Requirement specification Document - 39224996

1 Description

The project is meant to simulate the interactions between the servers of a multiplayer game.

1.1 Content

- Description
- Case Study
- Installation Guide
- Getting Started
- End Points and Usage Examples

1.2 How to use this document:

1. **If you want a quick start of the project, follow the steps in sequence:**
    1. Please go through **Installation Guide**. The section contains all the preparations and the requirements of the project.
    2. Please then go to **Getting Started**. The section is about how to starts the servers. Note that because each server would not work on its own, it's better to start all the servers.
    3. Finally go to **End points and Usage Examples** to see how to use each feature.
2. If you want to know more about the project:
    - **Architecture and Route** describes how all the servers are cooperating.
    - **Features** section is the blueprint/design of the whole project.
    - **Other Resources** section contains addtional and interesting resources.

1.3 Architecture and Route

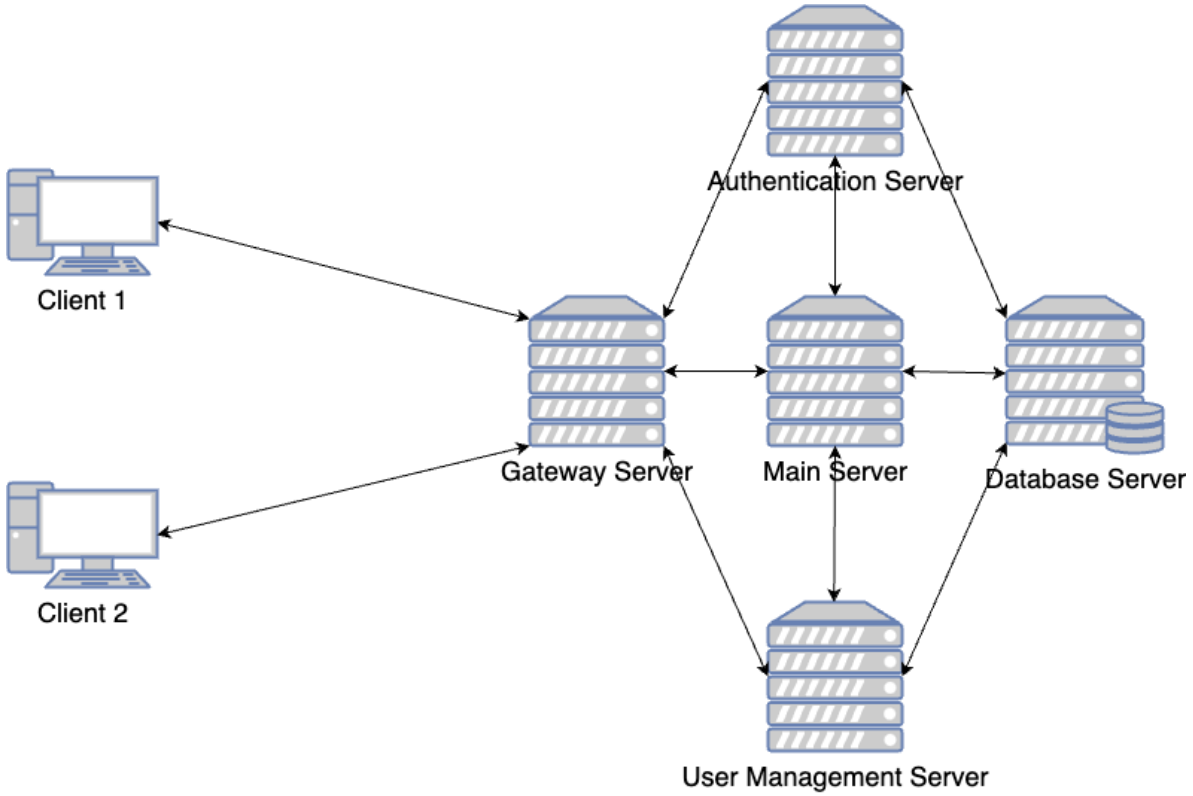

Figure 1 Architecture of Servers

1.3.1 Client and Frontend (HTML)

The whole application will displayed on browser webpage (as a client).

