CSC667 Spring 2017

Application: CaeZar

2017/05/20

Team CCZZ

Xinlu Chen (915757497)

Cheng Li(916422876)

Xuan Zhang(916409525)

Lijie Zhou(913821407)

https://github.com/SFSU-CSC-667/term-project-spring-2017-cczz

Table of Contents

1. **Project Architect**

Our project has four basic parts: registration/login, lobby page, game, and chat.

Registration/Login
Users have to register and login to enter the game.

Lobby page
The lobby page has three main parts: scoreboard, game board, and chatboard. The scoreboard is a bulletin board that displays game player's' current score(s). The game board displays the current game rooms that users can enter. If the user doesn't want to enter the current room(s), he/she can choose to create a new room. The chatboard is used for users to post messages. One use case is for users to notify their friends which game to enter.

Game page
After users enter the room, the game page will display. The game room uses canvas to show game interface. In the game page, users can use the "start game" button to start the game. Users can also use chat board to communicate with each other in the game room.

2. **Database diagram**

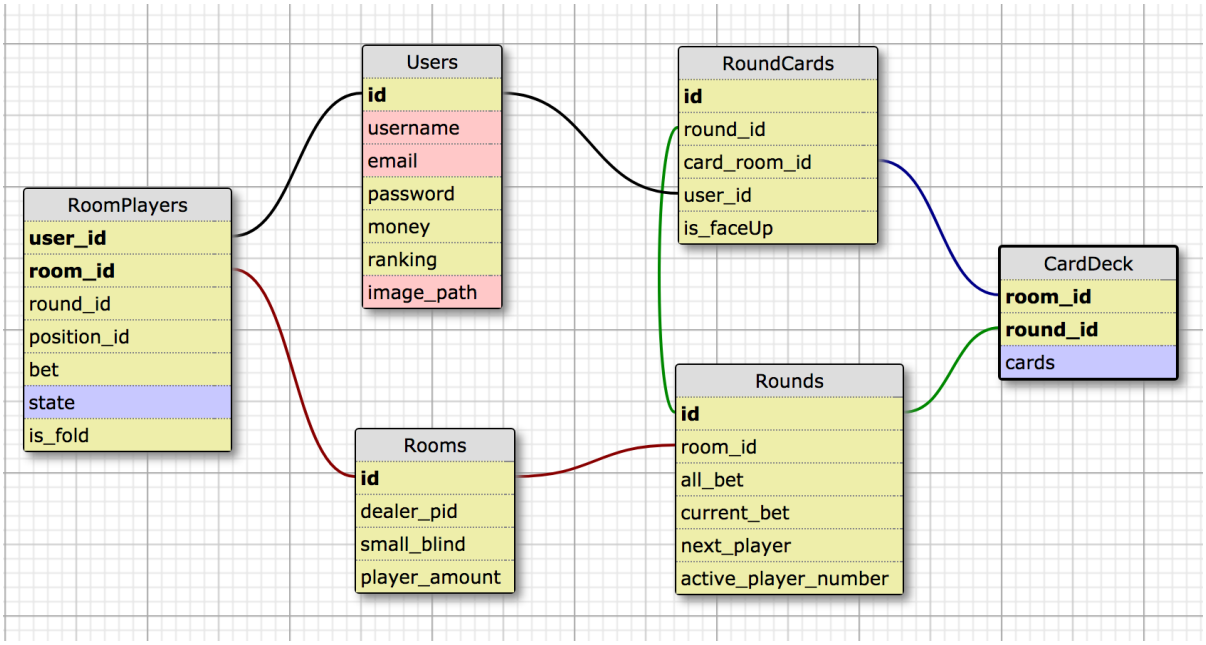

Figure 1 Database Diagram

3. **Technology Stack**

We use code snippets to show the main technologies we used in this project.

3. 1 Frontend: Pug/Bootstrap , CSS, and Javascript/JQuery

We use Pug/Jade to render the views in /views folder. Here below is one example of the lobby.pug file.

```
extends layout

block content
  .container.lobby-body
    #ranking-board.col-md-4
      h3 Score Board
        .score1

      #game-room.col-md-4
        h3 Game Board
        #rooms.list-group


      #chat-board.col-md-4
        h3 Chat Board
        div.display-message.well.well-sm

        .input-group
           input.form-control(type='text',
           placeholder='Message...')
           span.input-group-btn
           button.btn.btn-default(type='button')
           Send

```

Figure 2 lobby.pug

In lobby.pug, we use Bootstrap to layout three columns as seen in *col-md-4*. The pug file is a skeleton of the user interface. We use JQuery to append the dynamic data on the pug file.

```
/*Fill the score board and room board*/
$.get("/api/rooms", function (data, status) {
  for (var i = 0; i < data.length; i++) {
    $('#rooms').append(listItem(data[i]));
  }
});
```

Figure 3 lobby.js

3. 2 Backend: NodeJS, Express, Postgresql

We use socket to handle in-app communication. There are two features that use socket intensively: first, user message exchange both in lobby and game room. The second is game state update. Below, we will use the message exchange as an example to explain.

