INSTITUTE OF TECHNOLOGY OF CAMBODIA

PROJECT 5:

RESEARCH CONFERENCE REGISTRATION AND SESSION MANAGEMENT SYSTEM

Lecturer: Roeun Pacharoth

TEAM MEMBERS

1. SOPHAT ODOM
2. SOPHEAP SOTHIPHAK
3. THY PHAROTH
4. RA SOCHEATEY
5. PHE RITHIKA

TABLE OF CONTENTS

PROJECT OVERVIEW

A centralized platform to manage the lifecycle of a Research Conference.

- **Goal:** To automate the coordination between organizers and participants.
- **The System:** A web application that handles user registrations, session scheduling, and room assignments.
- **Core Purpose:** To replace manual spreadsheets with a secure, automated system that prevents scheduling conflicts and room overbooking.

PROJECT OBJECTIVE

1. **Secure Infrastructure:** Build a safe environment using **Spring Security** and **BCrypt hashing** to protect user data.
2. **Role-Based Control:** Create clear boundaries so that only Admins can change system settings, while Chairs focus on their specific sessions.
3. **Data Reliability:** Ensure the database is always consistent across our team of 5 using Flyway version control.
4. **Enforced Integrity:** Automate the rules of a conference—no double-booking rooms and no overlapping schedules for participants.

PROJECT PLANNING

Methodology

Followed an Iterative Development Lifecycle focused on core stability before frontend polishing.

Requirement Mapping

Analyzed the three user lifecycles (Admin, Chair, Participant) to define the necessary Spring Security matchers and controller endpoints.

Database-First Strategy

Utilized Flyway Migrations (V1–V7) to establish a firm relational schema before writing business logic. This ensured data consistency across the team.

Architecture Choice

Selected Spring Boot MVC to ensure "Separation of Concerns"—keeping validation logic in the Service layer and display logic in Thymeleaf.

TASK DIVISION

Sophat Odom

Project Architecture, Merger, Fixer and add missing files, DB Migrations

Phe Rithika

Handles main entity CRUD

Thy Pharoth

Security and Authentication Lead – Handles login, roles.

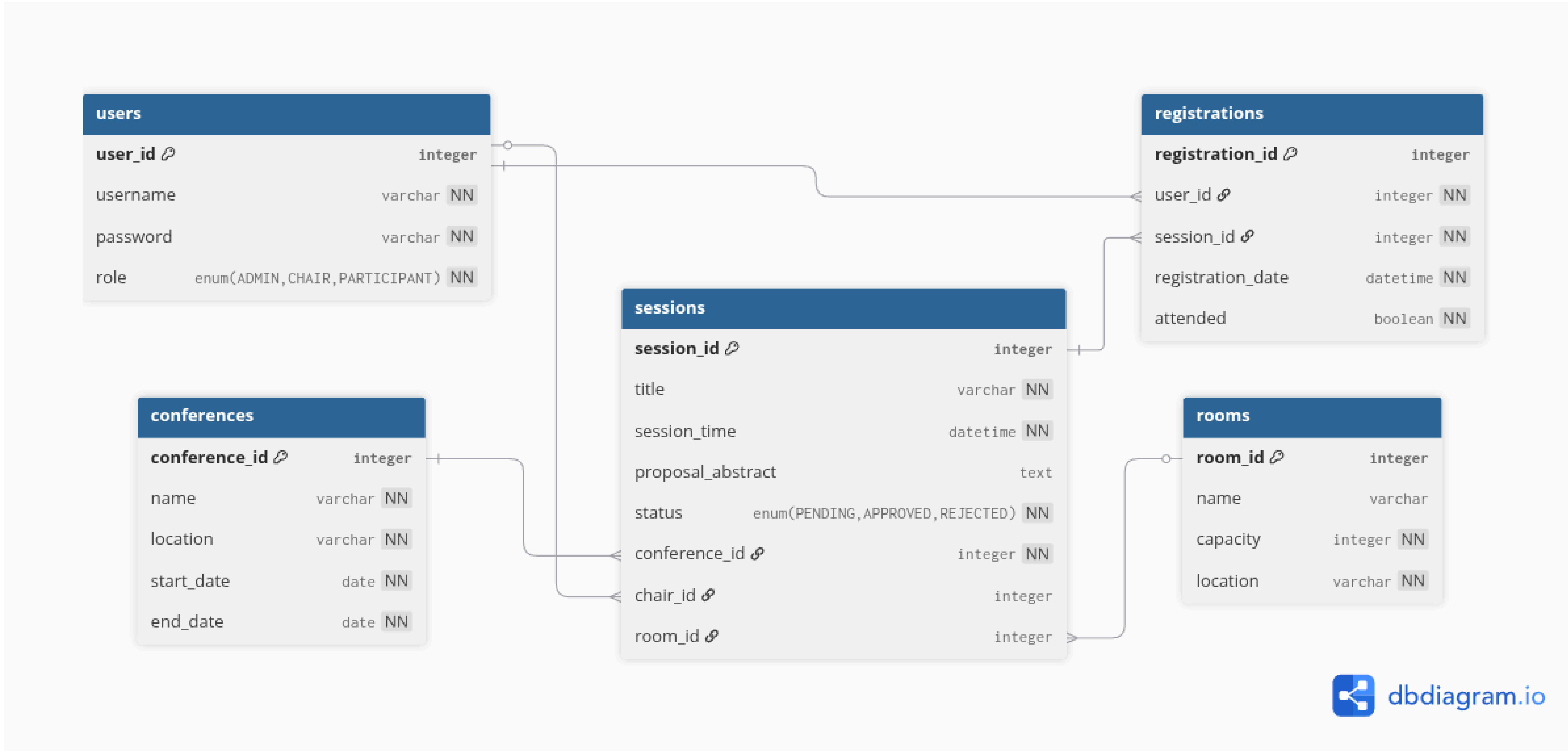
Sopheap Sothiphak

Frontend/Thymeleaf: Templates

Ra Socheatey

Handles secondary module CRUD.

