PICTIONARY

ABSTRACT:

This project aims to develop a multiplayer Pictionary web application using Node.js, Express.js, Socket.IO, MongoDB, HTML, and CSS. The application allows multiple users to connect in real-time and engage in collaborative drawing and guessing sessions reminiscent of the classic Pictionary game. The server, powered by Node.js and Express.js, facilitates communication between clients, while Socket.IO enables real-time bidirectional event-based communication. MongoDB is used to store game data such as player scores and selected words. The frontend is crafted using HTML and CSS to provide an intuitive and interactive user interface for drawing and guessing. Through this project, users can experience the excitement of Pictionary in an online multiplayer environment, fostering social interaction and creativity.

INTRODUCTION:

In the digital age, online multiplayer games have become increasingly popular, providing players with opportunities for social interaction and entertainment. Drawing inspiration from the classic party game Pictionary, this project introduces a web-based multiplayer Pictionary application. Leveraging the power of modern web technologies such as Node.js, Express.js, Socket.IO, MongoDB, HTML, and CSS, this application offers users an immersive and collaborative gaming experience. Players can connect from anywhere in the world, join rooms, and engage in real-time drawing and guessing sessions, just like the traditional tabletop game. The introduction of this project sets the stage for exploring the development process and the integration of various technologies to create an engaging multiplayer gaming platform. Through this project, we aim to demonstrate the potential of web technologies in recreating beloved offline games in a digital, multiplayer format, fostering connections and enjoyment among players worldwide.

Here, basically there is a login window where the players can join the rooms which they want by entering the room number and their name. Then they will be taken to an actual drawing and guessing page. Here, one player will draw at a time. The rest of them will guess the picture using the chat window. If the guess is correct according to the time points will be awarded in the players window. There will be a secret word given to the person who is drawing, they should draw that picture. There will be a time limit of 60s for drawing and guessing per word. After the 60s the controls will be shifted automatically in FCFS order to the next person. Also, who is drawing will be transmitted to all the guessing players. The drawing controls have a change of colour of the stroke, change of canvas colour, erase the drawing, undo the drawing, change in stroke width and finally clear the canvas.

TOOLS AND TECHNOLOGIES USED:

- **Frontend Technologies:**
    - HTML (Hypertext Markup Language):
        - HTML forms the backbone of the Scribble game's frontend development, providing the structure for all content displayed to the user.
        - It defines the layout of various elements such as buttons, forms, and text inputs, allowing developers to organize the user interface effectively.
    - CSS (Cascading Style Sheets):

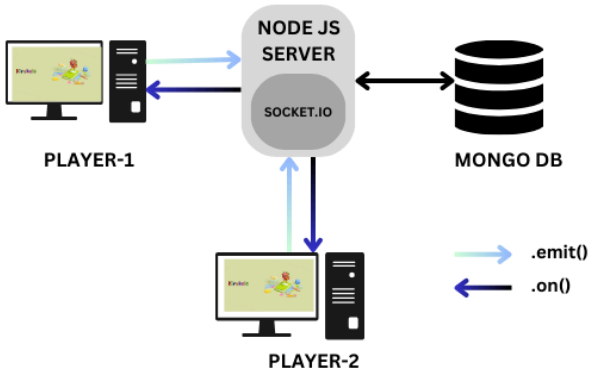
        - CSS plays a crucial role in enhancing the visual appearance of the Scribble game's frontend.
        - By applying styles such as colors, fonts, and layouts, CSS ensures a consistent and appealing design across all pages and components.
        - Additionally, CSS enables responsive design, allowing the game interface to adapt seamlessly to different screen sizes and devices.
    - JavaScript:
        - JavaScript provides the interactivity and dynamic functionality of the Scribble game's frontend.
        - Through event handling, DOM manipulation, and AJAX requests, JavaScript enables features such as real-time updates, user input validation, and interactive elements like buttons and sliders.
- **Backend Technologies**:
    - Node.js:
        - Node.js serves as the backend runtime environment for the Scribble game, allowing developers to write server-side JavaScript code.
        - It provides a non-blocking, event-driven architecture that is well-suited for handling asynchronous operations, such as database queries and network requests.
        - Node.js also offers a rich ecosystem of packages through npm (Node Package Manager), allowing developers to easily integrate third-party libraries and modules into their backend code.
    - Express.js:
        - Express.js is a web application framework for Node.js, providing a robust set of features for building scalable and efficient backend APIs.
        - It simplifies the process of defining routes, handling middleware, and managing HTTP requests and responses.
        - Express.js also offers support for template engines like Handlebars, facilitating server-side rendering of dynamic content.
    - Handlebars (hbs):
        - Handlebars is a templating engine used in the Scribble game for server-side rendering of HTML content.
        - It allows developers to create reusable templates with dynamic data, improving code organization and maintainability.
        - Handlebars templates can include variables, helpers, and partials, enabling developers to generate HTML content based on data fetched from the backend.
- **Database Technology:**
    - MongoDB:
        - MongoDB serves as the database solution for storing player data, including names, scores, and guessing history, in the Scribble game.
        - As a NoSQL document-oriented database, MongoDB offers flexibility in schema design, allowing developers to store and retrieve data in JSON-like documents.

        - Its scalability and performance make it well-suited for handling the dynamic and evolving data requirements of the game, while its support for real-time updates ensures that player data is synchronized across all connected clients.

- **Real-time Communication:**
    - Socket.io:
        - Socket.io facilitates real-time bidirectional communication between clients and the server in the Scribble game, enabling seamless interaction between players in virtual rooms.
        - It provides WebSocket support, allowing for low-latency, full-duplex communication between the client and server, which is essential for real-time multiplayer gaming.
        - Socket.io's event-driven architecture and automatic reconnection mechanisms ensure reliability and efficiency, even in the presence of network disruptions or server downtime.
        - Additionally, Socket.io enables features such as chat functionality, live updates, and notifications, enhancing the overall multiplayer gaming experience.

In conclusion, the Scribble game leverages a comprehensive stack of frontend, backend, database, and real-time communication technologies to deliver an immersive and engaging multiplayer gaming experience. From HTML, CSS, and JavaScript on the frontend to Node.js, Express.js, and MongoDB on the backend, each technology plays a crucial role in ensuring the functionality, performance, and scalability of the game. Combined with Socket.io for real-time communication, these technologies enable seamless interaction between players, dynamic updates, and a rich gaming experience that keeps players coming back for more.

DIAGRAM:



SOURCE CODE:

user.js - User schema for mongodb

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
```

```
  },
  room: { // Add room field to store room number
    type: String,
    required: true
  },
  socketid: {
    type: String,
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});
const User = mongoose.model('User', userSchema);
module.exports = User;
```

socket.js - Containing socket connection

```
const socket = io();
```

index.js - Server file

```
const express = require('express');
const app = express();
const http = require('http').createServer(app);
const io = require('socket.io')(http);
const path = require("path");
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const User = require('./models/user');
const tempelatePath = path.join(__dirname, '../templates')
const publicPath = path.join(__dirname, '../public')
app.set('view engine', 'hbs');
app.set('views', tempelatePath)
app.use(express.static(publicPath))
app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.json());
// MongoDB connection
mongoose.connect('mongodb://localhost:27017/pictionaryUsers', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log('MongoDB connected'))
.catch(err => console.error('MongoDB connection error:', err));
// Timer functionality
let timerInterval;
let seconds = 60; // Start from 60 seconds
let minutes = 0;
let socketId=0;
let flag =0;
let flag1=0;
let flag2=0;
let connectedPlayers = [];
```

```
let drawingHistory = [];
let connectedPlayersMap = {};
let word = "";
// Function to handle drawing data received from clients
function handleDrawing(data) {
  // Push the drawing data into the drawing history
  drawingHistory.push(data);
  // Broadcast the drawing data to all connected clients
  io.emit('drawing', data);
}
// Function to send the drawing history to a newly connected client
function sendDrawingHistory(socket) {
  // Send the drawing history to the newly connected client
  drawingHistory.forEach(data => {
      socket.emit('drawing', data);
  });
}
app.get('/', async (req, res) => {
  try {
    res.render('login');
  } catch (err) {
    console.error('Error fetching data:', err);
    res.status(500).send('Internal Server Error');
  }
});
app.post('/submit-name', (req, res) => {
  const playerName = req.body.playerName;
  connectedPlayers.push(playerName);
  res.redirect('/home'); // Redirect to home page after processing the name
});
app.get('/home', async (req, res) => {
  try {
    res.render('home');
  } catch (err) {
    console.error('Error fetching data:', err);
    res.status(500).send('Internal Server Error');
  }
});
async function addToDB(playerName, socketid) {
  try {
    // Save the player name and room number to MongoDB
    const newUser = new User({ name: playerName, room: 1, socketid: socketid });
    await newUser.save();
  } catch (err) {
    console.error(err);
    res.status(500).send('Error saving user to database');
  }
}
async function deleteFromDB(socketid) {
  try {
    // Save the player name and room number to MongoDB
    const result = await User.findOneAndDelete({ socketid: socketid });
      if (result) {
```

```
        console.log("Document deleted successfully");
      } else {
        console.log("Document not found or could not be deleted");
      }
  } catch (err) {
    console.error(err);
    res.status(500).send('Error saving user to database');
  }
}
io.on('connection', socket => {
    console.log('A user connected');
    console.log(socket.id);
    // Emit updated player list to the newly connected client
    const playerName = connectedPlayers.pop();
    connectedPlayersMap[socket.id] = {playerName: playerName, drawing: false, score:0 };
    io.emit('updatePlayerList', connectedPlayersMap);
    addToDB(playerName,socket.id);
    if(flag2==0)
    {
      socketId = socket.id;
      connectedPlayersMap[socketId].drawing = true;
      io.to(socketId).emit('switchDrawingPlayer');
     // io.emit('switchDrawingPlayer', currentPlayer);
    }
    else
    {
     for (const socketIds in connectedPlayersMap) {
       if (socketId !== socketIds) {
         // Emit the data to the current socket ID
         console.log("socket Id "+socketId);
         console.log("socket Ids "+socketIds);
         io.to(socketIds).emit('drawer', {name: connectedPlayersMap[socketId].playerName});
      }
    }
    }
    if(flag==0)
    {
      startTimer();
    }
    flag2=1;
      // Modify your disconnect event handler
      socket.on('disconnect', () => {
        console.log('A user disconnected');
        console.log(socket.id);
        const playerName = connectedPlayersMap[socket.id];
        if (playerName) {
            if(Object.keys(connectedPlayersMap).length>1 && socket.id == socketId)
            {
              const playerIds = Object.keys(connectedPlayersMap);
              const currentIndex = playerIds.indexOf(socketId);
              const nextIndex = (currentIndex + 1) % playerIds.length;
              const nextSocketId = playerIds[nextIndex];
              currentPlayer = connectedPlayersMap[nextSocketId];
```

