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.

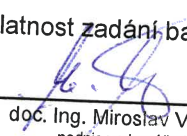Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Miroslav Virius, CSc.   katedra softwarového inženýrství   FJFI
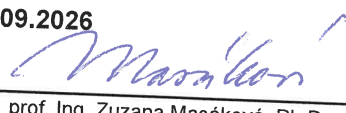
Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

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

Název práce:

Webová aplikace pro organizaci týmové práce

Autor: Kirill Borodinskiy

Studijní program: Celý název studijního programu (nikoliv zkratka)

Specializace: Celý název specializace (Pokud se studijní program nedlí na specializace, tuto ádku odstranit.)

Druh práce: Bakaláská práce

*Vedoucí práce:*doc. Ing. Miroslav Virius, CSc., DrSc., pracovit kolitele (název instituce, fakulty, katedry)

Abstrakt: Abstrakt max. na 10 ádk.

Klíová slova: klíová slova (nebo výrazy) seazená podle abecedy a oddlená árkou

Title:

Title of the Work

Author: Kirill Borodinskiy

Abstract: Max. 10 lines of English abstract text.

Key words: keywords in alphabetical order separated by commas

Contents

Chapter 1

Introduction

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.

Here we will talk about why the calendar is needed

Firstly, lets take a look at the current solution used by my university,CTU. Rozvrh.fjfi.cvut.cz is a website where students can see their schedule.

1.1 The current solution

It can be seen on 1.1 that while the website completes its main purpose, it is not customizable, which makes it hard to use. For example, if a student has a class that is from another year or/and program, they have to look at another picture and manually compare them. From my experience, many students have screenshots on their phones and they cross-out the classes that they are not registered to. They may have a few screenshots, for different programs or years. It is not a good solution, as it allows for misunderstandings and mistakes.
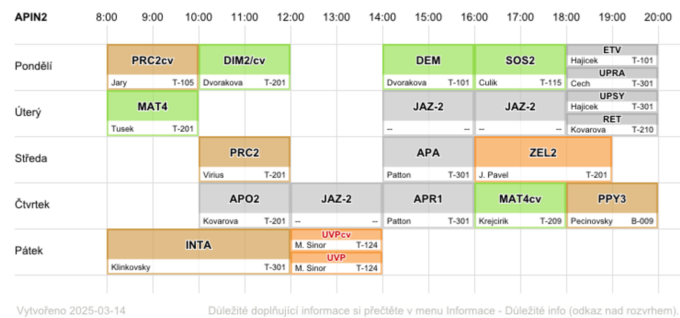
Figure 1.1: Screenshot of the current solution

This is why a project focused on creating a calendar that is easy to use and customizable is needed.
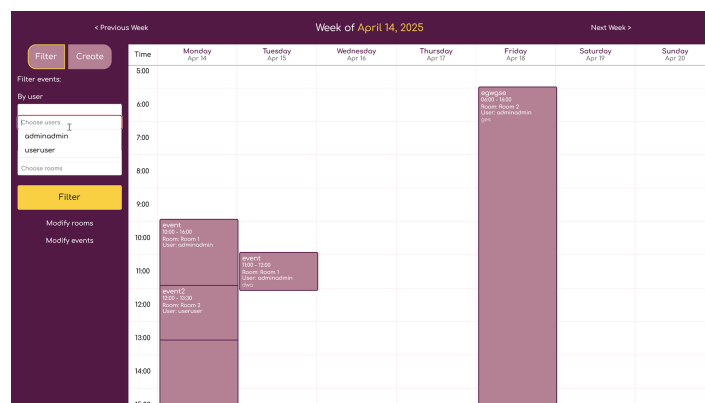
1.2 The solution

Figure 1.2: Screenshot of the new solution

TeamJob is a web application that allows users to filter exactly what they want to see. By default, all the events that are happening this week are shown. The user can filter the events by the room and the person that is assigned to this event. This allows for a more precise view of the calendar, where the user can see only the events that are important to them. The user can also move between weeks. A specific day-view allows the user to see the events that are happening on a specific day, if there are too many events to show them all at once.

Chapter 2

Methods

This section will focus on the tools and techniques used to create a web application tShreyash-Vardhan2024o manage time and space resources of an organization.

