**CS102 - Spring 2024/2025**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructor:** | Uğur Güdükbay | **Section No:** | **3H** |
| **Assistant:** | Ebrar Bozkurt |
|  |  |  |  |

**~ Bilkom~**  
**Group0**

**Group Members**

|  |  |
| --- | --- |
| **Name** | **Surname** |
| **Ali İhsan** | **Sevindi** |
| **Elif** | **Bozkurt** |
| **Sıla** | **Bozkurt** |
| **Salih Mert** | **Uzun** |
| **(Utku** | **Kabukçu)** |

**Detailed Design Report V1**  
**Due: 26 April 2025**

# Introduction

This section presents the detailed backend design of the Bilkom project, focusing exclusively on the responsibilities and implementations of the backend developers: Elif Bozkurt and Salih Mert Uzun. The backend is responsible for handling user authentication, account verification, user data management, WebSocket communication, and providing secure REST APIs for the frontend to interact with. It is implemented using Spring Boot, with data persistence managed via Spring Data JPA.

# Detailed Design

## 2.1 Technologies Used

* Java 17
* Spring Boot 3.2.3
* Spring Data JPA
* Spring Security
* WebSocket (STOMP protocol)
* Maven (build tool)
* MySQL (database)
* Android Studio
* Java
* Retrofit (for HTTP networking)
* Glide (for image loading)
* Material Design Components

# 2.2 Backend Architecture Overview

The backend is implemented in Java with Spring Boot and follows a clear MVC-based, layered architecture. Each layer has a well-defined responsibility, ensuring separation of concerns and maintainability:

## Controller Layer

* Exposes RESTful HTTP endpoints for client interaction.
* Utilizes Spring's `@RestController` and `@RequestMapping` annotations to define API routes.
* Implements role-based access control using `@PreAuthorize` annotations to secure endpoints.

## Service Layer

* Implements business logic and validation.
* Utilizes Spring's `@Service` annotation for service classes.
* Handles transactions using `@Transactional` annotations to ensure data integrity.

## Repository Layer

* Provides data access using Spring Data JPA interfaces.
* Utilizes `@Repository` annotation for repository interfaces.
* Supports CRUD operations and custom queries through method naming conventions.

## Model Layer (Entity Layer)

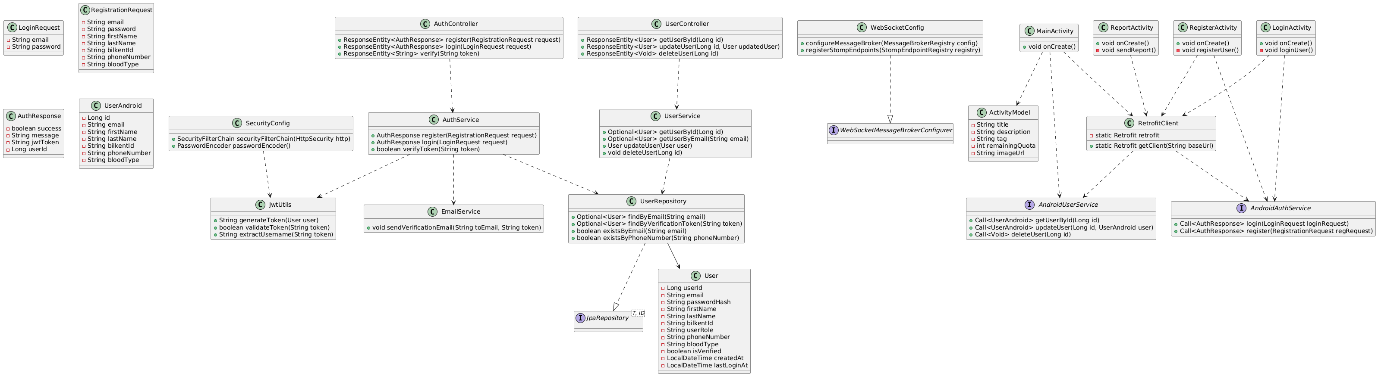
* Defines classes that map to database tables using JPA annotations.
* Utilizes `@Entity`, `@Table`, and other JPA annotations to define entity relationships and constraints.
* Includes enums for role and status management, enhancing type safety.

