Design Document

PROJECT: **ORACLE**

Team #15

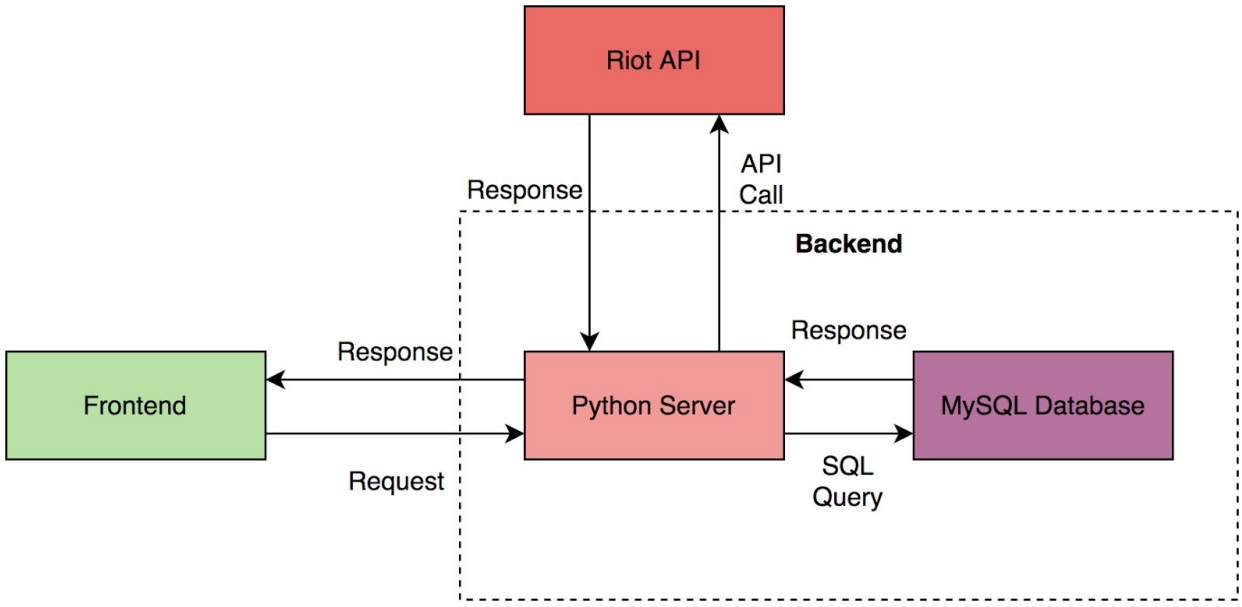
Rami Bitar, Alex Shelley, James Shao, Rohan Swaroop, Yash Pujara

Purpose

We hope to develop a website that provides a resource for people looking to improve at League of Legends. Our user base would consist of self-appointed coaches and students. After pairing students with coaches, we can use the Riot Games API to pull specific match data and even watch the match replay. From there, a shareable chat room is created where the coach can go over the game with the student and help improve their gameplay. This concept is directed towards the community where all members regardless of rank are always willing to donate their time.

Design Outline

We chose to employ a client-server model for our application. In our case, the client is a web frontend. When the user performs an action, for example, attempting to log in, the client will make a request to the server for the information it needs. In this example, since the user's information will be stored in the database, the server will make a query to the database, and retrieve the appropriate information. However, if the client makes a request that requires new data, for example, concerning data from a game that was just completed, the server will make a call to the Riot API, parse the response, and send it back to the frontend to be displayed as appropriate.



Design Issues

Functional

1. Do users have to log-in and/or create an account to use the service?

    a. No account is necessary, users can be paired up as a coach and or student and be given a guest username
    b. Accounts are required for all users
    c. Only coaches are required to have accounts

    Solution A was quickly discarded because we decided it was important to have user accounts in order to track, verify, and rate users. Because the application is volunteer based, it is important to keep track of the community and have some method of ensuring that the "free" accounts are not being abused. Coach accounts are especially important to be registered because students will need to rate their coach on various gameplay matters. On the other hand, student accounts could be guest accounts, however that creates a problem of verifying the user's account and their rank. Students also have an option to set various flags and preferences for coaching techniques and focuses. Overall, Solution B is best because it makes the most sense and helps us pair students with coaches.