```
          connectedPlayersMap[socketId].drawing = false;
          socketId = nextSocketId;
          connectedPlayersMap[socketId].drawing = true;
          seconds = 60;
          io.to(socketId).emit('switchDrawingPlayer');
          console.log(socketId);
          for (const socketId in connectedPlayersMap) {
            if (socketId !== nextSocketId) {
              // Emit the data to the current socket ID
              io.to(socketId).emit('drawer', {name: connectedPlayersMap[nextSocketId].playerName});
          }
          }
          if(flag1==0)
            startTimer();
          console.log(connectedPlayersMap);
        }
        delete connectedPlayersMap[socket.id];
        // Emit updated player list to all clients
        io.emit('updatePlayerList', connectedPlayersMap);
      }
      deleteFromDB(socket.id);
    });
  sendDrawingHistory(socket);
  socket.on('drawing', (data) => {
    handleDrawing(data);
});
    // Listen for clear canvas message from clients
  socket.on('clearCanvas', () => {
    // Broadcast the clear canvas message to all clients
    io.emit('clearCanvas');
    drawingHistory = [];
  });
  // Listen for player name submission
  socket.on('submitName', (playerName) => {
    // Store the player name associated with the socket
    socket.playerName = playerName;
    // Add the player name to the array
    connectedPlayers.push(playerName);
    // Emit updated player list to all clients
    io.emit('updatePlayerList', connectedPlayersMap);
});
socket.on('canvasColour', (data) => {
  io.emit('canvasColour',data);
});
socket.on('guess', (data) => {
  const data1 = connectedPlayersMap[socket.id];
  console.log(data.guess);
  if(data.guess == word){
    io.emit('guess',{guess: data.guess, playerName: data1.playerName,});
    connectedPlayersMap[socket.id].score += (seconds*2);
    io.emit('updatePlayerList', connectedPlayersMap);
  }
  else{
```

```
    io.emit('guess',{guess: data.guess, playerName: data1.playerName});
   }
  });
  socket.on('word', (data) => {
   word = data.word;
   console.log(word);
  });
});
function startTimer() {
  timerInterval = setInterval(() => {
    flag=1;
    flag1=1;
    seconds--;
    if (seconds < 0) {
       if (minutes > 0) {
          minutes--;
          seconds = 59;
       } else {
          clearInterval(timerInterval);
          flag1=0;
          switchToNextPlayer();
          return;
       }
    }
    // Emit timer updates to all connected clients
    io.emit('timer', { minutes, seconds });
  }, 1000);
}
function switchToNextPlayer() {
  if(Object.keys(connectedPlayersMap).length > 1){
   const playerIds = Object.keys(connectedPlayersMap);
   const currentIndex = playerIds.indexOf(socketId);
   const nextIndex = (currentIndex + 1) % playerIds.length;
   const nextSocketId = playerIds[nextIndex];
   currentPlayer = connectedPlayersMap[nextSocketId];
   connectedPlayersMap[socketId].drawing = false;
   io.to(socketId).emit('switch');
  for (const socketId in connectedPlayersMap) {
   if (socketId !== nextSocketId) {
    // Emit the data to the current socket ID
    io.to(socketId).emit('drawer', {name: connectedPlayersMap[nextSocketId].playerName});
  }
  }
   socketId = nextSocketId;
   connectedPlayersMap[socketId].drawing = true;
   io.to(socketId).emit('switchDrawingPlayer');
   //io.emit('switchDrawingPlayer');
   seconds = 60;
   console.log(connectedPlayersMap);
   startTimer(); // Restart the timer
}
  else{
   seconds = 60;
```

```
    startTimer();
  }
}
const PORT = process.env.PORT || 3000;
http.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
});
```

login.hbs - Login Page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
  <script type="module" src="https://unpkg.com/@splinetool/viewer@1.0.66/build/spline-viewer.js"></script>
  <!-- Bootstrap CSS -->
  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <style>
    body {
      margin: 0;
      overflow: hidden;
      position: relative;
    }
    spline-viewer {
      display: block;
      width: 100vw;
      height: 100vh;
    }
    .typing-container {
      position: absolute;
      top: 40%;
      left: 4%;
      transform: translateY(-50%);
      white-space: nowrap;
      font-size: 90px;
      font-family: 'Indie Flower', cursive;
      font-weight: bold;
      color: #006400;
      animation: popUpDown 2s infinite alternate;
    }
    .typing-container span {
      display: inline-block;
    }
    .K {
      color: #800000; /* Dark red */
    }
    .i {
      color: #008000; /* Dark green */
    }
    .r {
      color: #000080; /* Dark blue */
```

```
    }
    .u {
      color: #8B4513; /* Saddle brown */
    }
    .k-2 {
      color: #4B0082; /* Indigo */
    }
    .a {
      color: #800080; /* Purple */
    }
    .l {
      color: #808000; /* Olive */
    }
    @keyframes popUpDown {
      0% {
        transform: translateY(0);
      }
      100% {
        transform: translateY(-20px);
      }
    }
    .button-container {
      position: absolute;
      top: 50%;
      right: 5%;
      transform: translateY(-50%);

    }
    .button-container {
        text-align: center; /* Center the button */
        margin-top: 40px; /* Add margin on top of the button */
        margin-bottom: 50px;
      }
  </style>
</head>
<body>
  <!-- Spline viewer -->
  <spline-viewer url="https://prod.spline.design/b0zyYHmK7tK2Wbl0/scene.splinecode"></spline-viewer>
  <!-- Animated text container -->
  <div class="typing-container">
    <span class="K">K</span><span class="i">i</span><span class="r">r</span><span
class="u">u</span><span class="k-2">k</span><span class="a">a</span><span class="l">l</span><span
class="s">s</span>
  </div>
  <!-- Play button and input container -->
  <div class="button-container">
    <form>
      <div class="input-container">
        <!-- Bootstrap text box -->
        <input type="text" name="playerName" id="playerNameInput" class="form-control" placeholder="Enter your
name">
        <br>
        <input type="text" name="roomNumber" id="roomNumberInput" class="form-control" placeholder="Enter
```

```
room number">
      <br>
    </div>
    <!-- Dark green play button -->
    <div class="button-container1">
      <button type="submit" class="btn btn-primary">Play</button>
    </div>
  </form>
 </div>
 <!-- Bootstrap JS -->
 <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
 <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
 <script src="/socket.io/socket.io.js"></script>
</body>
<script>
 // Assuming you're using jQuery for AJAX
$('form').submit(function(event) {
   event.preventDefault(); // Prevent the default form submission behavior
   // Get the player name from the input field
   const playerName = $('#playerNameInput').val();
   // Send a POST request to submit the player name
   $.post('/submit-name', { playerName: playerName}, function(data, status) {
      console.log('Player name submitted:', playerName);
   });
   // Redirect to home page after processing the name
   window.location.href = '/home';
});
</script>
</html>
```

home.hbs - Home Page

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Pictionary</title>
   <!-- Bootstrap CSS -->
   <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
   <!-- Custom CSS -->
   <link rel="stylesheet" href="home.css">

</head>
<body class="bg-light">
{{!-- <header class="header">
   <div class="container">
      <div class="row justify-content-between"> <!-- Use justify-content-between -->
         <div class="col-md-auto"> <!-- Auto-sizing column -->
            <h4 class="text-center1" id="drawguess">
              Draw the Word
            </h4>
```

```
        </div>
        <div class="col-md-auto text-right"> <!-- Auto-sizing column -->
          <span id="timer">00:00</span>

        </div>
      </div>
    </div>
    <div class="container" id="wordContainer">
    </div>
</header> --}}
<header class="header">
    <div class="container header-container">
      <div class="row justify-content-between">
        <div class="col-md-auto">
          <br>
           <h4 class="text-center1" id="drawguess">
             Draw the Word
           </h4>
        </div>
        <div class="col-md-auto text-right">
          <br>
          <span id="timer">00:00</span>
        </div>
      </div>
    </div>
    <div class="container" id="wordContainer">
    </div>
</header>
    <!-- Main Content -->
    <div class="container-fluid">
      <div class="row">
        <!-- Left Section (Chat History) -->
       <div class="col-md-3 chat-container">
    <h4 class="text-guess">Guess History</h4>
    <div id="chatHistory" class="chat-history">
      <!-- Initially display a GIF here --><center>
      <img src="/teddy_guess.gif" alt="Loading GIF">
      </center>
    </div>
    <!-- Guess input and button -->
    <form>
      <div class="guess-input">
        <input type="text" id="guessInput" placeholder="Enter your guess">
        <button type="submit" id="guessButton"></button>
      </div>
    </form>
</div>
        <!-- Center Section (Canvas) -->
        <div class="col-md-6 canvas-container">
          <canvas id="canvas" width="680" height="550" class="border" style="pointer-events: none;"></canvas>

          <div class="mt-3 text-center" id="tools-container">
            <div class="color-palette">
```