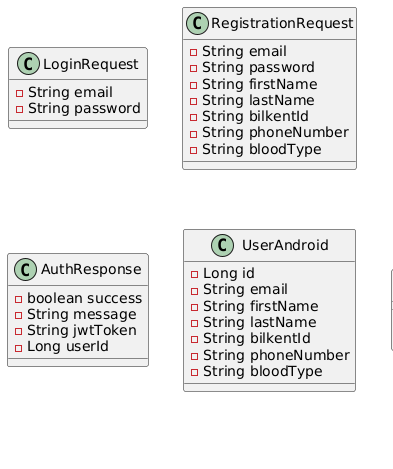
## Configuration Layer

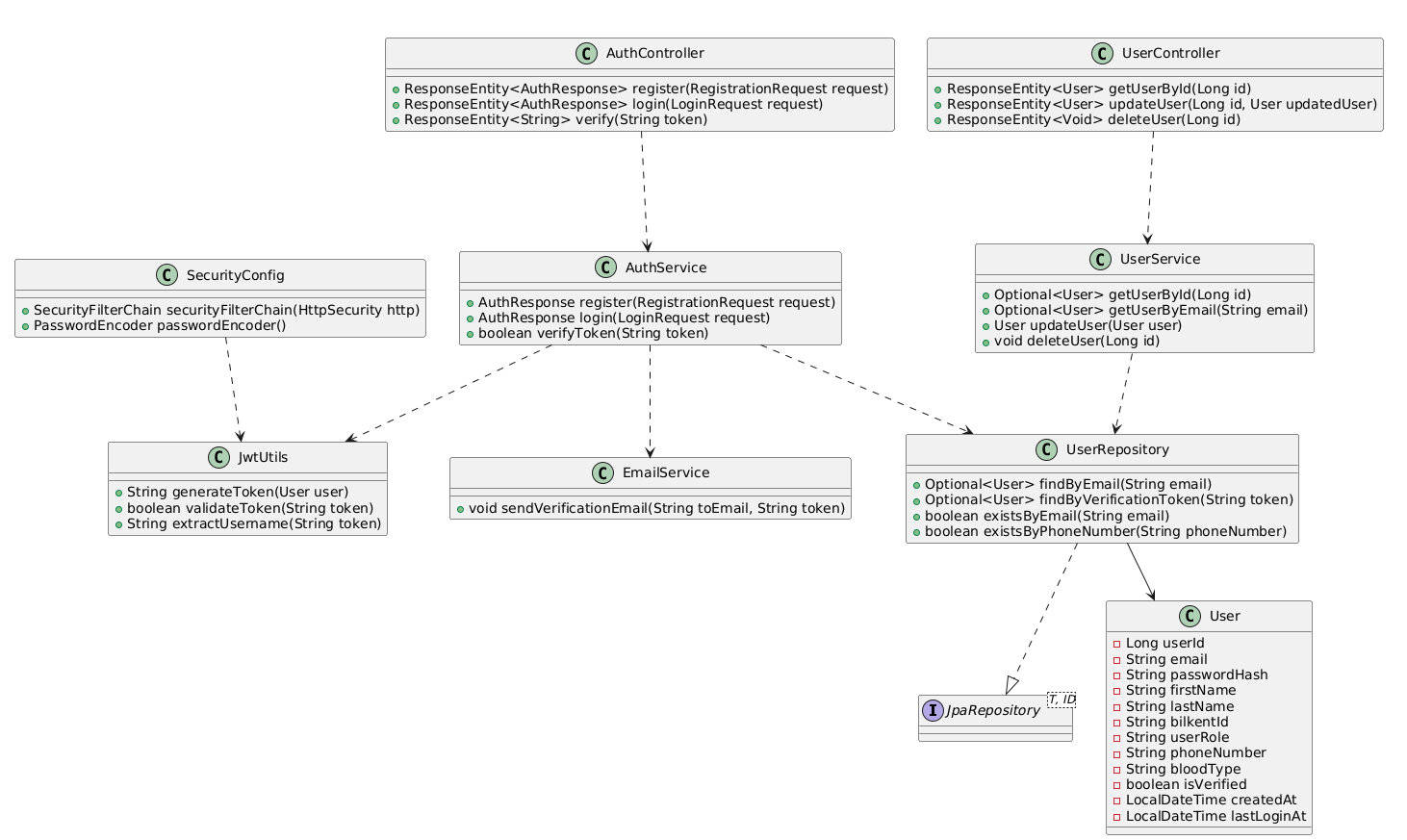
* Manages setup for security, WebSockets, CORS, and JWT.
* Utilizes `@Configuration` and `@EnableWebSecurity` annotations for security configurations.
* Configures JWT authentication with a custom `JwtAuthenticationFilter`.
* Sets up CORS policies using `WebMvcConfigurer` to allow cross-origin requests.
* Configures WebSockets for real-time communication using `@EnableWebSocketMessageBroker`.

