

DTP:

A Video Game Match-Making Service

Designed & Developed By:

Michael Liguori and Christopher Chen

Created For:

CS492A-02 - Senior Project

April 30th, 2021

Table of Contents

1- Technical Overview	4
1.1 System Architecture	4
1.2 System Features	5
1.2.1 User Authentication	5
1.2.2 User Types	6
1.2.3 Queue As Host	7
1.2.4 Queue As Player	7
1.2.5 Chat	7
1.2.6 Accepting/Rejecting Actions	9
1.2.7 Misc features	9
2- User Guide	
2.1 Login Page	11
2.1.1 Signing Up	12
2.2 Home Page	13
2.3 My Queues	13
2.3.1 Chat box	13
2.3.2 Accept/Reject Actions	13
2.4 Queue Up	14
2.5 Games	15
2.6 My Friends	17
2.7 Support	17
2.8 About Us	18

1 - Technical Overview

1.1 System Architecture

Down To Play (DTP) is developed with three main components, the database (DB) server, the Web Server, and the Client. This is in an attempt to satisfy the need to create a queuing system wherein users can queue up for multiple games simultaneously, increasing their chance of finding a match. Samples of the code are pasted in between explanations of certain aspects and features for clarification and examples of implementation throughout the document.

The DB Server is a MySQL Server. With the help of MySQL Workbench, setting up a local root to connect to the Web server was achieved with the attributes set during MySQL Workbench installation process. Afterwards, the workbench interface helped interpret the data within the DTP DB Schema. The entirety of MySQL Statements used from this project can be found in the file "MySQL Test Statements.txt"..

The Web Server is created through the several dependencies: Node js, and Express js, Pug js, Socket.io, and mysql npm packages. Node js serves as the foundation of the web server, sending and receiving data to and from clients, as well as to and from the DB server. The mysql package helps set up the connection used for MySQL workbench. Socket io is used for authentication for the queue system and handling appropriate responses from server to client and client to server. Pug js is used to create html pages for the client to render using data from the DB server and express helps streamline Node server event handling and routing.

The Client in this case is a user accessing the web server through a web browser. The Client will request and receive services from the Web Server through the use of any major

web browsers such as Google Chrome, Apple Safari, and Mozilla Firefox. Testing was done on the Mozilla Firefox web browser on a windows pc.

1.2 System Features

1.2.1 User Authentication

Users are required to sign in, or up if new, and they must be active, 30 min of inactivity will require another sign in, to verify authentication.

Implementation:

```
//Assignes a cookie with unique token for every connection
app.use((req, res, next) => {
  console.log("App Use: Route Requested: " + req.path)
  let cookies = req.cookies

  // If there is no cookie
  if (!cookies.token) {
    console.log("App Use: No token cookie found")
    res.cookie('token', makeid(10), { maxAge: 3600000 })
    req.requiresLogin = true
    next()
  }

  // If there is a cookie
  else {
    console.log("App Use: cookie found")

    // get user's login data
    User.getUserLoginData(req.cookies.token)
      .then(
        function completed(firstResults) {
          console.log('App Use: User Credentials Match');

          if ((firstResults[0].TimeDiff > -3000)) {
            // if the user's last login is recent, refresh the last login
            console.log('App Use: User TimeDiff > -3000')

            User.updateUserLastLogin(req.cookies.token)
          }
        }
      )
  }
})
```

Middleware Functions: using app.use, We could ask for information for each Client connection and make decisions based on what is returned. This takes advantage of the client web browser cookie feature, as it assigns the randomized id to the user's

successful login in/sign in at the DB server. Along with the randomized token now assigned to the logged in user, a timestamp is used to measure time between the last connection made. This is to simulate inactivity error handling.

