**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, which could possibly harm the experience for younger users.

**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. Accounts
   1. Users can log in
   2. Leader boards showing user’s scores
   3. Save user’s scores to an SQL database
   4. Encrypt users’ passwords for security
2. Game
   1. Users can join rooms using a room code
   2. Users can draw
   3. Users can chat
   4. Users can see the current drawer’s drawings
   5. Users can type in chat to guess what the drawer is currently drawing
   6. Multiple games can be played in parallel without affecting each other
3. Forum
   1. There is a forum that will let users discuss various topics
   2. Users can make threads that support CRUD (Create/Read/Update/Delete)
   3. Users can make CRUD replies to threads
   4. Word suggestion thread that lets users suggest new words to be permanently added to the game
   5. Users can upvote or downvote threads and replies
   6. Users will receive notifications with updates to their threads/ replies
   7. All posts will be stored in an SQL database

**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.

Diagram

Description automatically generatedFlask 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.

[Fig 3] Flowchart for

what happens when a

user presses the join

button

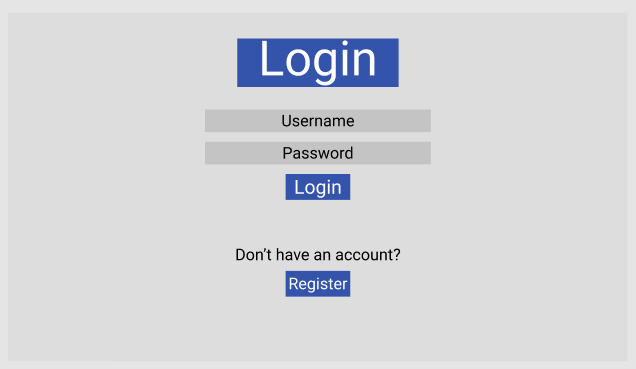
**Home Page**

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.

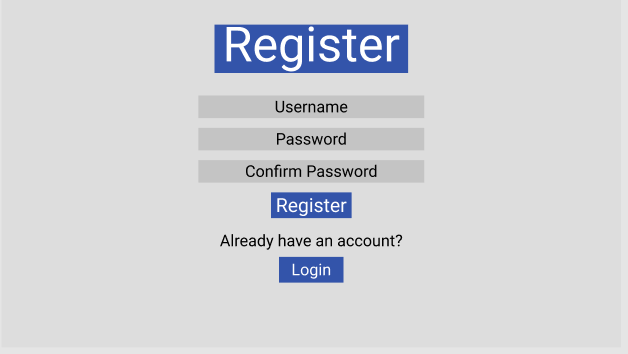
[Figure 4] Mock-up of the home page

**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.

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.

Graphical user interface

Description automatically generated[Figure 9] Mock-up of the drawing page

The 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.

Graphical user interface

Description automatically generated[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

There will also need to be a database system for the forums, which will use the user table mentioned previously, but will also need many more new tables. It will need a table storing all threads, which will link to the user which created the thread. The thread will also be linked to all of the replies in the thread, which will themselves also be linked to the users who posted the replies. The threads and replies will need to store their content, with threads also needing to store their title, as well as both needing to store the amount of upvotes and downvotes on the post.

Implementing the voting system on threads comes with some challenges, as there are a lot of possible ways to implement it and each come with their own pros and cons. A possible solution is keeping a list of all users that voted on the post, and what their vote was on the post table, however this creates problems with needing to query the post table every time a user visits a thread to see if they have already voted for it and what their vote was, then sending that data to the client to change the colour of the voting button on the user’s gui to represent their vote. The pros of this method, however, are that it will make it very easy to calculate the total number of upvotes on the post when it first gets shown to the user. A better method would be to store the total number of upvotes on the post in its table, and a list of all of the posts that a user has voted on in the user table. This would make it very simple not only to see the total number of votes on the post, but would also provide a more elegant solution to the problem of showing the user their vote on a certain post, and not letting them vote on the same post twice.

**Technical Solution**

Server.py

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

from flask.globals import session

from flask.helpers import url\_for

from flask\_login.utils import login\_required

from flask\_socketio import SocketIO, join\_room

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

from flask\_sqlalchemy import SQLAlchemy

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

import random

import json

import time

import re

db = SQLAlchemy()

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

app.config['SECRET\_KEY'] = 'HNz898OEWw3qdq8tpkeatPC8GqvExMdw'

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

app.config['SEND\_FILE\_MAX\_AGE\_DEFAULT'] = 0 #This line makes the browser not cache any files, which is very useful for testing and also going to be kept like this as it allows for very easy deployment of any updates to the website.

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

db.init\_app(app)

sessions = {} #A dictionary of sessions, so that each room is easily accessible by its room code

sids = {} #A dictionary of session ids, so that each user's session id is easily accessible with their username.

login\_manager = LoginManager()

login\_manager.init\_app(app)

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

class User(UserMixin, db.Model): #This class is an sqlalechemy class, which creates the User SQL table. UserMixin is a flask\_login object that lets the user table be used to login users.

    \_\_tablename\_\_ = "user" #This sets the name of the table to user

    id = db.Column(db.Integer, primary\_key = True) #The primary key for a user. This variable has to be called id for flask-login to work properly

    username = db.Column(db.String(14)) #The username of a user. This cannot be longer than 14 characters for storage space purposes.

    password = db.Column(db.String(100)) #A user's password. This is going to be encrypted with sha256 for security.

    score = db.Column(db.Integer) #This will be the total score that the user has acquired over their account lifetime.

    games = db.relationship("LinkingTable", back\_populates="user") #This creates a relationship object between the user and the linking table, which will be used to create a many to many relationship between games and users.

class Game(db.Model):  #This class is an SQLAlchemy class, which creates an SQL table called Game.

