Design Document

Design Decisions

**Why use spring boot and MySQL?**

Spring Boot is a framework that simplifies the setup and development of Java-based applications. It provides a wide range of pre-built features and configurations, enabling developers to quickly start building applications without getting bogged down in boilerplate code. It is also particularly well-suited for backend development. It provides a robust foundation for creating RESTful APIs and handling database operations. MySQL, being a widely-used relational database, integrates seamlessly with Spring Boot, making it easy to store and retrieve data.

**Why use React?**

React is a powerful JavaScript library for building user interfaces, especially for single-page applications (SPAs). When combined with Spring Boot for the backend, React allows for a clear separation of concerns between the frontend and backend development, promoting a more organized and maintainable codebase. It also enables the creation of dynamic and interactive user interfaces by efficiently managing the state of the application and efficiently updating the DOM. It allows for real-time updates and provides a smooth user experience, making it a great choice for building modern web applications.

C4 Model Diagrams

C1

A diagram of a system

Description automatically generated

Here we can see the system context, a very general look at the structure of the application. We have two different types of users who can interact with the application, and also an external system for emailing users.

C2

A diagram of a diagram

Description automatically generated

In the C2 we have a more technical view of the application, where its broken down into 3 layers: Front end, API Application and the database. The users only interact with the front end, with the front end sending API requests to the API application. The front end cannot interact with the database, only the API application can.

**C3**

A diagram of a diagram

Description automatically generated

The C3 is much more detailed than the C1 and C2 diagrams. The back end is separated by 4 REST Controllers: Forum controller, User Controller, Messaging Controller and Review Controller. Each of these controllers then calls a service that then can read or write to the MySQL database. Here is depicted that process that is the same for all 4 controllers.

**C4**

A diagram of a software company

Description automatically generated

This is the C4 class diagram. In it you can see all domain classes.

**Classes and explanations**

**Chat message:**

Has 4 datafields:

sentAt, which stores the date of when the message was sent.

messageTo, which stores to whom the message was sent

messageFrom which stores the sender of the message

messageContent which stores the actual message

**FriendRequest:**

Has 2 datafields:

toUser, which stores the receiver of the friend request

fromUser, which stores the sender of the friend request

**Friendships:**

Has 2 datafields:

Friends, which stores a list of every user that has is friends with the user datafield

user, which stores the user of which the previous data fields refers to

**User:**

Has 6 datafields, all are self explanatory

**Forum:**

The forum class represents a forum for a given video game, where users can make posts about anything they wish regarding the said game. This is not a place for reviews, as that’s that job of the VideoGameReview class. It is intended to work like websites such as reddit.

Connected to the forum class we have ForumPost which is simply a post under a certain video games forum, and connected to the ForumPost we have the PostComment, which is simply a comment under a post in a forum.

Forum posts and comments can also have positive and negative ratings, as demonstrated by the ForumPostCommentRating class and the ForumPostRating class.

**VideoGame:**

In order to have forums first we need video games, so the application will have a main video game page in which we can find general information about the video game, its reviews and its forum.

**VideoGameReview:**

This class stores a positive or negative rating on the video game, who submitted the review, the date of the review and obviously the description of the review.

**Repo Interfaces:**

Every domain class will have a repo associated with it in order to follow the single responsibility principle. So Forum and every Forum related class will be under the ForumRepo, for example.

**Use cases:**

Since the use cases work the same way for every class, in the C4 diagram “X” can be replaced by the domain classes name as to not bloat the C4 with unnecessary data.

CI/CD Diagram

A diagram of a pipeline

Description automatically generated

Here we can see the process through which our CI/CD goes through.

The diagram illustrates the architecture of a CI/CD pipeline involving the following components:

GitLab:

1. Represented as a central node in the CI/CD setup.
2. Initiates and manages the CI/CD pipeline.
3. Interacts with both the developer's machine and the pipeline runner.

My / Developers machine:

1. Depicted as a separate node connected to GitLab.
2. Responsible for code development and pushes/pulls code to/from GitLab.

Pipeline Runner:

1. Connected to GitLab and performs various tasks related to pipeline execution.
2. Triggered by GitLab, executes the pipeline tasks, and sends status updates/logs back to GitLab.

SonarQube:

1. Connected to the pipeline runner for code analysis.
2. Receives code from the pipeline runner for analysis and provides analysis results back to GitLab.

How it woks:

1. The developer interacts with GitLab from their machine, pushing and pulling code as part of the development process.
2. GitLab triggers the pipeline runner, which executes the CI/CD pipeline tasks.
3. The pipeline runner communicates with GitLab, sending pipeline status updates and logs.
4. Additionally, the pipeline runner performs code analysis tasks by interacting with the SonarQube instance.
5. SonarQube provides analysis results back to GitLab.