CS5031 Practical 3

190031593, 200024627, 240030041, 200031957

Overview

This report presents a room booking system designed to enable organisers to book rooms to host events and allow attendees to register their attendance for these events. The system consists of a terminal and a GUI client, facilitating interaction with the system through its RESTful API. The API provides users with various capabilities, such as creating a user account, creating and cancelling bookings, signing up and withdrawing from an event, and retrieving all event attendees. The system intends to provide a simple and effective means to coordinate events, track attendance and use shared areas.

Running the application

1. Build the project `mvn clean install`
2. Run the server: `mvn spring-boot:run`
3. To run the terminal client, use `mvn compile exec:java "-Dexec.mainClass=com.stacs.cs5031.p3.client.terminal.TerminalClient"`
4. To run the GUI, use `mvn compile exec:java "-Dexec.mainClass=com.stacs.cs5031.p3.client.gui.login.LoginGUI"`
5. To test the application, use `mvn clean test`

Key features

- Basic user authentication with username and password
- Role-based access: distinct interfaces and functionalities for organisers and attendees
- Room management: view rooms, their capacity and availability
- Booking management: create, view, update, and delete bookings
- Attendee registration: register for and unregister from bookings
- Capacity management: automatic enforcement of room capacity limits
- Conflict detection: prevention of double-booking rooms

Upon running the application, a user can:

1. Log in
2. Register as an organiser or attendee

An organiser can:

1. Create a new booking by selecting a room, date and time, and event duration
2. View and manage created bookings
3. Edit booking details or delete bookings
4. View attendees registered for bookings

An attendee can:

1. View available bookings
2. Register for bookings with available capacity
3. View bookings one has registered for
4. Unregister from bookings

Technical stack

- Backend: Java 11, Spring Boot 2.6.7, Spring Data JPA
- Database: H2 in-memory database (for development)
- Frontend: Java Swing for GUI, Java console application for terminal
- API Communication: REST with JSON

System Architecture

The application follows an MVC architecture with a service-repository pattern. This architecture was selected to provide enhanced modularity, separation of concerns, and testability. The backend implementation is structured into distinct layers, each with specific responsibilities.

Server (Spring Boot)

Model (entity) layer

- Represents the core domain objects and business entities
- Contains JPA annotations for ORM mapping
- Manages entity relationships (one-to-many, many-to-many, etc.)
  Examples include User, Room, Admin, Attendee, Organiser

Repository layer

- Provides data access abstraction
- Implements Spring Data JPA repositories for CRUD operations
- Contains custom query methods in some classes
- Handles database interaction and persistence

DTO (Data Transfer Object) layer

- Defines objects for data exchange between layers
- Prevents exposure of internal entities
- Optimises network traffic by transferring only necessary data

DTO Mapper layer

- Contains mappers for entities-DTO conversion

Service layer

- Implements business logic and domain rules
- Manages interactions across multiple services
- Performs input validation

Controller layer

- Exposes RESTful API endpoints
- Handles HTTP requests and responses
- Routes requests to appropriate service methods
- Implements proper HTTP status code handling

Terminal Client

- Offers command-line interaction with the system
- Consumes the RESTful API provided by the server

GUI (Java Swing)

Due to time constraints, not all API endpoints can be invoked or visualised through the GUI. However, we prioritised delivering a visual interface that offers users basic yet useful functionalities.

Upon launching the GUI, users are taken to the login/register page, where they can either register as a new user or login with an existing account. Users who log in using an attendee account are taken to the attendee home page. This page allows them to retrieve and view all events that have available spaces and register their attendance. Additionally, attendees can view a list of the events for which they have already registered. They can also retrieve information about events that are currently full, ensuring they are aware that these events are running in case spaces become available at a later date.

Organisers logging in are taken to the organiser home page. Here, they can create a new booking or view bookings they have previously made. On their view bookings page, they can see a preview of their bookings, where by selecting a specific booking entry on the page, a dialog with extended information appears. This dialog contains additional information such as the room capacity, booking duration, and a list of attendees.

Scrum Practices

Our team adopted the Scrum framework to manage and deliver this practical. We organised our work into weekly sprints, used Jira to track tasks, and held regular meetings to support sprint planning, backlog grooming, and sprint reviews.

