**INTRODUCTION**

The **Web-Based Word Antakshari Game** is an interactive online version of the traditional word-based game, where players take turns forming words based on the last letter of the previous word. This project leverages **HTML, CSS, and JavaScript** for an engaging user interface, while **Mongo DB** handle the backend, ensuring smooth game logic, user authentication, and score management. The game will feature **single-player and multiplayer modes**, real-time word validation using a dictionary API, and a leaderboard to track player progress. By integrating these elements, the project aims to provide an entertaining and seamless gaming experience for users of all ages.

One of the key features of this web-based game is its **real-time word validation**, which ensures that only valid words are accepted, making the gameplay fair and challenging. Additionally, the game can include a **timer** to increase the difficulty and encourage quick thinking. With its simple yet effective gameplay mechanics, this project is not only entertaining but also enhances players’ vocabulary and linguistic skills.

**PURPOSE**

The **primary purpose** of the Web-Based Word Antakshari Game is to create an interactive and entertaining online platform that enhances vocabulary skills while providing a fun and competitive experience. By bringing the classic word game to a digital format, the project aims to make the game more accessible, allowing players to enjoy it anytime and from anywhere.

This game is designed to promote **language learning and cognitive skills**, as players must think quickly and recall words that fit the given criteria. Additionally, it encourages **social engagement** by offering multiplayer functionality, where players can compete against friends, family, or even random opponents. The **real-time word validation** feature ensures fairness and accuracy, making the game both educational and enjoyable.

From a technical perspective, the project serves as a practical implementation of **web development technologies**, integrating frontend and backend components to create a seamless user experience. It also provides an opportunity to explore concepts such as **database management, API integration, and real-time gameplay mechanics**.

**SCOPE**

The **Web-Based Word Antakshari Game** has a broad scope, covering various aspects of **education, entertainment, and technology**. It is designed to provide an engaging word-based gaming experience while enhancing players' **vocabulary, linguistic skills, and cognitive abilities**. The project can be expanded in multiple ways, making it suitable for different user groups, including students, language learners, and casual gamers.

#### ****1. Functional Scope****

* **Single-Player Mode** – Players can play against an AI or practice their vocabulary skills.
* **Multiplayer Mode** – Users can challenge friends or random opponents online.
* **Real-Time Word Validation** – Integration of a dictionary API ensures valid words are accepted.
* **Leaderboard and Score Tracking** – Players can view their scores and compete for high rankings.
* **User Authentication** – Players can register, log in, and maintain a personal profile.
* **Timed Rounds (Optional)** – A timer can be added to increase difficulty and engagement.

#### ****2. Technological Scope****

* **Frontend Development:** Built using **HTML, CSS, and JavaScript** for a responsive and interactive UI.
* **Backend Development:** Powered by Mongo DB for user management, game logic, and data storage.
* **API Integration:** Use of a dictionary API for real-time word validation.
* **Database Management:** Efficient storage and retrieval of user profiles, scores, and game history.

#### ****3. Future Enhancements****

* **AI Opponent** – A smart AI that can play against users in single-player mode.
* **Mobile-Friendly Version** – Ensuring compatibility with smartphones and tablets.
* **Chat Feature** – Enabling real-time communication between players in multiplayer mode.
* **Multiple Game Modes** – Variations like themed word challenges (e.g., animals, countries).
* **Integration with Social Media** – Allowing users to share scores and invite friends to play

**OBJECTIVE**

The objective of the **Web-Based Word Antakshari Game** is to create an engaging and interactive platform that combines entertainment with vocabulary enhancement. This project aims to digitalize the traditional word-based game, allowing players to form words based on the last letter of the previous word in an intuitive and user-friendly environment. By incorporating **single-player and multiplayer modes**, the game encourages both individual learning and competitive play.

A key focus is on **enhancing language skills**, as players must recall and construct words while adhering to real-time validation using a dictionary API. Additionally, the project ensures a smooth gaming experience through a **robust backend system** powered by Mongo DB, which manages user authentication, game logic, and score tracking. The implementation of a **leaderboard and scoring system** adds a competitive element, motivating players to improve their performance.

Furthermore, the game is designed to be **responsive and scalable**, ensuring accessibility across different devices and allowing room for future enhancements such as AI-based opponents, themed word challenges, and social media integration. Overall, the Web-Based Word Antakshari Game aims to provide an enjoyable and educational experience that fosters **language development, quick thinking, and interactive engagement**.

**PROJECT REQUIRMENT**

To successfully develop the **Web-Based Word Antakshari Game**, several **hardware, software, and functional requirements** must be considered. These requirements ensure smooth gameplay, efficient backend operations, and a user-friendly interface.