2.1 Defining the toolbelt

For the programming language, the options were as follows: Java Spring Boot framework, C#.NET, and Python's Django and Flask. C# was first removed, as learning about it would take a considerable amount of time. Python's Flask is easy to use, but not customizable enough. Subsequent analysis, as presented in the study [**ShreyashVardhan2024**], indicates that Spring Boot exhibits superior performance. Moreover, my familiarity with Java might reduce the time required for development. Consequently, Spring Boot is selected as the back-end framework. Spring Boot, which is constructed on the Spring Framework, is recognized as a leading framework within the Java ecosystem due to its widespread popularity. It streamlines the original Spring Framework, thereby facilitating more straightforward maintenance and expediting deployment procedures. Henceforth, to maintain clarity, the term Spring Boot shall be used exclusively in reference to both the Spring Framework and Spring Boot.

An often-utilized integration of back-end and front-end frameworks with Spring Boot is accomplished through Thymeleaf, a template rendering engine which processes page rendering on the server side, thereby reducing computational demand on the client-side systems. However, alternative solutions are available, including the currently prevalent React framework along with other JavaScript frameworks. Opting for these alternatives requires considerable investment in research. Given my proficiency in HTML, the Thymeleaf template system presents a straightforward learning curve. Nevertheless, a certain degree of JavaScript is essential for contemporary websites, thereby necessitating its use for handling specific tasks such as requests.

For our database, PostgreSQL was chosen, as it is a popular ACID-compliant database. ACID stands for:

- **Atomicity**: Transaction is either fully completed, or not, with no in-betweens.

- **Consistency**: Guarantees that a transaction brings the database from a valid state to a valid state.

- **Isolation**: Concurrent transactions do not interfere with each other.

- **Durability**: Once a transaction is committed, it stays committed.

2.2 Introduction to Spring Boot

Spring Boot is a tool that allows the programmer to create a web server that uses the Model-View-Controller pattern, MVC for short.

The model is a part responsible for the data logic. The connection to the database, the processing of the requested data and other back-end transactions are what this part consists of.

The view is a part that displays the data to a user or gathers them from them. Whether HTML, plain text, or any other format such as our Thymeleaf.

The controller is a connector between the previous two, where the data is additionally processed before being sent into either the database or a client of a user.
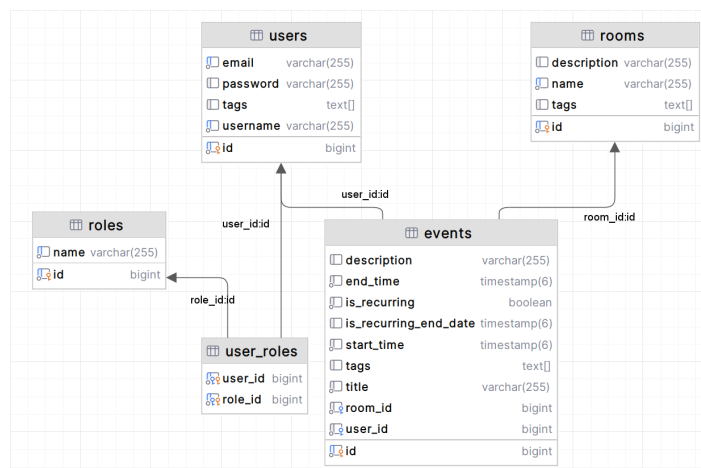
2.3 Schema of the database

Figure 2.1: Schema of the database for the application

Database 2.1 Architecture

- **users**: Stores user credentials and personal data. Each user has a unique ID.

- **roles** & **user_roles**: Implements role-based access control via a many-to-many relation between users and roles.

- **rooms**: Contains information on event locations. Each room has a `name`, `description` and a unique ID.

- **events**: Represents scheduled activities. Includes data such as `is_recurring`, `title`, `start_time` and `end_time`. Each event references a `user` and a `room` that were assigned to this event.

- **users, events** and **rooms** all contain `tags` to help categorize them.

2.4   The implementation of a Spring Boot application

In our application, the main controller is CalendarController. It renders the main page /calendar. The non-required parameters of a request are: "*date*", "*roomIds*" and "*userIds*".

Parameter "*date*" is simply the day of the week the calendar renders. If not provided, we use the default value of a current day.

Parameters "*roomIds*" and "*userIds*" are used to filter out which events the user wants to see in their calendar. If not provided, events from every room and every user are displayed.

In this controller a week around the current day is generated, all the events that are happening in that week are found and are put into an array that is then sent with the model.

2.4.1   Useful tools

In this application, *Lombok* was used to reduce the amount of boilerplate code. It allows for the usage of annotations such as @Getter,@Setter to automatically generate setters and setters for all fields that need them. In addition, annotations @AllArgsConstructor,@NoArgsConstructor can automatically create the correct construction function for the class. Finally, @Data combines @Getter,@Setter and some more functions in one annotation, so the classes remain clean and functional.

