**Online Pictionary Game With Flask And SocketIO**

Contents:

Analysis

Objectives

Design

Implementation

Testing

Evaluation

**Analysis**

**Abstract**

The project is a web-based implementation of Pictionary, a popular party game in which players take turns drawing things, and other players try to guess what the person is drawing. The aim of the game is to guess what the current artist is drawing. The next player in the room then becomes the drawer and the other players guess the drawing.

The target audience of this project is very wide, as it can be played by most ages, given that they have a basic understanding of how to use a computer and navigate the internet. In terms of interests, the game appeals to people who enjoy drawing or art, and people who are competitive or are just looking for some casual game to kill time with and have an internet connection. The game is also child friendly as there are no inappropriate words in the word list, however the chat is unfiltered.

**Research**

There are already quite a few Pictionary websites, which were perused to find areas in the market that could do with improving, as well as to get some ideas for areas in which the game could differentiate from its competitors.

One of the most popular Pictionary games out right now is <https://skribbl.io/>. Skribblio lets players choose a username and character, then either join a random public game or make a private room for them and their friends. The fact that players could make custom rooms was deemed an important feature, so it was decided to implement it on the website. It was decided that the avatar creation did not add much to the game and was not implemented to the project as to not overwhelm the user with useless features. An area that skribblio was lacking was the username aspect, as on Skribblio users choose a one-time username to play with. To improve on this, a login system will be added which will let players create permanent accounts they can use to play the game and store their high scores to compare with friends. The art style of the website was also very childish, which made the game seem as if it was only designed for children. The website will be designed with a professional style, so as to show the target audience is more than just children.

Graphical user interface, application, website

Description automatically generated

[Figure 1] The page a user is greeted with when they visit skribbl.io, the art style is very childish and the first thing that players see is the avatar creator and username picker.]

Another popular Pictionary website is <https://www.drawize.com>, which was an interesting one as it advertised as an online, multiplayer game however upon testing it seemed that all the other players were bots, with very standardized names and putting unrealistic phrases in chat. This came off as cheesy, and felt very disingenuous, and harmed the user experience to advertise as a multiplayer game then have the players be bots, so even though implementation of bots would be quite simple given the way the game is written, it was decided not to implement that into the program.

Graphical user interface, application

Description automatically generated

[Figure 2] The homepage of Drawize. There is a lot of ads, which make the homepage look cluttered and might make the user want to play a different game. The actual UI is also very cluttered and looks very childish.

There is also the Mattel board game Pictionary, which is a physical implementation of the game, in which players roll a die then move their piece on the board, with the colour they land on corresponding to a category that they will draw from. The player then draws a card and draws the word from the category that’s on the card, while the other player guesses what they are drawing. While this is an interesting take on the classic game, there is a lot of unnecessary complexity in the game, so this game was not used for inspiration, as the game is meant to be more casual, for players who are just looking for some fun without thinking too much.

After research was carried out, the following things were identified as to what the game will need. A login system is integral to the game, as one of the main ways the game will stand out is by having leaderboards and user stats. To implement said leaderboards and stats, a database will also have to be created. This will have to store a user’s username, total score, game score, which games a user is part of, and what users are in a given game. An SQL database will be used for this. It is also important that multiple games can be played in parallel without interfering with each other, as users might want to play with their friends in a small group rather than in a massive game of all players currently on the website. To implement this, SocketIO rooms will be used to send socket events to specific users.

**Objectives**

1. Users can log in
2. Users can join rooms
3. Users can draw
4. Users can chat
5. Users can see others’ drawings
6. Users can guess drawings
7. Leader boards showing user stats
8. The website looks professional
9. More than one game can be played at once
10. Players can create rooms using a room code
11. Save users scores to use in the leader boards
12. The website has various QOL features that make it enjoyable to use
13. Users’ passwords are encrypted

**Design**

Following the research, some mechanics for the game have been decided, that will differentiate it from the competition. Most of the current games let multiple people guess the word in a single round, with decreasing scores for each subsequent person that guessed the word. In order to make the game faster-paced, only one user will be able to guess the word per round. Hard to draw words would also sometimes ruin the game, as people would either just write the word out and spoil the fun, or not draw anything and hold the game up. To avoid this, a button will be implemented which will randomise the word. The artist can press this button however many times they deem necessary until they find a word they are happy with or are confident they can draw.