User
+ user_id : Int Auto
+ username : Int
+ user_email : String
- user_password : String
+ user_token: String
+ user_last_login: DateTime

This is the user class that represents the columns and datatypes that are associated with a user's account thus far. An arbitrary user_id is created for each new user, and their username, email, and password are stored. The token represents a randomly generated string which is used to identify a user throughout different web pages when logged in to DTP. The last login is updated whenever the authenticated user requests any activity to the DTP web server, and is checked to see whether or not they should be logged off for inactivity

1 • `SELECT * FROM dtp.user;`

user_id	username	user_email	user_password	user_token	user_last_login
1	User_1	User1@email.com	User1	RNYH35SHrF	2021-04-29 21:22:39
2	User_2	User2@email.com	User2	jjMSuUFBQW	2021-04-23 13:06:28
3	User_3	User3@email.com	User3	NULL	NULL
4	User_4	User4@email.com	User4	NULL	NULL
5	User_5	User5@email.com	User5	NULL	NULL
6	User_6	User6@email.com	User6	NULL	NULL
7	User_7	User7@email.com	User7	NULL	NULL
8	User_8	User8@email.com	User8	NULL	NULL
9	User_9	User9@email.com	User9	NULL	NULL
10	User_10	User10@email.com	User10	NULL	NULL

This is the user entity in MYSQL Workbench

1.2.2 User Types

There is only one type of user. All users have access to the same actions as others. However, users can be said to queue up in one of two ways: AS a host or as a player. A

host refers to the user who will create the room. A player is a user who is willing to join the created room of a host. NOTE: For each game and gamemode, a user can only be either a host or a player.

1.2.3 Queue as a Host

This feature is how the user will select whether they wish to host a match for a group of players. Note: For each game and gamemode selected, a player can only be a host or player, not queued for both. Certain actions are taken both on the web server and DB servers depending on which they choose.

```
//if host
if (req.body[element][0] != 0) {
  var queueToken = makeid(10)
  connection.query("INSERT INTO `dtp`.`queuedplayer` (
    if (error) {
```

1.2.4 Queue as a Player

This feature is how the user will select whether they wish to be a player and is only willing to join rooms. Note: For each game and gamemode selected, a player can only be a host or player, not queued for both. Certain actions are taken both on the web server and DB servers depending on which they choose.

```
//if player
connection.query("INSERT INTO `dtp`.`queuedplayer` (
  if (error) {
```

QueuedPlayer	
+	game_id : Int
+	gamemode_id : Int
+	user_id : int
-	user_is_host : tinyint
+	user_token: String
+	user_queued_time: DateTime

This is the queuedplayer class which represents the information that is stored in the database when someone queues for a game/games. Each game queued will store the game_id, gamemode_id, the user_id, whether the user queues as a host or not, and the time they began their queue. This will allow the server to make checks against the database to find hosts and players to fill the host's queues.

```
1 • SELECT * FROM dtp.queuedplayer;
```

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell C							
game_id	gamemode_id	user_id	user_is_host	user_queued_time	user_queue_token	player_acc	
1	1	1	1	2021-04-30 04:57:11	vehpnfgidy	0	
1	3	1	1	2021-04-30 04:57:11	B63WPHP3zz	0	
1	3	5	0	2021-04-07 12:27:26	B63WPHP3zz	0	

Sample data from MySQL Workbench.

1.2.5 Chat.

This feature is available to a user who queues as a host. This means a queue with a unique identifier tied to the user queuing as a host, game, and gamemode is created. If others have joined the room, they can chat with each other. This feature is also available to the users who get placed in a new room for the game and gamode the queue as a player for.

```
sendButtons.forEach((element) => {
  //console.log(element)
  let roomToken = element.className.split(' ')[1]
  let input = document.getElementsByClassName('input ' + roomToken)[0]

  element.addEventListener('click', function (e) {
    e.preventDefault();
    console.log('Send Button pressed for ' + roomToken)
    if (input.value) {
      socket.emit('chat message', { username: username, roomToken: roomToken })
      input.value = '';
    }
  });
});
})
```

The above code creates a chatbox for each queue found for that authenticated user from the DB server upon Web-Client Request, allowing the ability for all users who are placed in a host room the ability to talk to each other. See 2.3.1

1.2.6 Accepting/Rejecting Actions

When a potential full room, referring to a game's gamemode's required number of players (NOP) matching the number of players in a room, then a prompt is given for the user to decide whether to accept or reject the match. There are several possibilities.

If the user accepts the full room within a 30 second window, and all other players in that room also have accepted the full room within the 30 seconds, then those players and the user will be removed from all their queues and left with their room to continue to chat and set up a match.

