Milestone 5

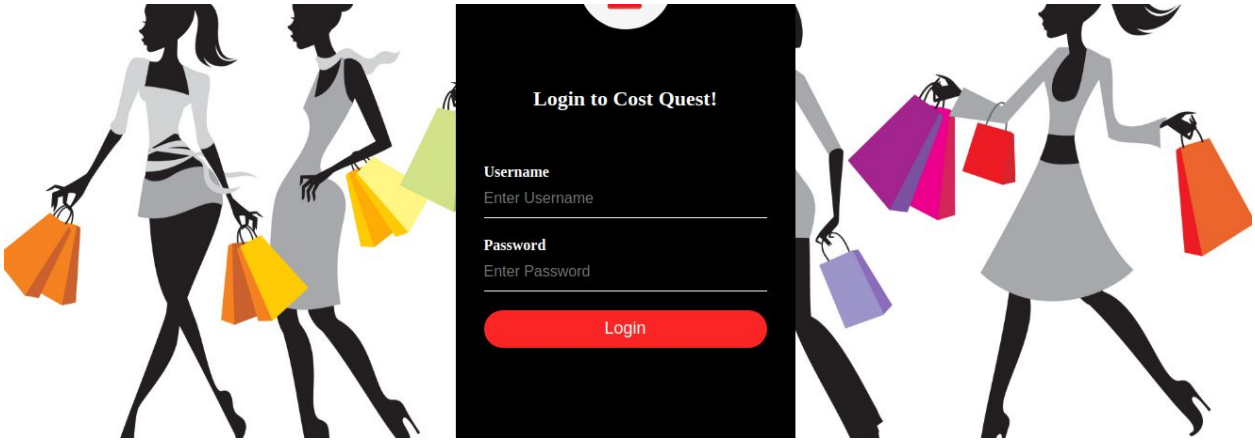
Team Members:
Rebekah Haysley
Zachary Asmussen
Jorge Pulido Lopez
Haotian Zheng
Binpeng Wu
Theodore Margoles

Project Title: Cost Quest

User Acceptance Tests:

The goal of user acceptance testing is to match functionality to requirements. We want to make sure that our user is able to play our game without any error.

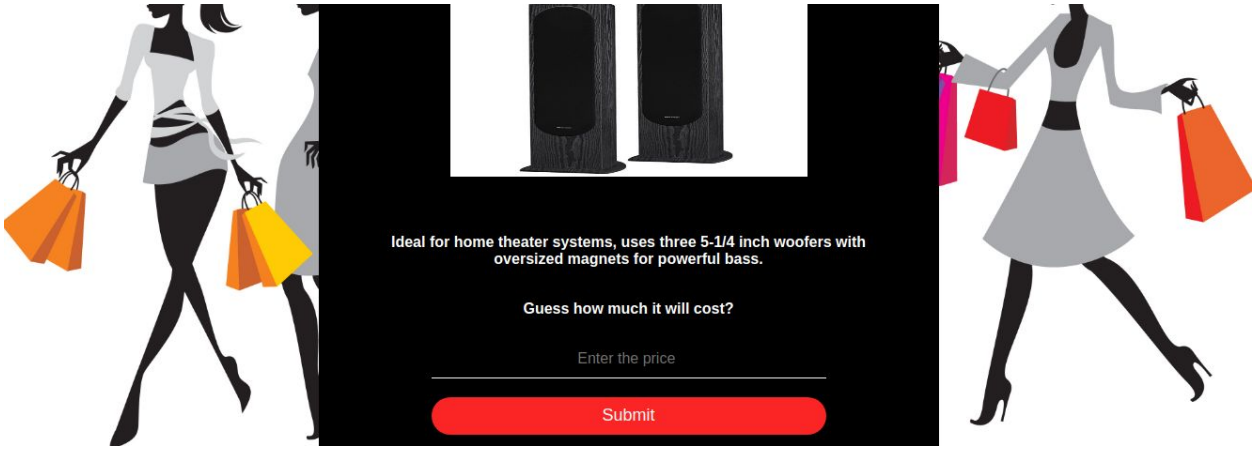
UAT Test #1: User is able to access the Cost Quest login page from a remote PC using our heroku link. For this test we will need the user to be on normal Wifi on CU campus. When the user enters the heroku link into the web browser, they should be able to see our login page for our game. As soon as this login page appears on the screen, the test is complete. This test should work regardless of what laptop it is run on.



