**.NET Application Programming**

**Project Status and Design Report**

|  |  |  |
| --- | --- | --- |
| **Topic:** | CLC Milestone 6: Final Project | |
| **Date:** | 2/20/2021 | |
| **Revision:** | 6.0 | |
| **Team:** | 1. Shawn Fradet | |
| 1. Richard Williamson | |
|  | |
|  | |
| **Weekly Team Status Summary:** | |  |  |  |  | | --- | --- | --- | --- | | **User Story** | **Team**  **Member** | **Hours**  **Worked** | **Hours Remaining** | | Create presentation of product to show final product to the stake holders. | Richard | 4.0 | 0 | | Incorporate Logging through dependency Injection | Shawn | 2.0 | 0 | | Have application obtain scores through REST service | Shawn | 2.0 | 0 | | Update design documentation. | Richard and Shawn | 1.0 | 0 | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | | |
| **GIT URL:** | <https://github.com/darthxvaderxd/CST-247-Project> | |
| **PRESENTATION**  **URL:** | https://youtu.be/BIalO2v0cGs | |
| **GIT URL:** | *Y* | We acknowledge that our team has reviewed this Report and we agree to the approach we are all taking. |

**Planning Documentation**

**Agile Scrum Product Backlog:**

<https://github.com/darthxvaderxd/CST-247-Project/blob/master/agile_artifacts/SprintProductLog.xlsx>

**Agile Scrum Sprint Backlog:**

<https://github.com/darthxvaderxd/CST-247-Project/blob/master/agile_artifacts/SprintBackLog.xls>

**Agile Scrum Burn Down Chart:**

<https://github.com/darthxvaderxd/CST-247-Project/blob/master/Planning%20and%20Design/SprintBurnDown_Sprint_2.xlsx>

**Agile Retrospective Results:**

|  |
| --- |
| **What Went Well** |
| Communication between the team went well |
| Breakdown of the work went well |

|  |  |  |
| --- | --- | --- |
| **What Did Not Go Well** | **Action Plan** | **Due Date** |
| Updating the application to use the REST service we implemented. | We did not realize that the application needed to call the REST service for high scores. This will be fixed. | 2/21/21 |
|  |  |  |
|  |  |  |

**Design Documentation**

**Install Instructions:**

This project is being designed with ASP.NET MVC. The required database can be configured with the provided DDL script in Microsoft SQL Server. Create a database within SQL Server and load the DDL script provided. This script will create a user table, highScore table, and a savedGames table .

To run the main Minesweeper application, you must load the project with Visual Studio. The Minesweeper application requires the MinesweeperRestService be running at the same time for high scores. To ensure both projects are running, right click the solution and select Set Startup Projects. Select “Multiple Startup Projects” and ensure both projects have “Start” as their action. The project can be started with F5. To play the game a user must register and login. Once logged in the user will be provided with the Play! Menu options.

localhost:[port]/Home

REST Services can be accessed with the following:

localhost:[port]/ScoreService.svc/GetScoresById/{id}

localhost:[port]/ScoreService.svc/GetScoresByUsername/{id}

localhost:[port]/GetTopTenScores/{boardSize}/{difficulty}

**General Technical Approach:**

The approach for this group project is to reuse Minesweeper code from a previous class and adapt it to a web application. We will be following the guidelines provided by the class to incorporate upgrades to the initial code. These updates are to include:

* A secure site
* User Registration/Login
* Saving game state and stats
* Displaying game stats and results with HTML
* Retrieving game stats and results with a Web Service Interface

Milestone 1: During our Milestone 1 meeting we laid out the user stories for the different milestones and divided work for the first Sprint. The first Sprint will cover creating code for an initial user registration and login screen.

Milestone 2: For milestone 2 we implemented registration and login pages. Each page uses a separate controller and model for handling the handling of routing and data. Both the Registration and Login pages send their information to their controllers with HTTP POST for security. The form data from both pages is validated in the controller with the use of validation rules on the data models. Services were added for handling all business logic concerned with storing registration data and validating credentials for login.

