**Computer Science Project**

**Test Report**

**Project 4**

Jake McNee

25198572

Monash University Clayton Campus

Semester 1, 2018

Class of  
Prof. David L. Dowe

**Introduction**

The overall approach to testing can be described as a modular approach. During each implementation of a different module, testing would be done using different cases and data. The idea try to anticipate anything the user might input in use that would cause the system to fail. Once all modular components were completed and tested, the system could be combined and final testing complete.

The different test modules can be broken down as follows

* Unit Test – Fixture Input to Firebase
* Unit Test – Users Input to Firebase
* Unit Test – Client Side table created
* Unit Test – Client Side table round/total scores
* Numeric Test – Client Side table round/total scores
* Unit Test – Logic Side and game mechanics
* System Tests – All components

Some are per the project specifications and will be followed on for tests with the final product, other have been decided to be implemented during the process to provide adequate testing.

**Unit Test – Fixture Input to Firebase**

This unit test is used to confirm the .xlsx conversion into firebase.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Steps** | **Description** | **Expected**  **Result** | **Pass/Fail** | **Comments** |
| **1** | Create fixture file to be stored in project code directory | System can locate the fixture file | Pass, system finds file without errors | Stored in ‘Input\_documents’ |
| **2** | Input fixture file into server using xlsx module | Server reads the file and is passed through xlsx module to be converted into JSON | Pass, on console.log of file after conversion, JSON dictionary is shown | Feedback is given through console.log |
| **3** | Server converts JSON dictionary into firebase database nodes | Data originally from spreadsheet can be viewed in a format in firebase | Pass, data is broken down into rounds, with a unique key for each match and its corresponding data | Unique key used so that matches do not replace each other |

**Unit Test – User Input**

Similarly, this is created so that users of different sizes can be tested to ensure data is successfully transferred over to firebase.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Steps** | **Description** | **Expected**  **Result** | **Pass/Fail** | **Comments** |
| **1** | Create user file to be stored in project code directory | System can locate the user file | Pass, system finds user spreadsheet without errors | Stored in ‘Input\_documents’ |
| **2** | Input the list of users into server using xlsx module | Server reads the file and is passed through xlsx module to be converted into JSON | Pass, on console.log of file after conversion, JSON dictionary is shown | Feedback is given through console.log |
| **3** | Server converts user JSON dictionary into firebase database nodes | Data originally from spreadsheet can be viewed in a format in firebase | Pass, data is broken down into player IDs, each with their respective data attached | PlayerID used for reference later, also implemented in a way such that number of users can be counted and saved in gameData node |

**Unit Test – Client Side Table Created**

The main goal of this test is to ensure a table can be created based on the number of users and number of games. Furthermore the functionality of updating the firebase on any new values entered into the table is also tested.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Steps** | **Description** | **Expected**  **Result** | **Pass/Fail** | **Comments** |
| **1** | Js code on HTML page accepts firebase calls to gameData node containing number of users and number of games | No error on calling the required nodes from firebase | Pass, Firebase functionality set up correctly | Had to use .child(“numUsers”) and .child(“numGames”) for this to work |
| **2** | Create a table using javascript to insert elements into HTML page, arbitrary rows/columns | Passed on arbitrary values a table is created on HTML page | Pass, HTML pages shows correct number of columns and rows |  |
| **3** | Table now makes the required number of cells based on number of users/games and any additional cells needed | Rows match number of users +1 and columns match number of teams +3 | Pass, table is empty however has the correct number of cells | Extra cells that were implemented will be used for titles/scores etc. |
| **4** | Function to update the cells in table based on data in firebase | If data exists for the cells, the should reflect in the table for the client to see | Pass, function uses an if statement to determine if there is data and if it should be updated | Table is created as empty first, then cells are updated after. |

