SafeSpace Final Documentation

1. Project Overview

Project Name: SafeSpace

Description: A social media platform with real-time posts.

Objectives: To understand and apply software production processes.

Scope:

- **Authentication:**
    1. Registration
    2. Login
    3. JSON Web Token (JWT)
- **User features:**
    1. Edit user information
    2. Add a profile picture
    3. Manage friendships
    4. Messaging between users
- **Post Features:**
    1. Create posts.
    2. Add or remove likes from posts
        - **Comment features:**
            1. Create comments for posts
- **UI Features:**
    1. Light and dark themes
    2. Filter posts (all / friends only)

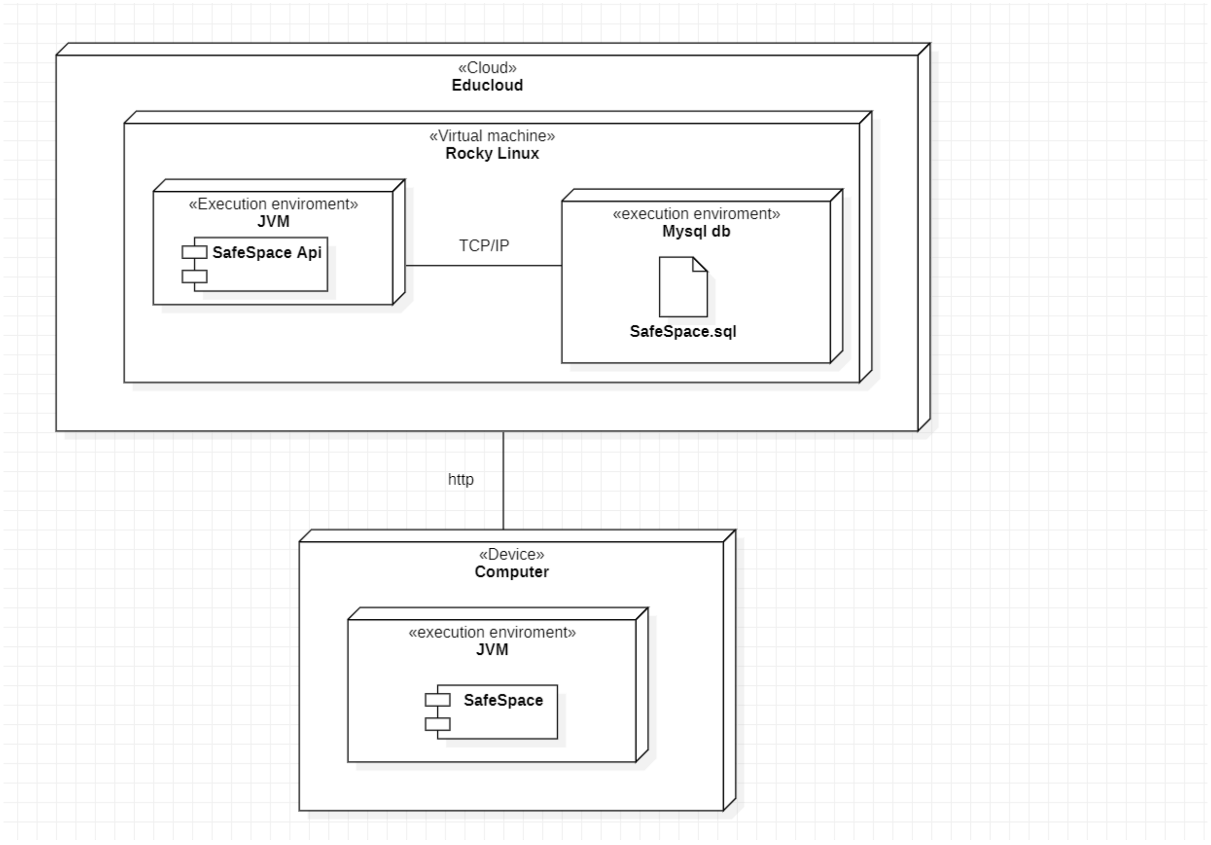
2. Architecture

Architecture Type: Client-server architecture with a RESTful API.