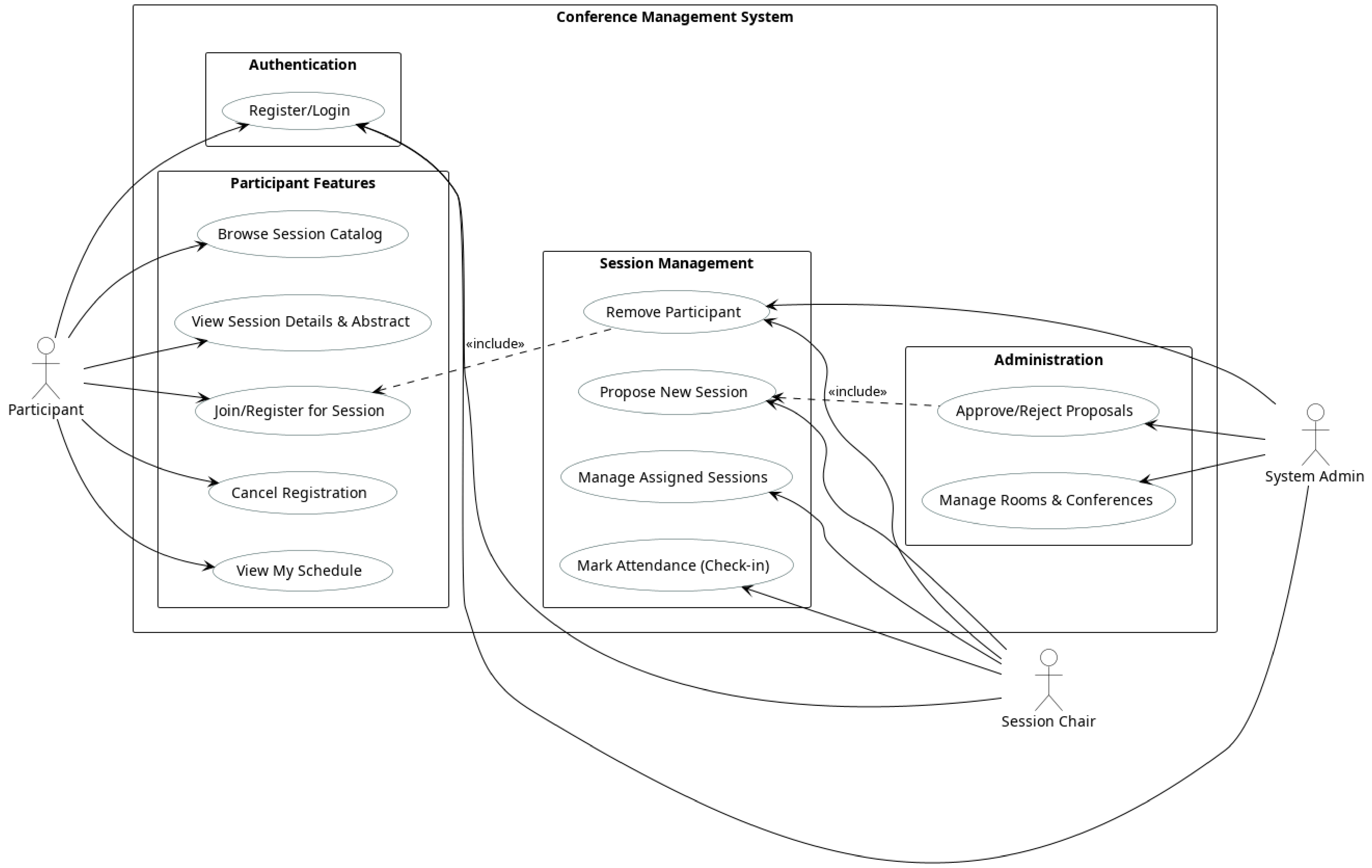
ER DIAGRAM



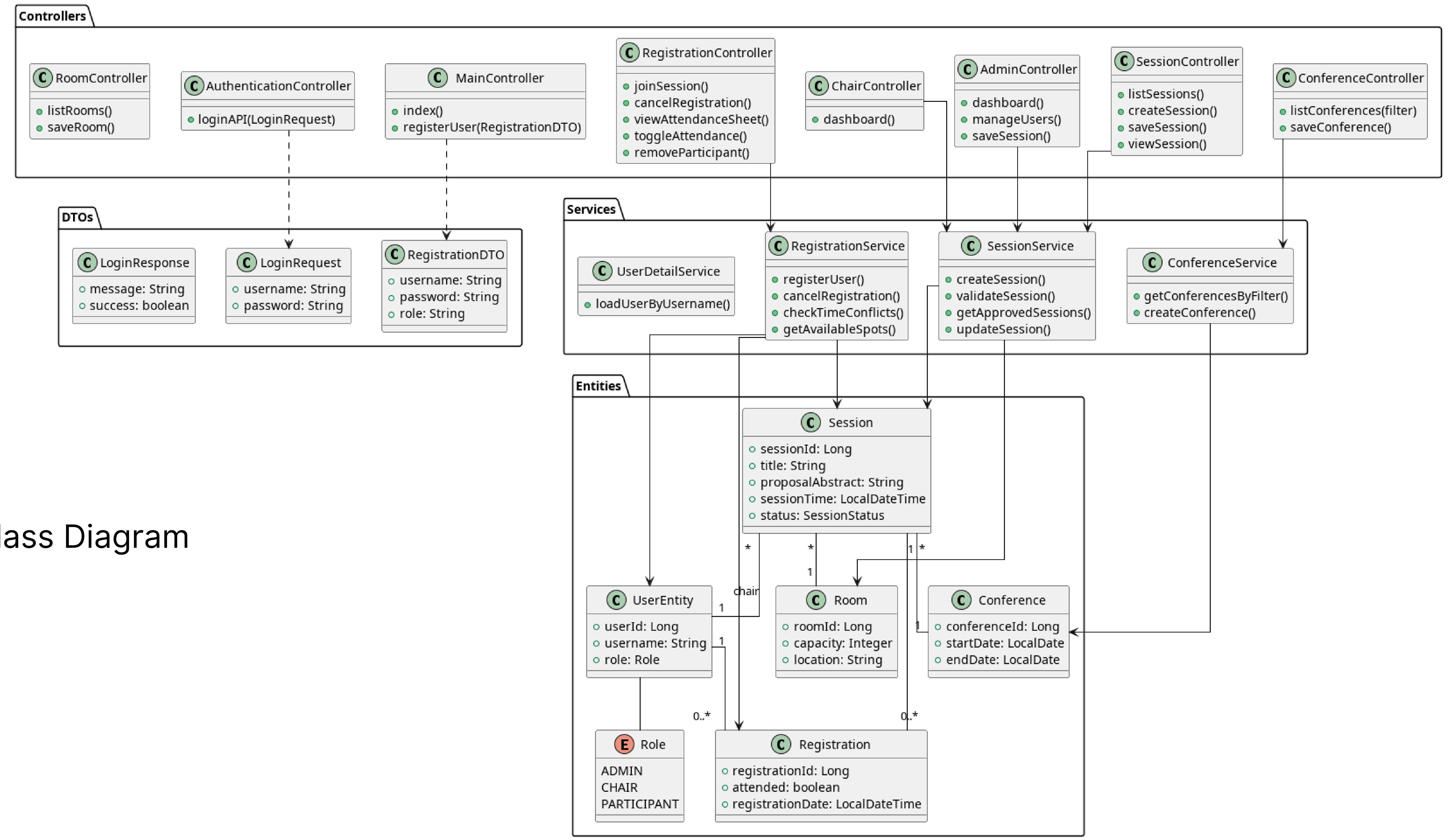
users
user_id integer
username varchar NN
password varchar NN
role enum(ADMIN,CHAIR,PARTICIPANT) NN
registrations
registration_id integer
user_id integer NN
session_id integer NN
registration_date datetime NN
attended boolean NN
sessions
session_id integer
title varchar NN
session_time datetime NN
proposal_abstract text
status enum(PENDING,APPROVED,REJECTED) NN
conference_id integer NN
chair_id integer
room_id integer
conferences
conference_id integer
name varchar NN
location varchar NN
start_date date NN
end_date date NN
rooms
room_id integer
name varchar
capacity integer NN
location varchar NN
dbdiagram.io