Most of this occurs due to the socket io connections being made via socket io package and code found in Socket.js.

```
socket.on('acceptQ', (msg) => {
  console.log(msg)
  token = msg.cookie.split('=')[1]
  user_queue_token = msg.user_queue_token
  reply = msg.reply
  console.log(token + ":" + user_queue_token + ":" + reply)
  connection.query('UPDATE queuedplayer AS QP, `
    if (error) {
```

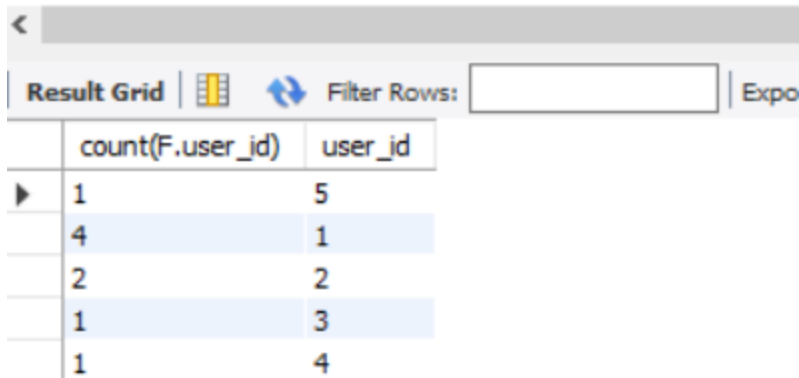
```
socket.on('rejectQ', (msg) => {
  console.log(msg)
  token = msg.cookie.split('=')[1]
  user_queue_token = msg.user_queue_token
  reply = msg.reply
  console.log(token + ":" + user_queue_token + ":" + reply)
  console.log("Socket on: rejectQ: Before Delete")
```

1.2.7 Misc features

Several other features were implemented but not really necessary for the scope of this project, which was to create a queue system which can handle multiple games and game modes at the same time and using a number of players (NOP) and unique room tokens as the means to verify a full room.

One such feature is the friends page. Dummy data was used to provide minimum basic information of a user to be reproduced client side. Although being able to send messages to friends online, or off, and to add or remove them is missing, and unnecessary for the scope of this project it was still a good form of coding exercise. The knowledge I gained was the ability of Pug js. Its functionality and purpose is to create dynamic pages based on DB server data processed at the Web server upon request from an authenticated user. I see the potential of customization for UI based on certain user preferences and User experience, but again out of scope for this project.

```
1 • SELECT count(F.user_id), F.user_id
2
```



	count(F.user_id)	user_id
▶	1	5
	4	1
	2	2
	1	3
	1	4

This retrieves the number of friends by user_id in the dummy data. This info was used to process the data and pass to pug to render an html file to send to client upon request.

Additional pages found outside the queuing pages required for testing. For example, the about us, support, friends, and games pages are not necessary for the functionality of the queueing system but were used to encapsulate a very rough idea of a user interface which would provide my functionality for DTP users once queueing system and hosting support is completed. These different “pages” are the results of routes and pug rendering of DB query results.

```

//List of routes for redirect
//Route to DTPSignIn.html
app.get('/', (req, res) => {
  console.log(pug.renderFile((process.cwd() + '/PUG/login.pug')))
  //res.sendFile(path.join(__dirname, "../HTML/DTPSignIn.html"))
  res.send(pug.renderFile((process.cwd() + '/PUG/login.pug')))
})
//Route to DTPSignInCreate.html
app.get('/DTPSignInCreate.html', (req, res) => {
  res.sendFile(path.join(__dirname, "../HTML/DTPSignInCreate.html"))
})
//Route to DTPSignInError.html
app.get('/DTPSignInError.html', (req, res) => {
  res.sendFile(path.join(__dirname, "../HTML/DTPSignInError.html"))
})
//Route to DTPSignInTimedOut.html
app.get('/DTPSignInTimedOut.html', (req, res) => {
  res.sendFile(path.join(__dirname, "../HTML/DTPSignInTimedOut.html"))
})
//Route to DTPRegister.html
app.get('/DTPRegister.html', (req, res) => {
  res.sendFile(path.join(__dirname, "../HTML/DTPRegister.html"))
})
//Route to DTPRegisterError.html
app.get('/DTPRegisterError.html', (req, res) => {
  res.sendFile(path.join(__dirname, "../HTML/DTPRegisterError.html"))
})
//Route to DTPHome.html
app.get('/DTPHome.html', (req, res) => {
  if (req.requiresLogin == true) {
    return res.redirect('/')
  }
})

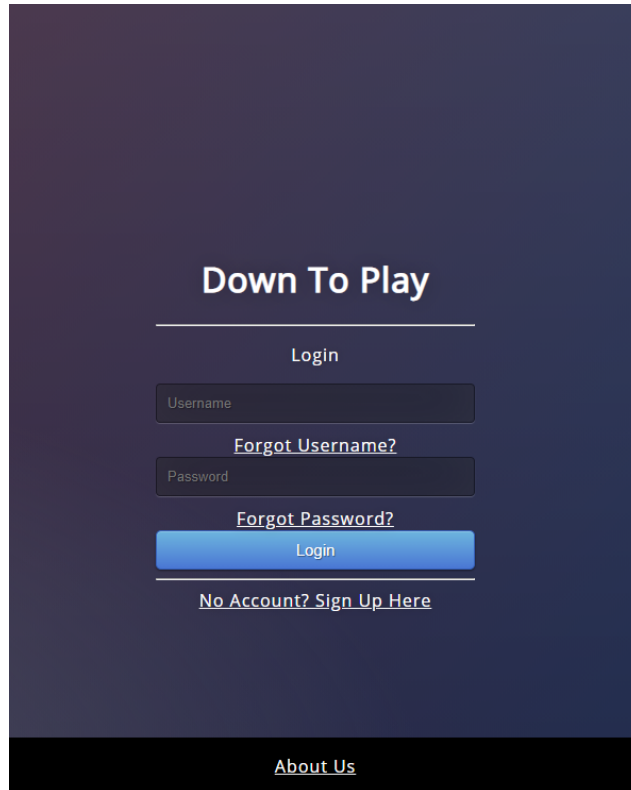
```

