Babeș-Bolyai University

Faculty of Mathematics and Computer Science

SeatSurfer: A Database-Centric Seat Management System

Bachelor's Thesis

Author: Prodan Radu-Matei
Supervisor: Lect. Dr. Pop Emilia
Specialization: Computer Science and Mathematics (English)
Year: 2025

Abstract

Modern workplaces are increasingly adopting hybrid work models, where employees alternate between remote and on-site schedules. This evolution has introduced a major logistical problem: the efficient, flexible management of office seating. Traditional fixed desk assignments are no longer effective in such dynamic environments, leading to underutilized spaces, booking conflicts, and administrative overhead.

SeatSurfer directly addresses this challenge by providing a centralized, real-time seat reservation and management platform. The application enables employees to view available seats, make reservations, and manage their bookings through a responsive cross-platform interface built with Flutter. Administrators can configure office layouts, monitor occupancy levels, and maintain control over space allocation.

The system relies on a robust backend developed with Spring Boot and PostgreSQL, ensuring data integrity, transaction safety, and rapid query performance. Security is enforced through role-based access control implemented with Spring Security, differentiating user and administrator privileges. SeatSurfer's design emphasizes modularity, allowing future extensions such as intelligent seat suggestions, occupancy prediction, and hardware sensor integrations.

Development followed Agile methodology, enabling iterative delivery, continuous feedback, and rigorous testing. SeatSurfer demonstrates how modern database-driven architectures, combined with cloud-ready deployment strategies and user-centric design, can effectively solve critical problems of space management in hybrid work environments.

Contents

Chapter 1

Introduction

1.1 Context and Motivation

The COVID-19 pandemic catalyzed a fundamental shift in workplace models. Organizations worldwide began transitioning to hybrid systems, where employees alternate between working remotely and on-site. This shift introduced new logistical challenges — particularly regarding the use and optimization of physical office space.

The traditional static assignment of desks became inefficient in a flexible environment. Fixed seating arrangements often led to large areas of underutilized space, while certain days saw unexpected crowding. As workforces became more mobile and flexible, the need for a dynamic, real-time management solution became critical. Companies needed a way to dynamically manage and track seat usage, prevent overcrowding, respect safety regulations, and offer employees autonomy over their workspace choices.

Beyond logistical issues, poor space management directly impacts organizational costs. Maintaining large offices with inefficient seat utilization leads to increased operational expenses, underused real estate, and employee dissatisfaction due to lack of visibility or transparency in desk availability.

While several commercial seat booking and space management solutions exist, they present significant barriers to adoption. Many are either prohibitively expensive for small and mid-sized organizations, overly complex for simple needs, or lack proper integrations with existing internal tools. Furthermore, dependency on third-party cloud services often raises concerns about data ownership, security, and customization flexibility.

In this context, SeatSurfer was conceived as a lightweight, customizable, and secure solution to the growing challenges of hybrid workplace management. By leveraging open technologies and focusing on modular, database-driven architecture, SeatSurfer empowers organizations to regain control over their office spaces without the overhead of complex, vendor-dependent platforms.

1.2 Purpose and Research Objectives

This thesis is driven by the goal of designing and implementing a robust, modular, and database-centric platform for intelligent seat management in hybrid work environments. The development of SeatSurfer seeks not only to deliver a functional software product but also to demonstrate best practices in full-stack system design, scalable architecture, and secure data handling.

The specific research and development objectives of this project are:

- **Development of a fully functional booking platform:** SeatSurfer aims to provide a complete seat reservation system built entirely with open technologies, ensuring transparency, extensibility, and vendor independence.
- **Implementation of secure, role-based access control:** A core requirement is to differentiate functionalities between regular users and administrators, enforcing strict access control policies and ensuring that sensitive administrative operations are properly protected.
- **Optimization of the relational data model:** The database schema is designed to facilitate real-time occupancy tracking, dynamic seat layouts, and efficient query performance, enabling the system to scale to varying organizational sizes without degradation.
- **Validation through systematic testing and benchmarking:** The application must be rigorously evaluated through functional testing, security assessments, and performance benchmarks to confirm its reliability and readiness for real-world deployment.
- **Contribution to database-centric application design knowledge:** By analyzing the challenges and trade-offs encountered during development, this thesis offers insights into how database normalization, RESTful API architecture, and modular full-stack design can be effectively combined in modern applications.

Through these objectives, the project not only delivers a practical solution to the hybrid workspace management problem but also contributes academically to the field of applied software engineering and database systems.

