Design Document for GameOn

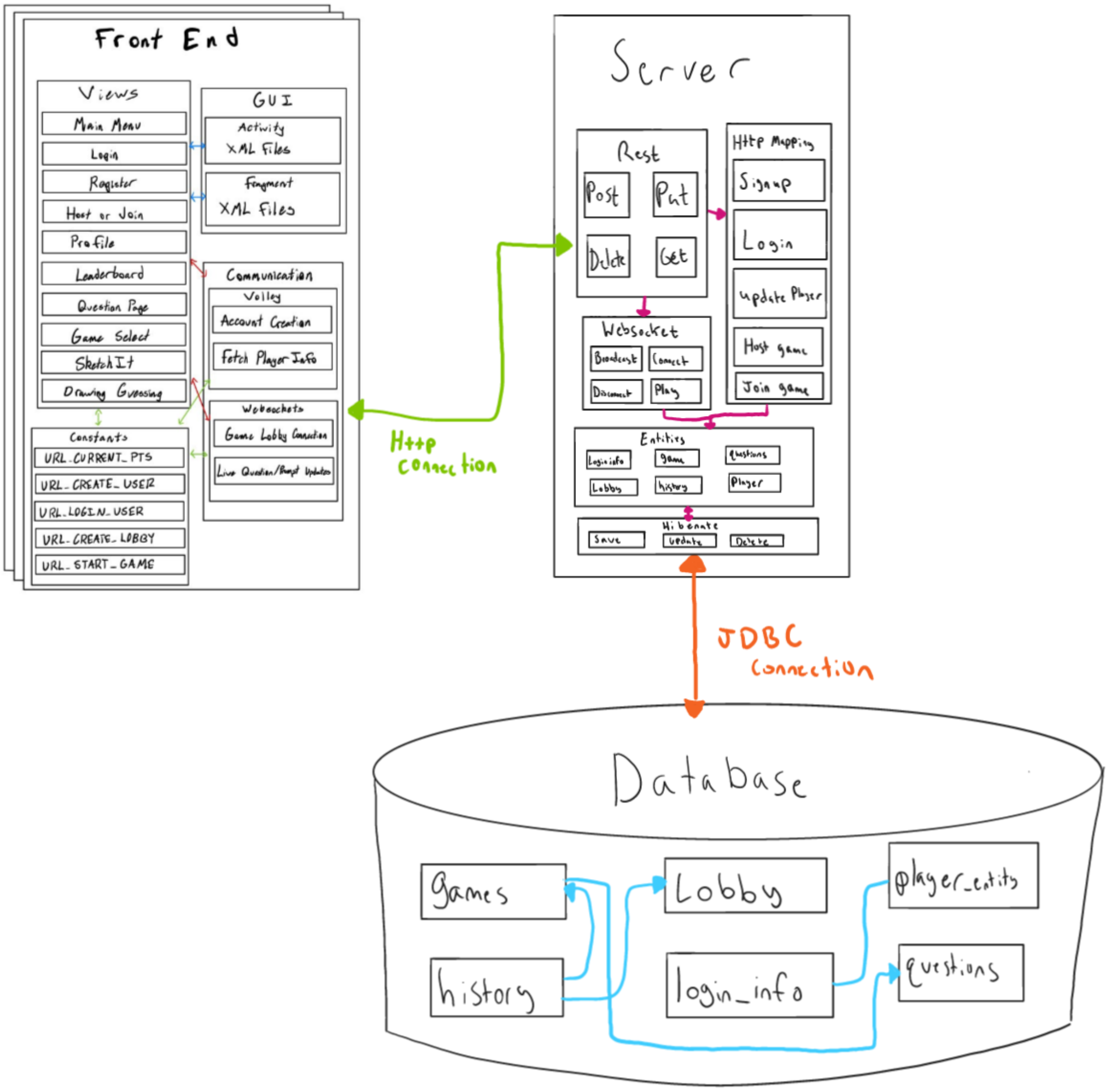
Group <1_HB_3>

Ahmed Nasereddin: 25% contribution

Ethan Douglass: 25% contribution

Tayler Barnhart: 25% contribution

Garrett Arp: 25% contribution


Front End
Views
Main Menu
Login
Register
Host or Join
Profile
Leaderboard
Question Page
Game Select
SketchIt
Drawing Guessing
GUI
Activity XML Files
Fragment XML Files
Communication
Volley
Account Creation
Fetch Player Info
Websockets
Game Lobby Connection
Live Question/Prompt Updates
Constants
URL_CURRENT_PTS
URL_CREATE_USER
URL_LOGIN_USER
URL_CREATE_LOBBY
URL_START_GAME
Http connection
Server
Rest
Post
Put
Delete
Get
Http Mapping
Signup
Login
update Player
Host game
Join game
Websocket
Broadcast
Connect
Disconnect
Play
Entities
Login info
game
questions
Lobby
history
Player
Hibernate
Save
Update
Delete
JDBC connection
Database
games
Lobby
player_entity
history
login_info
questions

Frontend:

The frontend designed and implemented their code in a few different ways. We took full advantage of both activities to run the larger processes, as well as fragments to show quick information that could be used across multiple activities, or to show multiple items in rapid succession (i.e. error messages, changing photos, displaying drawings). We also took advantage of the many different types of layouts, such as constraint, linear, frame, and scrolling to best fit the needs of each specific page. For saving the client's account information, we used a static class that can be called to retrieve that player's info as needed.

