**Software Engineering**

**Software Requirements Specification**

**(SRS) Document**

**Game & Go**

**September 17th, 2022**

**v1.0**

**By: Justin Kim**

**Theodore Okamura**

**Nicholas Ray**

**<Honor Code>**

|  |
| --- |
| **Table of Contents** |

[1. Introduction 2](#_Toc113453891)

[2. General Description 2](#_Toc113453892)

[3. Functional Requirements 3](#_Toc113453893)

[4. Technical Requirements 3](#_Toc113453894)

[4.1 Operating System & Compatibility 3](#_Toc113453895)

[4.2 Interface requirements 3](#_Toc113453896)

[4.2.1 User Interfaces 3](#_Toc113453897)

[4.2.2 Hardware Interfaces 4](#_Toc113453898)

[4.2.3 Communications Interfaces 4](#_Toc113453899)

[4.2.4 Software Interfaces 4](#_Toc113453900)

[5. Non-Functional Requirements 4](#_Toc113453901)

6. Use Case Model 5

6.1 Use Case Model Description 6

1. Introduction
   1. **Purpose:** Our goal is to have a one stop shop for searching for new games, reading descriptions, and downloading games.
   2. **Document conventions:** The purpose of this Software Requirements Document is to give a comprehensive overlook at our project and the goals we wish to accomplish. We want our users to be greeted with a simple, yet effective front page that will be able to find what they are looking for. We have all be in a situation where we are presented with so many choices it becomes overwhelming, that is where we step in to alleviate that problem. When you first log-in you will be allowed to purchase and return game, they will also have access to their list of games.
   3. **Definitions, Acronyms, and Abbreviations**

|  |  |
| --- | --- |
| **Term** | **Definition. Acronym, Abbreviation** |
| MongoDB | NoSQL database program |
| Spring boot | Java framework to create java-based applications |
| DB | An abbreviation for Database. |

* 1. **Intended audience:** This project is the brainchild of our very own, Justin, Theo and Nick. This Software Requirements Document will be useful for future users and used to show possible investors what we bring to the table and what we plan to deliver.
  2. **Project Scope:** Our overall goal is to allow people who are interested in new games a very simple shopping experience. Through this our business would be an online distribution of games from large companies possibly including, but not limited to: Activision, Blizzard, Bungie, and 343.
  3. **Technology Challenges:** Throughout development we have encountered little to no technological difficulties.
  4. **References:**  As of this moment we have no references since we are in infancy of our project.

## General Description

* 1. **Product perspective:** This project came to us when we all were able to relate and share less than ideal experiences with window shopping for games. The problem of having to go to multiple websites to find different pieces of information about a single triple A title can be downright frustrating, and that’s where we want to step in and be the solution to that problem.
  2. **Product features:** This program will be able list games sourced from an API provided by steam. Then the users will be able to purchase any game that they like, and it will be added to their game list which they can access at any time. From that list or the browse section any owned game will be able to be returned.
  3. **User class and characteristics:** Our website application relies on the assumption that the user has a vague idea of what they are looking for.
  4. **Operating environment:**

Server: Java 17

Client: Web browser

* 1. **Constraints:** We are required to use Java and the Spring Boot framework.
  2. **Assumptions and dependencies:** Our external dependencies for the backend are Steam’s API for the pictures, game list and descriptions. MongoDB is our database, specifically their free cloud service Atlas. We were required to use spring boot as our MVC. jwt.io allowed us to use JSON web tokens to authenticate users. Jakarta XML Binding API helped fix an issue with jwt.io not being supported in the latest version of Java. Lombok allowed us to make the class files simpler. For the frontend, we used react for our UI. axios was used to make the HTTP requests to our backend server. We used bootstrap to style the page. http proxy middleware was used to prevent cross origin request errors.

1. Functional Requirements
   1. **Primary**

The motivation behind our software is to connect consumers with the list of games available and allow them to mark them for purchase. The primary functional requirements are those which directly support this goal.

