Stour

STUDENT NAME: Radu-Todor Alexandru

GROUP: 30433

# CONTENTS

[1. Introduction 3](#_Toc97840780)

[2. Tech Stack 3](#_Toc97840781)

[3. Software Architecture 3](#_Toc97840781)

[4. Requirements 3](#_Toc97840781)

[5. Diagrams 4](#_Toc97840781)

# Introduction

The aim of this project is to develop a web application, making the database, the backend, and the frontend from scratch. I decided to make a video game online store, like Steam, where users can buy games, "play" them, and make friends. There is also a special type of "user", the publisher, which can make games.

# Tech Stack

The database is written in MySQL and is developed using MySQL Workbench 8.0 CE. The project includes a file “database.sql” which contains a basic database for testing purposes.

This backend is written in Java Spring and is developed using JetBrains IntelliJ IDEA. The backend is designed using layered architecture, and the layers are explained in the Package Diagram part of the documentation.

# Software Architecture

The database is build following the layered architecture pattern. In this architecture, we have the packages on the right with the following meanings:

* Repository: Used to make the connection between the database and the backend,
* Entity: Used to transfer lines from tables from the database into Objects,
* Service: Used to compute the logic of the backend,
* Controller: Used to make the connection between the database and the frontend,
* DTO: Used to make the Objects that are sent to the frontend.

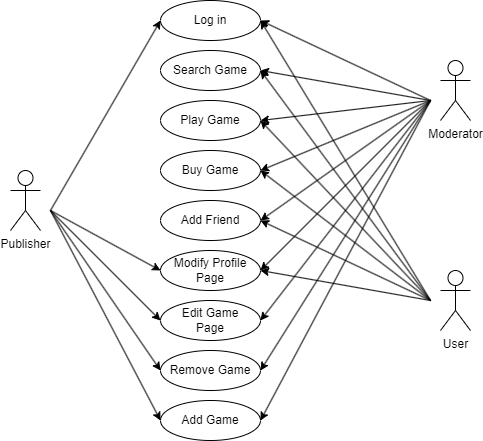
Another important thing to say about this architecture is how the packages Controller, Service and Repository are layered: Controller can only use Service and Service can only use Repository, any other combination between them is forbidden.

# Requirements

* **Functional Requirements**
* The CRUD operations
* Letting the user log in
* Allowing the user to “play a game”
* Adding friends
* Different interfaces for the normal user and the publisher
* **Non-functional Requirements**
* Speed
* Scalability: the system must perform well even under a big workload
* Usability: the application needs to be easy to use

# Diagrams

1. **Use Case Diagram**



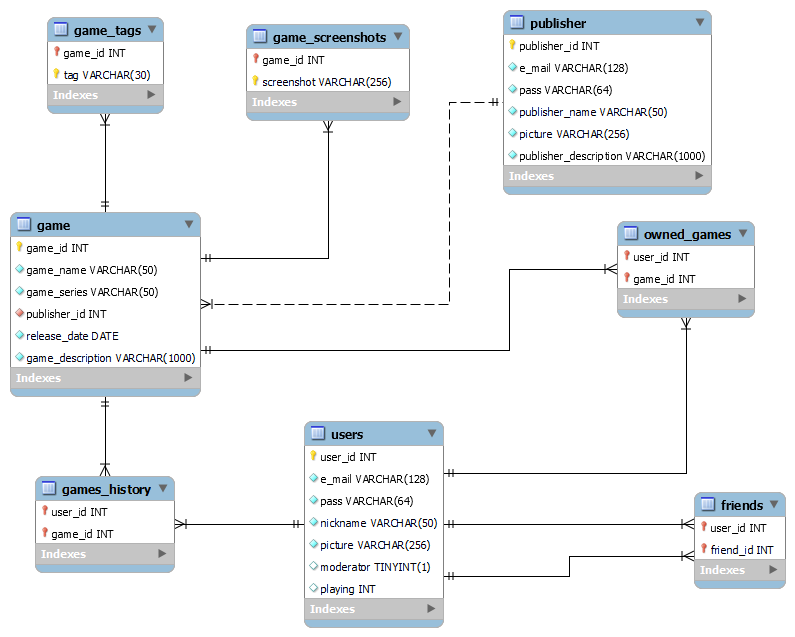
The Use Case diagram presents the actions that can be performed by each type of user.