Flask will be used for the back end, along with the Flask-SocketIO library to enable use of web sockets. The log in system will make use of the Flask-Login library and SQL databases to store things like user passwords, usernames, scores, etc. These databases will also be used to populate leader boards on the site.

Diagram

Description automatically generated

[Fig 3] Flowchart for

what happens when a

user presses the join

button

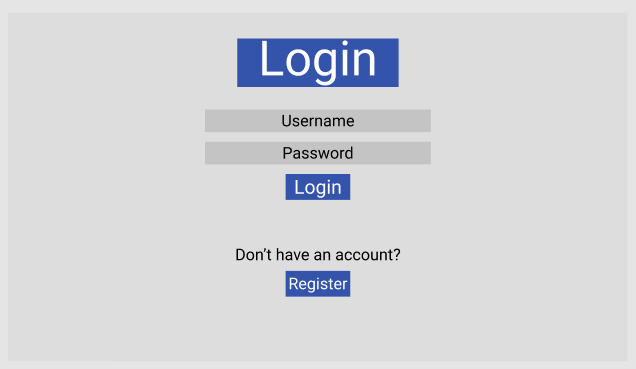
**Home Page**

Graphical user interface, text, application

Description automatically generatedThe website will be very simple and intuitive to navigate, with simple buttons with labels showing what the buttons do. When the user visits the site, they will be taken to a homepage, which will ask them to login or register, and give them information about the game and how it’s played.

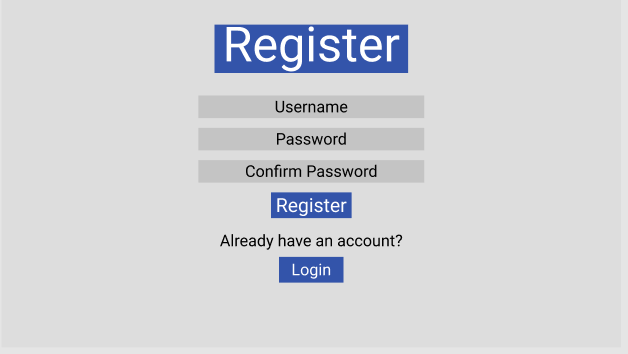
[Figure 4] Mock-up of the home page

**Login Page**

The login page will be very simple, with just entry fields for username and password and a button to register if the user does not yet have an account. A html form will be used to get the contents of the input fields then the server will check the username and password, then log in the user if both these fields are correct.

[Figure 5] Mock-up of the login page

**Register Page**

The register page will also be very simple, with just 3 input fields: one for username, then 2 for password and confirm password. This will also be created with the use of a html form, which will send the values of the input fields to the server, which will check if the 2 passwords match, and if the username is already taken. If everything is okay, the server will create a new SQL entry in the user table with the user’s username and encrypted password. There will also be a button to go to the login page if a user already has an account.

[Figure 6] Mock-up of the register page

**Index Page**

Graphical user interface, application, Teams

Description automatically generatedOnce the user has logged in, they will be taken to the index page, where they can logout, create or join a room, and view the leader boards.

[Figure 7] Mock-up of the index page

**Leader Board Page**

Graphical user interface, application

Description automatically generatedThe leader board page will also be very simple, with the actual leader board, a button to refresh it and a button to go back to the index page. When the refresh button is hit, the client will send a SocketIO request to the server to send the score data from the score table. SQL will be run to return users’ usernames and the corresponding scores, sorted in descending order of score. Once this data is received, the client will put the put this data in an unordered list element and display it on the website.

