*Design document*

Fontys University of Applied Sciences

***Logo

Description automatically generated***

|  |
| --- |
| **Date** : 16/06/2023 |
| **Version** : 6.0 |
| **State** : Final version |
| **Author**  : Anna Kadurina |

Table of Contents

[Overview 2](#_Toc136451887)

[Context 3](#_Toc136451888)

[C4 architecture diagrams 3](#_Toc136451889)

[System context diagram (C1) 3](#_Toc136451890)

[Containers and technology diagram (C2) 4](#_Toc136451891)

[Components diagram (C3) 4](#_Toc136451892)

[Code diagram (C4) 6](#_Toc136451893)

[CI setup diagram 8](#_Toc136451894)

[Important design decisions 9](#_Toc136451895)

[Why am I using SpringBoot? 9](#_Toc136451896)

[Why am I using React? 9](#_Toc136451897)

[What framework am I using and why? 9](#_Toc136451898)

[How do I separate concerns? 9](#_Toc136451899)

[References 10](#_Toc136451900)

# Overview

The GetawayGо project aims to make travelling a better experience for all people by providing travelers and homeowners with a user-friendly platfоrm that is easy to use. The application will have 3 user roles. The first one is a Guest or a Traveler. This user will be able to see all property listings, book a place, manage the booking, and write a review. The second user type is a Host or a Homeowner. They can publish listings, manage their properties, receive reviews, and see statistics. The last user role is Admin or Administration. This person (or people) will be responsible for managing the whole platform – users, properties, and reviews. All users will be able to manage their own infоrmation – such as changing their personal data or their profile picture. Along with that, the guests and the hosts will be able to access a chat to communicate with one another regarding the booking and the arrangements related to it.

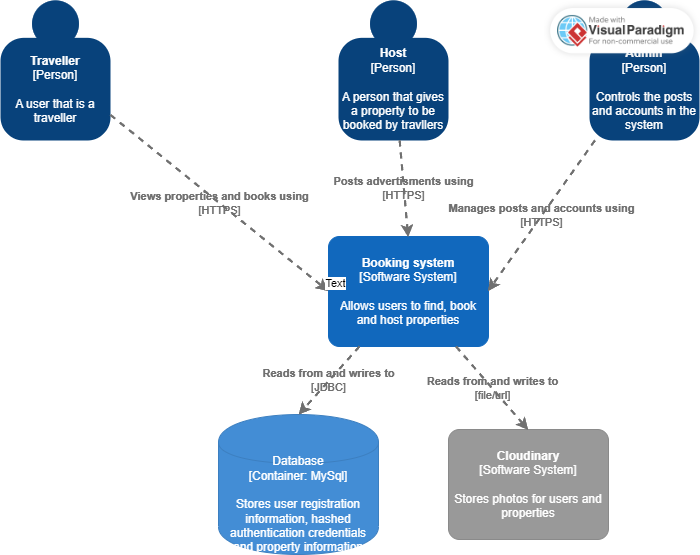
# Context

The GetawayGo application aims to please both travelers and homeowners by making booking accommodation and renting your own property easier tasks. The goal of the project is to create a platform that is user-friendly and efficient.

# C4 architecture diagrams

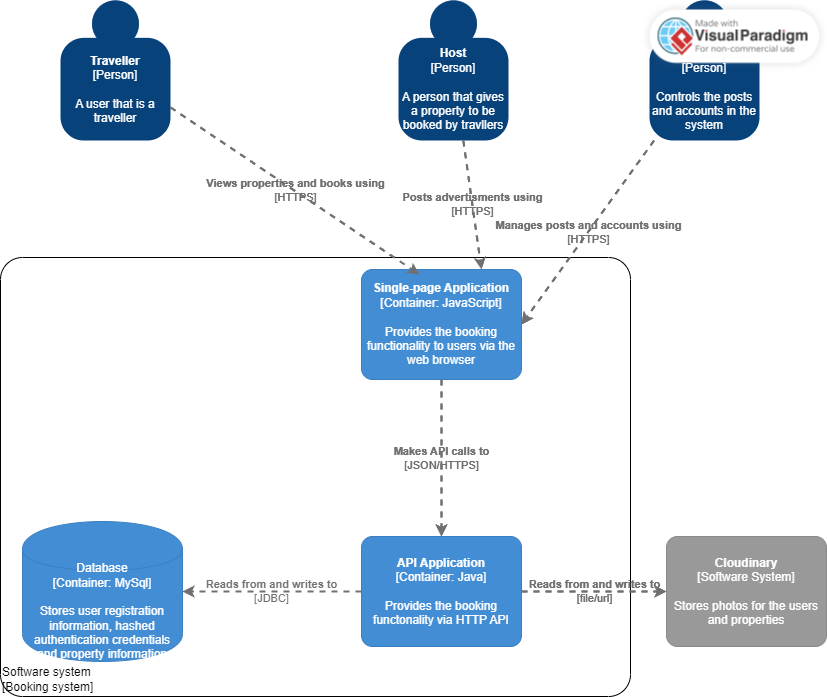
## System context diagram (C1)

