WordPulse Documentation

Project

WordPulse is a single page web application that allows users to create rooms and captivating word clouds quickly and easily. Users can join rooms and contribute to the word cloud in real-time. The application is built using Express, HTML, TailwindCSS, JavaScript, and WebSocket.

Main Objective

The main objective of WordPulse is to provide an engaging and interactive platform for users to generate and visualize word clouds dynamically. This tool is particularly useful for educational settings, meetings, and events where real-time feedback and interaction are valuable.

Functionalities

1. **Create Rooms**: Users can create a new room by providing a topic, which serves as the basis for the word cloud.
2. **Join Rooms**: Users can join existing rooms using a unique room ID.
3. **Contribute to Word Clouds**: Users can submit words related to the topic, which are then added to the word cloud in real-time.
4. **Real-time Updates**: The word cloud updates dynamically as new words are contributed.
5. **Profanity Filter**: Words submitted by users are filtered to prevent offensive content using ProfanityAPI.
6. **Random Topic Generator**: Users can generate random topics using WikipediaAPI.
7. **Download Word Clouds**: Users can download the word cloud as an image file.

Features

- **Responsive Design**: The application is designed to work seamlessly across different devices and screen sizes.
- **Real-time Interaction**: Uses WebSocket for real-time communication and updates.
- **Visual Appeal**: The word cloud is generated using Chart.js with a custom plugin for a visually appealing representation.
- **Ease of Use**: The interface is intuitive and user-friendly, making it accessible for users of all ages.

Program Structure

Main Directory

- **index.js**: The main server file that sets up and runs the Express server, handling routing and WebSocket connections.
- **package.json**: Contains metadata about the project and dependencies.
- **.gitignore**: Specifies files and directories to be ignored by Git.
- **.prettierrc**: Configuration file for Prettier, a code formatting tool.
- **package-lock.json**: Automatically generated file that locks the versions of the project's dependencies.
- **README.md**: Provides an overview of the project, installation instructions, and other relevant information.
- **node_modules**: Directory containing all the installed npm packages.

Public Directory

- **app.js**: Contains the client-side JavaScript code for handling user interactions and WebSocket communication.
- **index.html**: The main HTML file that defines the structure and content of the web application.
- **fonts**: Directory for custom fonts used in the application.
    - **DMSans-VariableFont_opsz,wght.woff2**: Custom font file for DM Sans used in the application.

Detailed File Descriptions

index.js

This is the entry point of the application. It sets up the Express server, serves static files, and handles WebSocket connections.

Key Functions

- **Express Setup**: Configures the Express server to serve static files from the `public` directory.
- **WebSocket Setup**: Initializes WebSocket to handle real-time communication between the server and clients.
- **Room Management**: Manages the creation and joining of rooms, as well as the distribution of words to the word cloud.

app.js

This file contains the client-side logic for the WordPulse application.

Key Functions

- **Creating and Joining Rooms**: Handles user actions for creating new rooms or joining existing ones.
- **Word Submission**: Manages the submission of words by users and sends them to the server.
- **Real-time Updates**: Listens for updates from the server to dynamically update the word cloud.
- **Profanity Filtering**: Ensures submitted words are clean and appropriate.

index.html

Defines the HTML structure of the application.

Key Sections

- **Head**: Contains metadata, links to stylesheets, and scripts.
- **Body**: Contains the main content, including the interface for creating/joining rooms and displaying the word cloud.

Fonts

Includes custom fonts used in the application to enhance the visual appeal.

DMSans-VariableFont_opsz,wght.woff2

A custom font used throughout the application for a consistent and modern look.

Code Documentation

Server-side Javascript Code (index.js)

This JavaScript file sets up a basic web server using Express and integrates WebSocket functionality to handle real-time communication. The server supports creating and joining rooms, and adding words to a room's word list.

Each room can have multiple clients, and messages are broadcasted to all clients in the room.