```
                <div class="color-row">
                    <button class="color-button" style="background-color: #000000"
data-color="#000000"></button>
                    <button class="color-button" style="background-color: #636363"
data-color="#636363"></button>
                    <button class="color-button" style="background-color: #ff0000" data-color="#ff0000"></button>
                    <button class="color-button" style="background-color: #ff8000" data-color="#ff8000"></button>
                    <button class="color-button" style="background-color: #ffff00" data-color="#ffff00"></button>
                    <button class="color-button" style="background-color: #008000"
data-color="#008000"></button>
                    <button class="color-button" style="background-color: #0000ff" data-color="#0000ff"></button>
                    <button class="color-button" style="background-color: #4b0082"
data-color="#4b0082"></button>
                    <button class="color-button" style="background-color: #ee82ee" data-color="#ee82ee"></button>
                </div>
                <div class="color-row">
                    <button class="color-button" style="background-color: #ffffff" data-color="#ffffff"></button>
                    <button class="color-button" style="background-color: #b4b4b4"
data-color="#b4b4b4"></button>
                    <button class="color-button" style="background-color: #ffadad" data-color="#ffadad"></button>
                    <button class="color-button" style="background-color: #ffd6a5" data-color="#ffd6a5"></button>
                    <button class="color-button" style="background-color: #fdffb6" data-color="#fdffb6"></button>
                    <button class="color-button" style="background-color: #caffbf" data-color="#caffbf"></button>
                    <button class="color-button" style="background-color: #9bf6ff" data-color="#9bf6ff"></button>
                    <button class="color-button" style="background-color: #bdb2ff" data-color="#bdb2ff"></button>
                    <button class="color-button" style="background-color: #ffc6ff" data-color="#ffc6ff"></button>
                </div>
            </div>
            <label id="fillColorPickerButton"><img src="fill.png" alt="Fill colour" width="20" height="20"
id="fillimg"></label>
            <input type="color" id="fillColorPicker" value="#ffffff" class="canvas-fillcolor">
            <label for = "colorPicker" id="colorPickerButton"><img src="colour.png" alt="Pen Colour"
width="20" height="20" id="colourimg"></label>
            <input type="color" id="colorPicker" value="#000000" class="canvas-brush">
            <label for="lineWidthRange"><img src="width.png" alt="Line range" width="20" height="20"
id="widthimg"></label>
            <input type="range" id="lineWidthRange" min="1" max="10" value="2" class="canvas-pencil">
            <button id="clearCanvas"><img src="clear.png" alt="Clear" width="20" height="20" id="clearimg"
class="canvas-clear"></button>
            <button id="undoStroke"><img src="undo.png" alt="Undo" width="20" height="20"
id="undoimg"></button>
            <button id="erase"><img src="erase.png" alt="Erase" width="20" height="20" id="earseimg"
class="canvas-eraser"></button>
        </div>
    </div>
    <!-- Right Section (Player List) -->
    <div class="col-md-3 player-list-container">
        <h4 class="text-player">Players</h4>
        <ul id="playerList" class="player-list">
            <!-- Player names will be displayed here -->
        </ul>
    </div>
</div>
```

```
    </div>
    <!-- Bootstrap JS and Socket.io -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
    <script src="/socket.io/socket.io.js"></script>
    <!-- Custom JavaScript -->
    <script src="socket.js"></script>
    <script src="home.js"></script>
</body>
</html>
```

home.css - Home Page Style Page

```
body {
    background-color: #d8e4bc;
}
.canvas-container {
    text-align: center;
    background-color: #d8e4bc;
}
.header img {
    margin-right: 10px; /* Adjust margin as needed */
}
/* CSS */
.canvas-eraser {
    /* cursor: url('erase.png'), auto; */
     cursor: url(eraser.png),auto;
}
.canvas-pencil {
    cursor: url(pencil.png), auto;
}
/* Additional CSS for popping animation and font change */
/* Additional CSS for popping animation and font change */
/* Additional CSS for jumping animation and font change */
.text-center1 {
    animation: jump 1s ease-in-out infinite; /* Apply jumping animation */
    font-family: monospace; /* Change font to monospace */
    white-space: nowrap; /* Prevent line breaks */
    font-size: 40px;
}
@keyframes jump {
    0%, 100% { transform: translateX(0); } /* Start and end position */
    50% { transform: translateX(-10px); } /* Adjust the jump distance as needed */
}
#timer {
    font-weight: bold; /* Make the timer bold */
    color: #ff5733; /* Set a different color, you can change it to any attractive color you prefer */
    font-size: 40px;
}
.text-guess {
```

```
    text-align: center; /* Center align the text */
    border-radius: 10px; /* Add rounded corners */
    border: 2px solid #ccc; /* Add border */
    padding: 10px; /* Add padding for spacing */
}
/* Style for the input box */
/* Style for the input box */
/* Style for the input box container */
.guess-input {
    border: 2px solid #8B4513; /* Set the border color to dark brown */
    border-radius: 5px; /* Add rounded corners */
    padding: 8px; /* Add padding for spacing */
    width: 200px; /* Set the width of the input box */
}
/* Style for the input box */
#guessInput {
    background-color: white; /* Set the background color to white */
    border: none; /* Remove the border */
    width: 100%; /* Set the width of the input box */
}
/* Style for the button */
#guessButton {
    background-image: url('/palm.png'); /* Set the background image */
    background-size: contain; /* Ensure the background image covers the entire button */
    background-repeat: no-repeat; /* Prevent the background image from repeating */
    background-position: center; /* Center the background image */
    border: none; /* Remove the border */
    border-radius: 50%; /* Make the button round */
    width: 50px; /* Set the width of the button */
    height: 50px; /* Set the height of the button */
    padding: 0; /* Remove padding */
    cursor: pointer; /* Set cursor to pointer for better UX */
}
/* Style for the container */
/* Style for the guess container */
.guess-container {
    margin-bottom: 10px; /* Add margin for spacing between guess containers */
    padding: 8px; /* Add padding for spacing */
    border-radius: 5px; /* Add rounded corners */
}
Additional CSS for header animation
.header-container {
    animation: bounce 2s ease-in-out infinite alternate; /* Apply bouncing animation */
}
@keyframes bounce {
    0%, 100% { transform: translateY(0); } /* Start and end position */
    50% { transform: translateY(-20px); } /* Midway position: move container up */
}
.canvas-fillcolor {
    cursor: url('brush.png'), auto; /* Replace 'fill_cursor.png' with your actual fill cursor image */
}
.canvas-brush {
    cursor: url('filler.png'), auto; /* Replace 'brush_cursor.png' with your actual brush cursor image */
```

```
}
canvas {
    background-color: #ffffff;
    border: 1px solid #000000;
    /* cursor: crosshair; */
    margin-bottom: 5px;
}
.text-player {
    border-radius: 20px; /* Make the container rounder */
    background-color: #333; /* Background color */
    color: white; /* Text color */
    padding: 10px; /* Padding for inner content */
    text-align: center; /* Center alignment */
    margin-bottom: 10px; /* Add margin for spacing */
}
input[type="color"] {
    margin-left: 5px;
    width: 30px;
}
input[type="range"] {
    margin-left: 5px;
    width: 70px;
}
.input-group-append {
    margin-left: -1px;
}
.input-group-append button {
    border-top-left-radius: 0;
    border-bottom-left-radius: 0;
}
.input-group-append button:not(:last-child) {
    border-top-right-radius: 0;
    border-bottom-right-radius: 0;
}
.input-group-append button:hover {
    background-color: #0069d9;
}
.input-group-append button:active {
    background-color: #0056b3;
}
/* .header {
    background-color: #333;
    color: #fff;
    padding: 10px;
    margin-top: 5px;
} */
.header {
    background-color: #3D2F2F;
    color: #fff;
    padding: 10px;
    margin-top: 5px;

    background-size: cover; /* Adjust background size */
```

```
    background-position: center; /* Adjust background position */
}
.header h4{
    margin-left: 460px;
}
.canvas-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    border: none; /* Remove border from canvas container */
}
.player-list-container {
    border-radius: 10px; /* Rounded border */
    background-color: #333; /* Background color */
    color: white; /* Text color */
    padding: 10px; /* Padding for inner content */
    text-align: center; /* Center alignment */
}
.player-list-container h4 {
    font-weight: bold; /* Bold text */
}
.chat-container,
.player-list-container {
    overflow-y: auto;
    border-right: 1px solid #ccc;
    border-left: 1px solid #ccc;
    background-color: #d8e4bc;
}
.chat-history,
.player-list {
    padding: 10px;
}
.canvas-container canvas {
    border: 2px solid #000;
    margin-top: 20px;
}
.btn {
    margin-top: 10px;
}
label {
    margin-top: 10px;
    margin-left: 20px;
}
.color-palette {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-bottom: 10px;
}
.color-row {
    display: flex;
    justify-content: center;
    margin-bottom: 5px;
```

```
}
.color-button {
    width: 25px;
    height: 25px;
    border: none;
    border-radius: 50%;
    margin: 0 5px;
    cursor: pointer;
}
#wordContainer{
    margin-left: 640px;
    font-weight: bold;
    font-size: larger;
}
.container-fluid h4
{
    margin-top: 15px;
    background-color: #333;
    color: #fff;
}
.guess-input {
    position: fixed;
    bottom: 20px; /* Adjust as needed */
    left: 12%;
    transform: translateX(-50%);
    z-index: 999;
    display: flex;
    align-items: center;
    background-color: #fff; /* Optional: Add background color */
    padding: 10px;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* Optional: Add box shadow */
    width: 330px;
    height: 55px;
}
.guess-input input {
    flex: 1;
    margin-right: 10px;
    padding: 8px;
    border: 1px solid #ccc; /* Optional: Add border */
    border-radius: 5px; /* Optional: Add border radius */
    height: 35px;
}
.guess-input button {
    margin-bottom: 10px;
    flex-shrink: 0; /* Prevent button from shrinking */
    height: 35px;
}
.disabled {
    display: none;
}
#canvas {
    width: 550px; /* Set the width and height to match the smaller dimension */
```

```
    height: 550px;
    border-radius: 50%; /* Set border-radius to 50% to create a circle */
    overflow: hidden; /* Ensure content inside the circle is clipped */
}
```

home.js - Home Page Driver JavaScript file

