Tug of Words

Team 05: Ammar Husain, Anoop Jain, Charlie Crouse, Jiwon “Daniel” Kim, Jenna Ellis, and Prashanth Koushik

Design Inspection
Code Inspection
Unit Testing Log

Design Inspection:

Product	Tug Of Words, an IO type racing game		
Date	02/11/2018		
Author	Development Team 05		
Inspectors	Anoop Jain and Prashanth Koushik		
Recorder	Jenna Ellis and Charlie Crouse		
Defect #	Description	Severity	How it was corrected
1	Client portion of product was planned to interact directly with the database. Since the client connected with the server, which also collected and stored information in the same database. Adding database connections to both product components was complicated and generated extra programming overhead that pushed back time spent on development efforts.	1	We made a design decision to only make database connections via the server. This was corrected before any implementation occurred.
2	It was initially thought that both public and private rooms would have random team assignments. However, in order to make private rooms more fun for users, we decided that it would be better to allow players in private rooms to pick the teams that they're on.	1	We made a design decision to allow players in private rooms to pick the team that they're on. This decision was made before any development of private rooms occured.
3	Creating url safe tokens for users to access their	2	We decided to generate URL safe

	rooms. Rooms IDs were not URL safe, and generating hashes from the room ID caused a dependency issue with either storing the hash in the database, or doing a decrypt on the hash to recover the room ID.		room IDs, instead of generating a room ID and also a URL safe token based upon the room ID.
4	Using an appropriate package manager that was compatible with both our client and server frameworks. The default option for both frameworks is npm (node package manager), however, we decided to use yarn, which integrates better with React because Facebook is the developer of both.	1	We switched package managers to yarn, which is a more stable and compatible package manager with React, and was also compatible with Node JS (our server framework).
5	Decision to make a menu for username creation and game creation as one item or separately. The issue with this was in how users could reset their username, and allowing users to create null usernames.	1	User now creates username before entering a main menu that allows them to enter a private/public game room, and gives them the option to reset their username.
6	It was initially thought that users would have to re-enter their username every time they opened the Tug Of Words tab. However, due to the way our database is structured, we decided that it would be a feature if the username that a user used during their last session	1	We store usernames in the javascript local store so that they can be reloaded automatically every time the user comes back to the Tug Of Words application instead of making them

	was saved in the javascript local store and was reloaded the next time the user opened the game.		have to type in the username again. However, we also have a button that users can click if they would like to reset their username to something other than what they selected the previous time they were on Tug Of Words. This decision was made before any development was done.
7	The original design for the client involved emitting socket events for any action but, this required to have a persistent open socket even while the user is not in game play.	1	To fix this we decided to implement routes on the server that handle CRUD operations when an HTTP request is made.
8	We originally started testing the client using MochaJS, but we discovered that React already has the Jest environment built into it.	2	We switched our client-side test environment to use Jest instead of MochaJS

Code Inspection:

Code inspection revealed the following defects:

Product	Tug Of Words, an IO type racing game		
Date	02/11/2018		
Author	Development Team 5		
Inspectors	Development Team 5		
Recorders	Development Team 5		
Defect #	Description	Severity	How
1	We had an issue in which there was no socket listening event for removing a user when the user gets reset.	1	We created a listener for when this event occurred in the server.
2	There was an unnecessary try/catch statement which had an empty block. This was an issue because it was looking for an exception which was not specific to the error.	1	We removed the try/catch block entirely and restructured the code.
3	We had implemented the random generation algorithm using a local database of words representing the English language (basically just a very large array). However, randomly selecting a word from an array is a relatively inefficient and ugly way to randomly generate words.	2	We implemented a javascript library that will automatically generate a word for us using a more efficient algorithm. This library will only draw from a couple thousand of the most common English words, but it has more than enough words for users to feel like words are not being

			repeated.
4	We had an issue when writing our unit tests in which there was a problem with our test were asserting “NULL”.	1	In order to solve this issue we needed to change our implementation of the test. This meant that we need to change the NULL to a null.

