Architecture

Text

Description automatically generated with medium confidence

24/03/2022 Eindhoven

Version: 0.1

Members:

Lars Kluijtmans: 4220269

Table of Contents

[Version 3](#_Toc105425837)

[Introduction 4](#_Toc105425838)

[How is SOLID guaranteed 5](#_Toc105425839)

[Single responsibility 5](#_Toc105425840)

[Open/closed principle 5](#_Toc105425841)

[Liskov substitution 5](#_Toc105425842)

[Interface segregation 5](#_Toc105425843)

[Dependency inversion 5](#_Toc105425844)

[Important design decisions 6](#_Toc105425845)

[Why user spring boot 6](#_Toc105425846)

[Front end library for building user interfaces 6](#_Toc105425847)

[Database to use 6](#_Toc105425848)

[C4 7](#_Toc105425849)

[C1 7](#_Toc105425850)

[C2 8](#_Toc105425851)

[C3 9](#_Toc105425852)

[C4 10](#_Toc105425853)

[CI/CD setup 11](#_Toc105425854)

[Security report 12](#_Toc105425855)

[1 Broken access control 12](#_Toc105425856)

[2 Cryptographic failure 12](#_Toc105425857)

[3 Injection 12](#_Toc105425858)

[4 Insecure Design 12](#_Toc105425859)

[5 Security Misconfiguration 12](#_Toc105425860)

[6 Vulnerable and outdated components 13](#_Toc105425861)

[7 Identification and authorization failures 13](#_Toc105425862)

[8 Software and data integrity failure 13](#_Toc105425863)

[9 Security logging and monitoring failures 13](#_Toc105425864)

[10 Server side request forgery 13](#_Toc105425865)

# Version

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Autor | Changes | Status |
| 0.1 | 08/03/2022 | Lars Kluijtmans | First edition. | Not finished |
| 0.2 | 02/04/2022 | Lars Kluijtmans | Adding versioning, introduction and index. | Not finished |
| 0.3 | 06/04/2022 | Lars Kluijtmans | Remaking the research questions. | Not finished |
| 0.4 | 10/04/2022 | Lars Kluijtmans | Decided to use mysql instead of mogodb. | Version 0 is finished |
| 1.0 | 13/05/2022 | Lars Kluijtmans | Updated the c4 diagram, added the CI/CD diagram, moved research question to other document. | Version 1 is started |
| 1.1 | 25/05/2022 | Lars Kluijtmans | Update c4 diagram, add captions to images. | Version 1 is up to data |
| 1.2 | 06/06/2022 | Lars Kluijtmans | Add security report. | Version 1 is up to data |

# Introduction

Game market is an online market place that allows users to sell and buy used games by connecting buyers and sellers or through an auction. For sellers we also included features such as statistics to see how their games are selling, easy management of products they are selling… For buyers we made the search features as easy to use as possible and made the contacting of sellers very simple.

# How is SOLID guaranteed

## Single responsibility

Every class is responsible for only one type of object, in this application the only typed of objects are users and products, although they are connected there are different classes in charge of managing them.

## Open/closed principle

By using interfaces, I can, if needed extend from them to another class without having to modify any of the already existing code.

## Liskov substitution

In this application only the user object implements this principle, it has 2 subclasses, NormalUser and Admin.

## Interface segregation

At the moment this principle is not used in the application.

## Dependency inversion

In between every layer of this application (controller, service and repository) there is a interface separating them.

# Important design decisions

## Why user spring boot

Spring Boot helps developers to start coding right away without wasting time on preparing and configuring the environment. In contrast to other Java frameworks. I have no alternatives.

## Front end library for building user interfaces

Most of the class wanted to use react so I just went with it. Still I believe that Angular would have been the better option, just because the client of our group project said he would prefer for us to use it and we have to use the same front end for both the individual and group projects.

