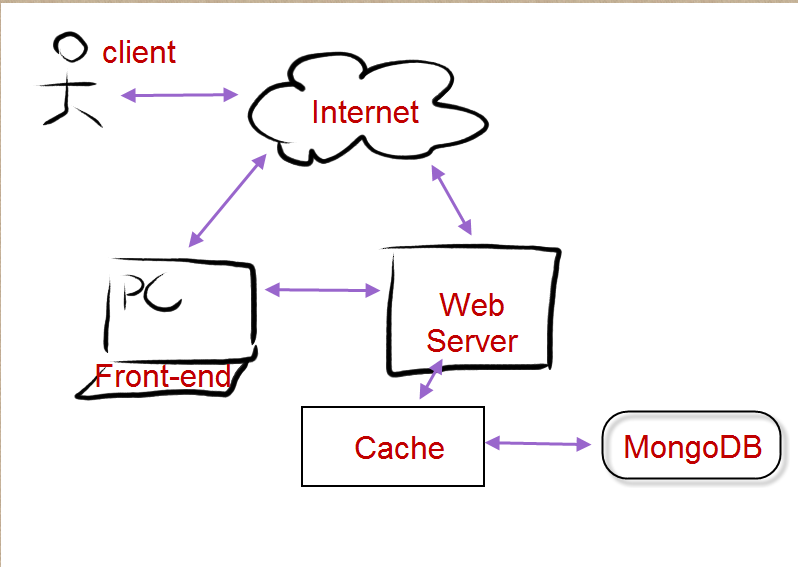
Read Me

I have created a server that based on the following assumption:

This is a MOBA game. Each user has a unique username and email when register. Both users and developers are allowed to use the server. But users can only use the server to get user public information, game statics and achievements, while developers are allowed to use the server create, post and update under certain circumstances, such as if a user is cheating in the game or being reported for several times, the developer is allowed to delete his achievements.

In the game, the user can join the team and pick a certain champion to fight with the other team. After each game, the game will update the game statics and achievements for each user automatically through the server.

Architecture:

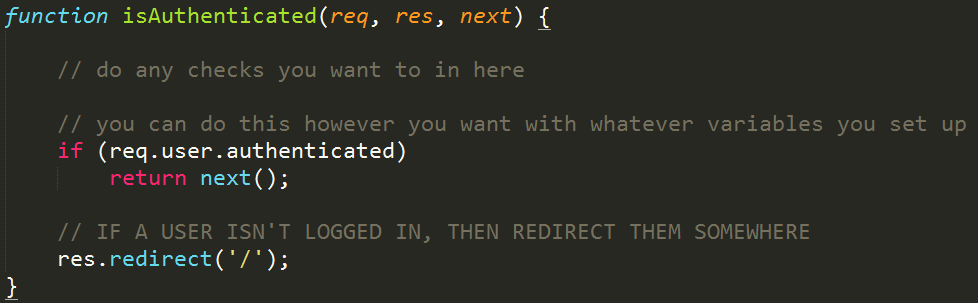


Pseudo:

1). Since user can pick any champion in each game, there should be a champion table for each user. I haven’t implemented the API for this table, but user and developer are both allowed to grab information from the server. This is very important in MOBA game, since users can check their game status and how their teammates performed in the game, as well as the developers can use the server to generate data and do the analysis on champions. This is a very important table in the whole database.



2). Since this server is exposed to all users, it should add a middleware authentication to prevent players post and update game statics and avoid cheating in some extent. We only allow admin or game server update the data when they passed the authentication.



3). Since this game is based on MOBA, which means game statics should be updated once each game is finished, we should actually build several servers to handle based on different functionality. In the real world the web server is consisted of several servers, like payment server, game statics server and achievements server to handle huge data interaction. We certainly should cache all internal data for each game, like game statics for 10 users in same game and post to the database to improve the performance.

4) Data validation. We need to make sure each data post to the server obeyed the certain rules, such as it’s not allowed to post both win and lose to certain user. We can create a rule-based model to verify data.

5) Friends. This is an important part in MOBA game since the platform provides more chances for people add friends and play together. A friendship is really a two-way relationship; each entity is linked to another. I would like to store as an array of user\_id and username in the User schema.

A few notes:

1). In order to protect user’s information, I only return public information and hide the password.

2) For password, I salt hash passwords using Node.js Crypto. Salt hashing is a technique in which we take user password and a random string of characters, called salt, hash the combined string with a hashing algorithm and store salt and hashed password in the database. This is a one-way process, which is quite safe to some extent. And this technology can avoid storing the same-hashed password in the database when different users have the same password.

3) For database, I chose non-relational database for several reasons. First, tradition DBs can’t efficiently store heavily structured hierarchical data. For each champion, we could build a custom object database and update the schema easily. What’s more, in the real game word, it’s more like that each champion is document stored. Secondly, mongodb provides more operations than SQL. Third, since MOBA is a real-time and allows hundreds users to play at the same, NoSQL is designed for high concurrency.

Tech Stack

Framework: Express

Database: MongoDB

QA: postman, mocha

To run the server:

* Install mocha: npm install mocha –save-dev
* Run database:

mongod –dbpath /path-to-folder/myapp/node\_modules/mongodb –port 27017

* Run application: npm start
* Run test: npm test