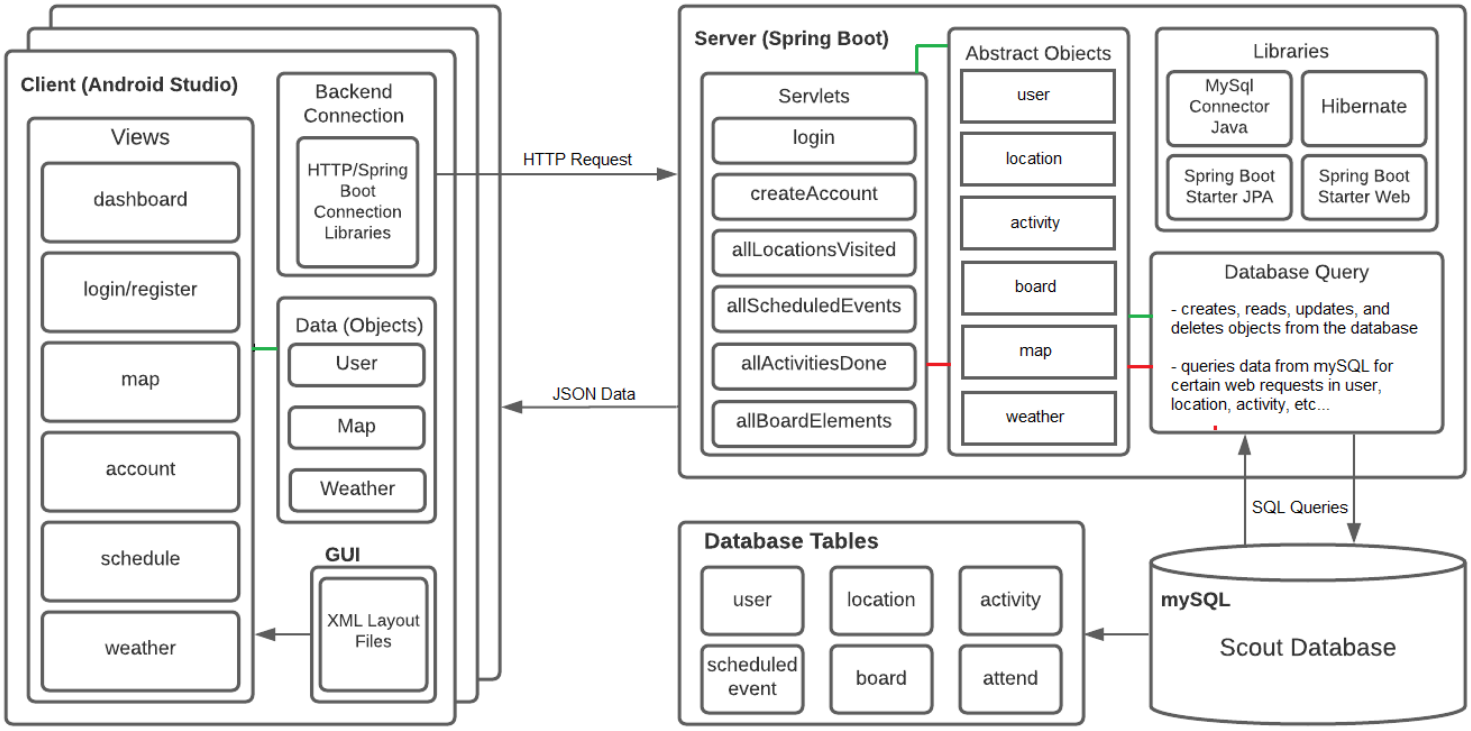
Design Document for *Scout*

Group **2_DO_1**

Haylee Lawrence 25% contribution
MyTien Kien 25% contribution
Sanjana Amatya 25% contribution
Britney Yu 25% contribution

