**Software Architecture Document**

**Gamehub**

|  |
| --- |
| **Date : 06.06.2024** |
| **Version : 1.3** |
| **State : Completed** |
| **Author : Catalin Mihai Popoiu** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Changes** | **State** |
| 1.0 | 28.03.2024 | Catalin Mihai Popoiu | Added initial version and added C1 – C3 level diagrams | In progress |
| 1.1 | 13.05.2024 | Catalin Mihai Popoiu | Added CI Diagram with explanations and updated existing diagrams | In progress |
| 1.2 | 29.05.2024 | Catalin Mihai Popoiu | Updated document and added the 4th layer of the C4 diagram. | In progress |
| 1.3 | 06.06.2024 | Catalin Mihai Popoiu | Updated CI/CD diagram to the current version with explanations. | Completed |

**Table of contents:**

[1. Introduction 3](#_Toc145410716)

[1.1 Purpose 3](#_Toc145410717)

[1.2 Scope 3](#_Toc145410718)

[2. System Context 3](#_Toc145410722)

[2.1 Business Context 4](#_Toc145410723)

[3. Containers and Technology Choices 5](#_Toc145410725)

[3.1 Backend Container 5](#_Toc145410726)

[3.2 Frontend Container 5](#_Toc145410727)

[4. Components 6](#_Toc145410729)

[4.1 Backend Components 6](#_Toc145410730)

[4.2 Frontend Components](#_Toc145410730) 8

[5. CI Pipeline](#_Toc145410729) 8

[6. Conclusion](#_Toc145410729) 8

**1. Introduction**

***1.1 Purpose***

This Software Architecture Document (SAD) offers a detailed overview of the architectural framework for GameHub. It delineates the high-level design choices, structural components, and technologies that underpin the development and functionality of the system.

***1.2 Scope***

GameHub is engineered to serve as a social networking platform tailored for the gaming community. It enables users to explore, share, and discuss their gaming experiences, discover new games, and connect with fellow gamers. The system is built using Java Spring Boot for the backend and React for the frontend, ensuring a scalable, maintainable, and interactive application.

**2. System Context**

***2.1 Business Context***

GameHub was conceived as a forward-thinking startup dedicated to enhancing digital interactions within the gaming industry. Located in Eindhoven, The Netherlands, Gamehub aims to revolutionize how gamers interact, share, and discover content through the platform.

***2.2 System Overview***

GameHub consists of a backend powered by Java Spring Boot, which provides RESTful APIs, and a frontend developed with React, facilitating a dynamic and engaging user interface.

A diagram of a game

Description automatically generated

**3. Containers and Technology Choices**

***3.1 Backend Container***

***3.1.1 Spring Boot Application***

Description: The backend services of GameHub, offering robust API support and server-side logic.

Technology Choices:

* Spring Boot: Selected for its rapid development features, auto-configuration, and extensive Spring ecosystem support, making it ideal for creating microservices.

- RESTful API: Ensures statelessness and a uniform interface, facilitating easy integration and communication between frontend and backend components.

***3.2 Frontend Container***

***3.2.1 React Application***

Description: Provides an interactive and responsive user interface for GameHub.

Technology Choices:

A diagram of a computer flowchart

Description automatically generated- React: Chosen for its efficiency in updating and rendering components, which is crucial for real-time interactions on the platform. The decision is also influenced by its component-based architecture, facilitating easier development and maintenance.

**4. Components**

**4.1 Backend Components**

The backend architecture consists of three layers, Persistence, Business, and Controller. Their description is as follows:

**Persistence:**

Responsibility: Manages data storage and retrieval, interacting with the MySQL database to ensure efficient and secure data handling.

Components:

* Entity Classes: Map to MySQL tables for data representation.
* Repositories: Leverage JpaRepository for ORM, simplifying database interactions.

**Business:**

Responsibility: Houses the application's core logic, processing data from the Persistence layer for use in the application.