UML DESIGN

Use Case



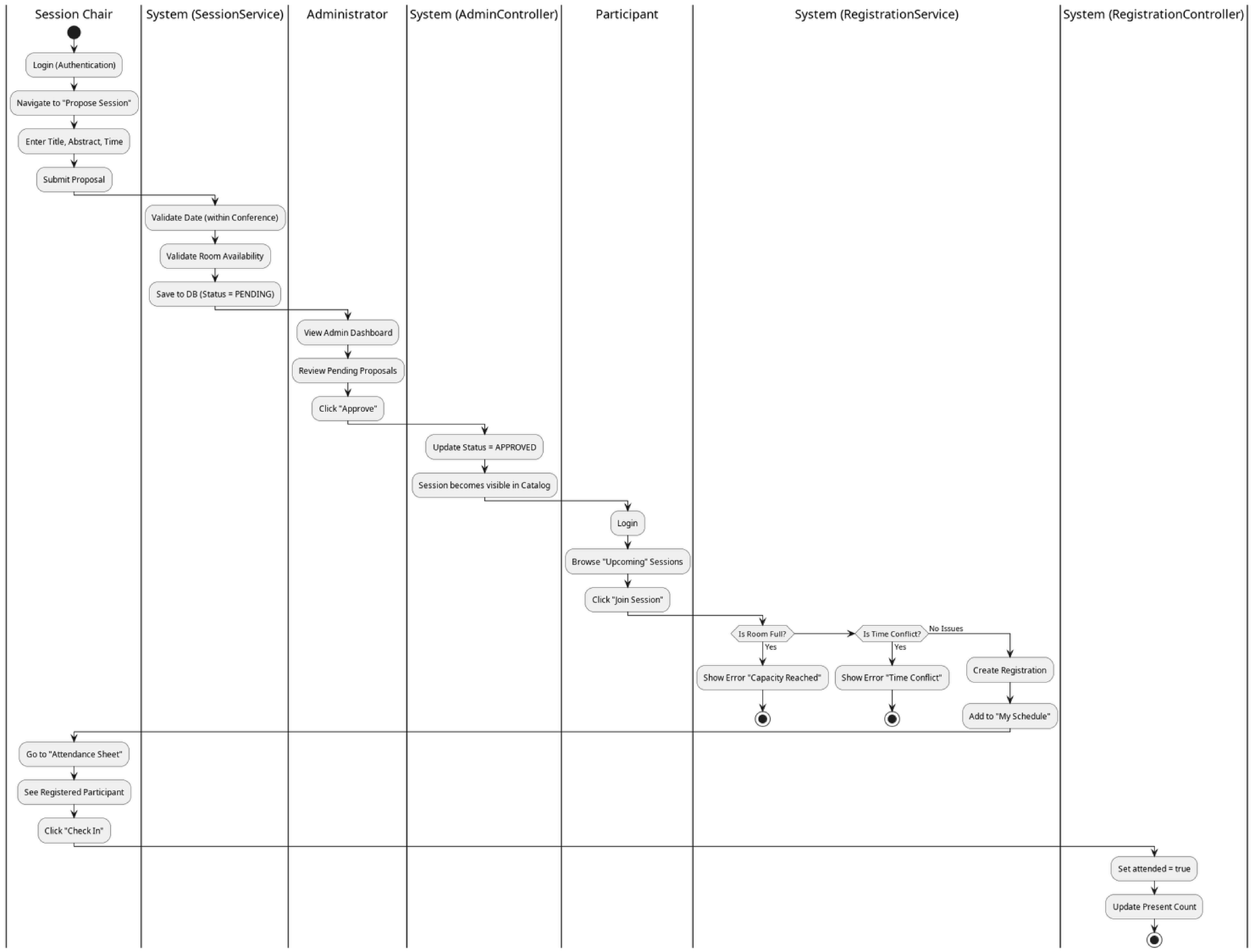
UML DESIGN



Class Diagram

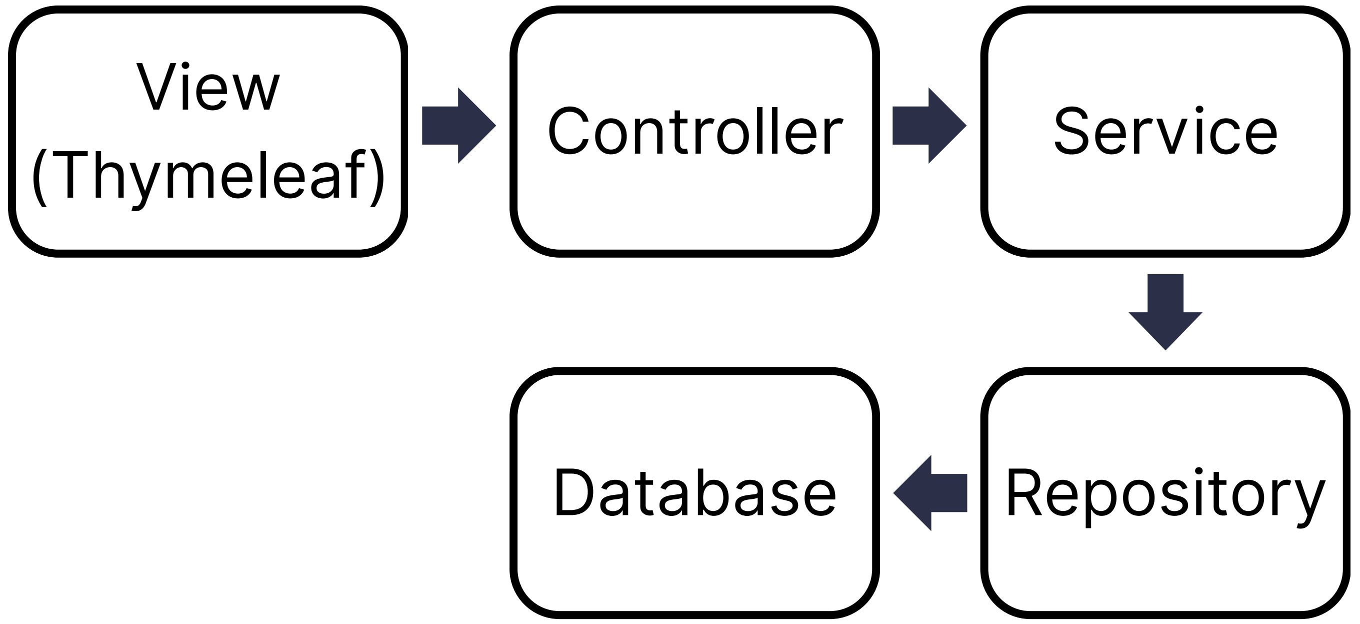
UML DESIGN

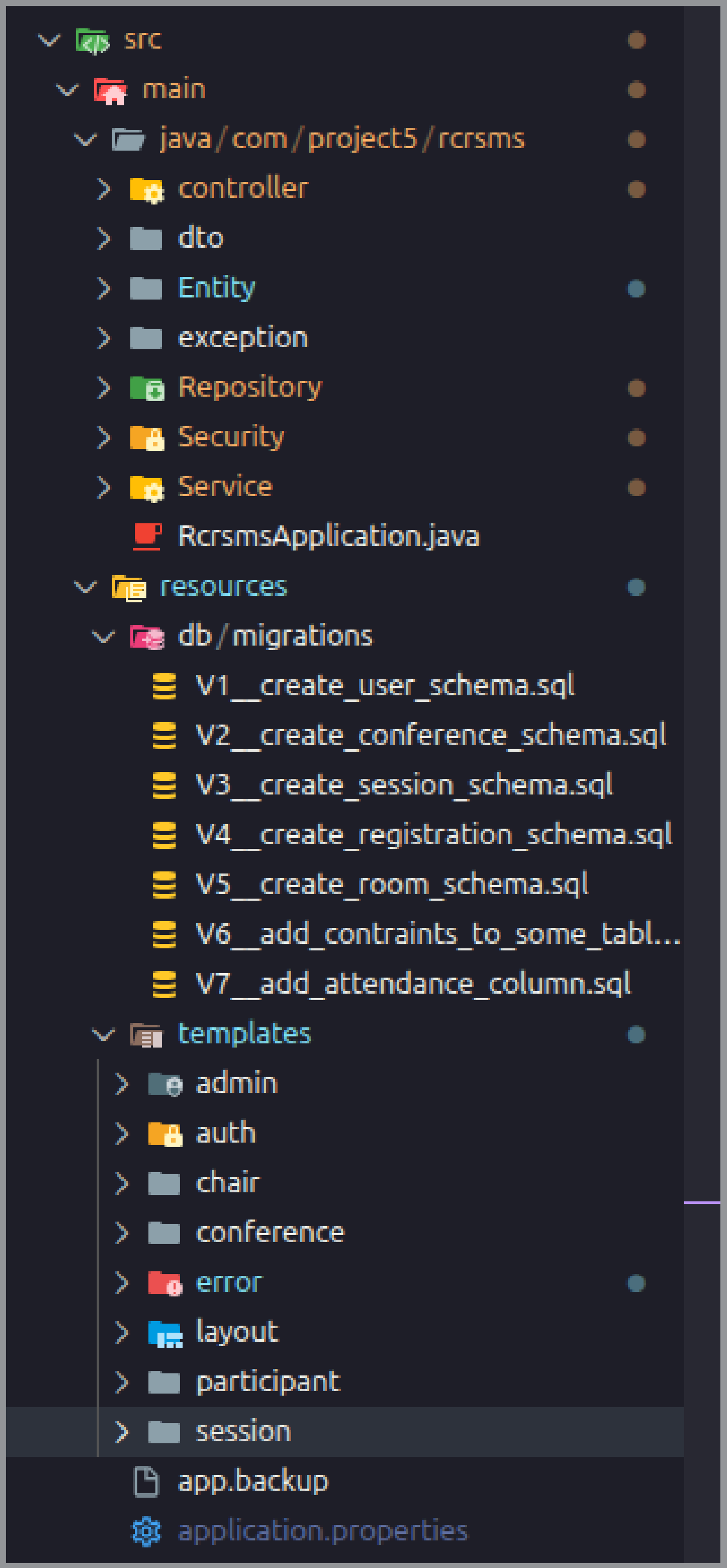
Activity



ARCHITECTURE

Utilizes the Model-View-Controller (MVC) design to ensure strict separation between data, logic, and presentation.





```
src
  main
    java / com / project5 / rcrsms
      controller
      dto
      Entity
      exception
      Repository
      Security
      Service
      RcrsmsApplication.java
    resources
      db / migrations
        V1__create_user_schema.sql
        V2__create_conference_schema.sql
        V3__create_session_schema.sql
        V4__create_registration_schema.sql
        V5__create_room_schema.sql
        V6__add_contraints_to_some_tabl...
        V7__add_attendance_column.sql
      templates
        admin
        auth
        chair
        conference
        error
        layout
        participant
        session
      app.backup
      application.properties
```

TECHNICAL PROCESS

- **Repository Layer (Data Access):** Leveraged Spring Data JPA and Hibernate for robust communication with the MySQL database.
- **Service Layer (Business Logic):** Centralized all "Business Rules" to keep logic separate from navigation. For example, the RegistrationService handles the complex calculation of Room capacity against active registrations.
- **Controller Layer (Traffic Control):** Managed the flow of data between the UI and Services, ensuring thin controllers that are easier to maintain.
- **Security Integration:** Embedded Spring Security directly into the technical flow to handle BCrypt password hashing and custom role-based redirects upon successful login.
- **Database Versioning:** Maintained professional version control of the schema using Flyway Migrations (V1–V6), allowing for seamless database updates across all developer environments.
- **Server-Side Validation:** Utilized @Valid and BindingResult to catch user errors—such as weak passwords or empty fields—beforValide they reach the database.