The /socket/index.js file handles all the server side socket code.

```
26  const init = function (app, server) {
27    const io = socketIo(server); // the websocket connection
28
29    //app.set('io', io);//useless
30
31    io.sockets.on('connection', function (socket) {
32      /*socket on the message from the lobby*/
33      socket.on('message', function (data) {
34        io.emit( 'message-display', data );
35        // console.log(data.data);
36      });
37      /*socket on the message from the room*/
38      socket.on('room-message', function(data){
39        // console.log(data.roomid);
40        socket.join(data.roomid);
41        io.sockets.in(data.roomid).emit('room-message-display', data);
42        // console.log(data.data);
43      });
```

Figure 4 /socket/index.js

The client side socket is in javascripts/lobby.js and javascripts/game.js. Here below is the code snippet of how to register the ‘room-message’ event from the client side’s socket.

```
178    /* Room message posted */
179    $('#chat-input button').click(function () {
180      const message = $('.room-form-control').val();
181      // console.log(message);
182      // var username = $.cookie(username);
183      socket.emit('room-message', {roomid: roomid});
184      socket.emit('room-message', {data: message});
185    });
```

Figure 5 /javascripts/lobby.js

The database codes are organized in /db and /model/index.js. /db stores SQL commands that will first create tables and insert records. /model/index.js stores commands that will be used to retrieve data from the database.

4. **Problems solved and challenges**

Problem 1: Heroku Database Migration

At first, we don't know how to migrate Postgresql database to Heroku if our website has already deployed on it. After reading the documentation and the hints John provided on Slack, we figured out a way to create a new postgresql database on Heroku and insert data into it.

Problem 2: Understanding Socket

We spent a lot time researching on how to use socket for in-app communication. One challenge is how to channel messages from the same socket in the server into different rooms on the client's side. For example, when we have to use server side messages back to the room based on its room number, we don't want players from a different room to see the messages from a different room. We found .join() is a method that specifically designed for channeling messages into different channels after extensive research.

Problem 3: Understanding the MVC architect

It took us a while to figure out the MVC architect and how to take data from database to populate on the page. One mistake we made early on is to hard code the pug file. Later, we figured we should first get all the data from the database and "store" it on the endpoints. For example, /api/users have all the records retrieved from the users table from the database. In order to use it from the frontend, we use JQuery .get('/api/users') to get all the records and manipulate it using JQuery. From there, we can dynamically append new records onto the pug file.

Problem 4: Different OS

In our group, we have 4 team members, some of us uses Mac OS and some of us uses Windows. The folder javascript in Mac is 'Javascript' and in Windows is 'javascript'. But in Heroku, it will become 2 folders—'javascript' and 'Javascript' which makes the application not work.

5. **Grading Rubric**

Category	Description	Total	Self Eval
Code Quality	Code is clean, well formatted (appropriate white space and indentation)	5	5
	Classes, methods, and variables are meaningfully named (no comments exist to explain functionality - the identifiers serve that purpose)	5	5
	Methods are small and serve a single purpose	3	3
	Code is well organized into a meaningful file structure	2	2
Documentation	A PDF is submitted that contains:	3	3
	Full names of team members	3	3
	A link to github repository	3	3
	A copy of this rubric with each item checked off that was completed (feel free to provide a suggested total you deserve based on completion)	1	1
	Brief description of architecture (pictures are handy here, but do not re-submit the pictures I provided)	5	5
	Problems you encountered during implementation, and how you solved them	5	5
	A discussion of what was difficult, and why	5	5
	A description of your test plan (if you can't prove that it works, you shouldn't get 100%)	5	5

Figure 6 Grade Rubric

We implemented login/logout, register, create a game, join a game, wait other players to begin the game, click play button to deal cards features. Although we don't have time to finish all the other functions, we learnt a lot from building this project! That is the most important thing to us.

6. **Test Plan**

Login:

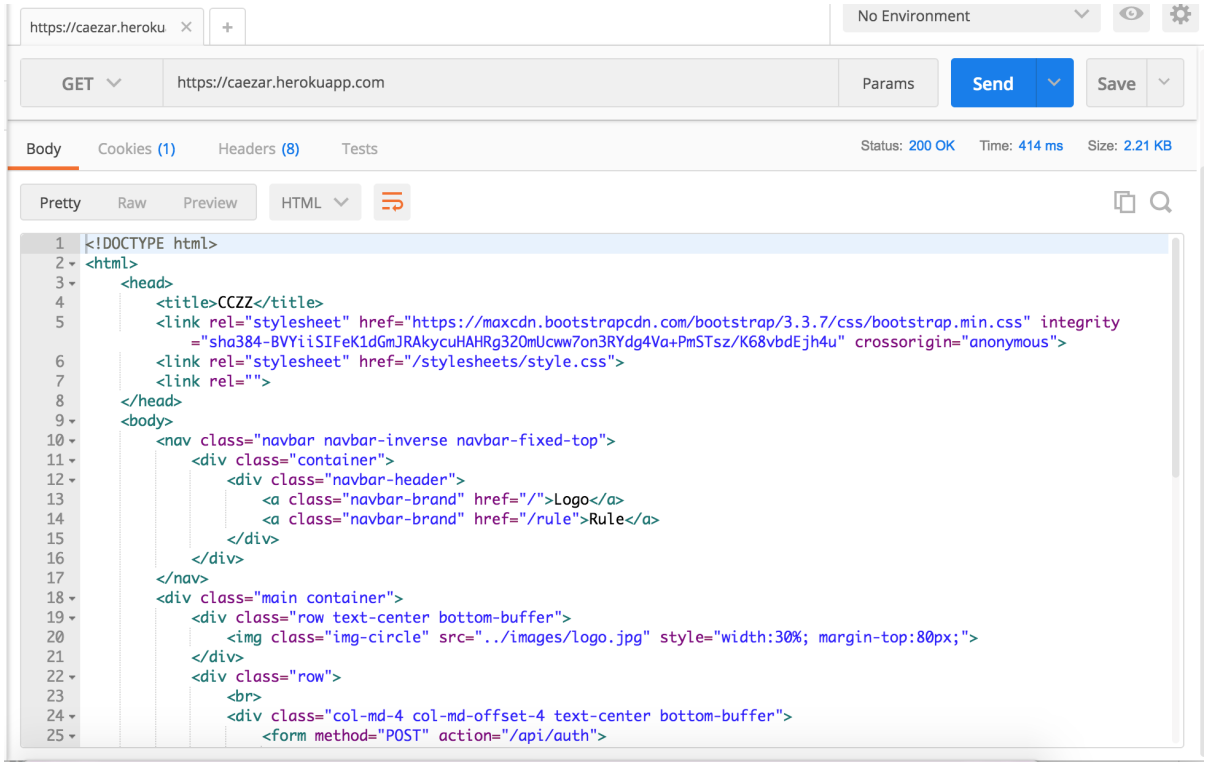

Figure 7 Test GET in Postman

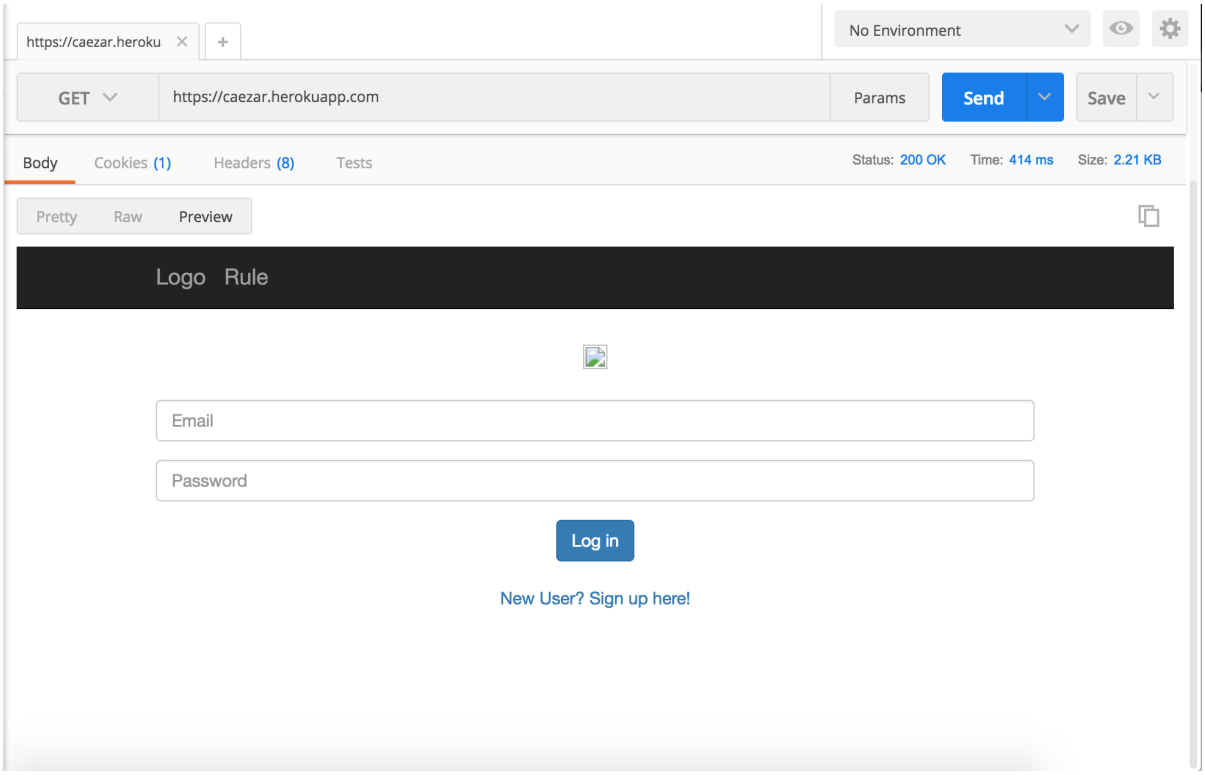

Figure 8 Test Login in Postman