    \_\_tablename\_\_ = "game" #This line just sets the name of the table

    game\_id = db.Column(db.Integer(), primary\_key = True, unique = True) #This is the primary key of the table, every game will have an integer id used to identify it in the database.

    game\_code = db.Column(db.String) #This game code will be the room code that users use to connect to the room. Storing it with the game id makes it easy to query the databse with the room code.

    users = db.relationship("LinkingTable", back\_populates="game") #This creates a relationship object to the linking table, which then links the game to the users in a many to many relationship.

class LinkingTable(db.Model): #This is an SQLAlchemy class that creates an SQL table called Linking Table.

    \_\_tablename\_\_ = "link" #This sets the name of the table to "link"

    link\_id = db.Column(db.Integer(), primary\_key = True, unique = True) #This is the primary key of the table, every link between game and user will have na integer id to make it easy to identify.

    game\_id = db.Column(db.Integer(), db.ForeignKey("game.game\_id")) #This foreign key will match a game id in the game table, and will be used along with the user id to create a many to many link between the user and the game.

    user\_id = db.Column(db.Integer(), db.ForeignKey("user.id")) #This foreign key will match a user id in the user table, and will be used along with the game id to create a many to many relationship between user and game

    score = db.Column(db.Integer()) #This stores a users score for the specifiedd game. This is different to the score that is stored in the User table, which is a total of all the score that user has gotten.

    user = db.relationship("User", back\_populates="games") #This creates a relationship between the link table and user table, back\_poulates tells the database that the relationship will be used to link the user to a game.

    game = db.relationship("Game", back\_populates="users") #This createsa  relationship between the link table and game table, the back\_populates parameter tells the database that the game will be linked to a user.

db.create\_all(app=app) #This line creates the database, using the Flask app object.

@login\_manager.user\_loader #This is the user loader, flask-login uses this to find the correct user to login when login is called.

def load\_user(id):

    return User.query.get(int(id)) #This queries the user database by id, then returns the user with that id.

@app.route("/") #This is the default route that users are taken to.

@login\_required #This makes it so this page is inaccessible unless a user is logged in with flask-login.

def connect():

    user = User.query.filter\_by(username=current\_user.username).first() #The user database is queried using the current username, then the user with that username is returned to this variable.

    db.session.commit() #This saves all the pending changes to the SQL database.

    session['username'] = current\_user.username #The session object is a cookie that stores various data about the user and can be easily accessed by the server. Here, the username is stored in the session cookie.

    return render\_template("index.html", username = user.username) #This sends the client the index html page, with the username. The username is added to the page with jinja.

@app.route("/register") #This is the route that users are taken to to register a new account.

def register():

    return render\_template("register.html") #This send the user the register html page.

@app.route("/register", methods=["POST"]) #This is the registration route, that takes in the username and password from a html form as a POST body.

def register\_post():

    username = request.form.get("username") #Makes the username from the form into a variable

    password = request.form.get("password") #Makes the password from the form into a variable

    if re.fullmatch(" \*", password):  #This is regex to make sure the password isnt empty. The " \*" means that if the password is made up of any number of spaces, it will be rejected.

        flash("Please enter a password") #Flash lets you run code in html using jinja, this runs the code on the register.html page and passes the message "please enter a password".

        return redirect("/register")

    elif len(password) < 8: #This makes sure that the password the user enters is at least 8 characters long.

        flash("Your password has to be at least 8 characters long") #Flashes the message if the password is less than 8 characters

        return redirect("/register")

    user = User.query.filter\_by(username=username).first() #If all of the password checks are completed, this queries the sql user database to check if a user with that username already exists.

    if user: #If the query returns a user, this code is run to let the user know the username is taken.

        flash("That Username Is Taken")

        return redirect("/register")

    new\_user = User(username=username, password=generate\_password\_hash(password, method=("sha256")),score=0) #This creates a new user object, with the entered username and the password encrypted with sha256. It also adds a score to the user and sets it to 0.

    db.session.add(new\_user) #This makes a new entry in the user table using the new\_user object to populate it.

    db.session.commit() #This commits the sql session.

    return redirect("/login") #Once a user has finished logging in, they are redirected to the login page to login.

@app.route("/login") #This is the route that users are taken to to login, and whenever they access a page that needs to have a user logged in to be accessed.

def login():

    return render\_template("login.html") #Sends the login page to the user.

@app.route("/login", methods=["POST"])  #This is the login route, that takes in the username and password from a html form as a POST body.

def login\_post():

    username = request.form.get("username") #Makes the username from the form into a variable

    password = request.form.get("password") #Makes the password from the form into a variable

    user = User.query.filter\_by(username=username).first() #This queries the user table to see if a user with the same username as the one entered exists.

    if not user or not check\_password\_hash(user.password, password): #This checks if the user with the username exists, and if it does it checks the password entered with the one saved in the database using sha256

        flash("Your username/password is incorrect") #If the login is unsuccessful, this message is flashed onto the screen to let the user know that their username/ password is incorrect.

        return redirect("/login") #This redirects the user back to the login page.

    session["id"] = user.id #This saves the user's id in the session cookie for easy access by the server

    login\_user(user) #Once everything else is done flask-login logs in the user with the user object from sqlalchemy

    return redirect("/") #The user is then redirected to the index page.

@app.route("/logout") #When the user presses the logout button or types the address into the address bar, this code is run.