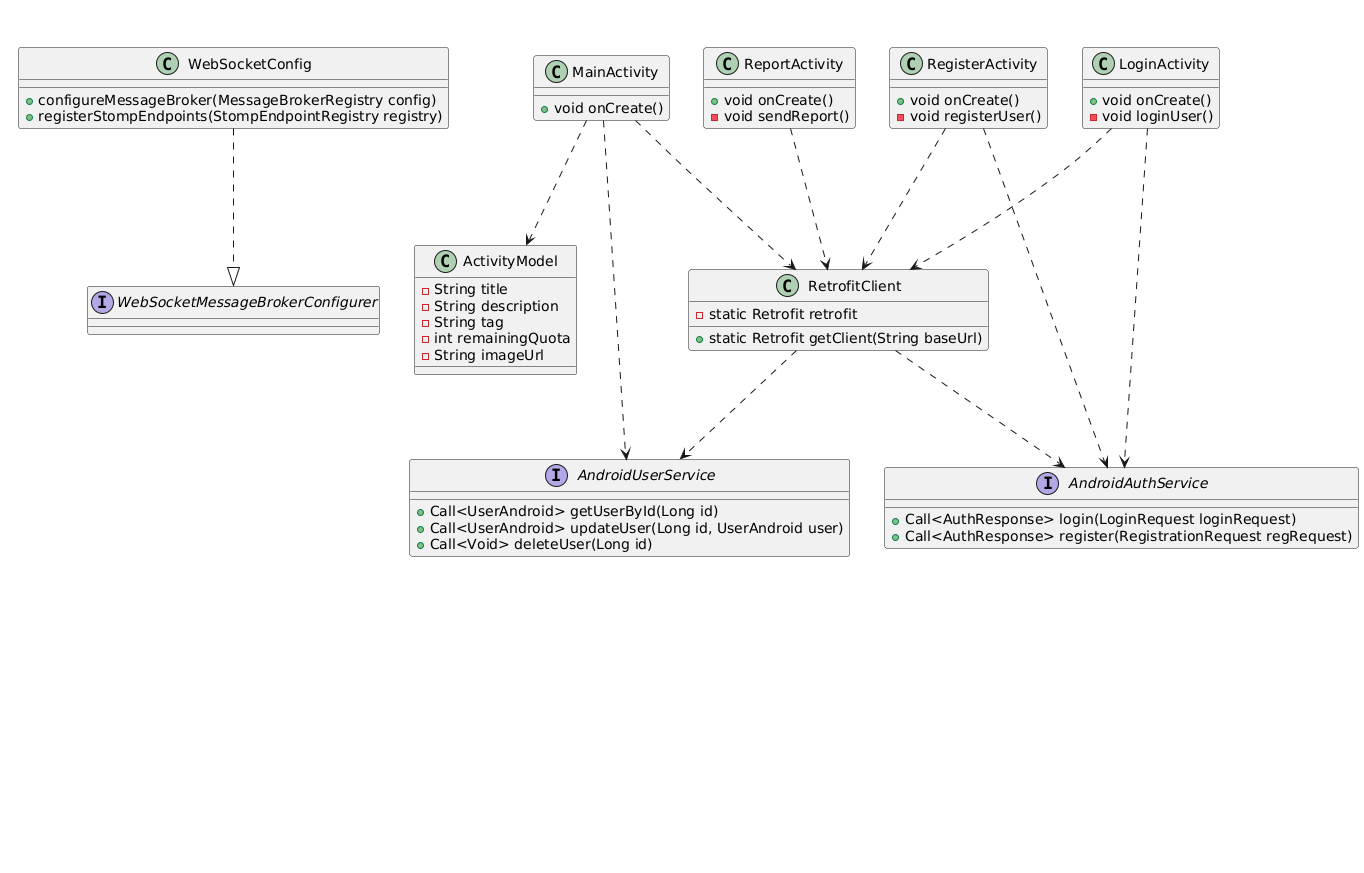
This architecture ensures a robust, scalable, and secure backend that supports the Bilkom platform's requirements for user management, event coordination, and real-time communication.