### ****1. Functional Requirements****.

* **Single-Player Mode:** A mode where users can play alone, either against an AI or for practice.
* **Word Validation:** Integration of a **dictionary API** to check word validity in real time.
* **Scoring System:** Points should be awarded based on correct words, with possible bonuses for longer words.
* **Leaderboard:** A ranking system to display top players based on their scores.
* **Timer :** A time limit for each turn to increase challenge and engagement.

### ****2. Non-Functional Requirements****

* **Performance Optimization:** The system should run smoothly without delays or crashes.
* **Security Measures:** User data ( scores) must be securely stored using encryption.
* **Scalability:** The system should allow future expansions like AI opponents, new game modes, and social media integration.

### ****3. Software Requirements****

**Frontend Technologies:**

* + **HTML, CSS, JavaScript** – For creating the user interface and interactivity.
  + **Bootstrap or Tailwind CSS** – To ensure responsive and visually appealing design.

**Backend Technologies:**

* + **MONGO DB** – To handle user authentication, game logic, and data processing..
  + **API Integration (Dictionary API)** – To validate words in real-time.

**Development & Deployment Tools:**

* + **Code Editor:** Visual Studio Code, Sublime Text, or any preferred IDE.
  + **Web Server:** XAMPP, WAMP, or an online hosting service for deployment.
  + **Version Control:** GitHub or GitLab for source code management.

### ****4. Hardware Requirements****

**For Development:**

* + **Processor:** Intel i3/i7 or AMD equivalent.
  + **RAM:** Minimum 2GB (Recommended: 16GB for smooth development).
  + **Storage:** At least 20GB free disk space.

**For Hosting (if deployed online):**

* + **Server:** Linux/Windows hosting with server.js and Mongo DB.

**CODE**

**HTML**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Antakshari Game 🎵</title>

  <link rel="stylesheet" type="text/css" href="style.css">

</head>

<body>

  <div class="container">

    <h1>🎤 Antakshari Game</h1>

    <div id="player-info">

      <p><strong>Player:</strong> <span id="player-name">Loading...</span></p>

      <p><strong>Score:</strong> <span id="player-score">0</span></p>

      <p><strong>Time Played:</strong> <span id="time-played">0s</span></p>

      <p><strong>Time Left:</strong> <span id="time-played">0s</span></p>

    </div>

    <div id="game-area">

      <p id="previous-player"></p>

      <p id="start-word">Loading word...</p>

      <input type="text" id="user-word" placeholder="Enter your word..." />

      <button onclick="submitWord()">Submit</button>

      <p id="result"></p>

    </div>

  </div>

  <script src="script.js"></script>

</body>

</html>

**CSS**

CSS

body {

    font-family: Arial, sans-serif;

    background-color: #f4f4f4;

    text-align: center;

    margin-top: 50px;

  }.container {

    background: #fff;

    padding: 20px;

    width: 400px;

    margin: 0 auto;

    border-radius: 8px;

    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

  }

  h1 {

    color: #333;

  }

  #player-info {

    margin-bottom: 20px;

    text-align: left;

    padding-left: 20px;

  }

  #game-area {

    margin-top: 20px;

  }

input {

    width: 80%;

    padding: 10px;

    margin-top: 10px;

    border: 1px solid #ccc;

    border-radius: 5px;

  }

  button {

    padding: 10px 20px;

    background-color: #28a745;

    color: #fff;

    border: none;

    margin-top: 10px;

    cursor: pointer;

    border-radius: 5px;

  }

  button:hover {

    background-color: #218838;

  }

  #result {

    margin-top: 20px;

    font-weight: bold;

  }

  #previous-player {

    margin-top: 15px;

    color: #555;

    font-style: italic;

  }

**Script.js**

let lastWord = "";

let lastLetter = "";

let playerName = "";

let score = 0;

let timer;

let wrongAttempts = 0;

const maxWrongAttempts = 3;

const timeLimit = 10;

// Retrieve highest scorer and score from localStorage

let highestScorer = localStorage.getItem("highestScorer") || "None";

let highestScore = parseInt(localStorage.getItem("highestScore")) || 0;

// Start the game and get the player's name

function startGame() {

  playerName = prompt("Enter your name:");

  if (!playerName) {

    playerName = "Player 1";

  }

  document.getElementById("player-name").innerText = playerName;

  document.getElementById("player-score").innerText = score;

  document.getElementById(

    "previous-player"

  ).innerText = 🏆 Highest Scorer: ${highestScorer}, Score: ${highestScore};

  wrongAttempts = 0;

  // Fetch a random word from the server

  fetch("/start")

    .then((response) => response.json())

    .then((data) => {

      if (data.word) {

        lastWord = data.word;

        lastLetter = lastWord.slice(-1).toLowerCase();

        document.getElementById("start-word").innerText = Start with: ${lastWord};

        startTimer();

      }

    });