DESIGN WORKFLOW

- **Requirement Analysis:** Defined three specific user lifecycles (Admin, Chair, Participant) to determine necessary security matchers and application endpoints.
- **Database-First Development:** Prioritized the relational schema using Flyway Migrations (V1–V6).
- **Prototyping & Layout Design:** Created a global Thymeleaf layout and navbar fragments to ensure a consistent, responsive UI across all modules.
- **Iterative Implementation:** Developed core CRUD features (Conferences and Rooms) first, followed by complex registration logic and role-based security.

PROJECT WORKFLOW

Feature	Week 1	Week 2	Week 3	Week 4	Week 5
Project Setup	🟩				
Database & Entity	🟨	🟨	🟨	🟨	🟩
Security & Authentication	🟨	🟨	🟨	🟨	🟩
Controllers	🟨	🟨	🟨	🟨	🟩
Frontend	🟨	🟨	🟨	🟨	🟩
Testing		🟨	🟨	🟨	🟨

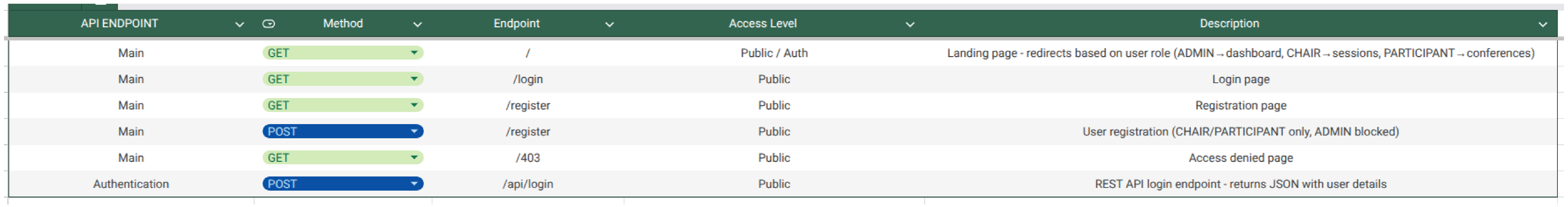
API ENDPOINT DESIGN

1. **API ENDPOINT USE FOR:**

- **Main**
- **Authentication**
- **Admin**
- **User**
- **Room**
- **Conferences**
- **Session**

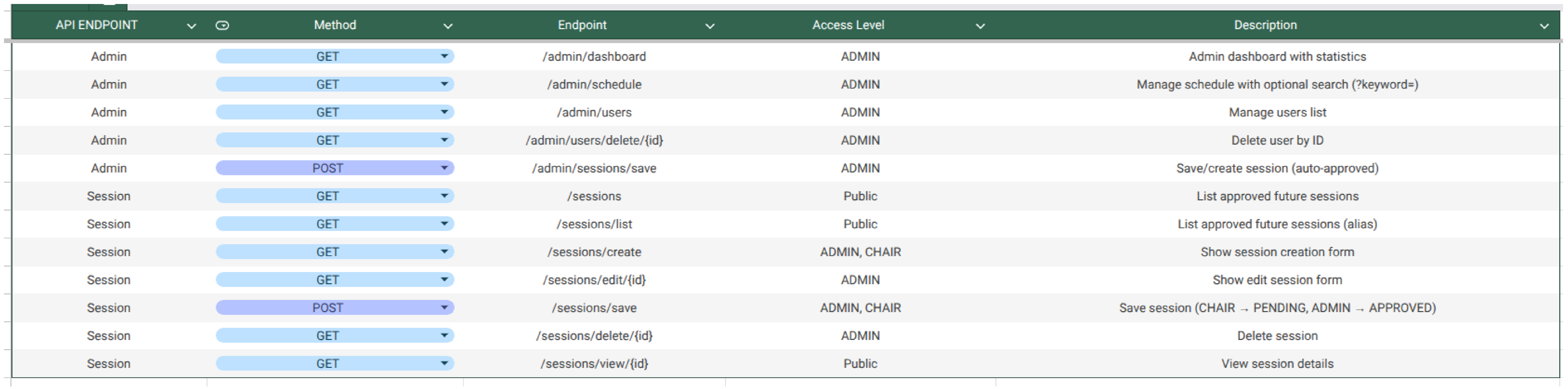
API ENDPOINT DESIGN

- **Main and Authentication**



API ENDPOINT	Method	Endpoint	Access Level	Description
Main	GET	/	Public / Auth	Landing page - redirects based on user role (ADMIN → dashboard, CHAIR → sessions, PARTICIPANT → conferences)
Main	GET	/login	Public	Login page
Main	GET	/register	Public	Registration page
Main	POST	/register	Public	User registration (CHAIR/PARTICIPANT only, ADMIN blocked)
Main	GET	/403	Public	Access denied page
Authentication	POST	/api/login	Public	REST API login endpoint - returns JSON with user details

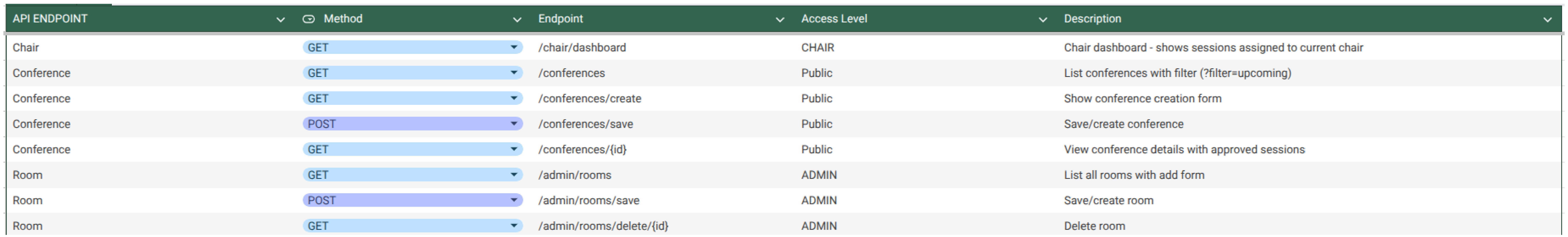
- **Admin And Session**



API ENDPOINT	Method	Endpoint	Access Level	Description
Admin	GET	/admin/dashboard	ADMIN	Admin dashboard with statistics
Admin	GET	/admin/schedule	ADMIN	Manage schedule with optional search (?keyword=)
Admin	GET	/admin/users	ADMIN	Manage users list
Admin	GET	/admin/users/delete/{id}	ADMIN	Delete user by ID
Admin	POST	/admin/sessions/save	ADMIN	Save/create session (auto-approved)
Session	GET	/sessions	Public	List approved future sessions
Session	GET	/sessions/list	Public	List approved future sessions (alias)
Session	GET	/sessions/create	ADMIN, CHAIR	Show session creation form
Session	GET	/sessions/edit/{id}	ADMIN	Show edit session form
Session	POST	/sessions/save	ADMIN, CHAIR	Save session (CHAIR → PENDING, ADMIN → APPROVED)
Session	GET	/sessions/delete/{id}	ADMIN	Delete session
Session	GET	/sessions/view/{id}	Public	View session details