Dependencies

- **express**: Web framework for Node.js to handle HTTP requests and serve static files.
- **http**: Core Node.js module for creating an HTTP server.
- **path**: Core Node.js module for handling file paths.
- **url**: Core Node.js module for handling and parsing URLs.
- **ws**: WebSocket library for Node.js to handle WebSocket connections.

Setup

Importing Modules

JavaScript

```javascript
import express from 'express';
import http from 'http';
import path, { dirname } from 'path';
import { fileURLToPath } from 'url';
import { WebSocketServer } from 'ws';
```

- **express**: Importing the Express library to create the web server.
- **http**: Importing the HTTP library to create the server.
- **path, { dirname }**: Importing the path library and the dirname function to handle file paths.
- **fileURLToPath**: Importing from the URL module to get the current file path.
- **WebSocketServer**: Importing the WebSocketServer class from the ws library to handle WebSocket connections.

Helper to Get __dirname in ES6 Module

JavaScript

```javascript
const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);
```

- **__filename**: Determines the current file path.
- **__dirname**: Determines the current directory name.

Express Application and Server Setup

JavaScript

```javascript
const app = express();
const PORT = process.env.PORT || 3000;

// Serve static files from the 'public' directory
app.use(express.static(path.join(__dirname, 'public')));

// Create HTTP server by passing the Express app
const server = http.createServer(app);
```

- **app**: Initializes the Express application.
- **PORT**: Defines the server port, defaulting to 3000.
- **app.use**: Serves static files from the 'public' directory.
- **server**: Creates an HTTP server using the Express app.

WebSocket Integration

JavaScript

```javascript
const wss = new WebSocketServer({ server });
```

- **wss**: Creates a WebSocket server that integrates with the HTTP server.

Room Management

Store Rooms in an Object

JavaScript

```javascript
const rooms = {};
```

- **rooms**: An object to store room data, including clients and words.

Create a Room

JavaScript

```javascript
const createRoom = (data, ws) => {
    if (!rooms[data.room]) {
        rooms[data.room] = {
            name: data.name,
            clients: [ws],
            words: [],
        };
        console.log('User created room:', data.room);
        ws.send(
            JSON.stringify({
                type: 'room-created',
                room: data.room,
                name: data.name,
                participants: 1,
            }),
        );
    }
};
```

- **createRoom**: Function to create a new room if it doesn't already exist. Adds the client to the room and sends a confirmation message.

Join a Room

JavaScript

```javascript
const joinRoom = (data, ws) => {
    if (rooms[data.room]) {
        rooms[data.room].clients.push(ws);
        console.log('User joined room:', data.room);

        const response = JSON.stringify({
            type: 'room-joined',
            room: data.room,
            name: rooms[data.room].name,
            participants: rooms[data.room].clients.length,
            words: rooms[data.room].words,
        });

        rooms[data.room].clients.forEach((client) =>
client.send(response));
    }
};
```

- **joinRoom**: Function to join an existing room. Adds the client to the room and notifies all clients in the room.

Add Words to a Room

JavaScript

```javascript
const addWord = (data) => {
    if (rooms[data.room]) {
        const words = data.words.split(' ');
        rooms[data.room].words.push(...words);
        const response = JSON.stringify({
            type: 'words-added',
            room: data.room,
            words: rooms[data.room].words,
        });

        rooms[data.room].clients.forEach((client) =>
client.send(response));
    }
};
```

- **addWord**: Function to add words to a room's word list. Broadcasts the updated word list to all clients in the room.

WebSocket Event Handlers

Handle Incoming Messages

JavaScript

```javascript
const handleMessage = (ws, message) => {
    const data = JSON.parse(message);

    switch (data.type) {
        case 'create-room':
            createRoom(data, ws);
            break;
        case 'join-room':
            joinRoom(data, ws);
            break;
        case 'add-word':
            addWord(data);
            break;
        default:
            console.error('Unknown message type:', data.type);
    }
};
```

- **handleMessage**: Parses incoming messages and calls the appropriate function based on the message type.