## 2.3 Core Design Details









## 2.4 Model Classes & Interfaces

Throughout this section, each subsection will indicate that the included files are in the given contributor’s responsibility.

### 2.4.1 Salih Mert Uzun, Backend

# Detailed Project Structure Overview

## Configuration Files

* EmailConfig.java: Configures email settings for the application, including SMTP server details and credentials.
* JpaConfig.java: Configures JPA auditing, allowing automatic population of audit-related fields in entities.
* SecurityConfig.java: Configures Spring Security settings, including authentication, authorization, and JWT filter integration.
* WebConfig.java: Configures CORS settings for the application, allowing cross-origin requests from specified origins.
* WebSocketConfig.java: Configures WebSocket settings, including endpoint and allowed origins.

## Controller Files

* AdminController.java: Manages admin-specific operations, such as approving or rejecting club registrations.
* AuthController.java: Handles authentication-related operations, including registration, login, logout, and password management.
* ClubController.java: Manages club-related operations, such as creating, updating, and retrieving club information.
* ClubExecutiveController.java: Manages operations related to club executives, including adding, updating, and removing executives.
* ClubMemberController.java: Manages club membership operations, including adding, updating, and removing members.
* ClubMembershipRequestController.java: Handles membership request operations, including creating, approving, and rejecting requests.
* ClubRegistrationController.java: Manages club registration operations, including registering new clubs and checking name availability.
* UserAdminController.java: Handles user management operations for admins, such as updating roles and activation status.
* UserController.java: Manages user-related operations, including creating, updating, and retrieving user information.
* UserSettingsController.java: Handles user settings operations, such as updating profiles, notification preferences, and privacy settings.

## DTO Files

* ApiResponse.java: A generic response wrapper for API responses, including data, message, and success status.
* AuthResponse.java: Represents authentication responses, including success status, message, token, and user ID.
* ClubDTO.java: Data transfer object for club information, including club details and associated members and executives.
* ClubExecutiveDTO.java: Represents club executive information, including user details and position.
* ClubMemberDTO.java: Represents club member information, including user details and membership status.
* ClubMembershipRequestDTO.java: Represents membership request information, including request details and status.
* ClubRegistrationRequestDTO.java: Represents club registration request information, including club details and executive information.
* LoginRequest.java: Represents login request data, including email and password.
* RegistrationRequest.java: Represents registration request data, including user details and credentials.
* UserDTO.java: Data transfer object for user information, including personal details and settings.