API ENDPOINT DESIGN

CHAIR , CONFERENCES AND ROOM

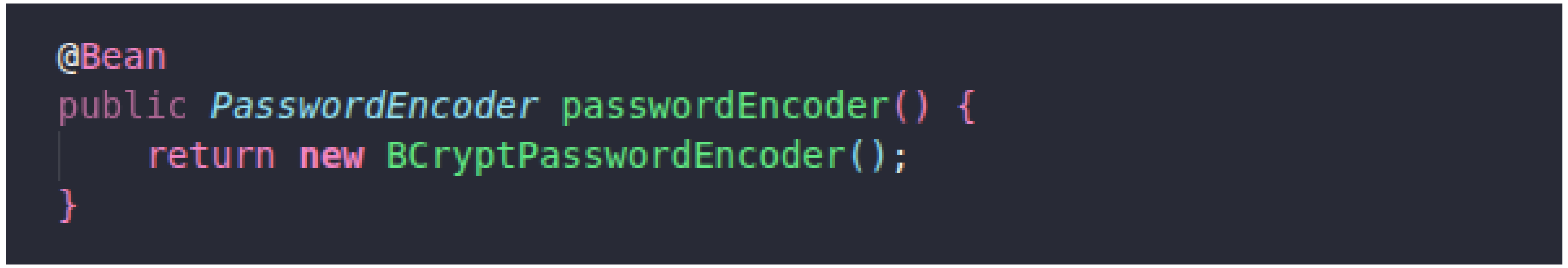


API ENDPOINT	Method	Endpoint	Access Level	Description
Chair	GET	/chair/dashboard	CHAIR	Chair dashboard - shows sessions assigned to current chair
Conference	GET	/conferences	Public	List conferences with filter (?filter=upcoming)
Conference	GET	/conferences/create	Public	Show conference creation form
Conference	POST	/conferences/save	Public	Save/create conference
Conference	GET	/conferences/{id}	Public	View conference details with approved sessions
Room	GET	/admin/rooms	ADMIN	List all rooms with add form
Room	POST	/admin/rooms/save	ADMIN	Save/create room
Room	GET	/admin/rooms/delete/{id}	ADMIN	Delete room

IMPLEMENTATION

Password Security

Utilized BCryptPasswordEncoder to ensure user credentials are
never stored in plain text.

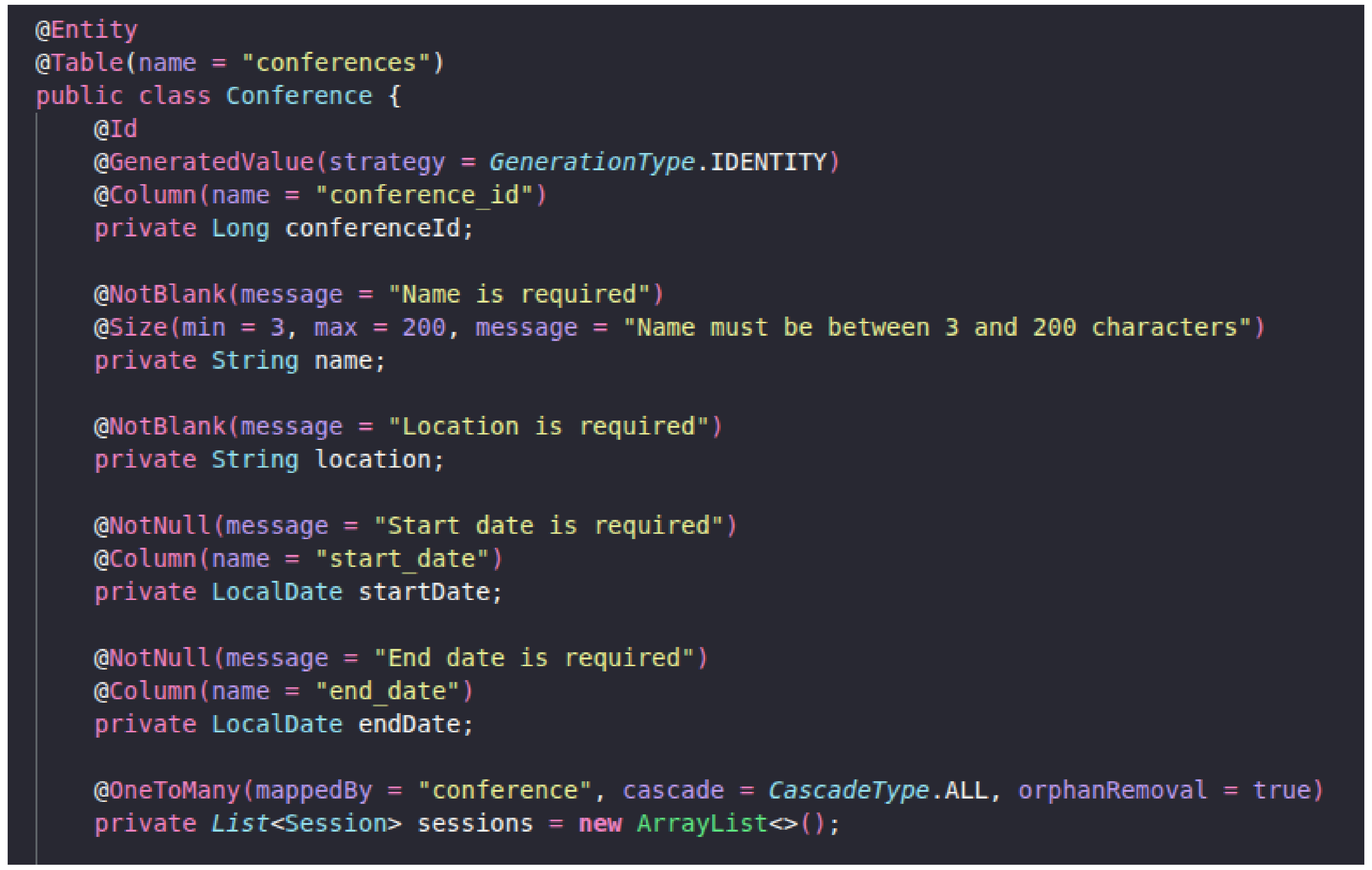


```java
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

IMPLEMENTATION

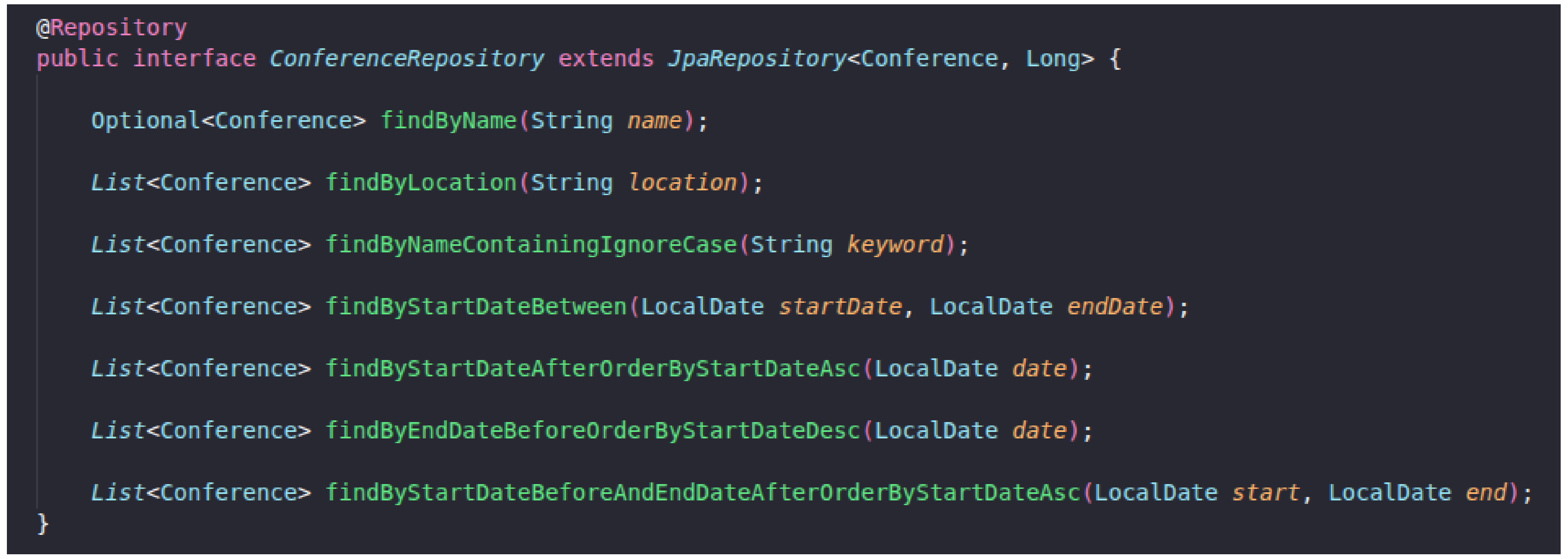
Data Persistence

Conference Entity



```java
@Entity
@Table(name = "conferences")
public class Conference {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "conference_id")
    private Long conferenceId;

    @NotBlank(message = "Name is required")
    @Size(min = 3, max = 200, message = "Name must be between 3 and 200 characters")
    private String name;

    @NotBlank(message = "Location is required")
    private String location;

    @NotNull(message = "Start date is required")
    @Column(name = "start_date")
    private LocalDate startDate;

