Joseph Lister
SWDV 691
Assignment 2
March 23, 2019

Service Layer Design

The database program used will be MongoDB. NodeJS will use the Express framework to implement a REST API for communication between NodeJS and MongoDB.

Sign Up page

- If the user clicks the `Sign Up` button, the `Password` and `Confirm password` fields must match. The `Username` and `Date of birth` field must not be empty.
  - If any of the above is not the case, then a message will be displayed underneath the respective field stating the issue.
  - If all fields are valid then a `GET` request will be used to invoke MongoDB's `findOne` function on the `accounts` collection. A query of the following form will be passed to the function: `{"username" : <string>}`.
    - If `findOne` returns `null` then an `account` document will be created with default values (besides `username`, `age`, and `password`). A `POST` request will then be automatically used to invoke MongoDB's `insertOne` function to insert the new document into the collection, at which point the user will be redirected to the `Game Selection` page.
    - If `findOne` returns a document, then a message must show up saying "This username is not available".

Login page

- If the user clicks the `Login` button, the `Username` and `Password` fields must not be empty.
  - If any of these fields are empty then a message will be displayed underneath the empty field stating that the respective field may not be empty.
  - If both fields are not empty then a `GET` request will be used to invoke `accounts.findOne`. A query of the following form will be passed to the function: `{"username" : <string>, "password" : <string>}`.

- If `null` is returned then a message will show up saying “invalid username or password”.
- If a document is returned then a `PUT` request will be used to invoke `accounts.findOneAndUpdate` to set `onlineStatus` to `true`. The username and birth date will be saved in the client (`birthDate` will be used for gaining access to age-restricted data, and `username` will be used in a variety of queries). If the Finally the user will be redirected to the “Game Selection” page.

Account Page (self)

- When the `accountSelf` page is accessed, a `GET` request will be used to invoke `accounts.findOne({“username” : <string>})` which will return the `account` document associated with the user. This document’s data will be used to populate the page.
- If the user clicks the `change mantra` button then the tag holding the mantra will be replaced with a text field. Upon pressing enter on the text field a `PUT` request will be used to invoke `accounts.updateOne({“username” : <string>, “mantra” : <string>})`. The text field will then be replaced with a paragraph tag holding the new mantra.

Profile Image Selection page

- When the `profileImageSelection` page is accessed, a `GET` request will be used to invoke `profileImages.find()` which will return a set of all available `profileImage` documents. This set of documents will be used to populate the page.
- When the user selects an image, a `PUT` request will be used to invoke `accounts.updateOne({“username” : <string>, $set: {profileImage : <BinData>})`, which will change the user’s profile image.

Account Page (other)

- When the `accountSelf` page is accessed, a `GET` request will be used to invoke `accounts.findOne({“username” : <string>})` which will return the `account` document associated with the user. This document’s data will be used to populate the page.
- When the `accountSelf` page is accessed, a `GET` request will be used to invoke `accounts.findOne({“username” : <string>, “friends” : <string> })` to check if the target is listed as a friend of the user.
  - If the target is a friend of the user, then the `friendship` button will say “delete friend”. Otherwise it will say “delete friend”
- When the `accountSelf` page is accessed, a `GET` request will be used to invoke `accounts.findOne({“username” : <string>, “blocked” : <string> })` to check if the target is listed as blocked to the user.

    - If the target is blocked by the user, then the `block` button will say “unblock”. Otherwise it will say “block”
- Upon pressing the `friendship` button,
    - If target is not already a friend, then a `POST` request will be used to invoke `account.insertOne({“username” : <string>, “friends” : <string> })` to insert the target-user’s username into the `friends` array field of the account associated with the user’s username.
    - If the target is a friend of the user, then a `DELETE` request will be used to invoke `account.remove({“username” : <string>, “friends” : <string> }, {justOne : true})` to remove the target as a friend of the user.
- Upon pressing the `block` button,
    - If target is not already blocked, then a `POST` request will be used to invoke `account.insertOne({“username” : <string>, “blocked” : <string> })` to insert the target-user’s username into the `blocked` array field of the account associated with the user’s username.
    - If the target is a friend of the user, then a `DELETE` request will be used to invoke `account.remove({“username” : <string>, “friends” : <string> }, {justOne : true})` to remove the target as a friend of the user.
- If the target user is offline or if the user is blocked by the target user, then the `challenge` button will be deactivated. If the button is clicked on while it is active then a `GET` request will be used to invoke `accounts.findOne({“username” : <string>, “challenger” : { $type: 10 }})` which will either return `null` or a document.
    - If `null` is returned, then the user has not been challenged yet and a new challenge may be sent. A `PUT` request will be used to invoke `accounts.updateOne( {“username” : <target username>, “challenger” : <username>})` to set the user as the target’s challenger. Challenging an opponent will immediately forfeit the user from any existing games, and remove any other pending challenges. A `DELETE` request will be used to invoke `multiplayerGames.remove({$or : [{“player1”, <username>}, {“player2” : <username>}]})` as well as `accounts.updateOne( {“challenger” : <username>}, $set : {“challenger” : null})`. If the player was removed from a current game then his losses must go up by one and his rank must be updated. The user will then be immediately redirected to the `multiplayerGame` page.
    - If a document is returned then an alert will pop up saying “The user already has a pending challenge”.