**Unit Test – Client Side table round/total scores**

This test is to make sure entering values in the input boxes on the client side behave as expected.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Steps** | **Description** | **Expected**  **Result** | **Pass/Fail** | **Comments** |
| **1** | Round score works for current round for all players | Based on the players tips for that round, the round score at the end sums all up | Pass, the round score is all calculated tips added together | Separate submit button is used to indicate all tips are ready to be submitted, empty tips = 0 |
| **2** | Total score shows round score for all players | Total score for first round should just be the round score | Pass, total score is round score for all players |  |
| **3** | Round score works for all rounds for all players | Whatever round is selected, round score shows that round only | Pass, round score works for all players on all rounds |  |
| **4** | Total score works for all rounds and all players, answer must be based on previous round score | The total score for each round should reflect that rounds score and all scores before it | Pass, total score shows all players current score to that round | Score had to be calculated off previous round score and added to current round, round score |

**Numeric Test – Round/Total Values**

This test adds some numbers to the previous test case, to try boundary conditions to make sure all cases work.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Steps** | **Description** | **Expected**  **Result** | **Pass/Fail Round Total** | **Pass/Fail Match Total** | **Comments** |
| **1** | Enter no values for input for round 1 | All values default to 0, since 9 rounds 0\*9 = 0 | Pass  Round total = 0 | Pass  Match total = 0 | Match is same as round since it is the first round |
| **2** | Enter 0.1 for each input for round 1 | 0.1 = -2.322 with probability function. -2.322\*9 = -20.898 | Pass  Round total = 20.898 | Pass  Match total = 20.898 | Match is same as round since it is the first round |
| **3** | Enter no values for 2 rounds | All values default to 0. Round score should be 0, total should be the same | Pass  Round total = 0 | Pass  Match total = 0 |  |
| **4** | Enter 0.8 for each input for 2 rounds | 0.8 = 0.678 with probability function.  9\*0.678 = 6.102 | Pass Round total = 6.102 | Pass Match total = 12.204 | Score had to be calculated off previous round score and added to current round, round score |

**Unit Test – Logic Side and game mechanics**

This aims to test the general game mechanics of the system, using current round and all users tipped.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Steps** | **Description** | **Expected**  **Result** | **Pass/Fail** | **Comments** |
| **1** | Current Round created and displayed on HTML | Players can see which round they are on | Pass, the current round is displayed on HTML page | Current round is useful for start and end of system |
| **2** | Next round displayed in database and information not removed from table | When round number changes, no data should be lost from firebase | Pass, round changes and data that has been collected is still on firebase |  |
| **3** | If all uses have tipped, reflect new round upon refresh of page | When last user submits tips, next refresh will be new round | Pass, new round is show on screen and round tab is able to be pressed | Round tab will generate new table |

**System Test**

The system test is aimed to conclude the final product works as expected. Since all modules were tested individually it was expected everything should work together.

In the input documents, there are 2 test versions of the simulated database and fixture input that were used for this system test.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Steps** | **Description** | **Expected**  **Result** | **Pass/Fail** | **Comments** |
| **1** | Users enters correct email address | User logs in successfully | Pass, email matches database |  |
| **2** | Users selects round 1 | Round 1 is displayed | Pass, table showing round 1 data is shown | Both users are performing these actions |
| **3** | Users enters tips and submits | Probabilities calculated and displayed | Pass, probabilities can be seen | Users see their own and each other’s results |
| **4** | User logs out and back in | Current round is increased and users can see the next round | Pass, round 1 and round 2 tabs are shown | Assuming both users finished their tips for the first round |
| **5** | Users enter round 2 tips | Scores Calulated | Pass, round scores and total scores correctly displaying | Only 2 rounds in this example |
| **6** | Users log out and back in | Since we are out of rounds alert should appear | Pass, alert appears stating which user is the winner |  |

**Conclusion**

The main objective with testing for this assignment was to perform multiple unit tests at each iteration. With every new module that was implemented, tests to find faults were run. The idea behind this was if every module could handle any kind of input, the final project would work together. The limitations to this method is, although a system test was performed, due to the lengthy nature of the teams and users, a smaller size had to be used for testing. The logic was developed in a way that if the system works for a size of 1, the system must also work for a size of n + 1 where n is the number of teams. In future testing, large scale inputs could be used to test the strength of the application.