CS372 Game Project Report

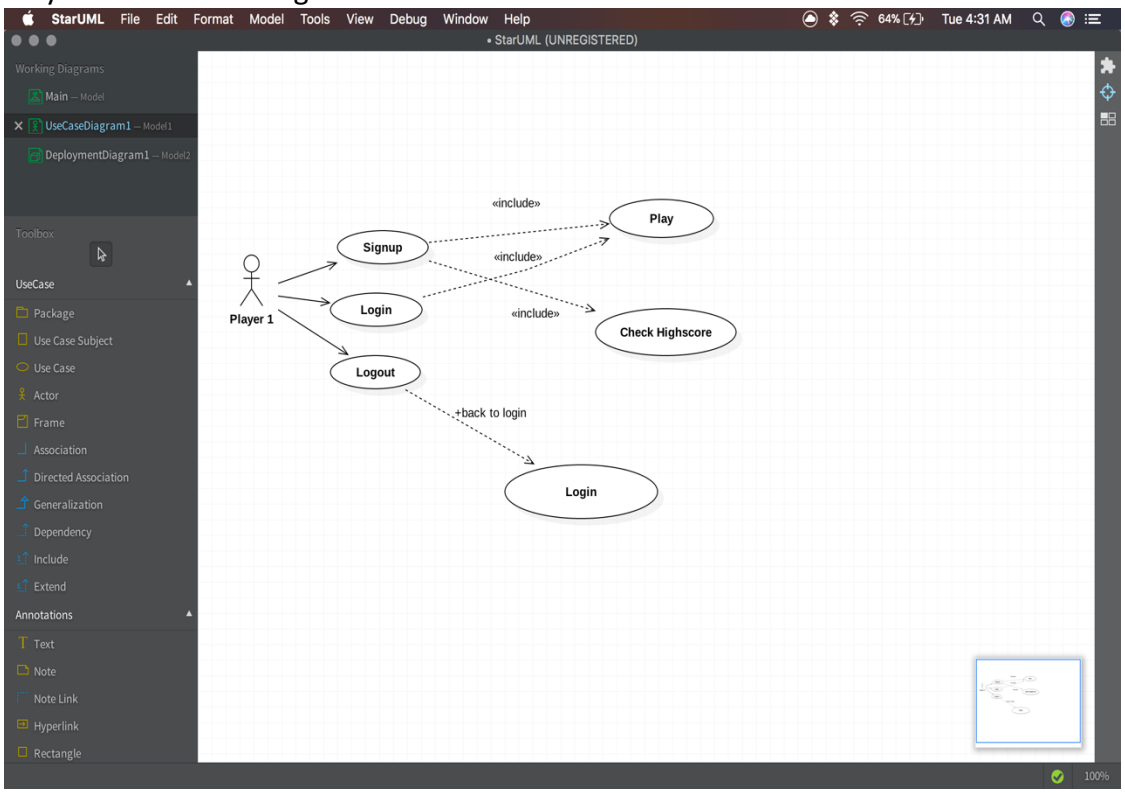
Group Members:
Dhvanil Patel
Cory Lausch
Karanvir Warraich
Sahil Verma
Kevin Peterson

1. Problem definition

   The goal of our project was to develop a web-based game application that could be used to interact with other players by playing against them on a scared canvas which is all connected using various development phases. Players use the web url to access the according game and are required to sign-up in order to be able to get access to the gameplay.

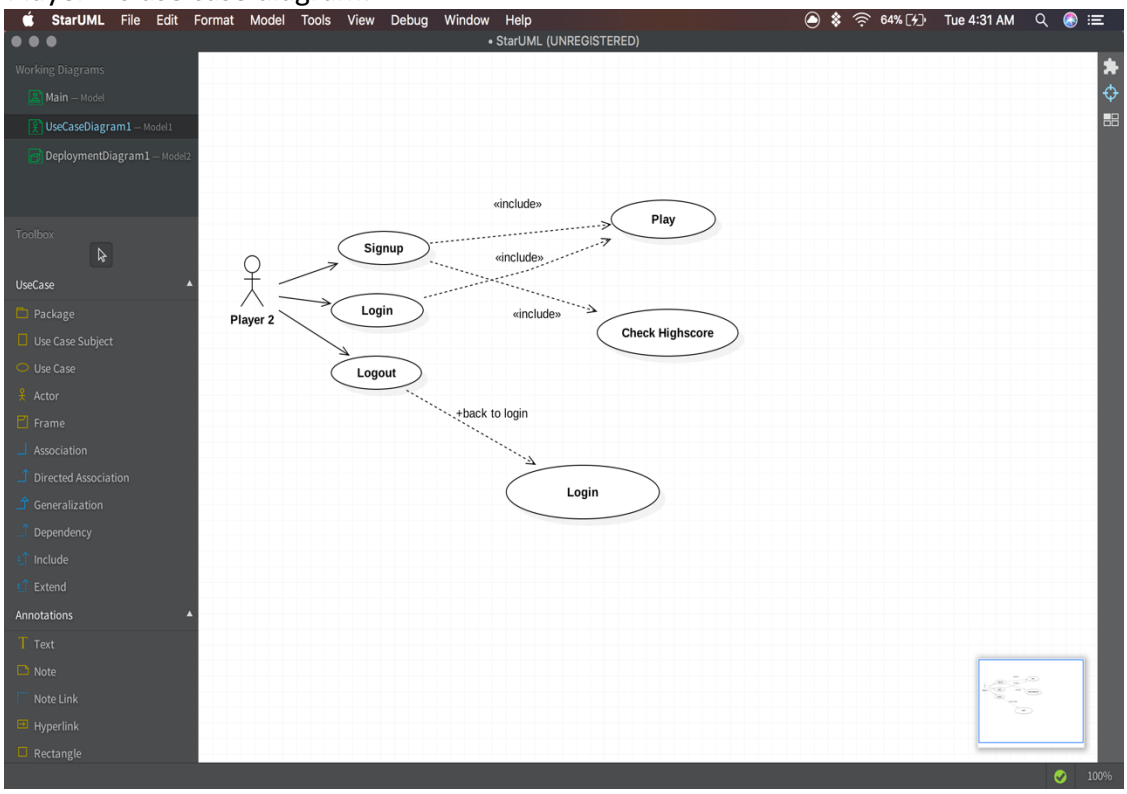
2. Software requirements specification document
   a) **For each type of users, the use case diagram with all use cases**

Player 1's use case diagram:



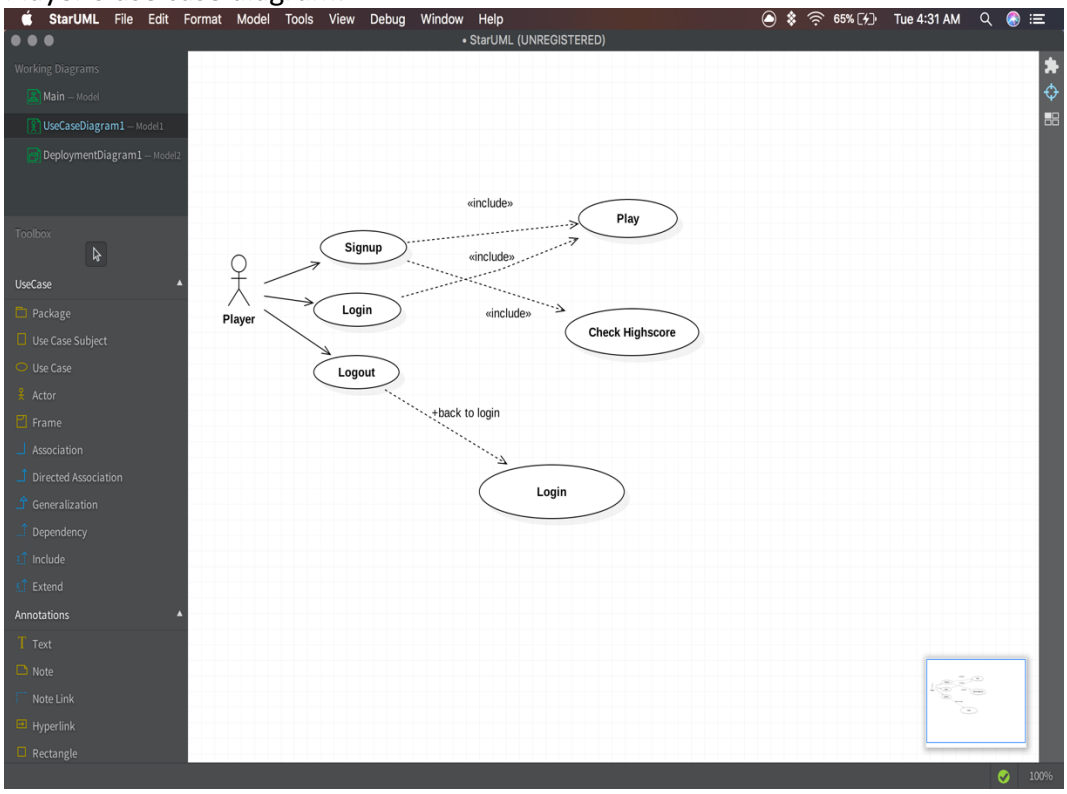
A. Sign-up
   a. Create an account on website
B. Login
   a. Enter username and password credentials to login to the account
   b. The account must exist in order for login to operate properly
C. Play
   a. Wait for opposing player to connect
   b. Play by using keyboard keys as suggested in how to play section
D. Logout
   a. Save the score into the respective profile
   b. Logout to return to the login page

Player 2's use case diagram:



A. Sign-up
   a. Create an account on website
B. Login
   a. Enter username and password credentials to login to the account
   b. The account must exist in order for login to operate properly
C. Check Score
   a. Click check score button to access scores page
   b. The score will be displayed on the screen accordingly
D. Logout
   a. Save the score into the respective profile
   b. Logout to return to the login page

Player's use case diagram:



A. Sign-up
   a. Create an account on website
B. Login
   a. Wait for opposing player to connect
   b. Play by using keyboard keys as suggested in how to play section
C. Play
   a. Wait for opposing player to connect
   b. Play by using keyboard keys as suggested in how to play section
D. Send Chat Message
   a. Use the appropriate keyboard key to enable chat function
   b. Type chat message to send to the opposing player
E. Logout
   a. Save the score into the respective profile
   b. Logout to return to the login page

b) Define in detail three use cases

User case 1: Player Sign-up

Initiating actor:

- Player which wants to sign up to the website

Pre-conditions:

- Customer has an account from which they can play

Scenario 1: Player Sign-up successful

1. Fill in form to sign-up
   a. Allow for fixing errors according to credential check
2. Click sign-up and/or login button
3. Proceed to the game screen

Scenario 2: Player Sign-up unsuccessful

1. Fill in the form to sign-up
   a. Allow for fixing errors according to credential check
2. Click sign-up and/or login button
3. Produce error stating the problem (for example, invalid email address or existing email address)

Post-conditions:

- Account created
- Account not created (i.e. it may already exist)

User case 2: Player check score

Initiating actor:

- Player which wants to login and check the score on the website

Pre-conditions:

- Customer has an account from which they can login and check the score

Scenario 1: Player Display Scores successful

1. Fill in form to login
   a. Allow for fixing errors according to credential check
2. Click login button
3. Proceed to the game screen
4. Click the scores button
5. Proceed to printed scores screen

Scenario 2: Player Display Scores unsuccessful

1. Fill in the form to login
   a. Allow for fixing errors according to credential check
2. Click login button
3. Proceed to the game screen

4. Click the scores button
5. Produce error stating the problem (for example, invalid scores list or no scores to display for the account)

Post-conditions:

- Player's score displayed
- Player's score not displayed (i.e. it may not exist as the player has not played a game yet)

User case 3: Player send message in chat

Initiating actor:

- Player which wants to login and send a chat message to the opposing player on the website

Pre-conditions:

- Customer has an account from which they can login, play and also send chat messages

Scenario 1: Player Send Message successful

1. Fill in form to login
   a. Allow for fixing errors according to credential check
2. Click login button
3. Proceed to the game screen
4. Click in the text box on the screen
5. Proceed to typing the message and press enter

Scenario 2: Player Send Message unsuccessful

1. Fill in the form to login
   a. Allow for fixing errors according to credential check
2. Click login button
3. Proceed to the game screen
4. Click in the text box on the screen
5. Produce error stating the problem (for example, invalid characters and/or words entered into textbox)

Post-conditions:

- Player's chat message successfully sent
- Player's chat message not sent (i.e. it may not exist as the player has entered prohibited text in chat box)

c) Software Qualities

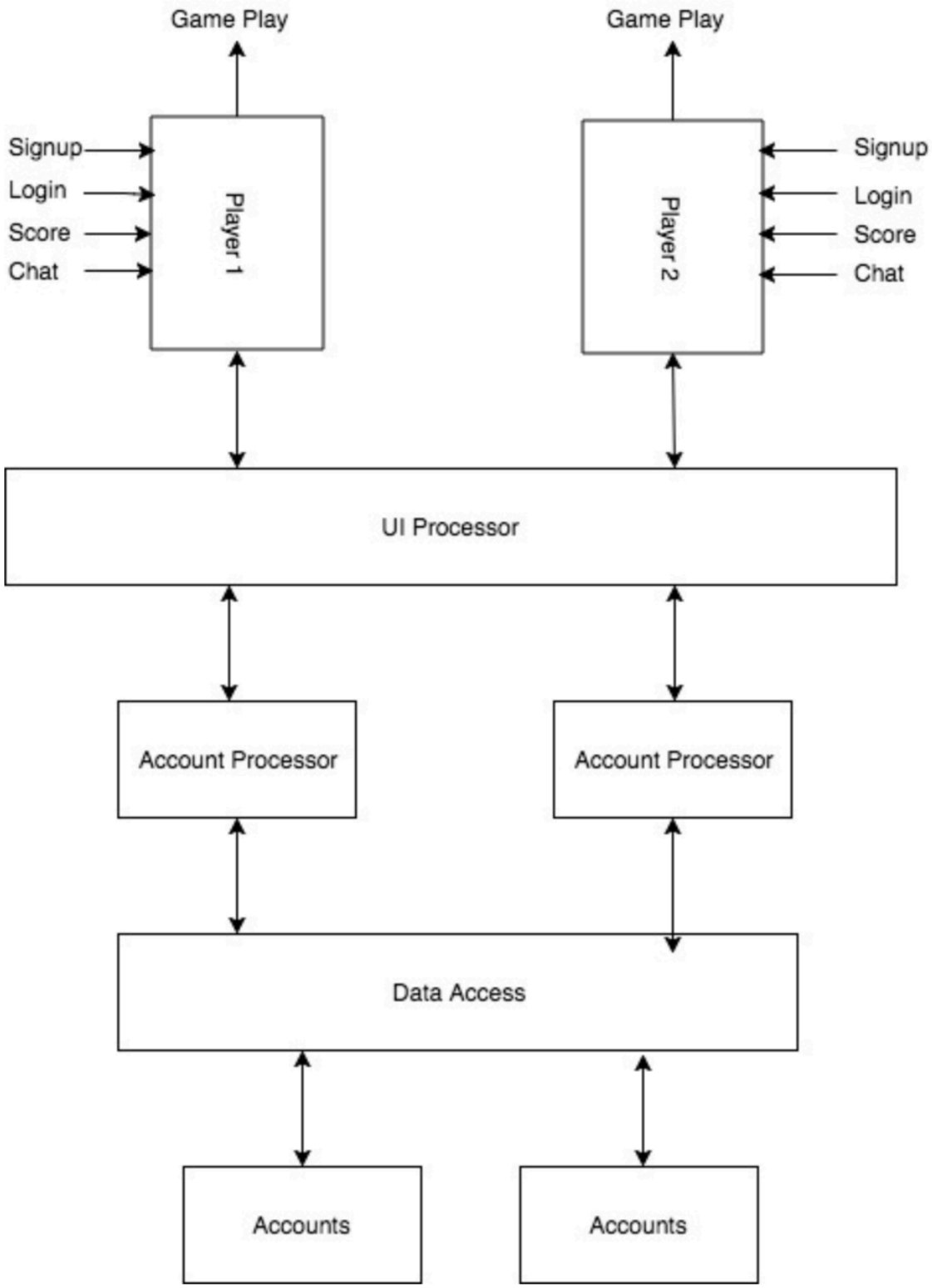
A. Correctness
   - When a player logs in, they are directed to the game screen where they can start playing against other opponents.
   - The players can all interact with each other using the chat box on the side of the screen.
   - Players can check and see their best score as well as the highest score achieved by an individual.
   - The website checks to see if login credentials exist and if they do, they are valid or not.

