Gamification Backend Documentation:

To login to server:

1. ssh ccain.eecs.wsu.edu
2. Username: auser
   Password: g0HUsk!E$ (don't ask me)

Viewing mongoDB data:

Command is mongo --port 443

From here you can query the database. The live collection for players is db.playersV2.

Example command for finding a player:

```
> db.playersV2.find({Username:"<USERNAME>"}).pretty()
```

Listing entire database:

```
> db.playerV2.find().pretty()
```

Game server information

Game server is located in **gameserver2017** directory inside the home directory.

In here you will find the source code for the game server as well as the tools to build and run it.

Building and running the server

The source code for the server is found in the **src** directory inside **gameserver2017**. To compile this code, follow these steps:

1. `$ sudo ./buildserver` (this will create the server.exe file, which will need to be ran in order for server to be functional)
2. `$ ps aux (`this will display all running processes on the computer, we will need to locate the process id (pid) of the server, it should be the entry that looks like this:
   `sudo mono --gc=sgen /home/auser/gameserver2017/server.exe`
3. Once you locate the correct pid of the server, kill the process: `$ sudo kill <pid>`
4. Now, to start the database and server, use the command:
   `$ sudo ./startDbAndServer`

5. If it hangs here while saying something about appending to output, just hit ctrl-c, the server will still be up and the output will still be appended to the log file.
6. Use another `$ ps aux` to verify that the server is up and running.

Source code:

The source code is a selection of C# scripts that control how the front end Unity game interacts with the backend server and mongoDB. In this section I will go into more detail on what each of these files accomplishes in regards to the project.

1. AccountCreation.cs: This script is somewhat negligible, and could be refactored to be in a different part of the code. It is used when a message that starts with “CREATE” is sent to the server, and it splits up the remainder of the message into a new username and password that gets sent to the function in Database.cs called CreateAccount(), which actually inserts a new account into the database.
2. Bonus.cs: Idk what this does.
3. Client.cs: A very important file whose main attraction is processing messages received by the server. Once a message is sent from the server from Unity, Client.cs calls the ProcessCommand() function which can perform the correct server-side function associated with that message, usually a database update or a request for login/leaderboard information. Beware, there is some unused code in this file that needs to be sifted, but is kept in case some of these features are to be implemented into the game again (Achievements, for example).
4. CSVHelper.cs: Helper class to parse comma separated string
5. Database.cs: Class that declares functions used by the MongoDatabase.cs class
6. Logger.cs:Class that helps log what the server does. The output of the logger is placed in nohup.out in the gameserver2017 directory.
7. MongoDatabase.cs: This is the file where the interaction with the MongoDatabase occurs. Inside you will find important functions for logging in, saving and retrieving data from the DB, and registering players.
8. playerInfo.cs: This file contains structs that are used for deserializing json strings containing incremental information.
9. Program.cs: The file that contains the main method used to start the program.
10. Server.cs: This file contains the code that starts the server and allows it to accept connections from users.
11. TcpNetwork.cs: This file contains legacy code that is currently unused by the server.

Tips and tricks:

- If a problem is found with the server’s behavior, check the nohup.out log first, as errors in server code will be put there as well as a log of anything that is happening. It has restricted access, I usually just use $ sudo cat nohup.out to read the last part of the file where the error should be.

- New features added to the backend should output information to the log, so it remains a helpful source of information. To do this, use: `logger.Debug(<String here>);`
- It is useful to check the status of the MongoDB to debug what could be going wrong. Make sure that the information saved in the DB is what you expect and that it persists in the DB after logging in to the game on Unity.
- In the future, it might be valuable to have a test server and a main server that are both functional when the game is deployed.