Unit Testing Log:

At the moment, we do have automated unit testing for the front end portion of our project. In order to automate our unit tests, we have used a library called enzyme that will allow us to initialize specific containers and components of our application and traverse through the output to make sure that there the render was successful. As any other testing framework, we are able to define suites for a full container, and different cases for properties and states of the component.

For the server unit testing framework, we decided to use mocha, a serial unit test framework which results in really nice reporting.

Our system relies on two sets of modules, one set for the client component, and one for the server component.

Client Modules:

- Main-Menu Module

Server Modules:

- App Module
- Firebase (Database connection) Module
- Game Module
- Room Module
- User Module

Client Module Testing Summary

Main-Menu Module:

The main menu module is a front end container that is used to direct the user to create a username; upon creating a username, the user is directed to a page that allows them to specify what sort of game to create (public or private). This page also allows users to "reset" their username, which allows them to return to the username input screen.

Product	Tug Of Words, an IO type racing game		
Module	Main-Menu Module		
Date	2/11/2018		
Author	Development Team 05		
Defect #	Description	Severity	How it was corrected
1	Tests were failing because the redux store was not inherited in any components rendered in the testing environment.	3	Install enzyme and react-adapter to render shallow components that do not need their parent component's properties passed down
2	Tests were failing because the MainMenu container requires several functions to be passed down even with shallow rendering	2	Create stubs for all required functions that imitate the functionality needed by parent functions.

Server Module Testing Summary

App Module: This contains the actual server implementation. The code initializes the express/socketio instance and has the logic for the socket listener functions.

Product	Tug Of Words, an IO type racing game		
Date	02/11/2018		
Author	Development Team 5		
Defect #	Description	Severity	How
1	The tests were failing because the server could not be accessed from the test environment	2	The server runner was not being exported from app.js so the tests were failing when trying to access the module

Firebase Module: The firebase module is a module that allows our application to connect to our Firebase real-time database

Product	Tug Of Words, an IO type racing game		
Date	02/11/2018		
Author	Development Team 5		
Defect #	Description	Severity	How
1	The wrong module was being imported due to using es6 syntax. The import created an undefined object rather than throw an error.	3	Switch the import to es5 to be compatible with our compiler.

Game Module: The game module supports all server operations for the in-game aspect of Tug Of Words. This includes incrementing and decrementing point values for users as they type words correctly and incorrectly as well as returning random words to clients as they request them.

Product	Tug Of Words, an IO type racing game		
Date	02/11/2018		
Author	Development Team		
Defect #	Description	Severity	How
1	While we were implementing a random word generation feature, we ran into linting issues from yarn because there was no other development that had been done that required that function.	1	We fixed this by implementing the boilerplate code in a different way that will pass all of the linting tests. We imported the random word generation function into the app.js file, where it will eventually be used, in order to pass all of the lint tests.

<u>Room Module:</u> The room modules is where the app is in a pre-game state. This includes several definitions of functions that

Product	Tug Of Words, an IO type racing game		
Date	02/11/2018		
Author	Development Team 5		
Defect #	Description	Severity	How
1	While creating the private room for the user, we were getting an undefined field error when trying to add a field to firebase.	2	This ended up being a merging error, in which we were adding a new owner field for the room, which was not being populated from the api. This was fixed by simply adding a correct field.

<u>User Module:</u> The user module is where the players of Tug Of Words inherit their different attributes. This module contains the attributes of a user such as user id (uid) and username. This module is important because using this information the players can interact with other players by joining rooms and collecting points. This user module will

allow us to expand upon what the users can do. As of right now, the user has been simplified for the sake of this sprint.

Product	Tug Of Words, an IO type racing game		
Date	02/11/2018		
Author	Development Team 5		
Defect #	Description	Severity	How
1	When the client would connect to the socket, the server would return an error, and the socket connection would break.	3	When our automatic unit test were failing, we were able to pinpoint which module showed a failure, corrected it by adding CORS standardization on our server, allowing multiple origins and domains on our react app.