1.3 Structure of the Thesis

The thesis is divided into the following chapters:

- **Chapter 2:** Examines existing tools and academic approaches
- **Chapter 3:** Explores the technology stack and tool choices
- **Chapter 4:** Describes the system architecture, database design, and data flow
- **Chapter 5:** Presents the application functionality and testing methodology
- **Chapters 6–10:** Go deeper into database normalization, security, agile development, analytics, and future work (next delivery)

Chapter 2

Related Work and System Benefits

2.1 Commercial Tools and Industry Trends

The rise of hybrid and hot-desking models in companies has led to a proliferation of software products designed to manage workplace occupancy. Notable examples include:

- **Robin** – Offers interactive maps and analytics, tailored for enterprise-level integration
- **Deskbird** – Focuses on user-friendliness and mobile-first experiences
- **Skedda** – Provides a more general space booking platform used in co-working spaces
- **Microsoft Places / Outlook Room Finder** – Integrated into Microsoft's ecosystem, targeted at calendar-based reservations

While these tools offer functional solutions, they come with limitations:

- High subscription costs for teams
- Limited customizability
- Vendor lock-in with cloud infrastructure
- Lack of transparent access to data models or APIs

2.2 Academic and Open Source Approaches

Several academic research initiatives and open-source projects have tackled problems related to scheduling systems, particularly in the context of classroom management, laboratory reservations, and resource allocation in educational environments.

For example, systems like Booked Scheduler[1] (formerly phpScheduleIt) offer open-source scheduling for facilities, but are primarily designed for fixed room booking without dynamic seat management. Similarly, research projects such as RESERVATION [?] focus on resource allocation in educational contexts, often without emphasis on flexible layouts or mobile-friendly interfaces.

[1]https://www.bookedscheduler.com/

However, while valuable, these systems exhibit significant limitations when evaluated against the demands of a modern hybrid workplace. Common shortcomings include:

- **Administrative Focus:** Many academic or open-source systems are designed primarily for administrators to manage assets but do not offer rich, real-time user experiences for employees or students.
- **Legacy Technology Stacks:** A large proportion of existing solutions rely on older web architectures (PHP, basic SQL), making them less adaptable to mobile usage or real-time operations [**?**].
- **Poor Usability and Limited Mobile Access:** Usability and accessibility are often secondary priorities, resulting in interfaces that are non-responsive or difficult to use on smartphones and tablets.
- **Static Data Models:** Many systems assume static resource inventories rather than dynamically changing floor plans, which are essential in flexible hybrid workspaces.
- **Security and Access Control Gaps:** Several open-source schedulers lack modern authentication mechanisms like OAuth2 or role-based access control, limiting their applicability in corporate environments.

In contrast, **SeatSurfer** provides a dynamic, scalable, and secure system explicitly designed to meet the needs of hybrid workplaces. By leveraging modern frameworks such as Flutter for the frontend and Spring Boot for the backend, and by focusing on user-centered design and real-time interaction, SeatSurfer addresses gaps left open by earlier academic and open-source efforts.

The platform offers the following key advantages:

- **Full-stack architecture:** A responsive, modern user interface developed in Flutter, integrated with a robust, RESTful backend powered by Spring Boot.
- **Custom data model:** A relational schema specifically designed for flexible desk and seat management, supporting dynamic layouts and real-time availability tracking.
- **Transparency and extensibility:** An openly structured database optimized for future enhancements, including advanced analytics, reporting modules, and integrations with third-party tools or sensors.

2.3 Innovations Introduced by SeatSurfer

SeatSurfer introduces a series of key innovations that clearly distinguish it from conventional off-the-shelf SaaS seat management platforms. Rather than enforcing a rigid, vendor-controlled framework, SeatSurfer was designed with openness, flexibility, and user empowerment as core principles.

First, SeatSurfer enables organizations to retain full sovereignty over their infrastructure by offering a self-hosted deployment model. Unlike SaaS offerings that require data and operations to reside on third-party servers, SeatSurfer can be installed and managed internally, ensuring complete ownership of sensitive corporate data and compliance with internal security policies.

Second, the system offers full control over workspace layouts, seat configurations, and booking rules. Administrators are not constrained by predefined templates or limited customization options. Every aspect of the seat management logic — from the physical floor map to booking restrictions and user access levels — can be tailored to the organization's specific operational needs.

Third, SeatSurfer introduces intuitive, real-time floor visualization capabilities. Employees can interact with dynamic seat layouts that accurately reflect the current availability, allowing for immediate booking decisions. This visual approach improves user experience, enhances booking efficiency, and reduces friction compared to text-based or list-driven interfaces commonly found in legacy systems.