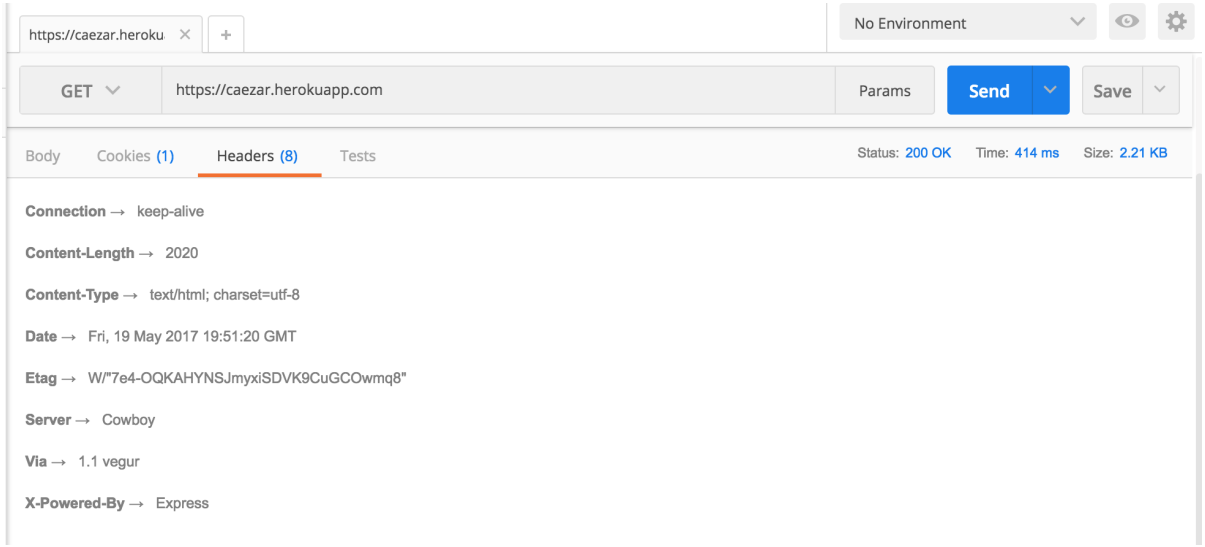

Figure 9 Get Response Back

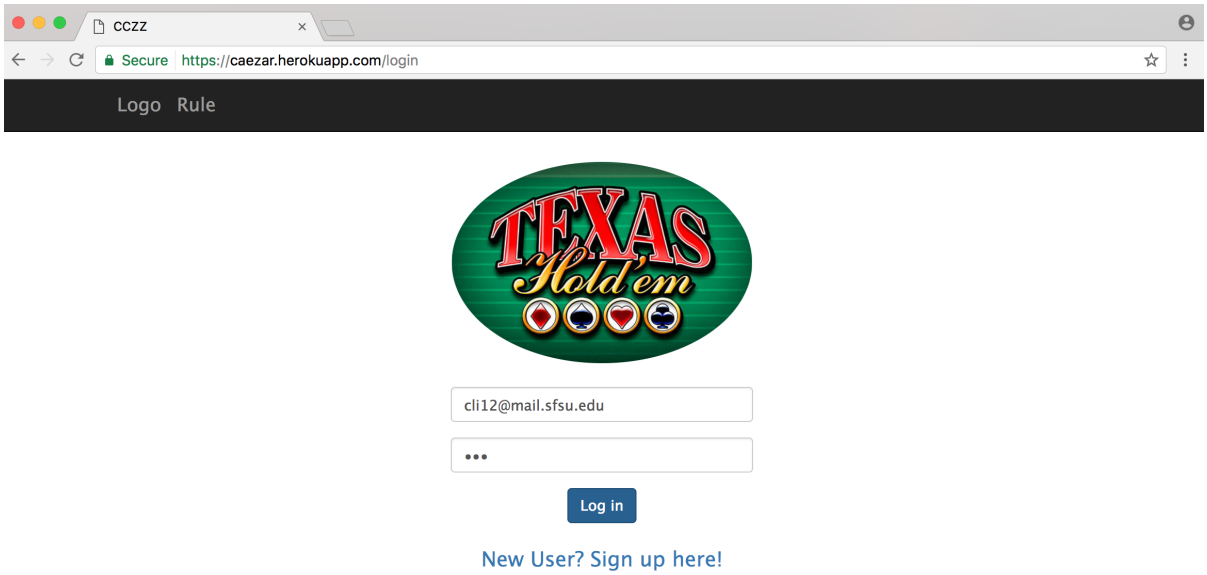

Figure 10 Login Interface

After Login:

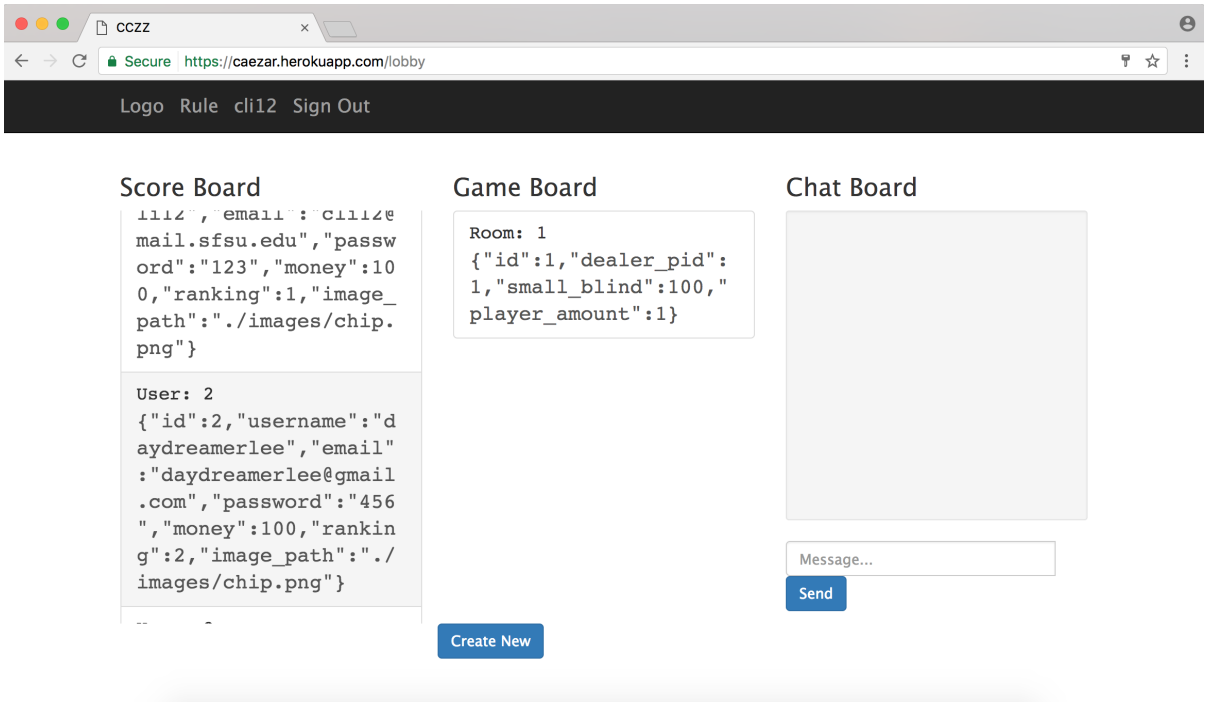

Figure 11 Lobby Interface

Register: After click the “New User” link on the login page:

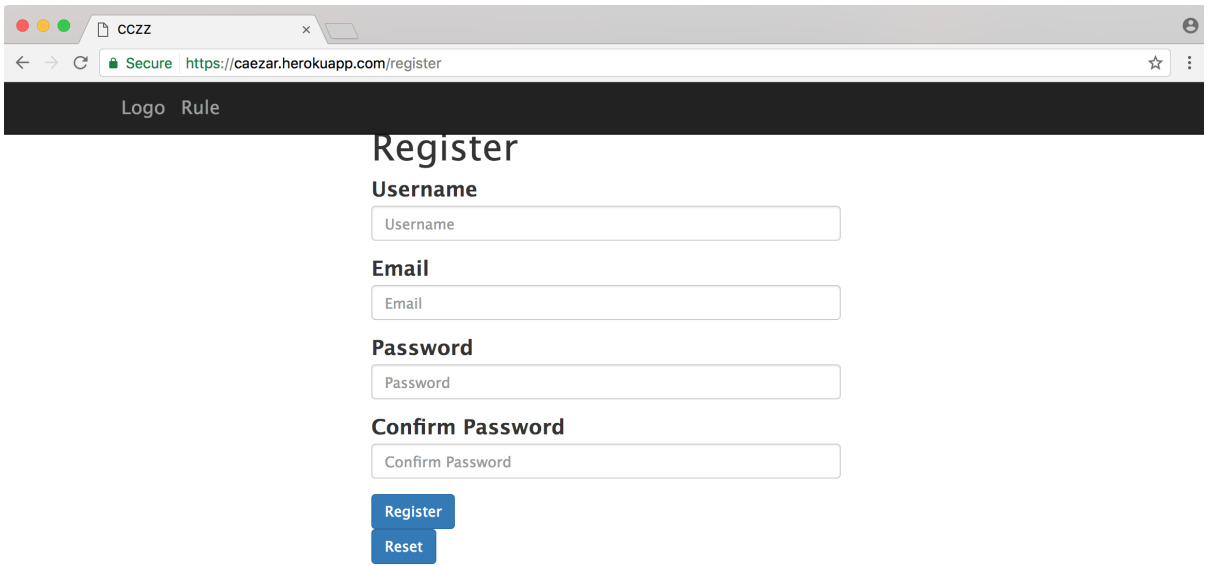

Figure 12 Test Registration

Test user validation: if the confirm password is different from password:

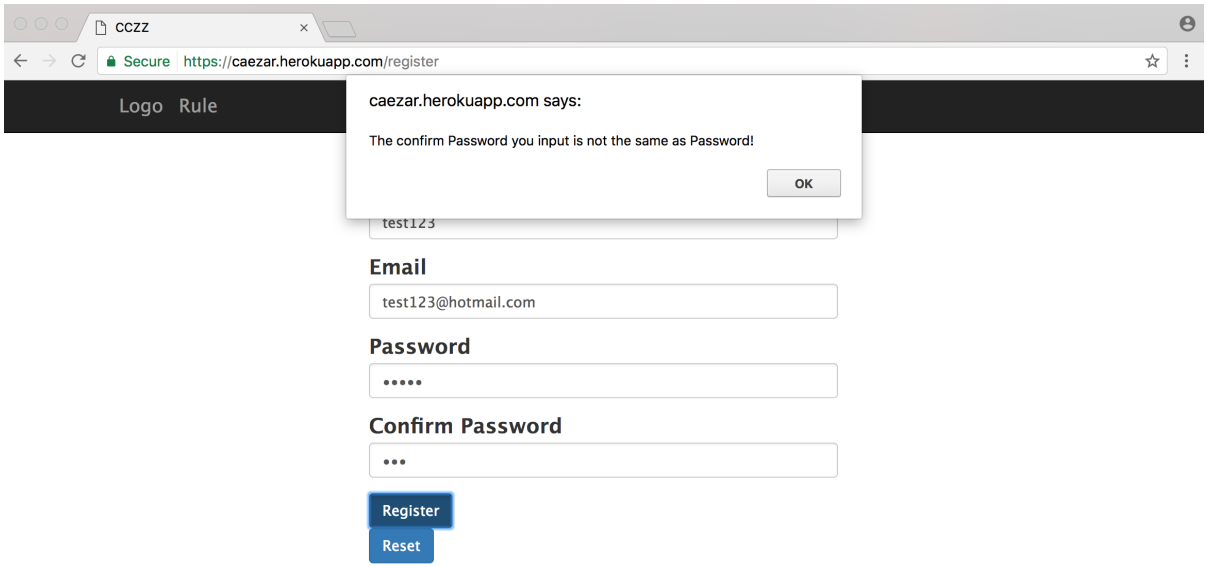

Figure 13 Test Registration Form Validation

There would be an alert window jumping out to tell the warning. And then if the user click OK, the page will stay on the register page.

