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.

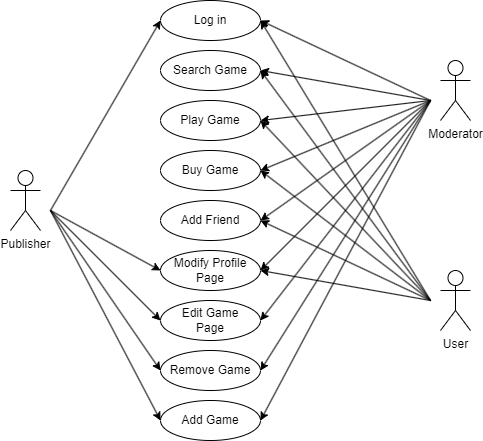
Another important thing to say about this architecture is how the packages Controller, Service and Repository are layered: Controller can only use Service and Service can only use Repository, any other combination between them is forbidden.

# 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**



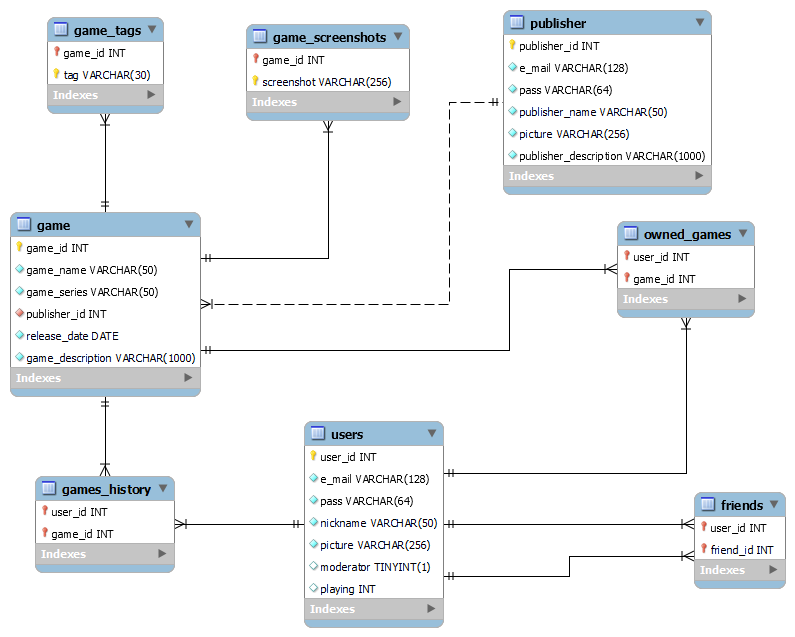
The Use Case diagram presents the actions that can be performed by each type of user.

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**



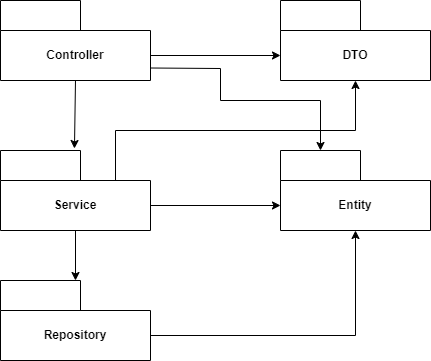
The database has a total of 8 tables. The main tables are “users”, “publisher” and “game”.

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.