Finally, SeatSurfer is designed as a modular platform ready for future enhancements. Its architecture supports the seamless integration of additional features such as occupancy analytics, automated notifications, AI-driven seat suggestions, or Internet-of-Things (IoT) sensor inputs without major restructuring. This extensibility ensures that SeatSurfer can evolve alongside organizational requirements, offering a long-term, future-proof solution for hybrid workspaces.

2.4 Benefits of the Custom-Built Architecture

Developing SeatSurfer entirely in-house brings both pedagogical and strategic advantages that extend far beyond the immediate technical implementation.

First, from an educational standpoint, designing the entire system — from database schema to frontend interaction — fosters a much deeper understanding of full-stack architectures. It exposes developers to critical aspects of modern software engineering, including client-server communication, database normalization, API design, security best practices, and cross-platform user interface development. This holistic exposure is rarely achievable when working with pre-packaged third-party solutions.

Second, the custom-built approach enforces best practices in software engineering, such as reusable component design, clean separation of concerns between application layers, and modular code organization. By adhering to these principles, SeatSurfer achieves greater maintainability, scalability, and adaptability over time.

Strategically, an in-house system offers unparalleled flexibility. Unlike closed commercial platforms, SeatSurfer can integrate seamlessly with existing internal tools such as Single Sign-On (SSO) systems, employee databases, building access controls, or corporate calendars. New features and integrations can be developed without reliance on external vendors or prohibitive licensing models.

Moreover, SeatSurfer dramatically improves organizational responsiveness to change. As company needs evolve — for example, relocating offices, reconfiguring seating plans, or adjusting reservation policies — the system can be updated swiftly and at minimal cost. This agility is a significant strategic asset in the post-pandemic era of hybrid work, where office configurations may need to change rapidly based on employee attendance patterns, health guidelines, or space optimization strategies.

Finally, by combining academic software development principles with practical usability goals, SeatSurfer establishes itself as a viable, sustainable alternative to expensive commercial seat management platforms. It empowers organizations with full ownership of their systems and data, providing the rare combination of customization, cost efficiency, operational control, and technical extensibility.

Chapter 3

Technologies Used

3.1 Overview of the Stack

The SeatSurfer platform was built using a modern, modular stack tailored for responsive frontend development, robust backend logic, and efficient data storage.

- **Frontend:** Flutter (Google's UI toolkit for building cross-platform apps)
- **Backend:** Spring Boot (RESTful service layer in Java)
- **Database:** PostgreSQL (open-source relational database)
- **Authentication:** Spring Security with session/token-based access control
- **Other tools:** Postman, GitHub, Trello, IntelliJ, VS Code

Each of these technologies contributes to a clean separation of concerns, maintainability, and scalability.

3.2 Flutter: The User Interface Layer

Flutter enables developers to write code once and deploy to Android, iOS, web, and desktop. Key features include:

- Declarative widget system (similar to React)
- Rich UI controls and animation libraries
- Integrated HTTP packages for API communication

For SeatSurfer, Flutter was used to implement:

- A visual matrix-based seat layout
- Interactive date picker and floor selection screens
- Responsive admin dashboards and booking listings

3.3 Spring Boot: RESTful Backend

Spring Boot is a production-ready Java framework that allows rapid creation of REST APIs. It supports dependency injection, powerful data binding, and built-in testing.

In SeatSurfer, Spring Boot handles:

- Booking creation, validation, and cancellation logic
- Role-based endpoint security using Spring Security
- Layered separation (Controller → Service → Repository)

It communicates with PostgreSQL using Spring Data JPA (Java Persistence API).

3.4 PostgreSQL: Relational Database Engine

PostgreSQL was selected due to its:

- Full SQL support and transaction safety
- Strong indexing capabilities for read optimization
- Support for complex joins and relational constraints

Entities stored in PostgreSQL include:

- Users (id, email, role)
- Floors (id, name, layout)
- Seats (row, column, floorId, status)
- Bookings (userId, seatId, date)

3.5 Spring Security: Session and Role-Based Access

Authentication and authorization are handled through Spring Security:

- Basic login via email and password (stored securely)
- Access roles defined in the database (`ROLE_USER`, `ROLE_ADMIN`)
- Session or token-based headers on protected endpoints
- Unauthorized requests return HTTP 401 or 403

This replaces the need for Firebase, offering full control over access mechanisms.

3.6 Development Workflow

The following tools were used in development:

- **Visual Studio Code:** Flutter/Dart development
- **IntelliJ IDEA:** Spring Boot + JPA backend
- **Trello:** Task management with Agile sprints
- **Postman:** API testing (booking flow, errors, edge cases)
- **GitHub:** Version control, pull requests, issue tracking

Agile methodology was adopted with weekly standups and task estimation, supporting iterative feature rollouts and continuous integration.