We have three types of users: a normal user, a moderator and a publisher. The normal user, after logging in, can search for the game, but it, play it, add other users as friends and modify his profile page.

The moderator is a special type of user. It can do the same things a normal user can, and along with that, he can also modify (or even delete) any game that doesn’t follow the platform guidelines.

The third type of user is the publisher, which can add a game on the platform and modify details about it.

1. **Database Diagram**



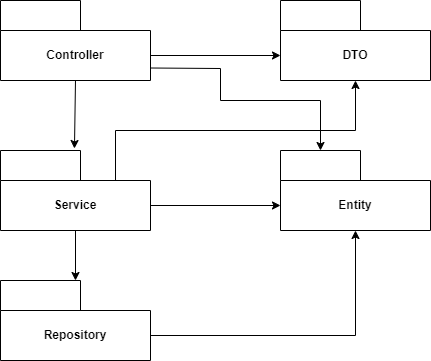
The database has a total of 8 tables. The main tables are “users”, “publisher” and “game”.

The tables “game\_tags” and “game\_screenshots” are in a one-to-many relationship with game and store information about the game.

The tables “games\_history” and “owned\_games” are used to save the user’s game library as well as the most recent games he/she played. It can be considered that the tables “users” and “game” are in a many-to-many relationship through these tables.

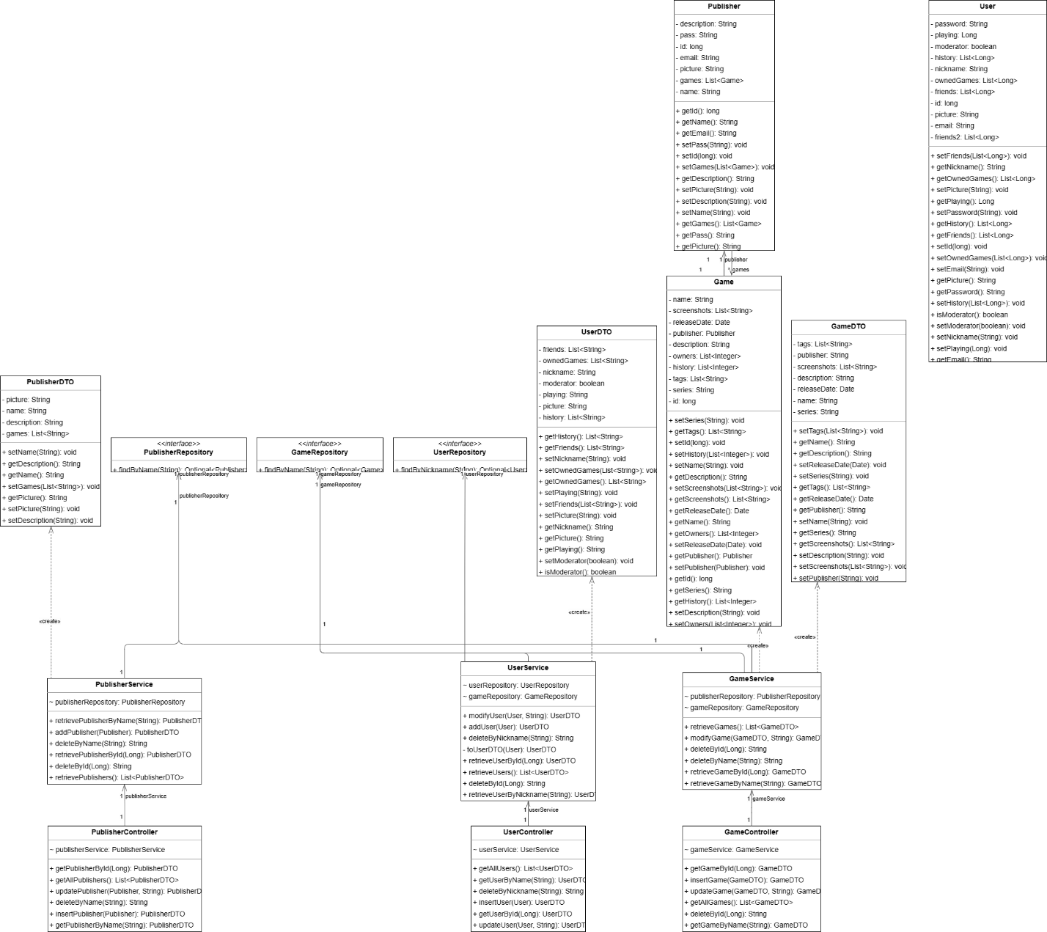
The table “friends” creates a self-reference for the table “users”, where a user can have many users as friends and a user can be the friend of many users.