Milestone 3: Milestone 3 required the integration of code from the previous Enterprise Application Programming class. The code we brought in included Board, Cell, HighScoreList, and PlayerStats classes. To utilize this code we created a GameService class that works with the previously mentioned classes to provide game logic. The GameService is used by a GameController and Game View to display the board and handle user interaction with it.

Milestone 4: In Milestone 4 we have updated the game board to clear all tiles from the board instead of exposing only the mines. We have provided the user choices in board size and difficulty. The Views were redesigned to implement Ajax calls and only refresh the game board. The additional elements of a timer and a turn counter have been added for the user to monitor their performance. User performance is saved to the database and used for a high score list that is displayed after the game.

Milestone 5: For Milestone 5 we have implemented a Save/Load game feature and REST service. The save game function takes the current time, turns count, username, board size, difficulty level, and board state and saves them to a database. The board state is serialized into a JSON string for storage. The GamesDAO allows for each user to have one saved game for each board size and difficulty. The load function restores the users saved game for the board and difficulty they are currently playing.

The REST service is implemented with Windows Communication Foundation (WCF). This REST service provides the ability to retrieve a user’s scores by either their database ID or their username. The results are returned in a JSON string along with a status message and any error codes.

Milestone 6: The first thing we did with Milestone 6 was change the source of the high scores that are displayed when a user wins a game. These high scores are now obtained through the REST service. For dependency injection we used a Unity container and created a ILogger interface with NLog. The logger is created through property injection for logging entry to an exit from controllers and their action methods. The logging is implemented into the controllers through a CustomerLoggingAttribute added to the class. We had already added a CutomAuthorizationAttribute earlier in the project. Logs can be located in base\_directory/logs.

**Key Technical Design Decisions:**

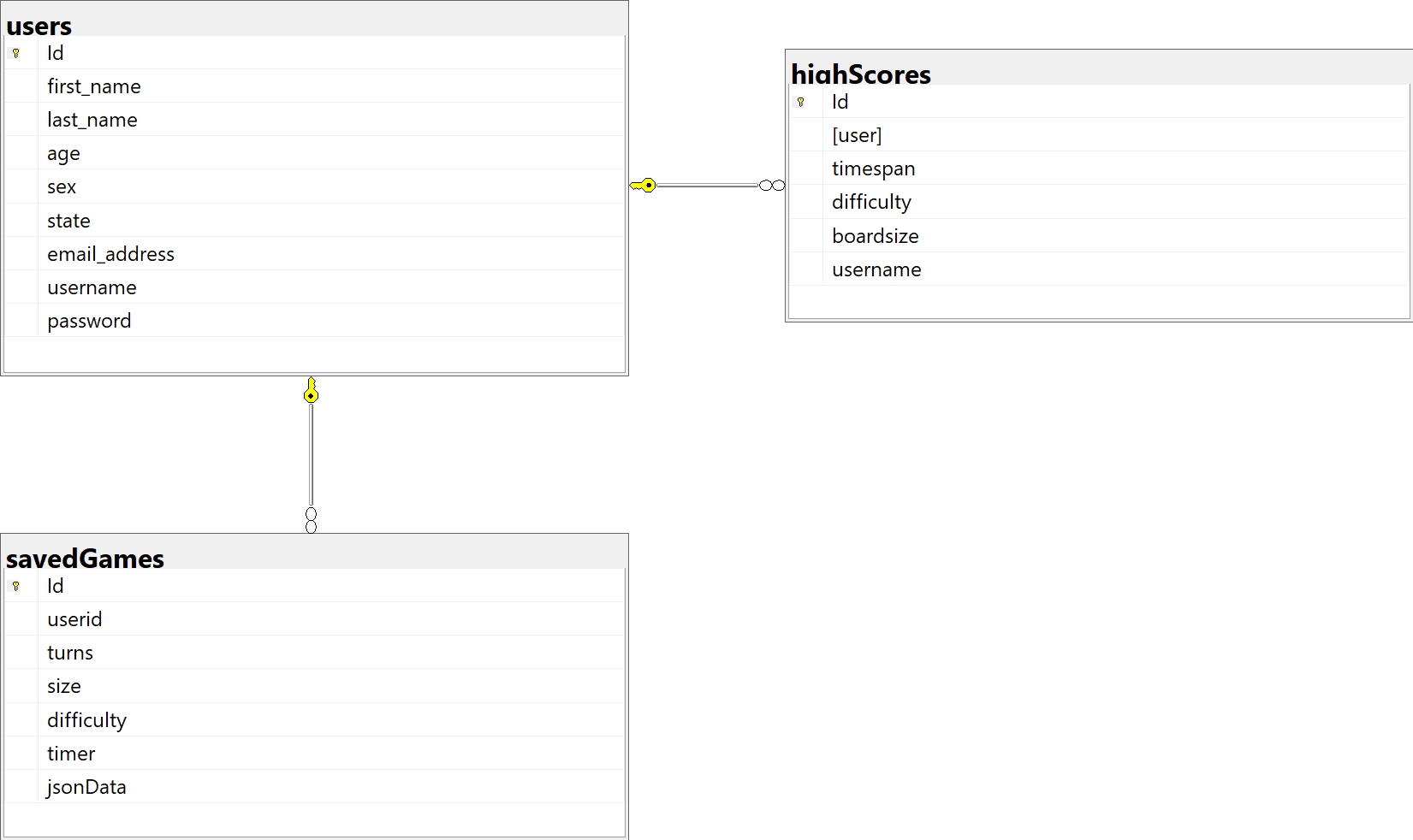
This project will be completed using ASP.NET MVC as the basis for the web application. ASP.NET MVC provides a framework that will let us build Controllers to work with data between a View and a Model. The View will use Razor pages and AJAX for the dynamic and responsive content needed for a game board. The Controllers and Models will be coded with C#.