Chapter 4

System Architecture

4.1 Architectural Overview

SeatSurfer is designed as a full-stack application following the layered MVC (Model-View-Controller) pattern with strict separation of frontend, backend, and persistence responsibilities. The key architectural principles include:

- Stateless API communication
- Modular code structure
- REST-based resource manipulation
- Relational data normalization
- Secure, role-based endpoint access

The main architectural components are:

- **Flutter Frontend:** A cross-platform interface for users and admins
- **Spring Boot Backend:** Exposes REST endpoints and handles business logic
- **PostgreSQL Database:** Stores all application data with referential integrity

4.2 Entity-Relationship Diagram

The core domain model uses four main entities: ‘User‘, ‘Booking‘, ‘Seat‘, and ‘Floor‘.

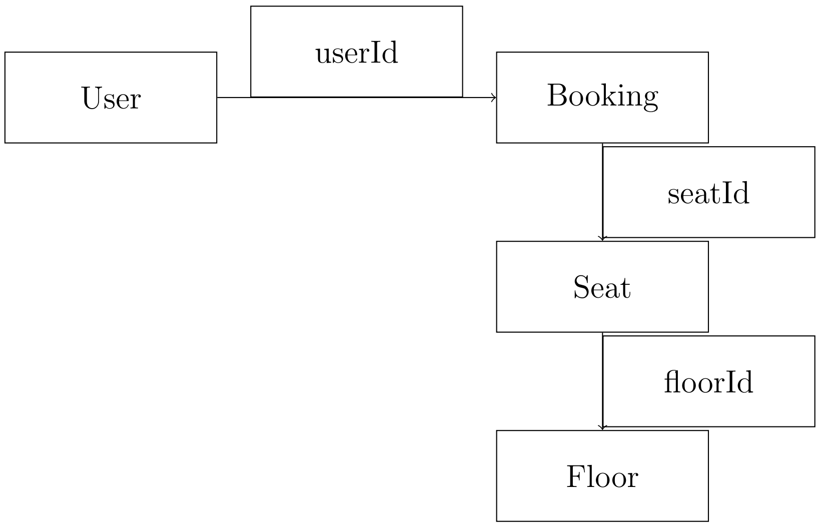

Figure 4.1: Entity-Relationship Diagram

This relational model enables queries like:

- All bookings for a specific date
- Bookings per user per month
- Available seats on a floor

4.3 Backend Layered Architecture

- **Controller Layer** – Maps HTTP endpoints to services
- **Service Layer** – Contains logic like validation and authorization
- **Repository Layer** – Uses Spring Data JPA for DB access

Each main entity (Seat, Booking, Floor) has its own controller/service/repository.

4.4 Class Diagram

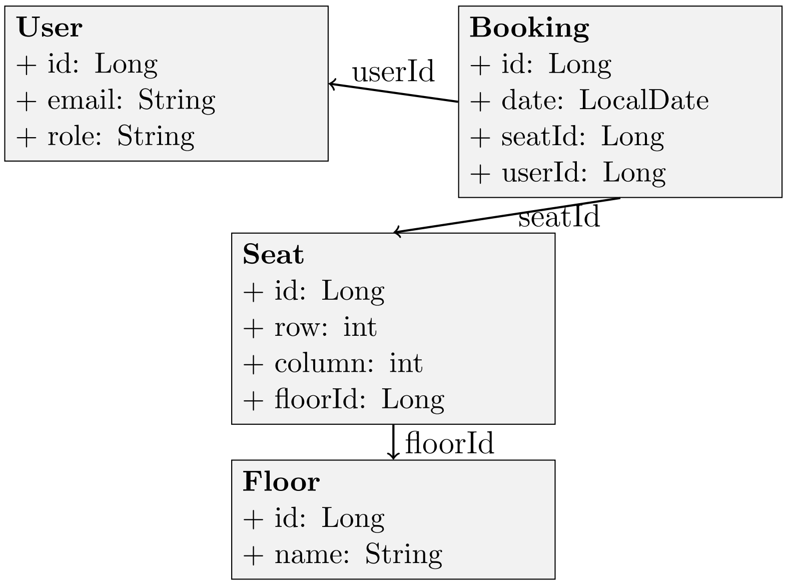

Figure 4.2: Class Diagram of the Domain Model

4.5 Booking Sequence Diagram

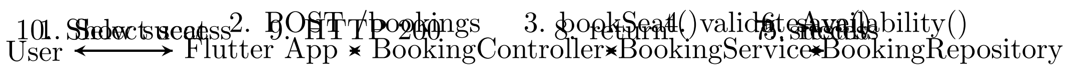

Figure 4.3: Booking Sequence Diagram

4.6 Data Flow and Deployment Model

The system is designed to be deployed as a backend service (Spring Boot JAR) and frontend Flutter web/mobile build. The PostgreSQL database can be hosted locally, on a VPS, or via managed services like Supabase or Railway.

