CSC667-867 Spring 2019

I. Title Page

Class Section: 02

Team Number: 08

List of Members: **Jianfei Zhao (Team Lead & Back-End Engineer)**
Alan Ng (Front-End Engineer)
Philip Yu (Front-End Lead)
Hang Li (Back-End Lead)
Eric Chen (Front-End Engineer)

Milestone: 5

Due Date: 05/21/2019

Link to Team Repo: https://github.com/csc667-02-sp19/csc667-sp19-Team08

Link to Web Application: http://54.183.220.150/uno/public/

II. Project Introduction

For our team's term project, we have developed a web application iteration of the card game, Uno. Our iteration of the Uno card game, follows the basic structure and rules of that of the official rules of the hand-playing card game. That is, it is a multiplayer game that involves 2 or more players, where each player at the start are dealt 7 cards from the top of the deck. The first player who plays all cards in his/her hand is declared the winner.

The game starts in a direction where each player are assigned with a number when they join the game room. During the game, when that number is called, it will be the turn for the corresponding player. The user who creates the game room is considered as the host player. And such host player will be the person who starts the game or restart the game after it ends. When the game begins, the card from the top of the deck after dealing the cards to the players will be drawn and discarded in the pile. The uno card on game board will be the card that the current player would have to match in either color or number. If the player holds a 'Wild' card in hand, s/he can change the color for the next one to play. If the player can not play a card, they must draw a card from the deck pile and miss his/her turn. A 'Draw 2' card may force the next player to draw 2 cards. Moreover, a 'Wild Draw 4' can both change the color and force the next one to draw 4 cards. Besides, a 'Skip' card can skip the next one’s turn, while a 'Reverse' card can reverse the current turn. All the participating players will try to either play or draw a card. The game will keep moving on until we have a winner or we run out of cards in deck. In order to win the game, the players would have to play their cards wisely and think plan strategically in order to defeat their opponents.

Instead of using any pre-made packages for uno game, we are developing the uno game logic as well as the instant chatting completely by ourselves. Our team’s project includes the implementation of every single event (e.g., typing, stop typing, game relevant events, etc.), which is based on the raw version of Socket.IO.

III. Assumptions for Game Rules

We have made some assumptions for the uno game rules, which may be different from the original ones.
1). If a player has no playable cards, s/he can draw a card from the deck and automatically miss the current turn, even if the card is playable.
2). If there is no cards left in pile, the game will be taken as a draw (tie).

3). The stacking of cards is the most important feature in uno. In our application, +2 (Draw 2) can be stacked on +2, regardless of the color. For example, +2 and +2 will lead to a total of +4 for the next player. And +4 (Wild Draw 4) can be stacked on +2, leading to +6 in total. But the opposite way is not allowed, which means that +2 cannot be stacked on +4, since +4 has the highest priority. Moreover, +4 can also be stacked on +4, which leads to a total of +8. This could be a huge bomb in our game! It means that the player needs to think twice even if s/he has this ace in hand.

IV. Software Stack

Below is the list of software stack used in our team project, together with their version numbers.

- Server Provider: Amazon Web Services - Elastic Compute Cloud (AWS - EC2)
- Version of Server System: Windows R2012
- HTTP Server: Apache 2.4.33
- Database: MySQL 5.7.21
- Server Side Language: PHP 7.1.16
- Front-end Framework: JavaScript + Bootstrap v3 + HTML5
- Back-end Framework: Laravel 5.8
- Additional Library Package: Socket.IO 1.4.3 (JavaScript Package for Client Side) and Workerman/PHPSocketIO 1.1 (PHP Package for Server Side)

V. Tools for Communication

During the entire process of development, we are using Discord as the tool for team communication. We have created a private channel for our group, where we can either launch a conversation or post any information related to the term project. Basically, we utilize the "@" function in discord. Within our group channel, we can leave messages to some specific group member(s) by using "@username". We can also notify the whole team with some newly posted updates by using "@everyone".

Besides, we take advantage of the weekly meetings to exchange ideas as well as solve technical issues in our web development. The face-to-face talking provides us with the opportunities to discuss problems more intuitively. This serves as the complementary communication for Discord, which is pretty helpful for our teamwork.

VI. Tools for Task Management

Task assignments and following check-up processes are managed by the combination of GitHub and Discord. For each milestone, we assign checkpoints (tasks) on Discord by "@"ing the involved members or the whole team. Once the checkpoint is done, the code is pushed to the "dev" branch on GitHub, and message is posted on Discord to notify the team.

To improve the parallelism of implementation progress, the workload in each milestone has been divided into multi-tasks and distributed to the sub-teams of front-end and back-end. For example, the front-end members are mainly responsible for the design of UI views and JavaScript events, while the back-end members are dealing with the logic of models and controllers on server side. In most cases, two sub-teams are working separately on their own tasks so that they do not have to wait for each other, during which the team lead serves as the floater to help solve the technical issues. In this way, our teamwork has been managed efficiently, with each milestone delivered in a timely manner.