Handle Closing Connections

JavaScript

```javascript
const handleClose = (ws) => {
    for (const room in rooms) {
        const index = rooms[room].clients.indexOf(ws);
        if (index > -1) {
            rooms[room].clients.splice(index, 1);

            const response = JSON.stringify({
                type: 'room-joined',
                room: room,
                name: rooms[room].name,
                participants: rooms[room].clients.length,
            });

            rooms[room].clients.forEach((client) =>
client.send(response));
        }
    }
};
```

- **handleClose**: Handles client disconnections, removes the client from their room, and updates the remaining clients.

Handle WebSocket Connections

JavaScript

```javascript
wss.on('connection', (ws) => {
    ws.on('message', (message) => handleMessage(ws, message));
    ws.on('close', () => handleClose(ws));
});
```

- **wss.on('connection')**: Sets up event listeners for new WebSocket connections. Handles incoming messages and client disconnections.

Start the Server

JavaScript

```javascript
server.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
});
```

- **server.listen**: Starts the HTTP server on the defined port and logs a message indicating the server is running.

Client-side JavaScript Code (app.js)

This JavaScript file enables real-time communication between a web client and a server using WebSockets. It initializes a word cloud chart and provides functionalities for creating and joining rooms, adding words, and managing UI updates based on WebSocket messages. The code also includes a profanity check for added words and allows downloading the word cloud as an image.

WebSocket Initialization

JavaScript

```javascript
const ws = new WebSocket(`ws://${window.location.host}`);
```

- **ws**: Establishes a WebSocket connection to the server using the current host.

Chart Initialization

JavaScript

```javascript
const chart = new
Chart(document.getElementById('canvas').getContext('2d'), {
    type: 'wordCloud',
    data: {},
    options: {
        title: { display: false },
        plugins: { legend: { display: false } },
        elements: {
            word: {
                color: '#003720',
                hoverColor: '#003720cc',
                hoverWeight: 'bold',
            },
        },
    },
});
```

- **chart**: Initializes a word cloud chart using Chart.js, setting the default styles and options.

WebSocket Event Handlers

Connection and Error Handling

JavaScript

```javascript
ws.onopen = () => console.log('Connected to the server');
ws.onerror = (error) => console.error('WebSocket error:', error);
ws.onclose = () => console.log('Disconnected from the server');
```

- **ws.onopen**: Logs a message when the WebSocket connection is established.
- **ws.onerror**: Logs any WebSocket errors.
- **ws.onclose**: Logs a message when the WebSocket connection is closed.

Message Handling

```javascript
ws.onmessage = (event) => {
    const room = JSON.parse(event.data);
    handleRoomEvent(room);
};
```

- **ws.onmessage**: Handles incoming messages from the WebSocket server and calls `handleRoomEvent`.

Room Event Handling

Function to Handle Room Events

```javascript
function handleRoomEvent(room) {
    const updateRoomDetails = (room) => {
        document.getElementById('room-name').innerText = room.name;
        document.getElementById('room-id').innerText = room.room;
        document.getElementById('room-participants').innerText =
room.participants;
    };

    switch (room.type) {
        case 'room-created':
            updateRoomDetails(room);
            showPage('page_active_room');
            break;
        case 'room-joined':
            updateRoomDetails(room);
            showPage('page_active_room');
            updateChart(room.words);
            break;
        case 'words-added':
            document.getElementById('words').value = '';
            updateChart(room.words);
            break;
        default:
            console.warn('Unknown room type:', room.type);
            break;
    }
}
```

- **handleRoomEvent**: Updates the UI and chart based on the type of room event received.

Utility Functions

Function to Generate a Random ID

```javascript
function randomID() {
    return Math.random().toString(36).slice(2, 11);
}
```

- **randomID**: Generates a random alphanumeric ID for rooms.

Function to Show a Specific Page

```javascript
const showPage = (page) => {
    document.querySelectorAll('[id^="page"]').forEach((p) =>
p.classList.add('hidden'));
    document.getElementById(page).classList.remove('hidden');
};
```

- **showPage**: Displays the specified page and hides others.