[Figure 8] Mock-up of the leaderboard page

**Game Pages**

There will be 2 variations of the game page. One that will be shown to the drawer, and one that will be shown to spectators.

Graphical user interface

Description automatically generated The drawing page will have buttons to select brush colour, as well as input fields to manually input hex codes for colour and brush width. It will have the canvas in the centre of the screen, with the chat on the right of it and the word to be drawn on the top of the screen.

[Figure 9] Mock-up of the drawing page

Graphical user interface

Description automatically generatedThe spectator page will be a simplified version of the drawing page, it will basically be the drawing page but without any of the controls related to being the drawer (colour buttons, brush controls, new word button, etc.) It will still have the chat, however, as the chat is required to guess the word.

[Figure 10] Mock-up of the spectator page

**Database**

The database for the game is made up of 3 tables. A user table, a game table, and a linking table used to set up a many to many relationship between user and game. The user table will contain a user id primary key, a username, password – encrypted in sha256, and a score. The score contained in the user table will be the total score the user has gathered over the lifetime of their account. The game table will contain a game id primary and a room code. The room code stored with the game id will be the room code that users will use to connect to the rooms, and the room id will be used to identify each game in the database easily. It also avoids repeating the relatively long room code in the link table compared to the room code. The linking table will have 4 columns: link id (primary key), user id (foreign key pointing to a user), game id (foreign key pointing to a game) and a score. The score in the linking table will be the user’s score for the current game, as opposed to the total score stored in the user table.

The database is designed to allow games contain many users, and users to be in many games. It’s important to be able to check both what games a user is in *and* what users are in a game. This means that the relationship between user and game is many to many bidirectional. This relationship is harder to create, however it was necessary for the application.

[Figure 11] Entity Relationship Diagram for the database

**Implementation**

Server.py

from flask import Flask, render\_template, request, flash, redirect, session

from flask.globals import current\_app, session

from flask.helpers import url\_for

from flask\_login.utils import \_get\_user, login\_required

from flask\_socketio import SocketIO, join\_room, leave\_room, rooms

from flask\_login import LoginManager, UserMixin, login\_user, login\_required, current\_user, logout\_user

from flask\_sqlalchemy import SQLAlchemy

from os.path import dirname, realpath

from werkzeug.security import generate\_password\_hash, check\_password\_hash

import random

import json

import time

newword = "Something"

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'fortnite'

app.config['SQLALCHEMY\_DATABASE\_URI'] ='sqlite:///./login.db'

app.config['SEND\_FILE\_MAX\_AGE\_DEFAULT'] = 0

socketio = SocketIO(app,async\_handlers=True)

sessions = {}

sids = {}

db = SQLAlchemy(app)

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = "/login"

class User(UserMixin, db.Model):

    \_\_tablename\_\_ = "user"

    id = db.Column(db.Integer, primary\_key = True) #Creates an id column, which will be used as the primary key for a user to link tables, required to be called id by flask-login

    username = db.Column(db.String(14), unique = True) #Creates a username column, usernames cant be more than 14 chars

    password = db.Column(db.String(100))

    score = db.relationship("Score")

class Score(db.Model):

    \_\_tablename\_\_ = "score"

    id = db.Column(db.Integer, primary\_key = True)

    player\_id = db.Column(db.ForeignKey("user.id"))

    words\_guessed = db.Column(db.Integer)

    words\_drawn = db.Column(db.Integer)

class Game(db.Model):

    \_\_tablename\_\_ = "game"

    id = db.Column(db.Integer, primary\_key = True)

    room\_code = db.Column(db.String(20))

@login\_manager.user\_loader

def load\_user(id):

    return User.query.get(int(id))

@app.route("/")

@login\_required

def connect():

    user = User.query.filter\_by(username=current\_user.username).first()

    db.session.commit()

    session['username'] = current\_user.username

    return render\_template("index.html", username = user.username)

@app.route("/register")

def register():

    return render\_template("register.html")