@login\_required #The user needs to be logged in in order to log out, if they’re not logged in they’re taken to the login page

def logout():

    logout\_user() #This is a flask-login method to logout the currently logged in user.

    return redirect("/login") #After the user is logged out, they are taken to the login page.

@socketio.on('drawing') #This runs whenever the server receives a drawing event, which the artist sends every 0.5 seconds.

def handle\_drawing(args):

    for key in sessions: #This runs through all of the sessions

            if session['username'] in sessions[key].clients: #If the person who sent the drawing is in the current session this activates

                for i in sessions[key].clients: #This runs through every client in the session

                    socketio.emit('drawreceive', args, room=sids[i]) #This then sends every client in the session the drawing that was sent to the server

            break #This stops the sessions being iterated once the correct one has been found

@socketio.on('chatsubmit') #This runs whenever a user sends a chat message

def handle\_chat(message):

    for key in sessions: #This iterates through all of the sessions

            if session['username'] in sessions[key].clients: #If the user who sent the message is in the current session, it saves the room key of the current session then breaks the loop so as to not keep iterating through the sessions needlessly

                goodkey = key

                break

    if message.upper() == sessions[goodkey].word.upper(): #This checks if the message that has been sent is the word that is the current word that is meant to be guessed.

        message = (session["username"] +  " Has Guessed The Word. The Word Was: " + sessions[goodkey].word) #If this is the case, a chat message is sent telling the other players who guessed the word and what it was.

        userrow = db.session.query(User).filter\_by(id = session["id"]).first() #The user table is queried using the user id, then the row corresponding to the current user is returned to the variable.

        userrow.score += 1 #The user's score is incremented by one

        gamerow = db.session.query(Game).filter\_by(game\_code = goodkey).first() #The game table is queried using the game code, then the row corresponding to the room with that room code is returned to the variable. This will allow the game id to be read from the variable.

        assocrow = db.session.query(LinkingTable).filter\_by(user\_id = session["id"], game\_id = gamerow.game\_id).first() #This queries the LinkingTable table by the user id and game id, then returns the link between the game and the user to the variable. This lets the user's game score be incremented.

        assocrow.score += 1 #The user's game score is incremented by one

        db.session.commit() #The changes are then commited to the database

        for i in sessions[goodkey].clients: #This iterates through all the users in the current session

            socketio.emit('chatprint', message, room = sids[i]) #This sends the chat message to every user in the session

        time.sleep(3) #The program waits for 3 seconds before starting a new round in order to let everyone realise that the word was guessed, and to let them prepare for a the next round

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

            new\_round(goodkey) #Starts a new round in the room specified by the good key.

    else:

        message = session['username'] + ":" + str(message) #If the message wasn’t a correct guess, the message becomes the user's who sent the message username, followed by a colon then their message.

        for i in sessions[goodkey].clients: #Iterates through all the clients in the current session

            socketio.emit('chatprint', message, room = sids[i]) #Sends the message to all the clients in the current session

@socketio.on('changeword') #This runs whenever a user pressed the new word button, which sends a "changeword" event.

def handle\_word\_change():

    with open("words.json") as f: #This opens the json file of all the possible words.

        data = json.loads(f.read()) #This reads the json file and puts its contents into the data variable

        randomint = random.randint(0,66) #This picks a random number between 0 and 62 (the number of words in the json), which will be used to decide which word to use

        for key in sessions: #This iterates through all the sessions

            if session["username"] in sessions[key].clients: #If the user who requested the word change is in the current session, the word becomes a random word chosen from the data variable.

                sessions[key].word = data['words'][randomint]

                break #This breaks the loop so as to not needlessly iterate through all the sessions

        socketio.emit('wordchanged', sessions[key].word, room=sids[session["username"]]) #This emits the new word to the user who requested it, which upon being received will be displayed at the top of their screen.

class Session(): #This is a session (also known as a room) class, it stores a list of clients, the code for the room, the index of the current drawer, wether the game has started, and the current word.

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

        self.clients = []

        self.code = roomcode

        self.drawer = 0

        self.started = False

        self.word = "Square"

@socketio.on("newRound") #This is called whenever a word is guessed or when a game starts

def new\_round(room\_code): #The room code decides which room the new round is started in

    current\_room = sessions[room\_code] #This just makes the room code easily accessible with a variable

    if (len(sessions[room\_code].clients) == 1): #If theres only one person in the room, then they become the drawer.

      current\_room.drawer = 0

    else:

      current\_room.drawer += 1 #Increment the drawer index by one.

      current\_room.drawer = current\_room.drawer % len(current\_room.clients) #This mods the drawer index with the amount of people in the room, so that the index cycles through each of the clients without going out of range.

      for i in current\_room.clients:

        socketio.emit('refresh', room = sids[i]) #Sends a refresh socket event to each user in the room so that their pages are refreshed and they get the correct gui for their role (drawer/spectator)

@socketio.on("join") #This runs whenever someone joins a room

def handle\_joining(room\_code):

    if room\_code in sessions: #If the room already exists, the following code runs

        join\_room(room\_code) #Adds the user to the socketio room

        gamerow = db.session.query(Game).filter\_by(game\_code = room\_code).first() #When a user joins a room, the Game table is queried using the room code to find the room row with that room code.

        newassoc = LinkingTable(user\_id = session["id"],score=0,game\_id = gamerow.game\_id) #This creates a new entry in the linking table, with the current user id, the current game id and the users score, which is currently 0.

        db.session.add(newassoc) #This adds the new entry to the database

        db.session.commit() #This commits the pending changes to the database.

