|  |  |
| --- | --- |
| A picture of a winding road and trees  Software architecture document  MetaRate Project | Ooi, John Q.Q.J.  17-3-2023 |

**Table of Contents**

[Architecture constraints and design decisions 2](#_Toc137160656)

[Application of the SOLID principles 2](#_Toc137160657)

[Why Spring Boot? 2](#_Toc137160658)

[Why React? 2](#_Toc137160659)

[Why MySQL? 2](#_Toc137160660)

[CI Diagram 3](#_Toc137160661)

[C4 Model Diagrams 4](#_Toc137160662)

[System Context (C1) 4](#_Toc137160663)

[Containers and tech choices (C2) 5](#_Toc137160664)

[Components (C3) 6](#_Toc137160665)

[Implementation details (C4) 7](#_Toc137160666)

# Architecture constraints and design decisions

The backend will be written in Java, the Spring Boot framework will be used so that it is possible to create an API application for the backend. The backend will be in charge of making the connection with the database and providing data to the frontend.

The frontend will be responsible for representing the data in a single page application as well as sending requests to the API application (backend). It will be written using JavaScript and React.

The project will follow some of the SOLID principles to make sure the project will be as scalable, maintainable, and extendable as possible.

## Application of the SOLID principles

This application will follow the following SOLID principles:

* Single Responsibility Principle: Each component of the layer should have its own task. The GameRepository should only serve the purpose of interacting with the database and reading/writing from it.
* Open/Closed Principle: When implementing new functions to the application, you would only have to extend the already existing classes and there should not be any need to modify them.
* Dependency Inversion Principle: High-level modules should not depend on low-level modules, but both should depend on abstractions. In this application, we could ensure this principle by ensuring that the higher-level components, such as the GamesController, depend on abstractions such as the interfaces for the use cases, rather than depending on implementations of those use cases.

## Why Spring Boot?

Spring Boot is extremely flexible and can be used with a vast amount of different technologies and libraries. This flexibility also makes it easy to integrate with other parts of a software system. Spring Boot makes it possible to build REST APIs using a powerful and flexible platform to ensure high functionality.

## Why React?

React's component-based architecture enables for the building of reusable UI components that can be easily put together to build sophisticated interfaces. This can reduce the amount of code needed to build the UI compared to if it was all written in plain JavaScript.

## Why MySQL?

I have already had experience in MySQL in previous semesters and it’s very easy to use with docker. Because of the ease of access and my previous experience in MySQL I think it was very easy for me to choose which software to use for my database.

# CI Diagram

1. A picture containing text, font, handwriting, line

   Description automatically generatedDeveloper: Represents the developer who writes and pushes code changes to the GitLab repository.
2. GitLab Repository: The central repository that stores the source code and tracks changes made by developers.
3. GitLab Runner: Executes the CI/CD pipeline by pulling code from the repository and executing the defined pipeline stages.
4. Build: Compiles the source code, resolves dependencies, and creates the build artifacts.
5. Tests: Runs automated tests to verify the functionality of the code.
6. SonarQube Analysis: Conducts code analysis, identifies code quality issues, detects bugs, and enforces coding standards.
7. Staging Deployment: Deploys the code to a staging environment for further testing and review.
8. Production Deployment: Deploys the code to the production environment if all previous stages pass successfully.

# C4 Model Diagrams

## System Context (C1)

Diagram

Description automatically generated

This is the C1 model, a regular user uses the Internet Review System to view information about games and writes reviews. An admin is able to add/edit game information using the Software System.

## Containers and tech choices (C2)

Diagram, text, application

Description automatically generated

The regular user and the admin both interact with the Single-Page application. The Single-Page application is the frontend and is responsible for representing the data and making API calls to the API application. The API application provides the functionality for the review application and interacts with the database to pass information to the single page application and to save information that was called from the Single-Page application.

## Components (C3)

Diagram

Description automatically generated

The Single-Page application makes API calls to the Game Controller which provides the interaction between the Single-Page application for all “Game” services. This means that it is only responsible for all CRUD services for a game. The Game Service will process the data and send it to the database. The Game Data Access interacts with the database to save the new data.

## Implementation details (C4)

A picture containing text, screenshot, diagram, parallel

Description automatically generated

The process of creating a game starts with the GamesController, which receives the necessary inputs. It then passes the inputs to the CreateGameUseCase, which is implemented by CreateGameUseCaseImpl. The CreateGameUseCaseImpl processes the inputs and generates a CreateGameResponse, which contains relevant information about the game creation. The CreateGameRequest holds the required data for creating a game. Finally, the CreateGameUseCaseImpl interacts with the GameRepository, which handles the storage and retrieval of game-related data. Together, these components form a structured flow for creating a game in the REST API.