@app.route("/register", methods=["POST"])

def register\_post():

    username = request.form.get("username")

    password = request.form.get("password")

    user = User.query.filter\_by(username=username).first()

    if user:

        flash("That Username Is Taken")

        return redirect("/register")

    new\_user = User(username=username, password=generate\_password\_hash(password, method=("sha256")))

    db.session.add(new\_user)

    db.session.commit()

    return redirect("/login")

@app.route("/login")

def login():

    return render\_template("login.html")

@app.route("/login", methods=["POST"])

def login\_post():

    username = request.form.get("username")

    password = request.form.get("password")

    user = User.query.filter\_by(username=username).first()

    if not user or not check\_password\_hash(user.password, password):

        flash("Your username/password is incorrect")

        return redirect("/login")

    login\_user(user)

    return redirect("/")

@app.route("/logout")

@login\_required

def logout():

    logout\_user()

    return redirect("/login")

@socketio.on('drawing')

def handle\_drawing(args):

    print("received drawing")

    #print(session["key"])

    socketio.emit('drawreceive', args)

@socketio.on('chatsubmit')

def handle\_chat(message):

    if message.upper() == newword.upper():

        message = (session["username"] +  " Has Guessed The Word. The Word Was: " + newword)

        for key in sessions:

            print("key" + key)

            print ("session username, " +  session['username'])

            print("sessions[key], " + sessions[key])

            print("sessions[key].clients" + sessions[key].clients)

            if session['username'] in sessions[key].clients:

                for i in sessions[key].clients:

                    socketio.emit('chatprint', message, room = sids[i])

                    break

        time.sleep(3)

        for key in sessions:

            if session['username'] in sessions[key].clients:

                new\_round(key)

                break

    else:

        message = session['username'] + ":" + str(message)

        for key in sessions:

            print("key" + key)

            print ("session username, " +  session['username'])

            #print("sessions[key], " + sessions[key])

            print(sessions[key].clients)

            if session['username'] in sessions[key].clients:

                for i in sessions[key].clients:

                    print(sids[i])

                    socketio.emit('chatprint', message, room = key)

                break

@socketio.on('changeword')

def handle\_word\_change():

    with open("words.json") as f:

        data = json.loads(f.read())

        randomint = random.randint(0,23)

        global newword

        newword = data['words'][randomint]

        socketio.emit('wordchanged', newword)

class Session():

    def \_\_init\_\_(self, roomcode) -> None:

        self.clients = []

        self.code = roomcode

        self.drawer = 0

        self.started = False

@socketio.on("newRound")

def new\_round(room\_code):

    current\_room = sessions[room\_code]

    if (len(sessions[room\_code].clients) == 1):

      current\_room.drawer = 0

    else:

      current\_room.drawer += 1

      current\_room.drawer = current\_room.drawer % len(current\_room.clients)

      socketio.emit('refresh')

@socketio.on("join")

def handle\_joining(room\_code):

    if room\_code in sessions:

        join\_room(room\_code)

        sessions[room\_code].clients.append(session['username'])

        sessions[room\_code].started = True

        sids[session['username']] = request.sid

        print(request.sid)

        socketio.emit('redirect', {'url': url\_for('.gameconnect',r\_code=room\_code)}, room = sids[session['username']])

    else:

        print("Creating room ", room\_code)

        sessions[room\_code] = Session(room\_code)

        sessions[room\_code].clients.append(session['username'])

        join\_room(room\_code)

        print(request.sid)

        sids[session['username']] = request.sid

        socketio.emit('redirect', {'url': url\_for('.gameconnect',r\_code=room\_code)}, room = sids[session['username']])

@app.route("/game/<r\_code>")

@login\_required

def gameconnect(r\_code):

    print(sessions[r\_code].clients[sessions[r\_code].drawer])

    print(session['username'])

    if session['username'] == sessions[r\_code].clients[sessions[r\_code].drawer]:

         return render\_template("game.html")

    else:

        return render\_template("spectate.html")