        sessions[room\_code].clients.append(session['username']) #Adds the user to the clients list of the session

        sessions[room\_code].started = True #Marks the session as started

        sids[session['username']] = request.sid #This saves the user's session id in the sids dictionary so its easy to find the sid using the username

        socketio.emit('redirect', {'url': url\_for('.gameconnect',r\_code=room\_code)}, room = sids[session['username']]) #Redirects the user to the url for their current game

    else:

        print("Creating room ", room\_code) #If the room doesnt already exist, it creates a new room

        newgamerow = Game(game\_code = room\_code) #This creates a new entry in the Game table with the room code specified by the user.

        db.session.add(newgamerow) #This adds the new game entry to the database.

        db.session.commit() #This commits all the pending changes to the database.

        gamerow = db.session.query(Game).filter\_by(game\_code = room\_code).first() #This queries the game table with the room code to find the game id.

        newassoc = LinkingTable(user\_id = session["id"],score=0,game\_id = gamerow.game\_id) #This creates a new entry in the linking table with the game id and user id, as well as a score for the game, which is set to 0.

        db.session.add(newassoc) #This adds the entry to the database

        db.session.commit() #This commits all pending changes to the database.

        sessions[room\_code] = Session(room\_code) #Create a new session and add it to the sessions dictionary so its easily accessible with its room code

        sessions[room\_code].clients.append(session['username']) #Adds the current user to the new room's clients list

        join\_room(room\_code) #Adds the user to the socketio room

        sids[session['username']] = request.sid #This saves the user's session id in the sids dictionary so its easy to find the sid using the username

        socketio.emit('redirect', {'url': url\_for('.gameconnect',r\_code=room\_code)}, room = sids[session['username']]) #Redirects the user to the url for their current game

@app.route("/game/<r\_code>") #The route for games, <r\_code> makes it so that any room code can be used and it will still lead to this route

@login\_required #You need to be logged in to be in a game room

def gameconnect(r\_code):

    if session['username'] == sessions[r\_code].clients[sessions[r\_code].drawer]: #If the client is a drawer in the current game, it sends them the game.html

         return render\_template("game.html")

    else:

        return render\_template("spectate.html") #If a client is a spectator, it sends them the spectate.html

@socketio.on("syncSID") #Whenever a client emits a syncsid event (Every 30 seconds) this gets run

def handle\_sids():

    sids[session["username"]] = request.sid #This saves the user's session id in the sids dictionary for easy access to their sid

@app.route("/leaderboard") #This is the route that users will visit to see the leaderboards, they dont need to be logged in to view it as its not very important.

def handle\_leaderboards():

    return render\_template("leaderboard.html")

@socketio.on("requestleader") #Whenever a client requests a leaderboard update, this code is run.

def send\_leader(): #This SQL queries the User table, and returns all of the usernames and their corresponding scores, in descending order so that the client doesn't have to sort it. This is an example of thin-client computing.

    table = db.engine.execute("""

    SELECT user.username, user.score

    FROM user

    WHERE user.score IS NOT NULL

    ORDER BY user.score DESC

    """)

    score = []

    for row in table: #This iterate through every row returned by the query

        newrow = [row[0],row[1]] #This creates an array with each entry of a username along with their scores so that the score array can be emitted on the socket

        score.append(newrow) #This adds the newrow array into the score 2d array so that its ready to be sent by the server

    socketio.emit("sendleader", score, room = request.sid) #This emits the score array to the user who requested it.

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

    socketio.run(app, debug = True, host="0.0.0.0", port=5000) #This runs the flask server to enable clients to connect to it.

Gameapp.js

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

const socket = io.connect('http://' + document.domain + ':' + location.port); //This creates a socket connection with the flask server

socket.emit("syncSID") //This synchronises the socket id with the server so that the server can contact this user directly.

socket.emit("changeword") //This emits a change word event so that the user can get a word to draw as fast as possible.

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

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

ctx.canvas.width = 1000;

ctx.canvas.height = 680;

document.getElementById("title").innerHTML = ("Draw: Something"); //Sets the default title to say draw: if the client hasnt yet received a word from the server

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

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

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

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

  const rect = canvas.getBoundingClientRect();

  pos.x = e.clientX - rect.left;

  pos.y = e.clientY - rect.top;

};

document.getElementById("chatinput").addEventListener("keyup", function(event) { //This function lets the user send chat messages by pressing the enter button on their keyboard, without having to pres the send message button with their mouse.

  if (event.key === "Enter") {

    chatsubmit();

  };

});

function chatsubmit() { //This function reads the current text in the chat input field, if its empty the function doesnt run

  if (chatinput.value != ""){

    socket.emit('chatsubmit', chatinput.value); //This emits a chatsubmit socket event, along with the message that the user wrote.

    chatinput.value = ''; //This empties the chat input field.

  }

};

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"); //Creates a list element

  const textnode = document.createTextNode(message); //Creates a text node and populates it with the message

  node.appendChild(textnode); //adds the text node to the list element

  document.getElementById("chat").appendChild(node); //Adds the list element to the chat, which is an unordered list.

  document.getElementById("chat").scrollTop = document.getElementById("chat").scrollHeight; //Lets the chat box be scrolled properly whenever a message is added to it

});

function draw(e) { //This is the function used to enable drawing on the canvas by the user.

  if (e.buttons !== 1) return; //If the mouse button isnt pressed, the function doesn't run.

  const color = document.getElementById("hex").value; //This gets the value of the hex input field on the drawing page then sets the brush color to the color corresponding to the hex code.

  ctx.beginPath(); //This begins a brush path at the current mouse position.

  const width = document.getElementById("brush").value; //This gets the value of the width input field then sets the brush width to said value.