B. Efficiency
   - The data of the account and score is only sent to the database once a proper sign-up form has been filled and a game has taken place on the account.
   - The database is simply designed to store the values of the account credentials and the scores, no additional information is required to be populated in the database.
   - The login credentials are accordingly limited to prevent spam.
   - The whole web application refreshes without the need of reloading which in turn reduces the stress on the server and bandwidth.

C. Robustness
   - When a new user tries to create an account with account information already on the database, it shows an error message to represent unsuccessful account creation.
   - When a returning user tries to log in to an account with information which does not match the one from the database, it shows an error message to represent unsuccessful account login.
   - If any errors occur, the login/signup page is not bypassed

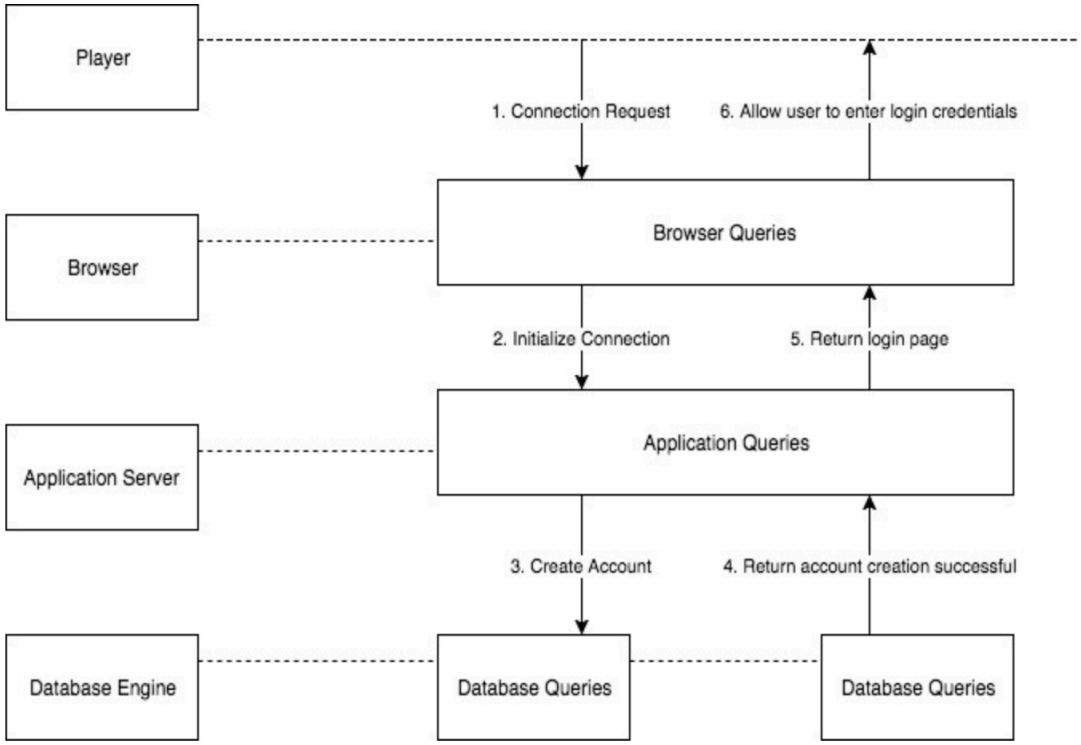
D. User Friendliness
   - Very simple and unique GUI design to maintain low profile.
   - Simple instructions to work their way around the game.
   - Chat feature to allow interaction between all players.
   - High score and score displayed to allow players to see their respective scores as well as the high score which they intend to beat.

3. Design Specification Document
    a. Software architecture (3-layer architecture)

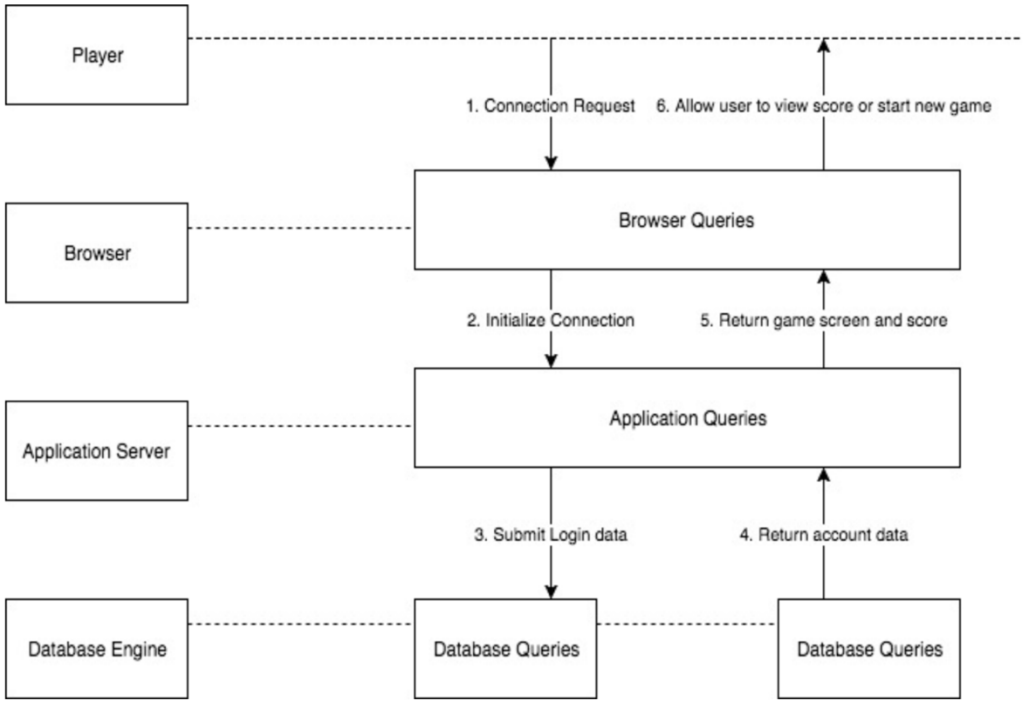


b. Three sequence diagrams for the three defined use cases

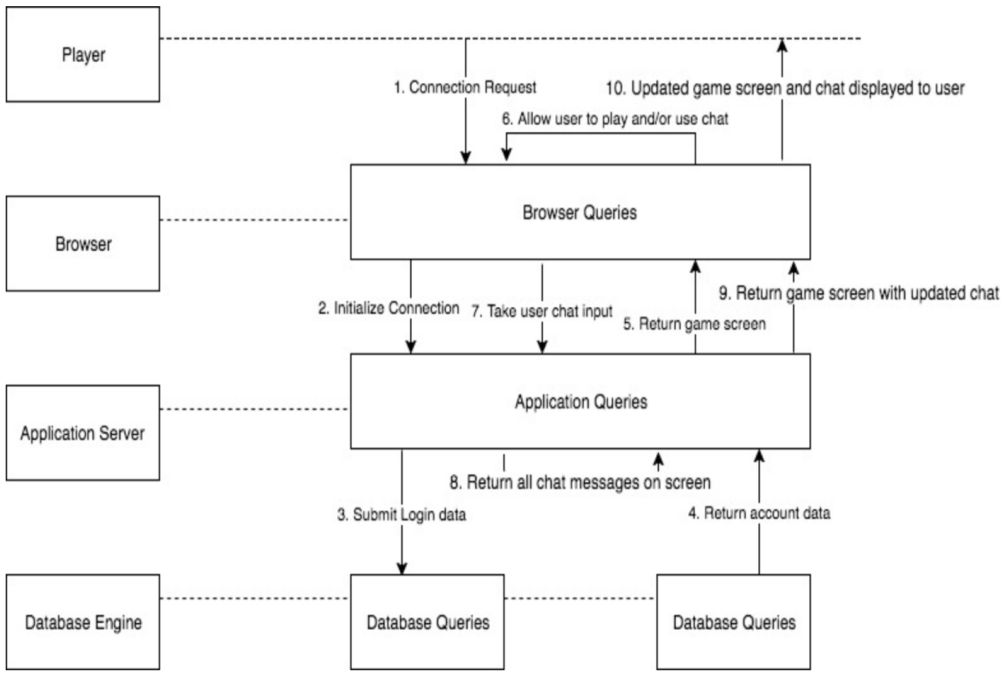
Sequence Diagram Case 1: Sign up



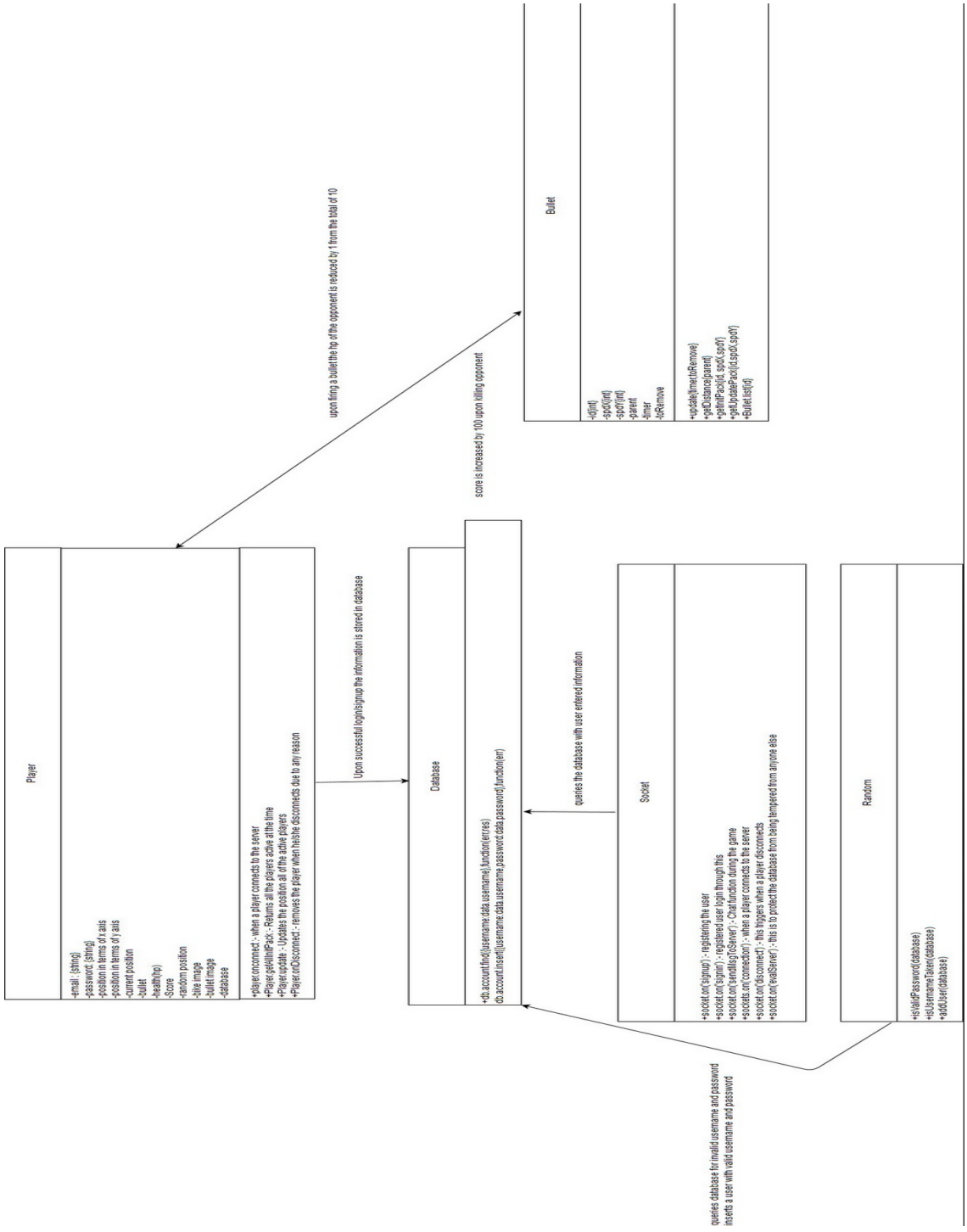
Sequence Diagram Case 2: Check Score



Sequence Diagram Case 3: Send Chat Message



c. Class diagrams with all the classes and their relationships



Bullet

-id{int}
-spdX{int}
-spdY{int}
-parent
-timer
-toRemove

+update{timer,toRemove}
+getDistance{parent}
+getInitPack{id, spdX,spdY}
+getUpdatePack{id,spdX,spdY}
+Bullet.list{id}

Database

+db.account.find({username:data.username},function(err,res)
db.account.insert({username:data.username,password:data.password},function(err)

Player

-email : {string}
-password: {string}
-position in terms of x axis
-position in terms of y axis
-current position
-bullet
-health(hp)
-Score
-random position
-bike image
-bullet image
-database

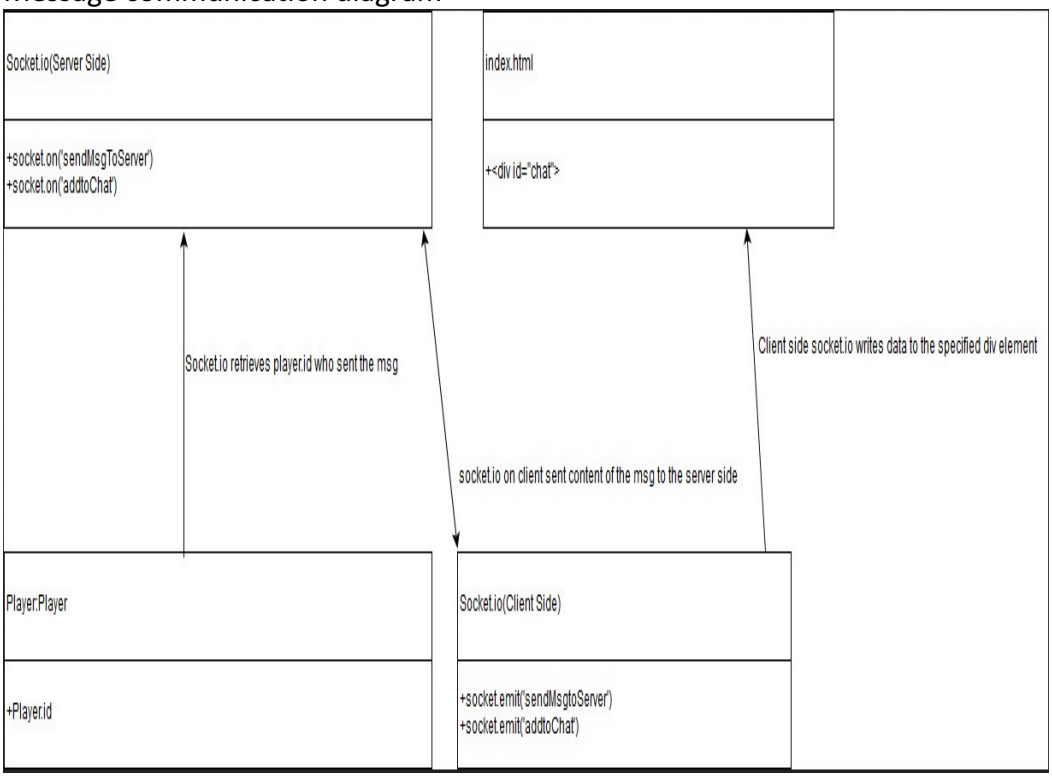
+player.onconnect :- when a player connects to the server
+Player.getAllInitPack :- Returns all the players active at the time
+Player.update :- Updates the position all of the active players
+Player.onDisconnect :- removes the player when he/she disconnects due to any reason