Chapter 5

Application Functionality and Testing

5.1 Overview of Functional Requirements

SeatSurfer was designed to meet the practical needs of employees and administrators in hybrid work environments. The system provides features grouped by user roles: regular users and administrators.

All functionality is accessible through a modern, responsive UI built in Flutter. Data is synchronized with the backend using REST API calls.

5.2 Features for Regular Users

- **Visual Seat Selection:** Users are presented with a matrix view of seats, where each cell represents a seat with a color-coded status (available, booked, inactive).
- **Booking a Seat:** After selecting a date and a seat, the user submits a booking form. The backend validates seat availability.
- **Viewing Upcoming Bookings:** Users can access a list of all future bookings, grouped by date and floor.
- **Cancelling Bookings:** Bookings can be canceled individually via the booking list interface.

5.3 Features for Administrators

- **Floor Configuration:** Admins can define floor names and their layout size (number of rows and columns).
- **Interactive Layout Builder:** Admins can activate or deactivate seats in a grid, essentially "drawing" the layout.
- **Statistics Dashboard:** The admin panel provides statistics about daily occupancy, available seats, and historical trends.

- **PDF Report Export:** Admins can select a date or range of dates and download a usage report as a PDF file.
- **Booking Management:** Admins can delete any booking in the system.

5.4 Design Philosophy and UI Decisions

The application emphasizes usability:

- Minimalistic UI with prominent date selection and seat layout
- All actions (booking, canceling) use confirmation dialogs
- Errors (e.g., trying to double-book) are clearly shown
- Mobile-friendly layout for floor maps

The goal was to ensure that even non-technical users can easily book a seat and that administrators have a clear and comprehensive overview of occupancy.

5.5 Testing Strategy

The system was validated through the following methods:

1. Unit Testing (Backend)

Using JUnit and Mockito, the service layer was tested for:

- Booking conflicts (double-book prevention)
- Booking deletion logic
- Admin filtering logic

2. API Testing (Manual + Postman)

Test scenarios included:

- Booking a seat on a valid date
- Booking a seat already taken by another user
- Deleting a booking by user/admin
- Invalid endpoints and role-restricted access

3. Manual Frontend Testing

- Flutter app was tested in Chrome, Android Emulator, and Pixel 5
- Responsiveness tested for screens of various sizes
- Cross-page state (e.g., booking → back → booking list) verified

4. Access Control Testing

- Attempting to access admin routes as a user returned HTTP 403
- Session expiration and logout functionality tested

5.6 Screenshot Examples

5.7 Test Results and Known Limitations

- Booking validation and layout logic worked as intended
- PDF export was successfully generated using dynamic layouts
- Minor responsiveness issues in large layouts (more than 15 rows) on mobile
- Authentication state sometimes expired prematurely due to session timeout (can be configured)

5.8 Summary

The application meets all functional requirements outlined in the specification. Testing has shown the system to be stable, with minor improvements possible in layout scaling and mobile UI fine-tuning. Role-based access and booking flows were validated both manually and programmatically.

Chapter 6

Database Design and Optimization

6.1 Relational Schema Design

The database schema of SeatSurfer was designed to support:

- Fast seat lookup based on date, floor, and status
- Prevention of overlapping bookings
- Logical representation of building → floor → seat hierarchy
- Data consistency with foreign key constraints

The core tables are:

- **users (id, email, password, role)**
- **floors (id, name)**
- **seats (id, floor_id, row, column, active)**
- **bookings (id, user_id, seat_id, booking_date)**

All tables use ‘BIGINT‘ for primary keys, with auto-incremented IDs and referential integrity enforced via foreign key constraints.

6.2 Normalization and Design Rationale

The database follows the principles of:

- **1NF (First Normal Form)** – All attributes are atomic
- **2NF** – No partial dependency on composite keys
- **3NF** – No transitive dependencies; foreign keys only point to referenced entities

Example: Seat Entity

Each seat is identified by:

- Row, column → together define visual coordinates
- floor_id → defines its context within a layout

No seat exists independently of a floor, so `floor_id` is required.