* Fp0: The webapp will allow the user to browse a catalog of games by their picture.
* Fp1: Upon clicking the picture of one of the games, the app will display the price and description of the requested game
* Fp2: The user will be able to register for an account on the sign-up screen with a valid email address, a username of more than 3 and less than 20 characters and a password of length between 6 and 40 characters (exclusive).
* Fp3: With valid account credentials, the user should be able to log in to the web app and have their session saved in a cookie.
* Fp4: Once logged in and navigated to the description page of a game, the user should be able to purchase or return a game.
* Fp5: When a registered user is on the home page, they have access to a “My Games” button that directly links them to a list of all the games they have purchased.
  1. **Secondary:**

The secondary functions of the web application are solely for streamlining the service of a web app which are pertaining to an admin, their account, and the powers they possess over your average user account.

* Fs0: An authorized Admin account should be able to access a “Admin Board” which shows them all the registered users and the option to delete any user.
* Fs1: Once logged in with admin credentials and navigated to the description page of a game, they have access to edit the price of the game to a new value.

## Technical Requirements

* 1. Operating System & Compatibility

Our web app is available on all web browsers and all operating systems.

* 1. Interface requirements
     1. User Interfaces

Not applicable at this time.

* + 1. Hardware Interfaces

Not applicable at this time.

* + 1. Communications Interfaces

Not applicable at this time.

* + 1. Software Interfaces

Not applicable at this time.

## Non-Functional Requirements

Constraints on the services or functions offered by the system (e.g., timing constraints, constraints on the development process, standards, etc.). Often apply to the system as a whole rather than individual features or services.

* 1. **Performance requirements**

Not applicable at this time.

* 1. **Safety requirements**

Not applicable at this time.

* 1. **Security requirements**

Not applicable at this time

* 1. **Software quality attributes**
     1. Availability

Software must be acceptable to the type of users for which it is

designed.

* + 1. Correctness

Software should run and execute the correct commands at any given time.

* + 1. Maintainability

Software should be written in such a way so that it can evolve to

meet the changing needs of customers.

* + 1. Portability

A set of attributes that bear on the ability of software to be transferred from one environment to another.

Detailing on the additional qualities that need to be incorporated within the software like maintainability, adaptability, flexibility, usability, reliability, portability etc.

* 1. **Process Requirements**
     1. Development Process Used

Plan Driven Process-Evolutionary Prototyping

* + 1. Time Constraints

This semester.

* + 1. Delivery Date

Sept. 20- SR Document Due

Oct. 4 Design Document Due

* 1. **Other requirements**
* All work subject to UNCG Academic Integrity Policy.

All SRS/SRD should be:

* **Correct:** A method of analysis that ensures that the software meets the requirements identified.
* **Unambiguous:** There is only one interpretation of what the software will be used for and it is communicated in a common language.
* **Complete:** There is a representation for all requirements for functionality, performance, design constraints, attributes, or external interfaces.
* **Consistent:** Must be in agreement with other documentation, including a systems requirements specification and other documents.
* **Ranked for Importance and/or Stability:** Since all requirements are not of equal weight, you should employ a method to appropriately rank requirements.
* **Verifiable:** Use measurable elements and defined terminology to avoid ambiguity.
* **Modifiable:** A well-defined organizational structure of the SRS document that avoids redundancies can allow easy adaptation.
* **Traceable:** Ability to trace back to the origin of development and move forward to the documents produced from the SRS.
* **Legible and Professionally Presented**: Must use a consistent font and style. Must have proper formatting of tables and charts. Must be grammatically correct. Use active tense and concise sentences.

## Use Case Model

Diagram

Description automatically generated

**Descriptions:**

**Actor 1: Not logged in user (all actors are able to, since this requires no authorization)**

* Register for an Account: A non-user will be prompted to enter a Username, Email, and a password. A username of more than 3 and less than 20 characters and a password of length between 6 and 40 characters are required.
* Browse Game Information: Any one can access our catalog and scroll page to page seeing each individual game and reading their description.