2 - User Guide

2.1 Login Page

On initial arrival, the user will be provided with a login screen in order to access Down To Play. Current access is restricted on local host computer and the local port 3000.

2.1.1 Login Page

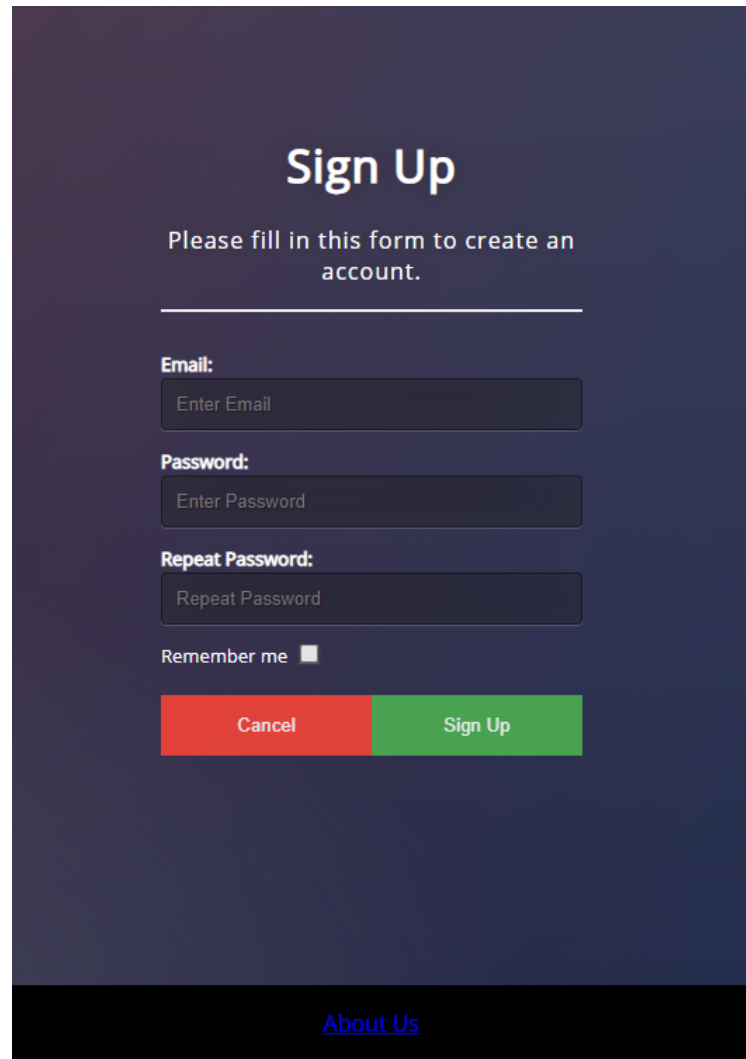


The user will need to enter their username and password to gain access to the application. If they submit a valid username and password, the user is brought to the home page.

```
//Route to post /sign_in
app.post('/sign_in', (req, res) => {
  connection.query('Select * FROM `dtp`.`user` WHERE username = ? AND password = ?')
    .end((error, results, fields) => {
      if (error) {
        res.send("Error");
      }
    })
})
```