The C1 diagram represents the 3 user roles in my application - the Traveler or Guest that can view and book properties, the Host or Homeowner that can rent his properties to travelers, and the administration or the Admin that can manage the posts and the accounts on the platform. They all communicate with the Software system by sending HTTPS requests. The sоftware system is a Booking system that uses a local MySql database and Cloudinary. The database stores all the data and Cloudinary stores all the photos for the application such as profile pictures and photos of the properties.



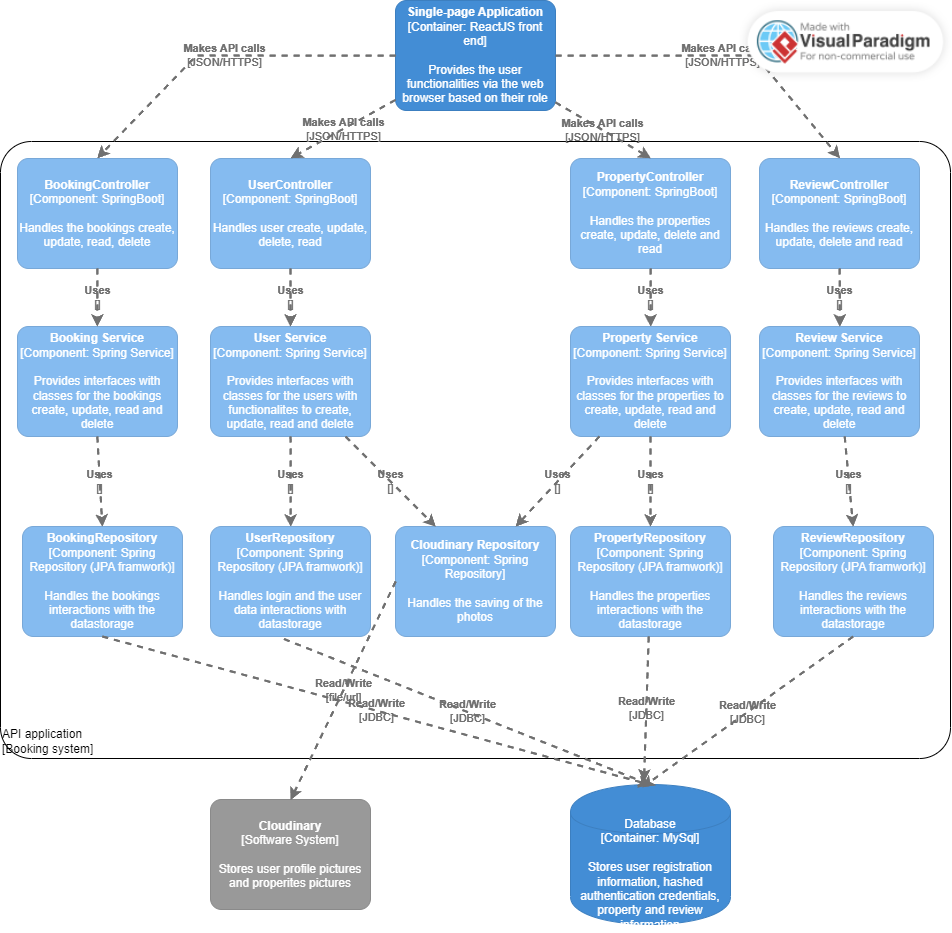
## Containers and technology diagram (C2)

The C2 diagram shows more in depth the technological decisions in my application. The users will communicate with a Single-Page Application that is build in React using JavaScript. This application provides the functiоnalities for the users via the web browser and sends API calls to an API application that is using Java. The API application reads from and writes to a local MySql database and Cloudinary.