2. Should the chat room be private or open to the public

    a. Keep the chat room private between the coach and student
    b. Create a shareable link that users in the room can see
    c. Allow the coach/student to toggle between options

    While most coaching cases will most likely be one on one, we discussed various scenarios where having more than one student could be useful. In particular, if a student wanted to invite their friend or teammates/opponents following a match. This eliminated Solution A, however B and C continued to be options. While we don't anticipate spamming, it could be an issue if a shareable link got spread around. Because of this, it makes more sense to combine Solutions B and C and create a toggleable privacy option that also provides a shareable link while the room is public.

3. What should the minimum requirement be to set yourself as a coach

    a. Set the minimum to a hard minimum (Diamond, Platinum, etc.)
    b. Don't have a minimum at all
    c. Set a minimum but allow users to rate coaches to eliminate fakes and trolls

There is no easy way to confirm the identity of any player. It would be trivial for me to pretend to be someone else, and riot does not provide an easy way of confirming identities. The compromise is if you set yourself as a player of Diamond elo but have the skill level and game knowledge of a silver player that will become apparent very quickly and so with the rating system for coaches in place once you fall below a certain rating you can not coach anymore and the email will be blacklisted. We can not ban the summoner name in case the real one appears.

Non-Functional

1. Is an API web-service required between the database, front-end, and pairing backend?

    a. Create an API that the front-end calls that connects it to the backend and database
    b. Directly pull data from the database using a server-side language such as PHP

    Directly pulling data is dangerous and also unwieldy, it makes more sense to create a "middleman" API service. This will allow us to standardize calls between all services. We'll create the API using a service class with functions that builds queries. This information can then be loaded into the chat room when the game ends.

2. What language should the backend (pairing functions) be written in?

    a. Python
    b. C++
    c. Java

    Though all these options are equally capable, option A is our preferred choice, due mainly to its simplicity. Python's immense library of modules and plugins, mixed with its simple and easy to learn syntax make it an ideal candidate for being able to quickly build a highly functional server capable of everything we need it to be able to do.

3. Should only "live" games be allowed?

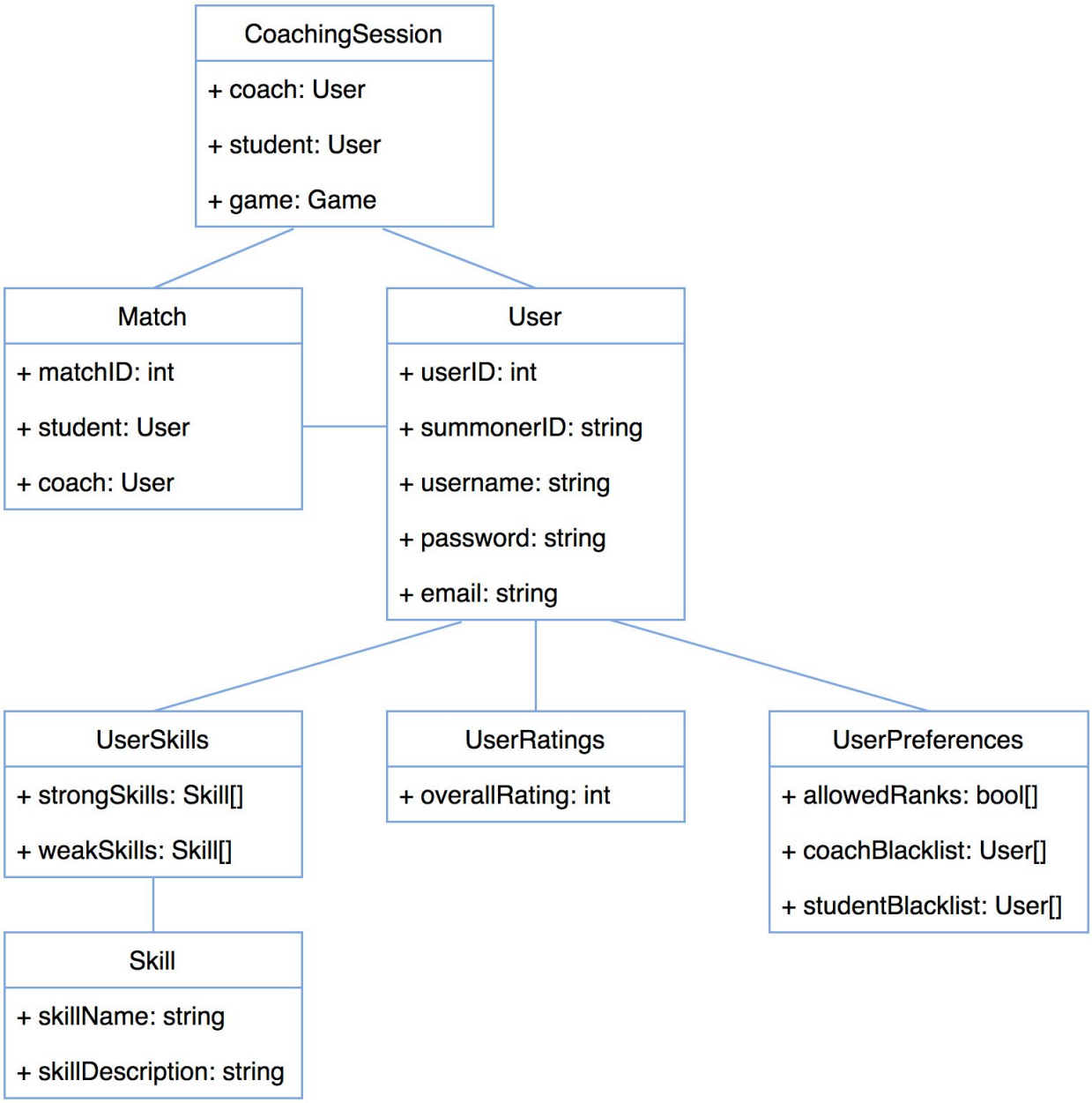
    a. Only current games
    b. Past games are allowed, given a username, past game data can be pulled
    c. Past games are allowed, given a replay file