All request heading to backend need to first make communication with Gateway Server. (Exception: After verified, in order to reduce delay, the player will communicate directly with Main Server)

1.3.2 Gateway Server (Express.js)

According to the address of the request, the Gateway Server will forward the request to different directions.

A gateway server can help both distributing the flow of data and preventing the client to contacting the backend without permission. (It can be set in backend servers that only the connections and requests from gateway server is allowed.)

1.3.3 Authentification Server (Express.js)

The Authentification Server is used to verify user account. It receive the request and data package forwarded by Gateway Server originally from client, then compare the request data with the data stored in database. If verified, it will generate a secure key, and send a package (contain the username and the key in pair) simultaneously to both the client and the Main Server. The "two" key must be matched so that the request is allowed to visit the mainpage.

Metaphorically speaking, the Authentification Server is like the hotel reception. It checks that the visitor is the one who has reserved a room, and give the visitor a unique door card to open the room.

1.3.4 Main Gaming Server (Express.js and Socket.io)

The Main Server is core section of the whole project. It uses socket to communicate with the clients. Once a client is successfully connected, the Main Server will hold the connection and keep receiving and broadcasting messages and data packages.

In my understanding, the normal HTTP protocol and the socket protocol are like communicating by email and by phone call. Although theoretically if you can write and read the email in a incredibly fast speed, it would also have an effect like real-time process with imperceptible delay.

1.3.5 User Management Server (Django)

The User Management Server is mainly for the purpose of initializing the models of the database, and handling the requests for static request like registration, charcter creation and profile management.

1.3.6 Database Server (MySQL)

The SQL Server will restore relational data all about the project.