**Actor 2: Logged in user (available to the logged in user & admins)**

* Purchase/Return Game: A registered user can access any game, and while on that game specific page there is a purchase button that will immediately add it to that user's “My Game” list.
* View Purchased Games: The feature of “My Games” is exclusive to registered users and is a concise list of all the games that user currently owns. From this page they can go to each game's page with their description or return the game from the list.

**Actor 3: Admin**

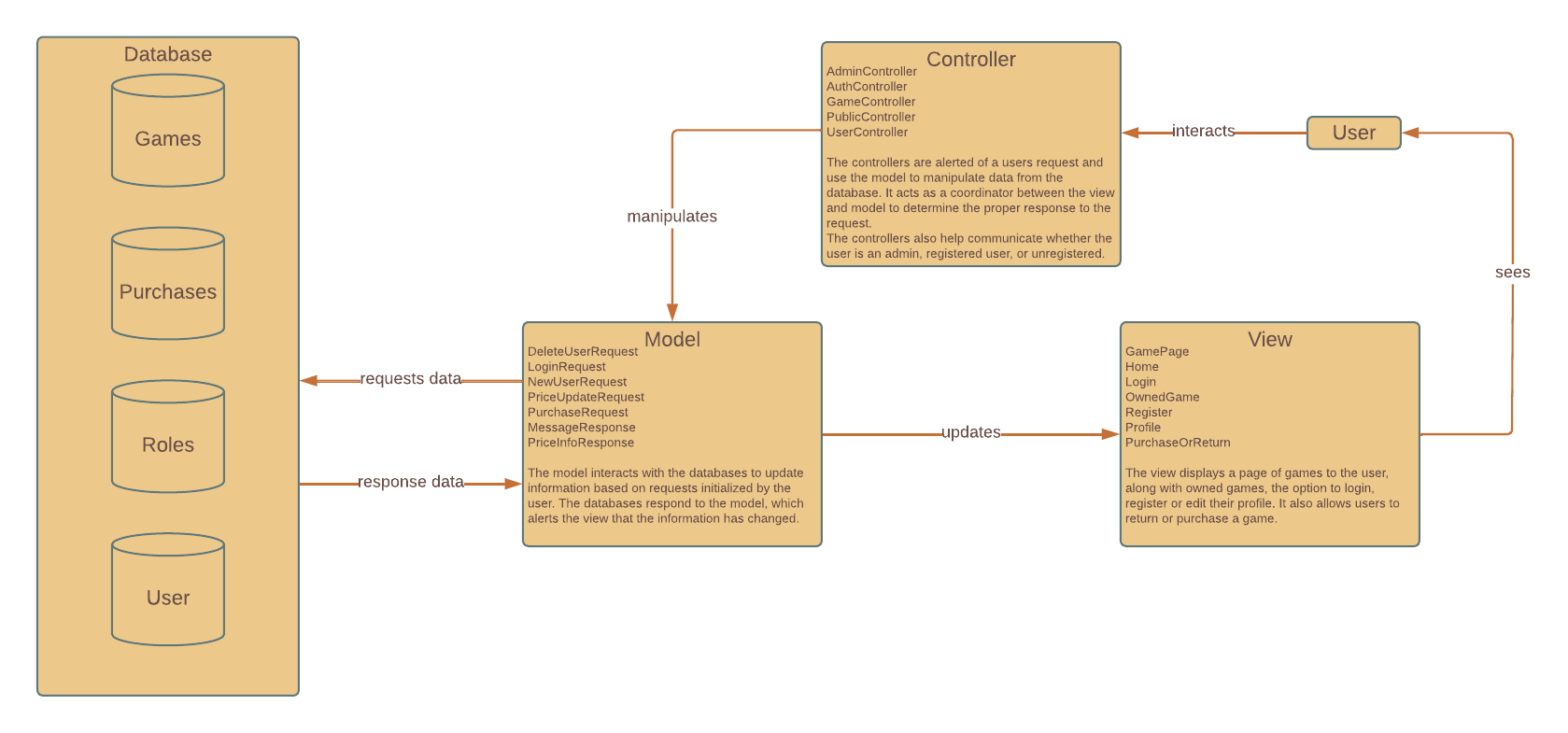
* Update Game Pricing:  While navigating through the catalog the admin can click on any game and update its price to any valid price the admin decides on.
* Remove users: The admin has access to an “Admin Board” which lists all the registered users of the website, and the admin has the ability to remove any users from the registration list removing their ability to log in with that specific account.