Random
+isValidPassword(database) +isUsernameTaken(database) +addUser(database)

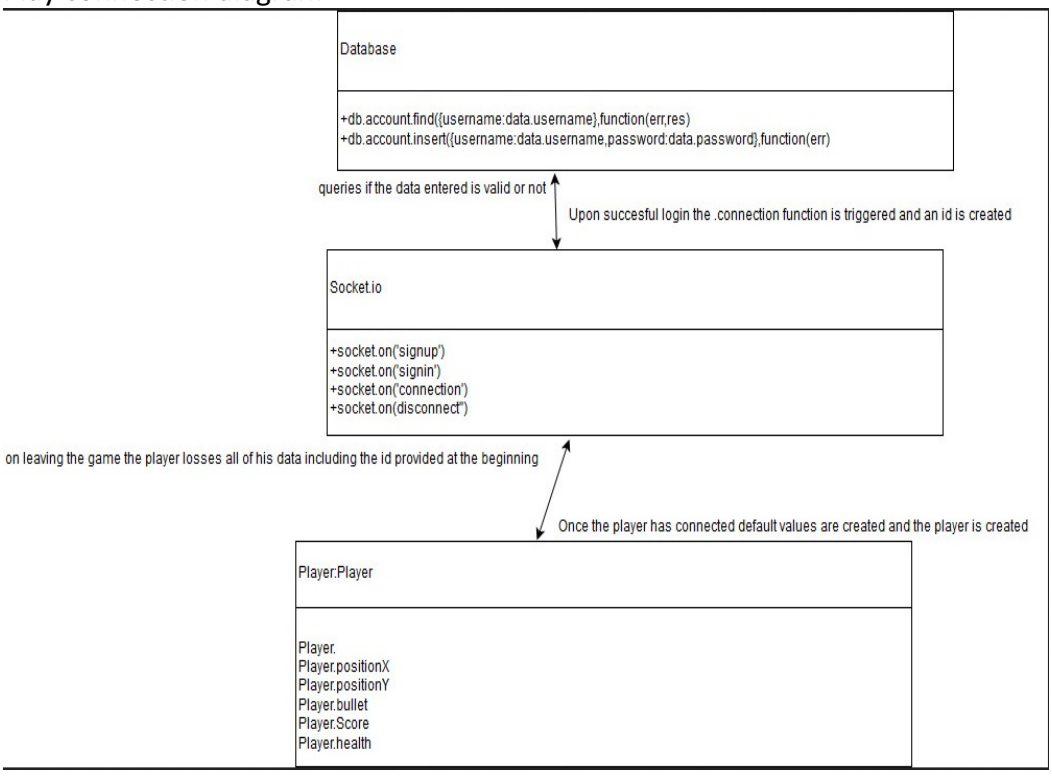
Socket
+socket.on('signup') :- registering the user +socket.on('signin') :- registered user login through this +socket.on('sendMsgToServer') :- Chat function during the game +sockets.on('connection') :- when a player connects to the server +socket.on('disconnect') :- this triggers when a player disconnects +socket.on('evalServer') :- this is to protect the database from being tempered from anyone else

d. Two examples of object diagrams

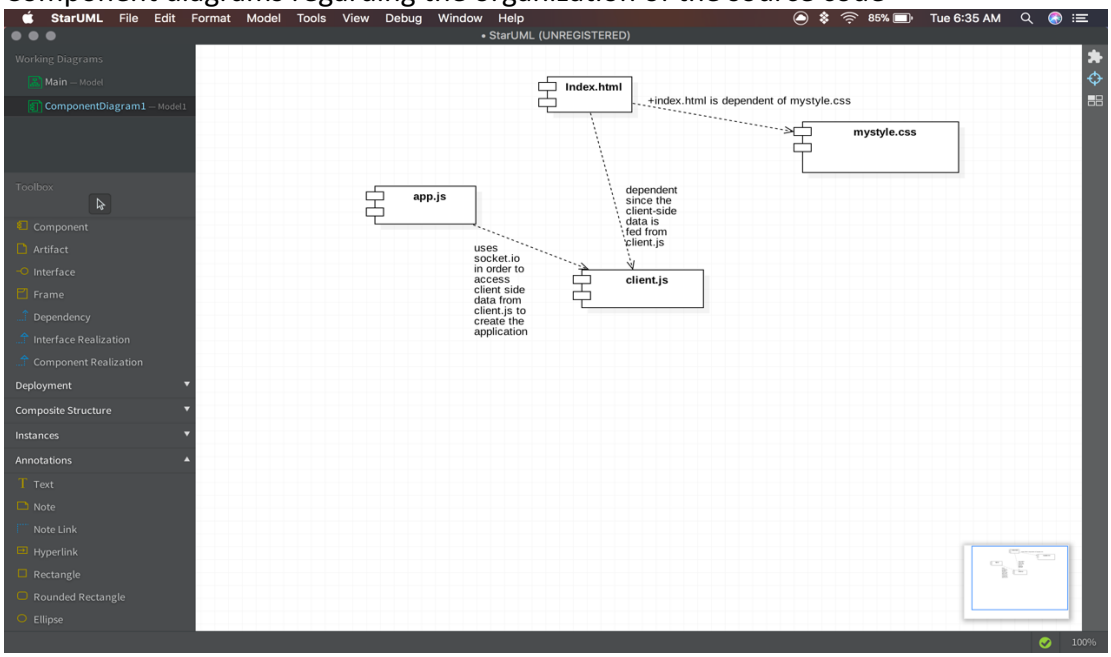
1. Message communication diagram



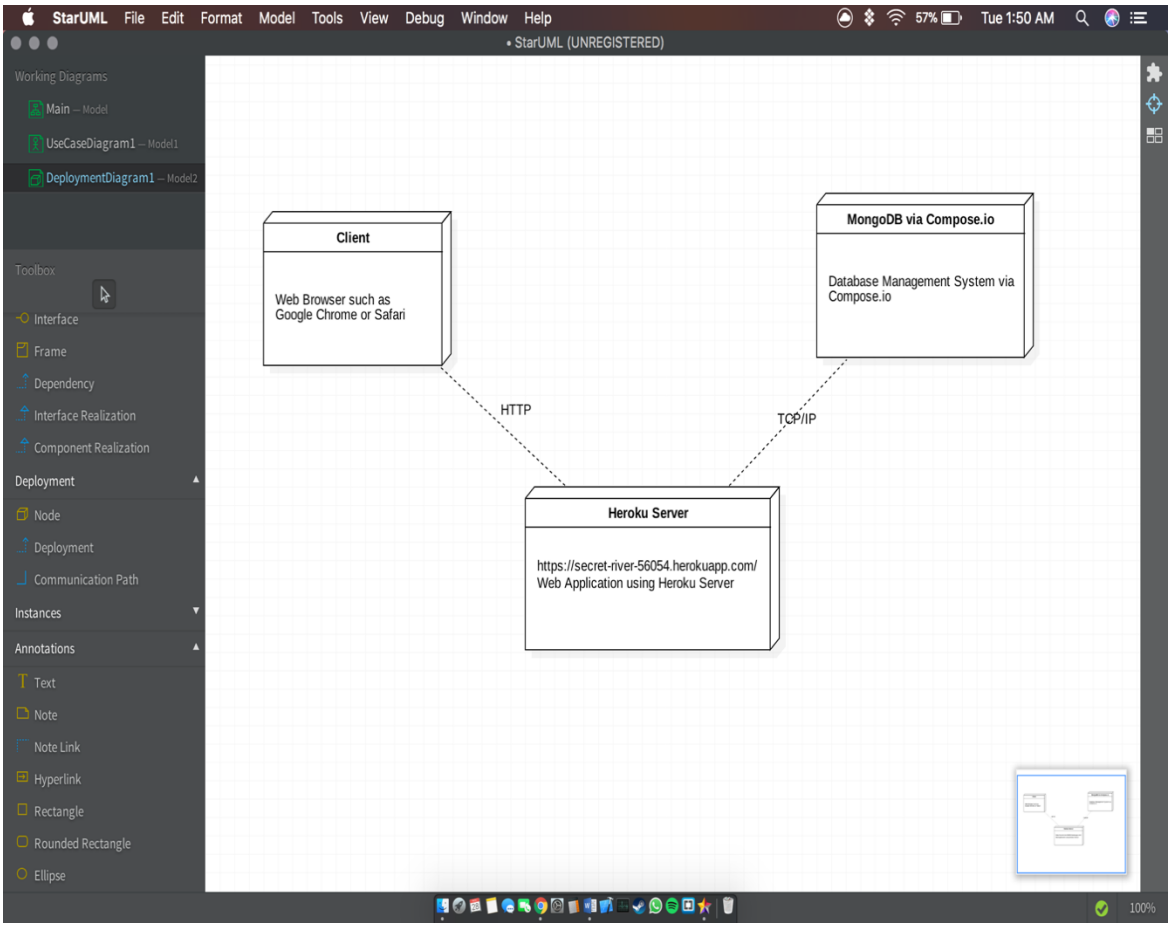
2. Play connection diagram



4. Code Description
   a. Component diagrams regarding the organization of the source code



   b. Deployment diagram regarding the hardware configuration of the software system



c. List the classes and functions that you have implemented
   - Bullet
     - Update(timer,toRemove)
     - getDistance(parent)
     - getInitPack(id, spdX, spdY)
     - getUpdatePack(id, spdX, spdY)
     - Bullet.list(id)
   - Database
     - db.account.find(username:data.username),function(err,res)
       db.account.insert(username:data.username,password:data.password),function(err)
   - Player
     - player.onconnect – when a player connects to the server
     - player.getAllInitPack – returns all players active at the time
     - player.update – updates position of all active players
     - player.onDisconnect – removes player when he/she disconnects due to any reason
   - Random
     - isValidPassword(database)
     - isUsernameTaken(database)
     - addUser(database)
   - Socket
     - socket.on('signup') – registering the user
     - socket.on('signin') – registered user login
     - socket.on('sendMsgToServer') – chat function for the game
     - socket.on('connection') – when a player connects to the server
     - socket.on('disconnect') – triggers when a player disconnects
     - socket.on('evalServer') – protection for the database to prevent any intruder action.

d. Include the link of your Web-based application
   https://secret-river-56054.herokuapp.com/

5. Technical Documentation
    a. UML supporting tool
        - StarUML

    b. Programming languages
        - HTML
        - CSS
        - JavaScript
        - Node.JS
        - mongoDB

    c. Reused algorithms and programs. Include their links.

        - RubentD - Tanks – Copyright © 2014 Rubent D - https://rubentd-tanks.herokuapp.com/

          Tanks was a great motivation for our group to create a game which moves the player in all four directions as well as shoot bullets in all directions. Some issues with learning how to create a code based on the tanks game was the need for various classes and functions which needed to be changed to support additional features.

        - Snake – Copyright © 2016 (unknown) - https://node-multiplayer-snake.herokuapp.com/

          Snake helped us learn how we could include movement and detect health for our player.

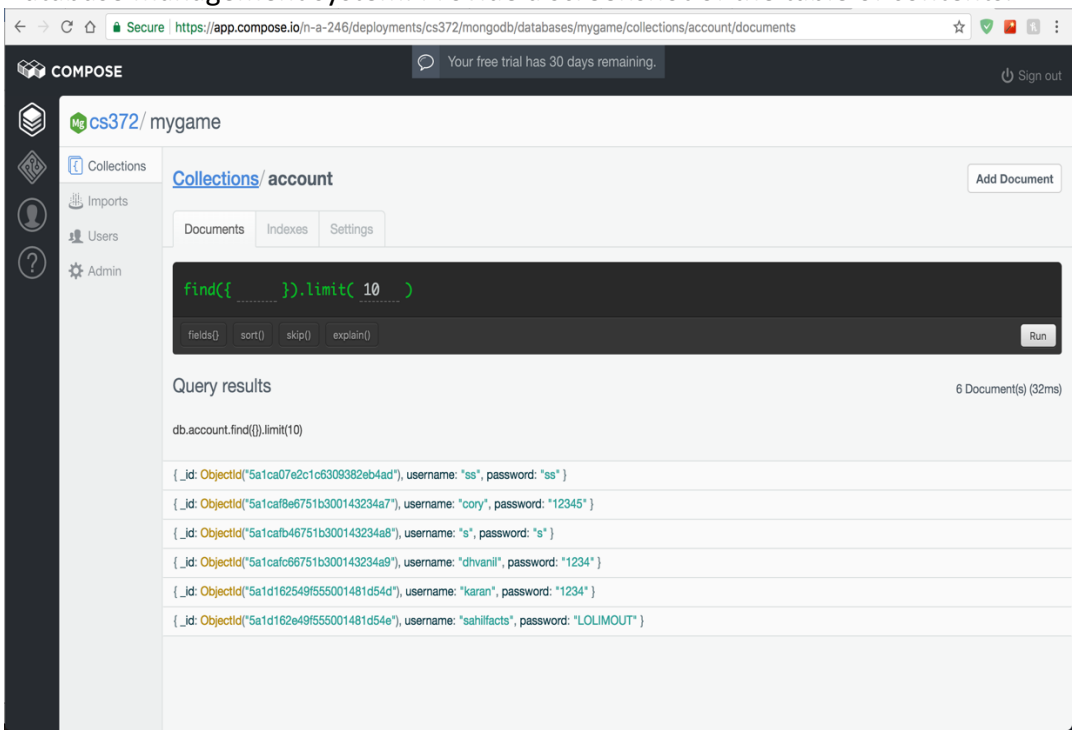
        - Multiplayer HTML5 Game – Copyright © 2016 RainingChain - https://www.youtube.com/watch?v=-N9XXDpde8g