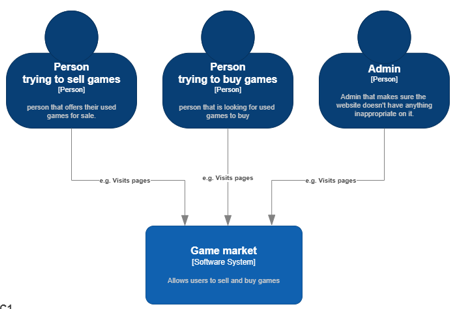
## Database to use

I will be using a MySQL database for this project because I’m already familiar with it and the teacher (Marcio) explained how to connect the API to a local MySQL database (which I couldn’t get working for Mongodb).

# C4

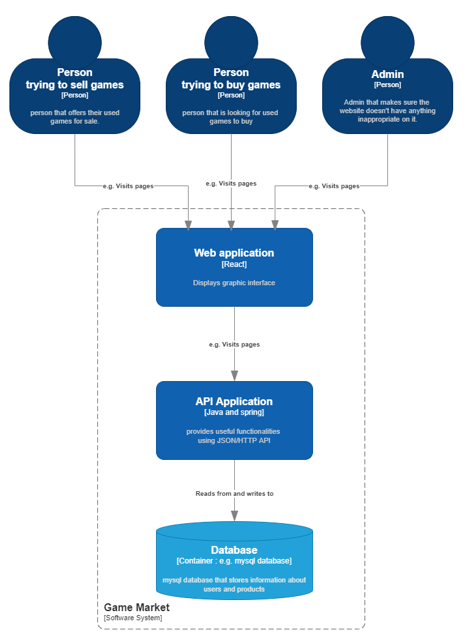
## C1

This system has only 2 types of users but one of these, the NormalUser, can preform 2 very different actions, which is why they are shown in the images bellow as different people. One using the system to sell old and or used games and the other to buy these same games. There is also a admin that can remove users and products if he finds the inappropriate.



1 C1

## C2

This system is divided into 2 separate containers the Web applications and API. The Web application is the front end made using react that the users interact with and the API is responsible for sending and receiving data from and to the Web application.

2C2

## C3

The API as display here is divided into 2 sections (Accounts and Applications) and 3 layers(Controller, service, repository).

