KASINTU

Web Collection Game

Software Architecture Document

Semester 3 - Individual Project

Airell Rasendriya Bachtiar
airell.bachtiar@student.fontys.nl

Table of Contents

1. Version

Version	Date	Description
0.1	25-03-2022	Make a software architecture document, added chapters and description to it.
0.2	14-04-2022	Update document based on feedback. Add database design and user interface design.
0.3	03-06-2022	Added reference based on APA formatting, updated C4 model, and added CI diagram.
0.4	10-06-2022	Edited CI/CD Diagram
1.0	16-06-2022	Version 1.0 of software architecture document

2. Introduction

2.1. Purpose

The purpose of this document is to provide detailed architecture design of a game called Kasintu for individual project at Fontys University of Applied Science.

2.2. Definitions, Acronyms, and Abbreviations

- Gacha: A method inspired by toy vending machine where you can get a toy randomly from what the vending machine provide. Instead of toy vending machine, here it is turned into an application game where you can get an item, in this project we called a creature, randomly with a set number of chances.
- Summon or Pull: The action performed when you are getting a creature from the gacha.
- Banner: The place where you summon or pull creature. Banner contains a list of creatures in which the player can obtained and a chance or percentage of how many chances you can obtain a specific creature.

3. System Overview

3.1. Description

This game is called Kasintu which means bird. Kasintu is a collection-based game where player can collect as much as they want. What they will collect is a different type of birds that is real and fictional thus the meaning of Kasintu is bird, a game where you collect birds. From now on these birds will be called creature.

3.2. Main User Activities

The main feature of this game is called a gacha system. Gacha system is where player can get a chance to receive a virtual item using in game currency. This is where player mainly get a new creature that will be release or has been released by the developer. They called this action of obtaining new creature as a summoning or pulling. In this case we will call this action as summon or summoning. As the where they summoning these creatures is called a banner. A banner contains a certain amount or all the creature available that can be obtained by the player who summoned on that banner.

For a future feature, Kasintu will also include a marketplace and breeding system. Marketplace is where player can but, sell or trade creatures from the other player. Breeding system is where player can breed their own creature to make new creature which may become rarer that the previous creature.

3.3. Project Goal

The goal of this project is to have a game that will entertain user by collecting creatures and to collect everything the game provides. For better user experience, this game will have to has a fast user interface to make user does not need to wait long in between action or input and a secure application so that data from user cannot be tracked or stolen by a third party.

4. Design Considerations

4.1. Operational Environment

This game can be played in any web browser the user chooses to use. It doesn't need any download to play the game.

4.2. Development Method

Agile approach will be used in this project using Scrum method. Agile is an approach where application is developed in small increments and Scrum is a method from Agile which project will be updated in sprints. This project will last for 17 weeks, and it will be divided into 6 sprints. Every 3 weeks there will be a sprint done where we review what we did on those 3 weeks. If the main feature, which is the gacha system, is done before week 17 timeframe, additional features such as marketplace and breeding system will be implemented. A meeting will be conducted at least once per sprint.

4.3. Architectural Strategy

The model shown for the web application is a C4 model. The model will be updated throughout the development of this project.

5. High Level Architecture

SOLID will be applied in this project. To guarantee of applying SOLID, below is rules that should be followed during this project:

- Single Responsibility Principle: Every class should have a few of specific function that are within a category of that class.
- Open Closed Principle: Every function should return or have an unchangeable product, for example if a function return X, it should always return X regardless the changes.
- Liskov Substitution Principle: Interface should be used extensively. This should prevent an error whenever we change class when the functions are the same.
- Interface Segregation: Interface should be made only for specific purpose to avoid unused functions. This is the same as single responsibility principle, but it is applied to interface instead of class.
- Dependency Inversion Principle: From using interface as parameters, this principle is applied. This way, we can change into different class that implement same interface. This principle will be used for switching to local database and external database.