2.1.2 Signing Up

If the user does not have an account, they can click the link “No Account? Sign Up Here”. The user will be brought to a form to create an account.

A dark-themed sign-up form with a blue gradient background. The title "Sign Up" is centered at the top in white. Below it, a message says "Please fill in this form to create an account." followed by a horizontal line. The form contains three input fields: "Email:" with placeholder "Enter Email", "Password:" with placeholder "Enter Password", and "Repeat Password:" with placeholder "Repeat Password". Below these is a "Remember me" checkbox. At the bottom are two buttons: a red "Cancel" button and a green "Sign Up" button. A footer bar at the very bottom contains a blue link "About Us".

Sign Up

Please fill in this form to create an account.

Email:
Enter Email

Password:
Enter Password

Repeat Password:
Repeat Password

Remember me ☐

[Cancel](#) [Sign Up](#)

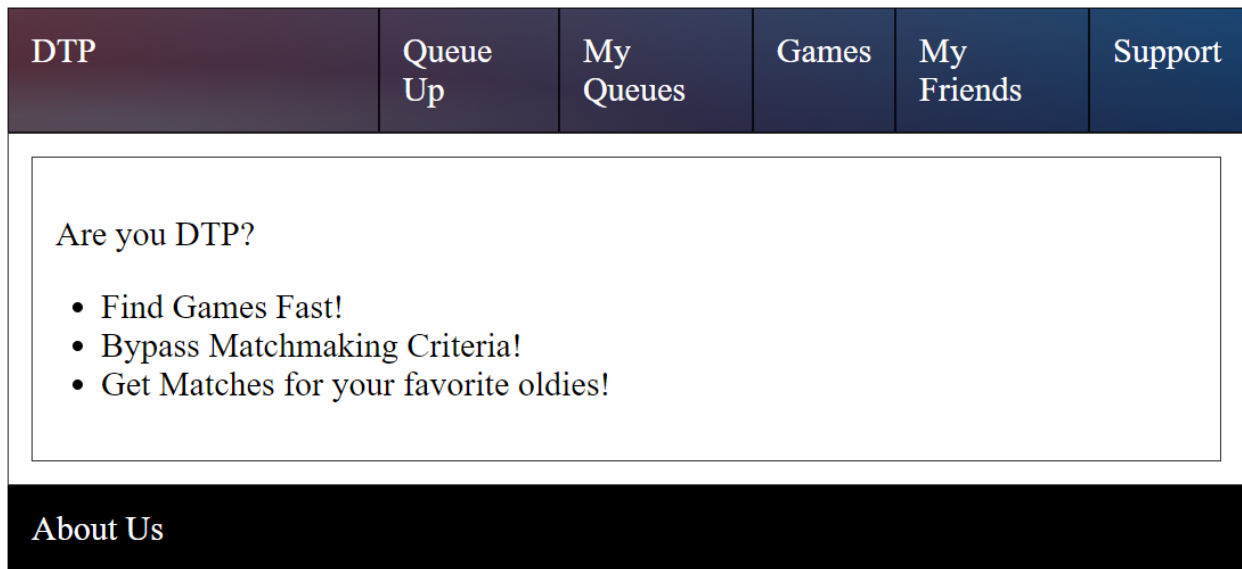
[About Us](#)

After submitting the sign up form, users “simulated ” a sent email in order to confirm their account. The user can then log in and access the web application.

```
//route to post /add_user
app.post('/create_account', (req, res) => {
  connection.query("INSERT INTO `dtp`.`user`(`use
    if (error) {
      res.redirect('/DTPRegisterError.html')
    }
  })
})
```