1.4 Features Designed

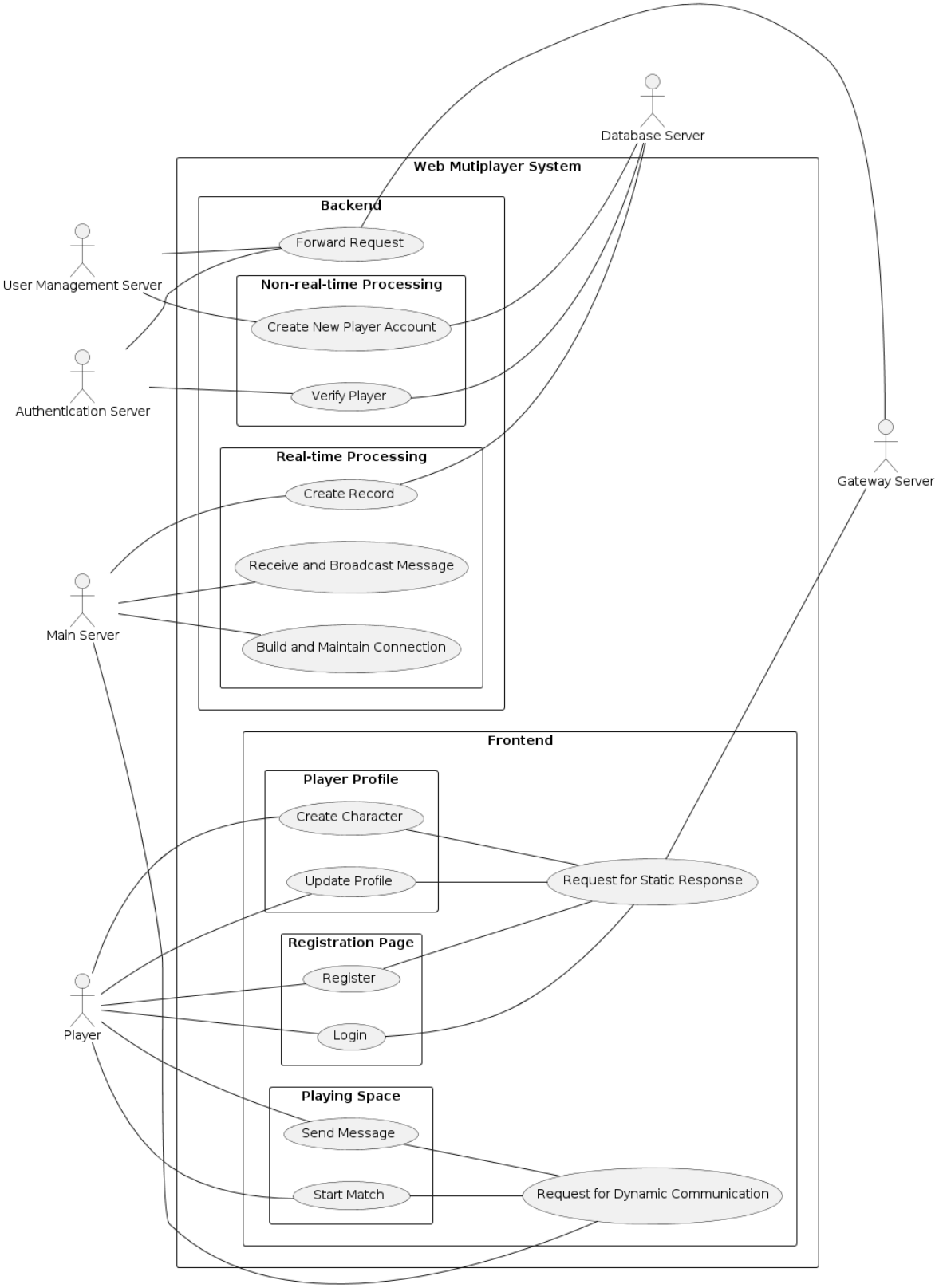

Figure 2 Usecase Diagram

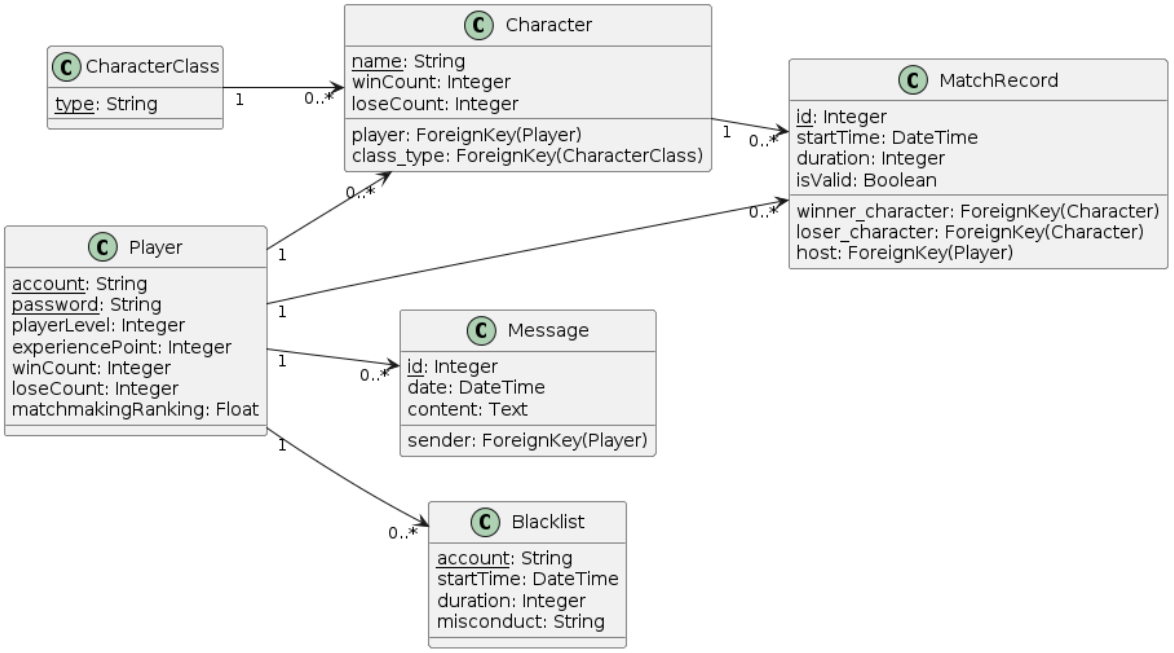

Figure 3 Class Diagram

Please note this part is mainly about the design of all functions. To see the practical usages, please refer to 'Usage Examples'