6. Architectural Representation

In this document, C4 architecture model will be depicted the software architecture as accurately as possible. It allows to visualize a software architecture at various levels of detail. The model is divided into 4 different parts, Context, Containers, Components, and Code.

a) Context

System context diagram shows the software system you are building and how it fits into the world in terms of the people who use it and the other software systems it interacts with.

b) Containers
Container diagram zooms into the software system, and shows the containers, such as application and data stores, which make up that software system. Technology decisions are also a key part of this diagram.

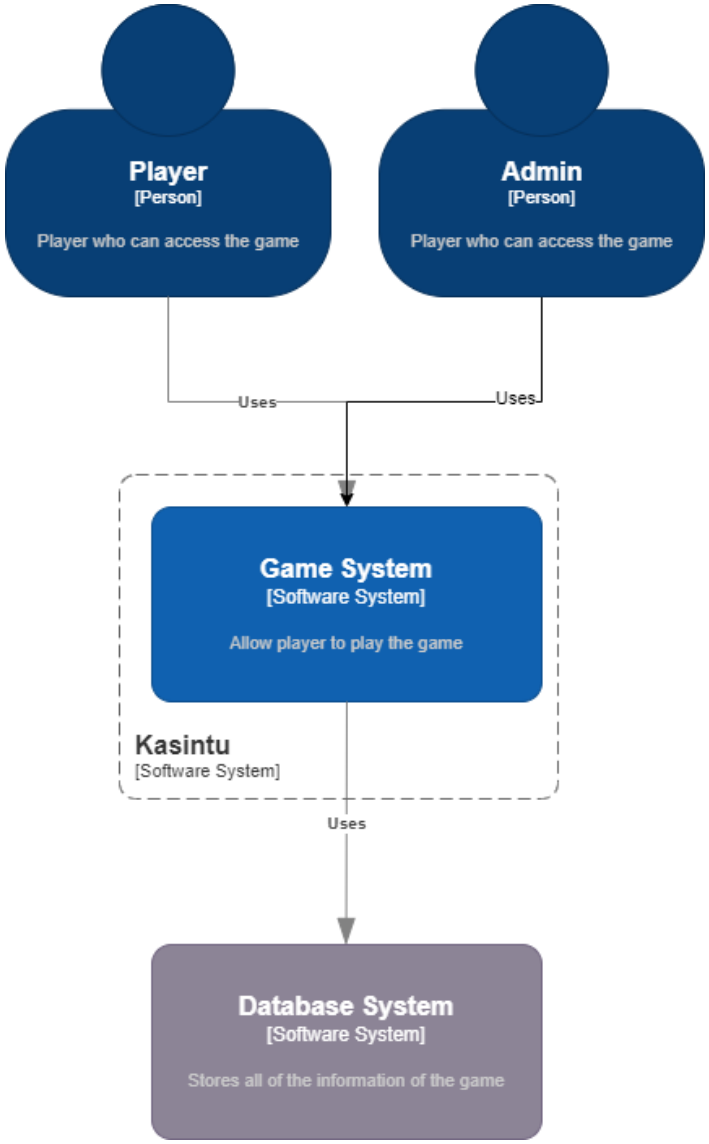
c) Components
Component diagram zooms into an individual container to show the components inside it. These components should map to real abstractions, like grouping code, in the codebase.

d) Code
Code diagrams zoom into an individual component to show how that component is implemented.
(Brown, 2018)

7. System Architecture

7.1. Context

For this diagram, we have 2 main users, player, and admin. They can operate the system as user. Player can access and play the game and admin acts as the user that can add more features to the game such as creatures and banners. The database of the system is outside the game system to preserve security for the data.