Components:

- User Interface: Java with JavaFX framework, Maven build automation tool
- Server: Java with Spring Boot framework, Maven build automation tool

- Database: MySQL



3. Technologies Used

Programming Languages: The user interface and server are implemented using the Java programming language.

Tools: IntelliJ, SceneBuilder, Postman, HeidiSQL

Quality Assurance: Qodana, SonarLint, Junit, TestFX

Project Management: Trello, Miro, Discord

User Interface: JavaFX, Maven, CSS

Server: Spring Boot, Maven

Database: MySQL

Version Control: Git, GitHub

CI/CD: Jenkins

Virtual Environment: Docker, Rocky Linux

4. Software Design

Authentication

- **Objective:** The authentication system ensures that only registered users can access the platform's features, and their identity is securely managed through JWTs
- **Flow:**
  - **Registration:** When a new user registers, their username is validated against the server to ensure uniqueness. If the username is available, the user is added to the database. JSON Web Token (JWT) is created for the user and the user is then logged in on the client
  - **Login:** Users can log in using their existing credentials. Upon successful login, a JWT is generated and sent back to the client. This token contains claims about the user and is used to authenticate subsequent API requests.
  - **JWT Verification:** For all endpoints (except login and registration), the user must provide a valid JWT in the request header. The backend verifies this token before processing the request, ensuring that the user is authenticated.

User Feature

- **Objective:** Users are individual entities that interact with the platform by following other users and engaging with posts and messages.
- **Details:**
  - When a new user is created, there are no predefined relationships to other entities
  - Users can establish relationships with other users by following them, which are stored in the friendship table. This table captures connections between users enabling them to manage friendships.

Post Feature

- **Objective:** Users can create, like and comment on posts. The platform supports real time global feed where posts are shared with other users

- **Details:**
  - Posts have **Many-to-Many** relationships with the following entities:
    - **Creator:** The user who created the post.
    - **Likers:** Users who like the post.
    - **Commenters:** Users who comment on the post.

  - When a post is created, an **EventListener** is triggered. The listener listens for new posts and, upon detecting a new post, it sends a **PostDTO** object back to the client using **Server-Sent Events (SSE)**. This enables the posts to be rendered in real-time on the global feed for all users.

Comment Feature

- **Objective:** Users can interact with posts by adding comments, creating a more engaging and interactive social experience.

- **Details:**
  - Comments have a **Many-to-One** relationship with Users (each comment is associated with one user).
  - A **Many-to-One** relationship also exists between Comments and Posts (each comment belongs to one post).
  - Comments are stored in the **post_comment** table, which holds the foreign keys of both the Post and User entities.

Like Feature

- **Objective:** Users can like or remove likes from posts, providing engagement on posts.
- **Details:**
  - The Like feature follows a **Many-to-Many** relationship between Users and Posts.
  - When a user likes or removes a like from a post, an API request is triggered, and the server processes the request.
  - An **EventListener** listens for these like events, and when a like is added or removed, it creates a **LikeDTO** object and sends it to the client via SSE.
  - This ensures that the like status is updated in real time on the client side for all the users viewing the posts.

Comment Feature

- **Objective:** Users can send messages to each other, allowing for private conversations and interactions.
- **Details:**
  - When a user sends a message, a **SendsMessage** entity is created to store the relationship between the two users and message.
  - The **SendsMessage** entity golds **Many-to-One** relationships with both the User entities (sender and recipient), and a **One-to-One** relationship with the **Message** entity (each message is associated with one SendsMessage).

SSE Feature in Client

- **Objective:** Design and implement a mechanism in the client application to establish a Server-Sent Events (SSE) connection, handle incoming data events, and ensure seamless rendering across the user interface, regardless of the user’s current screen.

- **Details**

  Establishing the SSE connection

  1. **Initialization**
     - Upon user login, the client initiates an SSE connection by making an HTTP GET request to the server's /api/v1/events endpoint.
     - To prevent blocking the main application thread, the connection is established on a separate thread, ensuring that other client processes remain responsive.

