Design Document for CyHunt

Group 1_yn_4

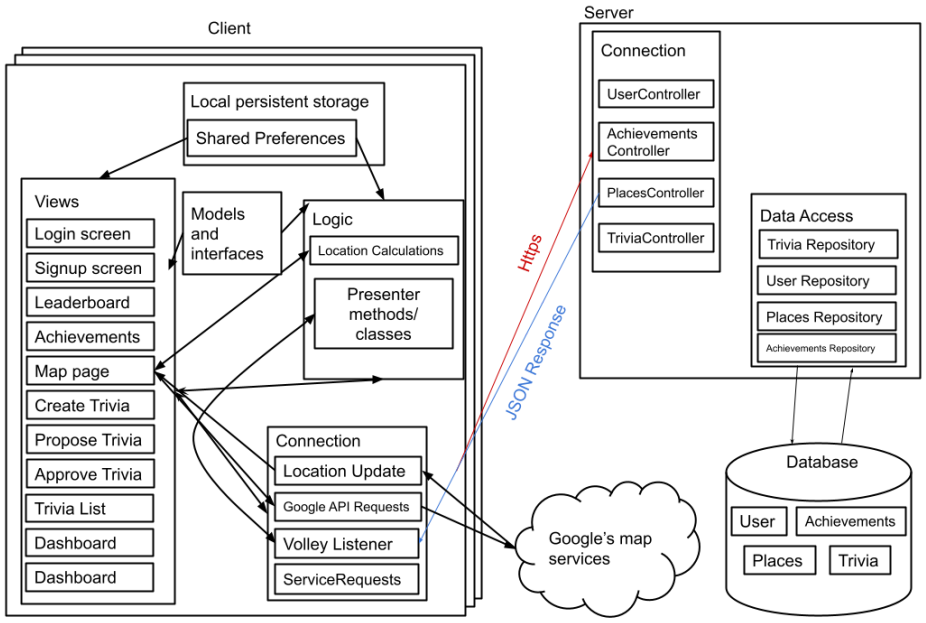
Member1 Name: Grace Wigen

Member2 Name: Lexi Ancona

Member3 Name: Erica Hollander

Member4 Name: Lakin Jenkins

PUT THE BLOCK DIAGRAM PICTURE ON THIS PAGE! (Create the picture using pencil or drawIO)



Use this third page to describe complex parts of your design.

Client: (Grace & Lexi)