After successfully register and login, the page will redirect to the lobby page.

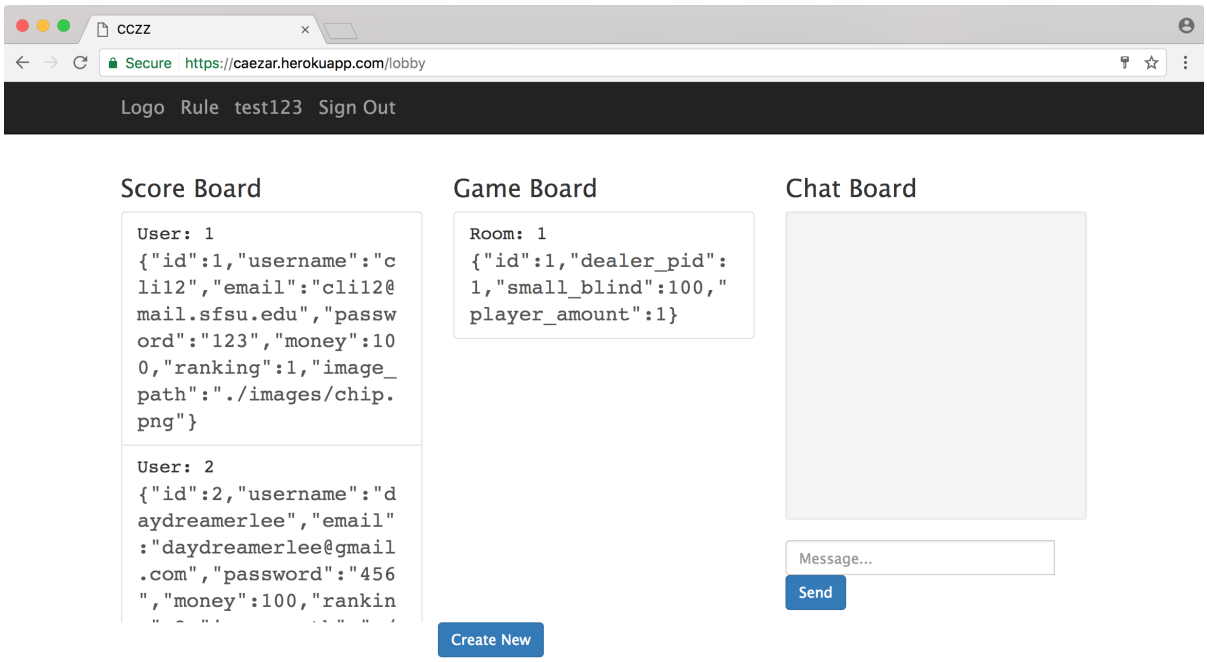

Figure 14 After Registration Direct to Lobby

In Heroku database, the new user has been inserted to the user table: (new user: test123)

```
caezar::DATABASE=> select * from users;
 id |   username    |          email          | password | money | ranking |
image_path
----+---------------+-------------------------+----------+-------+---------+----
---------------
  1 | cli12         | cli12@mail.sfsu.edu     | 123      |   100 |       1 | ./i
mages/chip.png
  2 | daydreamerlee | daydreamerlee@gmail.com | 456      |   100 |       2 | ./i
mages/chip.png
  3 | john          | jrob@sfsu.edu           | 456      |   100 |       3 | ./i
mages/chip.png
  4 | test123       | test123@hotmail.com     | 12345    |   100 |       0 |
  5 | max           | max@gmail.com           | 123      |   100 |       0 |
(5 rows)

caezar::DATABASE=>
```

Figure 15 Test Inserting New User in DB

Click the Rule:

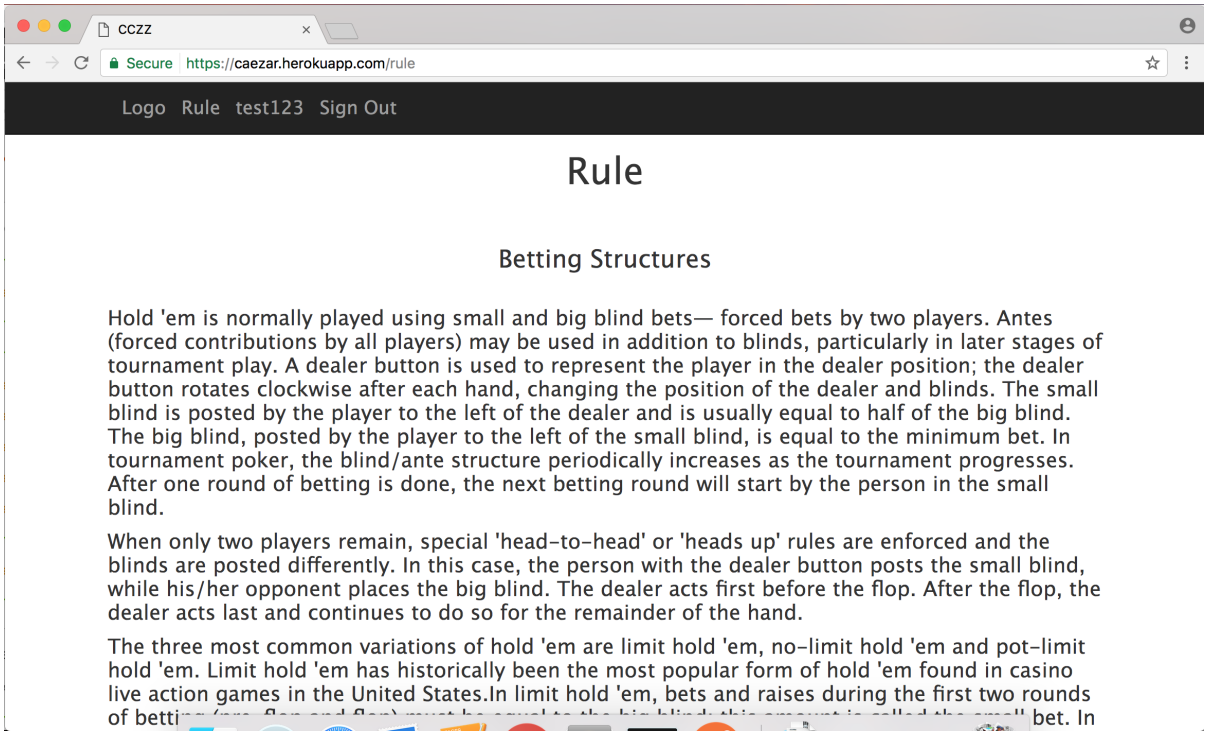

Figure 16 Test Rule Page

After login, test the username link:

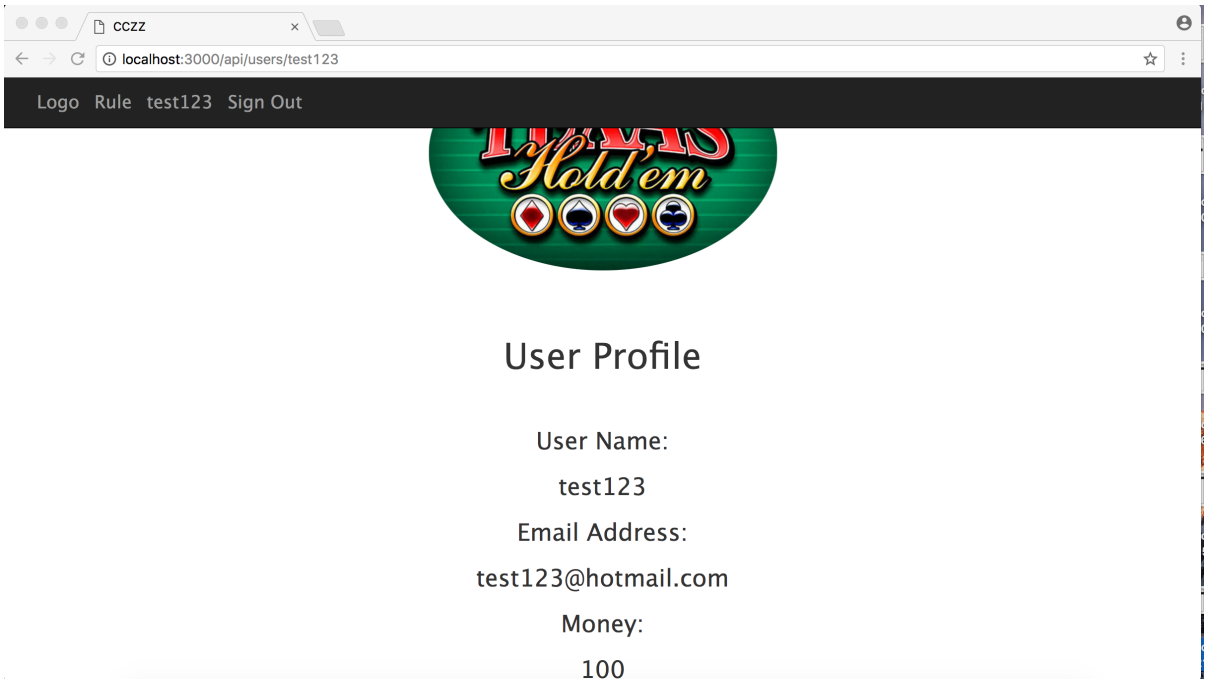

Figure 17 User Profile Interface

Test Sign Out in the navigation bar, the session will be destroyed and the page will be redirect to the login page.