## Entity Files

* Club.java: Entity representing a club, including details, members, executives, and status.
* ClubExecutive.java: Entity representing a club executive, including user, club, and position details.
* ClubMember.java: Entity representing a club member, including user, club, and membership status.
* ClubMemberPK.java: Composite key class for ClubMember, combining club and member IDs.
* ClubMembershipRequest.java: Entity representing a membership request, including user, club, and request status.
* EmergencyAlert.java: Entity representing an emergency alert, including alert details and status.
* Event.java: Entity representing an event, including details, participants, and associated club.
* EventParticipant.java: Entity representing an event participant, including user and event details.
* Tag.java: Entity representing a tag, associated with events or users.
* User.java: Entity representing a user, including personal details, role, and settings.

## Enum Files

* ClubRegistrationStatus.java: Enum representing the status of a club registration (PENDING, APPROVED, REJECTED).
* UserRole.java: Enum representing user roles (USER, CLUB\_EXECUTIVE, CLUB\_HEAD, ADMIN) with a method for Spring Security role format.

## Exception Files

* BadRequestException.java: Custom exception for handling bad requests, annotated with HTTP status.

## Repository Files

* ClubExecutiveRepository.java: Repository for managing ClubExecutive entities, including custom queries.
* ClubMemberRepository.java: Repository for managing ClubMember entities, including custom queries.
* ClubMembershipRequestRepository.java: Repository for managing ClubMembershipRequest entities, including custom queries.
* ClubRepository.java: Repository for managing Club entities, including custom queries.
* UserRepository.java: Repository for managing User entities, including custom queries.

## Security Files

* CustomUserDetailsService.java: Custom implementation of UserDetailsService for loading user details from the database.
* JwtAuthenticationFilter.java: Filter for processing JWT tokens in requests, integrating with Spring Security.
* JwtUtils.java: Utility class for generating, validating, and blacklisting JWT tokens.

## Service Files

* AdminVerificationService.java: Service for handling admin verification operations, such as approving or rejecting clubs.
* AuthService.java: Service for handling authentication operations, including registration, login, and password management.
* ClubExecutiveService.java: Service for managing club executives, including adding, updating, and removing executives.
* ClubMemberService.java: Service for managing club members, including adding, updating, and removing members.
* ClubMembershipRequestService.java: Service for handling membership requests, including creating, approving, and rejecting requests.
* ClubRegistrationService.java: Service for managing club registrations, including registering new clubs and handling verification.
* ClubService.java: Service for managing clubs, including creating, updating, and retrieving club information.
* EmailService.java: Service for sending emails, including verification and notification emails.
* TokenBlacklistService.java: Service for managing blacklisted JWT tokens, ensuring invalid tokens are not accepted.
* UserService.java: Service for managing users, including creating, updating, and retrieving user information.

## SQL Files

* data.sql: Placeholder for initial data to be loaded into the database.
* schema.sql: SQL script for creating database tables and defining their structure.

## HTML Files

* club-registration-verification.html: Template for club registration verification emails (deleted).

## Properties Files

* application.properties: Configuration file for application settings, including server, database, JWT, email, and logging configurations.

2.4.2 Elif Bozkurt, Backend

### Controller Files

* **EventController.java**: Manages event operations, including event creation, participation, and retrieval.
* **WeatherController.java**: Provides current weather information by integrating with external APIs.
* **NewsController.java**: Fetches and serves news articles from external sources.
* **EmergencyAlertController.java**: Manages emergency alert notifications and retrieval.

### DTO Files

* **LoginRequest.java**: Represents login request data, including email and password.
* **EventDto.java**: Data transfer object for events, including title, date, and related information.
* **WeatherDto.java**: Data transfer object for weather information, including description, temperature, and icon.
* **NewsDto.java**: Data transfer object for news articles, including title, summary, and link.