Sprint 1: Initial Planning and Requirement Analysis

Duration: 24 March – 30 March 2025

Meeting 1 – Sprint Planning

Date: 26 March 2025 (Online)

Objectives:

- Read and discussed the practical specification
- Understood the core project requirements

- Discussed the team's approach and collaboration methods

Meeting 2 – Backlog Grooming and System Design

Date: 29 March 2025 (In person)

Objectives:

- Revised functional requirements, finalised the user stories (see Figure 1)
- Created initial tasks in Jira (see Figure 2)
- Designed UML Class Diagram (see Figure 3)
- Discussed high-level system architecture

User Stories

- As a user I want to have the option to register as an organiser so that I can make bookings.
- As a user I want to have the option to register as an attendee so that I can register for an available booking.
- As a user, I want to have my own account so that I can check my bookings.
- As an organiser I want to be able to make bookings for a specific day/time and rooms so that
- As an organiser I want to be able to cancel bookings so that rooms can be booked by other organisers
- User can organise an event
- As an organiser I want to view the attendees who have registered for my booking so that I can find the registered capacity for my booking.
- As an organiser I want to view the available rooms so that I can book a room that is not already booked.
- As an Attendee, I want to be able to view all available bookings so that I can register for bookings that are of interest to me.
- As an Attendee, I want to be able to view all bookings that have reached full capacity in case spaces are available in future.
- As an Attendee, I want to be able to register for available booking so that I can secure my spot and make the organiser aware of my attendance.
- As an Attendee, I want to be able to view all the bookings I have registered for so that I can confirm that I have registered for an event.
- As an Attendee, I want to be able to deregister from a booking so that I can make the organiser aware that I am no longer able to attend, and other attendees are given the opportunity to register.