    @NotNull(message = "End date is required")
    @Column(name = "end_date")
    private LocalDate endDate;

    @OneToMany(mappedBy = "conference", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Session> sessions = new ArrayList<>();
```

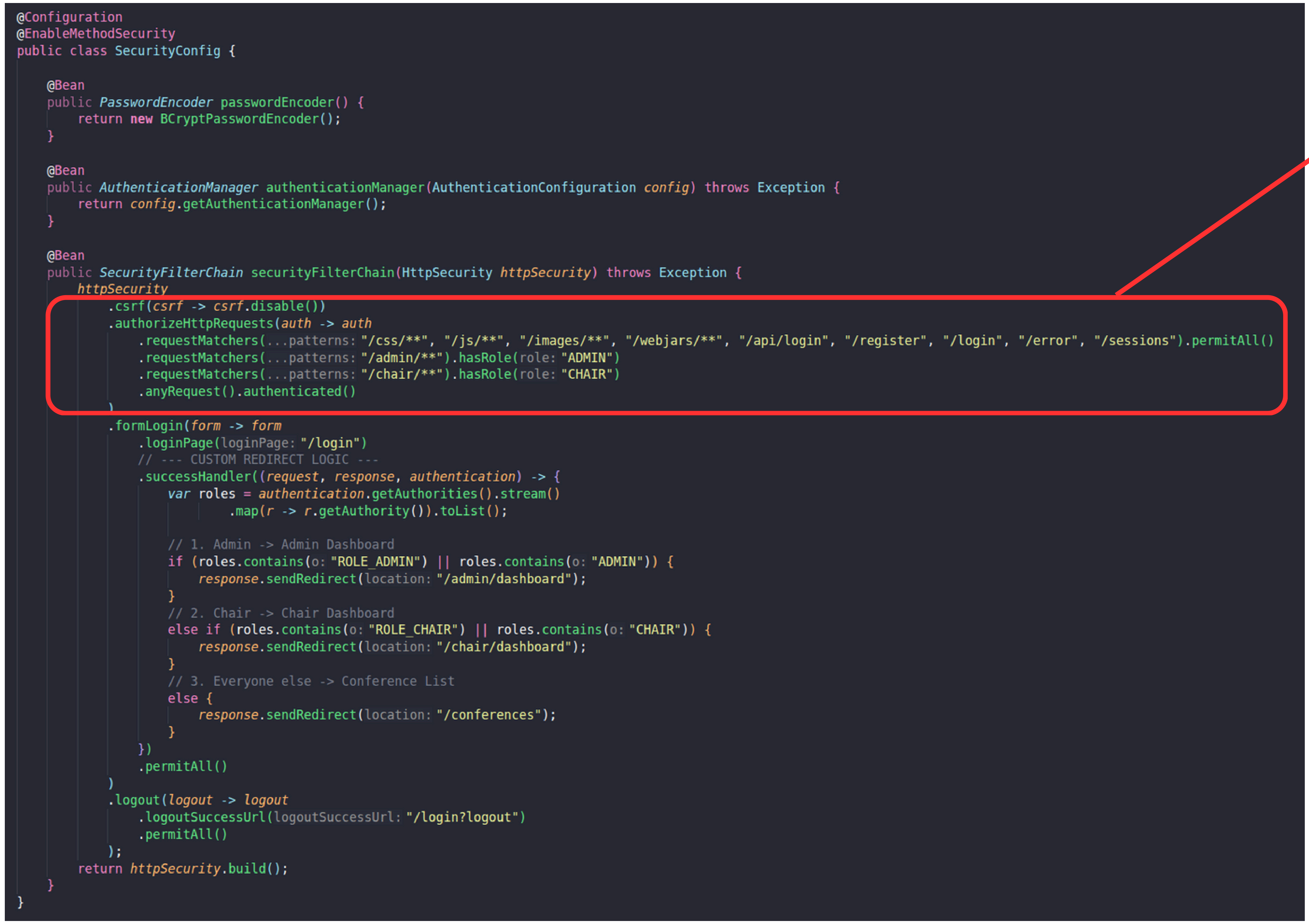
Conference Repository



```java
@Repository
public interface ConferenceRepository extends JpaRepository<Conference, Long> {

    Optional<Conference> findByName(String name);

    List<Conference> findByLocation(String location);

    List<Conference> findByNameContainingIgnoreCase(String keyword);

    List<Conference> findByStartDateBetween(LocalDate startDate, LocalDate endDate);

    List<Conference> findByStartDateAfterOrderByStartDateAsc(LocalDate date);

    List<Conference> findByEndDateBeforeOrderByStartDateDesc(LocalDate date);

    List<Conference> findByStartDateBeforeAndEndDateAfterOrderByStartDateAsc(LocalDate start, LocalDate end);
}
```

IMPLEMENTATION

Security Configuration

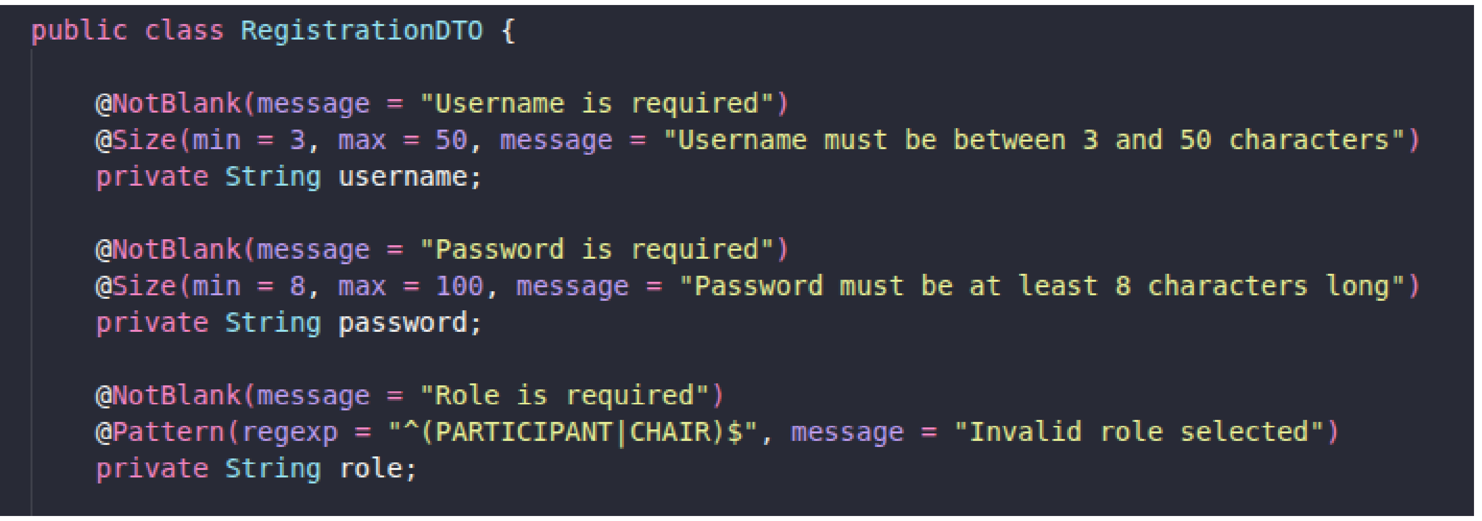


```java
@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(...patterns: "/css/**", "/js/**", "/images/**", "/webjars/**", "/api/login", "/register", "/login", "/error", "/sessions").permitAll()
                .requestMatchers(...patterns: "/admin/**").hasRole(role: "ADMIN")
                .requestMatchers(...patterns: "/chair/**").hasRole(role: "CHAIR")
                .anyRequest().authenticated()
            )
            .formLogin(form -> form
                .loginPage(loginPage: "/login")
                // --- CUSTOM REDIRECT LOGIC ---
                .successHandler((request, response, authentication) -> {
                    var roles = authentication.getAuthorities().stream()
                            .map(r -> r.getAuthority()).toList();