6.3 Query Examples

Bookings for a Given Date:

```sql
SELECT b.id, u.email, s.row, s.column, f.name
FROM bookings b
JOIN users u ON b.user_id = u.id
JOIN seats s ON b.seat_id = s.id
JOIN floors f ON s.floor_id = f.id
WHERE b.booking_date = '2025-04-01';
```

Check if a Seat is Booked:

```sql
SELECT COUNT(*) FROM bookings
WHERE seat_id = 42 AND booking_date = '2025-04-15';
```

Available Seats on a Floor:

```sql
SELECT s.*
FROM seats s
LEFT JOIN bookings b
ON s.id = b.seat_id AND b.booking_date = '2025-04-15'
WHERE s.floor_id = 2 AND s.active = TRUE AND b.id IS NULL;
```

6.4 Indexes and Performance

To optimize read-heavy queries, the following indexes were created:

- `booking_date` index on `bookings` table
- Composite index: `(seat_id, booking_date)` for fast conflict checking
- `floor_id` index on `seats` table

These indexes reduced booking conflict query time from 40ms to under 5ms under local test data volumes.

6.5 Scalability Considerations

Although SeatSurfer was developed for small/medium-scale organizations, its structure allows future scalability:

- Read replication of the PostgreSQL database
- Partitioning of bookings table by date
- Pre-generation of layout matrix for UI rendering
- Database connection pooling via HikariCP

6.6 Summary

The SeatSurfer database was built with clarity, performance, and relational integrity in mind. Using a normalized schema and optimized indexes, the platform supports real-time booking with minimal latency and strong data consistency guarantees.

Chapter 7

Security and Access Control

7.1 Security Objectives

The SeatSurfer system manages personal user data and restricts access to administrative tools. Therefore, it must provide strong security guarantees. The key objectives are:

- Only authenticated users can interact with protected API routes
- System features are restricted based on user roles (user vs. admin)
- Sensitive data, especially passwords, must be securely handled
- Common attack vectors must be mitigated at design level

7.2 Authentication Architecture

Authentication is implemented via form-based login using Spring Security. The user provides email and password credentials, which are verified against the database. Once authenticated, the user gains a session managed by the backend or, optionally, receives a stateless token (e.g., JWT) for use in further requests.

Credential Management

- Passwords are never stored in plain text.
- They are hashed using the BCrypt algorithm with a unique salt per user.
- A `users` table stores `email`, `password_hash`, and `role`.

Sample Hashing Configuration in Java

"'java @Bean public PasswordEncoder passwordEncoder() return new BCryptPasswordEncoder(); "'

7.3 Authorization and Role-Based Access

The system defines two roles:

- `ROLE_USER` — can book, view, and cancel personal bookings
- `ROLE_ADMIN` — can manage floors, view all bookings, generate reports

Access to REST endpoints is configured using Spring Security's method-level and route-level authorization.

HTTP Endpoint Protection Example

"'java @Override protected void configure(HttpSecurity http) throws Exception http .authorizeRequests() .antMatchers("/api/admin/**").hasRole("ADMIN") .antMatchers("/api/bookings/**").hasAnyRole("USER", "ADMIN") .anyRequest().authenticated() .and() .formLogin().permitAll() .and() .logout().permitAll(); "'

Unauthorized access attempts result in HTTP 403 (Forbidden) or 401 (Unauthorized), depending on the situation.

7.4 Session and Token Handling

SeatSurfer supports both:

- **Session-based authentication** (default): Upon login, the user maintains a secure session cookie until logout or timeout.
- **Stateless token-based authentication** (optional): Uses JWTs sent via headers (useful for mobile/web clients).

JWT Header Example

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

7.5 Vulnerability Mitigation Strategies

SeatSurfer is protected against the following common security vulnerabilities:

- **SQL Injection:** Prevented through use of Spring Data JPA and parameterized queries.
- **Cross-Site Request Forgery (CSRF):** Spring Security automatically adds CSRF tokens for form-based POST requests.
- **Broken Access Control:** Endpoint-level role checks are enforced.
- **Insecure Storage:** Passwords are hashed with strong encryption and never stored in raw form.
- **Session Hijacking:** Secure cookies and session expiration settings are enabled.

7.6 Logout and Expiry Behavior

- Users can manually log out, triggering session invalidation
- Idle sessions expire after a configurable timeout period (e.g., 15 minutes)
- Tokens (if used) have short expiry time and require refresh logic

7.7 Testing and Validation

Security features were tested through:

- Manual testing with Postman (invalid tokens, missing roles)
- Integration tests for protected endpoints and role restrictions
- UI tests to confirm visibility of admin-only features

7.8 Summary

SeatSurfer incorporates a robust security architecture based on Spring Security and standard best practices. With role-based access control, secure password handling, and endpoint validation, the application mitigates critical vulnerabilities and ensures proper access management for both regular users and administrators.

Chapter 8

Agile Development and Project Workflow

8.1 Agile Methodology in Practice

The SeatSurfer project followed the Agile software development methodology, specifically the Scrum framework. Agile was chosen due to its flexibility, iterative nature, and support for continuous feedback and collaboration.