Block Diagram



*there are colored lines because I had to finish this in Microsoft Paint

- Red line means arrow is facing left
- Green line mean arrow is facing right

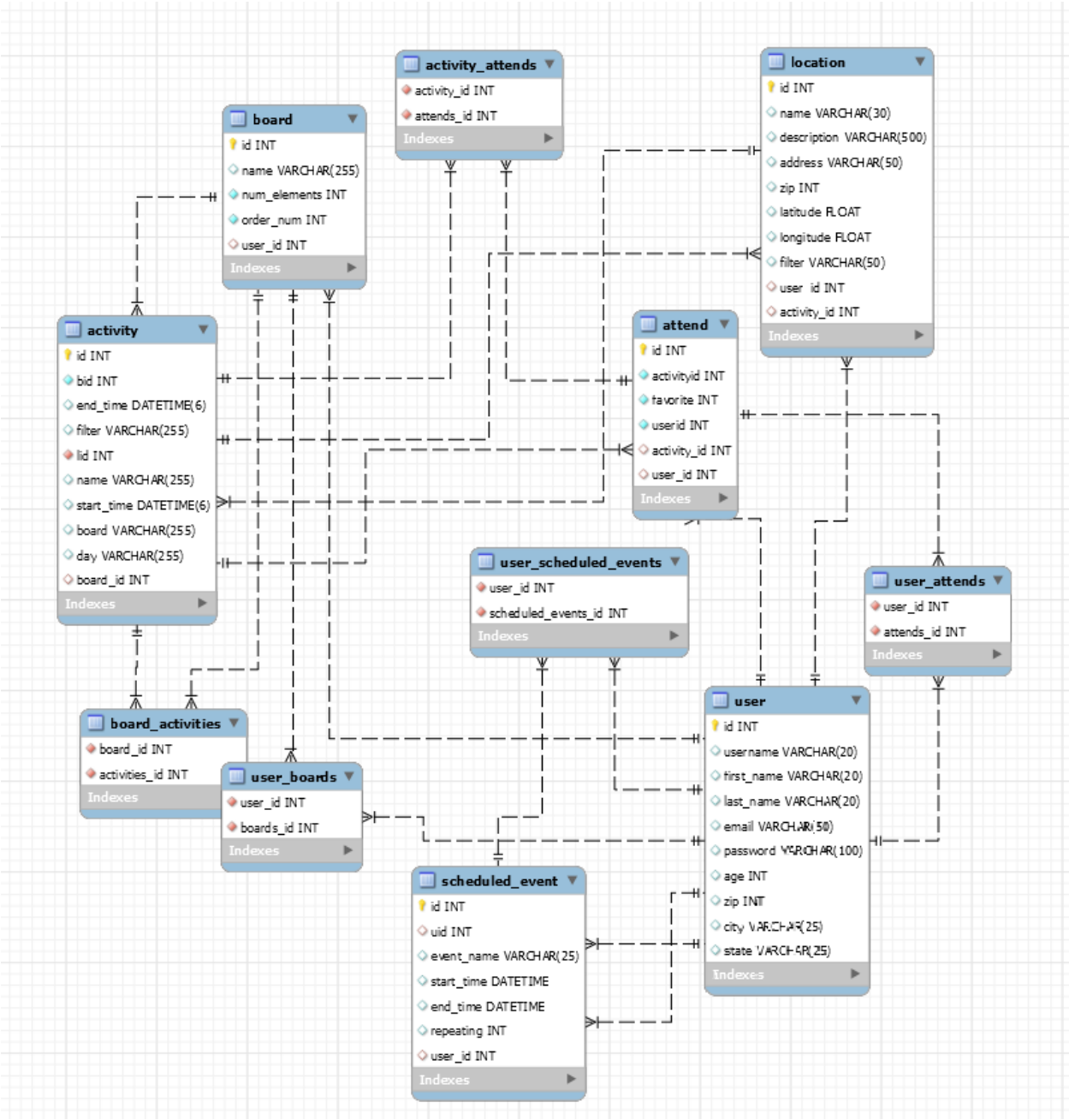
Complex Details

Client:
On the client side, we have the views: login, dashboard, map, weather, schedule, and account. Our application's frontend consists solely of Android clients therefore we use a combination of XML and Java classes to communicate between the application's GUI. Since the login.xml and signup.xml are more data driven, the application needs code written on the frontend java classes to retrieve data from the backend databases to display username and account information from the server. The Maps and Weather view utilize Google Maps and Weather APIs that allow a user to view their certain locations and the weather from their current location to find activities around the area. Any other views like the schedule.xml allows a user to view tasks and personalize their own schedule with certain activities.

Server:
Our application will pass a user's location to our server. The data obtained from the user's current location will notify the server to return place objects near that location. The place objects will fetch the specific information such as location, start time, end time, location, etc, to put into an activity card. We will develop code to determine places with the shortest distance to the user's current location. To change the database we use abstract objects. Abstract objects are representations of the data stored on the database tables and are created through the java objects. This allows the abstract objects to change the database through MySQL queries.