```
const socket = io(); // Initialize Socket.IO
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');
let isDrawing = false;
let lastX = 0;
let lastY = 0;
let strokeColor = '#000000';
let lineWidth = 2;
let strokes = []; // Array to store drawn strokes
let word ="";
// Listen for timer updates from the server
socket.on('timer', ({ minutes, seconds }) => {
    const formattedTime = padNumber(minutes) + ":" + padNumber(seconds);
    document.getElementById("timer").innerText = formattedTime;
});
// Function to pad numbers with leading zeros
function padNumber(num) {
    return (num < 10 ? "0" : "") + num;
}
// Chat history functionality
function addMessageToChatHistory(message) {
    const chatHistoryElement = document.getElementById("chatHistory");
    const messageElement = document.createElement("div");
    messageElement.textContent = message;
    chatHistoryElement.appendChild(messageElement);
}
canvas.addEventListener('mousedown', startDrawing);
canvas.addEventListener('mouseup', stopDrawing);
canvas.addEventListener('mousemove', draw);
document.getElementById('colorPicker').addEventListener('change', changeColor);
document.getElementById('lineWidthRange').addEventListener('change', changeLineWidth);
document.getElementById('clearCanvas').addEventListener('click', clearCanvas);
document.getElementById('undoStroke').addEventListener('click', undoStroke);
document.getElementById('erase').addEventListener('click', erase);
document.getElementById('fillColorPicker').addEventListener('change', changeCanvasColor);
function startDrawing(e) {
    isDrawing = true;
    [lastX, lastY] = [e.offsetX, e.offsetY];
}
function stopDrawing() {
    isDrawing = false;
    ctx.beginPath();
    // Store drawn stroke
    const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
    strokes.push(ctx.getImageData(0, 0, canvas.width, canvas.height));
}
```

```
function draw(e) {
    if (!isDrawing) return;
    socket.emit('drawing', {
        prevX: lastX,
        prevY: lastY,
        currentX: e.offsetX,
        currentY: e.offsetY,
        color: strokeColor,
        width: lineWidth
    });
    drawOnCanvas({ prevX: lastX, prevY: lastY, currentX: e.offsetX, currentY: e.offsetY, color: strokeColor, width:
lineWidth });
    [lastX, lastY] = [e.offsetX, e.offsetY];
}
function changeColor(e) {
    strokeColor = e.target.value;
}
function changeLineWidth(e) {
    lineWidth = e.target.value;
}
function undoStroke() {
    if (strokes.length > 0) {
        strokes.pop(); // Remove the last drawn stroke
        redrawCanvas();

        // Reset lastX and lastY to the end of the previous stroke
        if (strokes.length > 0) {
            const lastStroke = strokes[strokes.length - 1];
            const lastPointIndex = lastStroke.data.length - 4; // Last point index (x, y)
            lastX = lastStroke.data[lastPointIndex];
            lastY = lastStroke.data[lastPointIndex + 1];
        } else {
            lastX = 0;
            lastY = 0;
        }
    }
}
function redrawCanvas() {
    ctx.clearRect(0, 0, canvas.width, canvas.height); // Clear the canvas first
    console.log(strokes);
    strokes.forEach(stroke => {
        ctx.putImageData(stroke, 0, 0); // Redraw all remaining strokes
    });
}
function erase() {
    strokeColor = document.getElementById('fillColorPicker').value;
}
function changeCanvasColor(e) {
    canvas.style.backgroundColor = e.target.value;
    socket.emit('canvasColour',{colour: e.target.value});
}
socket.on('canvasColour', (data) => {
    // Call function to draw received data on the canvas
```

```
    canvas.style.backgroundColor = data.colour;
});
socket.on('drawing', (data) => {
    // Call function to draw received data on the canvas
    drawOnCanvas(data);
});
// Function to draw on canvas
function drawOnCanvas(data) {
    ctx.strokeStyle = data.color;
    ctx.lineWidth = data.width;
    ctx.lineCap = 'round';
    ctx.beginPath();
    ctx.moveTo(data.prevX, data.prevY);
    ctx.lineTo(data.currentX, data.currentY);
    ctx.stroke();
    ctx.closePath();
}
// Get color buttons
const colorButtons = document.querySelectorAll('.color-button');
// Set stroke color when a color button is clicked
colorButtons.forEach(button => {
    button.addEventListener('click', () => {
        const color = button.dataset.color;
        changeColor({ target: { value: color } }); // Call changeColor function with selected color
    });
});
// Define your set of words
const words = ["apple", "banana", "carrot", "dragon", "elephant", "fish"];
// Function to generate a random word from the set
function generateRandomWord() {
    const randomIndex = Math.floor(Math.random() * words.length);
    return words[randomIndex];
}
// Function to display a single random word in the container
function displayRandomWord() {
    const container1 = document.getElementById("drawguess");
    container1.innerHTML = "Draw the Word";

    const container = document.getElementById("wordContainer");
    container.innerHTML = ""; // Clear previous content
    // Generate and display a random word
    word = generateRandomWord();
    const wordElement = document.createElement("p");
    wordElement.textContent = word;
    container.appendChild(wordElement);
}
document.getElementById('clearCanvas').addEventListener('click', () => {
    // Emit a message to the server to clear the canvas
    socket.emit('clearCanvas');
});
// Listen for clear canvas message from the server
socket.on('clearCanvas', () => {
    clearCanvas();
```

```
});
function clearCanvas() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    strokes = [];
    canvas.style.backgroundColor = '#ffffff';
    strokeColor = '#000000';
}
// // Function to update the player list UI
// function updatePlayerListUI(players) {
//     const playerListElement = document.getElementById("playerList");
//     // Clear previous player list
//     playerListElement.innerHTML = "";
//     // Define an array of background colors
//     const colors = ["#FFA07A", "#ADD8E6", "#98FB98", "#FFB6C1", "#D8BFD8", "#87CEEB"];
//     // Add each player to the player list
//     players.forEach((player, index) => {
//         const playerItem = document.createElement("li");
//         playerItem.textContent = `${player.playerName} - Score: ${player.score}`;
//         playerItem.style.backgroundColor = colors[index % colors.length]; // Assign color from the array based on
index
//         playerItem.style.padding = "8px"; // Add padding for spacing
//         playerItem.style.borderRadius = "10px"; // Add rounded corners
//         playerItem.style.listStyleType = "none"; // Remove the list symbol
//         playerItem.style.marginBottom = "5px"; // Add margin bottom for spacing between items
//         playerListElement.appendChild(playerItem);
//     });
// }
// Function to update the player list UI
// function updatePlayerListUI(players) {
//     const playerListElement = document.getElementById("playerList");
//     // Clear previous player list
//     playerListElement.innerHTML = "";
//     // Define an array of background colors
//     const colors = ["#FFA07A", "#ADD8E6", "#98FB98", "#FFB6C1", "#D8BFD8", "#87CEEB"];
//     // Add each player to the player list
//     players.forEach((player, index) => {
//         const playerItem = document.createElement("li");
//         playerItem.textContent = `${player.playerName} - Score: ${player.score}`;
//         playerItem.style.backgroundColor = colors[index % colors.length]; // Assign color from the array based on
index
//         playerItem.style.padding = "8px"; // Add padding for spacing
//         playerItem.style.borderRadius = "10px"; // Add rounded corners
//         playerItem.style.listStyleType = "none"; // Remove the list symbol
//         playerItem.style.marginBottom = "5px"; // Add margin bottom for spacing between items
//         playerListElement.appendChild(playerItem);
//     });
//     // Add GIF for 2 to 3 seconds
//     const gifContainer = document.createElement("div");
//     gifContainer.innerHTML = `<img src="/your-gif.gif" alt="Your GIF" style="width: 100%; height: auto;">`;
//     playerListElement.appendChild(gifContainer);
//     // Remove the GIF after 2 to 3 seconds
//     setTimeout(() => {
//         gifContainer.remove();
```

```
//     }, Math.random() * 1000 + 2000); // Random timeout between 2 to 3 seconds
// }
// Function to update the player list UI
function updatePlayerListUI(players) {
    const playerListElement = document.getElementById("playerList");
    // Clear previous player list
    playerListElement.innerHTML = "";
    // Define an array of background colors
    const colors = ["#FFA07A", "#ADD8E6", "#98FB98", "#FFB6C1", "#D8BFD8", "#87CEEB"];
    // Add each player to the player list
    players.forEach((player, index) => {
        const playerItem = document.createElement("li");
        playerItem.textContent = `${player.playerName} - Score: ${player.score}`;
        playerItem.style.backgroundColor = colors[index % colors.length]; // Assign color from the array based on index
        playerItem.style.padding = "8px"; // Add padding for spacing
        playerItem.style.borderRadius = "10px"; // Add rounded corners
        playerItem.style.listStyleType = "none"; // Remove the list symbol
        playerItem.style.marginBottom = "5px"; // Add margin bottom for spacing between items
        playerListElement.appendChild(playerItem);
    });
    // Conditionally add GIF for 2 to 3 seconds
    if (Math.random() < 0.5) { // Example condition (adjust as needed)
        const gifContainer = document.createElement("div");
        gifContainer.style.textAlign = "center"; // Center the content horizontally
        gifContainer.style.margin = "auto"; // Center the container vertically
        gifContainer.style.marginTop = "20px"; // Add some top margin for spacing
        gifContainer.innerHTML = `<img src="/happy_teddy.gif" alt="Your GIF" style="width: 60%; height: auto;">`;
        playerListElement.appendChild(gifContainer);
        // Remove the GIF after 2 to 3 seconds
        setTimeout(() => {
            gifContainer.remove();
        }, Math.random() * 1000 + 2000); // Random timeout between 2 to 3 seconds
    }
}
// Listen for updated player list from the server
socket.on('updatePlayerList', (players) => {
    // Update the UI to display the updated player list
    const playersArray = Object.values(players);
    updatePlayerListUI(playersArray);
});
// Function to emit drawing data to the server
function emitDrawingData(data) {
    // Include the playerName along with drawing data
    socket.emit('drawing', { ...data, playerName });
    // Handle other drawing-related tasks if necessary
}
socket.on('connect', function() {
    console.log(socket.id);
 });
// $('#guessButton').click(function(event) {
//     event.preventDefault(); // Prevent the default form submission behavior
//     $('#chatHistory').empty();
//     // Get the guess from the input field
```

