LifePop

Overview: LifePop is a social media app designed to connect individuals with life coaches who fit they're criteria. The two primary users are clients and coaches. Coaches create business profiles outlining they're expertise, experience and availability. Clients can then view these profiles and select coaches who best fit their needs.

Architecture: A basic 3-layer architecture, front end: ionic framework, backend: Java Spring boot, database: MySQL. The backend used the following open source article https://www.baeldung.com/registration-with-spring-mvc-and-spring-security and codebase:https://github.com/Baeldung/spring-security-registration/tree/master/src/main/java/com/baeldung as a starting point

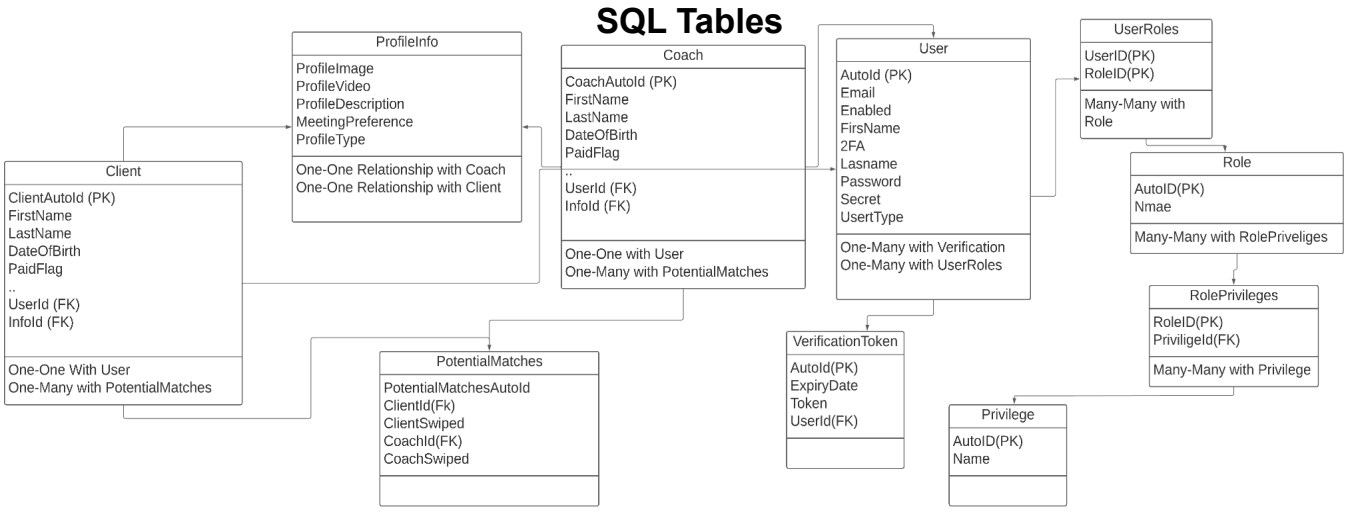
Backend Process Flow: The back end is dividing up into logical units consisting of a controller, a service, a model, a DTO and a repository. The frontend makes API calls to our controller, which will then call the appropriate service methods. Service methods hold most of the business logic. Models represent records in our SQL tables. Repository is the connection to the MySQL server allowing us to pull data.

Classes: The four primary logical units implemented are client, coach, registration and user.

