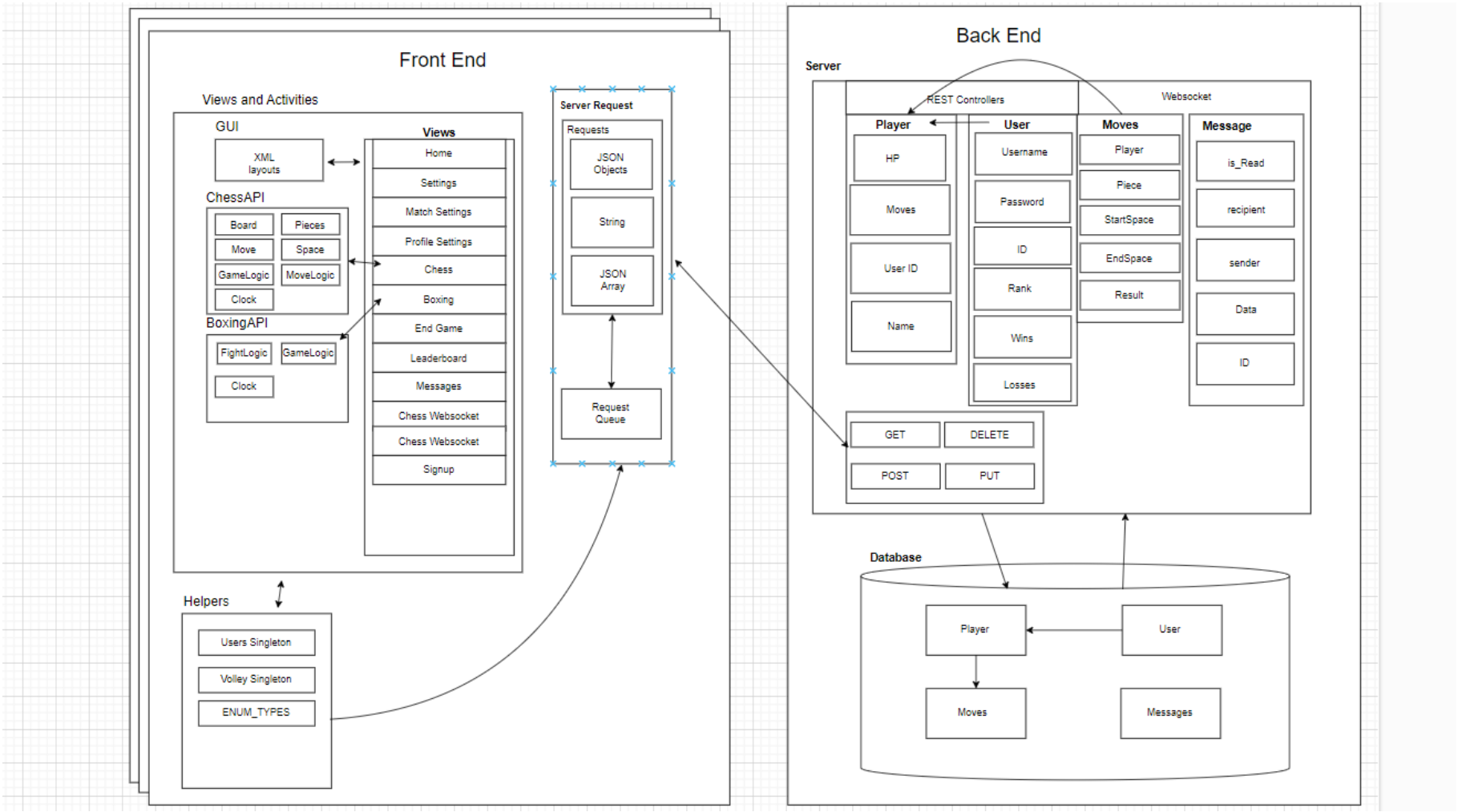
Design Document for Hooks and Rooks

Group <JK-221>

Member1 Lincoln: 33% contribution

Member2 Josh: 33% contribution

Member3 Ty: 33% contribution


Front End
Views and Activities
GUI
XML layouts
ChessAPI
Board
Pieces
Move
Space
GameLogic
MoveLogic
Clock
BoxingAPI
FightLogic
GameLogic
Clock
Views
Home
Settings
Match Settings
Profile Settings
Chess
Boxing
End Game
Leaderboard
Messages
Chess Websocket
Chess Websocket
Signup
Server Request
Requests
JSON Objects
String
JSON Array
Request Queue
Helpers
Users Singleton
Volley Singleton
ENUM_TYPES
Back End
Server
REST Controllers
Websocket
Player
HP
Moves
User ID
Name
User
Username
Password
ID
Rank
Wins
Losses
Moves
Player
Piece
StartSpace
EndSpace
Result
Message
is_Read
recipient
sender
Data
ID
GET
DELETE
POST
PUT
Database
Player
User
Moves
Messages