Room Management Functions

Function to Create a Room

```javascript
const createRoom = () => {
    const roomName = document.getElementById('topic').value;
    if (!roomName) {
        alert('Please enter a room name!');
        return;
    }

    const message = {
        type: 'create-room',
        room: randomID(),
        name: roomName,
    };
    ws.send(JSON.stringify(message));
};
```

- **createRoom**: Creates a new room and sends a request to the server.

Function to Join a Room

JavaScript

```javascript
const joinRoom = () => {
    const roomId = document.getElementById('id').value;
    if (!roomId) {
        alert('Please enter a room ID!');
        return;
    }

    const message = {
        type: 'join-room',
        room: roomId,
    };
    ws.send(JSON.stringify(message));
    showPage('page_active_room');
};
```

- **joinRoom**: Joins an existing room and sends a request to the server.

Function to Copy the Room ID

JavaScript

```javascript
const copyRoomID = () => {
    const roomID = document.getElementById('room-id').textContent;
    navigator.clipboard
        .writeText(roomID)
        .then(() => alert('Room ID copied to clipboard'))
        .catch((err) => console.error('Failed to copy: ', err));
};
```

- **copyRoomID**: Copies the room ID to the clipboard.

Function to Add Words

JavaScript ⌄

```javascript
const addWords = () => {
    const words = document.getElementById('words').value;
    const roomID = document.getElementById('room-id').textContent;
    if (!words) {
        alert('Please enter words!');
        return;
    }

    // Check for profanity
    checkProfanity(words).then((profanity) => {
        if (profanity) {
            alert('Profanity detected!');
            document.getElementById('words').value = '';
        } else {
            const message = {
                type: 'add-word',
                room: roomID,
                words,
            };
            ws.send(JSON.stringify(message));
        }
    });
};
```

- **addWords:** Adds words to the room and checks for profanity before sending them to the server.

Chart Update and Profanity Check Functions

Function to Update the Chart

JavaScript ⌄

```javascript
const updateChart = (data) => {
    if (data.length === 0) {
        return;
    }

    const wordsCount = data.reduce((acc, word) => {
        const found = acc.find((w) => w.key === word);
        if (found) {
            found.value += 1;
        } else {
            acc.push({ key: word, value: 1 });
        }
        return acc;
    }, []);

    chart.data = {
        labels: wordsCount.map((d) => d.key),
        datasets: [{ label: '', data: wordsCount.map((d) => 20 +
d.value * 10) }],
    };
    chart.update();
};
```

- **updateChart**: Updates the word cloud chart with the new words.

Function to Check for Profanity

JavaScript

```javascript
const checkProfanity = async (message) => {
    const res = await fetch('https://vector.profanity.dev', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ message }),
    });

    if (!res.ok) {
        console.error('Profanity check failed:', res.status);
        return false;
    }

    const { isProfanity } = await res.json();
    return isProfanity;
};
```

- **checkProfanity**: Checks the message for profanity using an external API.

Function to Download the Word Cloud

JavaScript

```javascript
const downloadWordCloud = () => {
    const canvas = document.getElementById('canvas');
    const link = document.createElement('a');
    link.download = 'wordcloud.png';
    link.href = canvas.toDataURL('image/png');
    link.click();
};
```

- **downloadWordCloud**: Downloads the current word cloud as a PNG image.

Function to fetch a random Wikipedia topic

JavaScript

```javascript
const fetchRandomWikipediaTopic = async () => {
    try {
        // Fetch a random article
        const randomResponse = await fetch(
            'https://en.wikipedia.org/w/api.php?
action=query&list=random&rnnamespace=0&rnlimit=1&format=json&origin
=*',
        );
        const randomData = await randomResponse.json();

        // Get the title of the random article
        const title = randomData.query.random[0].title;

        // Fetch the summary of the random article
        const summaryResponse = await fetch(
            `https://en.wikipedia.org/w/api.php?
action=query&prop=extracts&exintro&explaintext&titles=${encodeURICo
mponent(title)}&format=json&origin=*`,
        );
        const summaryData = await summaryResponse.json();

        // Extract the page ID to get the extract
        const pageId = Object.keys(summaryData.query.pages)[0];
        const extract = summaryData.query.pages[pageId].extract;

        // Display the topic and summary on the webpage
        document.getElementById('topic').value = `What words come
to mind when you hear about "${title}"?`;
        document.getElementById('summary').innerText = `Summary:
\n${extract}`;
    } catch (error) {
        console.error('Error fetching random Wikipedia topic:',
error);
    }
};
```