- Client: This process is used to retrieve and update client profiles
- Coach: This process is used to retrieve and update coach profiles
- Registration: This process is used to sign users up. It implements expiring verification tokens and email confirmation.
- PotentialMatches: This process is used to record, update and check connections between clients and coaches
- User: This process holds login information for both clients and coaches

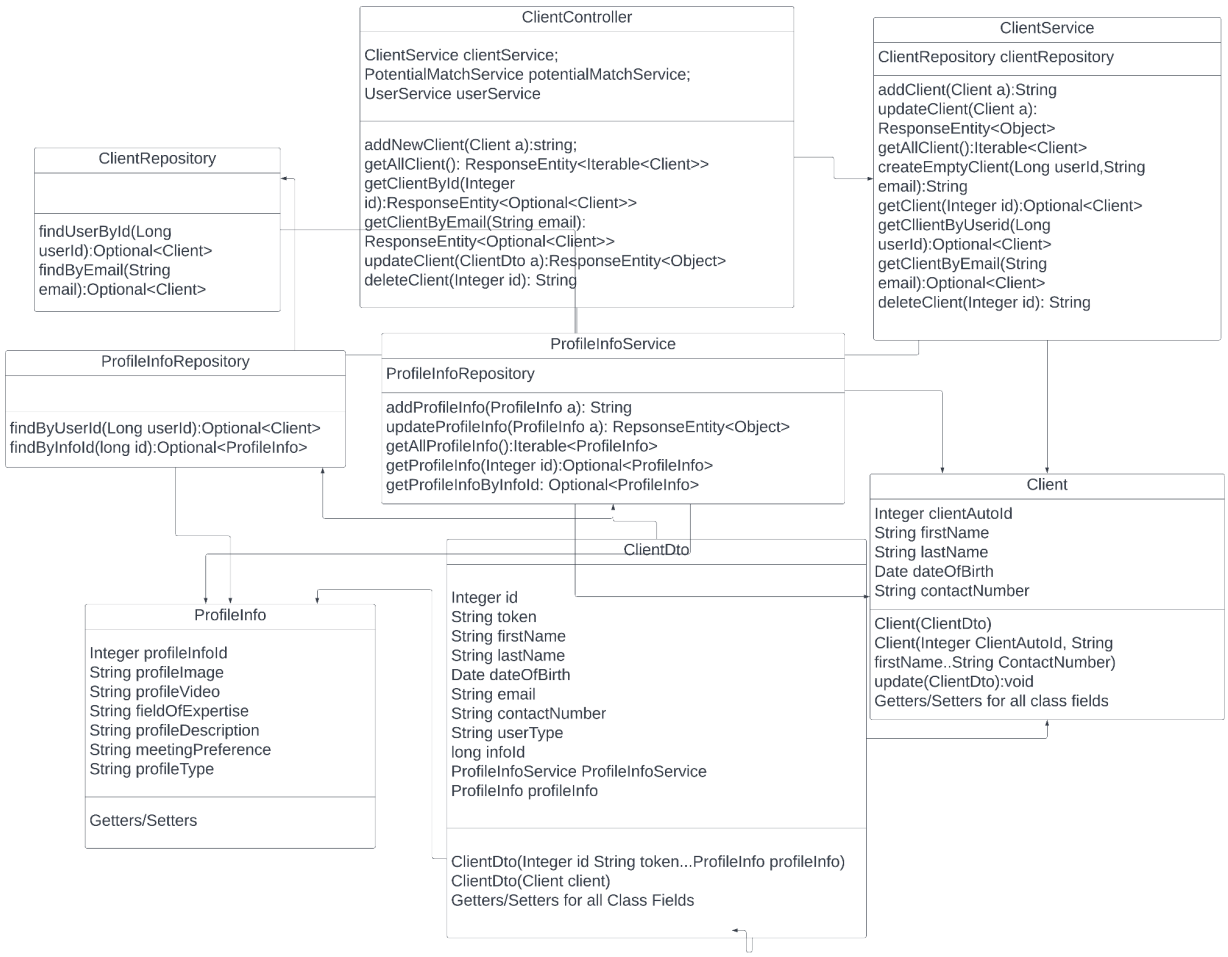
Unimplemented Classes: There are a number of classes and functionality existing that were not implemented, like location services, two factor authentication, etc. These are still in the project and can be repurposed in case

they’re needed down the line. You can find more information in the baeldung article referenced above. Messaging is handled by CometChat, a frontend api.