1.4.1 Login, Registration and Authentification

Account is the unique identificatio of player to register/login.

The player need to have an account for further functions.

1.4.2.1 Registration

New user need to input account and password to create new entity.

The account should not be used before in current database.

The password need to be at leat 8 character long.

1.4.2.2 Login and Authentification (double authentification steps)

The shining point of this system is that, it requires confirm and match from mutiple directions to verify if the login request is legal.

Please refer to the architecture diagram above for better understanding.

1. After taking the username and the password from input, the client will send to the Gateway server a request to login and a package of username and password.
2. The Gateway Server recognizes the login request and forwards the package to the Authentification Server.

3. The Authentification Server will compare the received data with the one in database. Only if the user information is verified, the process will continue.
4. If the Authentification Server has checked the data is valid, it will then generate a secure key.
5. Then the Authentification Server will simultaneously send the pair of data (the username as index and the secure key as value) to the Main Server and send the secure key alone back to the client.
6. Finally, the client should send the key it received to the Main Server. The Main Server will temporarily store the pairs of data in a list for convenience. Then, the Main Server compares the key from client and in the list. If the key matched, then the client will be allow to have access to the main page.

1.4.2 Player Profile

1.4.2.1 Player

Player is the user of the application.

This entity is designed to be **unique** for every user.

- The public attributes (the player can see and modify some):
  - account and password
  - level and experience Points
  - number of wins and loses (the total number of all characters)
- The private attributes (the player is forbiddened to see and to modify):
  - Matchingmaking Ranking (the index for matching and ranking!)

1.4.2.2 Character

Character is the avatar (virtual projection) of player.

Every user can have up to **10** characters.

- The public attributes (the player can see and modify some):
  - name of the avatar
  - class (such as Healer, Defender or Attacter)
  - number of wins and loses
- The private attributes (the player is forbiddened to see and to modify):

- player (who is the owner)

1.4.3 Real-time Gaming and Chat Box

This part is supposed to be the main body of the whole project.

In order to achieve the effect of real-time gaming and chatting here, Socket protocol is used to substitute for HTTP protocol.

1.4.3.1 Game Canvas

- place for visual elements of game (picture, timer, HP)

1.4.3.2 Chat Box

- history message
  - place for boardcasting message from server or from other player, and displaying history input
- input form
  - place for inputing command (for example to attack)