### Entity Files

* **EmergencyAlert.java**: Entity representing an emergency alert, including alert details and status.
* **Event.java**: Entity representing an event, including title, details, date, and participants.
* **EventParticipant.java**: Entity representing the participation relationship between users and events.
* **EventParticipantPK.java**: Composite key class for EventParticipant, combining event ID and user ID.
* **Tag.java**: Entity representing a tag used for categorizing events or user interests.
* **Weather.java**: Entity representing stored weather data.
* **News.java**: Entity representing stored news articles fetched from external sources.

### Repository Files

* **EventRepository.java**: Repository for managing Event entities.
* **NewsRepository.java**: Repository for managing News entities.
* **WeatherRepository.java**: Repository for managing Weather entities.
* **EmergencyAlertRepository.java**: Repository for managing EmergencyAlert entities.
* **EventParticipantRepository.java**: Repository for managing EventParticipant entities.

### Service Files

* **EventService.java**: Service for managing events and event participation.
* **NewsService.java**: Service for fetching and serving news articles.
* **WeatherService.java**: Service for fetching and serving weather data.
* **EmergencyAlertService.java**: Service for managing emergency alerts and sending notifications.
* **NotificationService.java**: Service for managing WebSocket-based real-time notifications.

2.4.3 Sıla Bozkurt, Ali İhsan Sevindi. Frontend Documentation

# Application Components Overview

## Activities

* LoginActivity.java: Manages user login functionality, handles authentication requests, and redirects upon success.
* RegisterActivity.java: Manages user registration process, sending user information to the backend.
* MainActivity.java: Hosts the main navigation structure of the application after user login.
* ProfileActivity.java: Displays and manages user's profile information with options to update settings.
* NewsActivity.java: Fetches and displays news articles from the backend server.
* WeatherActivity.java: Fetches and displays current weather information for Bilkent.
* EmergencyAlertActivity.java: Displays emergency alerts and important urgent notifications.
* ClubActivity.java: Displays the list of available clubs that users can join or view.
* EventActivity.java: Displays available events and allows users to participate.
* CreateEventActivity.java: Allows club executives or admins to create new events.

## Adapters

* ClubAdapter.java: Binds club data to a RecyclerView for display in club listings.
* EventAdapter.java: Binds event data to a RecyclerView for display in event listings.
* NewsAdapter.java: Binds news article data to a RecyclerView for displaying news items.
* EmergencyAlertAdapter.java: Binds emergency alert data to a RecyclerView.

## Models

* LoginRequest.java: Represents the data structure for a user login request (email and password).
* RegistrationRequest.java: Represents the data structure for a user registration request (user information).
* AuthResponse.java: Represents the response structure after a successful login, including the token.
* User.java: Model representing user information such as name, email, and phone number.
* Club.java: Model representing a club's basic information (name, description, etc.).
* Event.java: Model representing an event's basic information (title, description, date, etc.).
* News.java: Model representing a news article's title, link, and summary.
* Weather.java: Model representing weather details (description, temperature, icon).
* EmergencyAlert.java: Model representing emergency alerts with details.