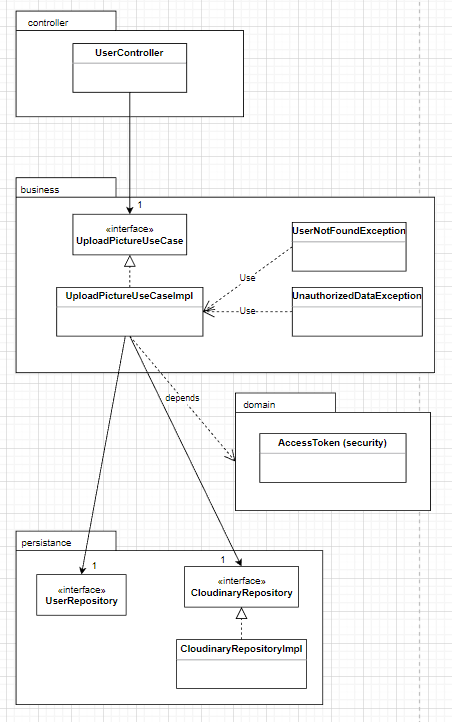
## Components diagram (C3)

The C3 diagram shows the architecture in detail. I have displayed 4 Spring Rest Controllers, each of them connected to a Spring Service with interfaces and classes that are implemented by following the SOLID principles. The Services are cоnnected to Repositories that are using the JPA framework. The Cloudinary Repository handles the saving of the photos in Cloudinary and returns the url of the photo. The Property and User Services receive the url and send it to the User or Property Repository where it is saved in the database. Separating the concerns this way, allows me to have a cleaner and more maintainable code, also to follow the Single Responsibility Principle for each class. Every module of the application (User, Property, Review or Booking) has all three layers – Cоntroller, Business (Service) and Repository. I am using Dependency Injection to inject the necessary components into each other. For example, I will explain the User module. In the User Controller, I inject the User Service and in the User Service, I inject the User Repository. By separating concerns like that, I can be sure that each component has a clear and distinct responsibility.



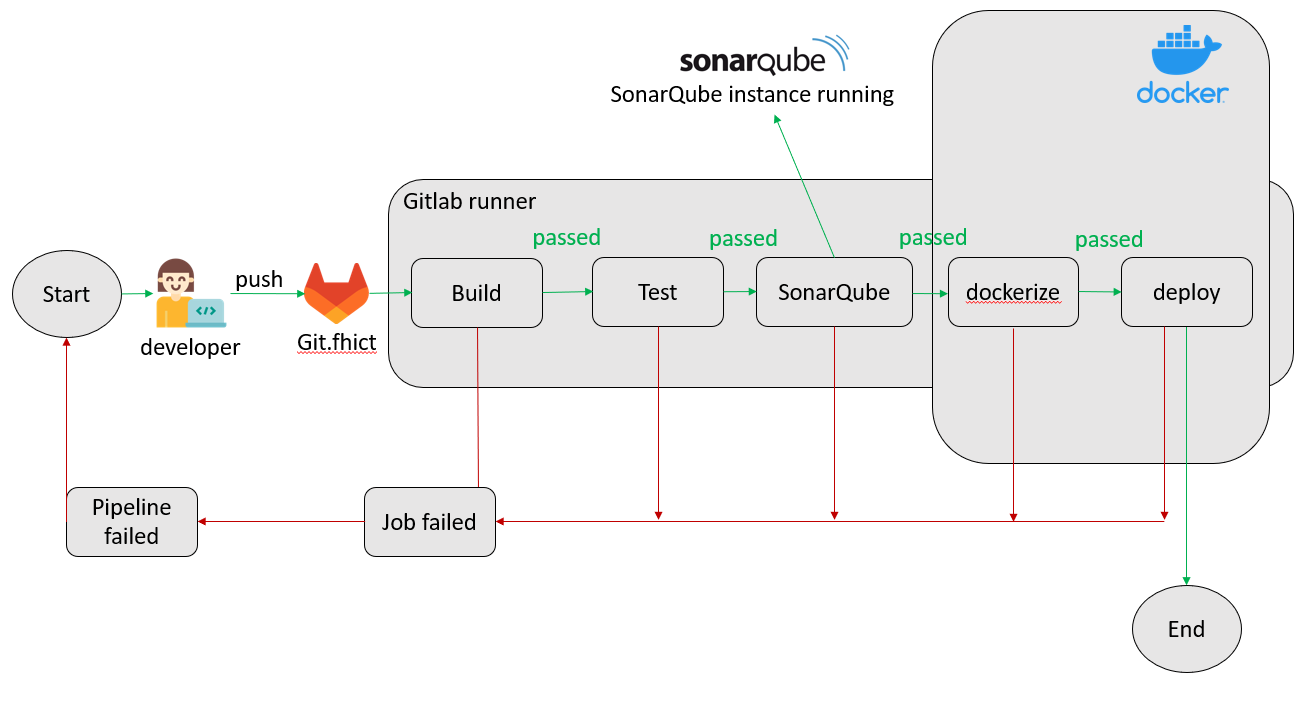
## Code diagram (C4)