Front End(currently implemented)

Registration Screen

- The registration screen sends an HTTP Post request to the backend that creates a User. The screen contains the following elements.
- EditText: Username
- EditText: Password
- Button: Sign Up

Login Screen

- Sends an HTTP GET request to the backend that checks for a matching User. If you don't have an account yet takes you to a different page where you can make one.
- EditText: Username
- EditText: Password
- Button: Login
- Button: Sign Up

Settings Screen

- Settings currently send a Delete HTTP request to the backend that deletes the logged-in user from the database.
- Button: Delete User

Chess Screen

- The chess screen is the UI for the chess aspect of the game. Contains the board, chess pieces, and a button for sending moves through a web socket.
- 2D Button Array: Chessboard
- Button: Send Move
- Image Views: Pieces
- Chess API: Outside of the main activity, this screen uses a lot of chess logic.
    - Board class, Space class, Piece abstract class, Move class, ENUM_TYPES Enum class
    - Pawn, King, Queen, Rook, Bishop, and Knight classes

Home Screen

- The starting screen when you open the app. Takes you to the Log In screen if you haven't yet. Access to the game screens, leaderboard, and settings from this screen.
- Image View: Logo
- Button: Settings
- Button: Play
- Button: Leaderboard

Back End

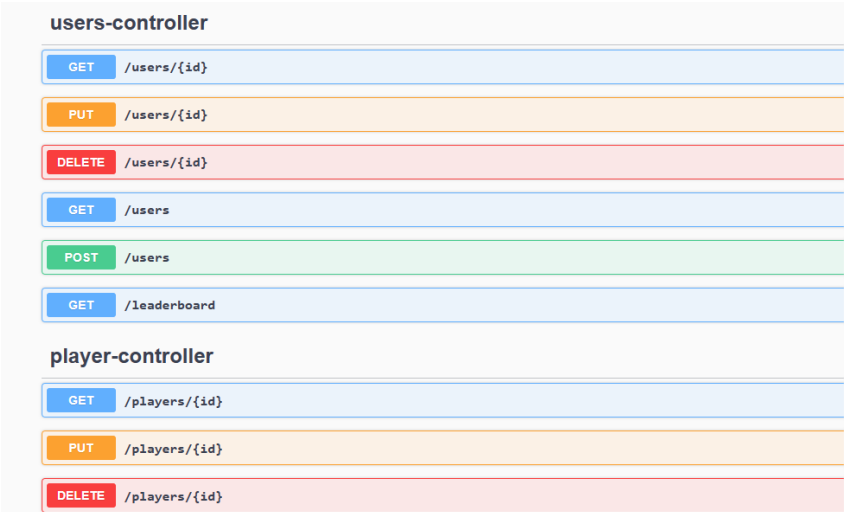
Communication

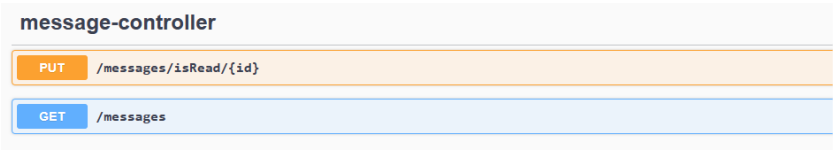
POST: Creates and sends a new object to the database and gives it a unique id.
GET: Requests the information of a certain object belonging to an id or an array of objects.
DELETE: Sends an id to be deleted from the database.
PUT: Sends an id to be updated or changed in the database.

Controllers

- UsersController: Contains a POST mappings to create users with a unique id, username, rank, and password. It also contains how many wins and losses a user has. A client could retrieve with a GET request all users, a specific user by ID, or a sorted list of users by their amount of wins, which is the leaderboard. A client could also update these User values with PUT requests. Clients are also able to remove users from the database with a DELETE request.
- PlayerController: This controller is used to create game-specific instances of players with the information needed for gameplay. It includes a unique ID, Name, HP, a list of moves played, and has a one-to-one mapping with a User. Clients can use GET requests to get a complete list of past players or a specific player by ID. A POST request will create a player with the given request body, and PUT requests can update the Player objects during gameplay. Clients are also able to use DELETE mappings to delete specific players in the database by ID.
- MessageController: This controller can be used to retrieve a complete history of messages in the chatroom and to change messages to read in the database. A GET request returns all prior messages saved in the database, and a PUT request to a specific message ID changes the messages isRead state to true.

SwaggerUI Screenshots





SwaggerUI Link:http://coms-309-015.class.las.iastate.edu:8080/swagger-ui/index.html#/

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