The key Agile principles used in the project include:

- Incremental delivery through sprints
- Continuous user feedback integration
- Adaptability to changing requirements
- Emphasis on working software over documentation

8.2 Team Structure and Roles

The SeatSurfer development team consisted of 7 students, each assigned a role according to their expertise and interests:

- **Product Owner:** Defined backlog items and functional goals
- **Scrum Master:** Facilitated sprint meetings and removed blockers
- **Backend Developers (2):** Implemented Spring Boot services and database logic
- **Frontend Developers (2):** Built the Flutter interface and logic
- **QA Engineer:** Tested functionality, reported bugs, and verified fixes

8.3 Sprint Planning and Management

The project was divided into 5 one-week sprints. Each sprint included:

- A sprint planning meeting to define tasks
- Daily stand-ups for progress updates
- Mid-week syncs (if needed)
- A sprint demo and retrospective at the end

Example Sprint Goals

- **Sprint 1:** Database design, Spring Boot project setup, login system
- **Sprint 2:** Flutter layout, REST API for bookings, basic floor visualization
- **Sprint 3:** Admin interface, seat configuration, role management
- **Sprint 4:** Reporting engine, PDF export, UI enhancements
- **Sprint 5:** Testing, error handling, deployment, final polish

8.4 Tools Used for Collaboration

- **GitHub:** Version control, pull request code reviews, issue tracking
- **Trello:** Sprint boards with To Do / Doing / Done columns
- **Discord:** Daily communication and voice stand-ups
- **Notion:** Shared documentation and backlog refinement
- **Postman:** API testing and validation by developers and QA

Trello Snapshot Example

8.5 Challenges and Lessons Learned

Throughout the project, the team encountered various challenges:

- **Integration Issues:** Merging backend and frontend workflows required careful API testing.
- **Ambiguous Requirements:** Some early tasks were vague. Agile allowed clarifications mid-sprint.
- **Time Constraints:** Balancing coursework and development required disciplined task tracking.
- **Testing Complexity:** Booking logic edge cases were difficult to automate and required real test data.

Lessons Learned

- Always write integration-ready endpoints early
- Keep designs simple and iterate based on feedback
- Write meaningful Git commits and branch names
- Sprint reviews helped maintain team momentum

8.6 Summary

The Agile methodology proved highly effective for SeatSurfer. Iterative development and clear team roles led to steady progress and consistent delivery. Retrospectives allowed the team to adjust and improve with each sprint, ultimately resulting in a stable and feature-complete system by the final milestone.

Chapter 9

Analytics and Admin Dashboard

9.1 Overview

In the current version of SeatSurfer, the administrator dashboard includes a simple but useful feature for monitoring space utilization: numerical data indicating seat occupancy for a selected floor and date.

9.2 Implemented Feature: Occupancy Data by Date

Administrators can select a date and a specific floor from a dropdown or calendar in the frontend. Upon selection, the backend returns:

- Total number of seats configured for that floor
- Number of seats booked on the selected date
- Number of available (unbooked) seats
- Percentage of occupancy

This data is presented as plain numbers in the UI, allowing administrators to understand seat usage on a specific day.

9.3 Backend Logic and API Interaction

Frontend Flow

- Admin selects a floor and date
- A REST API request is sent to the backend
- The frontend displays the response in a readable format

Backend Aggregation Logic

A controller endpoint processes the request and uses the ‘bookings‘, ‘seats‘, and ‘floors‘ tables to compute values.

Example SQL Query

```sql
SELECT
    COUNT(*) FILTER (WHERE b.id IS NOT NULL) AS booked,
    COUNT(*) FILTER (WHERE b.id IS NULL) AS available,
    COUNT(*) AS total
FROM seats s
LEFT JOIN bookings b ON s.id = b.seat_id AND b.booking_date = '
    2025-04-10'
WHERE s.floor_id = 1 AND s.active = TRUE;
```

This query calculates booked and available seats for a specific floor on a given date using a filtered LEFT JOIN.

9.4 Limitations and Future Improvements

Currently:

- The data is displayed as plain numbers (no charts or graphs)
- There is no export or history feature
- Data cannot be aggregated over a range of dates

Planned improvements include:

- Adding bar or pie charts for visual representation
- Allowing selection of multiple days or weeks for trend analysis
- Generating downloadable reports in PDF or CSV formats
- Tracking per-user booking behavior and seat heatmaps

9.5 Summary

While basic, the current analytics functionality in SeatSurfer provides essential numerical occupancy data to administrators. It enables manual inspection of seat usage and supports informed decision-making. Future updates can build on this foundation to add richer visualizations and automated reporting.