- **fetchRandomWikipediaTopic**: Fetches a random Wikipedia topic and displays it on the webpage

HTML File (index.html)

This HTML document provides the structure and styling for "WordPulse", an application that generates dynamic word clouds in real-time. The page includes sections for creating or joining a room, displaying active room details, and interacting with the word cloud. TailwindCSS is used for styling, and Chart.js with the word cloud plugin is used for rendering the word cloud.

Document Structure

Head Section

The `<head>` section contains metadata, links to external stylesheets, and scripts.

HTML

```html
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>WordPulse: Create Dynamic and Engaging Word Clouds in
Real-Time</title>
    <meta name="description" content="Engage your audience with
WordPulse. Quickly generate captivating word clouds that update in
real-time based on collective input. Perfect for meetings,
classrooms, and events." />

    <!-- TailwindCSS -->
    <script src="https://cdn.tailwindcss.com"></script>
    <script src="https://cdn.tailwindcss.com?
plugins=forms,typography"></script>
    <style type="text/tailwindcss">
        @layer base {
            @font-face {
                font-family: 'DM Sans';
                font-weight: 100, 200, 300, 400, 500, 600, 700,
800, 900;
                font-display: swap;
                src: url('/fonts/DMSans-
VariableFont_opsz,wght.woff2') format('woff2');
            }
        }
    </style>
    <script>
        tailwind.config = {
            theme: {
                extend: {
                    fontFamily: {
                        sans: ['DM Sans', 'ui-sans-serif', 'system-
ui'],
                    },
                    colors: {
                        primary: '#003720',
                        secondary: '#afb7b4',
                    },
                },
            },
        };
    </script>
</head>
```

- **meta**: Provides character encoding, viewport settings, and description for SEO.
- **TailwindCSS**: Links to the TailwindCSS CDN and plugins for forms and typography.
- **Custom Styles**: Defines custom fonts using the `@font-face` rule within TailwindCSS.
- **Tailwind Configuration**: Extends Tailwind's default theme to include custom fonts and colors.

Body Section

The `<body>` section contains the main content of the page, organized into different sections for various functionalities.

Body Attributes

HTML

```html
<body class="min-h-screen text-gray-900 antialiased h-full"
style="background: radial-gradient(at 53% 78%, rgba(255, 255, 0,
0.3) 0px, transparent 50%), radial-gradient(at 71% 91%, rgba(51,
255, 0, 0.3) 0px, transparent 50%), radial-gradient(at 31% 91%,
rgba(255, 128, 0, 0.17) 0px, transparent 50%);">
```

- **class**: Applies TailwindCSS utility classes for styling.
- **style**: Sets a custom background with radial gradients.

Main Page

```html
<div class="py-12">
    <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8" id="page_main">
        <div class="lg:text-center">
            <h2 class="text-base text-primary font-semibold tracking-wide uppercase">WORD PULSE</h2>
            <p class="mt-2 text-4xl md:text-5xl lg:text-7xl leading-8 font-extrabold tracking-tight text-secondary max-w-4xl lg:mx-auto">
                Engage your audience with dynamic word clouds using <span class="text-primary">WordPulse</span>.
            </p>
            <p class="mt-4 max-w-2xl text-xl text-gray-500 lg:mx-auto">
                WordPulse lets you create captivating word clouds quickly and easily. Pose your question and watch as vibrant words appear in real-time, reflecting the collective input of your audience.
            </p>
        </div>
        <div class="mt-10 flex gap-3 items-center lg:justify-center">
            <button onclick="showPage('page_create_room')" class="w-full px-6 py-3 border border-transparent text-base font-medium rounded-md text-white bg-primary shadow-sm hover:bg-primary/80 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-primary/80 sm:mt-0 sm:flex-shrink-0 sm:inline-flex sm:items-center sm:w-auto">
                Create Room
            </button>
            <button onclick="showPage('page_join_room')" class="w-full px-6 py-3 border border-transparent text-base font-medium rounded-md text-primary bg-gray-100 shadow-sm hover:bg-gray-200 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-primary/80 sm:mt-0 sm:flex-shrink-0 sm:inline-flex sm:items-center sm:w-auto">
                Join Room
            </button>
        </div>
    </div>
</div>
```