@socketio.on("newDrawer")

def handle\_joining(room\_code):

    join\_room(room\_code)

    sessions[room\_code].clients.append(session['username'])

    sessions[room\_code].started = True

    sids[session['username']] = request.sid

    socketio.emit('redirect', {'url': url\_for('.gameconnect',r\_code=room\_code)}, room = sids[session['username']])

if \_\_name\_\_ == "\_\_main\_\_":

    socketio.run(app, debug = True)

Gameapp.js

const canvas = document.getElementById("drawcanvas");

const socket = io.connect('http://' + document.domain + ':' + location.port);

const ctx = canvas.getContext("2d");

const todraw = document.getElementById("title");

ctx.canvas.width = 1000;

ctx.canvas.height = 680;

document.getElementById("title").innerHTML = ("Draw: " + word);

const hex = document.getElementById("hex");

const chatinput = document.getElementById("chatinput");

let drawing = false;

const pos = { x: 0, y: 0 };

let uid = null;

console.log(socket.id)

function setPosition(e) { //Gets mouse position relative to the canvas

  const rect = canvas.getBoundingClientRect();

  pos.x = e.clientX - rect.left;

  pos.y = e.clientY - rect.top;

};

socket.emit("changeword")

document.getElementById("chatinput").addEventListener("keyup", function(event) {

  if (event.key === "Enter") {

    chatsubmit();

  };

});

function chatsubmit() { //Send the current text in the chat box to the server then clear the chat box

  if (chatinput.value != ""){

    socket.emit('chatsubmit', chatinput.value);

    chatinput.value = '';

  }

  console.log("message");

};

socket.on('setDrawer', function(room\_code) {

  location.reload();

  socket.emit('newDrawer', room\_code);

});

socket.on('chatprint', function(message){ //When a message comes in, create a new list element then populate it with the message received

  console.log("incoming message");

  const node = document.createElement("li");

  const textnode = document.createTextNode(message);

  node.appendChild(textnode);

  document.getElementById("chat").appendChild(node);

  document.getElementById("chat").scrollTop = document.getElementById("chat").scrollHeight;

});

function draw(e) {

  if (e.buttons !== 1) return;

  const color = document.getElementById("hex").value;

  ctx.beginPath();

  const width = document.getElementById("brush").value;

  ctx.lineWidth = width;

  ctx.lineCap = "round";

  ctx.strokeStyle = color;

  ctx.moveTo(pos.x, pos.y);

  setPosition(e);

  ctx.lineTo(pos.x, pos.y);

  ctx.stroke();

};

function changeWord() {

  socket.emit('changeword');

};

socket.on('wordchanged', function(newword){

  todraw.innerHTML = ("Draw: " + newword);

})

socket.on("refresh", function(){

  location.reload()

})

colors = { red: '#F00', green: '#0F0', blue: '#00F', yellow: '#FF0', orange: '#F80', purple: '#B0F', black: '#000', gray: '#333', gray2: '#666', white: '#FFF' };

function changecolor(color) {

  hex.value = colors[color];

}

setInterval(function() {

  const newUrl = document.getElementById('drawcanvas').toDataURL();

  socket.emit("drawing", newUrl, );

}, 500000);

document.addEventListener("mouseenter", setPosition);

document.addEventListener("mousedown", setPosition);

document.addEventListener("mousemove", draw);

Game.html

<!DOCTYPE html>

<html lang="en">

    <head>

        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

        <script src="https://cdn.socket.io/4.1.2/socket.io.min.js" integrity="sha384-toS6mmwu70G0fw54EGlWWeA4z3dyJ+dlXBtSURSKN4vyRFOcxd3Bzjj/AoOwY+Rg" crossorigin="anonymous"></script>

