Team Number: 103-4
Team Name: TeamName
Team Members: Austin Lucas (A-Luc), Christopher Mccarroll-Gilbert (chrismcg14), Hayes Vavpetic (hayes-vavpetic), Pranav Subramanian (pranav-super), Justin Murillo (jumu3668)
Application Name: Cards Against Profanity

User Acceptability Testing - A Set of Test Cases

Feature #1: User Account System:

There are two fields that populate the login page, the username field and the password field. To login to the app, one must enter a username and password that match and exist in the database. When the user gets to the login page, their first instinct is to click on the username field, then type in their password and then click enter.

Test Case #1:

If the user enters an incorrect username what happens?
Should a user enter an unrecognized username, we need to return an error saying that the username is not recognized. We prompt the user with the error message “incorrect username/password combination”, prompting the user to try again.

Test Case #2:

If the user enters the correct username but the wrong password then we respond a little bit differently than above.
We do not want to give away that the username was right and that the password wasn’t so we return “incorrect username/password combination.” Like above, we refresh the page/fields so that the fields are empty again.

Test Case #3:

If the user enters a username and password that is correct and exists in the DB then we need to authenticate them.
So, we allow them to login and then redirect them to a landing page. From there, they can create a lobby, join a lobby or check their stats.

Test Case #4:

The user does not have an account yet.
The user can select the “I do not have an account yet” button, and then create an account by entering a username, a password, and confirming the password; the application instance will then forward this new data to the middleware that accesses the DB, and sends the user back to the login screen.

<u>Feature #2: Multiphone Sync:</u>

A main aspect of the game is playing with other people. There are two ways to do so in this app. The user can either join someone's game via an invite that is sent to them, or they can host their own game and send invites out to their friends.

<u>Test Case #1:</u>
The user distributes her invite code and the people on the other end accept said invite. Once this happens, the person/people who accepted the invite are sent into the user's lobby as guests. The user is the host and can start the game when everybody is ready.

<u>Test Case #2:</u>
The user follows an invite code.
When the user accepts this invite, they are sent into the lobby of whoever sent the invite and wait for the host of that lobby to start the game.

<u>Test Case #3:</u>
The invite fails.
There are 3 reasons why this might happen. The first is that when the invite was distributed, nothing actually happened. This would mean that nobody received the invite and therefore nobody could join. The second is that if the user successfully sent out an invite and nothing happened on the other end when it was accepted. This means the player who accepted the invite was not sent into the user's lobby, which would be a fault in either the HTTP request to the node server managing multiphone sync, or in navigating to the lobby page within the app, a fault of the React Navigation library. The third is the user accepted an invite and was not directly pushed into the lobby of whoever sent out the invite.

<u>Feature #3: Gameplay:</u>

The gameplay is the central feature of the game. There are two modes of gameplay: one where the player/user is a judge, who judges other players' submissions, and one where the player/user is a player, where they submit cards based on the topic.

<u>Test Case #1:</u>
The player/user is a judge (a judge of submissions).
In this case, when the round begins, the judge is sent to a wait screen, while waiting for the players to submit their cards. After waiting for about 30 seconds, the application will poll the multiphone sync server for responses to the prompt, and when they are collected, the user will be sent to a judge screen, where they pick their favorite submissions and determine a winner. Upon submitting the winner, they are sent to a

winner screen, displaying the prompt and the winning card (all players see this screen). Here, the player fetches their role for the next round.

Test Case #2:
The player/user is a player (a submitter of cards).
In this case, when the round begins, the user's application instance fetches the current prompt, and the current player's deck, and then the player is sent to a screen where they can see their cards and the prompt. The player has 30 seconds to select these cards, and submit. When they submit, or when the timer expires, the player is sent to a screen that notifies the user of their submission and waits for the judging period to finish. After a timer expires, the user's instance of the game polls the multiphone server to see if the judge has chosen a winner or not. When this happens, the player is sent to a winner screen, displaying the prompt and the winning card. Here, the player fetches their role for the next round.

Test Case #3:
The user is not assigned a state or role of player or judge.
Should this happen, the application instance will repeatedly poll the multiphone sync server for a role, until one is received.

Test Case #4:
The judge doesn't get any cards.
This means that there are still no submissions, likely a fault/mis-synchronization of timers across different phones/users. Should this happen, the judge instance of the game will wait, and then poll the multiphone sync server once more for submissions.

Test Case #5:
The user doesn't select any cards.
Should this happen, after the 30 second period for submission ends, the app will automatically submit, sending an empty submission if the player didn't submit anything. The judge won't even see their empty response.