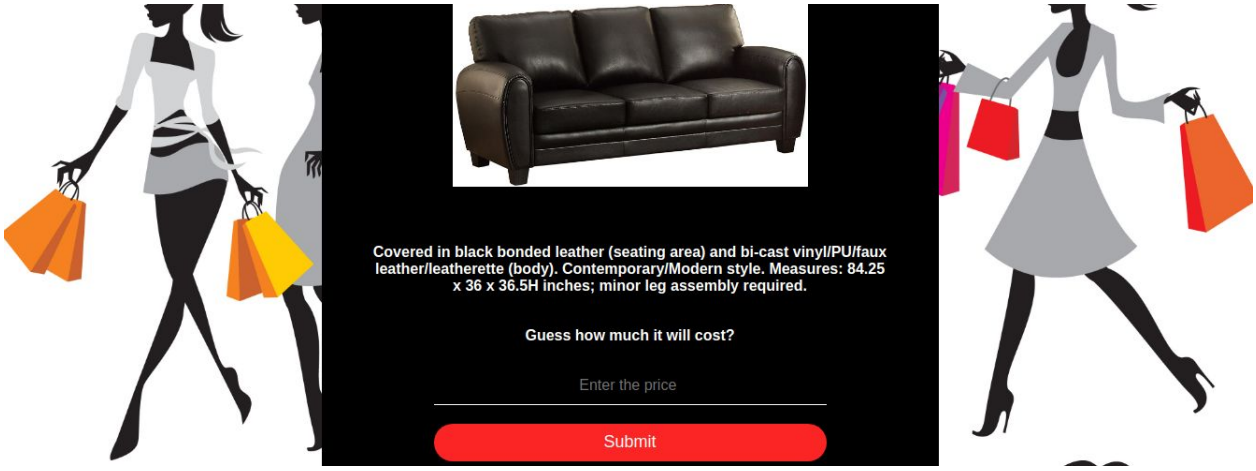
UAT Test #2: User is able to access the game only if they enter the right username and password. This helps ensure that the user profile (with email and scores) is protected. When the user enters a username that is not stored in the database, the user should be redirected to a page that says: “Username is incorrect”.

When the user enters the correct username paired with a password that is not stored in the database, the user should be redirected to a page that says: “Username and password do not match”. When the user enters a username and password that are both stored in the database on the same row, the user will be redirected to the page where they can begin to play the game.

UAT Test #3: User is successfully redirected to the game. After the user correctly enters login information, they will be sent to a page where they must click a button to enter the game. If the user pushes the button, they should be redirected to the actual game page.



UAT Test #4: Game remains functional when user enters an incorrect/absurd input for their product price guess. If the user tries to submit a guess that is not the expected input, our game should not break or crash or interpret the input in an incorrect format. For this test the user should try entering numbers like 0, 999999, -999999. They should also test strings of alphabetic characters and non-alphabetic characters, e.g. (<’:’as;’”kjhgf) .



UAT Test #5: User high score is properly updated and displayed when the user receives a new high score. For this test the user needs to play multiple rounds and try to beat their last highscore. Since the score is reported as the difference between the true product cost and the user guess, this test is successful if the user observes that their high score gets closer to 0.

<u>Automated Testing:</u>

The goal of automated testing is to purposefully try to break our code and recognize what errors should be thrown. This serves to make sure that our program is functioning correctly. (Note: We worked with one of the class CAs to come up with these automated test cases.)

Automated Test #1: Test MySQL by putting bad insert statement into the database creation code. An example of a bad insertion statement would be entering the wrong datatype in one of the row values. For this test, the tool we are using to automate the testing is just MySQL itself. We are testing whether or not the data that we actually want in our database is

entered correctly. When we run the .sql code in MySQL (“source QuestUsers.sql”), we should return errors for the incorrect insertion statements but pass the correct insertion statements.