- **Main Page Container**: Contains the introductory text and buttons to create or join a room.

- **Buttons:** Trigger JavaScript functions to navigate to the respective pages.

Create Room Page

HTML

```html
<div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 hidden"
id="page_create_room">
                <div class="lg:text-center">
                    <h2 class="text-base text-primary font-semibold
tracking-wide uppercase">WORD PULSE</h2>
                    <p class="mt-2 text-3xl leading-8 font-
extrabold tracking-tight text-primary/80 sm:text-4xl">Create a
topic</p>
                    <p class="mt-4 max-w-2xl text-xl text-gray-500
lg:mx-auto">ex. What words come to mind when you hear about climate
change?</p>
                </div>

                <div class="mt-10 max-w-lg lg:mx-auto">
                    <div class="mt-3">
                        <label for="topic" class="sr-only">Room
Name</label>
                        <textarea
                            name="topic"
                            id="topic"
                            required
                            class="block w-full py-3 text-base
rounded-md placeholder-gray-500 shadow-sm focus:ring-primary/80
focus:border-primary/80 sm:flex-1 border-gray-300"
                            placeholder="Enter topic here..."
                        ></textarea>
                    </div>
                    <div class="mt-3 flex flex-col gap-3">
                        <button
                            onclick="createRoom()"
                            class="mt-3 w-full px-6 py-3 border
border-transparent text-base font-medium rounded-md text-white bg-
primary shadow-sm hover:bg-primary/80 focus:outline-none
focus:ring-2 focus:ring-offset-2 focus:ring-primary/80 sm:mt-0
sm:flex-shrink-0 sm:inline-flex sm:items-center sm:justify-center
sm:w-auto"
                        >
                            Create
                        </button>
                        <button
                            id="random-topic-button"
                            onclick="fetchRandomWikipediaTopic()"
                            class="w-full px-6 py-3 border border-transparent text-base font-medium rounded-md text-white bg-primary shadow-sm hover:bg-primary/80 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-primary/80 sm:mt-0 sm:inline-flex sm:items-center sm:justify-center sm:w-auto"
                        >
                            Generate Random Topic
                        </button>
                    </div>
                    <div class="mt-6">
                        <p id="summary"></p>
                    </div>
                </div>
            </div>
```

- **Create Room Container**: Includes form elements to input the topic, a button to create the room and to generate random topic.

Join Room Page

HTML

```html
<div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 hidden" id="page_join_room">
    <div class="lg:text-center">
        <h2 class="text-base text-primary font-semibold tracking-wide uppercase">WORD PULSE</h2>
        <p class="mt-2 text-3xl leading-8 font-extrabold tracking-tight text-primary/80 sm:text-4xl">Join Room</p>
        <p class="mt-4 max-w-2xl text-xl text-gray-500 lg:mx-auto">Enter the room id to join the room.</p>
    </div>
    <div class="mt-10 max-w-lg lg:mx-auto">
        <div class="mt-3 sm:flex">
            <label for="id" class="sr-only">Room ID</label>
            <input type="text" name="id" id="id" class="block w-full py-3 text-base rounded-md placeholder-gray-500 shadow-sm focus:ring-primary/80 focus:border-primary/80 sm:flex-1 border-gray-300" placeholder="Enter Room ID here..." />
            <button onclick="joinRoom()" class="mt-3 w-full px-6 py-3 border border-transparent text-base font-medium rounded-md text-white bg-primary shadow-sm hover:bg-primary/80 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-primary/80 sm:mt-0 sm:ml-3 sm:flex-shrink-0 sm:inline-flex sm:items-center sm:w-auto">
                Join
            </button>
        </div>
    </div>
</div>
```

- **Join Room Container**: Includes form elements to input the room ID and a button to join the room.