Figure 1. Finalised User Stories

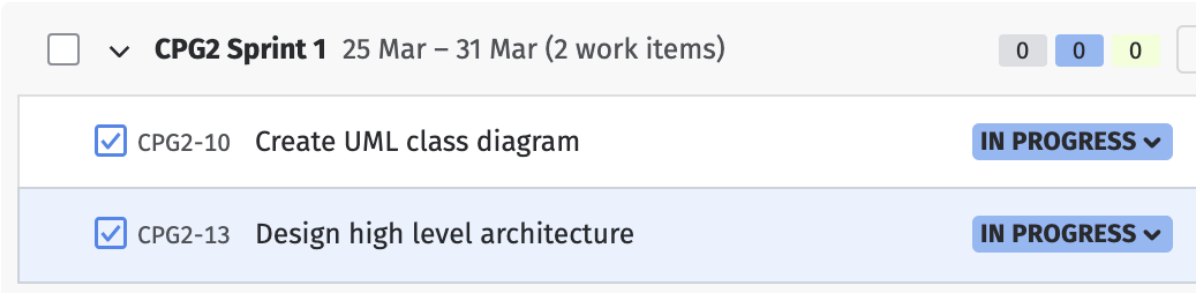

Figure 2. Jira back log during Spring 1

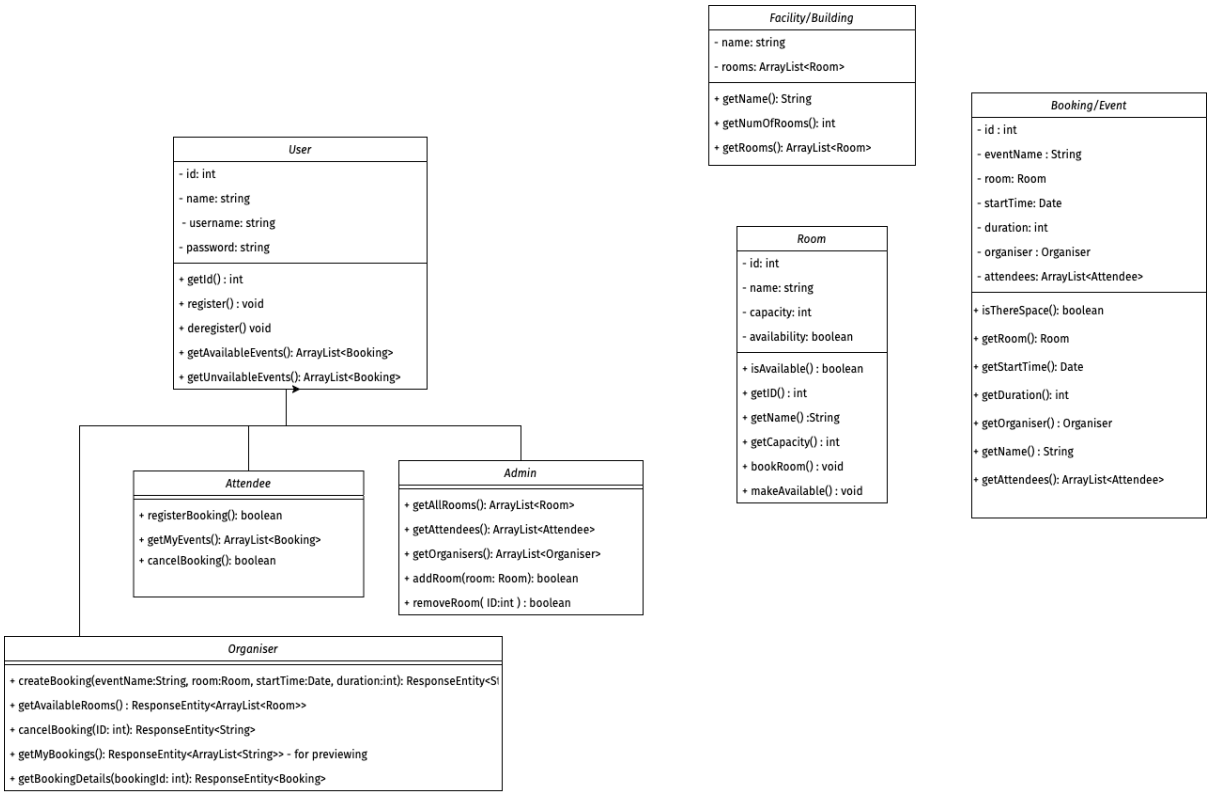

Figure 3. UML Class Diagram

Meeting 3 – Sprint Review and Presentation Preparation

Date: 30 March 2025 (Online)

Objectives:

- Drafted presentation structure and content
- Revised the user model to include admin functionality
- Finalised preparation for the upcoming client presentation (Figure 3)

1. Preamble and Scope
   1. Our team is building an application that acts as a platform for event organisers to book rooms for their events and attendees to register their attendance for those events
   2. Attendees:
      a. Attendees can view a list of bookings (events) and register or de-register for events.
      b. Can view their registered events
      c. Can view events that are at full capacity but cannot register
   3. Organisers:
      a. Organisers can view a list of available rooms,
      b. View attendees that have registered for their events
      c. View all the bookings they have made
      d. select room size and time slot (date and time) to book and unbook a room.
   4. Admin:
      a. Admin can add and remove available rooms to the database
      b. view all bookings and attendees.

2. Technologies we will use
   1. Spring to implement the API
   2. H2 for databses (tbd)
   3. Git for version control
   4. Swing for GUI (ask about FX / netbeans)
   5. JUnit for testing
   6. Mockito for testing

3. Design features/questions
   1. Are organisers admins? Or are they just another type of user?
   2. Can we incorporate netbeans for GUI

4. Time required and what is prioritised
   1. We will need 2 weeks to finish this project. Approx. 3 days to work on model classes. Approx. 6 days to work on API and database interactions. Approx. 4 days on the client.
   2. We will prioritise core requirements and a functional clean GUI.

Figure 3. Finalised Draft for Client Presentation

Key Deliverables from Sprint 1

- Agreed upon project scope and team responsibilities
- Initial Jira backlog populated with user stories
- UML Class Diagram in progress
- Client presentation prepared
- Set up project structure and Spring Boot configuration

What Worked Well

- Spring Boot's auto-configuration streamlined setup
- Effective team collaboration through online and in-person meetings
- Clear definition and agreement on user stories

- Efficient use of Jira for task management and visibility
- Team members quickly aligned on project vision and scope

Main Challenges

- Lack of understanding of JPA
- Deciding on the appropriate level of detail for the UML Class Diagram

Sprint 2: System Architecture Design and Back-End Development

Duration: 31 March – 6 April 2025

Sprint Goals:

- Begin development of core system functionality
- Finalise UML class diagram
- Finalise UML entity-relationship diagram
- Integrate any feedback from presentation to client

Client Presentation

Date: 31 March 2025 (In person)

- Delivered initial project presentation to client
    - Focused on requirement analysis
    - Served as the foundation for our product backlog
    - Communicated time estimates

Meeting 4 – Task Assignment

Date: 2 April 2025 (In person)

- Reviewed and updated the UML Class Diagram following team discussion
- Assigned system components to individual team members for development

Work areas assigned:

- Users and Attendees
- Organisers
- Rooms and Admin
- Bookings and Login functionality
- Entity-Relationship Diagram

Meeting 5 – Architecture Planning and Mid-Sprint Review

Date: 6 April 2025 (In person)

- Defined the overall system architecture, including key layers
- Identify interdependencies between domain classes and repository/service layers

Progress:

- Completed entity and repository layers for room functionalities

- Began implementation of service layers for room functionalities
- Began developing tests for and implementation of user and attendee functionalities
- Entity-relationship diagram in progress (see Figure 4)

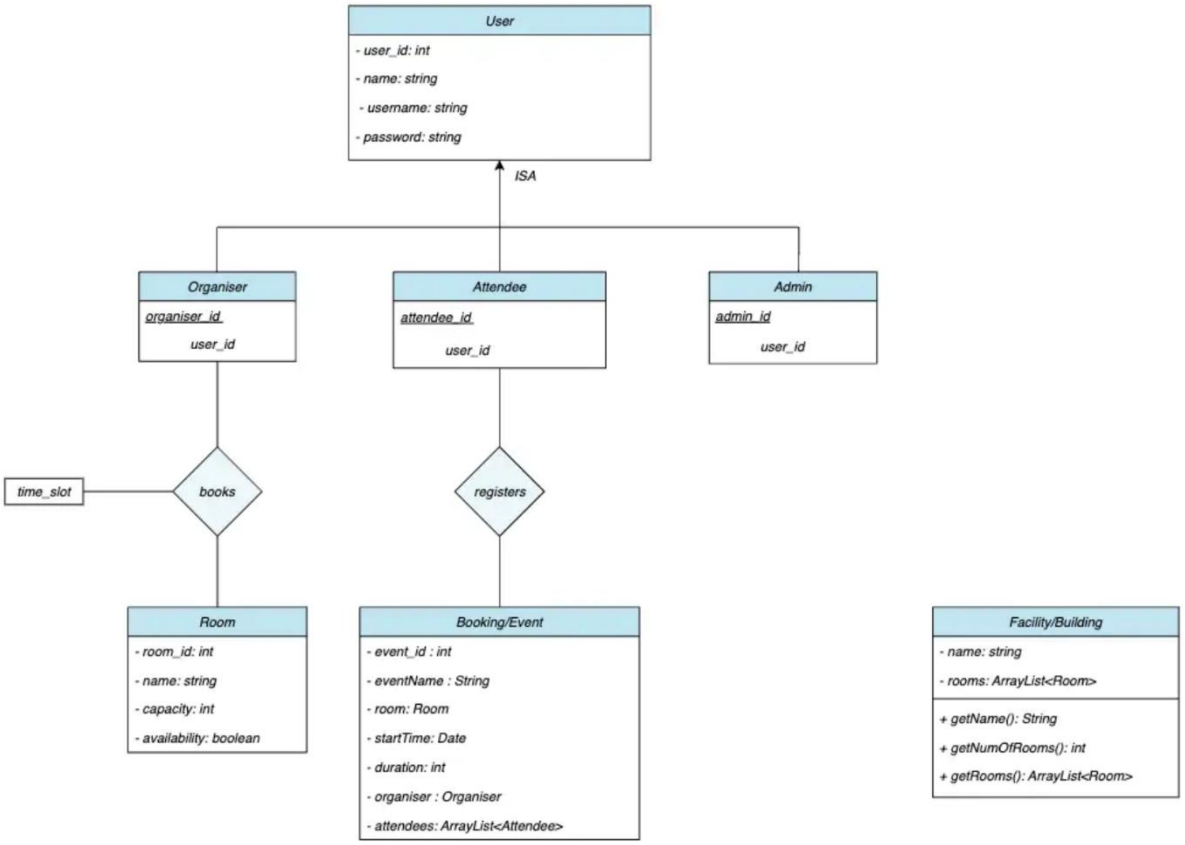

Figure 4. Entity-Relationship Diagram in Progress

Pain points:

- Certain modules (in particular, organiser functionalities) experienced delays due to interdependencies on booking functionalities

Plan to achieve before next meeting:

- Finish up the entity, repository and service layers

Meeting 6 – CI/CD Setup and Mid-Sprint Review

Date: 9 April 2025 (In person)

- Merged development branches to resolve dependencies
- Began implementing CI/CD using GitLab pipelines
- Reviewed development progress and plan upcoming Sprint Review
- Discussed readiness to begin client (front-end) implementation

Progress

- Completion of service, entity, DTO and controller layers for most of the core modules
- Backend logic for login and registration added

- Some core modules still pending integration due to interdependency between modules
- CI/CD pipeline still fails

Key Deliverables from Sprint 2

- CI/CD pipeline in progress
- Completion of entity, service, controller and DTO layers for most of the core modules – room, user, attendee
- Client functionalities in progress

What Worked Well

- Clear task assignment and ownership of specific modules
- In-person meetings significantly improved team communication
- Entity-Relationship Diagram helped to clarify database structure and relationships
- Regular meetings improved knowledge sharing across team members
- Effective use of GitLab for version control and collaboration

Main Challenges

- Merge conflicts due to concurrent development
- Interdependencies between modules slowed progress in some areas
- CI/CD pipeline setup proved more complex than anticipated
- Test coverage differed across the team and remained lower than targeted in some areas
- Integration issues when combining separately developed components, due to issues such as different data types employed for different entity IDs (`int`, `Integer` vs. `long`). These issues required significant refactoring
- Varying development speeds across the team affected the completion of interconnected features

Sprint 3: Integration and Frontend Development

Meeting 7 – GUI Planning and Mid-Sprint Review

Date: 12 April 2025 (In person)

- Assessed overall backend progress and finalised development tasks
- Began planning structure and functionality for the GUI (client side)

Progress:

- Back-end logic completed for several key modules
- Development of a basic terminal-based client started
- Database testing in progress
- Front-end planning initiated
- Some back-end progress delayed pending integration of other components

Plan before next meeting:

- Define a structure for GUI
- Fix CI/CD pipeline errors

Meeting 8 – GUI Task Allocation

Date: 16 April 2025 (In person)

- Assessed overall backend progress and remaining integration tasks
- Assigned specific GUI components to team members based on expertise (see Figure 5)