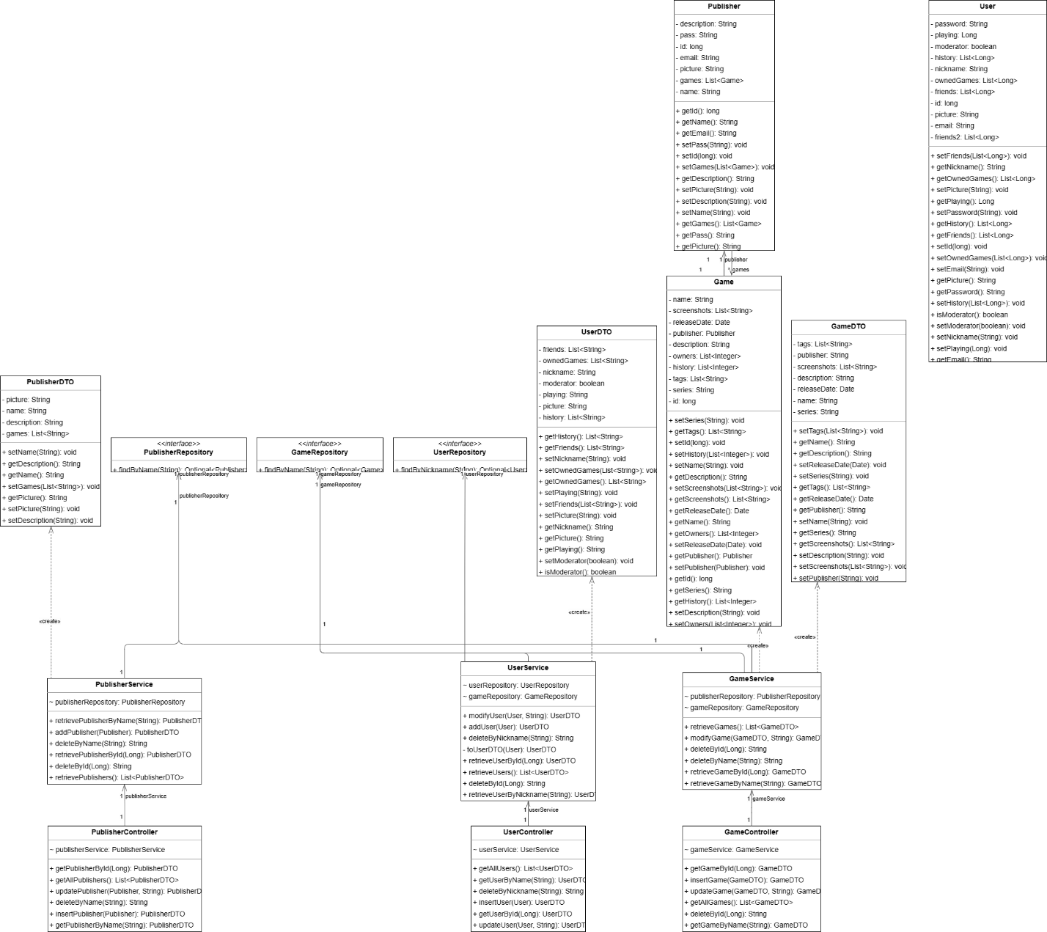
The table “friends” creates a self-reference for the table “users”, where a user can have many users as friends and a user can be the friend of many users.

1. **Package Diagram**



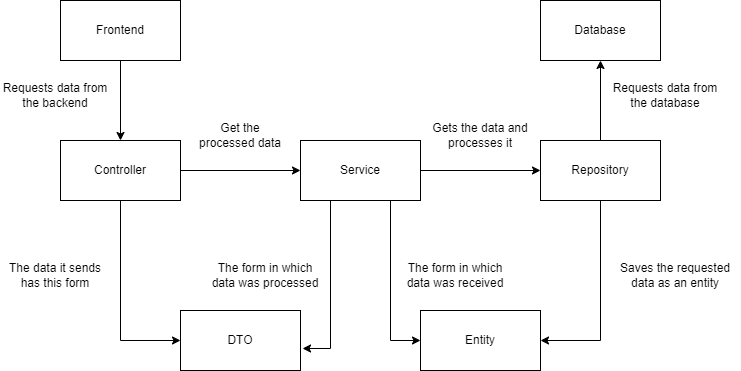
The database is built following the layered architecture pattern.

1. **Class Diagram**



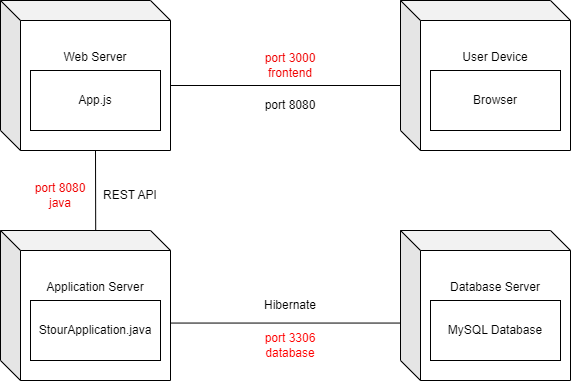
The classes follow the pattern described in the Package Diagram section, and each entity has a class in each package. We have three main entities, User, Publisher and Game. Each of them is an Entity and has a Repository, a Service, a Controller, and a DTO.