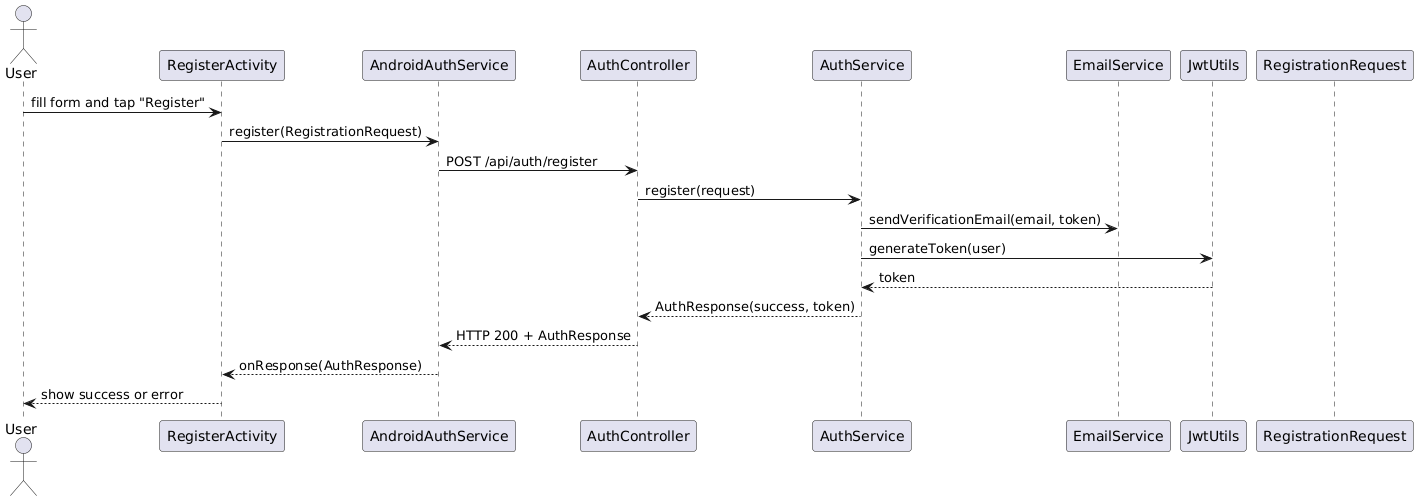
## API Service Interfaces

* AuthService.java: Defines authentication-related API endpoints such as login and registration.
* UserService.java: Defines user-related API endpoints for fetching and updating user data.
* ClubService.java: Defines club-related API endpoints for fetching club information.
* EventService.java: Defines event-related API endpoints for creating and joining events.
* NewsService.java: Defines API endpoints for fetching news articles.
* WeatherService.java: Defines API endpoints for fetching weather information.
* EmergencyAlertService.java: Defines API endpoints for fetching emergency alerts.

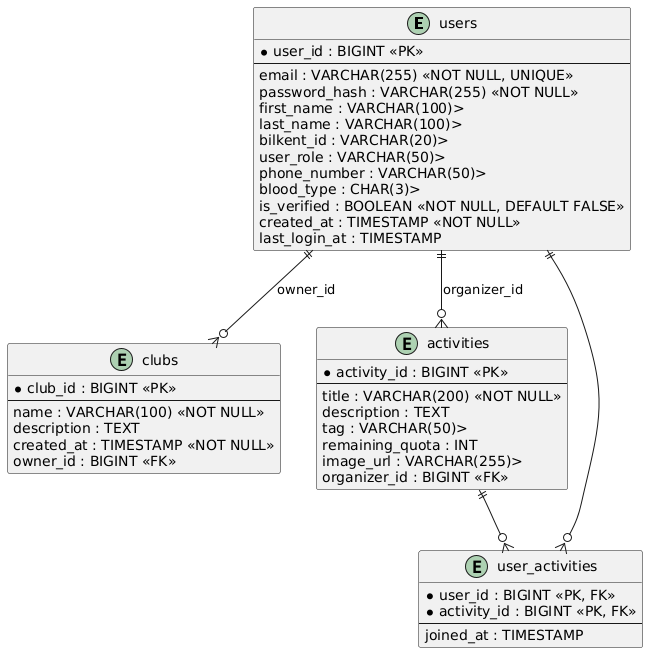
## Utility Classes

* ApiClient.java: Configures Retrofit client for API communication with the backend.
* SessionManager.java: Manages local storage of user session data such as JWT tokens.

## 2.5 Behavioral Diagrams

* 
* **User** fills out the form in **RegisterActivity** and taps “Register.”
* **RegisterActivity** calls **AndroidAuthService.register()**, which sends an HTTP POST to **AuthController**.
* **AuthController** forwards the request to **AuthService.register()**, where:
* **EmailService** sends a verification email with a secure token.
* **JwtUtils** generates a JWT for the new user.
* **AuthService** returns an AuthResponse back to the controller, which then replies over HTTP to **AndroidAuthService**.
* **RegisterActivity** receives the response and shows either a success message or an error to the **User**.