**Scenarios:**

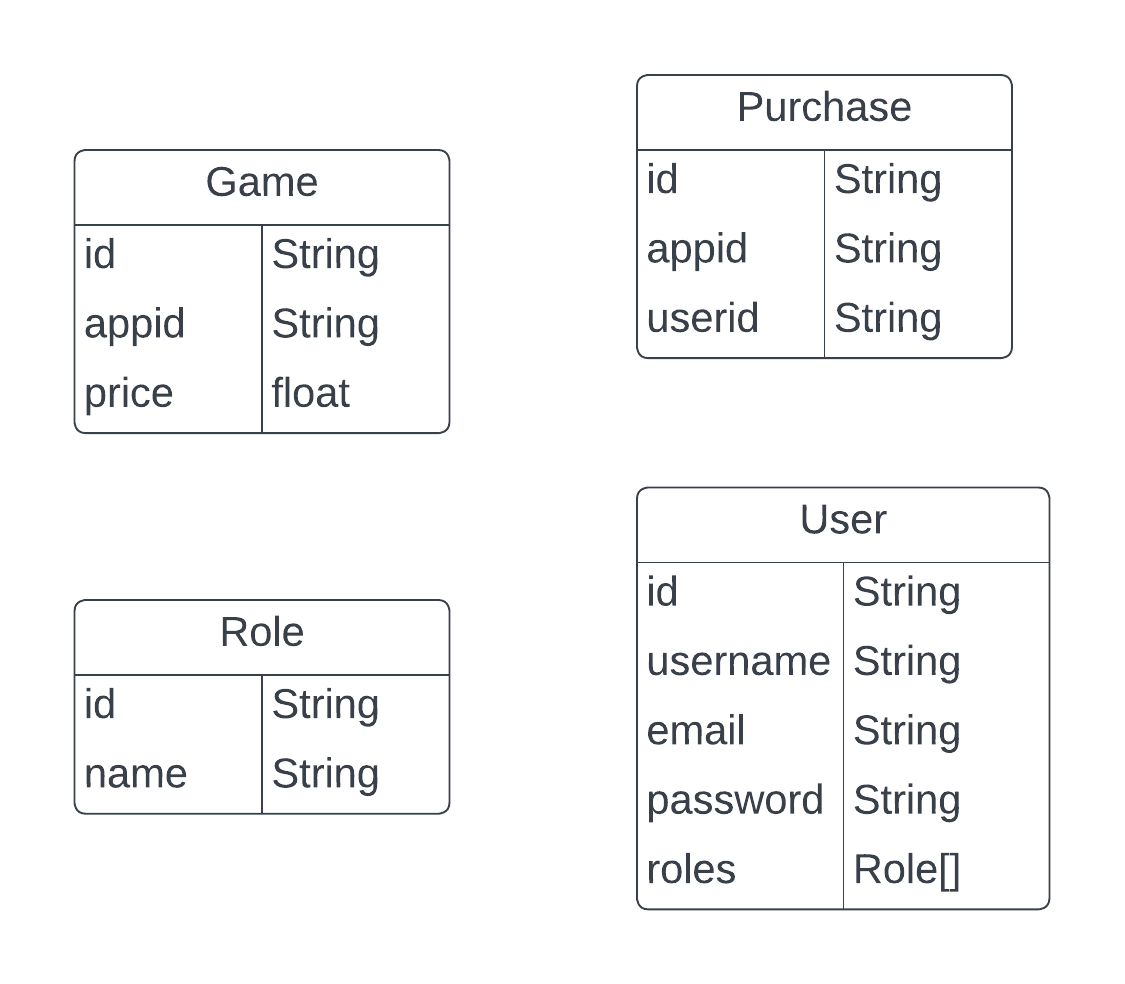
* **Register for an Account**
  1. Initial assumption: The username/email are not already associated with an existing account in the database. The user is not logged in already.
  2. Normal: A non-user will submit a username between 3 and 20 characters, a valid email, and a password between 6 and 40 characters (exclusive). The server will save the information to the MongoDB database, then redirect the user to a success page to confirm the user has registered successfully.
  3. What can go wrong: If the fields do not meet the length requirement, the user will be given immediate feedback of which field is not satisfactory. If the username or email are associated with an existing account, the app will display the rejected field for the user to fix.
  4. Other activities: After registration, the user will be able to use their credentials to log in to use the rest of the website as an authenticated user.
  5. System state on completion: After successful registration, the credentials will be stored in the database. They will be able to log in using the credentials with the role of regular user.
* **Browse Game Information**
  1. Initial assumption: The API for steam is up and running and the user has an internet connection.
  2. Normal: The user clicks from forward and previous pages viewing 32 games per page. From there they can decide which game they want to view more information about.
  3. What can go wrong: The user can press forward too many times and skip past a game they wanted. Pressing the previous button will fix the problem and let them navigate back to that game.
  4. Other activities: From this page you can click on a game and view its description and if you are registered you can purchase a game from there.
  5. System state on completion: The internal server does not get affected by this use case. Upon successful completion, the user will be able to view the catalog of games available by their picture.
* **Purchase/Return Game**
  1. Initial assumption: The user is logged into a registered account and has access to the list of games.
  2. Normal: The registered user returns the game, and it is removed from their list of owned games, and the game becomes available for purchase again. Or the game is purchased and is added to their list of owned games.
  3. What can go wrong: If a user tries to purchase a game, they already own they will be given an error message informing them that they already own the game. The button will update to offer the option to return instead.
  4. Other activities: They can browse for other games in our database of games