1.4.3.3 Ranking and Level

Still in developing

2 Case Study

2.1 Case Study 1

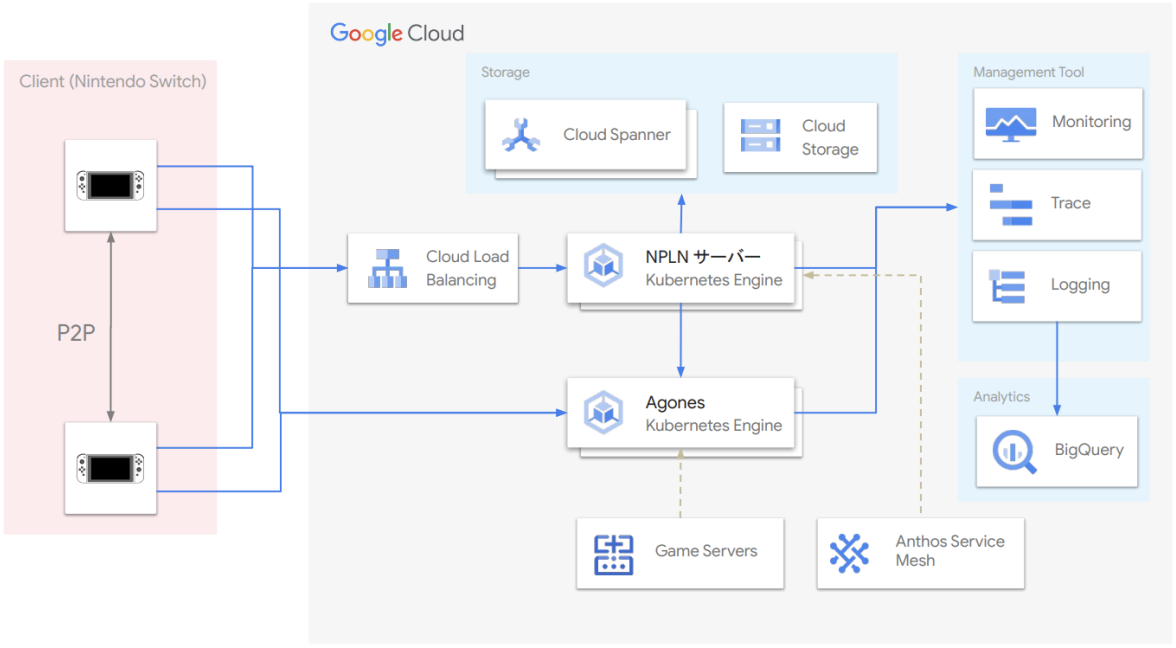

Figure 4 NPLN Server Structure

The Figure 4 (https://blog.lxdlam.com/post/24e3aa77/) illustrates the a real-time game server of Nintendo. The core mechanism of the architecture is that, the main server is only responsible for matching, directing (how the player clients should being connected) and recording the processes and results of the matches. In a word, one of the client device will server as the actual host of the whole match.

Figuratively speaking, you want to attend a party. Normally the organizer should book and arrange a public hall for all guests. But this server instead appoint one of the guest who has the most convenient location for everyone to join in.

By comparing how architecture like this works with other normal cases which I have experienced helps me understand how a real-time game server could work like. But in order to implement a real project theoretical knowledege of server architecture is far from enough.

2.2 Case Study 2

The second case is a simple mutiplayer online game project based on Java (https://scripterswar.com/tutorial/nodejs).

In this web application, users are allowed to create their personal accounts, set up characters and get involved in the real-time online mutiplayer game. All the user information and game process would be recorded.

In the case, the author use Express.js, socket.io and MangoDB to build up the main server, where Express is used for resources management and socket.io handle the most core real-time features. But learning to use socket.io to build the main server has a flaw on my project. Without enough knowledge on Java and networking, it is hard for me to split and expand the function of information storage and authentification.

3 Installation Guide

Please follow and check the steps before to prepare for the project.

For general the project require:

- Python 3
- Node js
- MySQL

1. **Install Mysql Community Server**

   (Very important!) Install and make sure your MySQL server works properly

   Here are the official documents of MySQL Server. (Containing all popular system)

   - Download MySQL Community Server
   - Get Started with MySQL

   (Very important!) Make sure you have successfully build a MySQL connection

   - Check List (will be used later):
     - your username of the connection (usually name `root` in default)
     - your password (recommend: `hellomysql`, so that no need to modify settings in codes)
     - name of the database (in the default setting in my code it is `Moba_Simulator`, however this is changable if you want to build your own one)
     - host (usually offline it should be **localhost (127.0.0.1)**)
     - port (in default 3306)

   Here are some additional tutorials and resources if you meet difficulties.