We have used multiple branches to manage our GitHub repo. For example, the "master" branch is always clean, with the latest workable version. The "dev" branch is for our regular development, with the modifications committed to this repo for code review. The newly pushed code will not be merged to "master" until all the bugs have been fixed. Moreover, "milestones" is the branch for documentation, while "beta" is the one for our beta launch.

VII. Build Instructions for Application

1). Download WAMP Server (Windows + Apache + MySQL + PHP) from http://www.wampserver.com/en/

2). Install WAMP Server on Window System (e.g., "D:\Software\Wamp"), and configure the root directory as "www" (e.g., "D:\Software\Wamp\www"). Then launch the WAMP Server. The icon should be green, if running normally.

3). Customize the username ("root" by default) and password (null by default) of MySQL database by accessing http://127.0.0.1/phpmyadmin/ in the web browser.

4). Download the .rar file of uno game project from our private GitHub repo
https://github.com/csc667-02-sp19/csc667-sp19-Team08/blob/milestones/milestones/m5/uno.rar

5). Unzip the .rar file. There will be a folder named "uno", and a database sql file named "sfsuuno.sql".

6). Cut/Copy and Paste the "uno" folder under "D:\Software\Wamp\www", so that the project is located at "D:\Software\Wamp\www\uno".

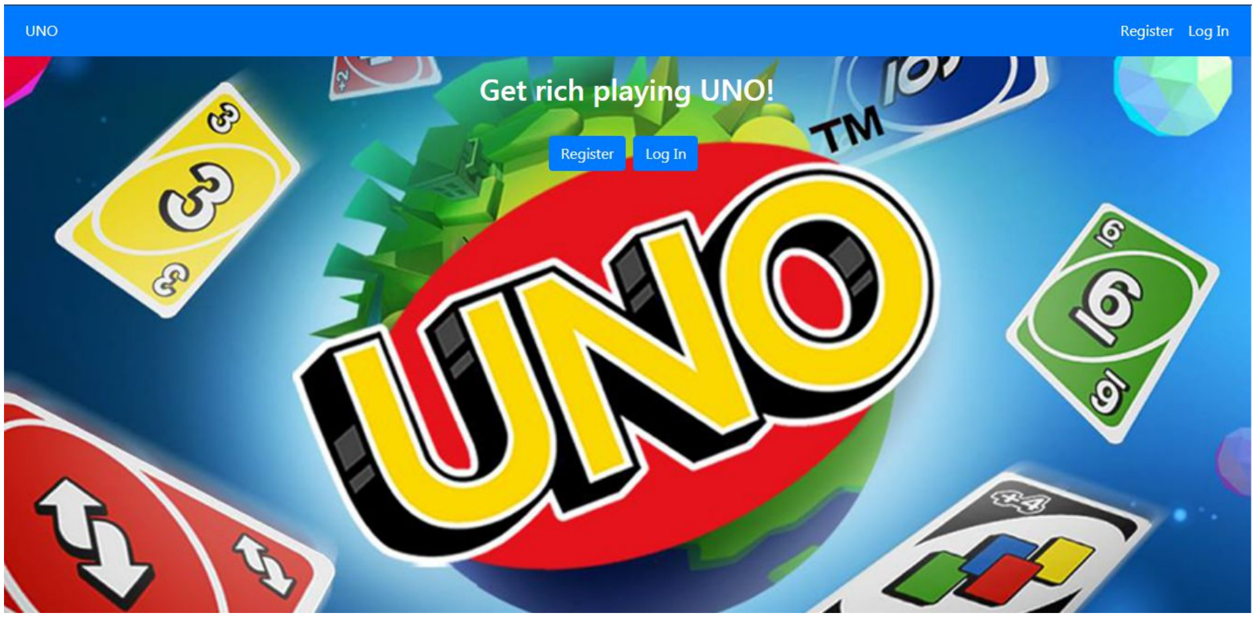
7). Create a database "sfsuuno" in MySQL by accessing http://127.0.0.1/phpmyadmin/ and import the sql file "sfsuuno.sql" to the database "sfsuuno".

8). Open the file "D:\Software\Wamp\www\uno\.env" with a text editor and customize the DB_USERNAME (line 13) and DB_PASSWORD (line 14) as the pair used in step 3). The DB_DATABASE (line 12) has been configured as "sfsuuno", with no need to edit.