## 2.6 Database Schema



This ER diagram illustrates the system’s core tables and their relationships: the users table stores all account information, the clubs and activities tables hold club and event data and are each linked to a user via owner\_id or organizer\_id in a one-to-many relationship, and the user\_activities table acts as a join table modeling the many-to-many relationship between users and activities, capturing participation timestamps in the joined\_at field.

# Database Tables Overview

## - users (linked to User.java)

Stores user information including email, password, personal details, and role.

## - clubs (linked to Club.java)

Stores club information, including name, description, and status. References the users table for the club head.

## - club\_executives (linked to ClubExecutive.java)

Stores information about club executives, including their position and active status. References the users and clubs tables.

## - club\_members (linked to ClubMember.java)

Stores information about club members, including their membership status. References the users and clubs tables.

## - events (linked to Event.java)

Stores event details, including name, description, and participant information. References the users and clubs tables for event creators and associated clubs.

## - tags (linked to Tag.java)

Stores tags associated with events and users. References the events and users tables.

## - event\_participants (linked to EventParticipant.java)

Stores information about participants in events. References the events and users tables.

## - emergency\_alerts (linked to EmergencyAlert.java)

Stores emergency alert information, including type, description, and contact details.

## 2.7 Security Configuration

The SecurityConfig class configures the filter chain to authenticate and authorize requests, disables CSRF for stateless JWT usage, adds a custom JWT authentication filter before the UsernamePasswordAuthenticationFilter, and exposes a BCryptPasswordEncoder bean.

## 2.8 Real-Time Communication

The WebSocketConfig class enables STOMP over WebSocket, configures an in-memory message broker at /topic and /queue destinations, and registers the /ws endpoint with SockJS fallback for clients.

## 2.9 Exception Handling

A GlobalExceptionHandler annotated with @ControllerAdvice catches application-specific exceptions (e.g. UserNotFoundException) and validation errors, maps them to meaningful HTTP status codes, and returns a standardized ErrorResponse JSON payload.

# Task Assignment

**Database**: Salih Mert Uzun

**Backend:** Elif Bozkurt, Salih Mert Uzun

**Frontend (UI):** Sıla Bozkurt, Ali İhsan Sevindi

# What We Could / Could Not

### We Could:

* **Clubs:** Allow users to view, create, and join clubs.
* **Events:** Implement event creation, joining, and management.
* **User:** Complete user registration, login, and profile management.
* **Tag System:** Enable tagging for events and user interests.
* **Emergency Alerts Page:** Notify users about urgent blood transfusion needs.
* **Weather Page:** Show live weather information for Bilkent.
* **News Feed:** Display up-to-date news articles from Bilkent News on the app.

### We Couldn't:

* **Carpooling System:** We faced an unexpected downfall that costed one of our group members who got an FZ grade (Utku Kabukçu). Since this feaure was marked as a possible extension, we decided to remove it from the project.

# Reflections

Throughout the project, working effectively as a team was crucial. We learned how to coordinate tasks, communicate frequently, and support each other during both the easy and challenging phases. At the beginning, we had to learn and adapt to various systems and technologies that were new to us, such as backend frameworks, database management, and API integration. In addition to backend development, we also gained experience in building an Android application, which was a completely new area for most of us. We were able to apply the foundational concepts we learned in the course, but also had to go beyond them — learning new tools and technologies independently. Adapting to and using systems we had no prior experience with was often difficult, but ultimately it made the project more rewarding and strengthened our technical skills significantly.

# Summary & Conclusions

The Bilkom project integrates both frontend and backend components to deliver a cohesive mobile solution for university life. The frontend prioritizes accessibility and clean user interfaces, while the backend ensures data security, real-time responsiveness, and robust business logic. Together, these layers provide essential services such as user authentication, content updates, and safety alerts. The architecture is modular, maintainable, and scalable, making it suitable for future extensions like club management or notification systems.