Active Room Page

HTML

```html
<div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 hidden relative"
id="page_active_room">
    <div class="lg:text-center">
        <h2 class="text-base text-primary font-semibold tracking-
wide uppercase">WORD PULSE</h2>
        <p class="mt-2 text-3xl text-pretty lg:mx-auto max-w-xl
lg:text-center leading-8 font-extrabold tracking-tight text-
primary/80 sm:text-4xl">
            <span id="room-name"></span>
        </p>
        <p class="mt-4 max-w-2xl lg:mx-auto text-xl text-gray-500
flex items-center lg:justify-center">
            <span class="mr-2">Room ID: <span id="room-id"></span>
</span>
            <button title="Copy to clipboard"
onclick="copyRoomID()" class="ml-2 text-gray-500 hover:text-gray-
700 focus:outline-none">
                <svg xmlns="http://www.w3.org/2000/svg" width="24"
height="24" viewBox="0 0 24 24" fill="none" stroke="currentColor"
stroke-width="2" stroke-linecap="round" stroke-linejoin="round"
class="lucide lucide-copy">
                    <rect width="14" height="14" x="8" y="8" rx="2"
ry="2" />
                    <path d="M4 16c-1.1 0-2-.9-2-2V4c0-1.1.9-2 2-
2h10c1.1 0 2 .9 2 2" />
                </svg>
            </button>
        </p>
        <p class="mt-4 max-w-2xl text-xl text-gray-500 lg:mx-auto">
            <span>Participants: <span id="room-participants">
</span></span>
        </p>
    </div>

    <!-- Exit Button -->
    <button type="button" class="absolute -top-4 lg:top-4 right-4
px-4 py-2 border border-transparent text-base font-medium rounded-
md text-white bg-red-600 shadow-sm hover:bg-red-700 focus:outline-
none focus:ring-2 focus:ring-offset-2 focus:ring-red-500"
onclick="location.reload()">
        Exit Room
    </button>

    <div class="lg:mx-auto w-full min-h-[60vh]">
        <canvas id="canvas"></canvas>
    </div>

    <div class="fixed inset-x-4 sm:inset-x-6 bottom-6 lg:inset-x-
0">
        <div class="max-w-lg lg:mx-auto">
            <div class="grid place-content-center">
                <button onclick="downloadWordCloud()"
class="underline">Download WordCloud</button>
            </div>
            <div class="mt-3 sm:flex">
                <label for="words" class="sr-only">Words</label>
                <input type="text" name="words" id="words"
class="block w-full py-3 text-base rounded-md placeholder-gray-500
shadow-sm focus:ring-primary/80 focus:border-primary/80 sm:flex-1
border-gray-300" placeholder="Enter words here..." />
                <button onclick="addWords()" class="mt-3 w-full px-
6 py-3 border border-transparent text-base font-medium rounded-md
text-white bg-primary shadow-sm hover:bg-primary/80 focus:outline-
none focus:ring-2 focus:ring-offset-2 focus:ring-primary/80 sm:mt-0
sm:ml-3 sm:flex-shrink-0 sm:inline-flex sm:items-center sm:w-auto">
                    Share
                </button>
            </div>
        </div>
    </div>
</div>
```

- **Active Room Container**: Displays room information and participants count.
- **Canvas**: Renders the word cloud.
- **Word Input**: Input field and button to add words to the word cloud.
- **Exit Button**: Allows the user to exit the room.

Scripts

HTML

```html
<script src="https://unpkg.com/chart.js@4.4"></script>
<script src="https://unpkg.com/chartjs-chart-wordcloud@4.4">
</script>
<script defer src="app.js"></script>
```

- **Chart.js and WordCloud Plugin**: External libraries for creating the word cloud.
- **app.js**: Custom JavaScript file for handling the application's functionality.