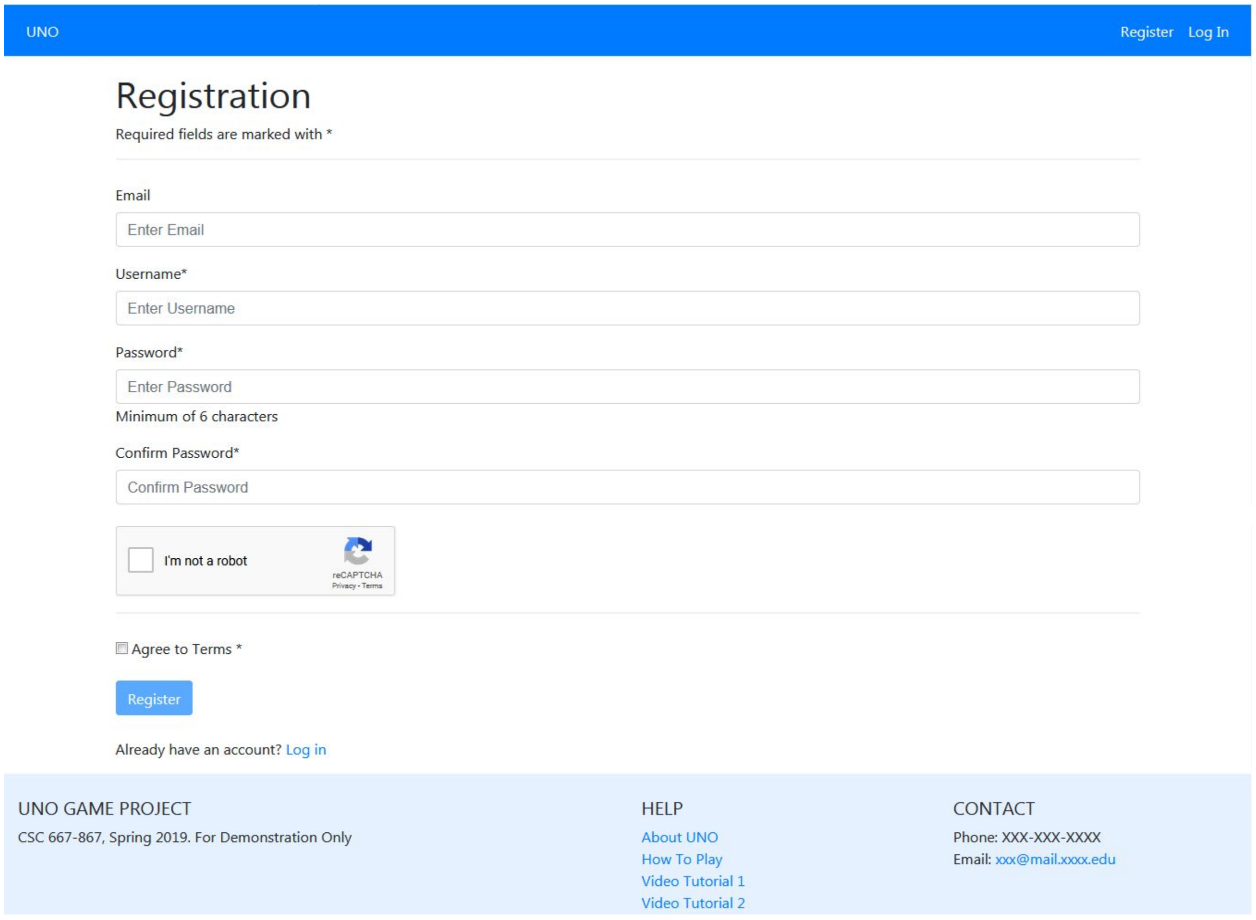
9). Open the web application by accessing http://127.0.0.1/uno/public/ in a web browser.