                    // 1. Admin -> Admin Dashboard
                    if (roles.contains(o: "ROLE_ADMIN") || roles.contains(o: "ADMIN")) {
                        response.sendRedirect(location: "/admin/dashboard");
                    }
                    // 2. Chair -> Chair Dashboard
                    else if (roles.contains(o: "ROLE_CHAIR") || roles.contains(o: "CHAIR")) {
                        response.sendRedirect(location: "/chair/dashboard");
                    }
                    // 3. Everyone else -> Conference List
                    else {
                        response.sendRedirect(location: "/conferences");
                    }
                })
                .permitAll()
            )
            .logout(logout -> logout
                .logoutSuccessUrl(logoutSuccessUrl: "/login?logout")
                .permitAll()
            );
        return httpSecurity.build();
    }
}
```

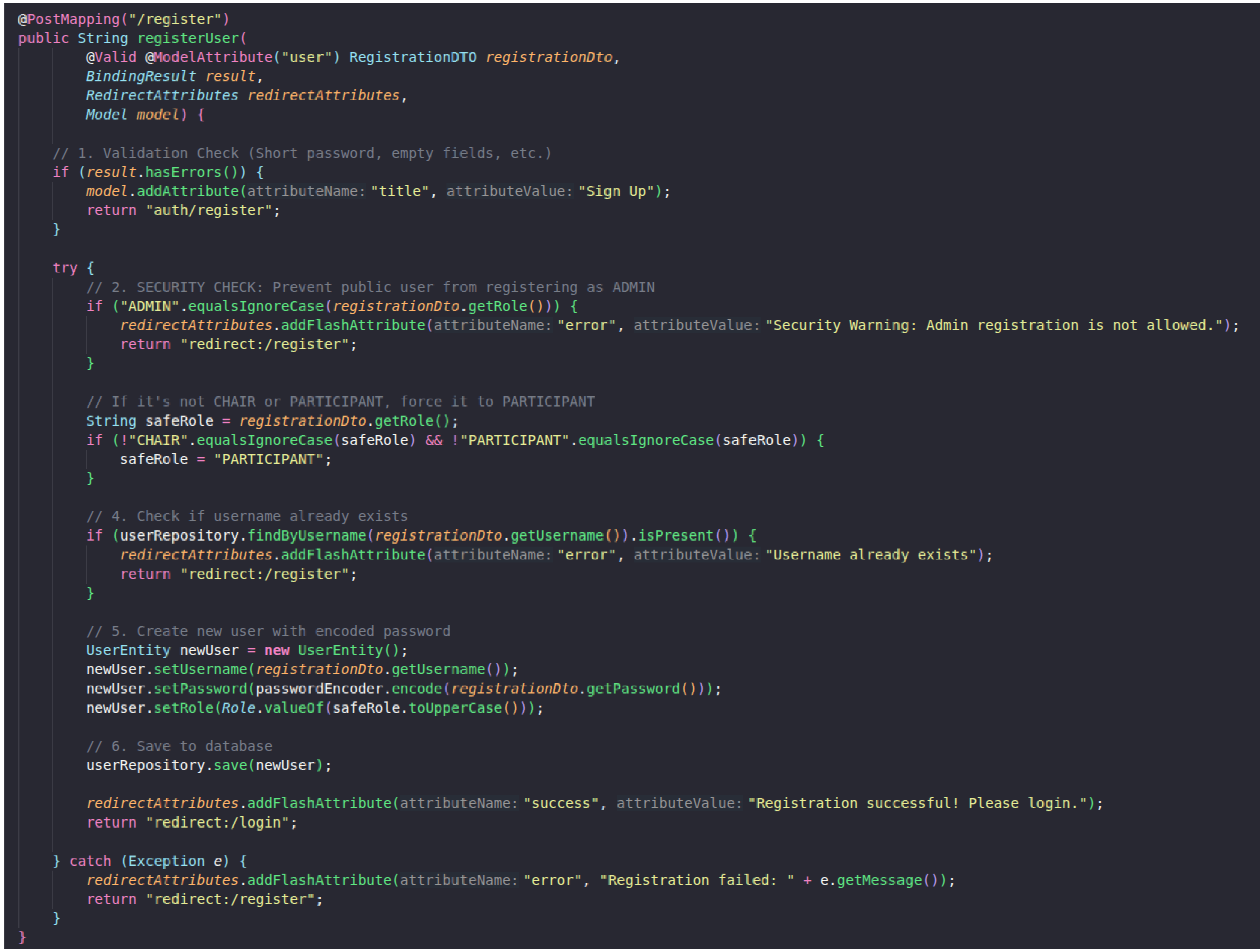
Path validation

IMPLEMENTATION

Register Validation



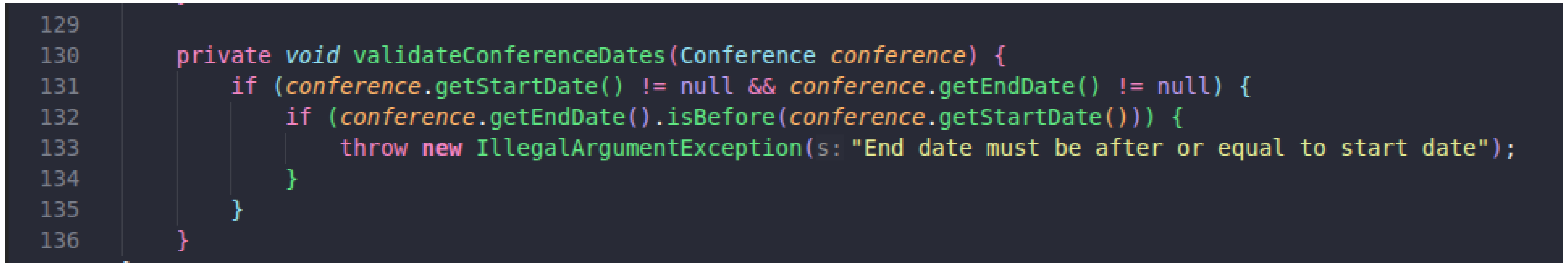
```java
public class RegistrationDTO {

    @NotBlank(message = "Username is required")
    @Size(min = 3, max = 50, message = "Username must be between 3 and 50 characters")
    private String username;

    @NotBlank(message = "Password is required")
    @Size(min = 8, max = 100, message = "Password must be at least 8 characters long")
    private String password;

    @NotBlank(message = "Role is required")
    @Pattern(regexp = "^(PARTICIPANT|CHAIR)$", message = "Invalid role selected")
    private String role;
```



```java
@PostMapping("/register")
public String registerUser(
        @Valid @ModelAttribute("user") RegistrationDTO registrationDto,
        BindingResult result,
        RedirectAttributes redirectAttributes,
        Model model) {

    // 1. Validation Check (Short password, empty fields, etc.)
    if (result.hasErrors()) {
        model.addAttribute(attributeName: "title", attributeValue: "Sign Up");
        return "auth/register";
    }

    try {
        // 2. SECURITY CHECK: Prevent public user from registering as ADMIN
        if ("ADMIN".equalsIgnoreCase(registrationDto.getRole())) {
            redirectAttributes.addFlashAttribute(attributeName: "error", attributeValue: "Security Warning: Admin registration is not allowed.");
            return "redirect:/register";
        }

        // If it's not CHAIR or PARTICIPANT, force it to PARTICIPANT
        String safeRole = registrationDto.getRole();
        if (!"CHAIR".equalsIgnoreCase(safeRole) && !"PARTICIPANT".equalsIgnoreCase(safeRole)) {
            safeRole = "PARTICIPANT";
        }

        // 4. Check if username already exists
        if (userRepository.findByUsername(registrationDto.getUsername()).isPresent()) {
            redirectAttributes.addFlashAttribute(attributeName: "error", attributeValue: "Username already exists");
            return "redirect:/register";
        }

        // 5. Create new user with encoded password
        UserEntity newUser = new UserEntity();
        newUser.setUsername(registrationDto.getUsername());
        newUser.setPassword(passwordEncoder.encode(registrationDto.getPassword()));
        newUser.setRole(Role.valueOf(safeRole.toUpperCase()));

        // 6. Save to database
        userRepository.save(newUser);

        redirectAttributes.addFlashAttribute(attributeName: "success", attributeValue: "Registration successful! Please login.");
        return "redirect:/login";

    } catch (Exception e) {
        redirectAttributes.addFlashAttribute(attributeName: "error", "Registration failed: " + e.getMessage());
        return "redirect:/register";
    }
}
```

IMPLEMENTATION

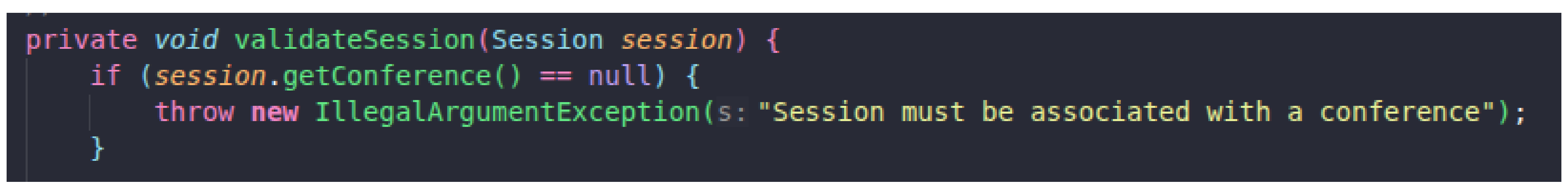
Validate Conference Date