    Option B can be immediately eliminated because coaches need some way of viewing the games, basic match data does not give enough information. Option C can be considered, however, League of Legends replays are notoriously buggy and difficult to work with. Our best bet is going with current games and using the spectate function at first. This

eliminates extra steps in the process such as matching a replay file with match history data.

Design Details

Class Structure



Class Descriptions/Interactions

CoachingSession

The CoachingSession class represents a single session between a coach and a student. It is tied closely to the Match class, as each unique CoachingSession is tied to a unique Match.

Match

The Match class represents a League of Legends match, and allows us to tie students to the match that they have entered in their game, so that coaches may be matched with them. The match ID will be given to CoachingSession so that it can pull data from the match api to help instruct the student.

User

The User class represents a user of our application. It stores all the users' identifying information, and is universal for both coaches and students. This class is referenced by all other classes because it holds on user data.

UserSkills

The UserSkills class stores the skills which users have specified that they are both good and bad at. This allows our matchmaking system to pair students with coaches who are best able to help them with the particular aspects of the game that they struggle most at. Skill is simply a list of possible skills to be categorized as strong or weak. Tied to each User and is referenced by the user class.

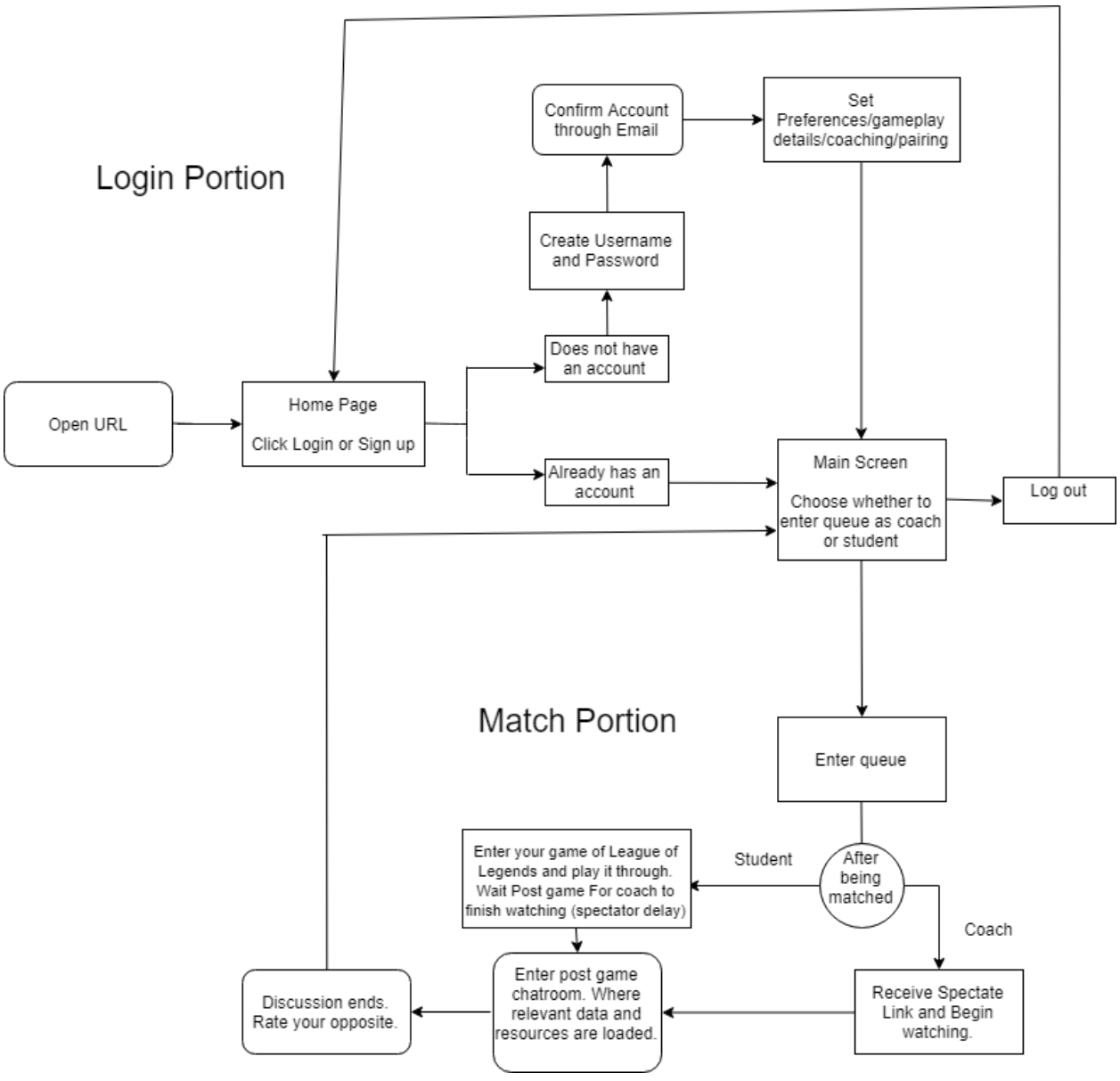
UserRatings

The UserRatings class stores a user's rating as a coach, allowing our matchmaking system to pair students with highly ranked, qualified coaches.