```
//     const guess = $('#guessInput').val();
//     socket.emit('guess', { guess: guess });
//     // Clear the input field after submitting the guess
//     $('#guessInput').val('');
// });
// // Listen for updated player list from the server
// socket.on('guess', (data) => {
//     // Create a container for the guess word
//     const guessContainer = document.createElement("div");
//     guessContainer.classList.add("guess-container");
//     // Set background color and append the guess word
//     guessContainer.style.backgroundColor = "#ffa07a"; // Set your desired background color
//     guessContainer.textContent = data.playerName + "-" + data.guess;
//     // Append the guess container to the chat history
//     const chatHistoryElement = document.getElementById("chatHistory");
//     chatHistoryElement.appendChild(guessContainer);
// });
$('#guessButton').click(function(event) {
    event.preventDefault(); // Prevent the default form submission behavior
    $('#chatHistory').empty();
    // Get the guess from the input field
    const guess = $('#guessInput').val();
    socket.emit('guess', { guess: guess });
    // Clear the input field after submitting the guess
    $('#guessInput').val('');
});
// Listen for updated player list from the server
socket.on('guess', (data) => {
    // Define an array of background colors
    const colors = ["#FFDAB9", "#BC8F8F"]; // Light gray and light brown
    // Create a container for the guess word
    const guessContainer = document.createElement("div");
    guessContainer.classList.add("guess-container");
    // Set background color based on the index of the guess in the array
    const colorIndex = document.querySelectorAll('.guess-container').length % colors.length;
    guessContainer.style.backgroundColor = colors[colorIndex];
    // Append the guess word to the container
    guessContainer.textContent = data.playerName + ":" + data.guess;
    // Append the guess container to the chat history
    const chatHistoryElement = document.getElementById("chatHistory");
    chatHistoryElement.appendChild(guessContainer);
});
document.querySelectorAll('.color-button, input[type="color"], input[type="range"], #clearCanvas, #undoStroke,
#erase, #clearimg, #undoimg, #earseimg, #fillimg, #colourimg, #widthimg').forEach(element => {
    element.classList.add('disabled');
    document.getElementById('canvas').style.pointerEvents = 'none';
    const container1 = document.getElementById("drawguess");
    container1.innerHTML = "Guess the Word";
});
// Enable all color buttons, input fields, and buttons when receiving a "switchDrawingPlayer" socket notification
socket.on('switchDrawingPlayer', () => {
    document.querySelectorAll('.color-button, input[type="color"], input[type="range"], #clearCanvas, #undoStroke,
#erase, #clearimg, #undoimg, #earseimg,#fillimg, #colourimg, #widthimg').forEach(element => {
```