  2. **Configuration**
     - The connection is set up to listen for specific types of events such as “new_post”, “like_added”, and “like_removed”.

  Handling Incoming Events

  1. **Event Parsing:**
     - Incoming events are processed in the Feed class through the processEvent method.
     - The method extracts two key substrings from the event payload:
       - eventType: Specifies type of event (“new_post”, “like_added” or “like_removed”).
       - eventDataLine: Contains the actual payload of the event.
     - The extracted data is parsed into appropriate object models using the Gson library for JSON deserialization.

  2. **Data Object Construction:**
     - Based on the event type, the processEvent method uses Gson to parse the eventData into a structured object, such as: Post-object or Like-object.

Storing Event Data

1. **Shared storage:**
    - All parsed event objects are stored in the SharedData singleton class, which acts as a centralized data store.
    - This design ensures that event data persists even when the user navigates away from the home screen.

2. **Concurrency Consideration:**
    - Since events arrive asynchronously, access to the SharedData class is synchronized to prevent race conditions and ensure thread safety.
    - This is particularly important in multi-threaded environments where simultaneous reads and writes may occur.

Rendering UI Updates

1. **Home Screen Updates:**
    - The MainController class contains the logic responsible for rendering data stored in SharedData whenever an event occurs.
    - If the user is on the home screen, the engine processes the event immediately, updating the relevant UI elements in real-time.

2. **Deferred Updates:**
    - If the user is not on the home screen, events are queued in SharedData.
    - Upon returning to the home screen, the startQueueProcessing engine in the MainController checks for queued events and updates the UI accordingly.

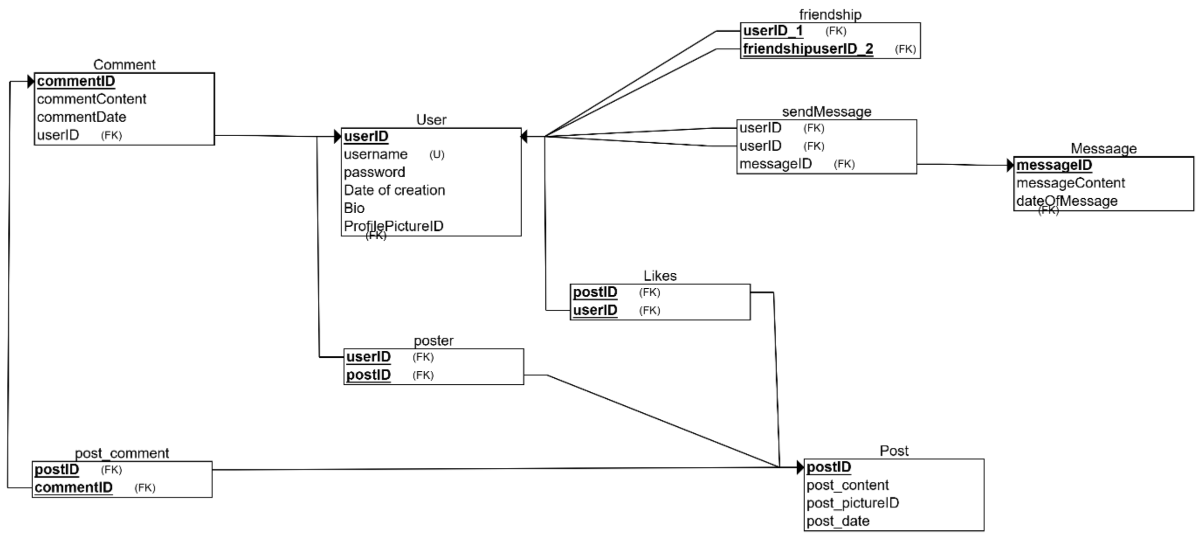
Error Handling and Logging

1. **Event Parsing Errors:**
   - If the payload of an event is malformed or incomplete, the processEvent method logs the error for debugging purposes and discards the event to prevent state corruption.
   - This ensures stability for application and while providing valuable diagnostic data.
2. **Connection Errors:**
   - When the application starts, the client automatically checks its connection to the server.