For data storage this project will use SQL Server. ASP.NET MVC will communicate with the SQL Server through ADO.NET connections.

The REST API is implemented with Windows Communication Foundation (WCF).

Dependency injection utilizes a Unity container for property injection and NLog for the logging service.

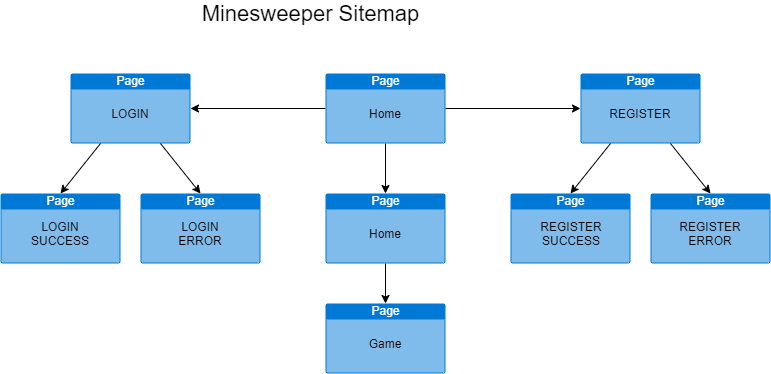
**ER Diagram:**

**

**DDL Scripts:**

[*https://github.com/darthxvaderxd/CST-247-Project/blob/master/Planning%20and%20Design/DDL%20File/*](https://github.com/darthxvaderxd/CST-247-Project/blob/master/Planning%20and%20Design/DDL%20File/)

**Sitemap Diagram:**

**

**Security Design:**

To enforce security for the website we have required a user to be logged in. Login is controlled through a custom authorization filter. The authorization filter checks to see if a valid user has been logged in and identified withing the Session information. The filter has been applied to the Home and Game controllers.