Diagram

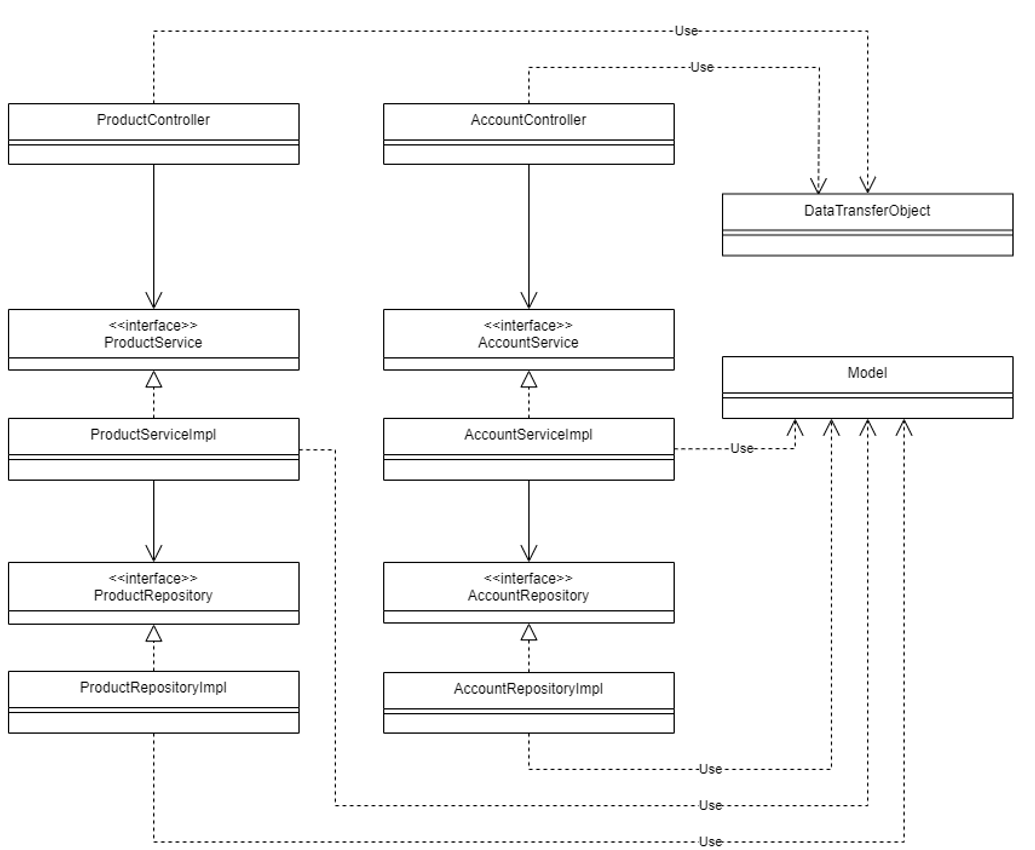
Description automatically generated

3C3

## C4

A simple UML, the controller layer uses a DTO (DataTransferObject) in contract to the Service and Repository layers that uses a Model. I think that this naming makes more sense then the other way around because the DTO is the object that will be transferred to the other containers that use the API.

Not been updated since sprint 1



4C4

# CI/CD setup

To ensure the code quality. Before a push is accepted by git it will run it through a CI/CD pipeline. This pipeline checks that the application can be build without errors, that all the unit tests are successful and finally runs a code quality check using SonarQube that verifies that at least 80% of the code is tested and that looks for any security issue. If at any of these stages the check fails push will be unsuccessful.

Explain more about git and why this is happening. Where it’s running match the proses.

Why things might fail



5CI/CD diagram

# Security report

How does this application deal with the OWASP top 10 security risks(2021):

## 1 Broken access control

To prevent this all recourses ,except some public once, are not accessible by default. Cross-origin resource sharing is used so only requests from some websites are permitted. Ownership of records is enforced so only users that own a record can update and delete them.

## 2 Cryptographic failure

All data except emails, phone number and password do not fall under the Privacy law. Sadly of there three that do only one (Password) is saved in the database as encrypted. No sensitive data is stored unnecessarily, the only sensitive data that is saved are emails and phone numbers for the purpose of contacting users when needed and passwords for security reasons.

All keys and algorithms are up to the standard. Security controls are in place for data as per it’s classification so for example: only admins and the users them selves can see their information.

There is a vulnerability in the data transfer as data in transit is not encrypted.

## 3 Injection

Great venerability for this application because all connections with the database go through JPA and JPA does not prevent injections there is also no limits tot eh amount to data requested in one call to the database because I don’t know how to use limit it in combination with JPA.

## 4 Insecure Design

The design is secure because there are unit tests to validate to threats(at least to a extent) and layers of the system are segregated based on the protection needed.

No use-cases and or misuse-case have been made.

## 5 Security Misconfiguration

Cross-site request forgery(CSRF) is disabled.

Error messages are not overly informative. There are no unnecessary features, components, documentations and samples.

Application architecture provides effective and secure separation between components with segmentation/containerization. (at least I would have made it that way of I had more time. Did it for the group project, know how to do it.)

## 6 Vulnerable and outdated components

There are no unused dependencies. All components are new, from trusted sources and being updated regularly.

## 7 Identification and authorization failures

Passwords only have a min length there is no checking for their ’strength’. There is no credential recovery or forgot-password process. So malicious users can’t take advantage of it? Passwords are stored encrypted and the encryption can be considered ‘strong’.

## 8 Software and data integrity failure

To ensure that integration is always successful we use a CI/CD pipeline that has proper segregation, configuration.

## 9 Security logging and monitoring failures

Sadly there is no monitoring of: Logins, failed logins and high value transactions, Logs are not supervised for suspicious activities.

There is nothing to trigger alerts if there is a case of Penetration testing or dynamic application security testing (DAST).

## 10 Server side request forgery

There is nothing in place to prevent this.