1. **Package Diagram**



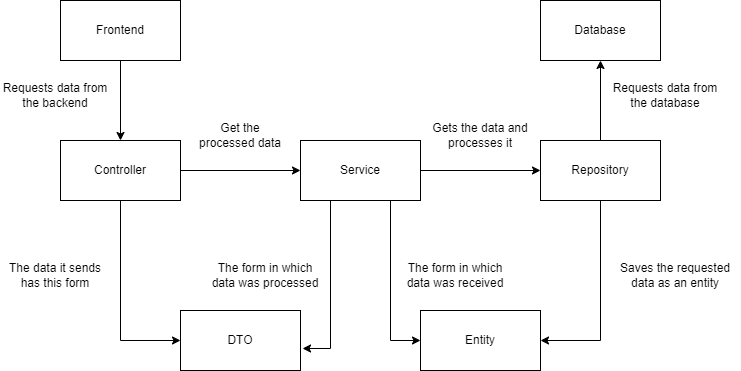
The database is built following the layered architecture pattern.

1. **Class Diagram**



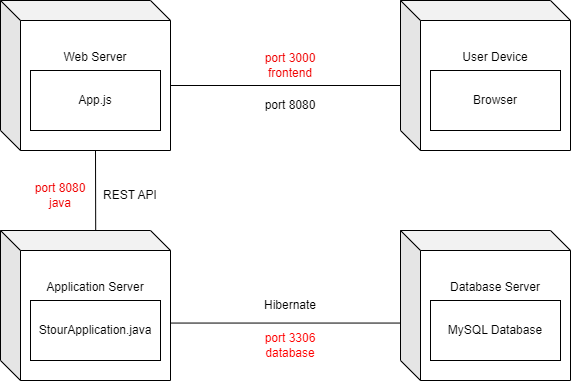
The classes follow the pattern described in the Package Diagram section, and each entity has a class in each package. We have three main entities, User, Publisher and Game. Each of them is an Entity and has a Repository, a Service, a Controller, and a DTO.

1. **Component Diagram**

****

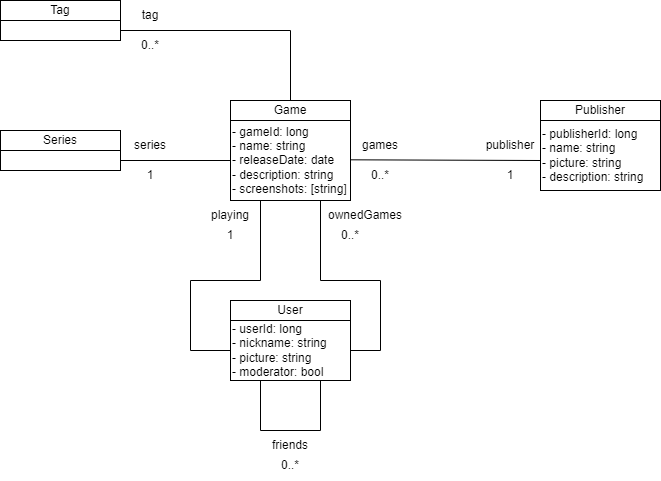
The component diagram is similar to the package diagram, and it shows how the major components are connected between them.

1. **Deployment Diagram**

****

This is the deployment diagram, and it shows how parts of the application communicate with each other. The user connects from a device using a browser through localhost on port 3000 to the application. In the background, the frontend communicates with the backend through port 8080 and REST API calls. In order to get data from the database, the backend uses Hibernate through port 3306 to fetch the data.

1. **Domain Model**

****

The domain model is a visual representation or diagram that captures the essential elements from the problem domain. It refers to a specific area or subject matter that the web application is addressing (in this case, a game store). It focuses on the entities and their relationships, helping with the understanding of the structure and behavior of the system being built.