Model:
Our app will integrate the Maps SDK for it's server side processing(Google Maps Engine). Google Maps allows us to view the google maps screen to help show users their current location. The use of this API proves to be useful because it provides us with a map view and the necessary details of the surrounding locations. For the backend communication, we will have threads running in the background to track where the user is and generate activity cards based on that.

Table and Relationships Diagram



Description of Database Schema

Core Tables:

- **user -** This table represents an individual user that exists in the database of our mobile application (AKA, this user is registered for our application)
    - **id -** this is the special unique ID that identifies the individual user
    - **username -** the username of the user
    - **first_name -** the first name of the user
    - **last_name -** the last name of the user
    - **email -** the user's email, user for login and forgot password
    - **password -** the password for user to login, always encrypted
    - **age -** user's age, this is to filter activities and location based on age
    - **zip -** the user's zip code based on location
    - **city -** the user's current city
    - **state -** the user's current state

- **location -** This table represents one individual location for our application based on the city the user is in (for now, the user will only be located in Ames). Examples include Ada Hayden Park, or Iowa State University, any locations that are near the current user's location in Ames.
    - **id -** the special, unique ID that identifies the individual location
    - **name -** the name of the said location
    - **description -** the description of the location, what it's known for, etc...
    - **address - the** address of specific location
    - **zip -** the location's zip code
    - **latitude -** for GPS authentication, the location's latitude
    - **longitude -** for GPS authentication, the locations' longitude

- **activity -** This table represents a certain activity a user can do or attend at a certain location. Each location has at least 0 or more activities to do. Examples may include a charity event, club social and so on.
    - **id -** the special, unique ID that identifies an individual activity
    - **lid** - the location ID that this activity is linked to, each each activity has at least one location
    - **name -** the name of the said activity
    - **start_time -** the time and day the activity is said to start
    - **end_time -** the time and day the activity is said to end

- **scheduled_event -** This table represents an individual event on the user's schedule that cannot be changed. This is to represent the times the user is not available to do anything, so the application can sort events based on the user's free time. Examples include class lectures, doctor appointments and so on.
    - **id -** the specific and unique ID that identifies an individual and specific scheduled event

    - **uid -** the user ID that this scheduled_event is linked to, a user can have 0 or more scheduled_events
    - **event_name -** the name of the scheduled event
    - **start_time -** the time and day this event is scheduled to start
    - **end_time -** the time and day this even is scheduled to end
    - **repeating -** tells us if this certain even is a repeating event or not

- **board -** An individual board represents a “folder” of activities that the user can organize their saved or favorite future/past activities in. For example, a board can be named “parks” to organize all the parks the user wants to go to in the future.
    - **id -** the special and unique ID that identifies a specific individual board object
    - **uid -** the user’s ID that this board is linked to, a user can have 0 or more boards
    - **name -** the name of the board
    - **num_elements -** the number of activities in a board
    - **order_num -** a number to help keep track of where the board is on the page (1 means the board is the first board shown).

- **attend -** This table represents an activity a user is planning to attend. This helps link the tables “user” and “activity”
    - **id -** the special and unique ID for a specific individual attend object
    - **userID -** the user ID that is linked to the attend object, an attend object but have 1 user
    - **activityID -** the activity ID that the user is planning on attending, each attend object must have 1 activity ID
    - **favorite -** whether or not the user is interested or saved this certain activity

Relationship Tables: These tables were made when putting in relationship mapping in the Spring Boot backend. We did not initially have these tables until relationship mapping was put in. Only tables create were those that had one to many relationships

Those Listed In Diagram

- **activity_attends:** links object activity to attend, and attend to activity
    - activity → attend: one to many
    - attend → activity: many to one
- **board_activities:** links board to activity, and activity to board
    - board → activity: one to many
    - activity → board: many to one
- **user_attends:** links user to attend, and attend to user\
    - user → attend: one to many
    - attend → user: many to one

- **user_boards:** links user to board, and board to user
    - user → board: one to many
    - board → user: many to one
- **user_scheduled_events:** links user to scheduled_event, and scheduled_event to user
    - user → scheduled_event: one to many
    - scheduled_event → user: many to one

Those Not Listed In Diagram

- **location_activity:** this links location to activity, and activity to location
    - location -> activity: one to one
    - activity -> location: one to one