Conclusion

- The implementation of SSE (Server-Sent Events) as the primary data source from the server offers a scalable and efficient solution for managing real-time updates. This approach enables the client to handle and render user data responsively. By utilizing a structured event processing pipeline, centralized data storage, and thread-safe synchronization, the client delivers a seamless and consistent user experience across all screens, regardless of navigation changes or connection interruptions.
- This design enhances the responsiveness of the application and lays a robust foundation for handling high-volume, real-time events in a scalable manner.

5. Database Model



Centre table is the User table. Table contains: userID which is primary key, username which is unique, password, date of creation is created automatically when field is created, Bio and profile picture id which referents to the picture name in pictures folder in the server storage

Friendship table has two (2) user ids. With a friendship table we can manage the users following and friendship.

The message table contains messageID which is the primary key, message content and date of creation is created automatically.

The sendMessage table contains two (2) foreign user keys (sender and receiver) and messages foreign key.

The post table contains postID which is primary key, text content, picture id if it has one and automatically generated date of creation.

The poster table connects the post and creator of the post.

The likes table contains postID and userID. Let's easily track the likes on the posts.

The comment table contains commentID which is the primary key, user foreign key, text content and automatically generated date of creation.

The post_comment contains postID and commentID. Let’s connect the comments to the posts.

6. API & JavaDoc Documentation

API documentation has been created using Postman.

Link to the Postman API documentation

Link to Javadoc documentation

Link to Server Javadoc documentation

7. Testing Strategy

Unit Testing:

- **Tool:** JUnit
- **Purpose:** Validate the functionality of individual methods and components.

Coverage Testing:

- **Tool:** JaCoCo
- **Purpose:** Measure the percentage of the codebase exercised by unit tests to ensure critical paths are tested.

Integration Testing:

- **Tools:** Postman
- **Purpose:** Testing and development of APIs

End-to-End Testing:

- **Tools:** TestFX
- **Purpose:** Testing both server and client.

Acceptance Testing:

- **Method:** Manual testing
- **Approach:**

- Team members conducted manual tests based on detailed test cases created during development.
- Test results were thoroughly documented, and any identified issues were addressed.

9. User Documentation

Installation Instructions

Prerequisites

Before installing, ensure the following tools are installed on your computer:

- Java Developement Kit (JDK) Version 21 or higher
- Apache Maven versio 3.13 or higher
- Metropolia VPN: GlobalProtect

Clone the Repository

1. Open terminal or command prompt.
2. Run the following command to clone the repository:

```
git clone https://github.com/hinmiro/SafeSpace.git
```

3. Navigate to the project directory:

```
cd SafeSpace/
```

Build the Project

1. Ensure all dependencies are downloaded by running:

```
mvn clean install
```

2. Verify the build completes without errors.

Run the Project

Note: Ensure VPN is active during this step.

1. Run the project using the Maven command:

```
mvn javafx:run
```

Ensure you are in the project's root directory when running the command.

10. Maintenance Plan

1. Bug Fixes and Updates

- Monitor user feedback and logs for identifying bugs.
- Release patches regularly to fix bugs and address security vulnerabilities.

2. Feature Enchantments

- Periodically review user needs and add new features as required.
- Ensure backward compatibility when making updates

3. Security Checks

- Keep the application's dependencies up to date with the latest features and changes.

4. Documentation updates

- Keep the API and user documentation up to date

11. Summary

The **SafeSpace** project has been developed as a comprehensive social media platform with real-time interaction capabilities.

Project GitHub.

Group Trello.

Challenges:

- Server-sent events (SSE). Implementing this feature took a lot of effort from multiple members.
- Adding docker implementation.

Future Improvements:

- Implement pagination for the main feed to improve performance at scale. Instead of rendering all posts at login, fetch and display posts in smaller batches (e.g., 10 at a time) and load more on user request.
- Real time messaging using Server-Sent Events (SSE).
- Notifications
- Deleting own posts / messages.
- Mobile version or change client platform to React.