  ctx.lineWidth = width;

  ctx.lineCap = "round"; //This makes the brush circular

  ctx.strokeStyle = color;

  ctx.moveTo(pos.x, pos.y); //This moves the context variable to the current mouse position relative to the canvas.

  setPosition(e); //This updates the mouse position relative to the canvas.

  ctx.lineTo(pos.x, pos.y); //This makes a line from the position created by beginPath()

  ctx.stroke(); //This colors in the line created by lineTo()

};

function changeWord() { //This function sends out a changeword socket event, which the server receives, chooses a random word then sends it back to the user with a word changed event

  socket.emit('changeword');

};

socket.on('wordchanged', function(newword){ //This function changes the word that appears on top of the drawer's screen that they have to draw whenever the server sends out a word changed event

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

})

socket.on("refresh", function(){ //This function refreshes the page every time the server sends a refresh message.

  location.reload()

})

colors = { red: '#F00', green: '#0F0', blue: '#00F', yellow: '#FF0', orange: '#F80', purple: '#B0F', lightblue:'#0FF', black: '#000', white: '#FFF' }; //Dictionary of colors used to set the brush color so that a new function doesnt have to be written for each button and its color.

function changecolor(color) { //This function sets the color that the brush will be set to on click of the button given the color of the button.

  hex.value = colors[color];

}

setInterval(function() {   //This function emits a drawing socket message, along with a representation of the current canvas to the server every 0.5 seconds.

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

  socket.emit("drawing", newUrl);

}, 250);

setInterval(function() {  //This function emits a syncSID socket message to the server every 30 seconds. This technically isnt 100% necessary as every time a user joins a game or a new round starts, the sids are updated. This is only here as a precaution incase somehow the sids change.

  socket.emit("syncSID");

}, 30000);

document.addEventListener("mouseenter", setPosition); //These event listeners are here so that the drawing works. Every time the cursor enters the area of the canvas, clicks or moves it updates its position.

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>

    <h1 id="title" class="display-1" style="text-align: center; color: #3354AA;"></h1>

    <div class="container" style="width: 100vw; margin: 0; display: fixed;">

        <div class="row" style="width: 100vw; align-content: center;">

            <div class="col" style="margin: 5px; width: 10vw; max-width: 145px; max-height: 240px; flex-direction: column; align-content: center; padding: 12px;">

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

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

                <div class="col" style="align-content: center; width: 100%; flex-direction: column; align-items: center; margin-left: 4px;">

                    <button id="red" type="button" onclick="changecolor('red')" class="btn colorButton" style="background-color: #F00; width: 25px;"></button>

                    <button id="green" type="button" onclick="changecolor('green')" class="btn colorButton" style="background-color: #0F0; width: 25px;"></button>

                    <button id="blue" type="button" onclick="changecolor('blue')" class="btn colorButton" style="background-color: #00F; width: 25px;"></button>

                    <button id="yellow" type="button" onclick="changecolor('yellow')" class="btn colorButton" style="background-color: #FF0; width: 25px;"></button>

                    <button id="orange" type="button" onclick="changecolor('orange')" class="btn colorButton" style="background-color: #F80; width: 25px;"></button>

                    <button id="purple" type="button" onclick="changecolor('purple')" class="btn colorButton" style="background-color: #F0F; width: 25px;"></button>

                    <button id="lightblue" type="button" onclick="changecolor('lightblue')" class="btn colorButton" style="background-color: #0FF; width: 25px; height: 25px;"></button>

                    <button id="black" type="button" onclick="changecolor('black')" class="btn colorButton" style="background-color: #000; width: 25px;"></button>

                    <button id="white" type="button" onclick="changecolor('white')" class="btn colorButton" style="background-color: #FFF; width: 25px;"></button>

                </div>

                <button id = "word" type="button" onclick="changeWord()" class="btn" style="background-color: #3354AA; height: 30px; width: 120px; padding: 0;">New Word</button>

            </div>

            <div class="col" style="margin: 0;">

                <canvas id="drawcanvas"></canvas>

            </div>

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

                <ul id="chat">

                    <li style="text-align: center; font-weight: bold;">Chat Messages Will Appear Here</li>

                </ul>

                <div class="row" style="margin-left: 0px; margin-top: 5px;">

                    <input id ="chatinput" placeholder="Message" type="text" class="form-control" style="background-color: #C4C4C4; color: #000;"></input>

                    <button id="chatsubmit" type="button" onclick="chatsubmit()" class="btn" style="padding: 0; margin: 0; background-color: #3354AA; margin-left: 5px; ">Send</button>

                </div>

            </div>

        </div>

    </div>

    <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); //This creates a socket connection with the flask server

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

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

socket.emit("syncSID") //This synchronises the socket id with the server so that the server can contact this user directly.

document.getElementById("chatinput").addEventListener("keyup", function(event) { //This function lets the user send chat messages by pressing the enter button on their keyboard, without having to pres the send message button with their mouse.

  if (event.key === "Enter") {

    chatsubmit();

  };

});

function chatsubmit() { //This function reads the current text in the chat input field, if its empty the function doesnt run

  if (chatinput.value != ""){

    socket.emit('chatsubmit', chatinput.value); //This emits a chatsubmit socket event, along with the message that the user wrote.

    chatinput.value = ''; //This empties the chat input field.