VIII. Pictures of Web Page in Application

1). Landing Page



2). Registration Page



UNO

Register Log In

Registration

Required fields are marked with *

Email

Enter Email

Username*

Enter Username

Password*

Enter Password

Minimum of 6 characters

Confirm Password*

Confirm Password

I'm not a robot

reCAPTCHA
Privacy - Terms

☐ Agree to Terms *

Register

Already have an account? Log in

UNO GAME PROJECT

CSC 667-867, Spring 2019. For Demonstration Only

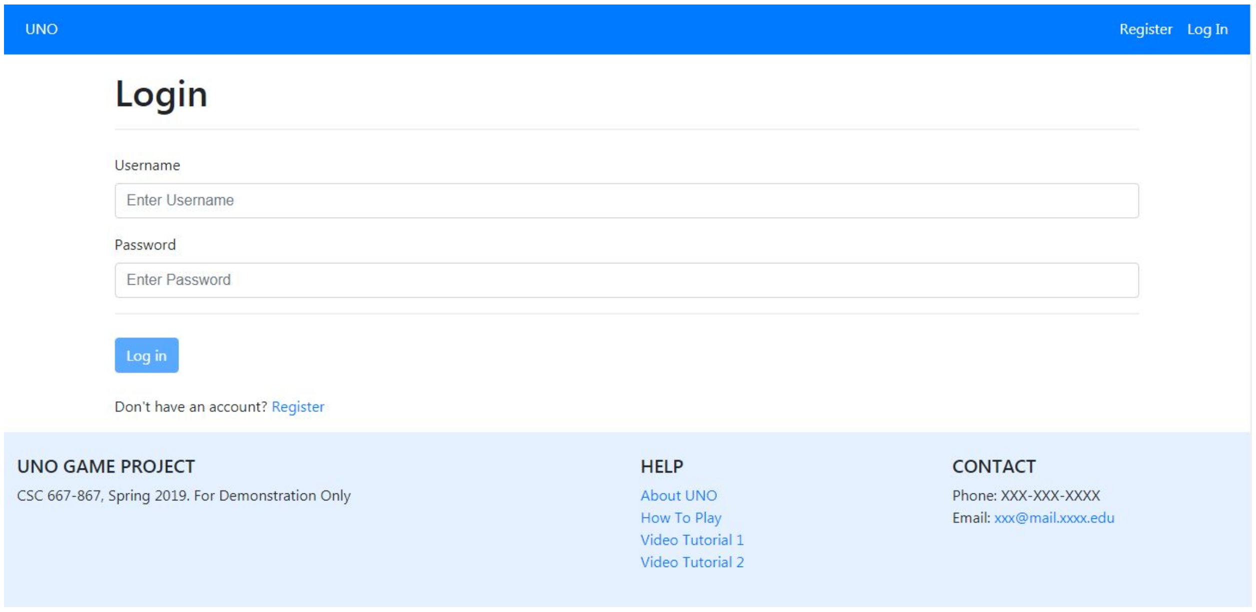
HELP

About UNO
How To Play
Video Tutorial 1
Video Tutorial 2

CONTACT

Phone: XXX-XXX-XXXX
Email: xxx@mail.xxxx.edu

3). Login Page



UNO

Register Log In

Login

Username

Enter Username

Password

Enter Password

Log in

Don't have an account? Register

UNO GAME PROJECT

CSC 667-867, Spring 2019. For Demonstration Only

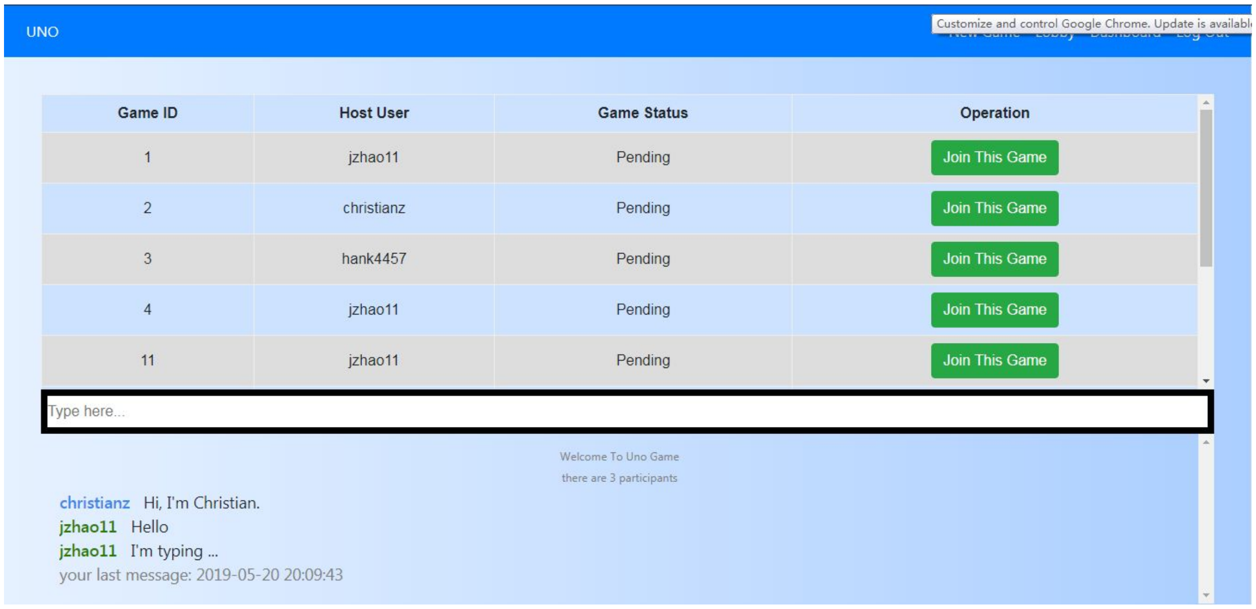
HELP

About UNO
How To Play
Video Tutorial 1
Video Tutorial 2

CONTACT

Phone: XXX-XXX-XXXX
Email: xxx@mail.xxxx.edu

4). Lobby Page (with chatting messages)