  5. System state on completion: The game is either added or removed from the MongoDB database of user game purchases depending on if they purchased or returned the game.
* **View Purchased Games**
  1. Initial assumption: The user is logged in to a valid registered account
  2. Normal: The user clicks on the “My Games” button and is linked to their list of purchased games.
  3. What can go wrong: If a non-user navigates to this page, they should be redirected to a different page. If the list of owned games is empty, the page should say empty instead of being a blank page.
  4. Other activities: From this list they can access the games they own and navigate directly to the games description and or return the game.
  5. System state on completion: On successful navigation to the My Games tab, the user will be able to view the games they own and have the option to return. Navigating to the My Games does not change the internal state of the server.
* **Update Game Pricing**
  1. Initial assumption: Admin has already logged in with a valid account.
  2. Normal: The admin navigates to the correct game and updates the price.
  3. What can go wrong: Updating the price might have one too many zeros. The admin is not limited to the number of times they can update the price, so they can quickly fix their mistake.
  4. Other activities: The admin can look at the description of the game , as well as decide to purchase the game for their own user account.
  5. System state on completion: , as well as decide to purchase the game for their own user account.
* **Remove users**
  1. Initial assumption: Admin has already logged in with a valid account
  2. Normal: The admin can select users to delete from the database
  3. What can go wrong: The admin is able to delete anyone with the click of a button, the admin should be prompted to confirm their decision. If a non-admin navigates to the admin board, they should be redirected to another page and unable to abuse the admin board.
  4. Other activities: Through the admin board, the admin can also view the list of all users. This is useful to decide which person they want to delete.
  5. System state on completion: Removal of a user is successful. The database deletes that user’s entry. They will no longer be able to log in with those credentials.

## Design Document

**MVC Controller**

****

**Data Schema**



This diagram is not an entity relationship diagram because we used MongoDB, a NoSQL document-based database. There are no inherent constraints on relationships between tables, even though some fields reference the same piece of data. We used Atlas, a free cloud-based MongoDB instance.