  }

};

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"); //Creates a list element

  const textnode = document.createTextNode(message); //Creates a text node and populates it with the message

  node.appendChild(textnode); //adds the text node to the list element

  document.getElementById("chat").appendChild(node); //Adds the list element to the chat, which is an unordered list.

  document.getElementById("chat").scrollTop = document.getElementById("chat").scrollHeight; //Lets the chat box be scrolled properly whenever a message is added to it

});

socket.on("refresh", function(){ //This function refreshes the page every time the server sends a refresh message.

  location.reload()

});

setInterval(function() { //This function emits a syncSID socket message to the server every 30 seconds. This technically isnt 100% necessary as every time a user joins a game or a new round starts, the sids are updated. This is only here as a precaution incase somehow the sids change.

  socket.emit("syncSID");

}, 30000);

socket.on('drawreceive', function(canvasReceived){ //This function runs whenver the client receives a canvas from the server.

  const receivedImage = new Image(1000,680); //This creates a new Image element with width of 1000 pixels and height 680 pixels.

  receivedImage.src = canvasReceived; //This makes the source of the image element the encoded string received by the client.

  ctx.drawImage(receivedImage, 0, 0); //This draws the received image on the context element of the canvas, the 0,0 means it draws the image starting from the top left corner of the canvas.

});

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>

    <h1 id="title" class="display-1" style="text-align: center; color: #3354AA;">Guess The Word!</h1>

    <div class="container" style="width: 100vw; margin: 0; display: fixed;">

        <div class="row" style="width: 100vw; align-content: center;">

            <div class="col" style="margin: 5px; width: 10vw; max-width: 145px; max-height: 240px; flex-direction: column; align-content: center; padding: 12px;">

            </div>

            <div class="col" style="margin: 0;">

                <canvas id="spectatecanvas"></canvas>

            </div>

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

                <ul id="chat">

                    <li style="text-align: center; font-weight: bold;">Chat Messages Will Appear Here</li>

                </ul>

                <div class="row" style="margin-left: 0px; margin-top: 5px;">

                    <input id ="chatinput" placeholder="Message" type="text" class="form-control" style="background-color: #C4C4C4; color: #000;"></input>

                    <button id="chatsubmit" type="button" onclick="chatsubmit()" class="btn" style="padding: 0; margin: 0; background-color: #3354AA; margin-left: 5px; ">Send</button>

                </div>

            </div>

        </div>

    </div>

    <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);//This creates a socket connection with the flask server

function goToRegister(){ //This function takes the user to the register route when they press the login button

    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 style="background-color: #DDD;">

        <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">

                                    <h1 class="display-1" style="text-align: center; color: #3354AA;">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" style="background-color: #C4C4C4; color: #000;">

                            </div>

                            <div class="col-lg-">

                                <input type="password" name="password" placeholder="Password" class="form-control" style="background-color: #C4C4C4; color: #000;">

                            </div>

                            <div class="col-">

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

                                    <button class="btn btn-primary" style="width:10vw; background-color: #3354AA;">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:10vw; background-color: #3354AA;" 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); //This creates a socket connection with the flask server

function goToLogin(){ //This function takes the user to the login route when they press the login button.

    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 style="background-color: #DDD;">

        <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">

                                    <h1 class="display-1" style="text-align: center; color: #3455AA;">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" style="background-color: #C4C4C4">

                            </div>

                            <div class="col-lg-">

                                <input type="password" name="password" placeholder="Password" class="form-control" style="background-color:  #C4C4C4;">

                            </div>

                            <div class="col-">

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

                                    <button class="btn btn-primary" style="width: 10vw; background-color: #3455AA">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%; background-color: #3455AA;" 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); //This creates a socket connection with the flask server

function joinroom(){

    room\_code = document.getElementById("room\_code").value //This function sends a join socket event along with the room code that the user entered to the server

    socket.emit('join', room\_code);

}

document.getElementById("room\_code").addEventListener("keyup", function(event) { //This function allows the user to press enter to join a room, so that they dont have to press the join room button with their mouse every time.

    if (event.key === "Enter") {

        joinroom();

    };

});

function logout(){ //This function takes the user to the logout route when they press the logout button on the page

    location.href = "/logout"

}

socket.on('redirect', function(data) { //This function redirects the user to the address specified by the event, the server will send the request to redirect along with the address to redirect to any time it needs the user to be redirected

    window.location = data.url;

});

function leaderboard(){ //This function takes the user to the leaderboard route when they press the leaderboard button on the page

    location.href = "/leaderboard"

}

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>

    <body style="background-color: #DDD;">

        <div class="container">

            <h1 class="display-1" style="margin: 10px; text-align: center; color: #3455AA;">Welcome, {{ username }}</h1>

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

                <div class="col">

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

                        <input id ="room\_code" type="text" placeholder="Room Code" class="form-control" style="width: 30%;">

                        <button id="joinroom" type="button" onclick="joinroom()" class="btn btn-primary" style="background-color: #3455AA; width: 10%;">Join</button>

                    </div>

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

                        <button id="logout" type="button" onclick="logout()" class=" btn btn-primary" style="background-color: #3455AA; width: 30%;">Logout</button>

                    </div>

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

                        <button class="btn btn-primary" id="epic" style="background-color: #3455AA; width: 30%;" type="button" onclick="leaderboard()">Leaderboards</button>

                    </div>

                </div>

            </div>

        </div>

    </body>

    <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>

Leaderboard.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/leaderstyle.css" rel="stylesheet">

        <title>

            Paint + Guess

        </title>

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

    </head>

    <body style="background-color: #DDD;">

        <div class = "container" style="align-content: center;">

            <h1 style="text-align: center; color: #3354AA;" class="display-1">Leaderboard</h1>

        </div>

        <div id="leaderboard" class="container" style="align-content: center; background-color: #C4C4C4; width: 60%;">