2.2 DTP Home Page

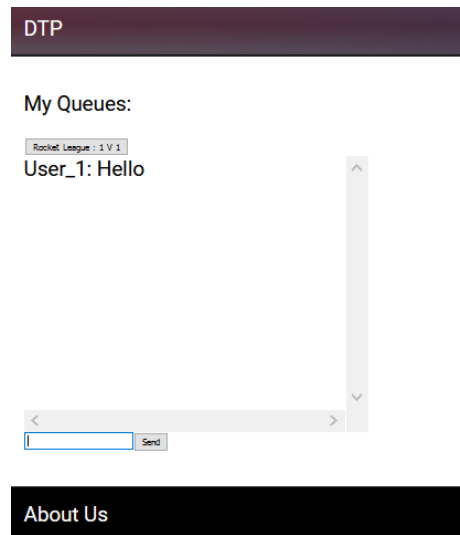
DTP Home Page is the main landing page after logging in. The user is able to access the other page via the navigation bar.



2.3 My Queues

2.3.1 Chat box

Users are able to chat with the other users in a game's queue. Depending on the number of games queued up, there will be a chat for the corresponding queue.



2.3.2 Accept/Decline Game

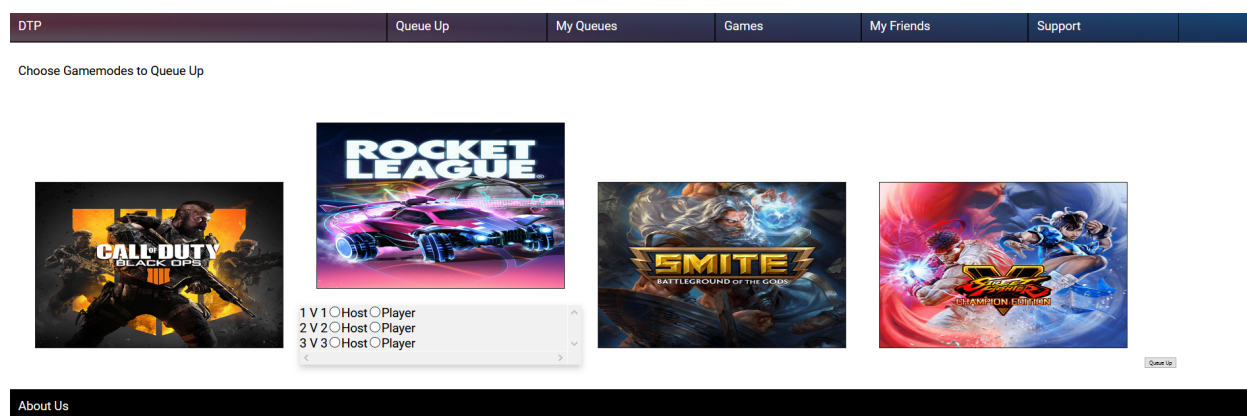
Once the queue has been filled, all the users in the queue will receive a pop-up that will give the user the option to reject or accept the game. Several different actions occur depending on the selection made.

A rejection, or no response within the 30 sec window frame, will result in the user or users to be “dequeued” from all the queues they may have entered prior to the room becoming full. This is checked with a boolean representing 0 for no response and 1 for response based on a set timeout on the web server allowing 30 secs to pass by before checking the DB to see which people have and have not accepted.

If a user has a zero for response and is not a host, then he is simply removed from the room and all other rooms they may have been in. If the user has a zero and is a host, all other players in the room will need to be replaced in the queue. This is achieved by setting their unique room token to null, then deleting the rooms of that user, following the same two paths above.