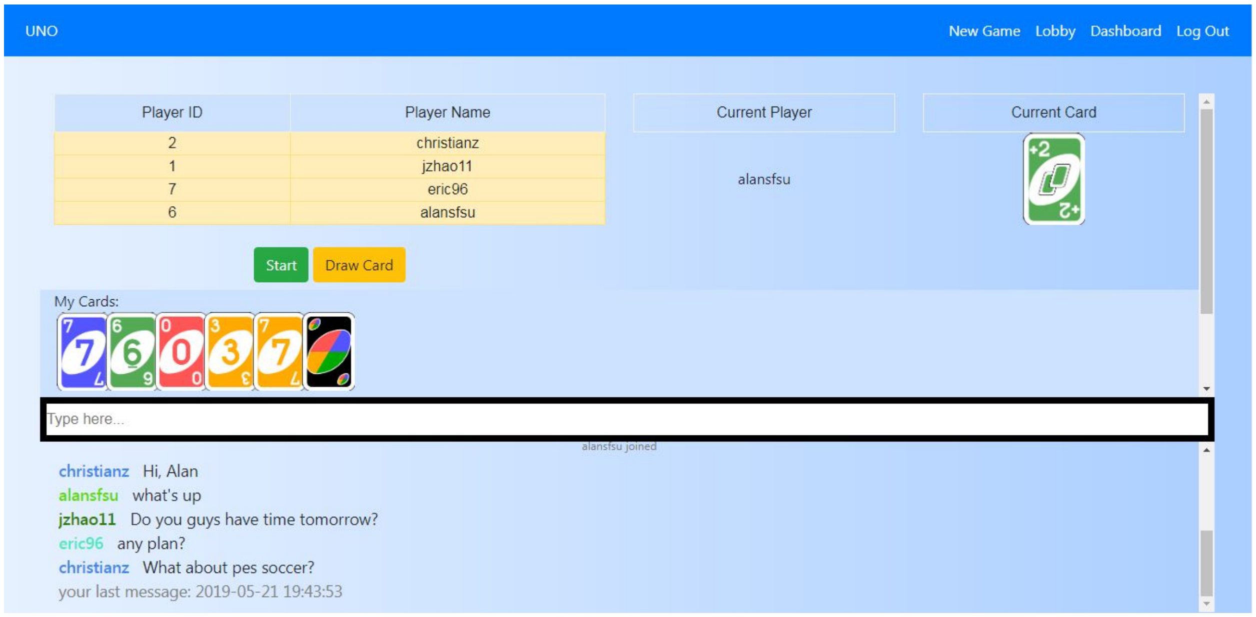
Game ID	Host User	Game Status	Operation
1	jzhao11	Pending	Join This Game
2	christianz	Pending	Join This Game
3	hank4457	Pending	Join This Game
4	jzhao11	Pending	Join This Game
11	jzhao11	Pending	Join This Game

Type here...

Welcome To Uno Game
there are 3 participants

christianz Hi, I'm Christian.
jzhao11 Hello
jzhao11 I'm typing ...
your last message: 2019-05-20 20:09:43

5). Game Page (with chatting messages)



Player ID	Player Name
2	christianz
1	jzhao11
7	eric96
6	alansfsu

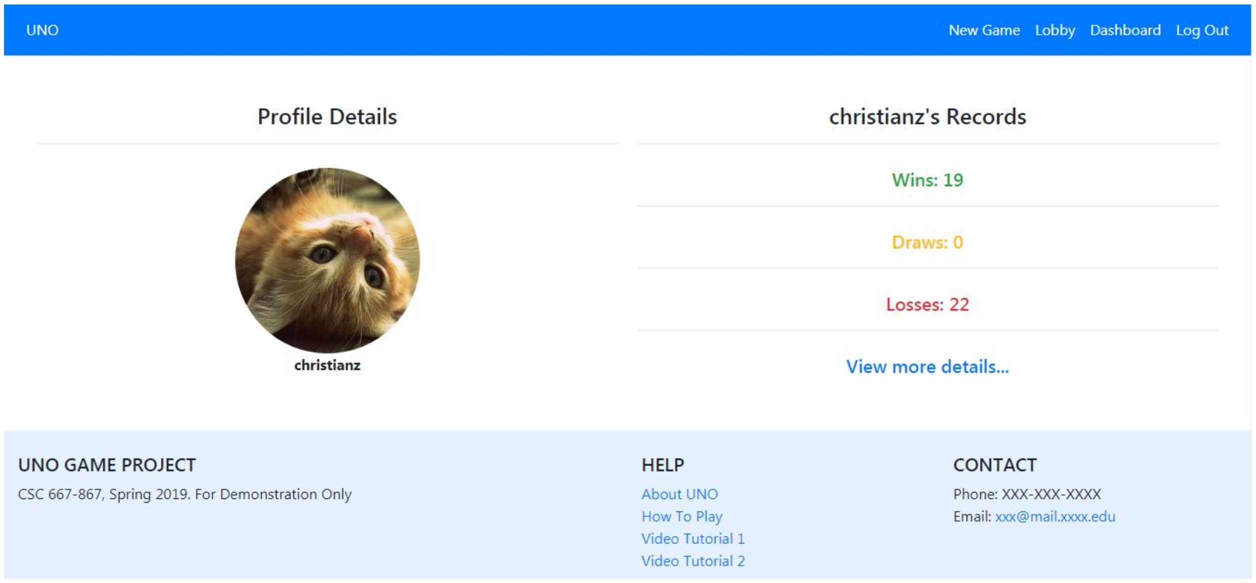
Current Player	Current Card
alansfsu	

Type here...

alansfsu joined

christianz Hi, Alan
alansfsu what's up
jzhao11 Do you guys have time tomorrow?
eric96 any plan?
christianz What about pes soccer?
your last message: 2019-05-21 19:43:53

6). Dashboard Page



Profile Details

christianz

christianz's Records

Wins: 19

Draws: 0

Losses: 22

View more details...