<u>Single Player Game page</u>

- When the `singlePlayerGame` page is accessed, a `GET` request will be used to invoke `singlePlayerGames.findOne({"username" : <string>})` which will either return `null` or a document.
  - If `null` is returned then that means the user doesn't currently have a single player game in session and a new one must be created. A game of Battleships must be randomly initialized at this point. Two grids will be generated, one for the user and one for the computer. Each grid will simply be represented by an array of 100 integers; 0 represents an empty cell, 1 represents a healthy section of a ship, 2 represents a hit section of a ship, and 3 represents a hit empty cell. A `singlePlayerGame` document will now be generated using the user's username and the two grids. A `POST` request will be used to invoke `singlePlayerGames.insertOne` to insert the generated document into the collection.
  - If a document is returned then the document's `playerGrid` and `cpuGrid` fields will be used to populate the game grids. The `gameStep` will be set to 3.
- `Every turn`, if the gameStep is 3, the updated state of both grids must be used to generate a query. A `PUT` request will then be used to invoke `singlePlayerGames.updateOne({"username" : <string>}, {$set : query})`.
- Once the game is over, a `DELETE` request must be used to invoke `singlePlayerGames.remove({"username" : <string>}, {justOne : true)}`.

Multiplayer Game page

- When the `multiplayerGame` page is accessed, a `GET` request will be used to invoke `multiplayerGames.find({$or : [{"player1" : <username>}, {"player2" : <username>}]})` to check which games the user is currently a part of. The function can return up to two games: one recent game and one challenger game.
  - If the function returns a set of two documents, then the recent game will have priority over the challenger game. Only the challenger game can (and always will) have a `gameStep` value of `1`, so the document whose `gameStep` is not `1` will be used to populate the page.
  - If the function returns a set with only one document then that document will be used to populate the page. If at this point, the `gameStep` is `1` then that means the player just entered a challenger game, and his account's challenger field must be set to `null`. A `PUT` request will invoke `accounts.updateOne({"username" : <username>}, $set : {"challenger" : null})`. HERE
  - If the function returns an empty set then the `multiplayerGames` collection will check for a game without a `player2`. A `GET` request will be used to invoke `multiplayerGames.findOneAndUpdate({"player2" : null}, $set: {"player2", <username>})`. If the function returns a document then the document will be used to populate the page. Otherwise a new `multiplayerGame` document will

be generated with the user as `player1`, `null` as `player2`, and default values for the rest of the fields. A `PUT` request will be used to invoke `multiplayerGames.insertOne(document)`. HERE

- At this point, both players must already be in the game session and the game must be initialized for both players. If the `gameStep` is `1` then the `gameStep` must be set to `2`. A PUT request will be used to invoke `multiplayerGames.updateOne({$or: [{"player1" : <player1 username>, "player2" : <player2 username>}, {"player1" : <player2 username>, "player2": <player1 username>}]}, $set : {"gameStep" : 2})`. This request should not be sent from both clients, so it will only be sent from player1.
- If the gameStep is 2, then update the gameStep to 3 when both players are ready
- Every turn, each player sends their updated grid (either when time is up or when the button is pressed). Change the player's ready status to true
- When both players are ready load the opponent's updated grid and change both players ready status to false
- Achievement related counters must be constantly updated
- Pressing forfeit will trigger the game to move into step 4
- On step 4 each player's game statistics will be updated and the game will be removed from the collection
- Every second the turnTimer must be updated. After each turn the turnTimer must be reset.

**Reconnecting to a disconnected game is already handled implicitly*

<u>Highscores Page</u>

- When the highscores page is loaded, a GET request will be used to invoke accounts.find({},{"username" : 1, "mantra" : 1, "wins" : 1, "losses" : 1, "rank" : 1, "achievementPoints" : 1}).sort("rank" : -1).limit(100) which will return a set of documents containing of the top 100 ranked players and their variables as related to the highscores page. This data will be used to populate the page.

<u>Achievements Page</u>

- When the achievements page is loaded, a GET request will be used to invoke achievements.find() which will return a set of all achievements documents which will be used to populate the page.

<u>Friends List page</u>

- When the friends page is loaded, a GET request will be used to invoke accounts.findOne({"username" : <username>}, {"friends" : 1}) which will return a documents containing an array of all the user's friends. The data from this document will

be used to find data about each friend. A loop will build a query out of all the friend's usernames like so: {$or : [{"username" : friend1}, {"username" : friend2}, …]}. The following function will now be called: accounts.find(friends query, {"username" : 1, "rank" : 1, "onlineStatus" : 1}). This will return a set of documents with all the necessary data to populate the page.

Disconnecting

- set challenger to null
- set online status to false
- save current game data

Real time

- In order to provide real time challenger alerts, the client must always check the challengers.
- Once a challenge is sent, the client must then check if the challenger is still online to unset the challenger in case he disconnects.
- check if opponent forfeited by challenging someone else

Notes:

- All age-restricted actions will be handled client side.
- Power-ups have been removed because they pose a huge design challenge (which I unfortunately don't have enough time for).
- Removing power-ups also removes the need for a matching algorithm, as now player rank is only based upon achievement points and does not provide any in-game advantage.
- The in-game chat box has been removed for the sake of time.
- Some of the descriptions in this document aren't complete as the assignment was already past due.