Taode Ogden

IGME430 Project 2 Documentation

**Overview:**

For my final project, I decided to make an online clone of the popular party game Cards Against Humanity. The user can join a lobby with up the three other people. Whenever the user wins a game of CAH, their win count will be tallied in the MongoDB database, along with other user info like their username, password, and premium status.

After the user creates an account / logs in, they can either create or join a lobby. When they create a lobby, they must set the length (in rounds) of the game and the lobby’s password – which other users need to join.

When the lobby is created, it is initially in a ‘waiting’ state. When all users indicate they are ready to start the game, via the interface, the game starts and the lobby locks out any other user from joining in.

In the case of a disconnect or page refresh, the offending user is simply removed from the lobby and the game skips to the next round.

**Tools:**

**Websockets.io:** Enables the real-time multiplayer aspect of the application. It allows me to manage multiple client sessions at once, and update a client based on other client’s actions.

**React:** React is used to dynamically render user interfaces without reloading the page. The entirety of the site, including menus, forms, and the game interface, relies on it. What React is used most for is rendering game information sent in from the server via Socket.

**MongoDB:** In this app, MongoDB is used to only store and access account models that are created when a user signs up. An Account model stores the following info:

* username - string (must be unique in the database)
* password – string (encrypted via BCrypt)
* wins – number (how many games the user has won in the account’s lifetime)
* premium – boolean (part of the monetary system)

Dev Process

The development process was definitely a challenge the entire time, as a real-time multiplayer model required me to approach my code in a new way, compared to my previous projects. In a sense, I had to think more like a switchboard operator, keeping track of multiple clients at once.

The biggest challenge was updating an individual client depending on the game state, such as when they are / aren’t the judge (in CAH, the judge waits for the other players to give them responses to the given prompt) for the round.

Another challenge was handling disconnects. I ultimately had to settle for kicking the user out, instead of persisting their page, because of the difficulty of determining whether they decided to leave the game or just refreshed the page. Having the game and site persist on refresh would be the first thing I would add if I continued working on this project.

There are definitely a few kinks – all related to mostly the interface not rendering the correct information. For example, when the server detects that all users have readied up, the ready button does not disappear. While this doesn’t break the app, it could certainly be a bit confusing to the user. But that seems to be the case for the other issues; none of them are gamebreaking, but they remove from the app’s polish.

**Monetary System:**

When the user enables / buys Premium Mode on the Account Settings page, they simply get a nice, entirely cosmetic, crown icon next to their name in the lobby interface.

**Looklng Forward…**

At this point, the app definitely feels complete. But there are a few things I would want to add:

* A loading screen while the site makes a request to the RestAgainstHumanity API for the entire base set of black & white cards; since it needs to get such a massive amount of data, it takes a noticeable amount of time to be fulfilled.
* Unlocking the hard lobby user cap of 4; this limit was set to scope down the project.
* If I were to add to the monetary system, I would give Premium status users access to all the CAH card expansions from the API.
* Make the interface more ‘gamey’; such as displaying the user’s hand as an actual hand of cards.

**Above and Beyond:**

The multiplayer aspect of my app is my main above and beyond contribution. I’m impressed that I was able to get it working, including the game loop itself, handling disconnects, and the lobby system. Besides that, I spent quite a long time styling the site with Bulma.

Resources

<https://cssgrid-generator.netlify.app/> CSS Grid Generator: an online tool to generate CSS code for grid layouts. Any grid code you see in my CSS was copied from here.

<https://www.restagainsthumanity.com/2.0/> API that stores black & white cards from every Cards Against Humanity set.

<https://socket.io/> Explained above.

<https://bulma.io/> Styling.

<https://fontawesome.com/> -For some of used icons

Official Cards Against Humanity Logo – Taken and edited for title & favicon purposes

Other code fragments – noted and linked in code

Endpoints

‘/’

Supported Methods: GET

Middleware: Require Logout

Query Params: none

Description: page on initial site startup

Return Type: JSON

‘/to-game’

Supported Methods: POST

Middleware: Require Login,

Query Params: None

Description: simply a redirect to /gamePage

Return Type: JSON

\*This URL is most likely redundant

‘/sign-up’

Supported Methods: POST

Middleware: Require Secure, Require Logout,

Query Params: username, password, password 2 (to reconfirm the password)

Description: attempts to create a user account in the database

-returns an error to client if username is already taken or the two passwords aren’t identical

-otherwise, creates account and sends user to main menu

Return Type: JSON

‘/login’

Supported Methods: POST

Middleware: Requires Logout, Requires Secure

Query Params: username, password

Description: Logs user in and sends them to the main menu if an account with both parmeters exists.

Return Type: JSON

‘/change-pass’

Supported Methods: POST

Middleware: Requires Secure, Requires Login

Query Params: old password, new password

Description: Changes the user’s password in the database – if the old password is correct and the new password is identical. When successful, logs user out.

Return Type: JSON

‘/setPremium’

Supported Methods: POST

Middleware: Requires Secure, Requires Login

Query Params: bool value

Description: Updates the account’s premium bool in the database to the value. Returns back the value, which is needed to re-render the site.

Return Type: JSON

‘/main-menu’

Supported Methods: GET

Middleware: Requires Login

Query Params: none

Description: Loads the Main Menu interface.

Return Type: handlebar

‘/gamePage’

Supported Methods: GET

Middleware: Requires Login, Requires Lobby

Query Params: none

Description: Loads the Game interface.

Return Type: handlebar

‘/login’

Supported Methods: GET

Middleware: Requires Logout, Requires Secure

Query Params: none

Description: Loads the Login interface.

Return Type: handlebar

‘/account’’

Supported Methods: GET

Middleware: Requires Login, Requires Lobby

Query Params: none

Description: Grabs the user’s Account model from the database as a JSON.

Return Type: JSON

‘/logout’

Supported Methods: GET

Middleware: Requires Login

Query Params: none

Description: Logs out the user back to the Login interface.

Return Type: JSON

‘\*’

Supported Methods: GET

Middleware: none

Query Params: none

Description: Redirects the user back to the Main Menu or Login on any other URL; **404 handler**

<https://stackoverflow.com/questions/6528876/how-to-redirect-404-errors-to-a-page-in-expressjs>