1. **Component Diagram**

****

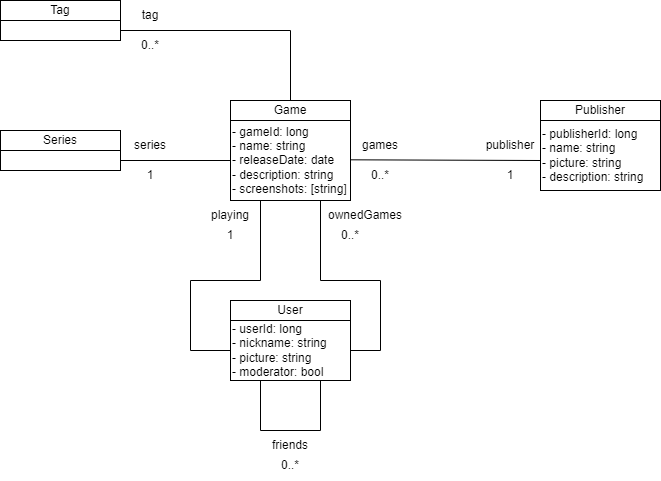
The component diagram is similar to the package diagram, and it shows how the major components are connected between them.

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.

1. **Design Model**

Design modeling in software engineering represents the features of the software that helps engineers to develop it effectively, the architecture, the user interface, and the component level detail.

* Data Design:

It represents data objects and their interrelationship. In our project, we have the entities Game, Publisher, and User. Along with fields like “name”, “email” and others, we have fields which consolidate interrelationships. We have

* One-To-Many between Game and Publisher (a game can have only one publisher and a publisher can publish many games),
* Many-To-Many between Game and User (a user can have many games and a game can be owned by many users),
* One-To-Many between User and Game (a user can play only one game at a time and a game can be played by many users at the same time)
* Many-To-Many between User (a user can have many other users as friends).
* Architectural Design:

The project is designed based on layered architecture. We have five layers which interact between themselves in a well-defined manner. The Repository layer handles the communication between the database and the backend, the Service layer handles the logic of the project, the Controller layer handles the communication between the frontend and the backend, the Entity layer handles the storing of the database objects and the DTO layer handles the conversion into entities which will be sent to the frontend.

* User Interfaces Design:

The user interface design is focused on developing an intuitive and efficient user experience. For example, users should have access to login screens, store pages, library, friend list and publishers should have access to interfaces through which they can update their game pages. To ensure a seamless and user-friendly interface, the design should consider the arrangement of elements, visual components, menu navigation, and interactive features.

* Component Level Design:

Each component of the project is organized in one of the previously mentioned layers. For example, the class that handles sending a Game entity to the frontend is GameController from the package Controller. Inside Controller we have GameController, UserController, and PublisherController, in Service we have GameService, UserService and PublisherService, and so on. Each entity is combined with a layer to perform a desired operation.

1. **Data Model**

Data Model is the modeling of the data description, data semantics, and consistency constraints of the data. It provides conceptual tools for describing the design of a database at each level of data abstraction. The types of data models are Relational Data Model, Entity-Relationship Data Model, Object-Based Data Model, and Semistructured Data Model.

In this project, the data model used is the Relational Data Model. Data is stored inside tables in the form of rows and columns, and both the entities and the relationships are represented using tables. The entities (Game, Publisher, User) are represented using these tables and the relationships between them are established using foreign keys. The database diagram can be seen on page 4.

We have the following tables:

* game: used to store information about the game, such as the name, the series, the publisher (through a foreign key), the release date, and the description.
* publisher: stores information such as the name, email and password of the account, a picture, and a description.
* user: stores the account credentials (email and password), nickname, picture, a boolean representing the special role “moderator”, and a field containing the id of the game the user is currently playing.
* game\_tags: each game has a list of tags through which it can be searched. This table stores the list of tags and has the fields game\_id (which is a foreign key to the game table), and tag, which is one of the tags. There can be many entries with the same game\_id but different tags.
* game\_screenshots: just like with tags, each game page has a series of screenshots from the game which are used to promote it, and this table stores them. We have to fields, game\_id and screenshot, and the table works in the same way as game\_tags.
* owned\_games: this table establishes a Many-To-Many relationship between user and game through the fields user\_id and game\_id, both foreign keys. The table represents the list of games owned by a user, or which users own a game.
* friends: this table establishes a Many-To-Many circular relationship with user. It has two fields, user\_id and friend\_id, both foreign keys to the table user. It’s used to store the list of users which are friends with a user.