Components:

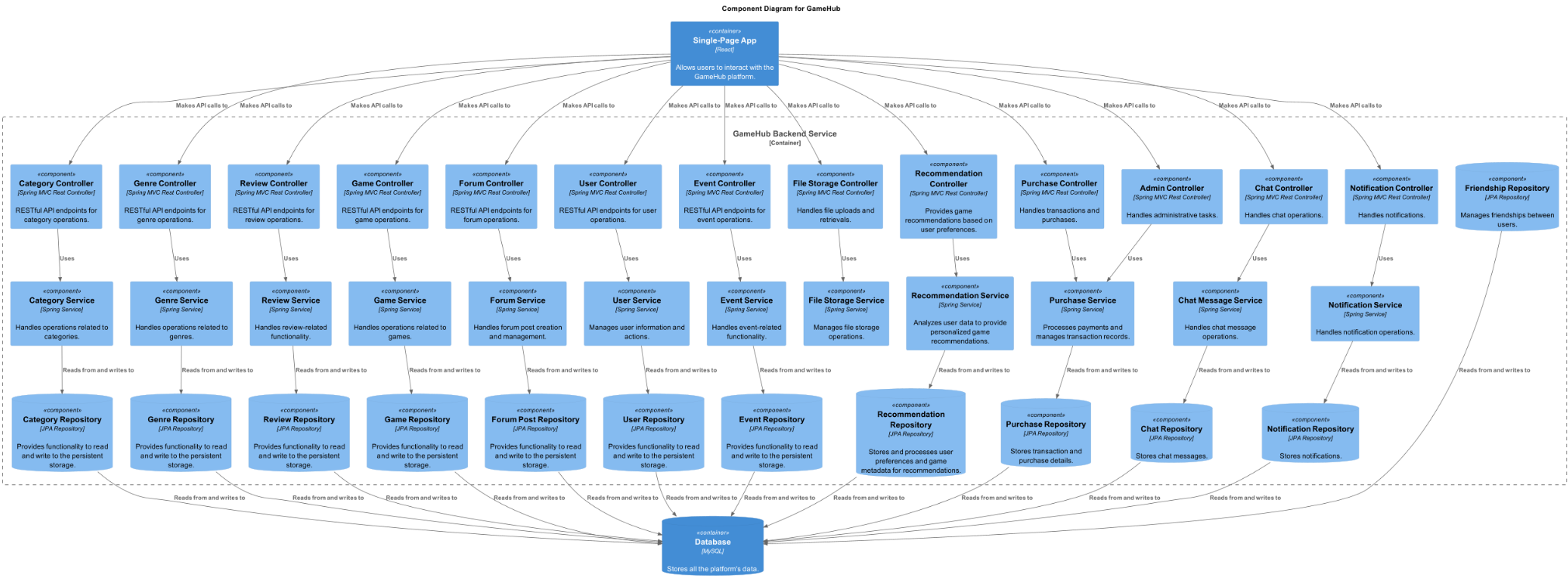
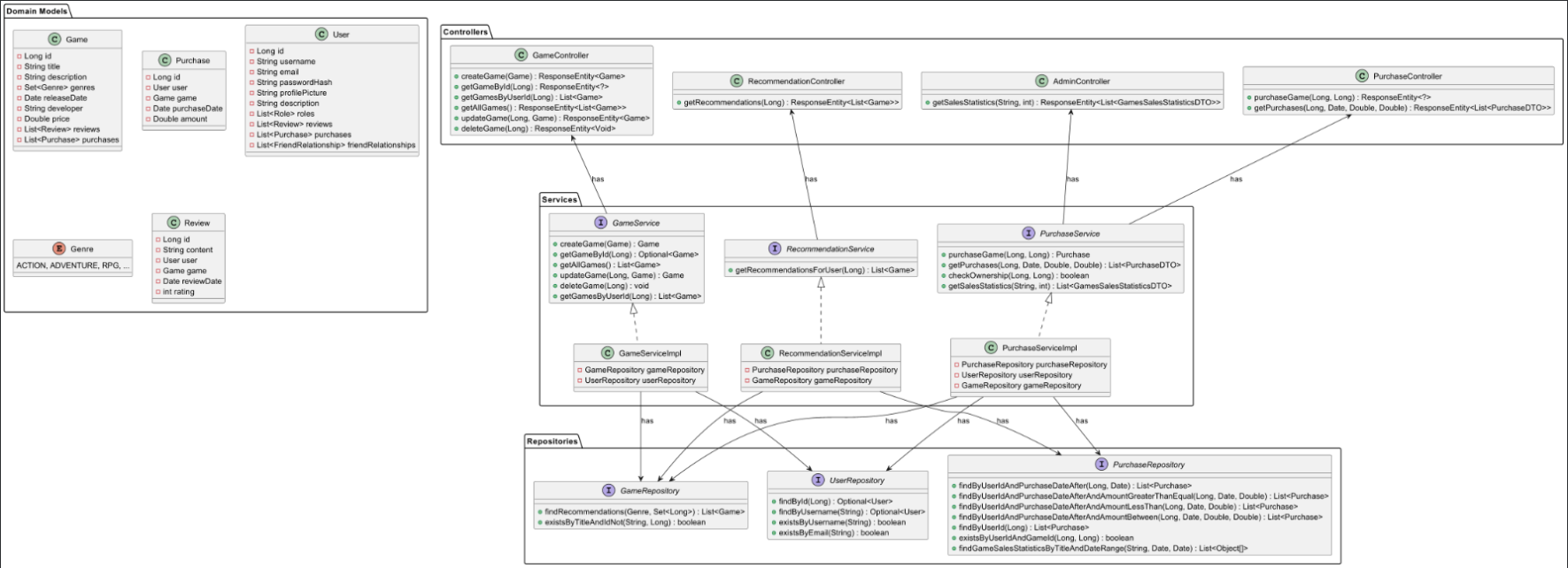
* Service Classes: Contain business logic, transforming data for the Controller layer.
* DTOs (Data Transfer Objects): Facilitate data transfer within the application, adhering to the YAGNI principle by avoiding unnecessary base classes.