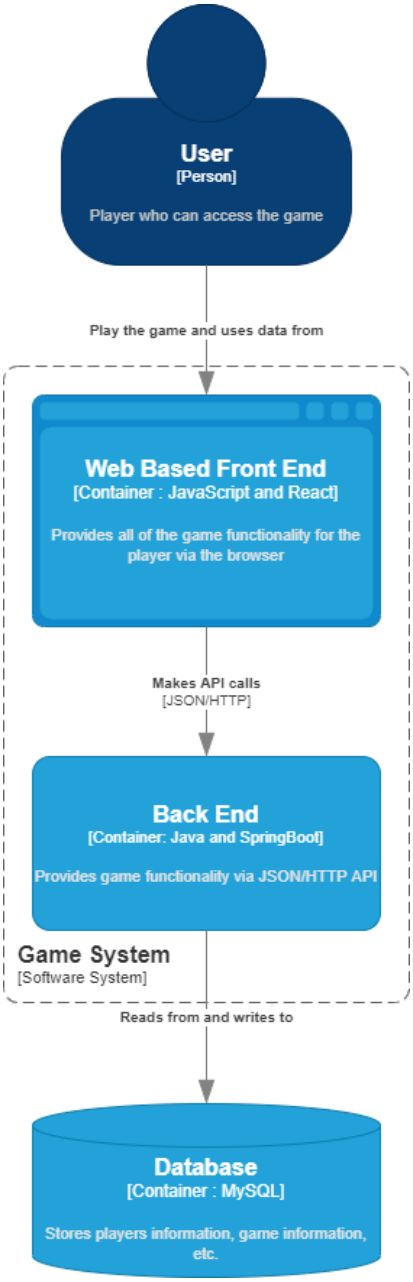


7.2. Containers and Tech Choices

For the front end, JavaScript and React will be used. JavaScript is used for a better user interface and experience for the web application. And react is a useful framework to help with the website front end and a connection with the back end. React will be used because it's one of the best open-source frameworks and one of the most popular. So, if there are any problems, a lot of sources can help to fix it.