MySQL test code (QuestUsers.sql):

```
DROP TABLE   QuestUsers  ;
 CREATE TABLE   QuestUsers
   (
   userID int(11) NOT NULL AUTO_INCREMENT,
   UserName    varchar(40)    NOT NULL,
   Email    varchar(200)    NOT NULL,
   Password    varchar(40)    NOT NULL,
   PRIMARY KEY ( userID )
   )
   CHARACTER SET utf8 COLLATE utf8_general_ci;

INSERT INTO QuestUsers VALUES (1, "test #1 - should pass", "email", "password");
INSERT INTO QuestUsers VALUES (2, "test #2 - should fail", NULL, "password");
INSERT INTO QuestUsers VALUES (3, "test #3 - should fail", "email", NULL);
INSERT INTO QuestUsers VALUES (4, "test #4 - should pass", "email", "password");
INSERT INTO QuestUsers VALUES (5, "test #5 - should fail", "missing one");
INSERT INTO QuestUsers VALUES ("string to break insert statement", "test #6 - should fail", "email", "password");
```

MySQL output to terminal window:

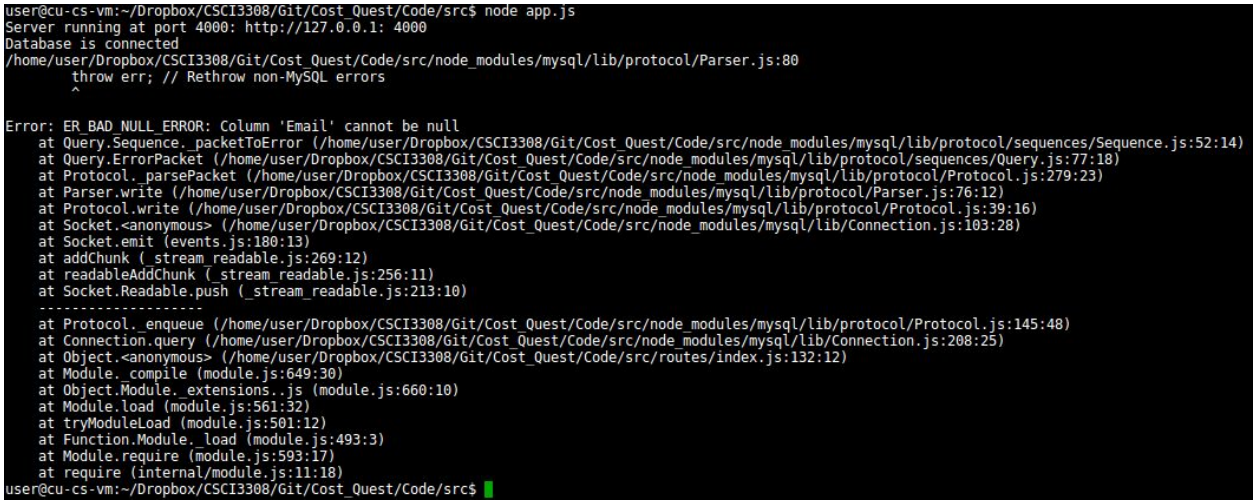
```
mysql> use CostQuest;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> source QuestUsers.sql;
Query OK, 0 rows affected (0.10 sec)

Query OK, 0 rows affected (0.04 sec)

Query OK, 1 row affected (0.01 sec)

ERROR 1048 (23000): Column 'Email' cannot be null
ERROR 1048 (23000): Column 'Password' cannot be null
Query OK, 1 row affected (0.01 sec)

ERROR 1136 (21S01): Column count doesn't match value count at row 1
ERROR 1366 (HY000): Incorrect integer value: 'string to break insert statement' for column 'userID' at row 1
```

Automated Test #2: Test program integration (MySQL, HTML, Node.js) by putting a bad insertion statement into our main script (index.js) and getting an error. For this test, the tool we are using to automate the testing is just Node.js itself. This test is twofold: first we are testing that the database is connected and responding, and second we are testing the Node.js functionality.

Wrong insertion statement in index.js:

```
connection.query('INSERT INTO QuestUsers VALUES (2, "test - should fail", NULL, "password");',
    function(err, result) {
        if (err)
            throw err
    });
```

Resulting error in terminal:



Automated Test #3: Test heroku server connection in an automated using the urllib library (Python script, check_server.py). This test case is still being developed, but below we have included our initial code.

Heroku connection test code:

```
#check_server.py
Import urllib2
link = ‘herokuprojectname.com’
try:
        urllib2.urlopen(link, timeout=5)
except urllib2.URLError as error_msg:
        print “error: test case failed: heroku connection not active.”
```