```
        element.classList.remove('disabled');
    });
    document.getElementById('canvas').style.pointerEvents = 'auto';
    displayRandomWord();
    console.log(word);
    socket.emit('word',{word: word});
});
// Enable all color buttons, input fields, and buttons when receiving a "switchDrawingPlayer" socket notification
socket.on('switch', () => {
    document.querySelectorAll('.color-button, input[type="color"], input[type="range"], #clearCanvas, #undoStroke,
#erase, #clearimg, #undoimg, #earseimg,#fillimg, #colourimg, #widthimg').forEach(element => {
        element.classList.add('disabled');
    });
    const container1 = document.getElementById("drawguess");
    container1.innerHTML = "Guess the Word";
    document.getElementById('canvas').style.pointerEvents = 'none';
});
// Enable all color buttons, input fields, and buttons when receiving a "switchDrawingPlayer" socket notification
socket.on('drawer', (data) => {
    console.log(data);
    const container = document.getElementById("wordContainer");
    container.innerHTML = ""; // Clear previous content
    const wordElement = document.createElement("p");
    wordElement.textContent = data.name + " is drawing";
    container.appendChild(wordElement);
});
document.getElementById('erase').addEventListener('click', () => toggleCursor('eraser'));
document.getElementById('lineWidthRange').addEventListener('click', () => toggleCursor('pencil'));
document.getElementById('fillColorPicker').addEventListener('click', () => toggleCursor('fill'));
document.getElementById('colorPicker').addEventListener('click', () => toggleCursor('brush'));
function toggleCursor(cursorStyle) {
    // Remove all cursor classes from the canvas container
    const canvasContainer = document.querySelector('.canvas-container');
    canvasContainer.classList.remove('canvas-pencil', 'canvas-eraser', 'canvas-fillcolor', 'canvas-brush');
