

Streaming Platform

Name: Tudor Ovidiu Bichilie

Group: 30234

Table of Contents

[Deliverable 1 3](#_Toc163407154)

[Project Specification 3](#_Toc163407155)

[Functional Requirements 3](#_Toc163407156)

[Use Case Model 1 3](#_Toc163407157)

[Use Cases Identification 3](#_Toc163407158)

[UML Use Case Diagram 5](#_Toc163407159)

[Supplementary Specification 5](#_Toc163407160)

[Non-functional Requirements 5](#_Toc163407161)

[Design Constraints 5](#_Toc163407162)

[Glossary 6](#_Toc163407163)

[Deliverable 2 6](#_Toc163407164)

[Domain Model 6](#_Toc163407165)

[Architectural Design 7](#_Toc163407166)

[Conceptual Architecture 7](#_Toc163407167)

[Package Design 7](#_Toc163407168)

[Component and Deployment Diagram 7](#_Toc163407169)

[Deliverable 3 7](#_Toc163407170)

[Design Model 7](#_Toc163407171)

[Dynamic Behavior 7](#_Toc163407172)

[Class Diagram 7](#_Toc163407173)

[Data Model 7](#_Toc163407174)

[System Testing 7](#_Toc163407175)

[Future Improvements 7](#_Toc163407176)

[Conclusion 7](#_Toc163407177)

[Bibliography 7](#_Toc163407178)

# Deliverable 1

## Project Specification

This project involves developing a Scala Play2 API for a streaming application. The API will manage users, streamers, and administrators, providing functionality such as user registration, stream management, post creation, live streaming, messaging, and comments. The database schema includes tables for users, credentials, posts, live streams, messages, and comments. The technology stack consists of Scala Play2 framework, Slick for database management, and PostgreSQL for database storage.

## Functional Requirements

Functional Requirements

1. User Management:

* Registration, login, and profile updates for users.
* Password recovery/reset functionality.

1. Streamer Management:

* Creation and management of live streams.
* Interaction with the audience during live streams.

1. Post Management:

* CRUD operations for posts.
* Commenting on posts.

1. Live Stream Management:

* Accessing ongoing live streams.
* Controls for starting, pausing, and ending live streams.

1. Messaging System:

* Sending direct messages.
* Notifications for new messages.

1. Comment Management:

* Leaving comments on posts and live streams.
* Moderation of comments by streamers and administrators.

## Use Case Model 1

### Use Cases Identification

Use-Case: User Registration

Level: Primary

Primary Actor: User

Main Success Scenario:

1. User provides registration details (username, email, password).
2. System validates the provided information.
3. User account is successfully created.

Extensions:

* If the provided email is already registered, display an error message.
* If the password does not meet the security requirements, prompt the user to choose a stronger password.

Use-Case: Create Post

Level: Primary

Primary Actor: User

Main Success Scenario:

1. User composes a post with content.
2. User submits the post.
3. Post is successfully created and added to the system.

Extensions:

* If the content of the post is empty, prompt the user to enter content.

Use-Case: Manage Live Stream

Level: Primary

Primary Actor: Streamer

Main Success Scenario:

1. Streamer starts a live stream.
2. Streamer interacts with viewers during the live stream.
3. Streamer ends the live stream.

Extensions:

* If technical issues occur during the live stream, provide troubleshooting steps.
* If inappropriate behavior is observed from viewers, allow the streamer to moderate or block them.

### UML Use Case Diagram

A diagram of a company

Description automatically generated

## Supplementary Specification

### Non-functional Requirements

1. Performance: The system should be able to handle a high volume of concurrent users and live streams without significant degradation in response time. This is crucial for ensuring a seamless streaming experience for users and streamers alike.
2. Security: The system should enforce strong authentication mechanisms to protect user accounts and sensitive data. Additionally, access controls should be implemented to ensure that only authorized users have access to certain features or data. This is essential for maintaining the integrity and confidentiality of user information.
3. Scalability: The system should be designed to scale horizontally to accommodate increasing numbers of users and streams over time. This will allow the platform to handle growth effectively and maintain performance under heavy loads without requiring significant architectural changes.
4. Reliability: The system should be highly available and resilient to failures. It should be able to recover gracefully from errors or downtime, ensuring minimal disruption to users and streamers. This is critical for maintaining user trust and satisfaction with the platform.

Explanation: These non-functional requirements are suitable for the implementation because they directly address key aspects of the system's performance, security, scalability, and reliability. Given the nature of a streaming application where real-time interaction is paramount, ensuring optimal performance, security, and reliability is essential for providing a satisfactory user experience.

### Design Constraints

* Technology Stack: The system must be developed using the Scala Play2 framework for the backend API, Slick for database management, and PostgreSQL for data storage, as mandated by project requirements.
* Use of Development Tools: The development team must adhere to the use of specific development tools such as IntelliJ IDEA for coding, Git for version control, and Docker for containerization, as specified by project guidelines.
* Architectural Constraints: The system architecture must follow a microservices-based approach, with each component responsible for specific functionalities, ensuring modularity and scalability.

## Glossary

1. User: An individual who accesses the streaming platform to view live streams, create posts, interact with other users, and engage in messaging.
2. Streamer: A user who broadcasts live streams on the platform, interacts with viewers during streams, and manages their streaming activities.
3. Post: A piece of content created by a user, containing text, images, or links, which can be viewed and interacted with by other users.
4. Live Stream: A real-time broadcast of audio or video content by a streamer, which can be accessed and viewed by users on the platform.
5. Authentication: The process of verifying the identity of a user, typically through credentials such as username and password, to grant access to the platform.
6. Authorization: The process of determining whether a user has permission to access certain features or data within the platform, based on their role and privileges.
7. Horizontal Scaling: Increasing the capacity of the system by adding more resources or instances horizontally, such as deploying multiple server instances to handle increased traffic.
8. Microservices: A software architectural approach where the system is composed of small, independent services that communicate with each other through APIs, promoting flexibility, scalability, and maintainability.

# Deliverable 2

## Domain Model

[Define the domain model and create the conceptual class diagrams]

## Architectural Design

### Conceptual Architecture

[Define the system’s conceptual architecture; use an architectural style and pattern - highlight its use and motivate your choice.]

### Package Design

[Create a package diagram]

### Component and Deployment Diagram

[Create the component and deployment diagrams.]

# Deliverable 3

## Design Model

### Dynamic Behavior

[Create the interaction diagrams (2 sequence) for 2 relevant scenarios]

### Class Diagram

[Create the UML class diagram; apply GoF patterns and motivate your choice]

## Data Model

[Create the data model for the system.]

# System Testing

[Describe the testing methides and some test cases.]

# Future Improvements

[Present some features that apply to the application scope.]

# Conclusion

# Bibliography