            <div id="title" class="row epicclassno1">

                <div id="nametitle" class="title epicclassno2 table" style="font-family: var(--bs-body-font-family);">Name</div>

                <div id="scoretitle" class="title epicclassno2 table">Score</div>

            </div>

        </div>

        <div class="centerdiv">

            <button id="update" type="button" onclick="updateLeaderboard()" style="background-color: #3354AA; color: #FFF; font-family: var(--bs-body-font-family); margin: 10px;" class="btn">Refresh Leaderboard</button>

        </div>

        <div class="centerdiv">

            <button id="update" type="button" onclick="goBack()" style="background-color: #3354AA; color: #FFF; font-family: var(--bs-body-font-family);" class="btn">Back To Home</button>

        </div>

        <script src= "/static/leaderapp.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>

    </body>

</html>

Leaderapp.js

const socket = io.connect('http://' + document.domain + ':' + location.port); //This creates a socket connection with the flask server

socket.on('sendleader', function(scores) { //This function runs whenever the client receives a sendleader event from the server, along with the scores the server sends.

    let leaderboard = document.getElementById("leaderboard"); //This gets the leaderboard div element and puts it into a variable so it's easily accessible

    leaderboard.innerHTML = `<div id="title" class="row epicclassno1">

    <div id="nametitle" class="title epicclassno2 table">Name</div>

    <div id="scoretitle" class="title epicclassno2 table">Score</div>

</div>`; //This html creates a div element which functions as a row in the table, then creates the Name div and Score div inside the row, which both function as titles for the leaderboard columns.

    for(let i=0; i < scores.length; i++) { //This for loop iterates through the received scores until it reaches the last one.

        let name = document.createElement("div"); //This creates a div element that will be used to store the name from the currently iterated row.

        let score = document.createElement("div"); //This creates a div element that will be used to store the score from the currently iterated row.

        name.classList.add("name", "epicclassno2"); //This adds the name and epicclassno2 classes to the div that will hold the username.

        score.classList.add("score", "epicclassno2"); //This adds the score and epicclassno2 classes to the div that will hold the score.

        name.innerText = scores[i][1]; //This takes the username from the currently iterated row in the received scores then puts it into the name div

        score.innerText = scores[i][0]; //This takes the score from the currently iterated row in the received scores then puts it into the score div.

        let scoreRow = document.createElement("div"); //This creates a div that will contain the name div and score div, it will function as a row for the table.

        scoreRow.classList.add("row","epicclassno1"); //This adds the row and epicclassno1 classes to the scoreRow div. This makes it so the row is properly centered

        scoreRow.appendChild(score); //This puts the score div into the scoreRow div

        scoreRow.appendChild(name); //This puts the name div into the scoreRow div

        leaderboard.appendChild(scoreRow); //This puts the scoreRow div into the leaderboard div, which makes it display on the user's screen.

    };

});

function updateLeaderboard(){ //This function is run to request an updated leaderboard from the server.

    socket.emit("requestleader");

};

function goBack(){ //This function takes the user to the index route when they press the leaderboard button on the page

    location.href = "/"

}

updateLeaderboard() //This is called to request an updated leaderboard from the server. Without this the user would manually have to press the update leaderboard button to get a leaderboard.

Spectatestyle.css

body {

    background-color: #DDD;

}

#spectatecanvas {

    display:block;

    width: 1000px;

    height: 680px;

    background-color: white;

}

#chatinput {

    width: 205px;

}

#chat {

    width: 300px;

    height: 600px;

    list-style: none;

    background-color: #C4C4C4;

    text-align: left;

    overflow-wrap: break-word;

    overflow-y: auto;

    margin: 0;

    padding: 5px;

}

#chatsubmit {

    background-color: #C4C4C4;

    color: black;

    width: 90px;

    height: 38px;

}

#title {

    display: block;

    font-size: 23pt;

    font-weight: bolder;

    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

}

Leaderstyle.css

.epicclassno1 {

    flex-wrap: nowrap;

    left: 5px;

    display: flex;

    align-items: center;

    justify-content: center;

    margin: auto;

}

.epicclassno2 {

    width: 30vw;

    display: flex;

    align-items: center;

    justify-content: center;

}

.title {

    font-size: large;

    font-family: Verdana, Geneva, Tahoma, sans-serif;

    font-weight: bold;

}

.centerdiv {

    display: flex;

    align-items: center;

    justify-content: center;

}

.table {

    border-color: black;

    border-width: 1px;

}

Drawstyle.css

#hex {

    width: 120px;

}

#brush {

    width: 120px;

}

body {

    background-color: #DDD;

}

#drawcanvas {

    display:block;

    width: 1000px;

    height: 680px;

    background-color: white;

}

#title {

    display: block;

    font-size: 23pt;

    font-weight: bolder;

    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

}

#chatinput {

    width: 205px;

}

#chat {

    width: 300px;

    height: 600px;

    list-style: none;

    background-color: #C4C4C4;

    text-align: left;

    overflow-wrap: break-word;

    overflow-y: auto;

    margin: 0;

    padding: 5px;

}

#chatsubmit {

    background-color: #C4C4C4;

    color: black;

    width: 90px;

    height: 38px;

}

#word {

    color: black;

}

.colorButton {

    height: 25px;

    width: 20px;

    margin: 4px;