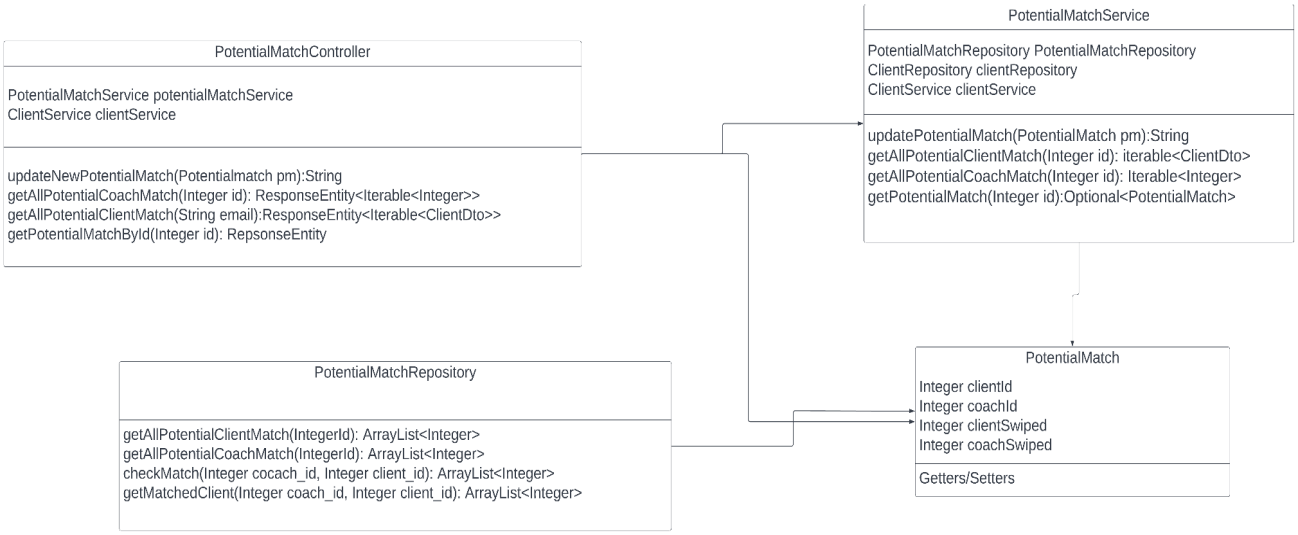
This diagram shows the relationship between our SQL tables. The two most important relationships are Coach and Client to PotentialMatches and Coach and Client to User.
PotentialMatches keeps a record of the connection between Coach and Client. This allows us to display clients who are interested in coaching to them and displays all available coaches to clients. The User table holds passwords and roles. This allows us to register users with the VerificationToken table via email and assign roles to admin users. Each role has specific privileges that can be customized for future functionality. If there are tables in the database not displayed in this diagram, their functionality is likely not implemented.

Client Class Diagram



This Class diagram shows one of the main workflows for a Client profile. The main functionalities implemented here are retrieving client profile with a UserId or email, adding a new client, deleting a client, making updates to client profiles. The ProfileInfo class serves as a supplement for extra profile information and holds picture and video urls.

PotentialMatches Class Diagram



PotentialMatches is a vital process flow. It ensures Clients only see Coaches they haven't swiped on and Coaches only see clients who have chosen them. The functionalities implemented allow you to get all Coach matches, all Client matches, get specific matches by id and update existing matches.

TODO: ProfileInfo is mostly implemented but not fully tested. The SQL tables need a revamp.There are no foreign key restraints implemented currently and column names aren’t following a uniform standard. Third party APIs and dependencies may be using outdated versions. Unit tests/QA. Image/video hosting, we explored using Amazon S3 or Firebase to hold users profile pictures and videos. Hosting we did not settle on a what service to host the site but were planning on using AWS for our MySQL database. App store requirements.