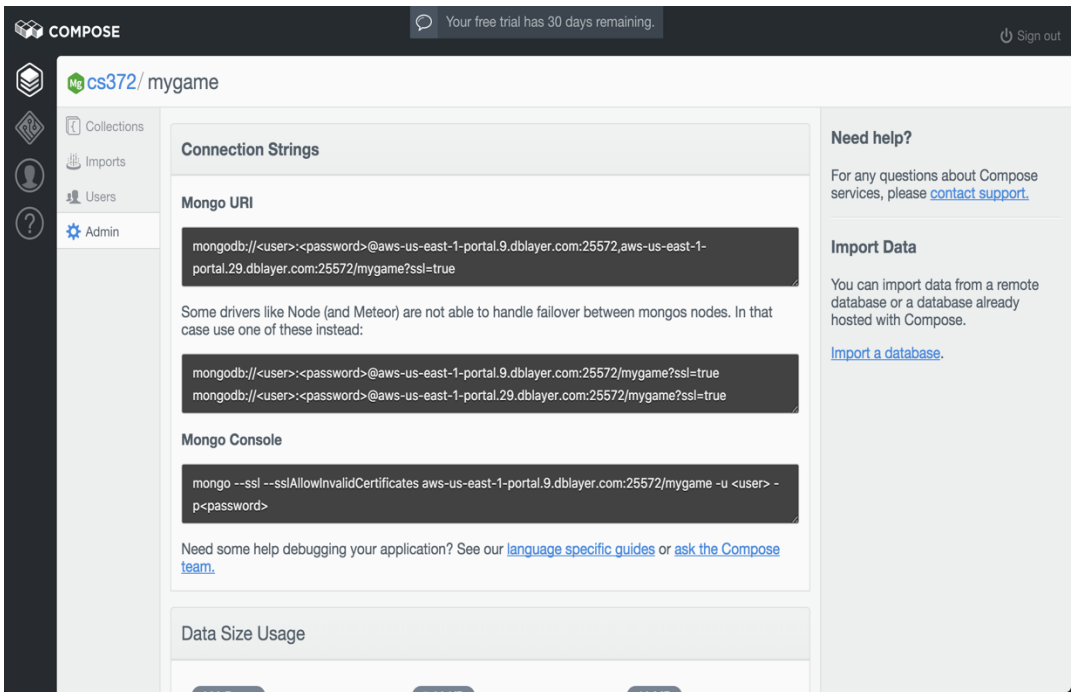
        - Compose.io and MongoDB Jump Start – Copyright © 2014 Life Michael - https://www.youtube.com/watch?v=CqyuwwnkpHk

    d. Software tools and environments. Indicate which software parts are using which tools.
        - StarUML
            - Supported on cross-platforms such as Linux/UNIX and Windows systems.
        - Brackets
            - Used to code the web application using HTML5, CSS and JavaScript
        - Sublime Text 3
            - Used to edit code for the web application using HTML5, CSS and JavaScript

- Node.JS
  - Used to create multiplayer functionality used Node and JavaScript
- Command Line
  - Used to join the codes to produce the web application
- GitHub
  - Used to create, share, and edit repository with updated project files
- Socket.io
  - Used to create connection between the server and client.
- Draw.io
  - Used to create software architecture as well as various object diagrams and class diagrams.
- Compose.io
  - Used to enable the database to be available for online use for the web application
- MongoDB
  - Used to create, and maintain the database management system of the web application.
- Heroku Online
  - Used to load the web application to the server for enabling multiplayer functionality.

e. Database management system. Provide a screenshot of the table of contents.

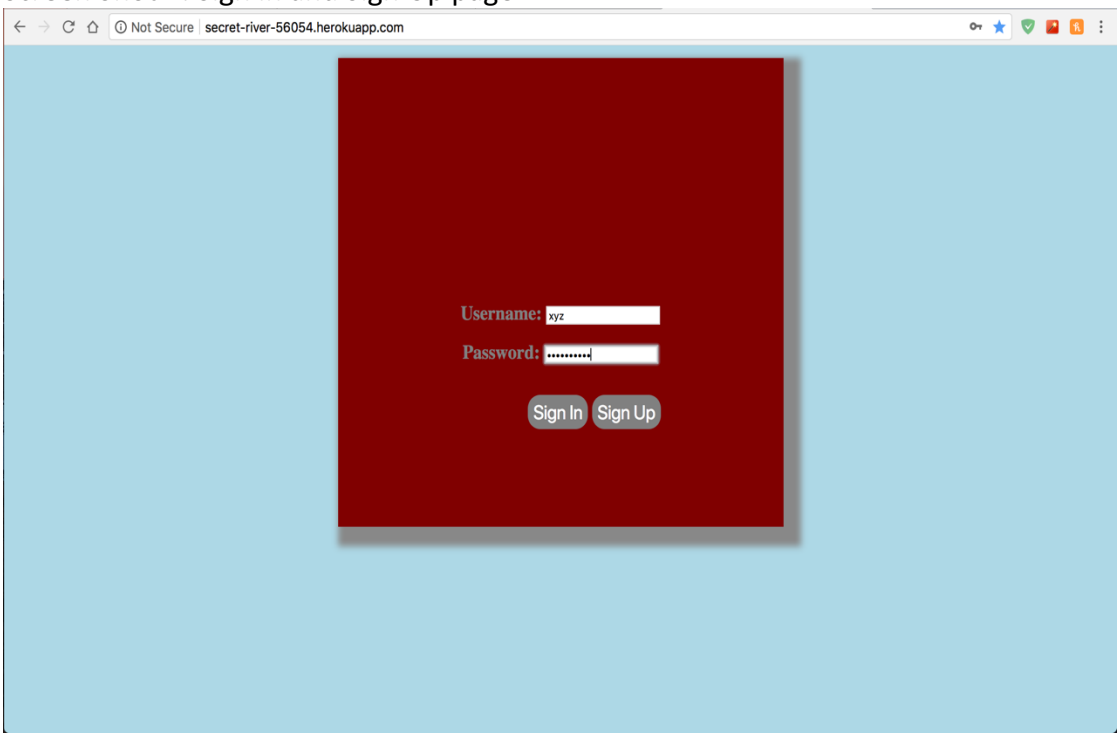




Note: Due to creating a web based game using Node.js and mongoDB, we do not have a proper table of contents as there is no such possibility.

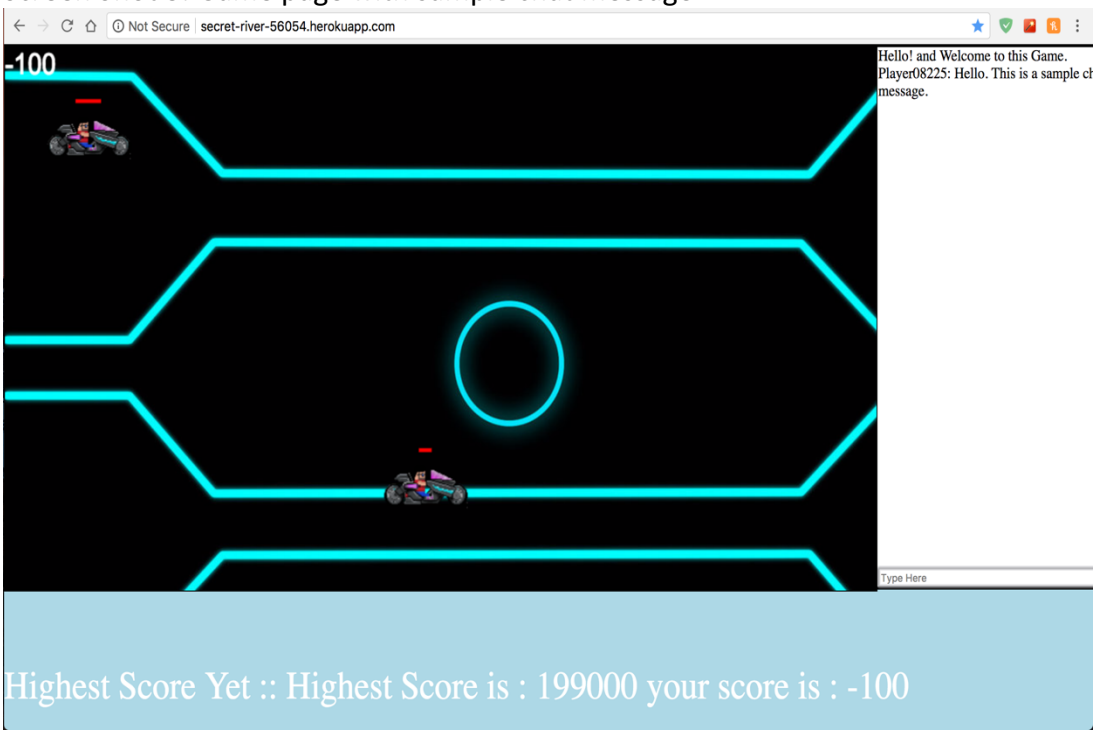
6. User documentation including at least 8 different screen shots of the GUIs.

Screen Shot 1: Sign in and Sign Up page

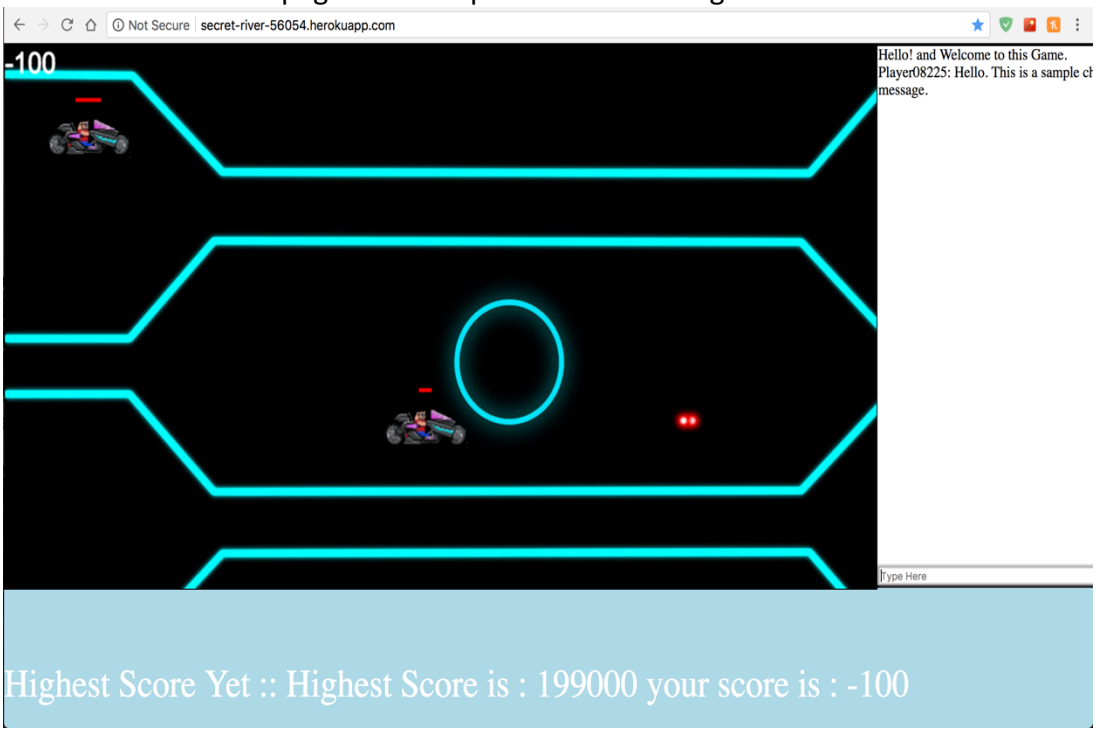


Screen Shot 2: Initial game page

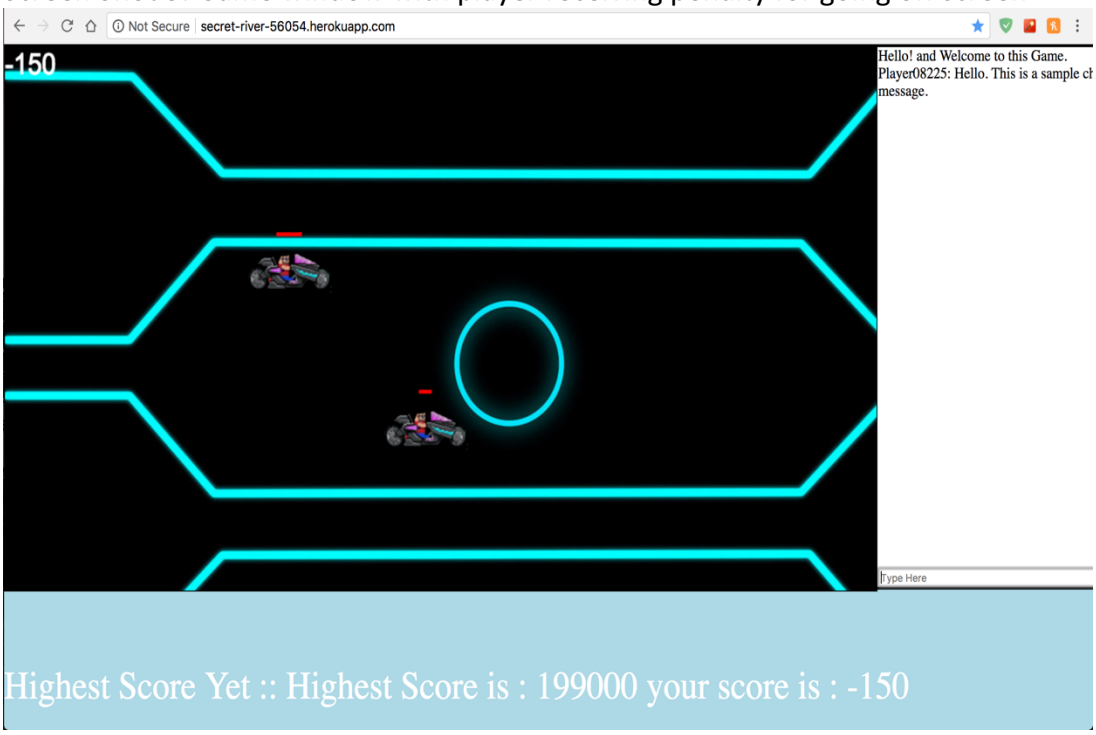
Screen Shot 3: Game page with sample chat message