**Controller:**

Responsibility: Manages data flow between the UI and business logic, processing user inputs and returning the appropriate responses.

Components:

* Controllers: Interface with both the business layer and the frontend, orchestrating the application's response to user actions.



(UML Class Diagram snippet, showing layering structure for the whole Games/Purchasing games systems.)

**4.2 Frontend Components**

**API:**

Responsibility: Handles API calls to the backend, employing HTTP methods to interact with the server.

**Components:**

Responsibility: Serves as the reusable building blocks of the UI, facilitating the creation of dynamic and interactive pages.

**Pages:**

Responsibility: Renders the application's various views, utilizing components for displaying content and interfacing with the user.

**5. CI Pipeline**

**Stages:**

The pipeline defines four stages: build, test, SonarQube-check, and deploy.

The pipeline stages are running through the GitLab Runner, that’s setup locally, same as the Docker environment.

**Build Stage**:

The build stage uses Gradle to assemble the project.

It ensures that the project’s code can be compiled and built successfully before moving on to further stages.

**Test Stage**:

The test stage utilizes Gradle to run tests for the project.  
 It verifies the functionality of the code through automated tests, ensuring the code meets the required standards and behaves as expected.

**Sonar Stage:** Executes Gradle tasks (“test” and “jacocoTestReport”) and then triggers SonarQube analysis using the “gradle sonar” command with an additional parameter for the quality gate. The “allow\_failure: true” attribute indicates that the pipeline can continue even if this stage fails, which mostly occurs due to the quality gate not being satisfied.  
 This stage ensures that code quality and security issues are identified and addressed. It provides a comprehensive analysis of the codebase, highlighting potential bugs, code smells, and vulnerabilities.