        <script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>

        <meta charset="utf-8">

        <link href="/static/drawstyle.css" rel="stylesheet">

        <title>

            Paint + Guess

        </title>

        <meta name="viewport" content="width=device-width, initial-scale=1">

    </head>

    <input id="hex" placeholder="Enter Hex Code"></input>

    <input id="brush" placeholder="Brush Width" type="number" min="1" max="100"></input>

    <input id ="chatinput" placeholder="Message" type="text"></input>

    <canvas id="drawcanvas"></canvas>

    <p id="title"></p>

    <ul id="chat">

        <li>Chat Messages Will Appear Here</li>

    </ul>

    <button id="chatsubmit" type="button" onclick="chatsubmit()" class="btn-primary">Send</button>

    <button id="red" type="button" onclick="changecolor('red')" class="btn-primary" color="#F00">Red</button>

    <button id="green" type="button" onclick="changecolor('green')" class="btn-primary" color="#F00">Green</button>

    <button id="blue" type="button" onclick="changecolor('blue')" class="btn-primary" color="#F00">Blue</button>

    <button id="yellow" type="button" onclick="changecolor('yellow')" class="btn-primary" color="#F00">Yellow</button>

    <button id="orange" type="button" onclick="changecolor('orange')" class="btn-primary" color="#F00">Orange</button>

    <button id="purple" type="button" onclick="changecolor('purple')" class="btn-primary" color="#F00">Purple</button>

    <button id="black" type="button" onclick="changecolor('black')" class="btn-primary" color="#F00">Black</button>

    <button id="gray" type="button" onclick="changecolor('gray')" class="btn-primary" color="#F00">Gray</button>

    <button id="gray2" type="button" onclick="changecolor('gray2')" class="btn-primary" color="#F00">Light Gray</button>

    <button id="white" type="button" onclick="changecolor('white')" class="btn-primary" color="#F00">White</button>

    <button id = "word" type="button" onclick="changeWord()">New Word</button>

    <script src= "/static/gameapp.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

</html>

Spectateapp.js

const canvas = document.getElementById("spectatecanvas");

const socket = io.connect('http://' + document.domain + ':' + location.port);

const chatinput = document.getElementById("chatinput");

const ctx = canvas.getContext("2d");

let drawing = false;

let uid = null;

document.getElementById("chatinput").addEventListener("keyup", function(event) {

  if (event.key === "Enter") {

    chatsubmit();

  };

});

function chatsubmit() { //Send the current text in the chat box to the server then clear the chat box

  socket.emit('chatsubmit', chatinput.value);

  chatinput.value = '';

  console.log("message");

};

socket.on('setDrawer', function(room\_code) {

  location.reload()

  socket.emit('newDrawer', room\_code)

});

socket.on('chatprint', function(message){ //When a message comes in, create a new list element then populate it with the message received

  const node = document.createElement("li");

  const textnode = document.createTextNode(message);

  node.appendChild(textnode);

  document.getElementById("chat").appendChild(node);

  document.getElementById("chat").scrollTop = document.getElementById("chat").scrollHeight;

});

socket.on("refresh", function(){

  location.reload();

});

socket.on('drawreceive', function(canvasReceived){

  const receivedImage = new Image(1000,680);

  receivedImage.src = canvasReceived;

  ctx.drawImage(receivedImage, 0, 0);

});

socket.on("NewUID", function(username){

  uid = username;

});

Spectate.html

<!DOCTYPE html>

<html lang="en">

    <head>

        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

        <script src="https://cdn.socket.io/4.1.2/socket.io.min.js" integrity="sha384-toS6mmwu70G0fw54EGlWWeA4z3dyJ+dlXBtSURSKN4vyRFOcxd3Bzjj/AoOwY+Rg" crossorigin="anonymous"></script>