UNO GAME PROJECT

CSC 667-867, Spring 2019. For Demonstration Only

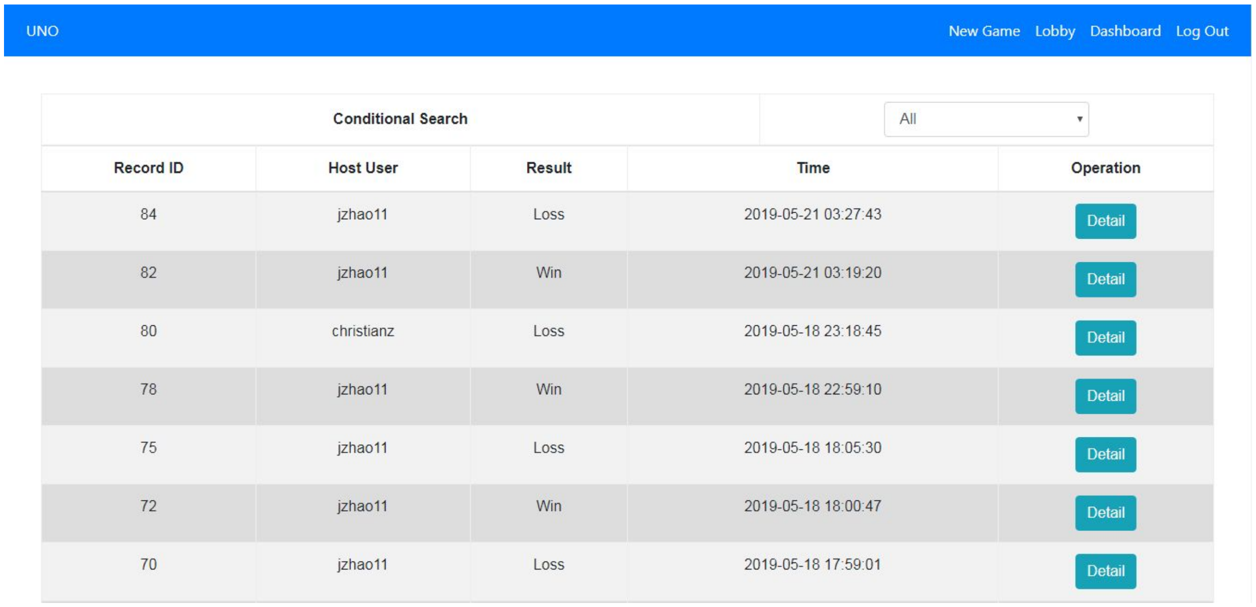
HELP

About UNO
How To Play
Video Tutorial 1
Video Tutorial 2

CONTACT

Phone: XXX-XXX-XXXX
Email: xxx@mail.xxxx.edu

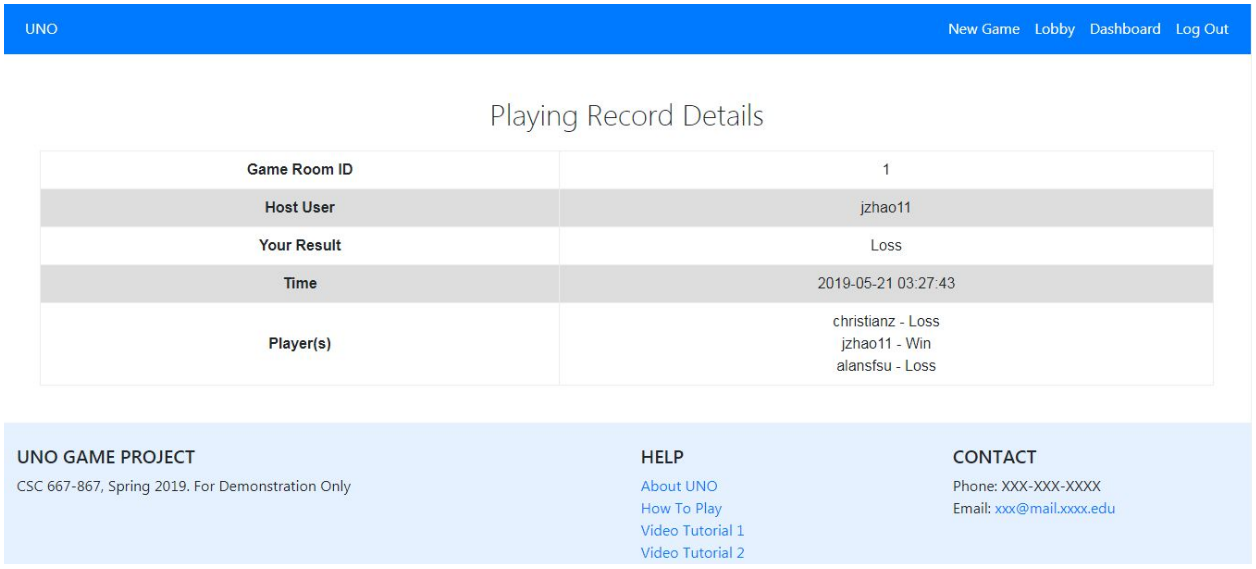
7). Record Page



Conditional Search

Record ID	Host User	Result	Time	Operation
84	jzhao11	Loss	2019-05-21 03:27:43	Detail
82	jzhao11	Win	2019-05-21 03:19:20	Detail
80	christianz	Loss	2019-05-18 23:18:45	Detail
78	jzhao11	Win	2019-05-18 22:59:10	Detail
75	jzhao11	Loss	2019-05-18 18:05:30	Detail
72	jzhao11	Win	2019-05-18 18:00:47	Detail
70	jzhao11	Loss	2019-05-18 17:59:01	Detail

8). Record Detail Page



Playing Record Details

Game Room ID	1
Host User	jzhao11
Your Result	Loss
Time	2019-05-21 03:27:43
Player(s)	christianz - Loss jzhao11 - Win alansfsu - Loss

UNO GAME PROJECT
CSC 667-867, Spring 2019. For Demonstration Only

HELP
About UNO
How To Play
Video Tutorial 1
Video Tutorial 2

CONTACT
Phone: XXX-XXX-XXXX
Email: xxx@mail.xxxx.edu

IX. List of API Routes

i). Socket Events

1). Display a new message in chatting area
Event name: "new message"
Parameters: $data => message
Sample usage: socket.emit("new message", $data);
Response: {
    "username" => username of the user who posts this message,
    "message" => message to be broadcasted within the channel,
}

2). Add a user to a game room
Event name: "add user"
Parameters: $user => object of user (id, username), $game_id => game room id
Sample usage: socket.emit("add user", $user, $game_id);
Response: {
    "username" => username,

"users" => array of users in this game room,
"cards" => user"s uno cards,
"curr" => current uno card on game board,
"num_users" => number of users in this game room,
"turn" => next turn in this game room,
"num_draws" => number of cards to draw,
"started" => if this room is in-game,
}

3). Start a game
Event name: "start game"
Parameters: $game_id => game room id, $username => username of the host user
Sample usage: socket.emit("start game", $game_id, $username);
Response: {
"cards" => array of cards for each user,
"num_users" => number of users in this game room,
"turn" => next turn in this game room,
"num_draws" => number of cards to draw,
}

4). Play a card on game board