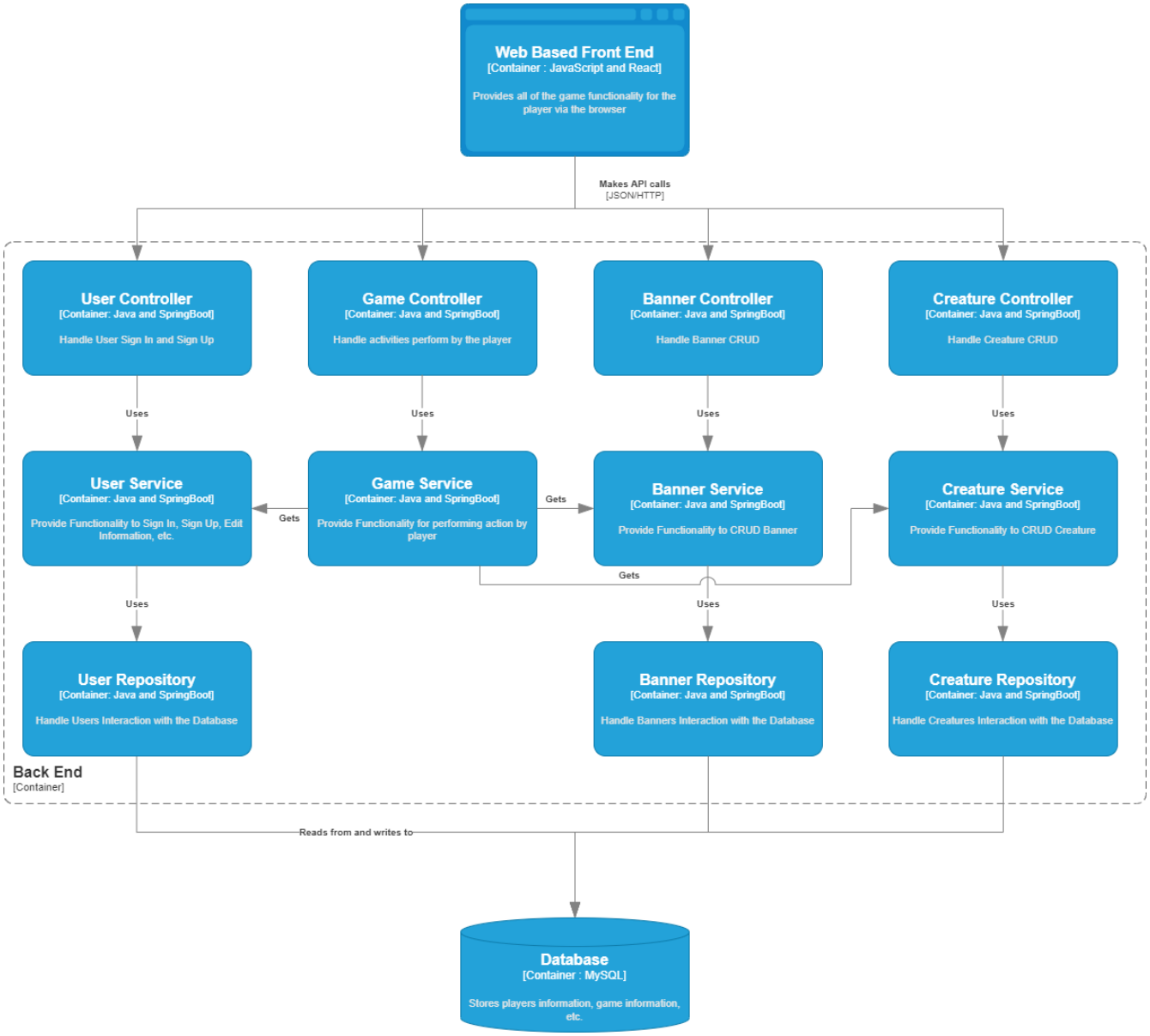
For the back end, Java and Spring Boot will be used. Java is chosen for the main programming language because it is powerful, flexible, and widely used set of tools for web application development. Java is flexible means that it can be used everywhere regardless the operating system and technology such as, computer or smartphone. And Spring Boot can help coding Java much faster with a lot of things already pre-configured from Spring Boot itself. It helps safe time to prepare coding in Java.

For the database MySQL is chosen. MySQL can handle large amount of data and data are displayed in tables and rows which makes it easier and familiar to read.