  }

Words.json

{

  "words": [

    "House",

    "Phil Collins",

    "Skateboard",

    "Craig David",

    "Charlie Chaplin",

    "Balloon",

    "Armour",

    "Catholicism",

    "Chihuahua",

    "Christianity",

    "Easter",

    "Frenchman",

    "Pharaoh",

    "Rodeo",

    "Sister",

    "Brother",

    "Phone",

    "Bingo",

    "Peat Bog",

    "Afro",

    "Rotary Engine",

    "Felix Wankle",

    "Nissan Silvia S13 (1989)",

    "Cheese",

    "Fart",

    "Instagram",

    "Thumb",

    "Music",

    "Piano",

    "Rock And Roll",

    "Guitar",

    "Oasis",

    "Bow",

    "Tie",

    "Eyeball",

    "Ice Cream",

    "Prapor",

    "Angle",

    "Angel",

    "Camera",

    "Pizza",

    "Brain",

    "Magnet",

    "Snowflake",

    "Attic",

    "Mr Potato Head",

    "Pirate",

    "Zombie",

    "Stork",

    "Koala",

    "Solar Eclipse",

    "Cinema",

    "Drive Thru",

    "McDonalds",

    "Swimming",

    "Chandelier",

    "Google",

    "Hockey Stick",

    "Wireless Fidelity",

    "Playground",

    "Mechanic",

    "Cruise Ship",

    "Apple",

    "Computer",

    "Television",

    "Eric Andre",

    "Android"

  ]

}

Game Table SQL

CREATE TABLE "game" (

"game\_id" INTEGER NOT NULL,

"game\_code" VARCHAR,

UNIQUE("game\_id"),

PRIMARY KEY("game\_id")

);

User Table SQL

CREATE TABLE "user" (

"id" INTEGER NOT NULL,

"username" VARCHAR(14),

"password" VARCHAR(100),

"score" INTEGER,

PRIMARY KEY("id")

);

Link Table SQL

CREATE TABLE "link" (

"link\_id" INTEGER NOT NULL,

"game\_id" INTEGER,

"user\_id" INTEGER,

"score" INTEGER,

FOREIGN KEY("user\_id") REFERENCES "user"("id"),

FOREIGN KEY("game\_id") REFERENCES "game"("game\_id"),

PRIMARY KEY("link\_id"),

UNIQUE("link\_id")

);

**Testing**

Multiple games were hosted in a group of people representative of the target audience 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” | Added a larger number of words. |
| 07/03/2022 | “Truly excellent, flawlessly implemented” | Thanked the play tester for their time |
| 07/03/2022 | “The drawings have a lot of latency before appearing on the other user’s screens.” | Reduced the time between the drawer’s client sending the canvas to the other players |
| 07/03/2022 | “The colour picker buttons don’t look very good” | Replaced the placeholder buttons with better looking bootstrap buttons and removed the text from them, leaving just the colour. Also arranged them into a grid. |

**Objective Testing**

Graphical user interface, application, Teams

Description automatically generatedTest Objective 1: Users can log in

Graphical user interface, application

Description automatically generated[Figure 12] Entering details into login page

[Figure 13] User logged in successfully  
Objective 1 Completed Successfully

Graphical user interface

Description automatically generatedTest Objective 2: Users can join rooms with a room code

Graphical user interface, application, Word

Description automatically generated[Figure 14] Entering a room code

[Figure 15] Room Successfully joined

Objective 2 Completed Successfully

Test Objective 3: Users can draw

Graphical user interface, application, Teams

Description automatically generated

[Figure 16] Drawing works

Objective 3 Completed Successfully

Test Objective 4: Users can chat

Graphical user interface, application

Description automatically generated

[Figure 17] Chat works as intended

Objective 4 Successfully Completed

Test Objective 5: Users can see the current drawer’s drawing

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application, Teams

Description automatically generated[Figure 18] The drawer’s screen

[Figure 19] The spectator’s screen

Objective 5 Successfully Completed

Graphical user interface, application

Description automatically generatedTest Objective 6: Users can type in chat to guess what the drawer is currently drawing

[Figure 20] Spectator Typing in their guess

Graphical user interface, application

Description automatically generated

[Figure 21] Chat showing that testuser2 has guessed the drawing

Objective 6 Successfully Completed

Test Objective 8: Multiple games can be played in parallel without affecting each other

A picture containing graphical user interface

Description automatically generated

[Figure 22] The drawer in room 1. Shows the drawing and 2 messages from 2 different users in the room.

A picture containing graphical user interface

Description automatically generated

[Figure 23] The spectator’s screen in room 1. The drawing and the messages were both received.

Icon

Description automatically generated

[Figure 24] The drawer’s screen in room 2. The drawing and messages from room 1 were not sent to this room and the messages and drawing from this room were not sent to room 1.

Objective 8 Completed Successfully

Test Objective 7: Leader boards showing user’s scores

Text

Description automatically generated with medium confidence

[Figure 25] Screenshot of the leader board page

Objective 7 Completed Successfully

Test Objective 9 & 10: Save users’ scores to an SQL database and encrypt users’ passwords for security

Text

Description automatically generated

[Figure 26] Screenshot of DB Browser showing the data in the user table. Passwords are encrypted and scores are saved.

Objectives 9 & 10 Completed Successfully

**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. These could be displayed to the user given the information currently stored in the database, however they were not added due to time constraints. Another feature that could be added given more time is chat filtering. This could be either done simply but in a limited way through a list of disallowed words that aren’t allowed in the chat, however depending on the way this is implemented it could prove ineffective as users could find ways around it. A better, though harder to implement way to filter the chat would be through an algorithm that can recognize words even if the user tries to obscure them through putting spaces into the words or other means. It could also be beneficial to add an algorithm that would check the similarity of some words to words that would need to be filtered out, but this could come with the side effect of filtering out words that shouldn’t be filtered.