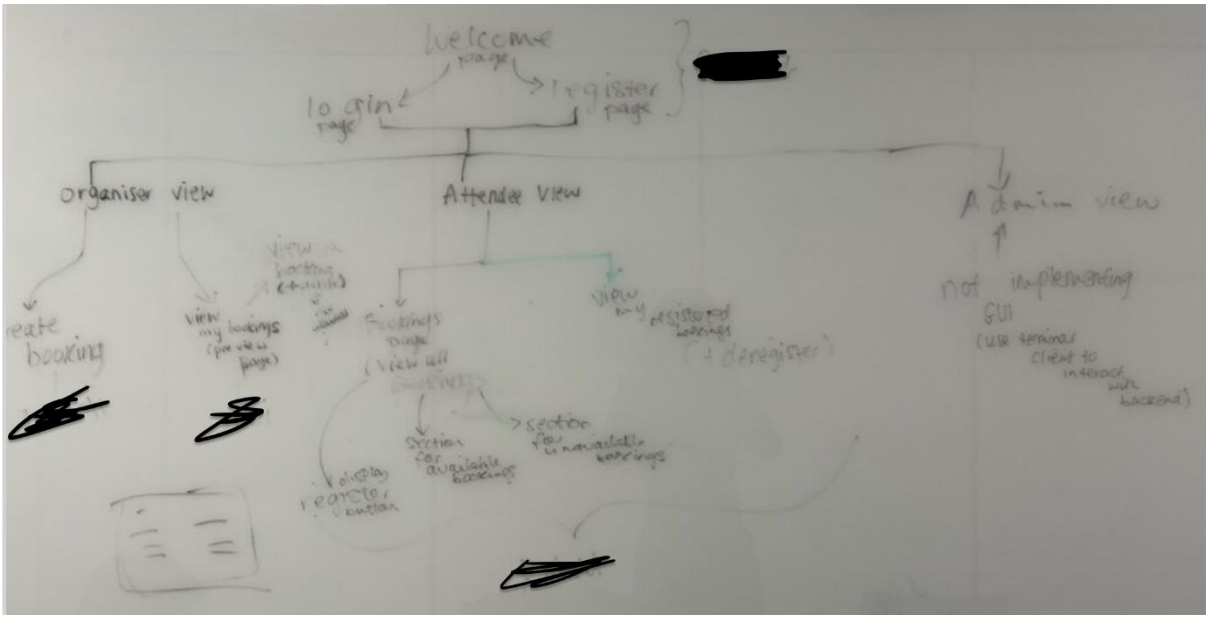

Figure 5. Task allocation for GUI implementation

Meeting 9 – Final Review

Date: 17 April 2025 (In person)

- Identified and fixed critical bugs affecting core functionality
- Finished remaining GUI components focusing on user experience
- Fixed last-minute styling issues and UI inconsistencies
- Finished final report and README file

What Worked Well

- Terminal-based client development helped validate backend functionality before GUI implementation
- Regular in-person meetings improved team communication and problem resolution
- Dividing GUI responsibilities by feature area allowed parallel frontend development

Main Challenges

- Experience with dealing with many layers was valuable but increased complexity
- CI/CD pipeline configuration required more effort than anticipated

- Interdependency between components created cascading effects when deadlines weren't met
- Inconsistent design decisions (e.g., using minutes vs. hours for event duration) led to lots of refactoring and code quality issues
- Branch management confusion had resulted in merge conflicts and lost work, though the issues were solved eventually, it required significant time
- Intricate connections between layers meant small changes required extensive refactoring across multiple files
- Limited time remained for comprehensive GUI testing after backend integration challenges

Test-Driven Development

We aimed to adhere to TDD principles throughout the project lifecycle. While we maintained strong adherence to TDD in the early sprints, increasing time pressures in later development stages occasionally led to implementing functionality before creating the corresponding test cases.

For each feature implementation, our standard TDD workflow consisted of [1]:

- Developing comprehensive test cases that precisely defined expected behaviours and outcomes
- Executing tests to verify they properly failed (confirming the absence of functionality)
- Writing minimal, focused code to satisfy test requirements
- Systematically refactoring while ensuring tests continued to pass

Server-Side Testing

Our server-side test suite provides extensive coverage across multiple architectural layers:

- Entity validation and behaviour verification
- Service layer business logic with appropriate mocking of dependencies
- Exception handling across normal and edge-case scenarios
- REST endpoint validation including request/response integrity

Client-Side Testing

TerminalClientTest.java provides unit testing for our terminal-based client interface. We focused our testing efforts primarily on backend functionality, as this represented the core system logic. While we established a framework for GUI testing, time constraints limited our ability to implement comprehensive automated tests for the graphical interface. Manual testing was conducted to validate critical user flows and interface functionality.

Conclusion

Our room booking system is designed to support role-specific functionalities, primarily tailored to two user types: attendees and organisers. The system allows organisers to create and manage room bookings, while attendees can view and register for available sessions. This role-based approach ensures that users interact with the system in ways that align with their responsibilities and permissions.

Due to time constraints, the current implementation focuses on delivering the core features required for basic booking operations, such as user registration and login, room booking, and event registration. While the system performs its intended tasks effectively, there are areas identified for future enhancement.

These include the addition of more robust error handling mechanisms, particularly to manage unexpected or malicious user input. This would enhance system security and stability under edge-case conditions. Additionally, several user experience improvements could be made, especially in the terminal-based client. For example, implementing intuitive commands such as a quit function to allow users to exit gracefully would provide a more user-friendly interface.

Future iterations may also consider expanding support for administrative roles, adding support for real-time updates and notifications, depending on user actions and room availability, and more comprehensive integration testing.

References

[1] M. Fowler, “Test Driven Development,” *martinfowler.com*, Dec. 11, 2023. https://martinfowler.com/bliki/TestDrivenDevelopment.html