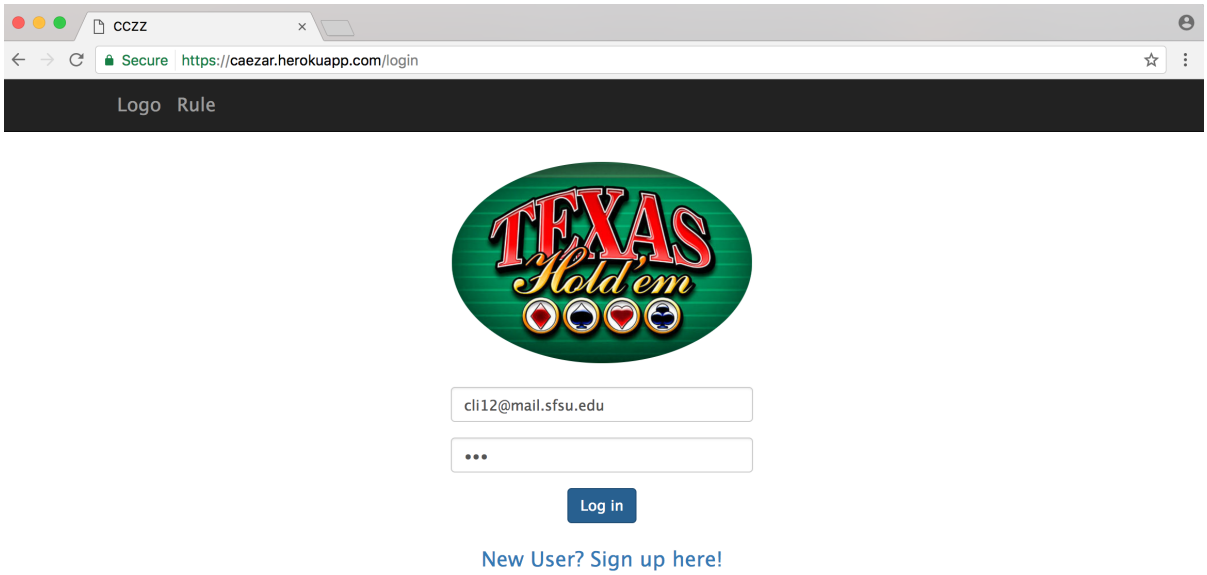

Figure 18 Interface After Signout

After joining the game room:

Figure 19 Game Page

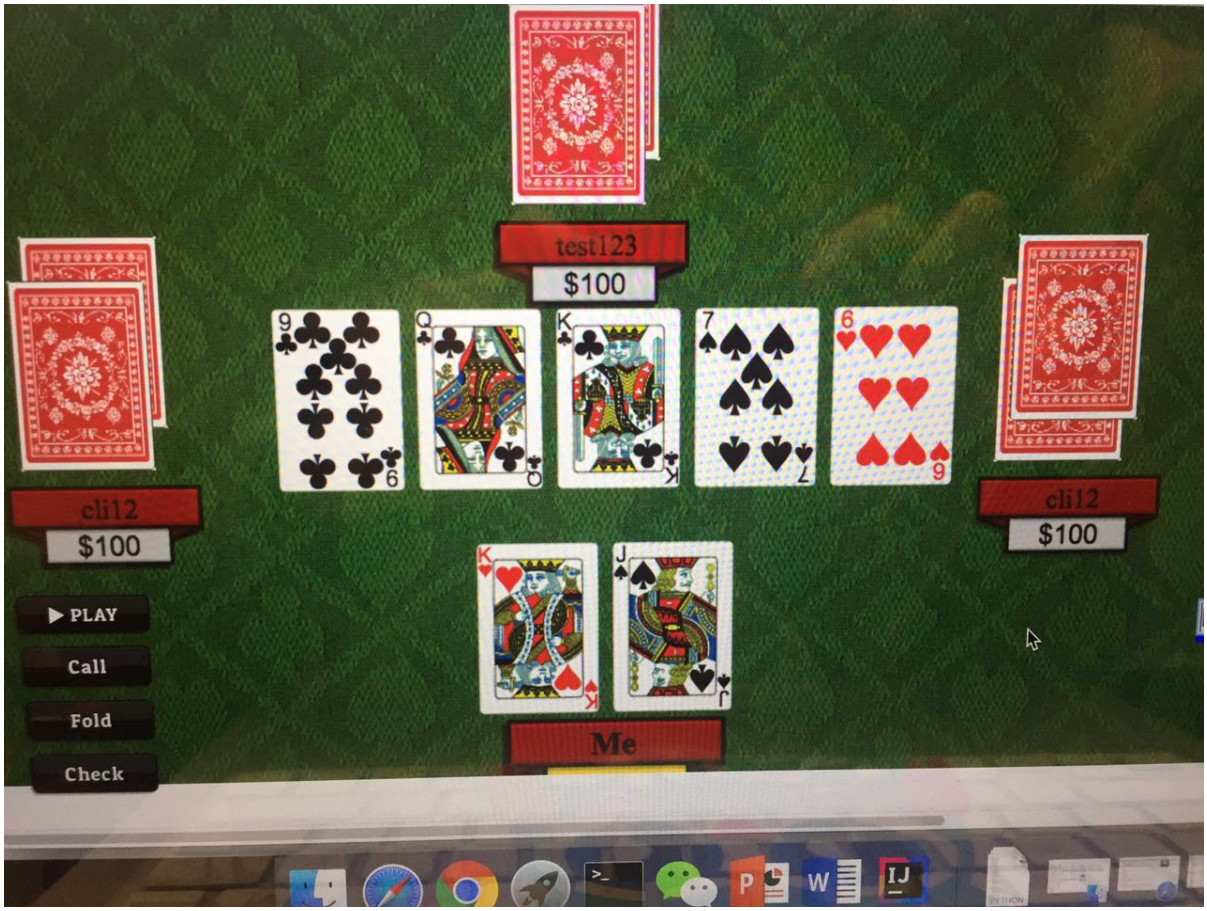

Figure 20 Play Game