7.3. Components

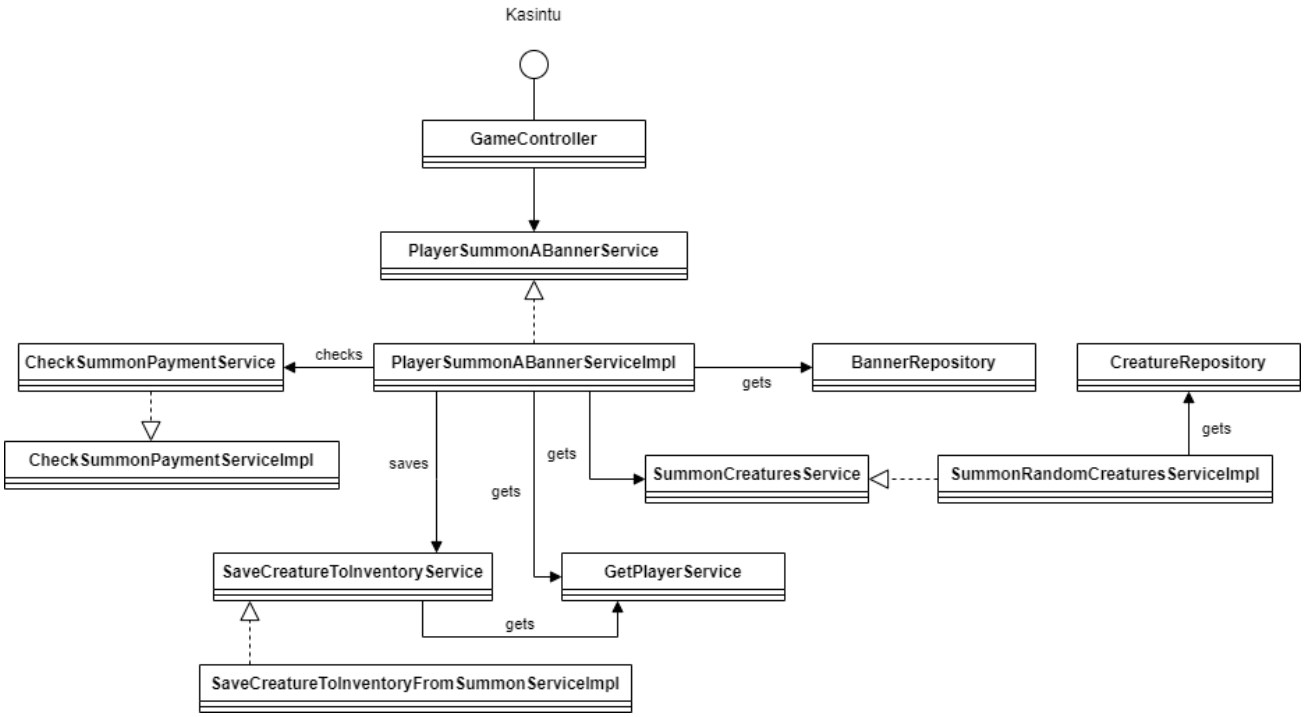
For this diagram, the way that frontend and backend communicate is through the controller class. These controller classes communicate to service classes where all the logics are kept. And through there, it connects to data access layer where it handles the external database and converting to readable code that can be used in the service classes.



7.4. Code

This diagram shows how summoning creatures and saving it to user inventory works. From the game controller class, it called out a method from player summon a banner service that has been implemented by player summon a banner service implementation. In player summon a banner service implementation, it gets the banner from banner repository and from that, it gets the player who wants to summon from get player service. After the banner and player are set up, it checks if it eligible for the player to summon in that banner by checking if the payment is sufficient by using check summon payment service that has been implemented by check summon payment service implementation and after that creatures are summoned from that banner using summon creature service interface that has been implemented by summon random creatures service implementation and it gets creatures from creature repository. After the creatures are obtained, it saves to obtained creatures to player's inventory by using save creature to inventory service interface that has been implemented by save creature

to inventory from summon service implementation. After it saves the creature, player summon a banner service return a list of obtained creatures back to game controller where it will be forwarded to the front end