   - Tutorial to install MySQL server and visualization database controlling platform
     https://www.youtube.com/watch?v=7S_tz1z_5bA&t=65s
   - Tutorial to connect Django and MySQL server
     https://studygyaan.com/django/how-to-use-mysql-database-with-django-project
     https://www.youtube.com/watch?v=SNyCV8vOr-g&t=49s

- If you have problem on Mac when installing MySQL client in python
  https://stackoverflow.com/questions/66669728/trouble-installing-mysql-client-on-mac

2. **Install Node js and npm**

   Node js and npm is mandantory to run this project. According to different systems the way to install may differ. Make sure npm is also installed.(usually they are binded)

   - Node js

   To verify the installation of Node js and npm, run:

   ```
   node -v
   npm -v
   ```

   Other Tips:

   - If you have problem on MacOS when installing, you can try Homebrew. It is like environment package manager for MacOS. After installing Homebrew, you can use it to install, run:

     ```
     brew install node
     ```

   - For me it is always a good idea to seek for the help of Homebrew, when meeting trouble installing environments on MacOS.

3. **Install Python**

   Make sure Python 3 is installed on your device.

4 Getting Started

1. **Clone Repository**

   Make sure you know how to use git command; or you can use Github Desktop without using commands for convinence.

   ```
   git clone https://github.com/lzpmpc005/Simple_MOBA_Server_Simulator.git
   ```

   Get into the file (according to your file structure)

```
cd Simple_MOBA_Server_Simulator
```

2. **Install Dependencies**

   For Servers using Node js, make sure you are in the file Simple_MOBA_Server_Simulator and run:

   npm will automatically read the package-json file and install all the dependencies.

```
cd authentication_server
npm install
cd ../gateway_server
npm install
cd ../gaming_main_server
npm install
```

   For Django Server, first back to file Simple_MOBA_Server_Simulator, then run:

```
cd user_management_server
pip install -r requirements.txt
```

3. (Optional) Check setting of Database (if you set up your MySQL connection using your own username and password)

   - In every server folder you will see a file named `main.js` , `auth.js` or `gateway.js` .
   - Open and check if there are settings like:

```
const pool = mysql.createPool({
    connectionLimit: 10,
    host: 'localhost',
    port: '3306',
    user: 'root',
    password: 'hellomysql',
    database: 'Moba_Simulator'
  });
```

   - In Django Server, go `user_management_server` --> `user_management` , find a file named settings.py and open it, you may see settings like:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
```

```
        'NAME': 'Moba_Simulator',
        'USER': 'root',
        'PASSWORD': 'hellomysql',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

- If you have use any parameter other than the one showed above, you need to change to yours. Otherwise the connection between will not be set properly.

4. **Migrate Database using python**

   After everything below checked, you should initialize your database.

   - make sure you are in `user_management_server` directory
   - run:

   ```
   python manage.py makemigrations
   ```

   ```
   python manage.py migrate
   ```

5. **Start Servers**

   The final step is to start server. Either you can manually run each one or try:

```
# Start Authentication Server
cd authentication_server
node auth.js

# Start Gateway Server
cd ../gateway_server
node gateway.js

# Start Main Server
cd ../gaming_main_server
node main.js

# Start User Management Server
cd ../user_management_server
python manage.py runserver
```

5 End Points and Usage Examples

Before starting, make sure all the servers are running properly.

For the first 2 completed endpoints, use the browser to see effects. Click here to try the main flow of the project.

For the rest of uncompleted endpoints, it is recommended to use Postman (click the link to register and download) to test. Here is also the official tutorial for Postman.

- Register and Create Player
    1. go to http://localhost:3000/
    2. input username and password (more than 8 characters)
    3. if successed, you will stay at the same page
- Login
    1. should have register first
    2. go to http://localhost:3000/
    3. input username and password
    4. if successed, you will see the main framework of the game canvas
- Create Character

    Here you need to use Postman to send and receive request and response

    url used: http://127.0.0.1:2000/django/player/create_character/

    - To receive success response, make sure:
        1. you have registered successfully before
        2. there is at least one data in Table: player_profile_characterclass in database

    request example:

```
{
    "name": "Ora",
    "account": "user123",
    "character_type": "Healer"
}
```