**Deploy Stage:**

The deploy stage involves deploying the built artifacts to a staging or production environment. This stage typically includes the following steps:

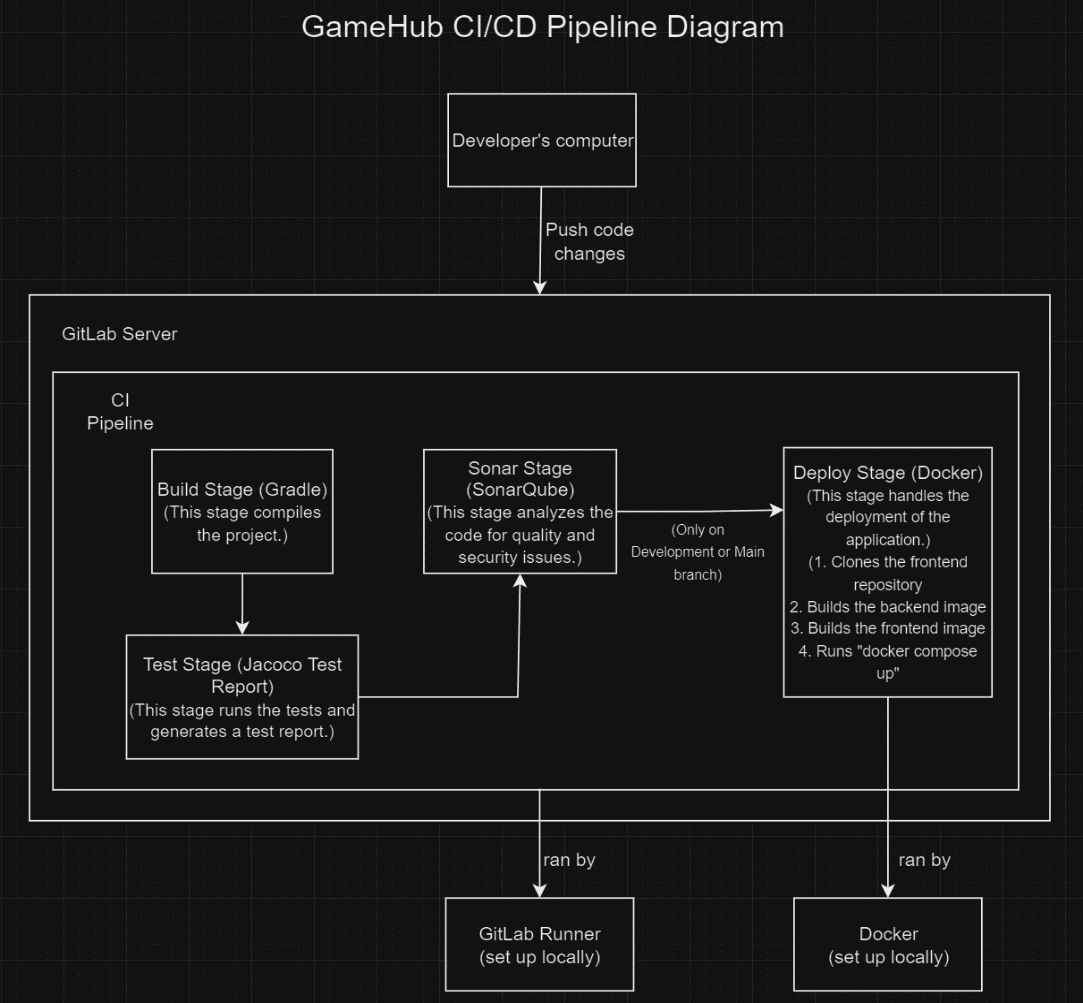
Setup Docker Environment: Uses Docker to set up the necessary environment for deployment.

Build Docker Images: Creates Docker images for both the backend and frontend services using the respective Dockerfiles.

Deploy Services: Uses docker-compose to deploy the application, ensuring that all services are up and running. This can include starting containers, setting up networks, and ensuring that the application is accessible.

Environment Configuration: Configures environment-specific settings, such as database connections, API keys, and other necessary configurations for the application to run smoothly.

The deploy stage ensures that the latest version of the application is available for testing or use in a real-world environment, facilitating continuous integration and continuous delivery practices.



(CI/CD) Pipeline diagram showing current setup)

**6. Conclusion**

This Software Architecture Document encapsulates the architectural framework, design rationale, and technological underpinnings of the GameHub project. By embracing principles such as SOLID and YAGNI, alongside leveraging modern, efficient technologies like Spring Boot and React, GameHub is poised to offer a scalable, maintainable, and highly interactive platform for the gaming community.