The use of the tool is demonstrated in the list 1. It can be seen that no setter or getter functions are needed, no construction function is needed, and the class looks clean and complete.

```java
@Data
@Entity
@Table(name = "rooms")
public class Room {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String name;

    @Column()
    private String description;

    @Type(io.hypersistence.utils.hibernate.type.array.ListArrayType.class)
    @Column(name = "tags", columnDefinition = "text[]")
    private Set<String> tags = new HashSet<>();
}
```

Listing 1: Lombok-annotated JPA entity

2.4.2 The architecture of the application

The main file responsible for most tasks is CalendarController, where the `/calendar` endpoint is located.

The `/calendar` takes 3 non-required parameters.

- **date**: Show the events that happen on the week of the day sent.

- **userIds**: Filter the users shown on the central calendar.

- **roomIds**: Filter rooms displayed in the central calendar.

Then, the `EventRepository` is used to find all the events that are happening in the week of the date sent. The `EventRepository` is a Spring Data JPA repository that allows for easy access to the database. They are split into a list of events that are happening on a specific day, where the days are filtered by comparing the `userIds` and `roomIds` with the events that are in the database. the data inside each event is such:

- **eventId**

- **eventTitle**

- **eventDescription**

- **eventStartTime** - the time when the event starts in LocalDateTime format.

- **eventEndTime**

- **eventRoomId** - the room that is assigned to this event.

- **eventUserId** - the user that is assigned to this event.

- **eventTags** - the tags that are assigned to this event.

- **eventIsRecurring** - if the event is recurring or not.

The `/calendar` then sends such data to `calendar.html`:

- **userIds** & **roomIds** (if supplied from request).

- **selectedDate** (if not provided, current date is used).

- **nextWeek** & **previousWeek** to enable navigation within weeks.

- **currentWeekStart**.

- **weekDays** - a list of events that take place on a specific day.

- **eventRepository**, **userRepository** and **roomRepository**

2.4.3 Additional Calendar Endpoints

The CalendarController also provides a day view endpoint at `/calendar/day`, which displays events for a specific day organized by room. This endpoint accepts the following parameters:

- **date** (required): The specific date to display events for.

- **userIds** (optional): Filter events by specific users.

- **roomIds** (optional): Filter events by specific rooms.

The day view provides a more detailed perspective of events occurring on a single day, organized by room. This is particularly useful when there are many events scheduled on a specific day, making the weekly view potentially cluttered.

Another important endpoint is `/calendar/findAvailable`, which allows users to search for available resources (rooms, users, or events) based on various criteria:

- **searchType**: Specifies whether to search for "rooms", "users", or "events".

- **tags**: Allows filtering by specific tags associated with the resources.

- **date**: The date to search for availability (defaults to current date if not provided).

- **startTime** & **endTime**: The time range to check for availability.

This functionality is particularly valuable for quickly identifying available resources during a specific time slot, facilitating efficient scheduling and resource allocation.

2.4.4 The findAvailable Algorithm

The findAvailable algorithm is a sophisticated process that identifies available time slots for rooms, users, or events within a specified time range. Here's how it works:

1. **Input Processing**: The algorithm accepts several parameters:

   - `searchType`: Determines whether to search for available rooms, users, or events
   - `tags`: Optional tags to filter resources by
   - `date`, `startTime`, `endTime`: Define the time range to check
   - `durationMinutes`: Minimum duration required for an available time slot

2. **Data Collection**: The system retrieves all events that overlap with the specified time range.

3. **Entity Filtering**: Based on the searchType, the algorithm filters entities (rooms, users, or events) by the provided tags.

4. **Availability Calculation**: For each entity, the algorithm:

   - Identifies all events associated with the entity
   - Creates a list of occupied time slots from these events
   - Sorts occupied slots by start time and merges any overlapping slots
   - Calculates unoccupied time slots by finding gaps between occupied slots
   - Filters unoccupied slots by the minimum duration requirement

5. **Result Generation**: The algorithm creates a list of available entities along with their unoccupied time slots.