**State Machine Diagrams**

Diagram

Description automatically generated

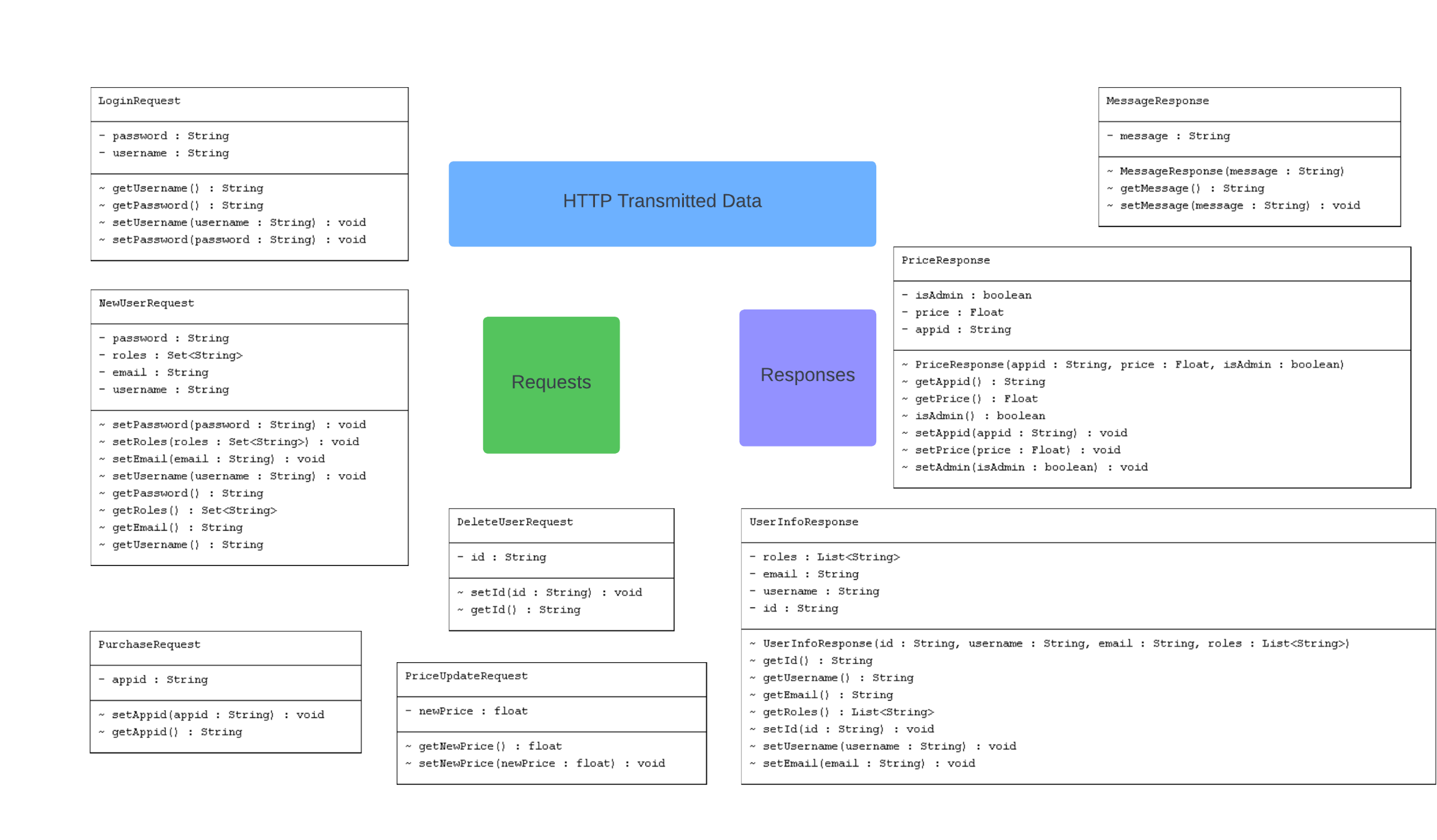
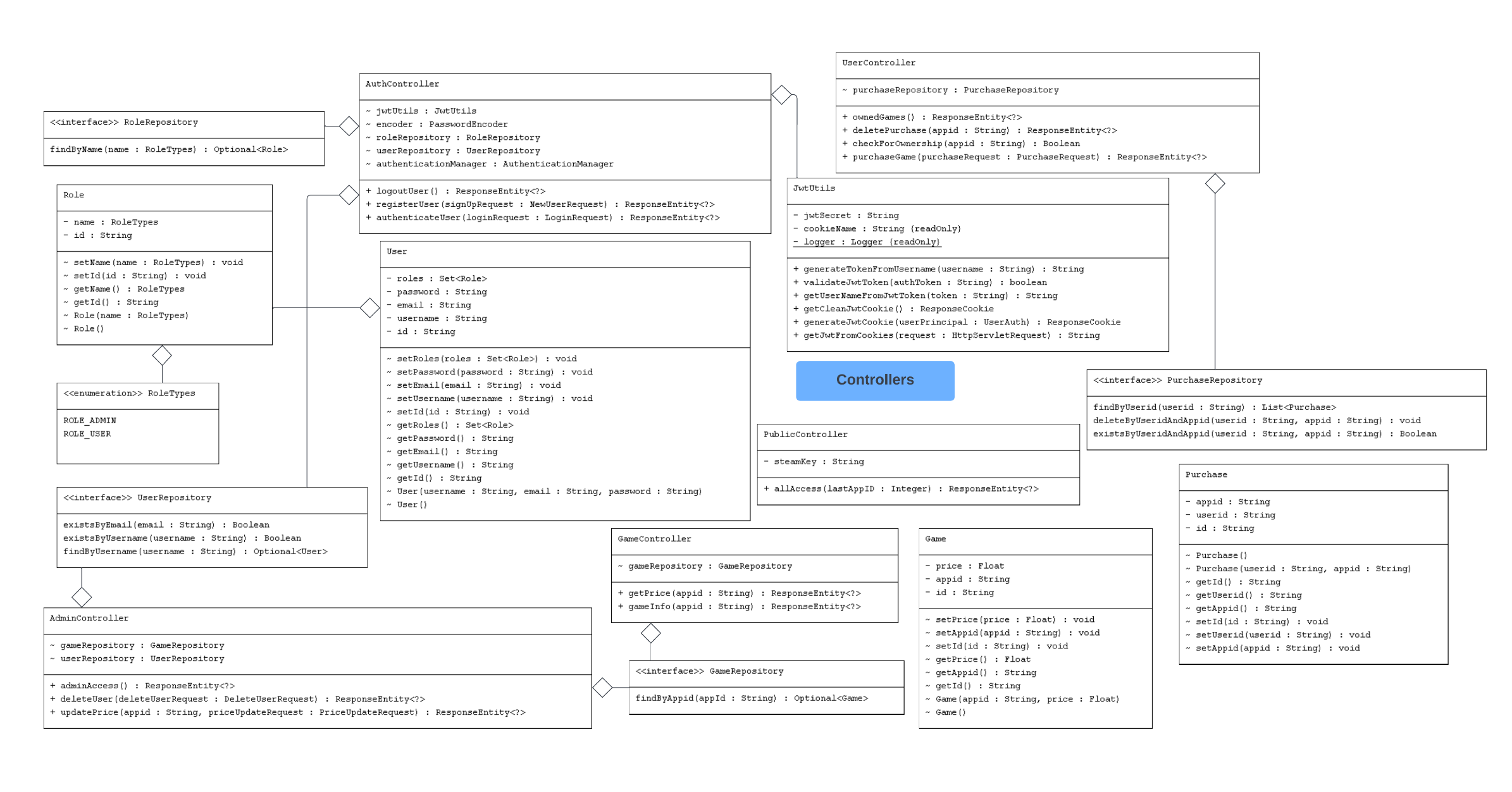
Diagram

Description automatically generated

Diagram

Description automatically generated

**UML Class Diagram**

****

****