Suryotrisongko, Dedy Puji Jayanto, and Aris Tjahyanto. "Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot." In: *Procedia Computer Science*. Vol. 124. Accessed: 2024-05-01. 2017. DOI: 10.1016/j.procs.2017.12.212.

[12] Shreyash Vardhan. *Benchmarking Django vs Spring Boot : A comparative study*. Accessed: 2024-05-01. July 2024. URL: https://medium.com/codex/benchmarking-django-vs-spring-boot-a-comparative-study-e233dcb047c2.