    // Add the appropriate cursor class based on the cursorStyle parameter
    if (cursorStyle === 'pencil') {
        canvasContainer.classList.add('canvas-pencil');
    } else if (cursorStyle === 'eraser') {
        canvasContainer.classList.add('canvas-eraser');
    } else if (cursorStyle === 'fill') {
        canvasContainer.classList.add('canvas-fillcolor');
    } else if (cursorStyle === 'brush') {
        canvasContainer.classList.add('canvas-brush');
    }
}
```

HOW THE PROJECT IS USEFUL FOR THE SOCIETY?

For Children:

- ***Educational Benefits:*** Enhances vocabulary, spelling, and language skills through word recognition and

interpretation.

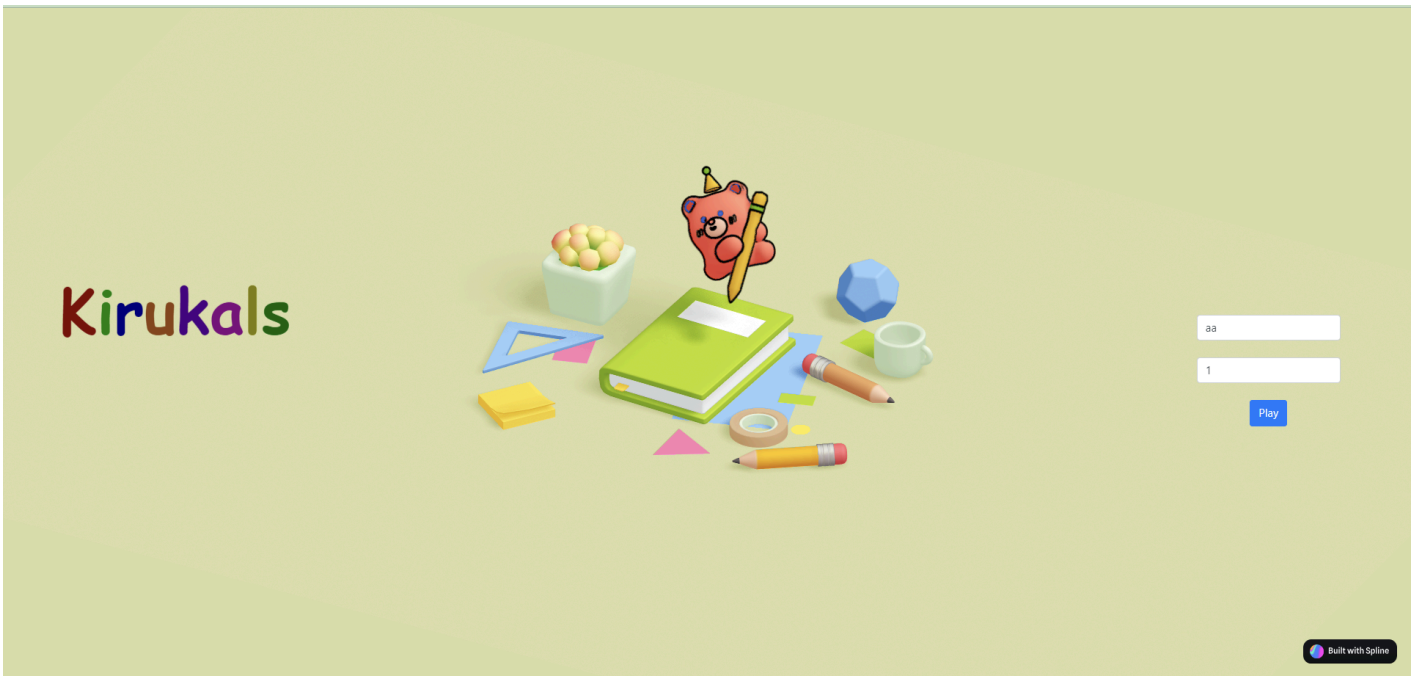
- ***Cognitive Development:*** Stimulates critical thinking, problem-solving, and spatial awareness.
- ***Social Skills:*** Promotes social interaction, teamwork, communication, and cooperation.
- ***Creativity Enhancement:*** Provides a creative outlet for expression through drawing and imagination.
- ***Entertainment and Engagement:*** Offers enjoyable and interactive gameplay that keeps children entertained while also learning.

For Adults:

- ***Social Interaction:*** Provides a platform for connecting with friends, family, or strangers in a fun and engaging way.
- ***Stress Relief:*** Offers an opportunity to unwind, relax, and have fun, providing stress relief and promoting mental well-being.
- ***Community Building:*** Facilitates the formation of online communities around shared interests in gaming and creativity.
- ***Creativity and Expression:*** Allows adults to unleash their creativity through drawing and guessing, providing an outlet for self-expression.
- ***Accessible Entertainment:*** Makes a classic game accessible to adults regardless of geographical location, fostering inclusivity and accessibility.

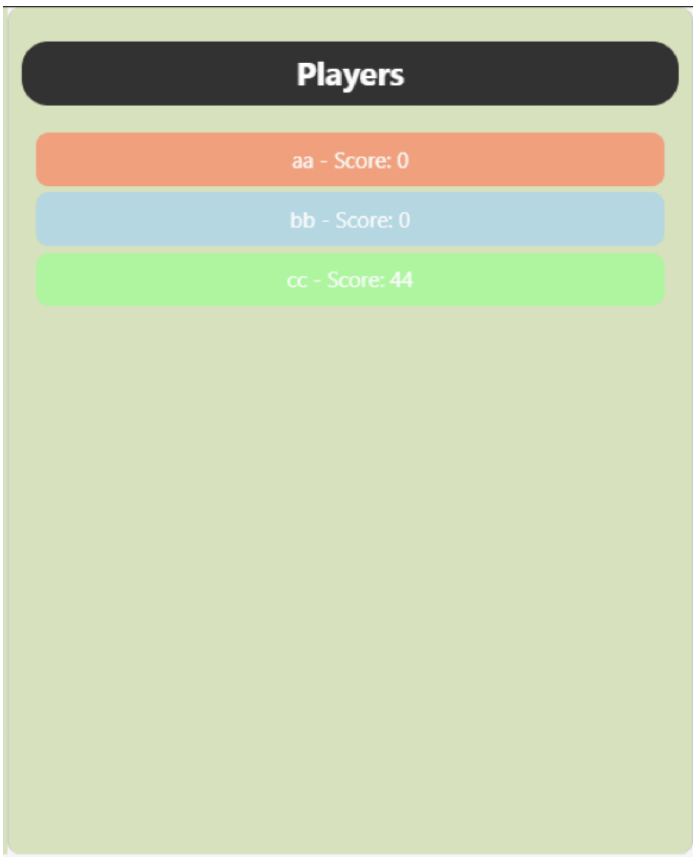