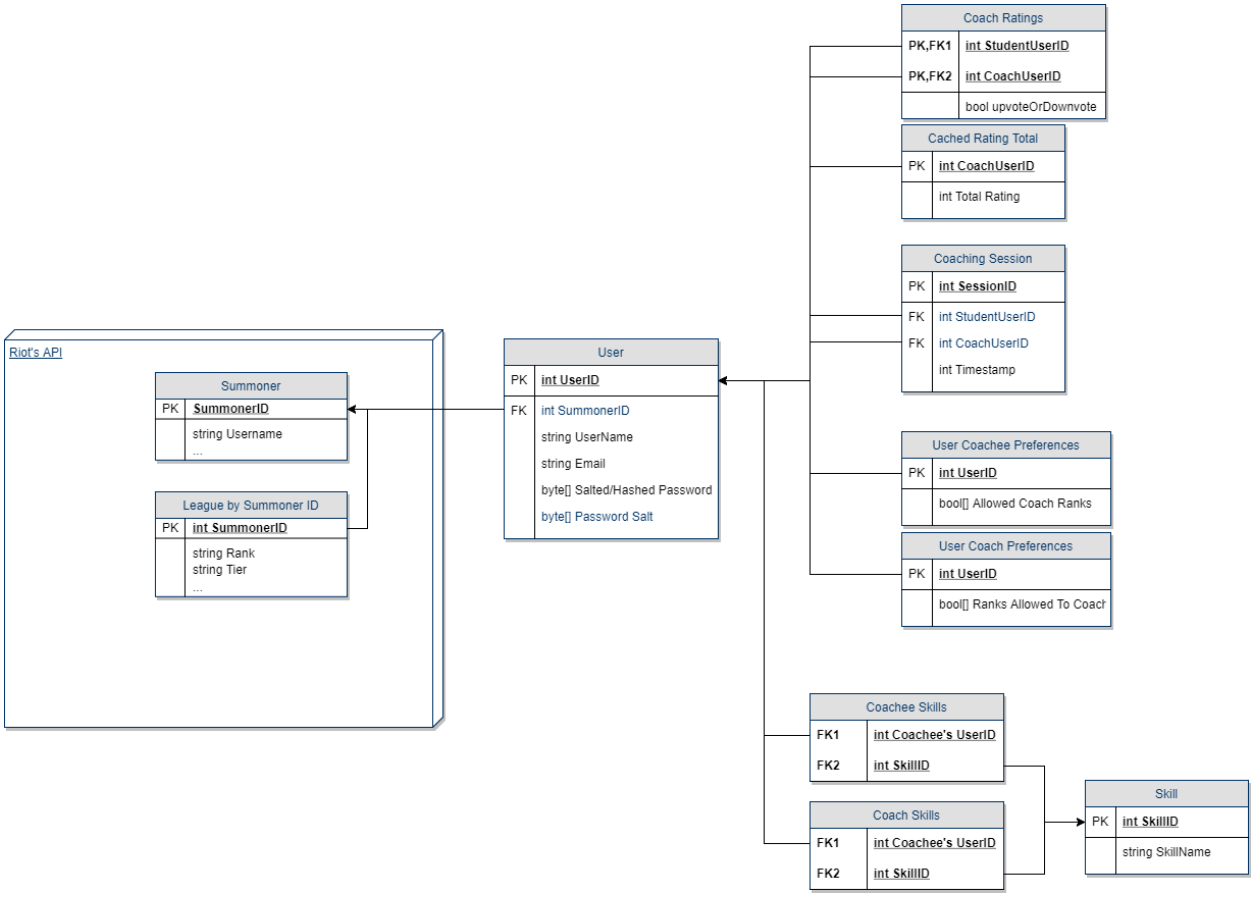
UserPreferences

The UserPreferences class allows users to specify which ranks of players that they would like to coach and be coached by, as well as allowing them to maintain a blacklist of users who they do not wish to coach or be coached by.

Frontend


Login Portion
Confirm Account through Email
Set Preferences/gameplay details/coaching/pairing
Create Username and Password
Does not have an account
Open URL
Home Page
Click Login or Sign up
Already has an account
Main Screen
Choose whether to enter queue as coach or student
Log out
Match Portion
Enter queue
Enter your game of League of Legends and play it through. Wait Post game For coach to finish watching (spectator delay)
Student
After being matched
Coach
Discussion ends. Rate your opposite.
Enter post game chatroom. Where relevant data and resources are loaded.
Receive Spectate Link and Begin watching.

Database


Coach Ratings
PK,FK1 int StudentUserID
PK,FK2 int CoachUserID
bool upvoteOrDownvote
Cached Rating Total
PK int CoachUserID
int Total Rating
Coaching Session
PK int SessionID
FK int StudentUserID
FK int CoachUserID
int Timestamp
Riot's API
Summoner
PK SummonerID
string Username
...
League by Summoner ID
PK int SummonerID
string Rank
string Tier
...
User
PK int UserID
FK int SummonerID
string UserName
string Email
byte[] Salted/Hashed Password
byte[] Password Salt
User Coachee Preferences
PK int UserID
bool[] Allowed Coach Ranks
User Coach Preferences
PK int UserID
bool[] Ranks Allowed To Coach
Coachee Skills
FK1 int Coachee's UserID
FK2 int SkillID
Coach Skills
FK1 int Coachee's UserID
FK2 int SkillID
Skill
PK int SkillID
string SkillName