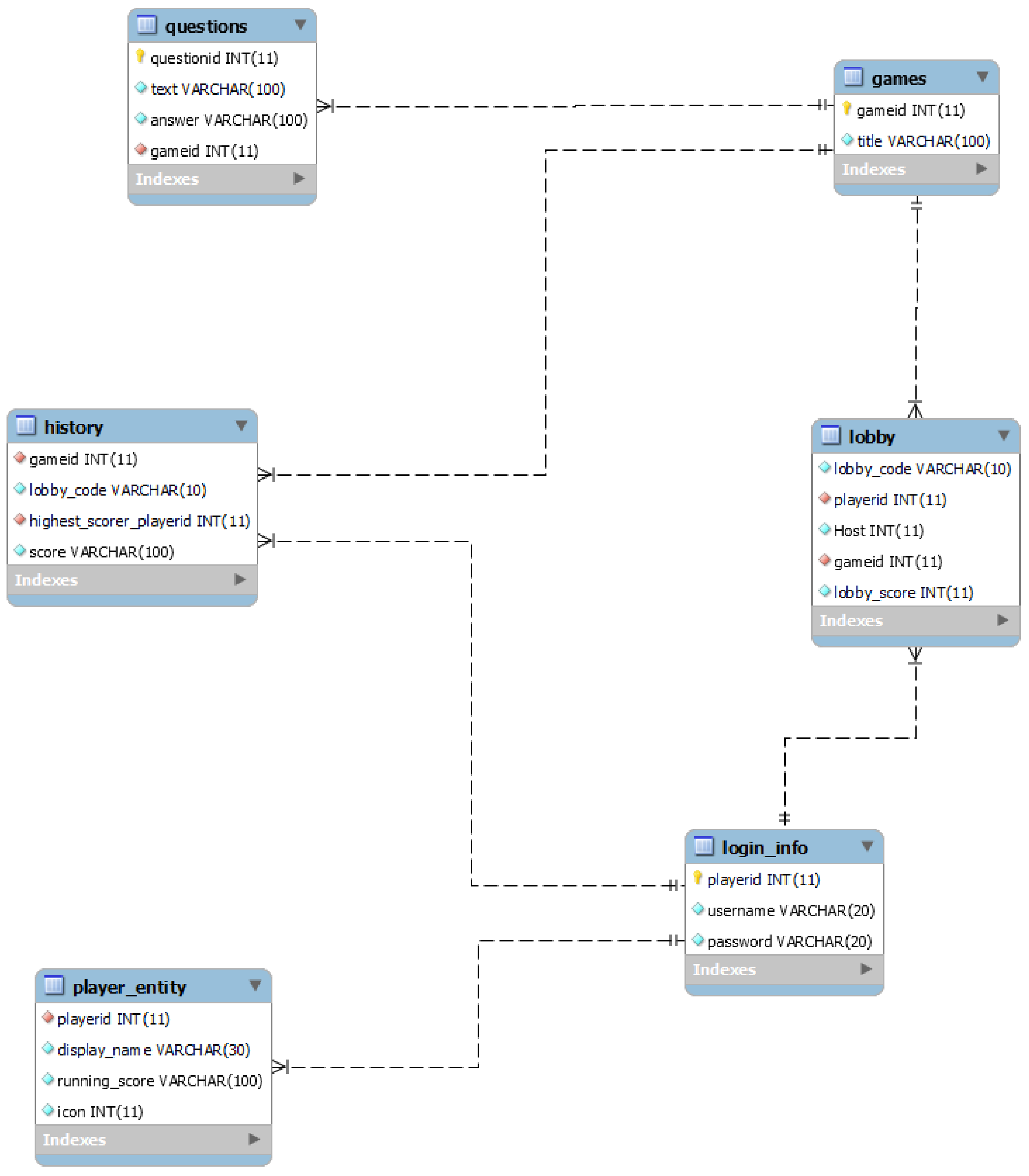
With regards to communication, we used 1 overviewing main point for both Volley and Websocket communication. For each of these, we made a separate class with generic methods that would then be called by activities in order to keep all methods localized to 1 location. This allows us to open only 1 connection to the database and server for each communication type, instead of opening a new connection for each instance, thus preventing potential overflow issues.

Backend:

The backend will consume JSON objects through a HTTP connection and then decide whether it is handled by a request mapping or needs to be upgraded to a websocket. The controller will update or create the neccessary table records with the infromation from the JSON and then return a HTTP Response code and JSON object that includes a server response and the requested information. When upgraded to a websocket, the client is identified as either a host or player and has permissions as such. This decides what controls and actions they have over the lobby. Communication between clients is handled through broadcasting JSON string messages to other clients in the same lobby.

Database:

The database uses hibernate and the JDBC connector to connect to MySQL. It contains information for player entities, game scores, history of games, login info and questions to pull from. Our database also has a table that holds temporary data such as Lobby in which records are deleted when the lobby closes. This table allows the server to identify which playerids are connected to which lobby codes and what game they are currently playing. The tables include one-to-many, many-to-many, and one-to-one relationships. Like other groups, we utilize MySQL and MariaDB for our database, this is because it was easy to implement as well as easy to set up a client-server relationship.


questions
questionid INT(11)
text VARCHAR(100)
answer VARCHAR(100)
gameid INT(11)
Indexes
games
gameid INT(11)
title VARCHAR(100)
Indexes
history
gameid INT(11)
lobby_code VARCHAR(10)
highest_scorer_playerid INT(11)
score VARCHAR(100)
Indexes
lobby
lobby_code VARCHAR(10)
playerid INT(11)
Host INT(11)
gameid INT(11)
lobby_score INT(11)
Indexes
login_info
playerid INT(11)
username VARCHAR(20)
password VARCHAR(20)
Indexes
player_entity
playerid INT(11)
display_name VARCHAR(30)
running_score VARCHAR(100)
icon INT(11)
Indexes