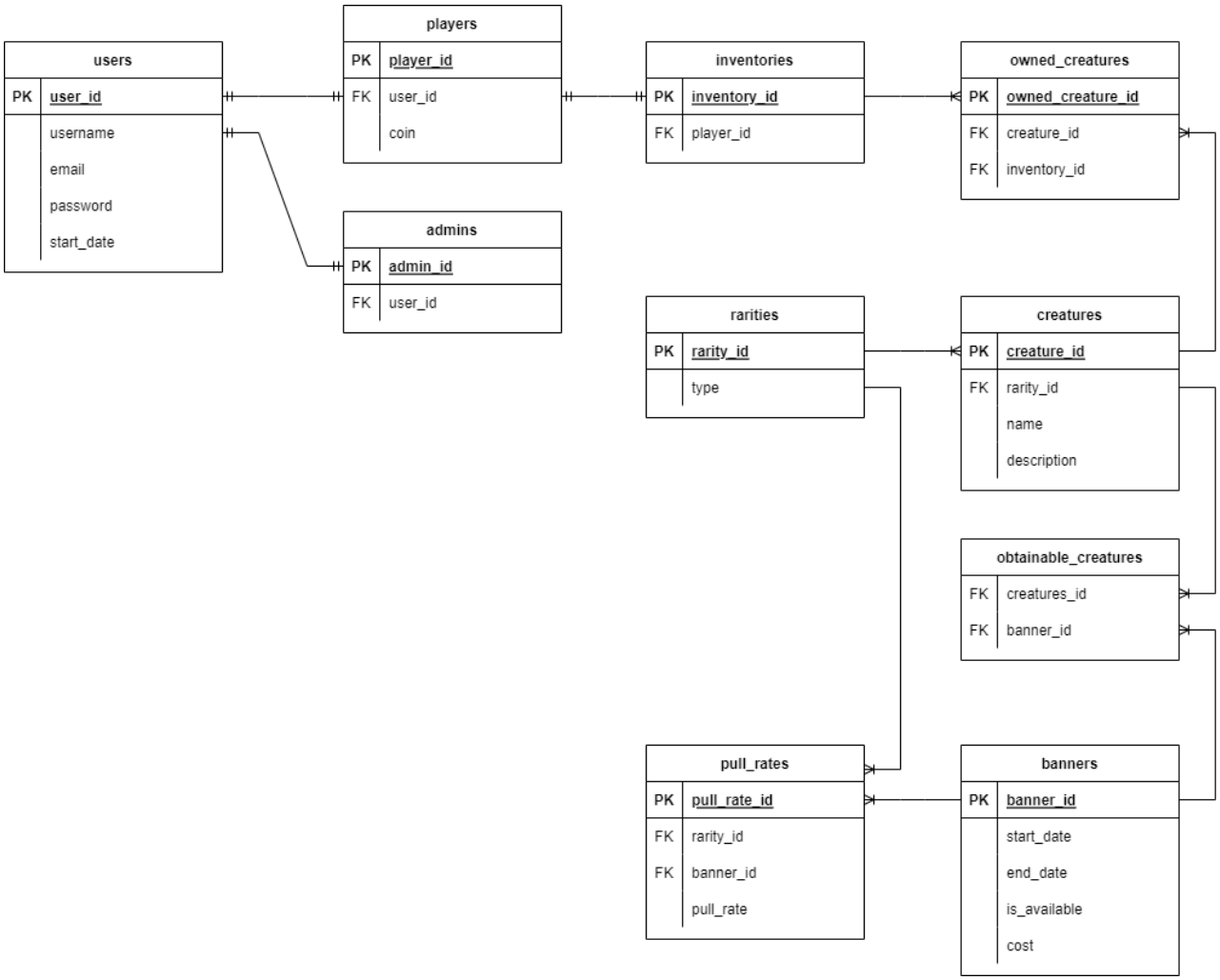


. (Brown, 2018)

8. Design Specification

8.1. Database Design

Image below is the database design in this application.



This is the diagram for the database in this application. It starts with the users table where it holds the common information about the user. And it branches off to 2 different types of users, players, and admins. Players hold an additional information where it stores players' coin data.

Every player has an inventory, but it is separated from the players table to insure privateness for the players. Owned creatures stored the id of what inventory it is located and what creature it is.

A creature has a rarity that are stored in rarities table.

Banners table is where banner information is stored, and the id is used for pulling rates and obtainable creatures' reference. Pull rates table store the rarity and its value of chance of getting it and obtainable creatures table store the id of a banner and id of the creature. It indicates what creatures can be obtained in specific banner.