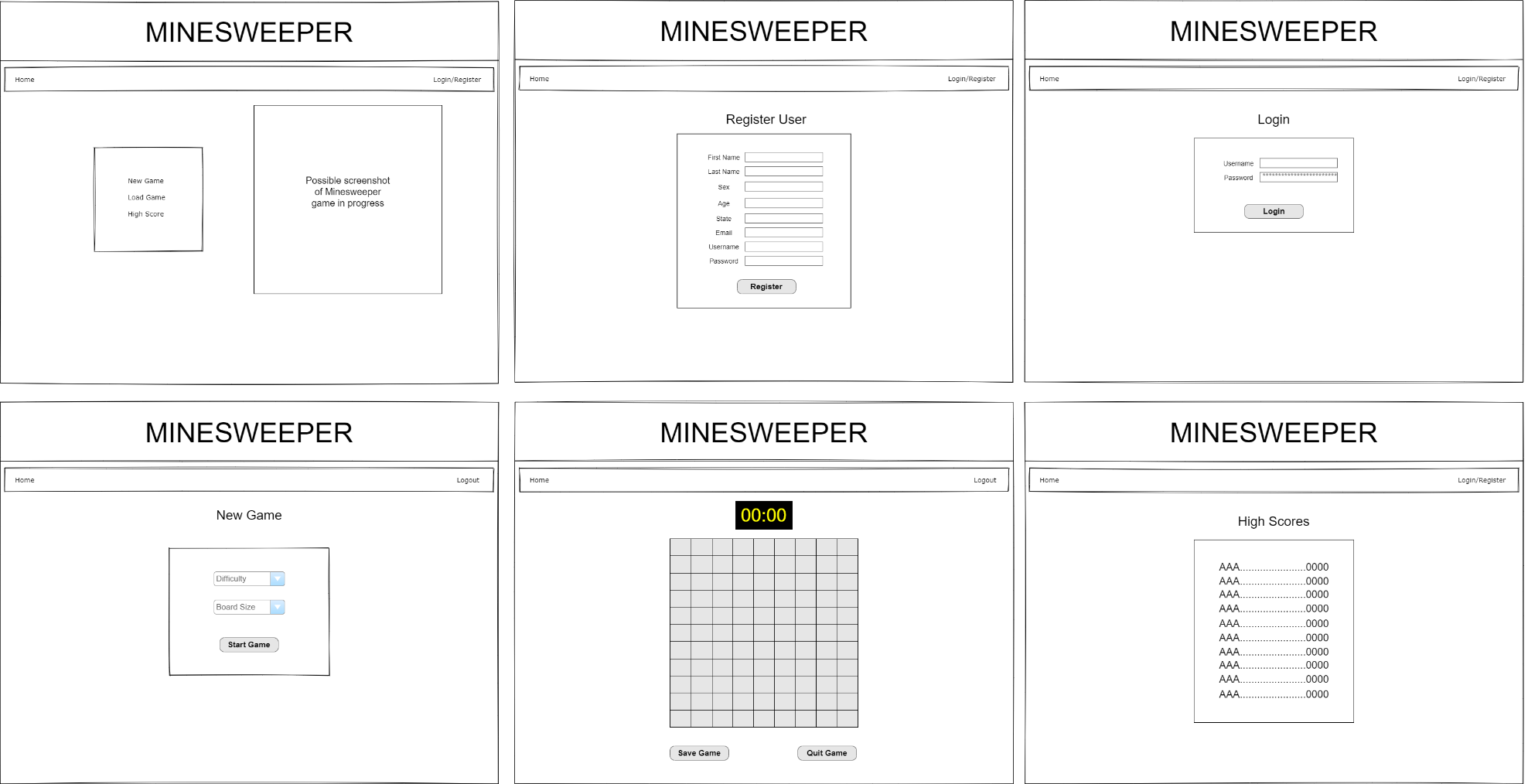
**Third Part Interface Design:**

*This section should fully document any Third Party Service Interface API’s, how to access the service, what parameters are required by the API, and the detailed JSON data format specification that could be used by a third party developer to integrate with the service and API.*