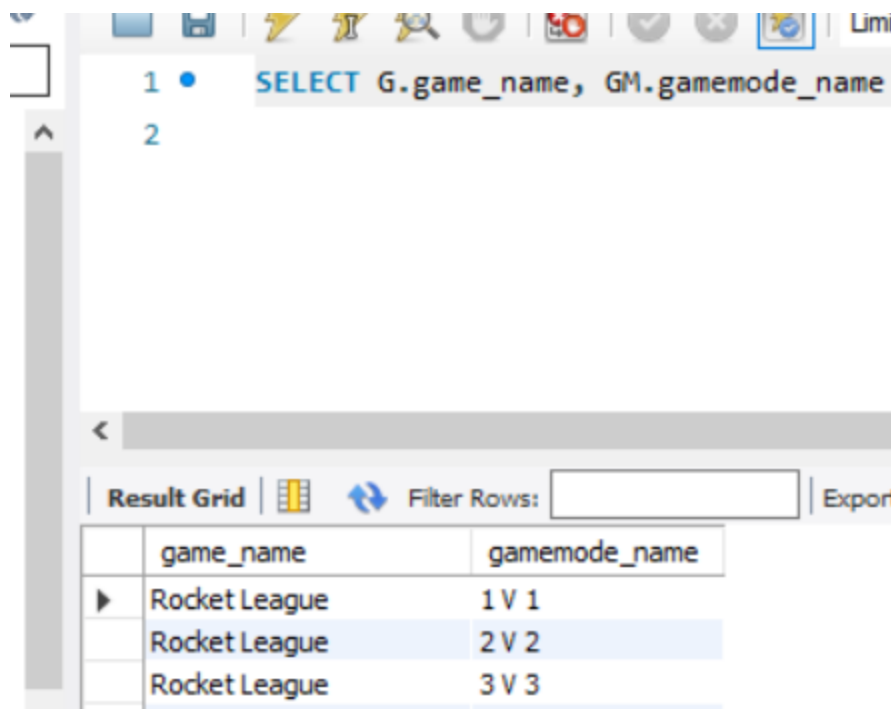
2.4 Queue Up

The Queue Up page will list all available games, when hovered over, a dropdown will show all the available game modes and the user will be given the option to either host or play. The user will then be able to hit the “Queue Up” button to enter the chosen queues or host the game.



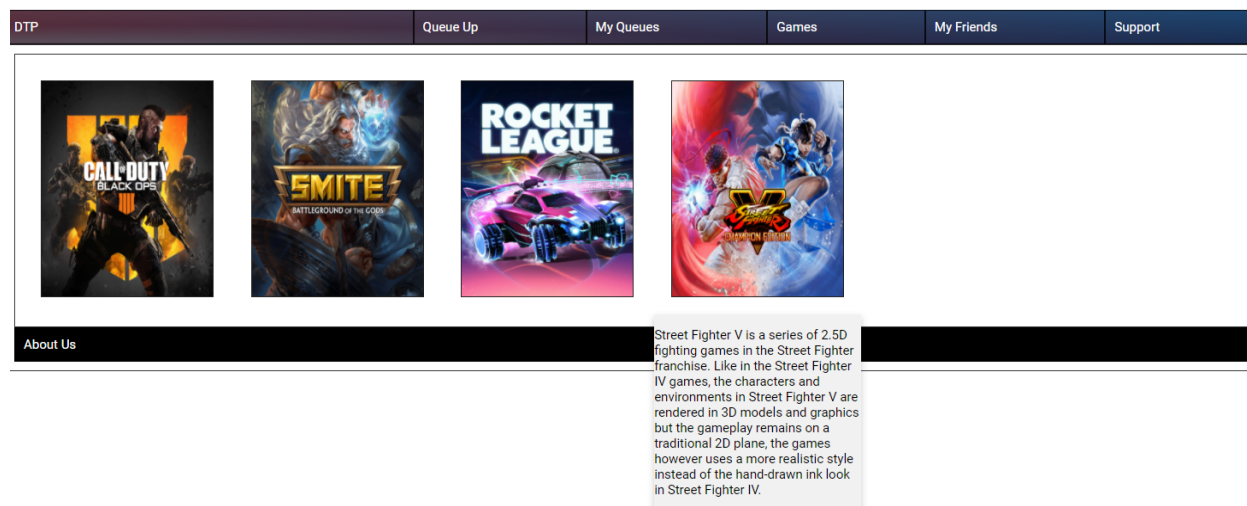
```
app.post('/queue_up', (req, res) => {
  var numberOfQueues = Object.keys(req.body
```

Route for the post to queue up. The data being sent by the client is a radio button list for each game and game gamemode combination possible, with two choices, Host or Player. With the ability to render this page using pug, as long as the new games entered into the database follow the same guidelines laid out in the MySQL Schema, then they will be able to be created here for users to find matches for.