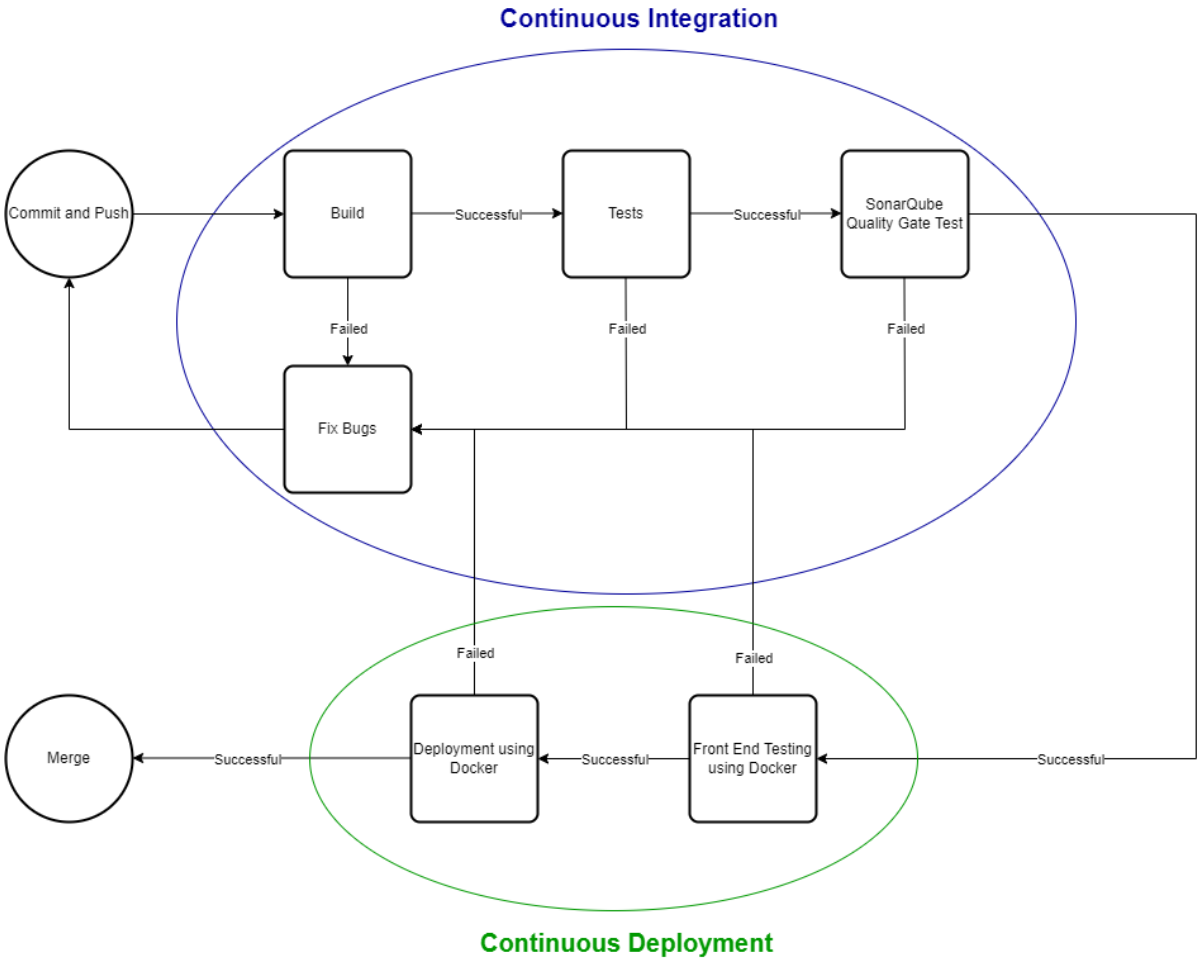
8.2. User Interface Design

Design for the user interface should be simple and recognizable.

8.3. CI/CD Design

Whenever the codes are committed and pushed, it automatically run various commands that tests the application. It has 3 stages, build, test, and SonarQube. If there's a failure in one of the tests, it will be fixed, and re-commit and re-push and the system will run the tests again. If all the tests are successful, application is ready to be merged into another branch.

For the continuous deployment, there are 2 stages and both required Docker. The first one is front end testing or end to end testing. It tested the connection between frontend, backend, and database using fake database as well as test the user interface. If the test has passed, it will continue to deploy using the real database.



9. References

Brown, S. (2018, 06 25). Retrieved from InfoQ: https://www.infoq.com/articles/C4-architecture-model/