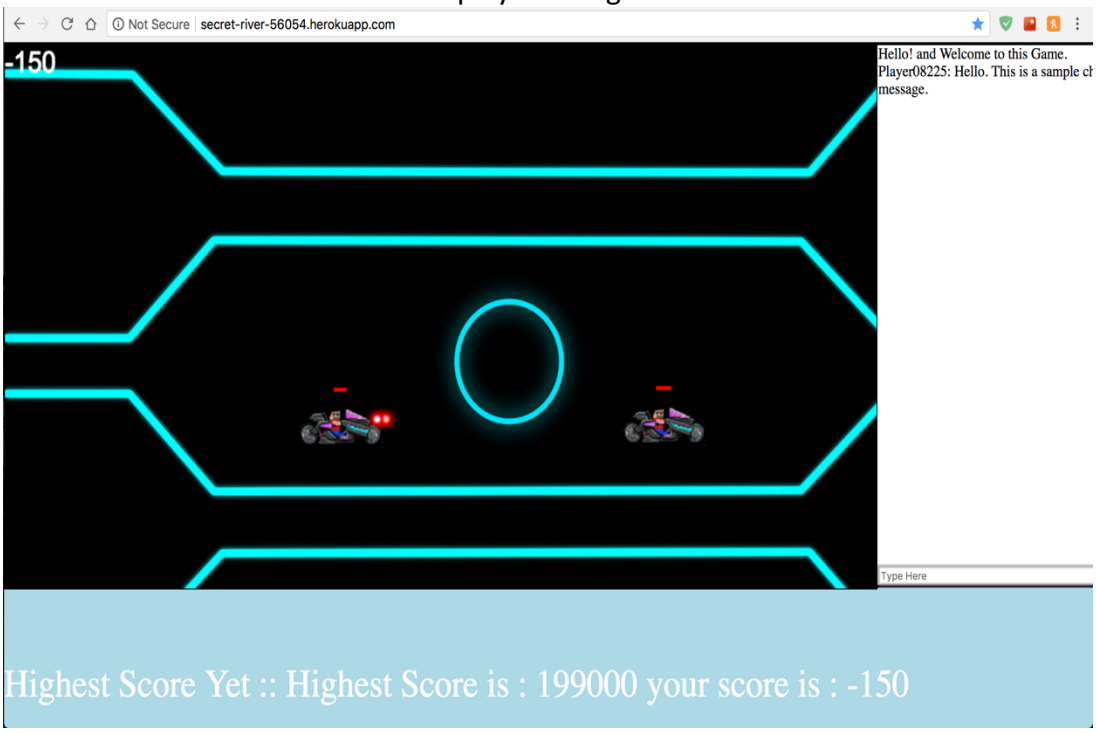
Screen Shot 4: Game page with sample bullets shooting



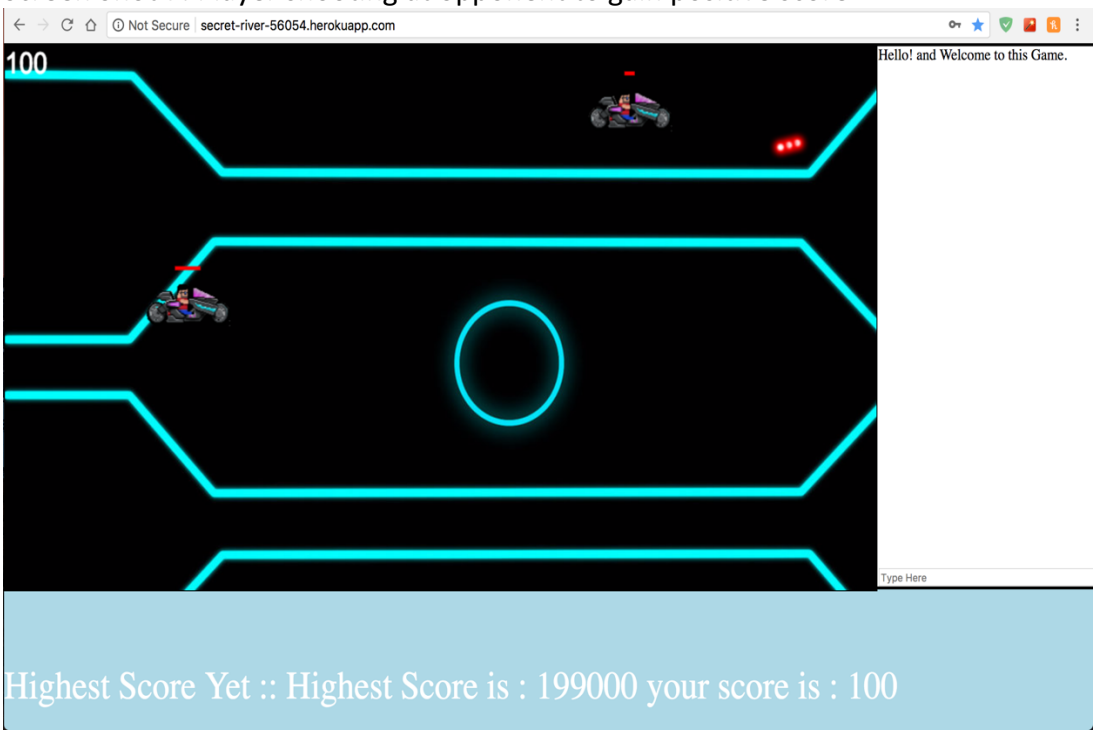
Screen Shot 5: Game window with player receiving penalty for going off screen



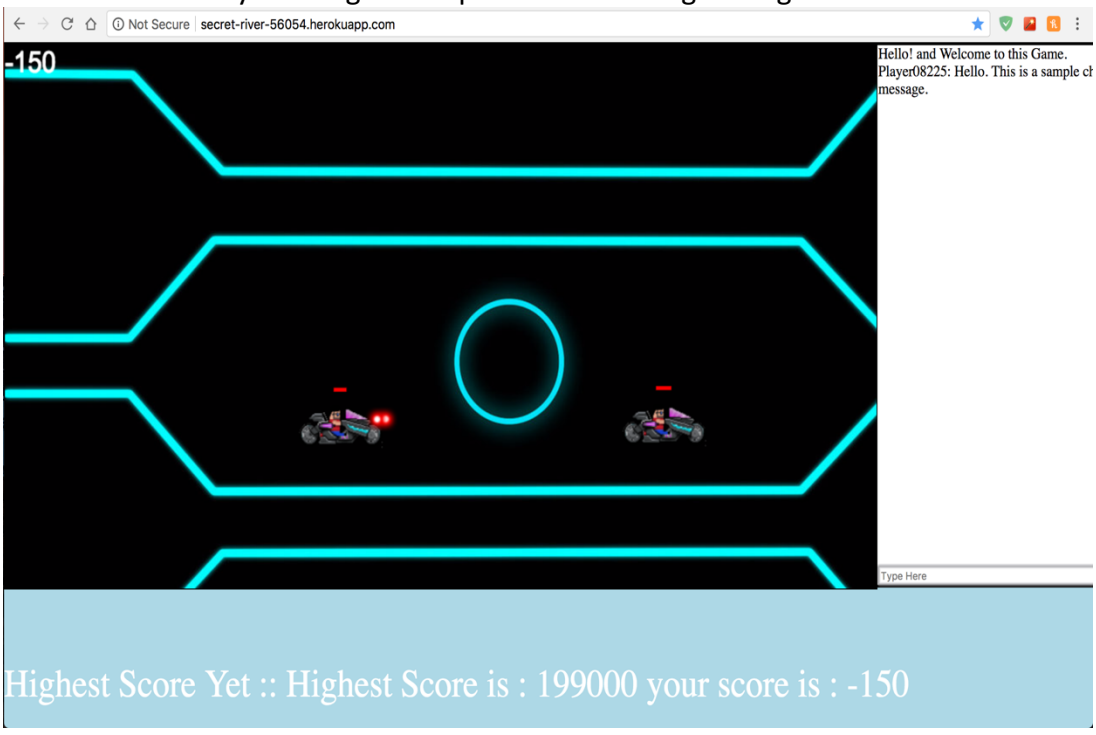
Screen Shot 6: Game window with player losing health from bullet



Screen Shot 7: Player shooting at opponent to gain positive score

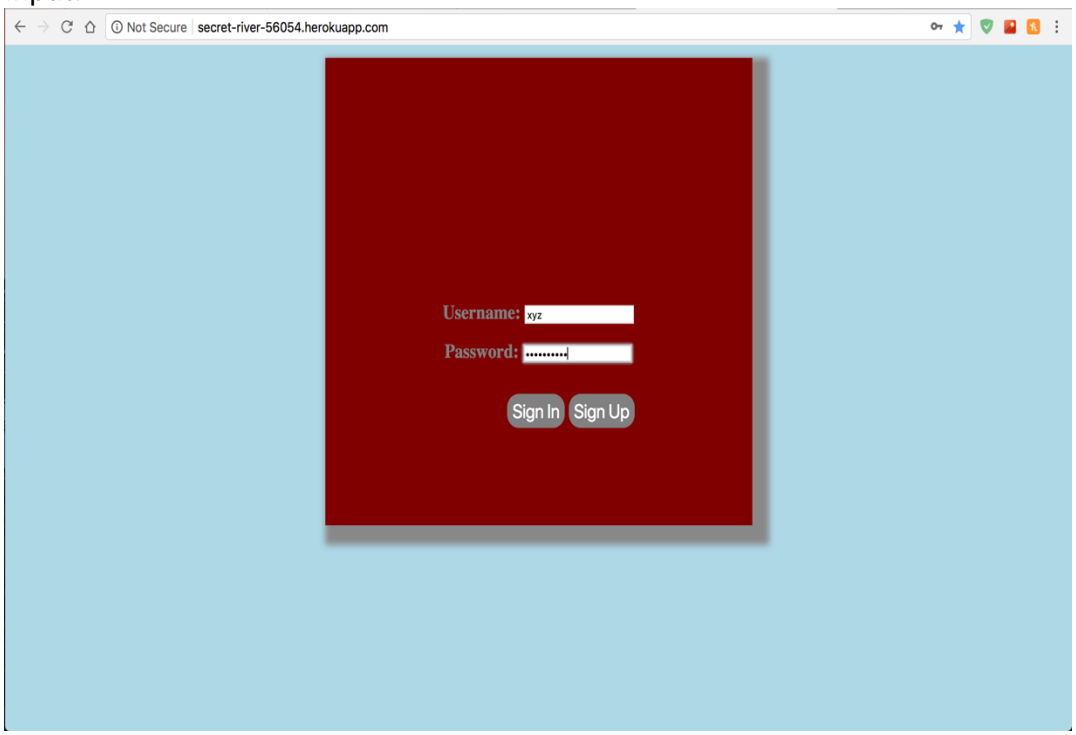


Screen Shot 8: Player losing health points after taking damage



7. Code Testing
   a. Correctness (five cases)

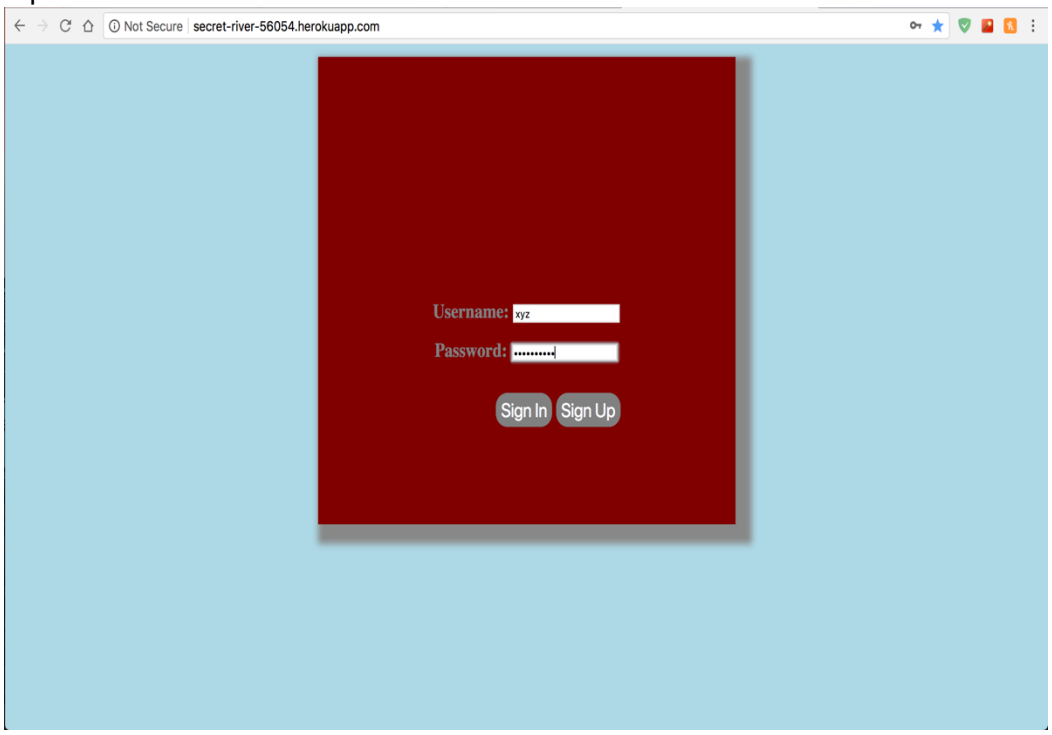
Case 1: Valid email and pass used for login
Input:



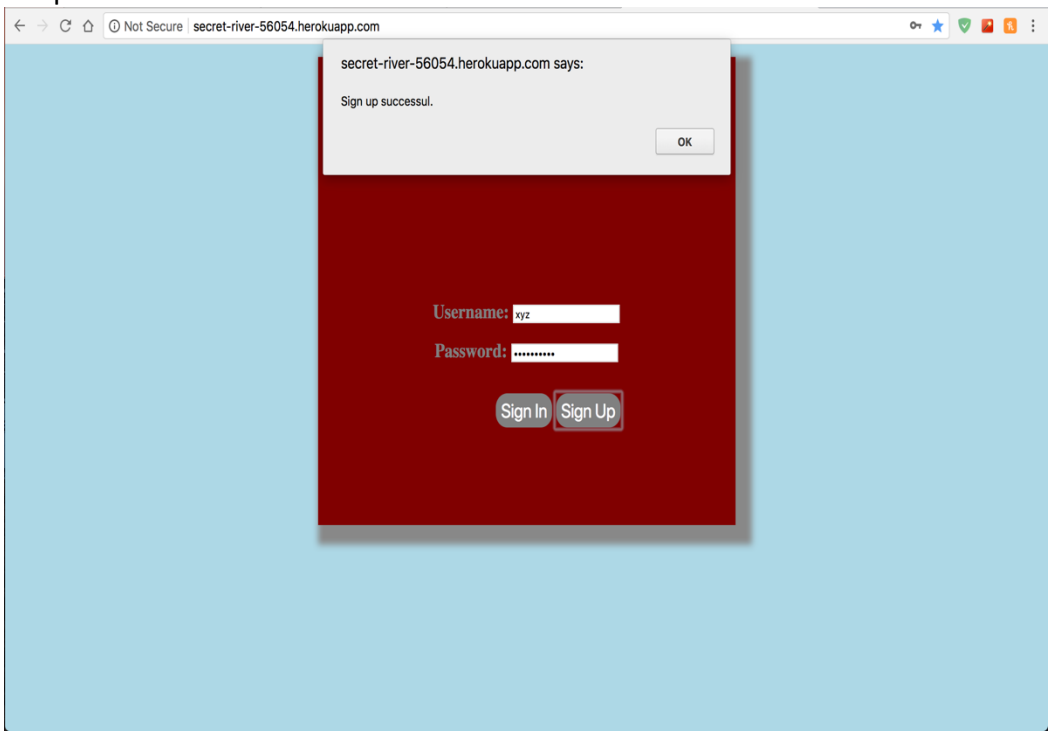
Output:

Case 2: Valid email and password used for account creation

Input:

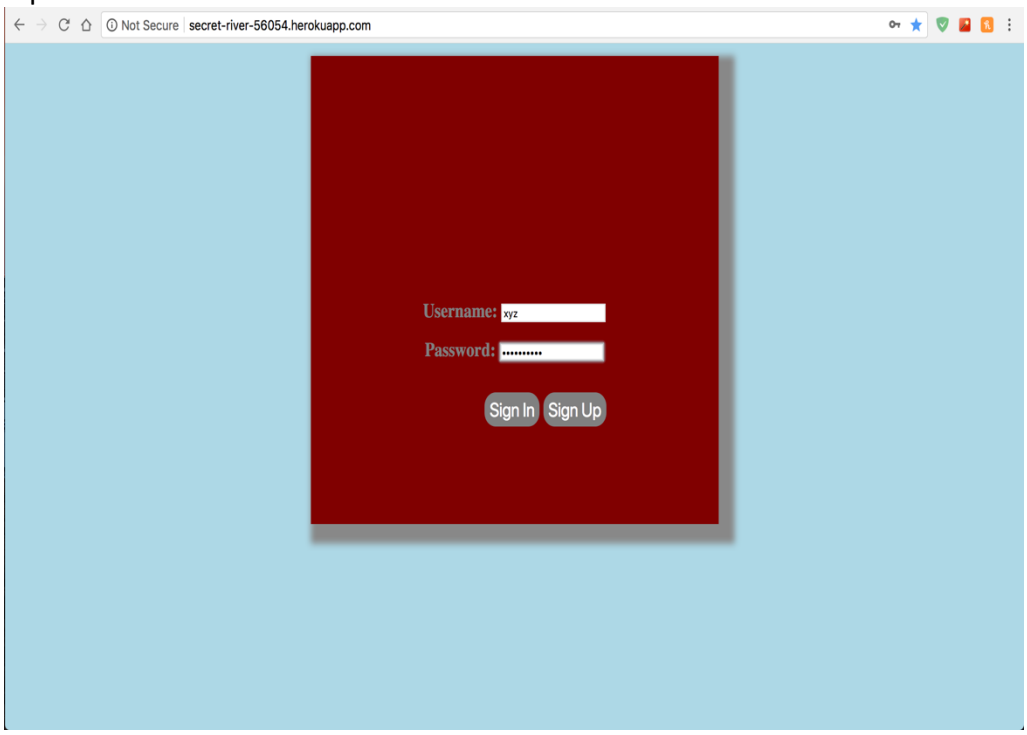


Output:



Case 3: Valid password used without any space

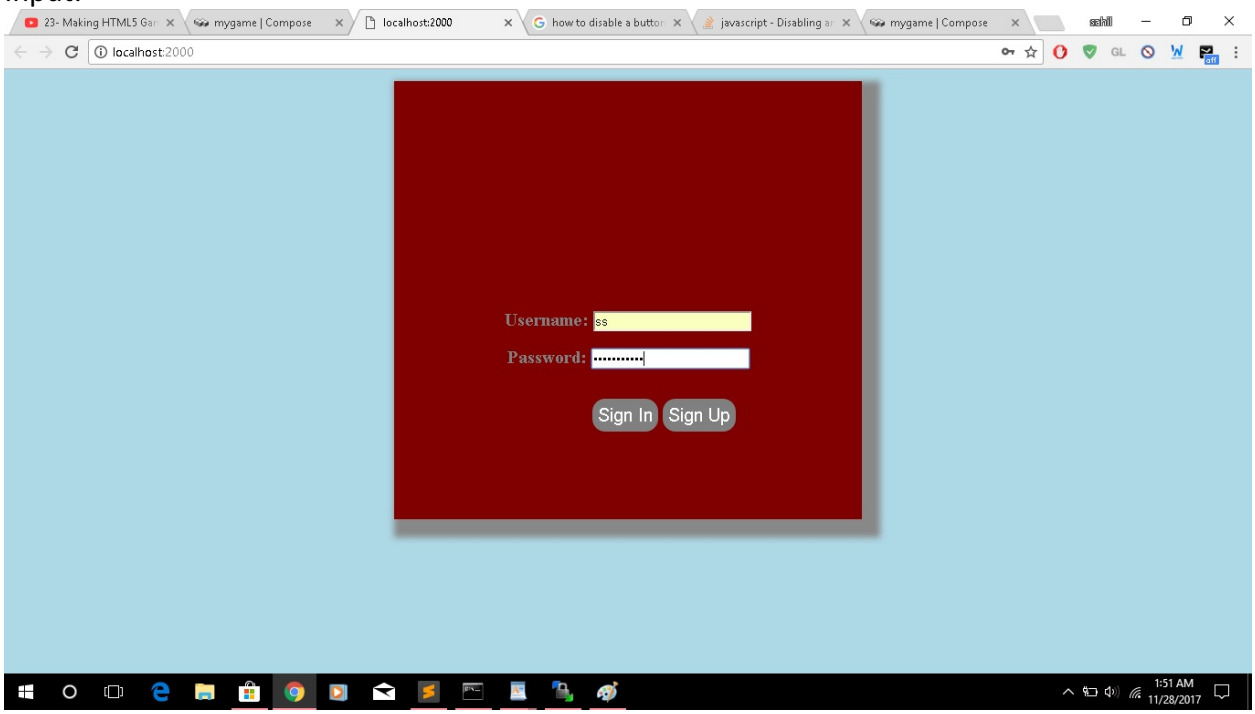
Input:



Output:

Case 4: Password has not been left blank

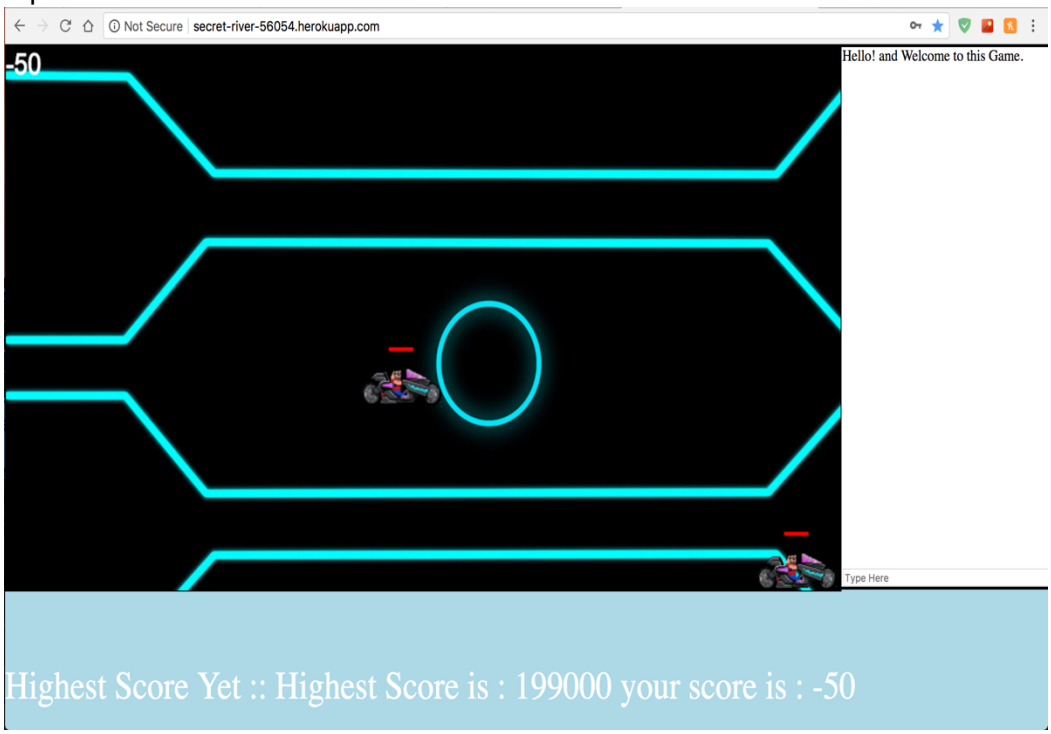
Input:



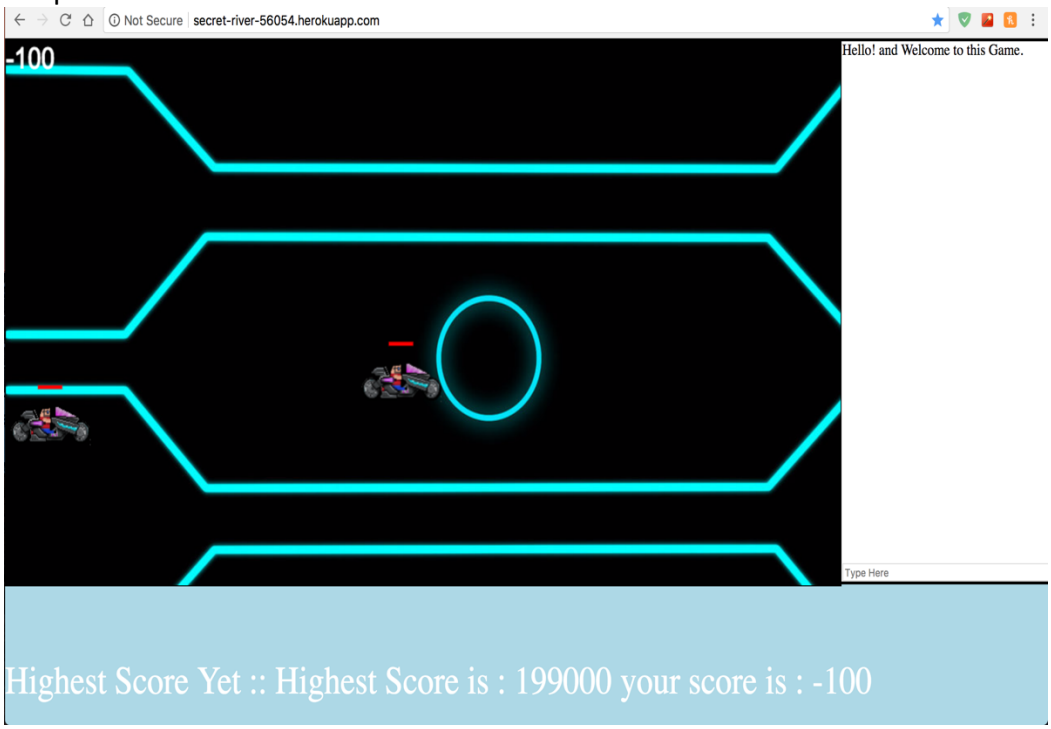
Output:

Case 5: User spawns back on canvas if goes out of game screen

Input:

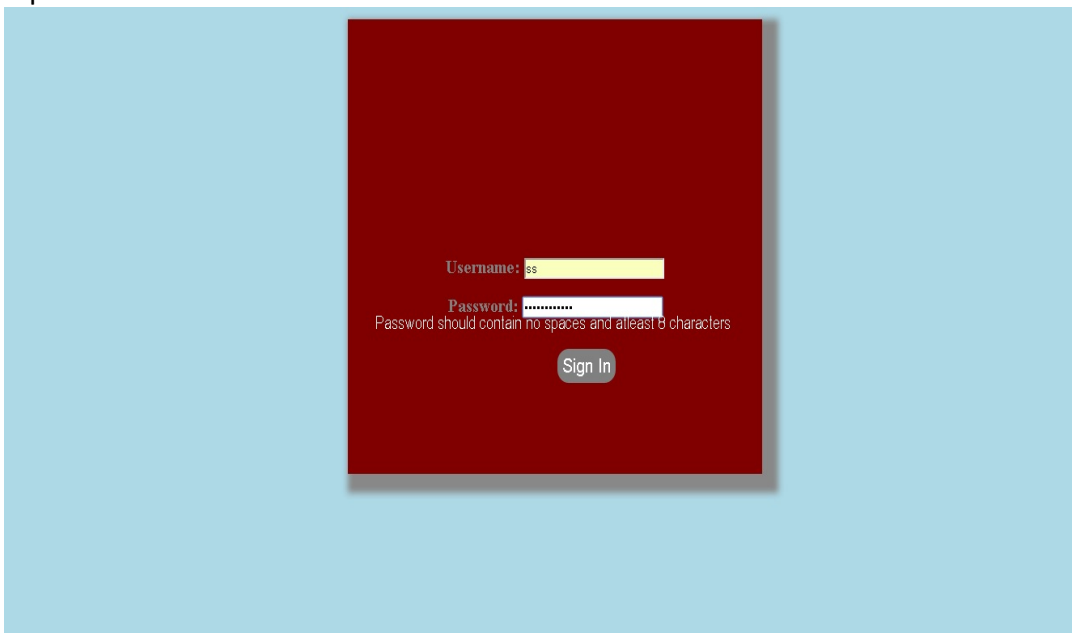


Output:

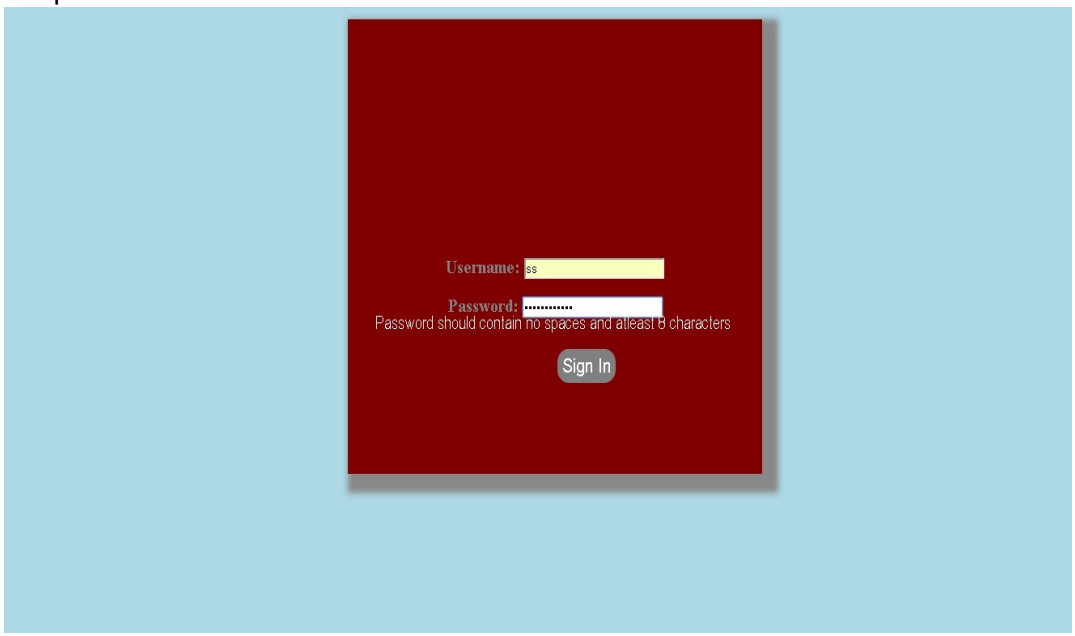


b. Robustness (five cases)

Case 1: No space or less than 8 characters
Input:

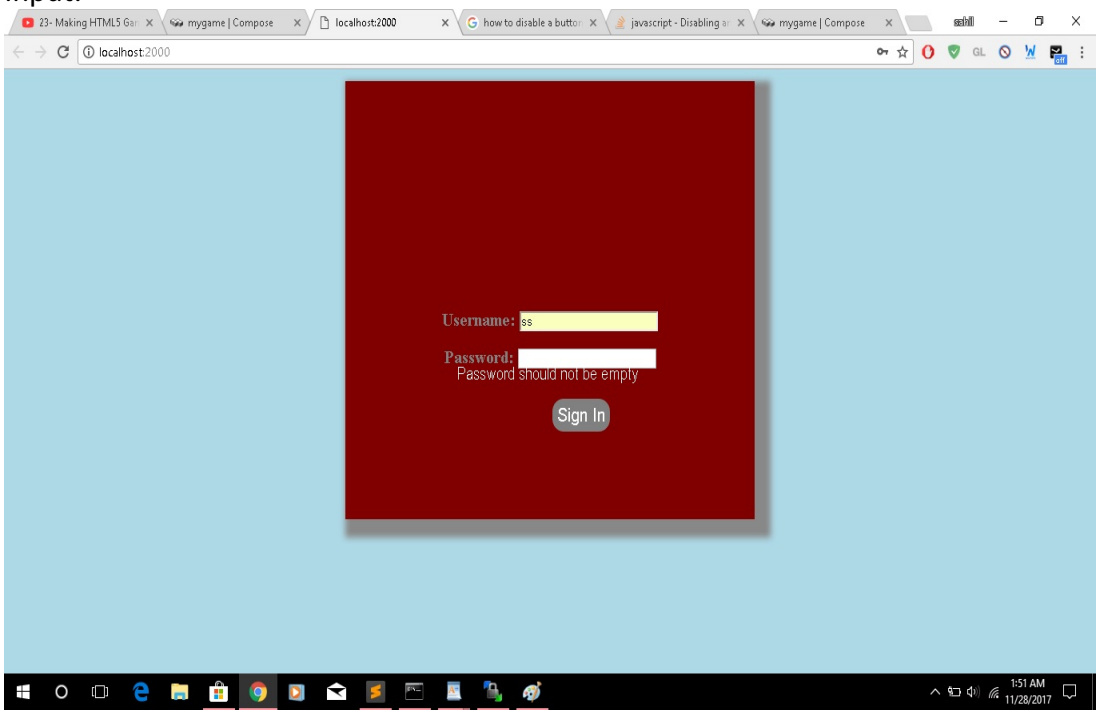


Output:

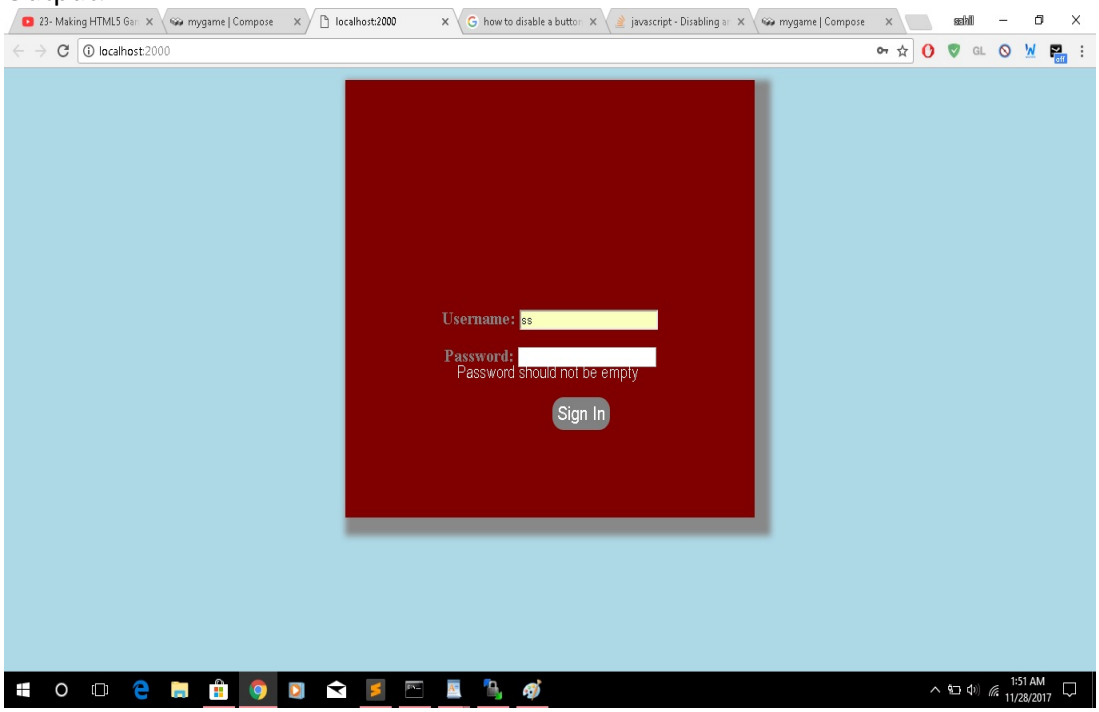


Case 2: Password cannot be empty

Input:

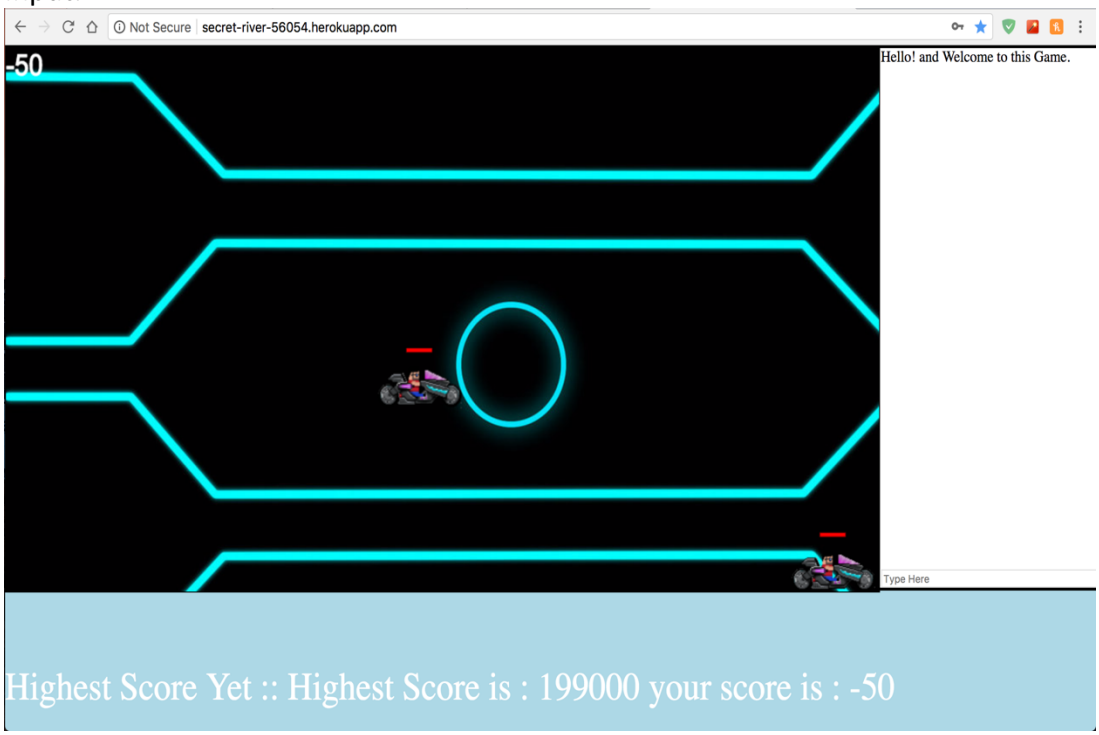


Output:

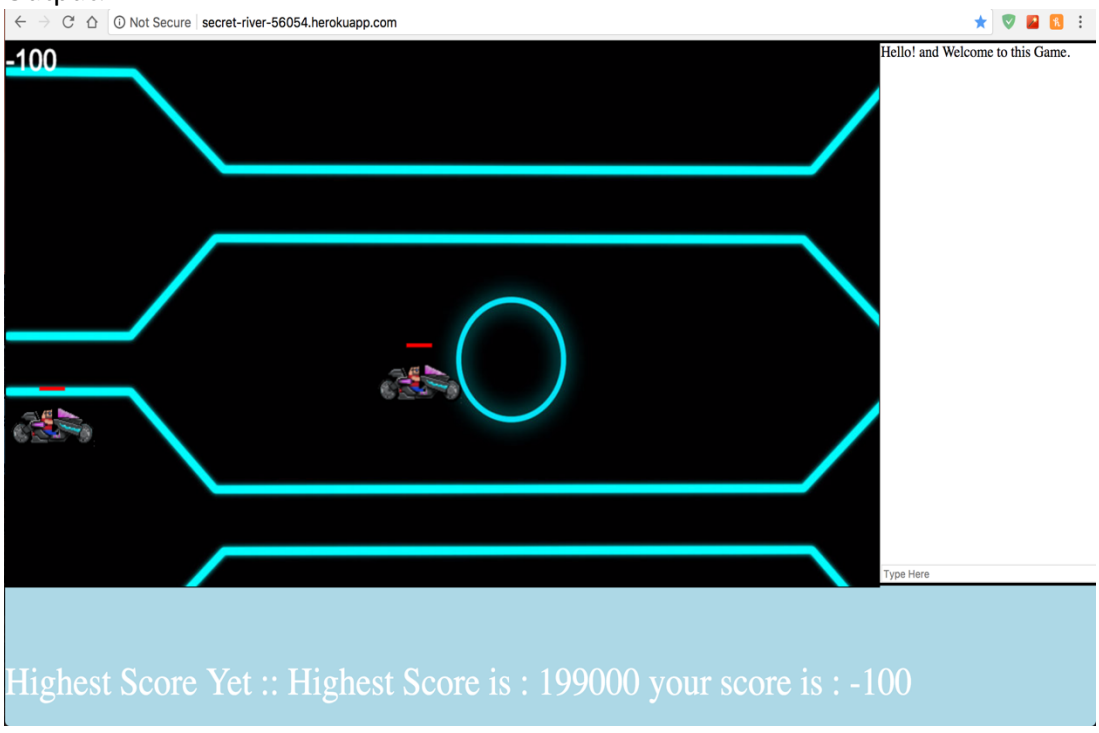


Case 3: Player returns back to screen if accidentally goes off screen and gives a penalty

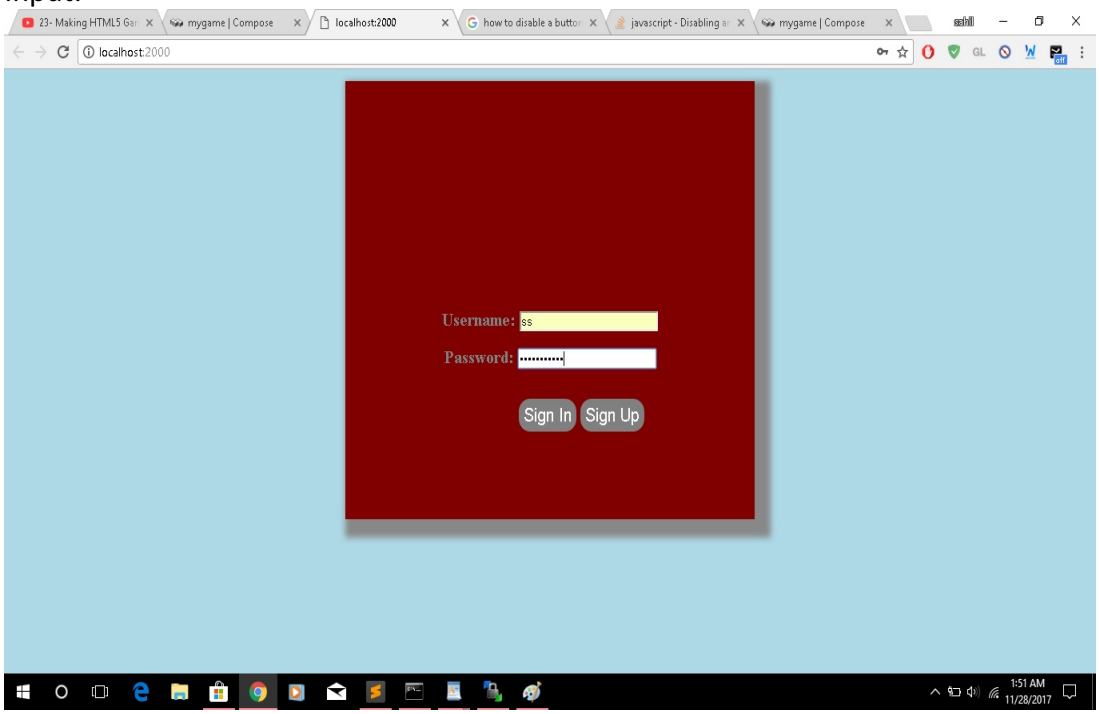
Input:



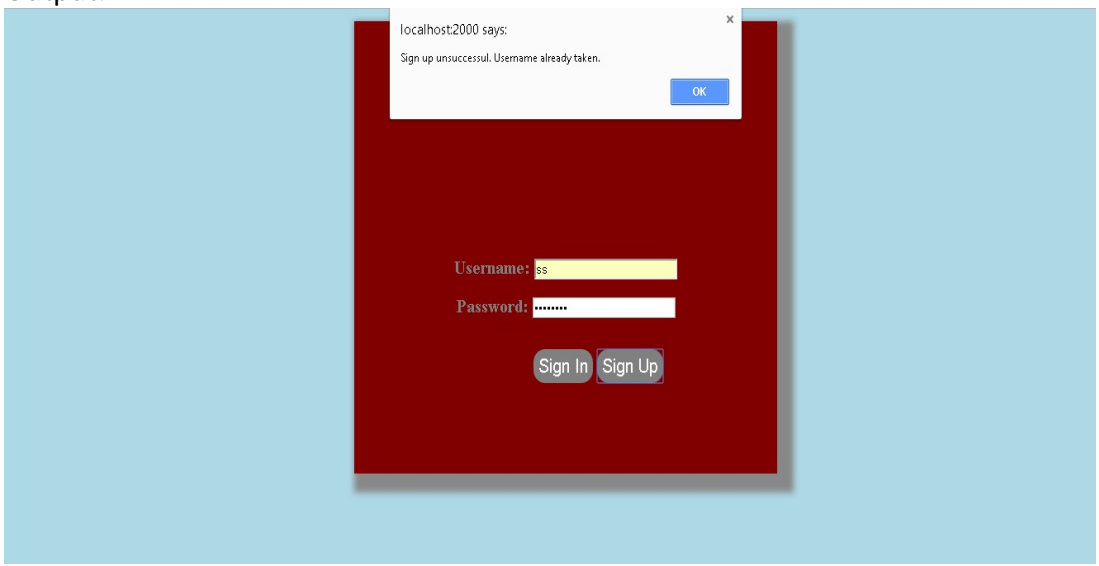
Output:



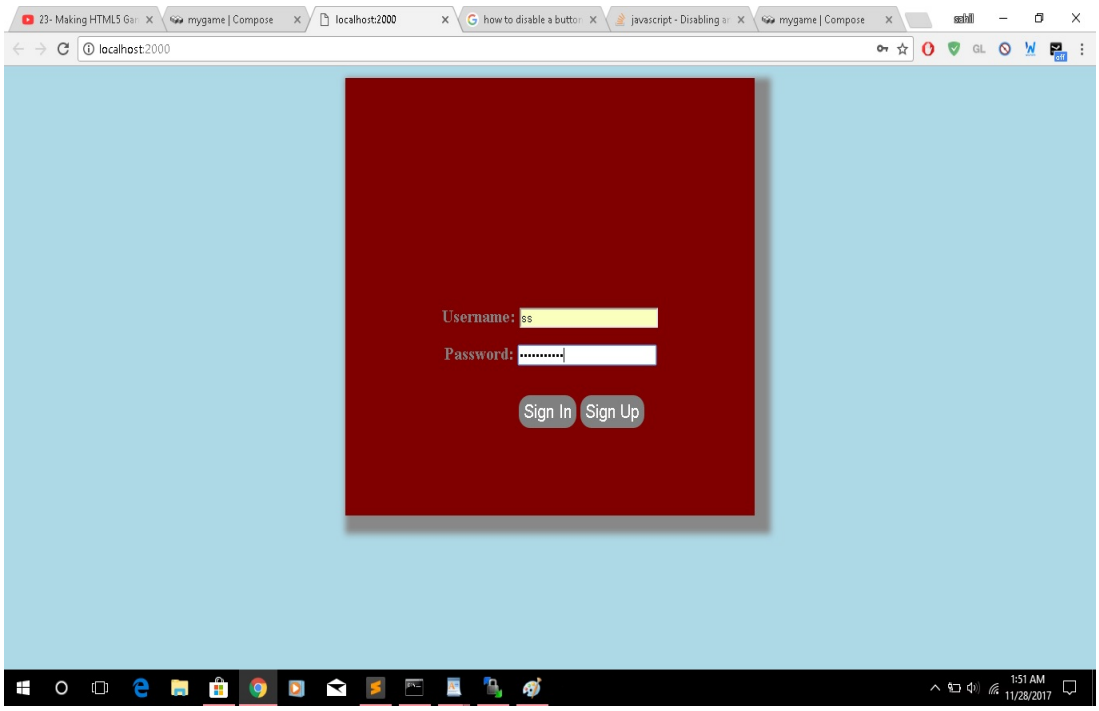
Case 4: User account already exists
Input:



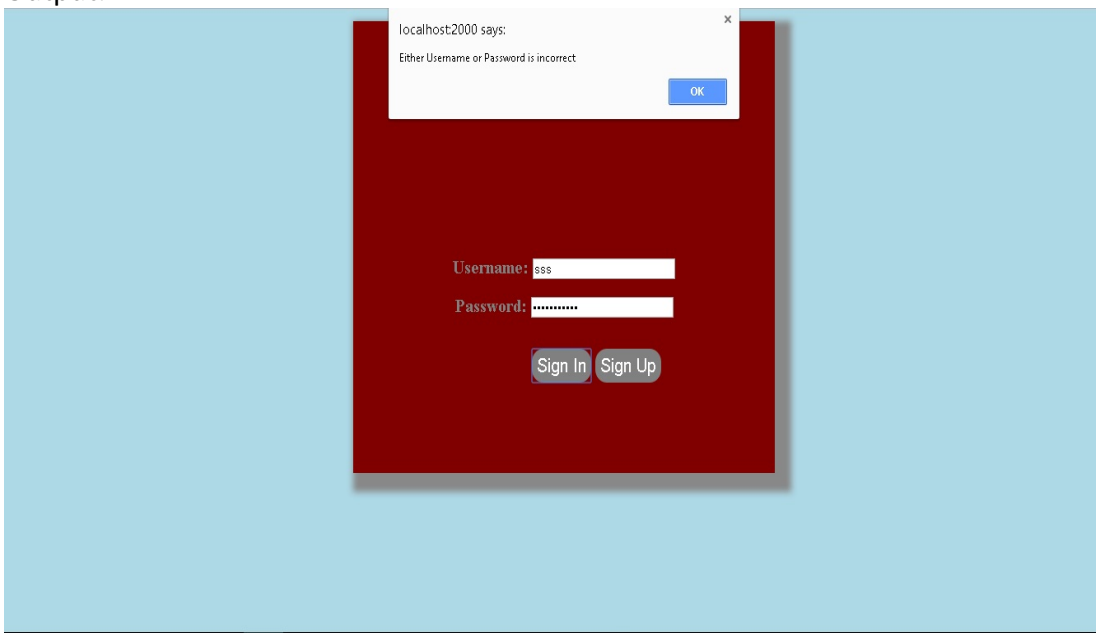
Output:



Case 5: Username and password do not match with database credentials
Input:


localhost:2000
Username: ss
Password:
Sign In
Sign Up
1:51 AM
11/28/2017

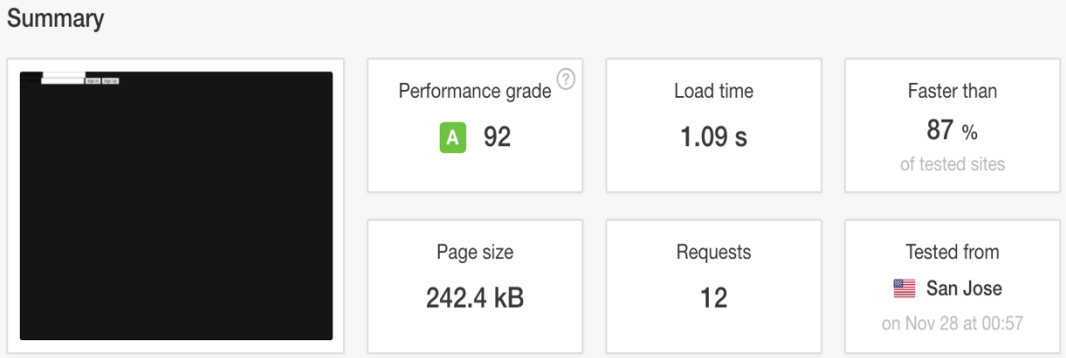
Output:

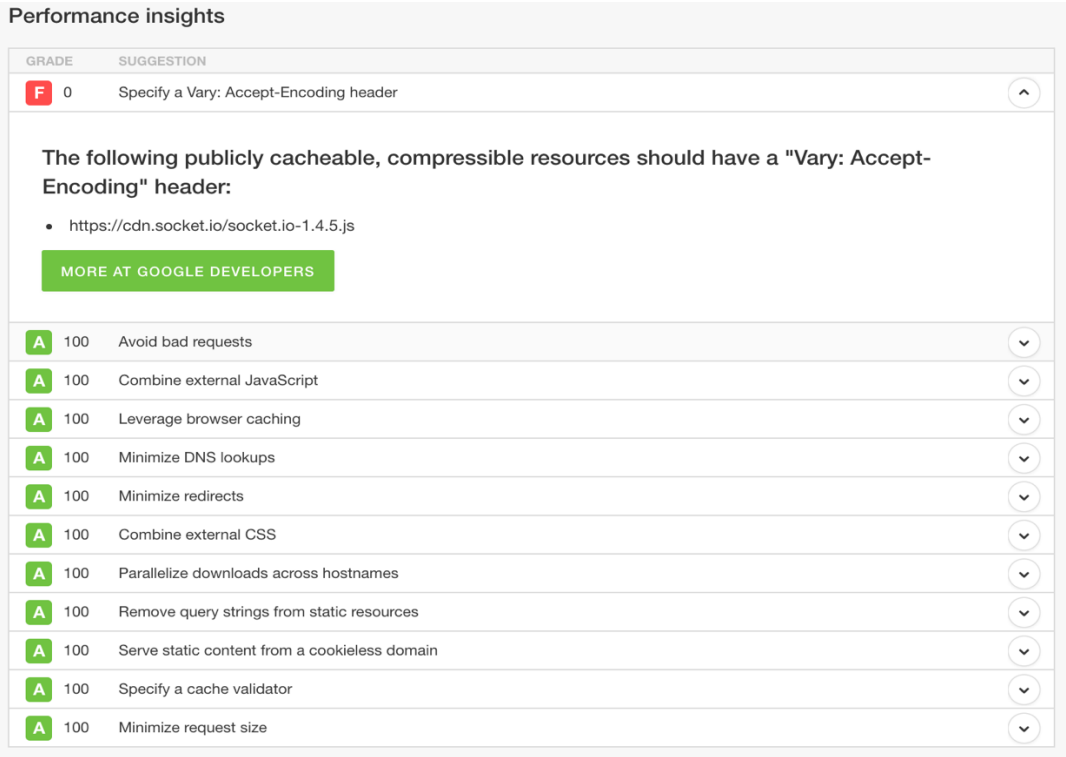

localhost:2000 says:
Either Username or Password is incorrect
OK
Username: sss
Password:
Sign In
Sign Up

c. Performance testing with some benchmarks

For testing the performance of our website we used two different benchmark tests.

1. Pingdom Tools Benchmark Test
   https://tools.pingdom.com/#!/dR7UI6/https://secret-river-56054.herokuapp.com/





Response codes

RESPONSE CODE	RESPONSES
200 OK	12

Content size by content type

CONTENT TYPE	PERCENT	SIZE
Image	55.1 %	133.47 KB
Script	42.8 %	103.72 KB
Plain text	1.0 %	2.44 KB
CSS	0.6 %	1.42 KB
HTML	0.6 %	1.34 KB
Total	100.00 %	242.39 KB

Requests by content type

CONTENT TYPE	PERCENT	REQUESTS
Plain text	41.7 %	5
Image	25.0 %	3
Script	16.7 %	2
HTML	8.3 %	1
CSS	8.3 %	1
Total	100.00 %	12

Content size by domain

DOMAIN	PERCENT	SIZE
secret-river-56054.herokuapp.com	60.5 %	146.53 KB
cdn.socket.io	39.5 %	95.86 KB
Total	100.00 %	242.39 KB

Requests by domain

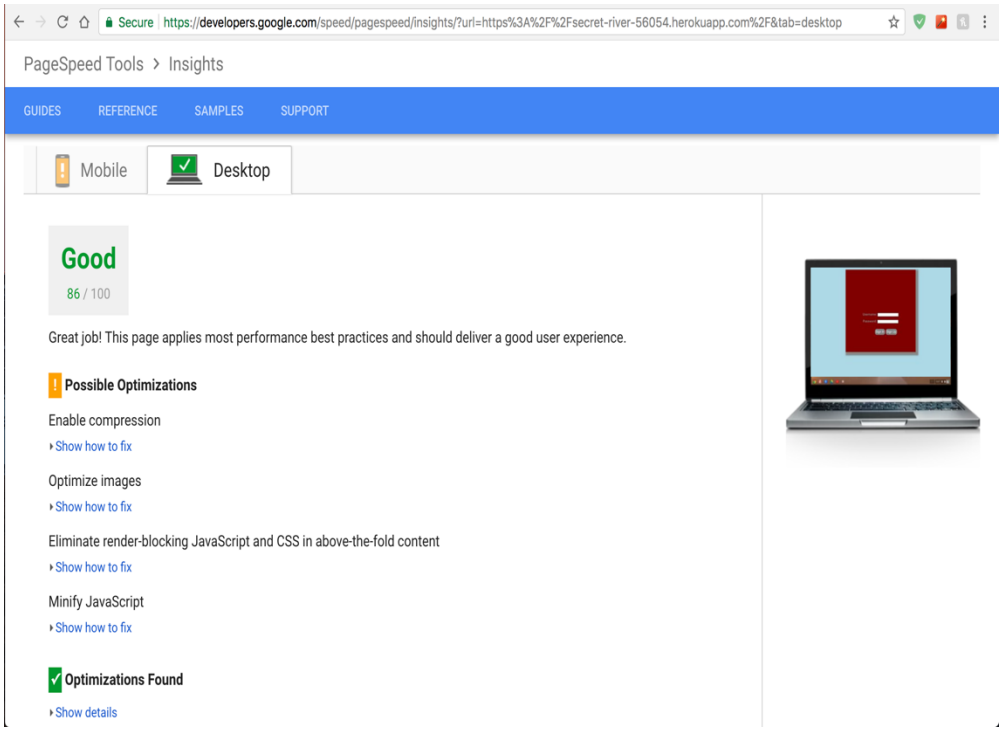
DOMAIN	PERCENT	REQUESTS
secret-river-56054.herokuapp.com	91.7 %	11
cdn.socket.io	8.3 %	1
Total	100.00 %	12

File requests

Sort by Load order Filter DNS SSL Send Wait Receive Connect

FILE	SIZE	0.0s 0.2s 0.4s 0.6s 0.8s 1.0s
https://secret-river-56054.herokuapp....	1.3 kB	
mystyle.css secret-river-56054.herokuapp.com/css/	1.4 kB	
socket.io-1.4.5.js cdn.socket.io/	95.9 kB	
client.js secret-river-56054.herokuapp.com/	7.9 kB	
https://secret-river-56054.herokuapp....	365 B	
newbike.png secret-river-56054.herokuapp.com/img/	5.3 kB	
beams.png secret-river-56054.herokuapp.com/img/	54.1 kB	
lol.jpg secret-river-56054.herokuapp.com/img/	74.0 kB	
https://secret-river-56054.herokuapp....	745 B	
https://secret-river-56054.herokuapp....	382 B	
https://secret-river-56054.herokuapp....	503 B	
https://secret-river-56054.herokuapp....	503 B	
12 requests	242.4 kB	1.09 s

2. PageSpeed by Google Developers
   https://developers.google.com/speed/pagespeed/insights/?url=https%3A%2F%2Fsecret-river-56054.herokuapp.com%2F&tab=desktop



8. A document that clearly indicates the completed tasks of each member
    a. Dhvanil Patel
        - Assisted in most of the front end and some back end development such as creating the HTML and CSS documents as well as maintaining them according to the progress of the Project in addition to helping put the JavaScript together to allow validation for login and signup. Also assisted in putting the Project Report together with the help of other group members.
        - HTML
            - Created the login and sign-up content on the index.html page
        - CSS
            - Created the CSS for the layout of the website
        - JavaScript
            - Added functionality for validating the login and signup
        - Project Report
            - Software Requirements Document
            - Technical Documentation
            - Design specification document
            - Final proof reading of the whole project and report
    b. Karan Warraich
        - Assisted in most of the back end development such as creating JavaScript and Node.js functions as well as assist Sahil Verma in setting up mongoDB. Also assisted in putting the Project Report together with the help of other group members.
        - JavaScript
            - Created code for the backend which allows the validation process to work for allowing the canvas to present playing functionality for multiple players.
        - Node.js
            - Created code to support client and server side connections to further allow multiple player functionality for the web application.
        - mongoDB
            - Setup the database to support the Node.js functionalities for allowing players to interact with the server and database accordingly.
        - Project Report
            - Design specification document
            - Code Description
            - Code Testing
            - Final proof reading of the whole project and report
    c. Cory Lausch
        - Assisted in most of the front end development such as creating and maintaining the HTML and CSS documents to suit application needs as well as assist in putting the Project Report together.
        - HTML

        - Assisted Dhvanil Patel in creating the login and sign-up content on the index.html page
    - CSS
        - Adjusted CSS properties to suit the needs of the web application
    - Node.js
        - Created code to support client and server side connections to further allow multiple player functionality for the web application.
    - mongoDB
        - Setup the database to support the Node.js functionalities for allowing players to interact with the server and database accordingly.
    - Project Report
        - Design specification document
        - Technical Documentation
        - Code Testing
        - Final proof reading of the whole project and report

d. Sahil Verma
    - Assisted Karan Warraich in most of the back end development such as creating and deploying the Node.js and mongoDB functions as well as use JavaScript to allow the Node.js to allow multiplayer functionality for the game. Also, assisted in the Project Report according to the appropriate section.
    - JavaScript
        - Created code for the backend which allows the validation process to work for allowing the canvas to present playing functionality for multiple players.
    - Node.js
        - Created code to support client and server side connections to further allow multiple player functionality for the web application.
    - mongoDB
        - Setup the database to support the Node.js functionalities for allowing players to interact with the server and database accordingly.
    - Project Report
        - User documentation
        - Code Testing
        - Design specification document
        - Final proof reading of the whole project and report

e. Kevin Peterson
    - Assisted by working on putting project proposal together as well as setting up meetings and planning.