Event name: "play card"
Parameters: $game_id => game room id, $card_id => uno card id, $username => username of the current player
Sample usage: socket.emit("play card", $game_id, $card_id, $username);
Response: {
"curr" => current uno card on game board,
"turn" => next turn in this game room,
"num_draws" => number of cards to draw,
}

5). Draw card(s)
Event name: "draw card"
Parameters: $game_id => game room id, $username => username of the current player
Sample usage: socket.emit("draw card", $game_id, $username);
Response: {
"turn" => next turn in this game room,
"nocard" => if there is no card left in deck (if so, game considered as a draw/tie),
"num_draws" => number of cards to draw (always reset to 1 after drawing cards(s)),

}

6). Change color in a game
Event name: "change color"
Parameters: $game_id => game room id, $new_color => new color
Sample usage: socket.emit("change color", $game_id, $new_color);
Response: {
    "curr" => current uno card on game board,
    "color" => new color after the change (which will be broadcasted within the channel),
}

7). Say "uno" in a game (when the player has only 1 card left)
Event name: "uno"
Parameters: $game_id => game room id, $username => username of the user who says "uno"
Sample usage: socket.emit("uno", $game_id, $username);
Response: {
    "username" => username of the user who says "uno",
}

8). Win a game
Event name: "win game"
Parameters: $game_id => game room id, $user_id => user id of the winner, $username => username of the winner
Sample usage: socket.emit("win game", game_id, user_id, username);
Response: {
    "username" => username of the winner,
}

9). Type characters in the chatting box
Event name: "typing"
Parameters: no
Sample usage: socket.emit("typing");
Response: {
    "username" => username of the current websocket (which will be blinking on the screen),
}

ii). Back-End Controllers (excluding the APIs that only render web pages)

1). Create a new game room
HTTP Method: POST
Parameters: $host_user_id => user id of the host, $status => game status
Sample usage: "project_url/creategame" (project_url can start from localhost or remote server, e.g., "localhost/uno/public") with HTTP body

```
Response: {
        if success, return game id; otherwise, return 0
}
```