2.5 Games

The Games page will list all the available games for hosting and playing. The user is able to hover over the game, a drop down will provide a simple summary of the game.




```

.grid-container-column2
.grid-item
  each game in gamelist
    .dropdown
      if "" + game.game_name + "" == "Call of Duty Black Ops 4"
        input.dropbtn(type='image' src='cod.png' id="" + game.ga
        .dropdown-content
          p
            | Call of Duty: Black Ops 4 is the fifteenth instal
      else if "" + game.game_name + "" == "Smite"
        input.dropbtn(type='image' src='smite2.png' id="" + game

```

Here is some pug implementation. As you see, the data is checked and sorted to match appropriate characteristics.

2.6 My Friends

The My Friends page shows a table of all the user's friends. The table shows the friend's usernames and email.

DTP

Queue Up

My Queues

Games

My Friends

Support

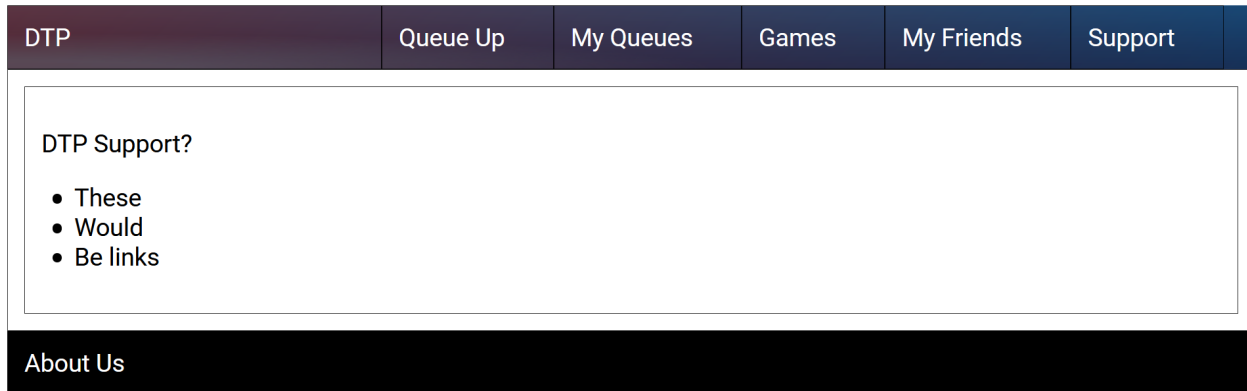
My Friends:

Friend Name	Friend Email
User_5	User5@email.com
User_2	User2@email.com
User_3	User3@email.com
User_6	User6@email.com
User_7	User7@email.com

About Us

2.7 Support

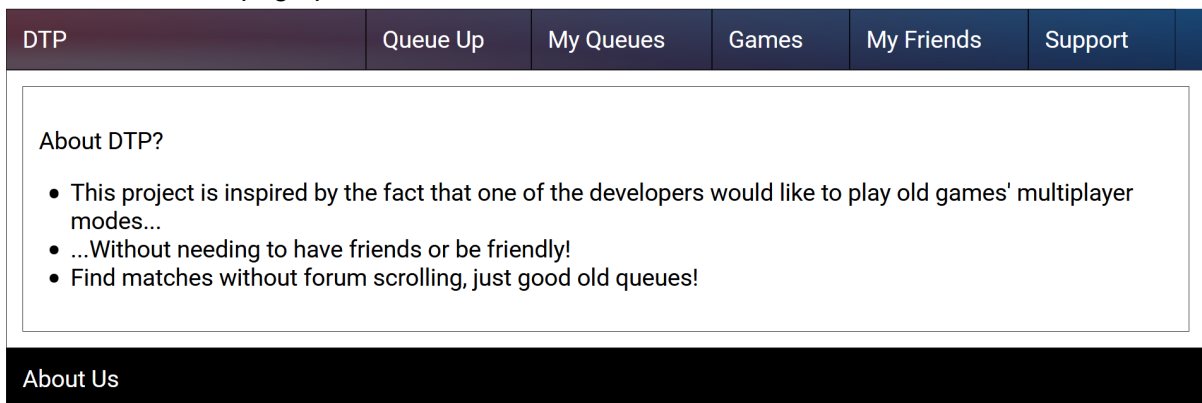
The Support page provides the user with links and FAQs about how to use the web application.



If DTP would be a full service for getting users to successful host queues after one is found, the support links that help explain some of the processes and problems that may occur

2.8 About Us

The About Us page presents the user with basic information about DTP and it's uses.



Normally this section would be filled with information about the company and product. Again, this is not important for the purpose and scope of this project.