```java
    private void validateConferenceDates(Conference conference) {
        if (conference.getStartDate() != null && conference.getEndDate() != null) {
            if (conference.getEndDate().isBefore(conference.getStartDate())) {
                throw new IllegalArgumentException(s: "End date must be after or equal to start date");
            }
        }
    }
```

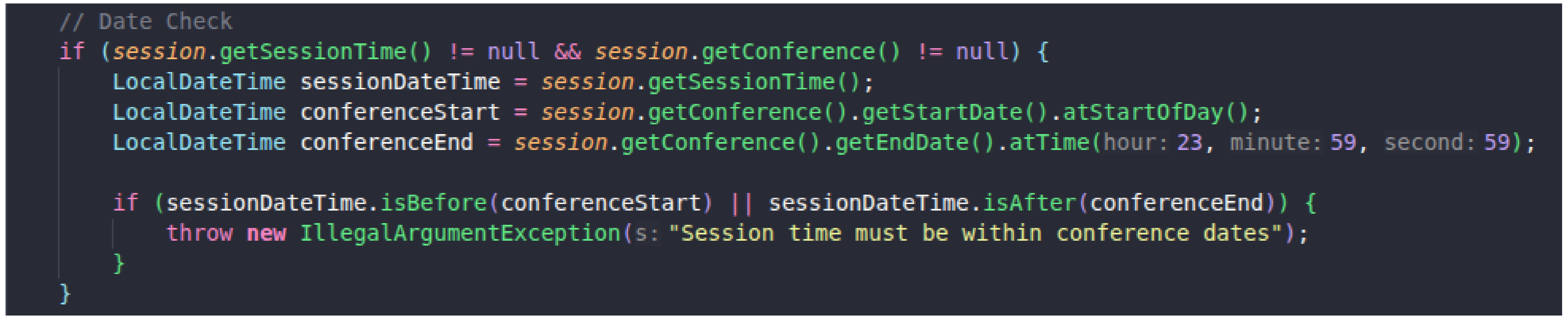
IMPLEMENTATION

Session date validates within Conference dates



```java
private void validateSession(Session session) {
    if (session.getConference() == null) {
        throw new IllegalArgumentException(s: "Session must be associated with a conference");
    }
```

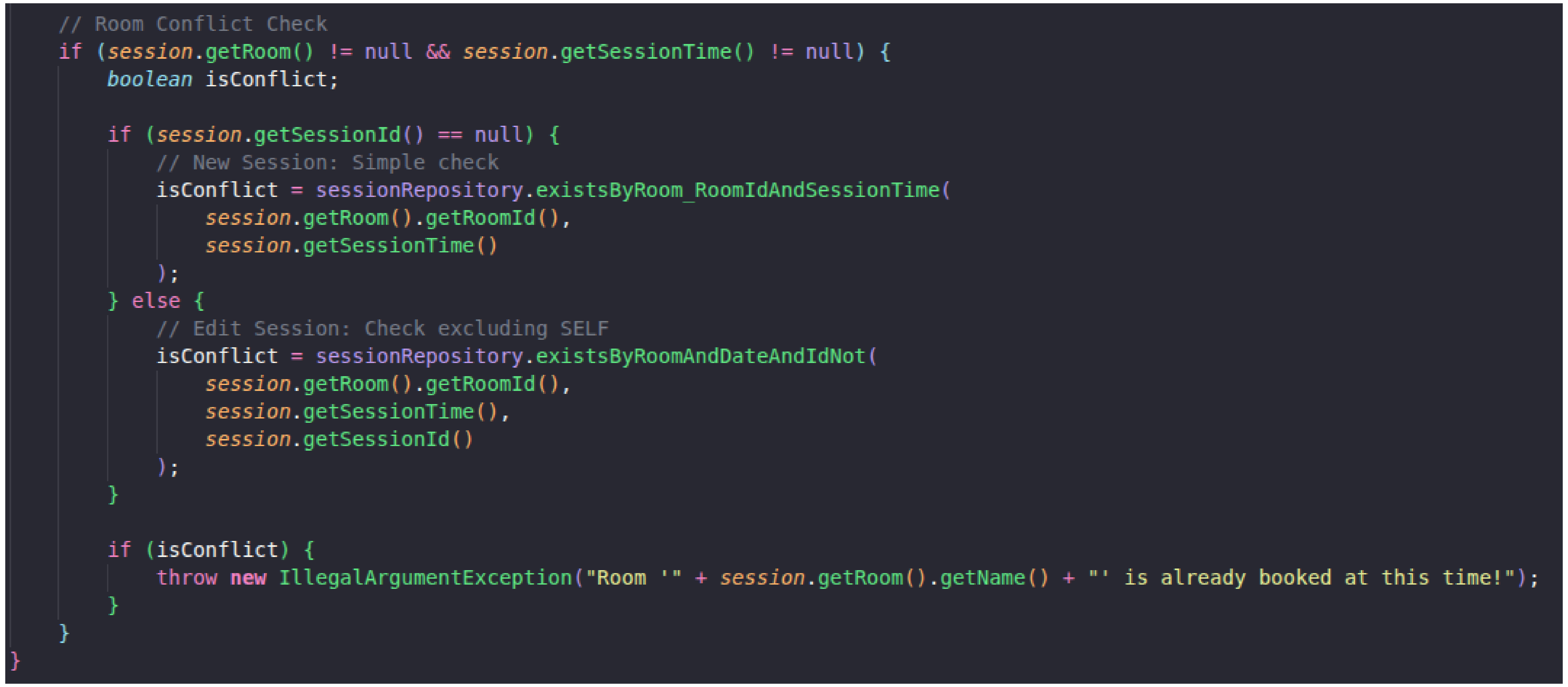
Session date validates within Conference dates



```java
    // Date Check
    if (session.getSessionTime() != null && session.getConference() != null) {
        LocalDateTime sessionDateTime = session.getSessionTime();
        LocalDateTime conferenceStart = session.getConference().getStartDate().atStartOfDay();
        LocalDateTime conferenceEnd = session.getConference().getEndDate().atTime(hour: 23, minute: 59, second: 59);

        if (sessionDateTime.isBefore(conferenceStart) || sessionDateTime.isAfter(conferenceEnd)) {
            throw new IllegalArgumentException(s: "Session time must be within conference dates");
        }
    }
```

IMPLEMENTATION

Prevent Room Conflict for Session



```
    // Room Conflict Check
    if (session.getRoom() != null && session.getSessionTime() != null) {
        boolean isConflict;

        if (session.getSessionId() == null) {
            // New Session: Simple check
            isConflict = sessionRepository.existsByRoom_RoomIdAndSessionTime(
                session.getRoom().getRoomId(),
                session.getSessionTime()
            );
        } else {
            // Edit Session: Check excluding SELF
            isConflict = sessionRepository.existsByRoomAndDateAndIdNot(
                session.getRoom().getRoomId(),
                session.getSessionTime(),
                session.getSessionId()
            );
        }

        if (isConflict) {
            throw new IllegalArgumentException("Room '" + session.getRoom().getName() + "' is already booked at this time!");
        }
    }
}
```

DEMONSTRATION

CONCLUSION

- DEVELOPED CONFERENCE AND SESSION MANAGEMENT SYSTEM
- IMPLEMENT SECURITY, VALIDATIONS, PASSWORD BCRYPT
- SHARED DATABASE WITH FLYWAY
- LEARNED TO WORK IN A TEAM
- HANDLED MESSY INTEGRATION BETWEEN LAYERS

CHALLENGES:

- SECURITY IMPLEMENT
- API AND FRONTEND PATH MISMATCH
- DATABASE MIGRATION MANAGEMENT (FLYWAY)
- CODE CONFLICTS WITH 5 PEOPLE
- MESSY BUGS IN API CONNECTIONS

FUTURE WORK

- ENHANCED COMMUNICATION (EMAIL)
- BETTER UI DESIGN
- SESSION FEEDBACK AND RATINGS
- PDF ATTENDENCE EXPORT
- VIEW AND EDIT USER PROFILE
- PRINT CERTIFICATE FOR PARTICIPANT PARTICIPATING

THANK YOU FOR YOUR PATIENT!