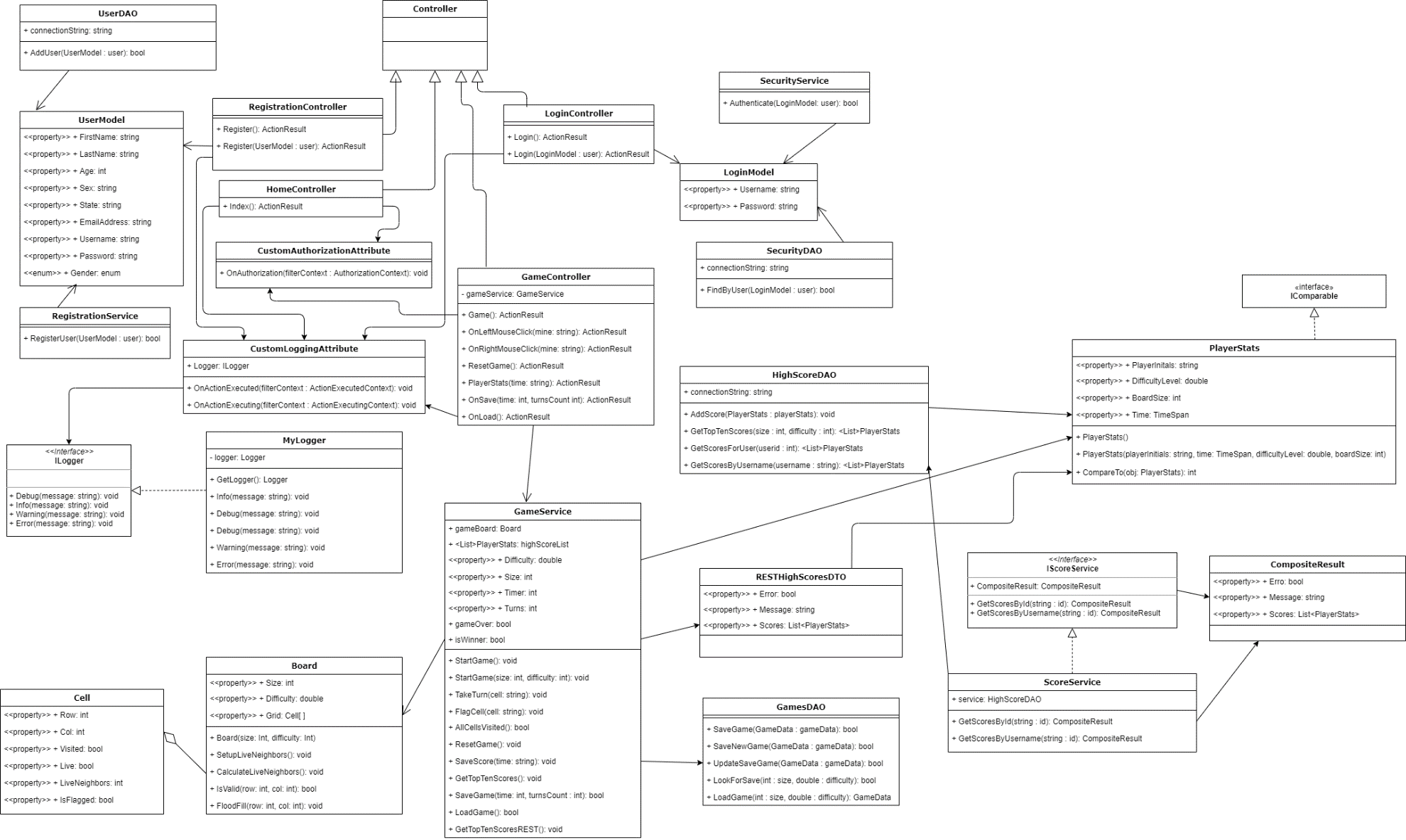
**Flow Charts:**

*You should insert any flow charts here. Flow charts should document algorithms or workflow that will be implemented in your program. At a minimum this should contain a flow chart of the Minesweeper game logic.*

**User Interface Diagrams:**



**Class Diagrams:**



**Pseudo Code:**

*You should provide BitBucket URL references to any code stubs & pseudo code. If you have no supporting documentation please explain the rational why you are able to leave this section as N/A.*

**Other Documentation:**

Team will be meeting daily on a Discord channel to update progress for Scrum.