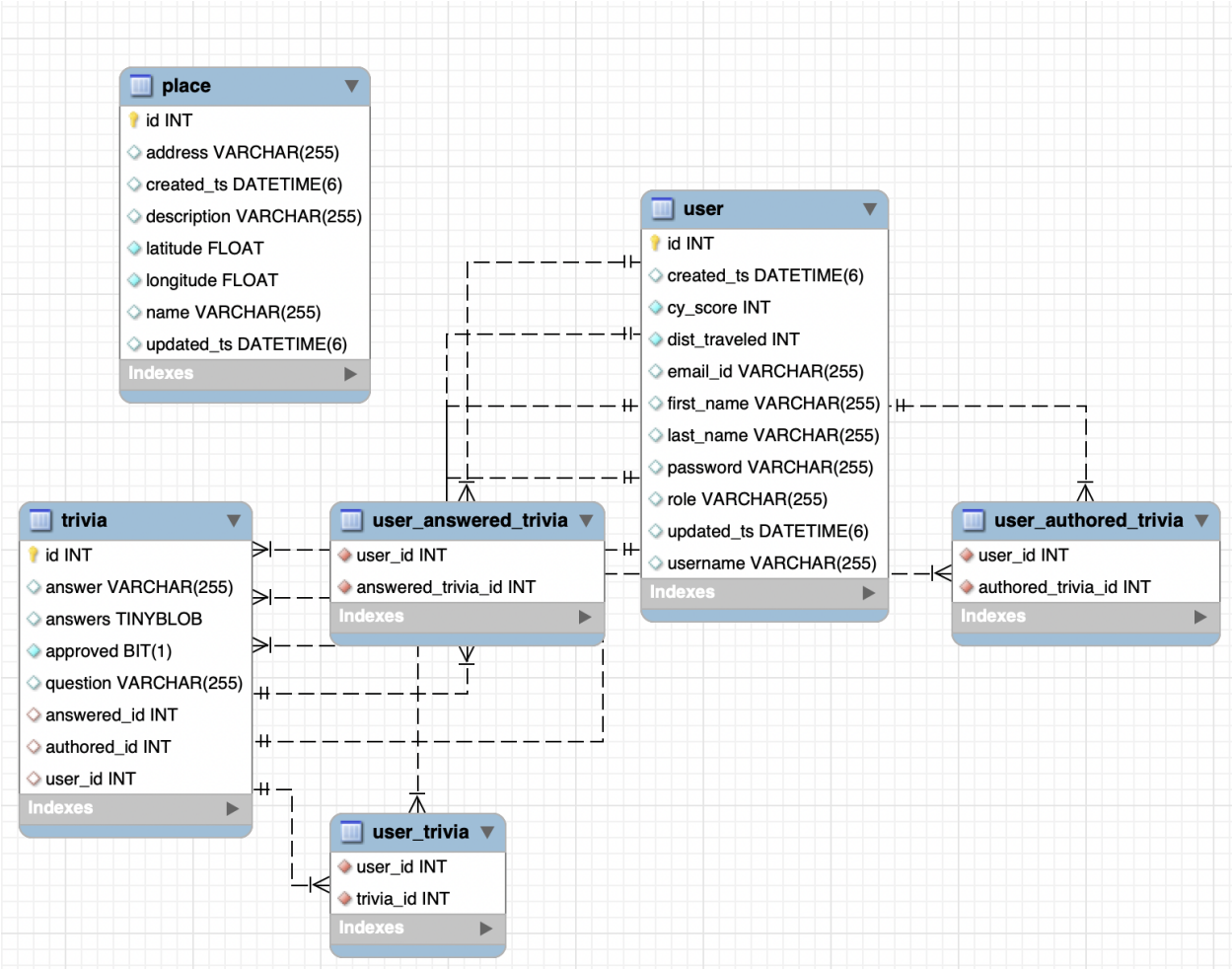
- Use of http requests
    - Use of multiple types of JSON requests to send and receive information to and from the server. In many cases, multiple types of requests are used in any given class, especially the login and signup screens.
- Shared preferences (local persistent storage)

    - We are using shared preferences to store a few key pieces of information about the user. When the user logs in, the app retrieves the user's ID, email ID, and role. This information doesn't change, so it's stored once and then used later when updating the user inside the database. This also allows the user to log in to the app once and bypass the login screen next time since the stored information persists after closing the app. Lastly, the app retrieves the user's score and continuously updates the value inside shared preferences so that the correct value can be displayed to the user.
- Dependency injection
    - Many classes are split up between view, presenter, and model classes. The views are enumerated in the block diagram, but the presenter classes are just given one block. They do the heavy lifting of the program, most notably sending http requests, which are then received by the views. The models exist for both views and presenters to allow for dependency injection. Many views extend interfaces with methods that let them receive returned data from the database in a usable form, after the presenters send and process the data.

Server: (Erica & Lakin)

- Map API Requests
    - One of the more complicated parts of the system is our map and its API calls to Google Maps. In order for our location based game to work, our system needs to monitor and update the user's location as they move locations. Our system is currently set up to use asynchronous calls so when the user's location changes, the map updates according to their new coordinates by using the google maps api. This allows for the app to be location based and for the user to see their progress by viewing a map that shows their progress in an easy to understand manner.
- Spring Boot
    - Spring Boot is the microservice that helps what could be a really complicated backend, a much simpler thing. Spring Boot helps the backend communicate with our database by creating the database, watching for changes, reject entries that have the same primary key as another entry (great for login/sign up features), and updating our database's entries. Tags like JSON ignore make it so it's easier to read our entries in postman or client side.

PUT THE TABLE RELATIONSHIPS DIAGRAM on this fourth page! (Create the picture using MySQLWorkbench)



Notes from this diagram:
We will be in our final sprint, adding ‘last place’ to user, which helps bring in a relationship between the user and place data tables. In addition, there is one more table, Achievements, with a many to many relationship to Users. We wanted main functionality before adding the achievements, but in our last and final iteration of the project those additions will be made.