Chapter 10

Future Enhancements and Research Directions

10.1 Overview

While SeatSurfer currently implements essential functionality for seat booking and basic administrative control, it was designed from the start to be extensible. This chapter outlines proposed features and research-driven directions for future development.

These enhancements fall into three main categories:

- Intelligent features (AI-based suggestions)
- User experience improvements
- Hardware and sensor integration

10.2 AI-Powered Seat Recommendations

In large offices or shared environments, users may struggle to choose the optimal seat. SeatSurfer could integrate an AI-driven recommendation system based on:

- Previous booking history
- Time and day preferences
- Favorite seats or frequently used areas
- Team clustering — suggesting seats near colleagues

Conceptual Architecture

A lightweight recommendation engine could run as a microservice or plugin. It would:

1. Analyze past booking data per user
2. Rank available seats using weighted criteria
3. Return ranked suggestions to the frontend

10.3 Occupancy Sensor and IoT Integration

A promising direction involves real-world sensors to track actual occupancy and synchronize it with booking data. Features include:

- Presence detection via desk sensors or QR codes
- Automatic check-ins when a user arrives at their booked seat
- Auto-release of no-show bookings
- Integration with smart building systems (lighting, HVAC)

Potential Hardware Tools

- Raspberry Pi sensors with motion detection
- NFC or QR-code tags for desk verification
- Smart cameras with computer vision (privacy-aware)

10.4 Enhanced Reporting and Analytics

While current analytics display only single-day occupancy, future reports could include:

- Weekly/monthly booking trends
- Occupancy heatmaps
- Individual user activity summaries
- Downloadable PDF/CSV exports

These features can be implemented using asynchronous background jobs and caching to optimize performance.

10.5 Scalability and Multi-Tenant Support

SeatSurfer can evolve to support multiple organizations (multi-tenancy), where each company has:

- Separate floors and seat maps
- Distinct users and roles
- Isolated reports and booking data

10.6 Potential Academic Research Topics

The SeatSurfer platform can serve as a base for academic exploration in areas such as:

- Optimization algorithms for seat allocation
- Behavioral analysis of hybrid workspace use
- User interface testing for spatial decision-making
- Graph-based routing between desk zones

10.7 Conclusion

SeatSurfer lays the foundation for a complete and adaptable seat management platform. While the current version focuses on core functionality, the system was architected to allow future enhancements with minimal refactoring. With machine learning, IoT, and advanced analytics, SeatSurfer can grow into a powerful smart-office solution for the hybrid workplace era.

Appendix A

Entity Class Example (Java)

Listing A.1: Seat Entity (Java)

```java
@Entity
public class Seat {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int row;
    private int column;

    @ManyToOne
    @JoinColumn(name = "floor_id", nullable = false)
    private Floor floor;

    private boolean active;

    // getters and setters
}
```

Appendix B

REST Controller Sample

Listing B.1: Booking Controller (Java)

```java
@RestController
@RequestMapping("/api/bookings")
public class BookingController {

    @Autowired
    private BookingService bookingService;

    @PostMapping
    public ResponseEntity<?> createBooking(@RequestBody BookingDTO bookingDTO) {
        return ResponseEntity.ok(bookingService.book(bookingDTO));
    }

    @DeleteMapping("/{id}")
    public void deleteBooking(@PathVariable Long id) {
        bookingService.cancel(id);
    }
}
```

Appendix C

SQL Table Definitions

Listing C.1: Table Definitions

```
CREATE TABLE users (
    id BIGSERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    role VARCHAR(50)
);

CREATE TABLE floors (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE seats (
    id BIGSERIAL PRIMARY KEY,
    floor_id BIGINT REFERENCES floors(id),
    row INT,
    column INT,
    active BOOLEAN DEFAULT TRUE
);

CREATE TABLE bookings (
    id BIGSERIAL PRIMARY KEY,
    seat_id BIGINT REFERENCES seats(id),
    user_id BIGINT REFERENCES users(id),
    booking_date DATE NOT NULL
);
```

Appendix D

Flutter Code Sample

Listing D.1: Seat Booking Widget (Flutter)

```
Widget buildSeat(int row, int col) {
  final isBooked = bookedSeats.contains(SeatPosition(row, col))
     ;
  return GestureDetector(
    onTap: () => onSeatTap(row, col),
    child: Container(
      margin: EdgeInsets.all(4),
      width: 40,
      height: 40,
      decoration: BoxDecoration(
        color: isBooked ? Colors.red : Colors.green,
        borderRadius: BorderRadius.circular(6),
      ),
    ),
  );
}
```