        <script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>

        <meta charset="utf-8">

        <link href="/static/spectatestyle.css" rel="stylesheet">

        <title>

            Paint + Guess

        </title>

        <meta name="viewport" content="width=device-width, initial-scale=1">

    </head>

    <input id ="chatinput" placeholder="Message" type="text"></input>

    <canvas id = "spectatecanvas" width="1000" height="680"></canvas>

    <p id="title"></p>

    <ul id="chat">

        <li>Chat Messages Will Appear Here</li>

    </ul>

    <button id="chatsubmit" type="button" onclick="chatsubmit()">Send</button>

    <script src= "/static/spectateapp.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

</html>

Loginapp.js

var socket = io.connect('http://' + document.domain + ':' + location.port);

function goToRegister(){

    location.href = "/register";

}

Login.hmtl

<!DOCTYPE html>

<html lang="en">

    <head>

        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

        <script src="https://cdn.socket.io/4.1.2/socket.io.min.js" integrity="sha384-toS6mmwu70G0fw54EGlWWeA4z3dyJ+dlXBtSURSKN4vyRFOcxd3Bzjj/AoOwY+Rg" crossorigin="anonymous"></script>

        <script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>

        <meta charset="utf-8">

        <title>

            Paint + Guess

        </title>

        <meta name="viewport" content="width=device-width, initial-scale=1">

    </head>

    <body>

        <div>

            <form action="login" method="POST">

                <div class="container">

                    <div class="d-flex justify-content-center">

                        <div class="row g-2">

                            <div class="col-">

                                <div class="row justify-content-center">

                                    <img src="/static/logo.png" alt="logo" style="width: 1000px;height: 200px;"></img>

                                    <h1 class="display-1">Login</h1>

                                </div>

                                {% with alert = get\_flashed\_messages() %}

                                {% if alert %}

                                    <h3 class="display-20"> {{alert[0]}} </h3>

                                {% endif %}

                                {% endwith %}

                            </div>

                            <div class="col-lg-">

                                <input type="text" name="username" placeholder="Username" class="form-control">

                            </div>

                            <div class="col-lg-">

                                <input type="password" name="password" placeholder="Password" class="form-control">

                            </div>

                            <div class="col-">

                                <div class="row justify-content-center">

                                    <button class="btn btn-primary" style="width:25%">Login</button>

                                </div>

                            </div>

                        </div>

                    </div>

                </div>

            </form>

            <div style="margin-top: 50px;">

                <div class="col-">

                    <div class="row justify-content-center" style="font-size: 15pt;">

                        Don't Have An Account?

                    </div>

                </div>

                <div class="col-">

                    <div class="row justify-content-center">

                        <button class="btn btn-primary" style="width:20%" onclick="goToRegister()">Register</button>

                    </div>

                </div>

            </div>

        </div>

    </body>

    <script src= "/static/loginapp.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

</html>

Registerapp.js

var socket = io.connect('http://' + document.domain + ':' + location.port);

function goToLogin(){

    location.href = "/login";

}

Register.html

<!DOCTYPE html>

<html lang="en">

    <head>

        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

        <script src="https://cdn.socket.io/4.1.2/socket.io.min.js" integrity="sha384-toS6mmwu70G0fw54EGlWWeA4z3dyJ+dlXBtSURSKN4vyRFOcxd3Bzjj/AoOwY+Rg" crossorigin="anonymous"></script>

        <script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>

        <meta charset="utf-8">

        <title>

            Paint + Guess

        </title>

        <meta name="viewport" content="width=device-width, initial-scale=1">

    </head>

    <body>

        <div>

            <form action="register" method="POST">

                <div class="container">

                    <div class="d-flex justify-content-center">

                        <div class="row g-2">

                            <div class="col-">

                                <div class="row justify-content-center">

                                    <img src="/static/logo.png" alt="logo" style="width: 1000px;height: 200px;"></img>

                                    <h1 class="display-1">Register</h1>

                                </div>

                                {% with alert = get\_flashed\_messages() %}

                                {% if alert %}

                                    <h3 class="display-20"> {{alert[0]}} </h3>

                                {% endif %}

                                {% endwith %}

                            </div>

                            <div class="col-lg-">

                                <input type="text" name="username" placeholder="Username" class="form-control">

                            </div>

                            <div class="col-lg-">

                                <input type="password" name="password" placeholder="Password" class="form-control">

                            </div>

                            <div class="col-">

                                <div class="row justify-content-center">

                                    <button class="btn btn-primary" style="width:25%">Register</button>

                                </div>

                            </div>

                        </div>

                    </div>

                </div>

            </form>

            <div style="margin-top: 50px;">

                <div class="col-">

                    <div class="row justify-content-center" style="font-size: 15pt;">

                        Already Have An Account?