2.4.4.1 Implementation Details

The key component of this algorithm is the `calculateUnoccupiedTimesFromOccupied` method, which efficiently identifies available time slots by:

1. Sorting occupied time slots by start time

2. Merging overlapping occupied slots

3. Iterating through occupied slots to find gaps (unoccupied times)

4. Filtering unoccupied slots by minimum duration

5. Sorting the resulting unoccupied slots

2.4.4.2 Handling Overlapping Time Slots

A critical aspect of the algorithm is the `removeOverlaps` method, which merges overlapping time intervals:

1. The method requires that time slots are already sorted by start time

2. It iterates through adjacent pairs of time slots

3. When two slots overlap (the end time of one is after the start time of the next), they are merged

4. The merged slot spans from the start time of the first slot to the end time of the second slot

5. The second slot is removed from the list, and the iteration index is decremented to recheck the newly merged slot

6. This process continues until no more overlaps exist

2.4.4.3 Calculating Unoccupied Time Slots

The algorithm identifies unoccupied time slots through the following process:

1. Start with the requested start time as the current point

2. For each occupied time slot (after sorting and merging):

   - If the current point is before the start of the occupied slot, add an unoccupied slot from the current point to the start of the occupied slot
   - Update the current point to the end of the occupied slot

3. After processing all occupied slots, if the current point is before the requested end time, add a final unoccupied slot from the current point to the requested end time

4. Filter the resulting unoccupied slots to include only those that meet or exceed the minimum duration requirement

2.4.4.4   Edge Case Handling

The algorithm handles several edge cases effectively:

- **Empty tag sets**: If no tags are specified, all entities of the requested type are included

- **No events**: If there are no events in the specified time range, the entire range is considered unoccupied

- **Completely occupied time range**: If the entire time range is occupied, no unoccupied slots are returned

- **Invalid entity type**: If an invalid entity type is provided, an IllegalArgumentException is thrown

2.4.4.5   Performance Considerations

Several optimizations enhance the algorithm's performance:

- **Early filtering**: Events are filtered by time range at the database level before further processing

- **Efficient tag filtering**: Uses `Collections.disjoint()` to quickly check for tag matches

- **Sorting before processing**: Ensures that time slot operations can be performed in a single pass

- **Stream operations**: Leverages Java streams for concise and efficient filtering operations

This approach ensures that users can quickly find available resources that meet their specific time and duration requirements, optimizing resource allocation and scheduling efficiency. The algorithm's modular design also allows for easy extension to support additional entity types in the future.

2.5   Authentication and Security

The application implements a robust security system using JSON Web Tokens (JWT) for authentication. The SecurityController handles the authentication process through several REST endpoints:

- `/auth/signin` (POST): Authenticates a user with username and password, generating a JWT token stored in a secure HTTP-only cookie.

- `/auth/signup` (POST): Registers a new user with username, email, and password. The password is securely hashed before storage.

- `/auth/signout` (POST): Logs out a user by invalidating their JWT cookie.

The JWT implementation enhances security by:

- Using HTTP-only cookies to prevent JavaScript access to the token

- Setting the Secure flag to ensure transmission only over HTTPS

- Implementing token expiration to limit the window of vulnerability

The MainController provides view endpoints for authentication-related pages:

- `/`, `/login`, `/signin`: All route to the sign-in page

- `/register`, `/signup`: Route to the registration page

- `/signout`: Routes to the sign-out page

2.6 Configuration and Administration

The application provides both view-based and REST API endpoints for configuration and administration.

The ConfigController offers view endpoints for managing system entities:

- `/config/rooms`: Lists all rooms in the system

- `/config/events`: Lists all events in the system

- `/config/users`: Lists all users in the system

For programmatic interaction, the RestConfigController provides REST API endpoints:

- `/api/v1/addrooms` (POST): Creates a new room

- `/api/v1/addevents` (POST): Creates a new event, with conflict checking

- `/api/v1/checkavailability` (POST): Verifies if a room is available during a specific time period

- `/api/v1/deleterooms/{id}` (DELETE): Removes a room by ID

- `/api/v1/deleteevents/{id}` (DELETE): Removes an event by ID

- `/api/v1/validateJWT` (POST): Validates the current JWT token

2.7 Error Handling

The application implements a CustomErrorController that provides a unified approach to error handling. When an error occurs, the controller captures details such as:

- Error status code

- Exception type

- Error message

- Request path that caused the error

- Stack trace (in development environments)

This information is then rendered in a user-friendly error page, enhancing the debugging process while maintaining a consistent user experience even when errors occur. This could be further improved by adding a system that would notify the administrator about the error, ensuring that it is addressed promptly.

Conclusion

Text of the conclusion...

Bibliography

[1] S. Allen, J. W. Cahn: *A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening*. Acta Metall., 27:1084-1095, 1979.

[2] G. Ballabio et al.: *High Performance Systems User Guide*. High Performance Systems Department, CINECA, Bologna, 2005. `www.cineca.it`

[3] J. Becker, T. Preusser, M. Rumpf: *PDE methods in flow simulation post processing*. Computing and Visualization in Science, 3(3):159-167, 2000.