The level 4 diagram shows closely what the components are in the code and the relationships between them. In this diagram there are 4 packages – controller, business, domain, and persistence. I decided to show in detail the User Controller. In this diagram, the upload picture functionality is shown. I am using dependency injection between the layers. The User Controller has an instance of the interface Uplоad Picture. By this way, I am depending on abstractions, not concretions. The class that implements the interfaces, consists of the business logic for uploading the picture. The exception that can be thrown is that the user is not found, and the exception is dependent on that class. Another exception that can be thrown is that the data is unauthorized. As shown in the diagram, the Upload picture use case depends on the Access Token. The Access Token is a class in the domain package related with the security of the application. If the user tries tо upload a picture for a different id than his own, then the exception will be thrown, and this will not be allowed. The Upload Picture class has also instances of the 2 interfaces of the repositories. The User Repository is extending the JPA framework. The Cloudinary Repository uploads the picture to Cloudinary and returns the url to be saved in the database. As shown in the diagram, every class has its own responsibility (SOLID). By doing things like that, I can make sure that everything is working properly and also is as simple as possible (KISS). This helps with code quality, maintaining the code and finding the root of errors.



# CI setup diagram

The developer's commit of the local changes to git.fhict marks the beginning of the diagram. The build, test, and SonarQube, dockerize, deploy jobs are taken оver by the Gitlab runner. The SonarQube instance is running and is crucial for the SonarQube task. The dockerize and deploy jobs make the CD pipeline, along with that, they require a presence of a Docker instance. It provides the necessary environment and tools to package the application into a Dоcker container and deploy it to the target environment. If every jоb completes successfully, the pipeline has been successful, and the application is ready for deployment. If a stage fails, the pipeline and the task will both fail, requiring a restart of the process and adjustments from the developer.



# Important design decisions

## Why am I using SpringBoot?

I started using SpringBoot because of the requirements for this semester. The advantages of SpringBoot that I soon discovered are the Embedded Web Servers, the rich IDE support and the fast development.

## Why am I using React?

I started using React because of the requirements for this semester. Things that I like about it are the flexibility and it is not hard to learn how and what to do.

## What framework am I using and why?

I am using the JPA repository framework. It is easy to set up, it generates queries automatically, which is saving a lot of time.

## How do I separate concerns?

I separate concerns by applying layering in my application for each different module as it follows:

The Data Access Layer – Obtains the data from the database.

The Service Layer – I have interfaces with implementations for them to keep the application extendable and maintainable. Doing this, I am following the SOLID principles.

The Controller – At the moment, I have only one controller – User Controller, but in the future, I am planning on having more linked to the specific modules.

I am also using Dependency Injection to make my code more modular, which means that everything depends on abstractions, allowing to easily switch between implementations without affecting the rest of the codebase.

When separating concerns, I try to keep things as simple as possible (KISS principle), which allows me to have a more maintainable, as well as easier to understand code.

# References

1. FreeCodeCamp.org. (n.d.). SOLID principles explained in plain English. Retrieved from https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/
2. SpringHow. (n.d.). Why use Spring Boot? Retrieved from https://springhow.com/why-use-spring-boot/