2). Read detailed info of a game
HTTP Method: GET
Parameters: $game_id => game room id
Sample usage: "project_url/readgame?game_id=3"

```
Response: {
        "view" => resource view of game page,
        "game"  => game detail
}
```

3). Update game status after it ends
HTTP Method: GET
Parameters: $game_id => game room id
Sample usage: "project_url/endgame?game_id=3"

```
Response: {
        if success, return game id; otherwise, return 0
}
```

4). Read detailed info of a card
HTTP Method: GET
Parameters: $card_id => card id
Sample usage: "project_url/readcard?card_id=2"

```
Response: {
        "id" => card id,
        "number"  => card number,
        "color" => color of the card,
        "img"  => image of the card,
}
```

5). Read game turn for a player (Check if it is his/her turn to play)
HTTP Method: GET

Parameters: $game_id => game room id, $user_id => user id of the player
Sample usage: "project_url/playcard?game_id=3&user_id=2"
Response: {
    if success, return 0; otherwise, return -1
}

6). Read current player's username (for game board update)
HTTP Method: GET
Parameters: $game_id => game room id, $turn => current turn
Sample usage: "project_url/readplayer?game_id=3&turn=0"
Response: {
    if success, return username; otherwise, return null
}

7). Create playing record for a player
HTTP Method: GET
Parameters: $game_id => game room id, $user_id => user id of the player, $result => game result for that player (1 - win; 0 - draw; -1 - loss)
Sample usage: "project_url/createrecord?game_id=3&user_id=2&result=1"
Response: {
    if success, return record id; otherwise, return 0
}

8). Read detail of multiple records
HTTP Method: GET
Parameters: no
Sample usage: "project_url/readrecord"
Response: {
    "view" => resource view of record page,
    "records" => array of records
}

9). Read detailed info of a record
HTTP Method: GET
Parameters: $record_id => game record id,
Sample usage: "project_url/recorddetail?record_id=2"
Response: {
    "view" => resource view of record detail page,
    "record" => record detail

}

10). Add a user to a game room
HTTP Method: POST
Parameters: $game_id => game room id, $user_id => user id of the player
Sample usage: "project_url/createchannel" with HTTP body
Response: {
    if success, return 1; otherwise, return 0
}

11). Delete a user from a game room
HTTP Method: GET
Parameters: $game_id => game room id, $user_id => user id of the player
Sample usage: "project_url/deletechannel?game_id=3&user_id=2"
Response: {
    if success, return 1; otherwise, return 0
}

X. Team Member Contributions

Jianfei Zhao (Team Lead & Back-End Engineer):

1. Designed and implemented database models
2. Implemented back-end controllers
3. Implemented PHPSocket.IO events on server side
4. Helped with JavaScript functions on client side
5. Helped fix bugs for Priority-1 functionalities
6. Solved data issues for teammates
7. Managed teamwork by checking points

Alan Ng (Front-End Engineer):

1. Implemented landing page
2. Helped refine the initial game logic and fix bugs
3. Helped with the game logic prototype
4. Translated much of the prototype onto javascript
5. Implemented lobby page

Philip Yu (Front-End Lead):

1. Created wireframes for web application
2. Organized front-end design tasks
3. Implemented profile page
4. Implemented registration page with AJAX
5. Implemented record page with JS functions
6. Designed game logic prototype
7. Helped with documentation

Hang Li (Back-End Lead):

1. Created logical schemas for database
2. Retrieved data and displayed in web pages
3. Designed the routes for back end
4. Implemented the socket.io APIs for chatting
5. Managed github branches and code review

Eric Chen (Front-End Engineer):

1. Helped design wireframes for web application
2. Implemented Login page, Home page, and Record details
3. Implemented part of the socket events for Game page
4. Helped with game logic prototype
5. Fixed the layouts for front-end views

XI. Project Reflection

Before we started the group project, each of us were at different levels of experience and knowledge in regards to designing and developing web applications. Some of us have already taken or were concurrently enrolled in CSC 648, a Software Engineering class at SFSU, where students were grouped up and worked on a web application in a similar manner. But a few others of us did not take that course. As a result, only a few of us had the experience with working on a web application in a team-oriented environment. However, our team worked as one and collaborated with one another, filling in for the weaknesses and empty knowledge that others did not have. We split our team into two sub-teams, one for the front-end, whom worked on the design of our web application for the client side, and the backend team, whom were responsible for the the behind-the-scenes server-side web application logic and the implementation of front-end design. In the end, we learned from one another and built a final product together, which was presented in front of the class to showcase our teamwork and collaboration.

XII. Project Conclusion

This group project was a fun assignment overall, and a learning experience for all of us. We have each learned more about web development for both the design and development phase. We all enjoyed working on this assignment as it offered us an opportunity to work in a team-oriented environment, which some of us have had little or no experience of doing. Some aspects of the assignment were challenging, but also rewarding when we got the product to finally work in the end. We all have known more about our strengths and weaknesses and have each improved and learned from this term project. In the end, we as a team, have designed and developed a fully operational web application of the card game, Uno.