Overall, this project offers both children and adults a range of benefits, including educational enrichment, cognitive stimulation, social interaction, creativity enhancement, and accessible entertainment.

OUTPUT:

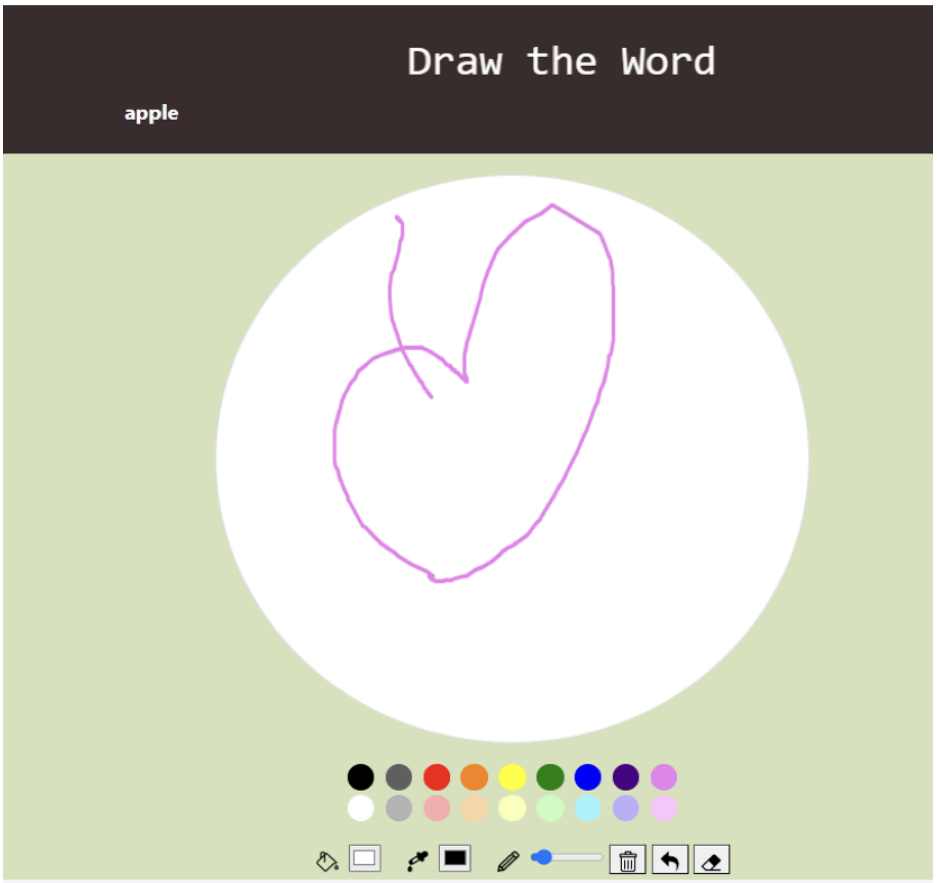


Login page

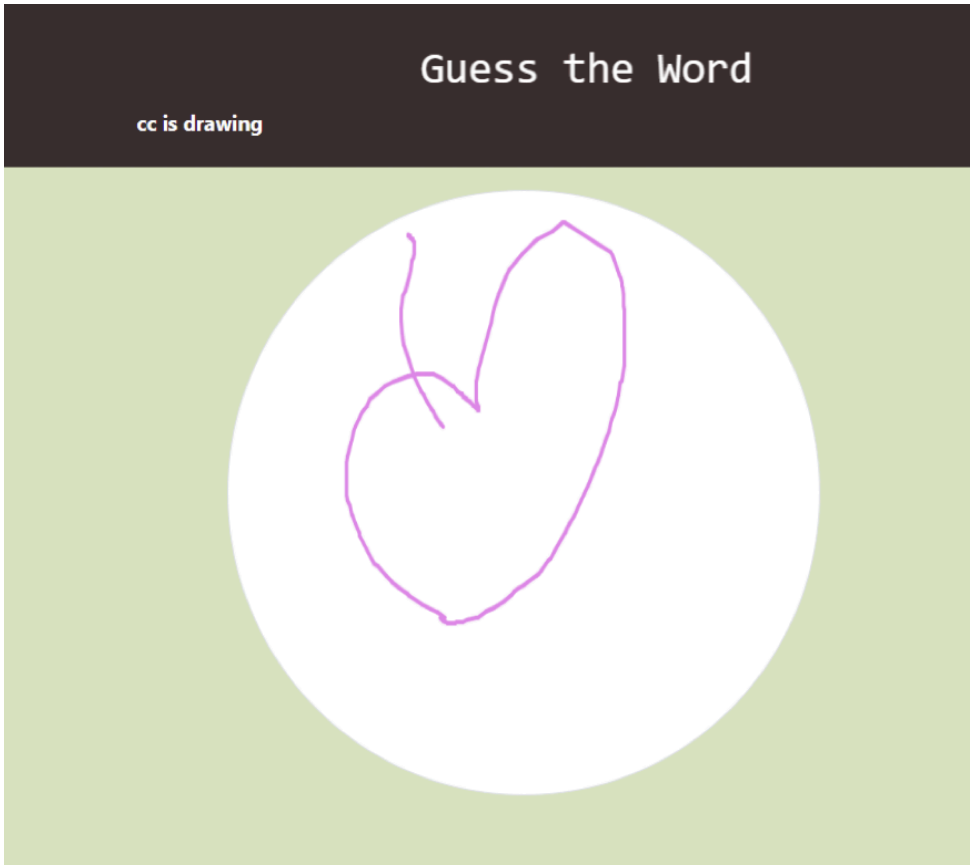
Guess History



Players with their scores

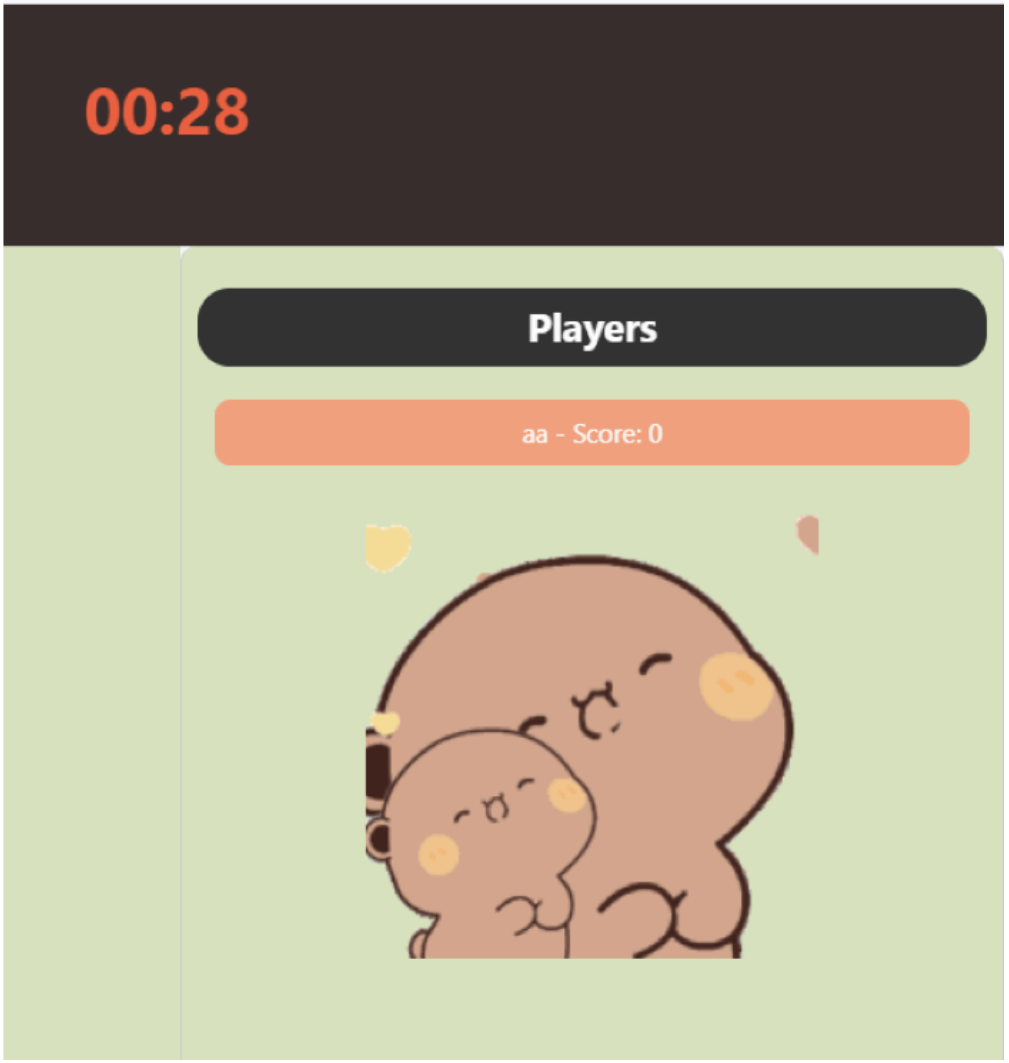


Drawing window with the secret word, canvas and drawing tools

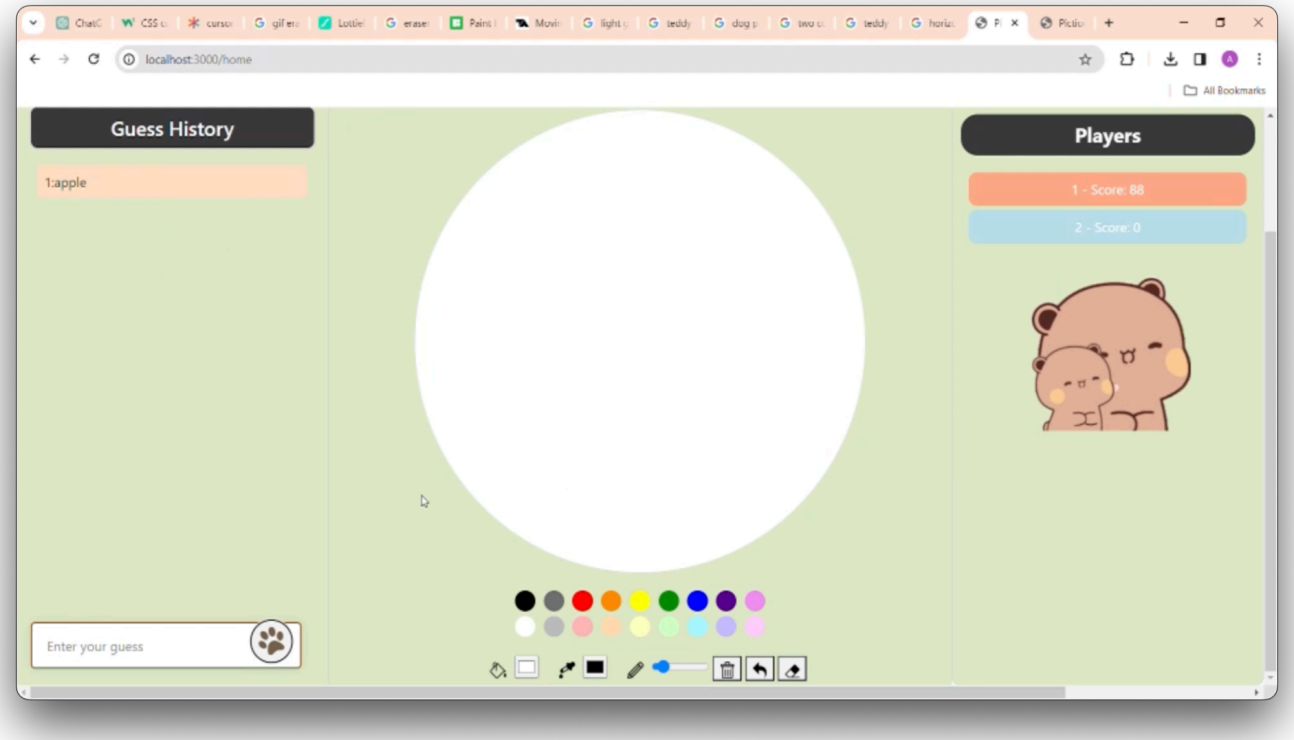


Guessing window with the canvas and the name of player who is drawing

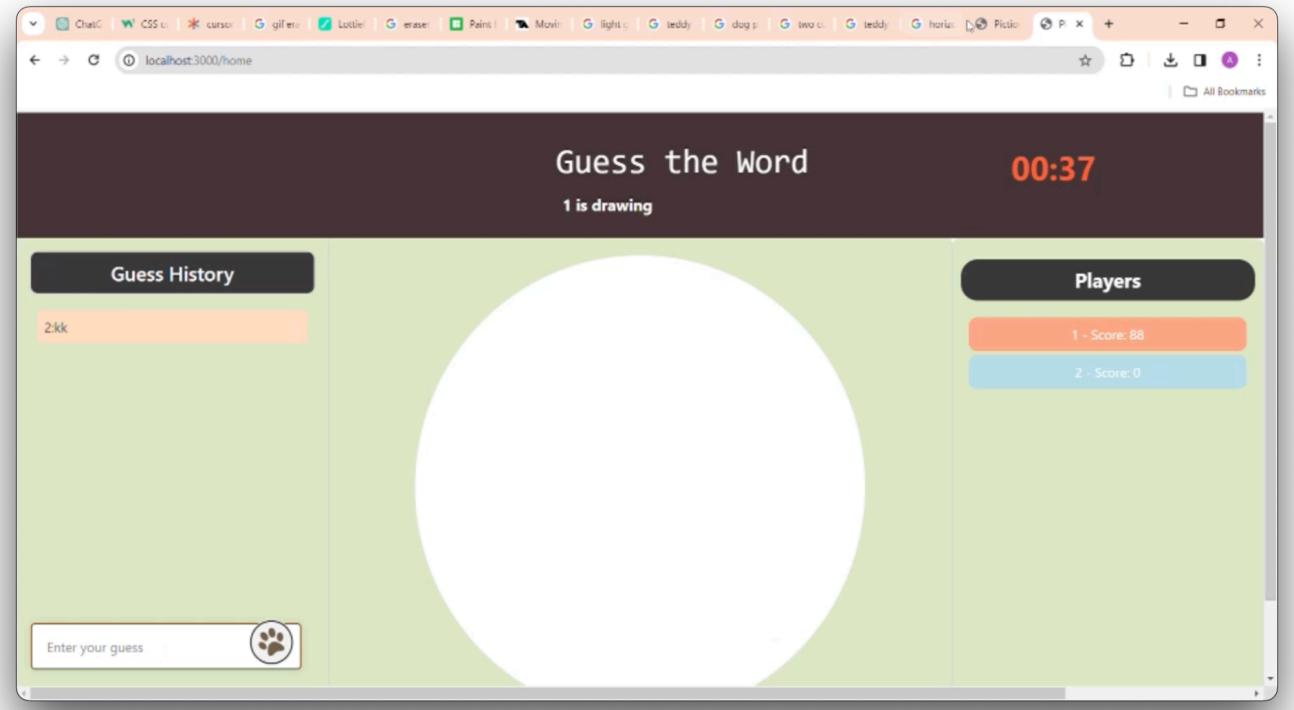
Guess history chat window



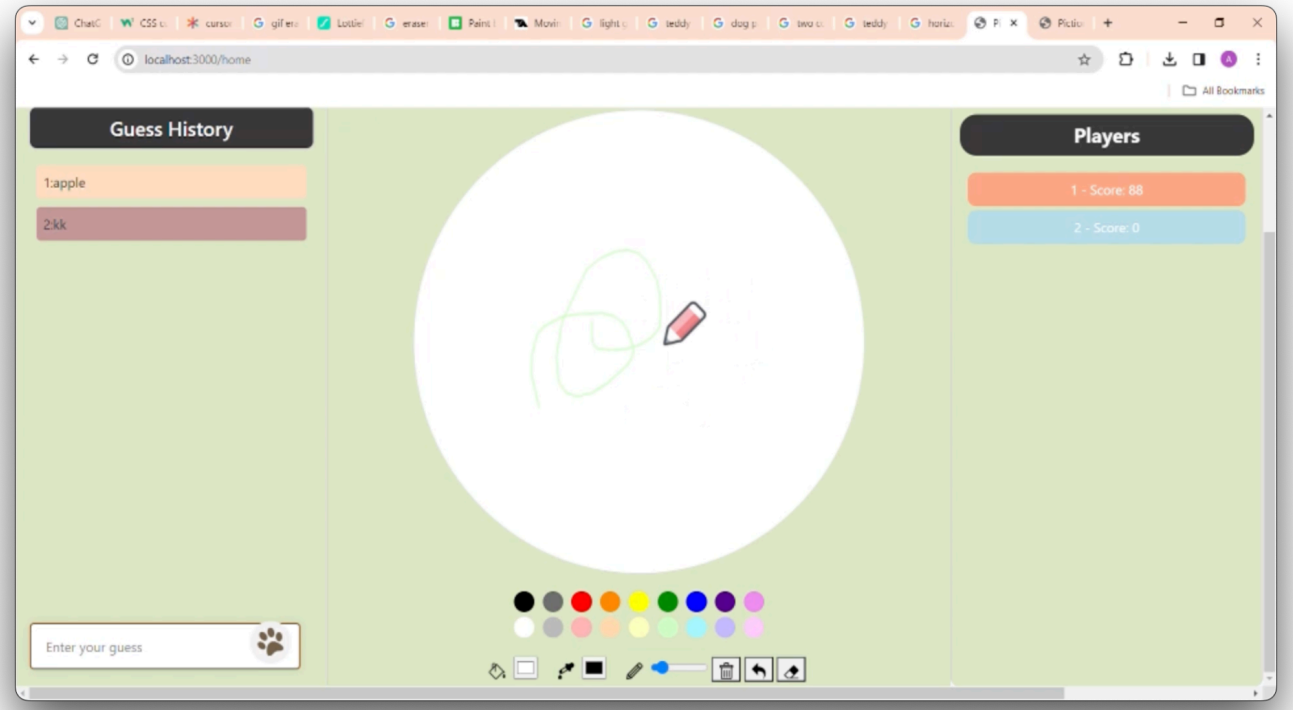
Running time with limit of 60s



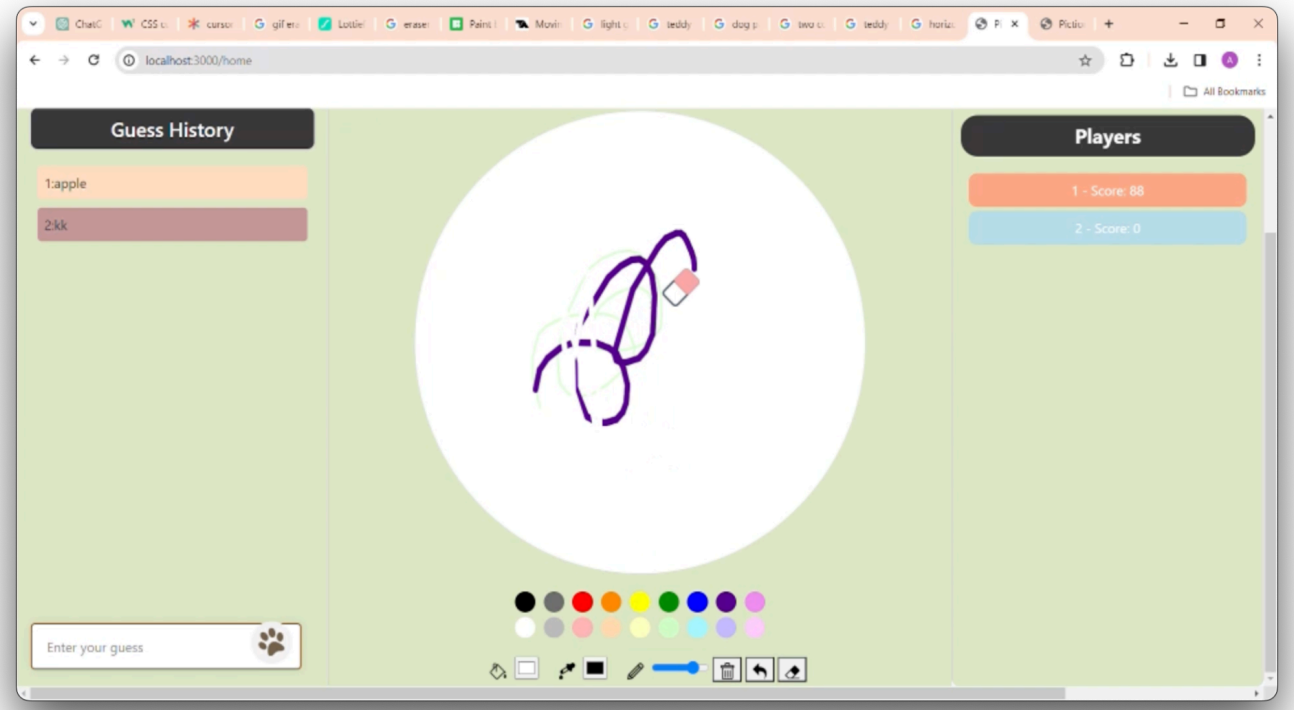
Drawing page with drawing controls



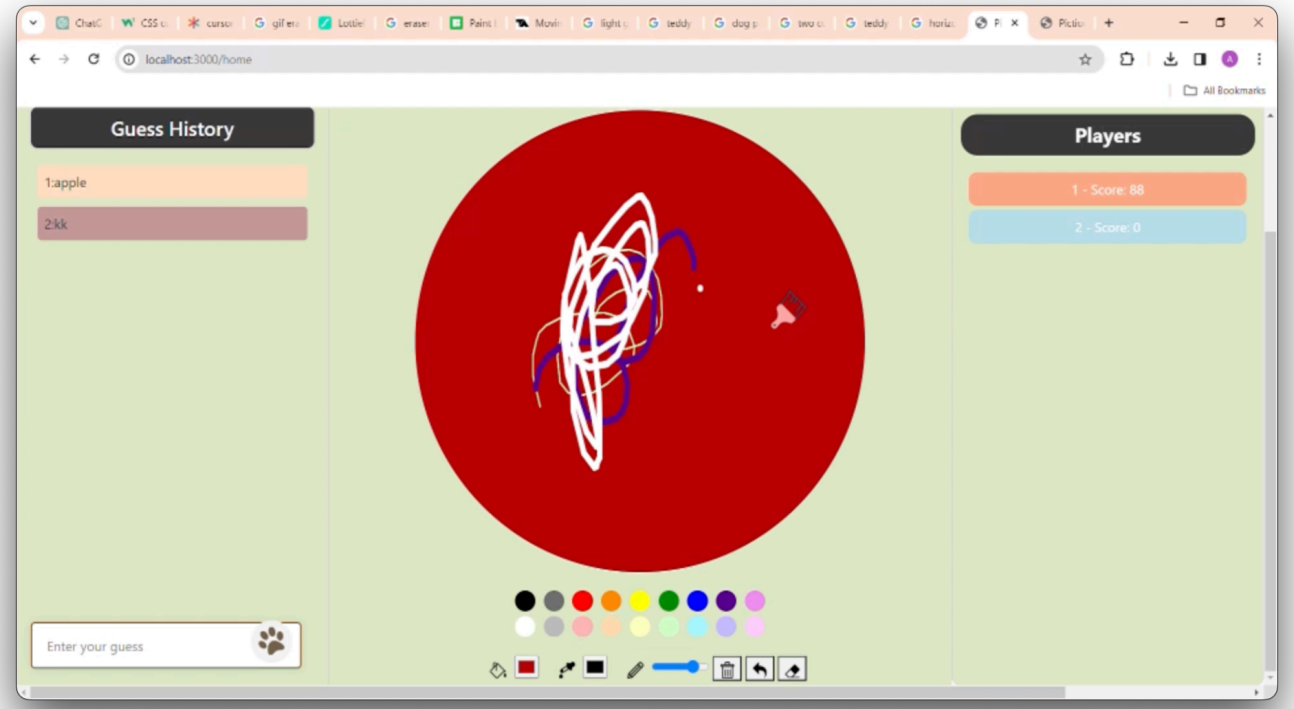
Guessing page with guessing chat and player points



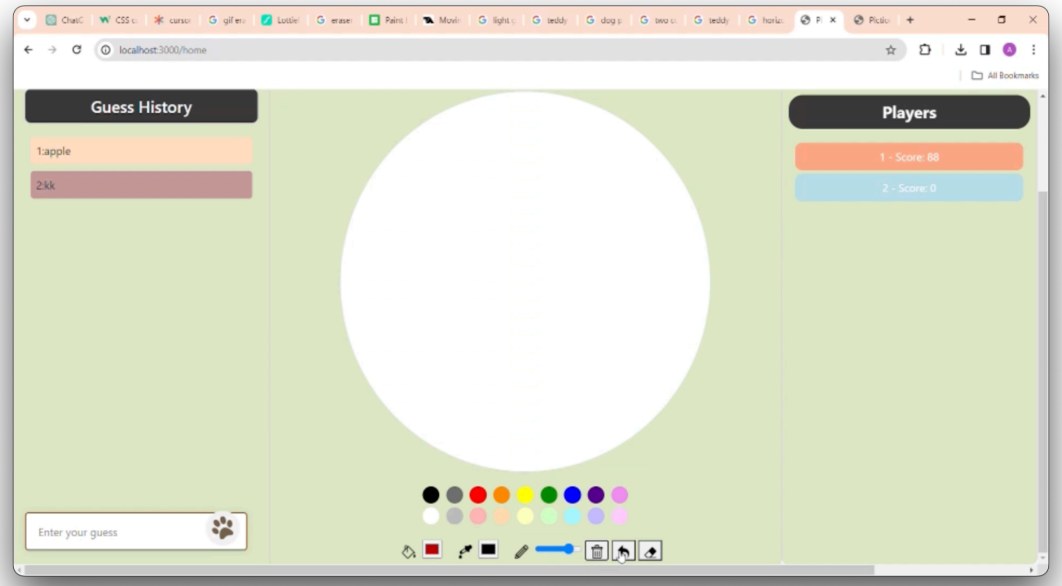
Drawing with pen



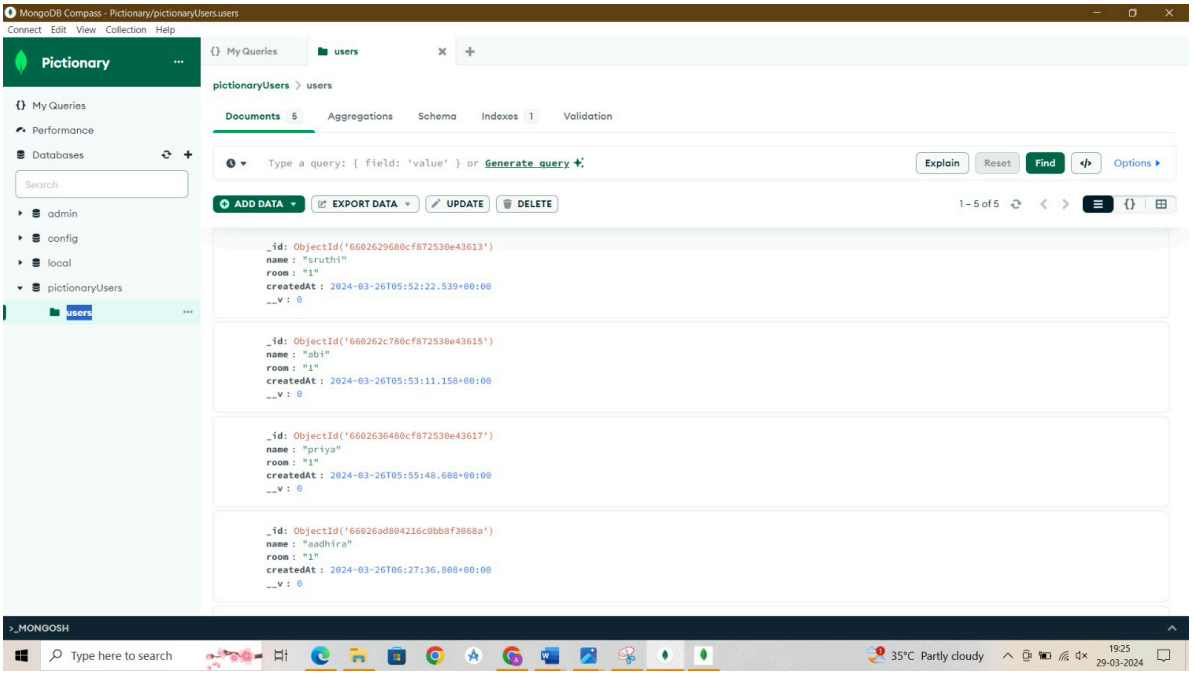
Erasing the drawing



Fill colour in the canvas board



Clearing the canvas board



Mongodb pictionaryUsers database

CONCLUSION:

In summary, the multiplayer Pictionary web application developed using Node.js, Express.js, Socket.IO, MongoDB, HTML, and CSS offers a versatile and engaging platform for both children and adults. By combining educational, cognitive, social, and creative benefits, the project promotes learning, communication, collaboration, and leisure in an accessible online environment. Through its utilization of modern web technologies, the project exemplifies the potential of digital innovations to enhance traditional games and foster social interaction and community building. Overall, the multiplayer Pictionary web application provides an enjoyable and enriching experience for users of all ages, contributing positively to their entertainment, learning, and social well-being.