                    </div>

                </div>

                <div class="col-">

                    <div class="row justify-content-center">

                        <button class="btn btn-primary" style="width:20%" id="goToRegister" onclick="goToLogin()">Login</button>

                    </div>

                </div>

        </div>

    </body>

    <script src= "/static/registerapp.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

</html>

Indexapp.js

var socket = io.connect('http://' + document.domain + ':' + location.port);

function joinroom(){

    room\_code = document.getElementById("room\_code").value

    socket.emit('join', room\_code);

}

document.getElementById("room\_code").addEventListener("keyup", function(event) {

    if (event.key === "Enter") {

        joinroom();

    };

});

function logout(){

    location.href = "/logout"

}

socket.on('redirect', function(data) {

    window.location = data.url;

});

Index.html

<!doctype html>

<html lang="en">

    <head>

        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

        <script src="https://cdn.socket.io/4.1.2/socket.io.min.js" integrity="sha384-toS6mmwu70G0fw54EGlWWeA4z3dyJ+dlXBtSURSKN4vyRFOcxd3Bzjj/AoOwY+Rg" crossorigin="anonymous"></script>

        <script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>

        <meta charset="utf-8">

        <title>

            Paint + Guess

        </title>

        <meta name="viewport" content="width=device-width, initial-scale=1">

    </head>

    <h2 style="margin: 10px;">Welcome, {{ username }}</h2>

    <div class="row justify-content-center" style="margin: 10px;">

        <input id ="room\_code" type="text" placeholder="Room Code" class="form-control" style="width:20%">

        <div class="col" style="margin-top: 5px;">

            <button id="joinroom" type="button" onclick="joinroom()" class="btn-primary">Join</button>

            <button id="logout" type="button" onclick="logout()" class="btn-primary">Logout</button>

        </div>

    </div>

    <script src= "/static/indexapp.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

</html>

**Testing**

Multiple games were hosted in computer science lessons in order to gather feedback on the game and to find bugs, the following are the testing sessions and the feedback gathered from them.

|  |  |  |
| --- | --- | --- |
| **Test Date** | **Feedback** | **Action** |
| 27/01/2022 | “The website is hard to navigate” | Added buttons to the login, register and join game pages to navigate to other pages. |
| 27/01/2022 | “The game is very confusing as to who is drawing, who guessed the word, etc” | Added a period after someone guesses the word where chat displays who guessed the word before switching artists. |
| 27/01/2022 | “The spacing on the game page looks bad” | Amended the margins of the different elements on the game pages to make them look better |
| 27/01/2022 | “You can register with an empty password” | Implemented regex for password isn’t empty or lots of spaces |
| 27/01/2022 | “You cannot send chat messages by pressing the enter button” | Added functionality that lets users send messages by pressing enter. |
| 31/01/2022 | “When there are multiple games running, one game can influence another” | The bug was fixed, now actions in one room don’t affect a different room. |
| 31/01/2022 | “You can’t do SQL injection on the register page” | This is a good thing, and therefore there was no change made to this. |
| 31/01/2022 | “There aren’t very many words” | [TODO – ADD A LOT OF WORDS] |
|  |  |  |

**Evaluation**

Overall, the project was a success. All of the objectives have been achieved to a good standard, as well as a few little things that could not have been a good objective on their own. The project could be improved on, however. A major way this could be done is through more statistics being added to the game. For example, how many games the user has played, or average score per game.