}

// Start the 10-second timer

function startTimer() {

  clearInterval(timer);

  let timeLeft = timeLimit;

  document.getElementById("time-played").innerText = ${timeLeft}s;

  timer = setInterval(() => {

    timeLeft -= 1;

    document.getElementById("time-played").innerText = ${timeLeft}s;

    if (timeLeft <= 0) {

      clearInterval(timer);

      handleTimeout(); // Handle timeout if player doesn't answer

    }

  }, 1000);

}

// Submit user's word

async function submitWord() {

  const userWord = document.getElementById("user-word").value.trim();

  if (!userWord) {

    document.getElementById("result").innerText = "❗ Please enter a word.";

    return;

  }

  const response = await fetch("/check-word", {

    method: "POST",

    headers: {

      "Content-Type": "application/json",

    },

    body: JSON.stringify({ userWord, lastLetter }),

  });

  if (response.ok) {

    const data = await response.json();

    document.getElementById("result").innerText = data.message;

    lastWord = userWord;

    lastLetter = lastWord.slice(-1).toLowerCase();

    document.getElementById("start-word").innerText = Your word: ${lastWord};

    document.getElementById("user-word").value = "";

    // Increase score by 10 points

    score += 10;

    document.getElementById("player-score").innerText = score;

    // Reset timer after successful attempt

    startTimer();

  } else {

    const errorText = await response.text();

    handleWrongAttempt(errorText);

  }

}

// Handle wrong attempt

function handleWrongAttempt(message) {

  wrongAttempts++;

  document.getElementById("result").innerText = ❌ ${message};

  if (wrongAttempts >= maxWrongAttempts) {

    endGame(); // End game if 3 wrong attempts

  } else {

    startTimer(); // Restart timer for next attempt

  }

}

// Handle timeout if player doesn't submit in 10 seconds

function handleTimeout() {

  document.getElementById("result").innerText = "⏰ Time's up! Game over.";

  endGame(); // Automatically end game on timeout

}

// End the game and restart with a new name

function endGame() {

  clearInterval(timer);

  updateHighestScore(); // Check and update highest score

  alert(Game Over! Final score: ${score});

  resetGame();

}

// Update the highest score if the current score is higher

function updateHighestScore() {

  if (score > highestScore) {

    highestScore = score;

    highestScorer = playerName;

    localStorage.setItem("highestScorer", highestScorer);

    localStorage.setItem("highestScore", highestScore);

  }

}

// Reset the game and ask for a new name

function resetGame() {

  score = 0;

  wrongAttempts = 0;

  startGame();

}

// Save highest scorer before leaving

window.onbeforeunload = () => {

  updateHighestScore();

};

// Start the game when the page loads

window.onload = startGame;

**Server.js**

const express = require("express");

const { MongoClient } = require("mongodb");

const path = require("path");

const app = express();

const PORT = 3000;

// MongoDB connection URI

const uri = "mongodb://127.0.0.1:27017";

const client = new MongoClient(uri);

// Middleware to parse JSON

app.use(express.json());

// Serve static files (for frontend)

app.use(express.static(path.join(\_\_dirname, "public")));

// Track used words for the session

let usedWords = [];

// Connect to MongoDB and run server

async function run() {

  try {

    await client.connect();

    console.log("✅ Connected to MongoDB!");

    const db = client.db("antakshari\_db");

    const wordsCollection = db.collection("words");

    // Get a random word to start the game

    app.get("/start", async (req, res) => {

      const words = await wordsCollection.aggregate([{ $sample: { size: 1 } }]).toArray();

      if (words.length > 0) {

        const startingWord = words[0].word;

        usedWords = [startingWord.toLowerCase()]; // Reset used words

        res.json({ word: startingWord });

      } else {

        res.status(500).send("No words found in the database.");

      }

    });

    // Check user-submitted word

    app.post("/check-word", async (req, res) => {

      const { userWord, lastLetter } = req.body;

      const wordLower = userWord.toLowerCase();

      // Check if the word starts with the correct letter

      if (!userWord || wordLower[0] !== lastLetter.toLowerCase()) {

        return res.status(400).send(❌ Invalid word. It must start with "${lastLetter.toUpperCase()}".);

      }

      // Check if the word has already been used

      if (usedWords.includes(wordLower)) {

        return res.status(400).send("❌ Word already used! Try another word.");

      }

      // Check if the word exists in the database

      const foundWord = await wordsCollection.findOne({ word: userWord });

      if (foundWord) {

        usedWords.push(wordLower); // Add word to used list

        res.json({ valid: true, message: "✅ Valid word! Continue..." });

      } else {

        res.status(400).send("❌ Word not found in the database.");

      }

    });

    // Start server

    app.listen(PORT, () => {

      console.log(🚀 Server running at http://localhost:${PORT});

    });

  } catch (err) {

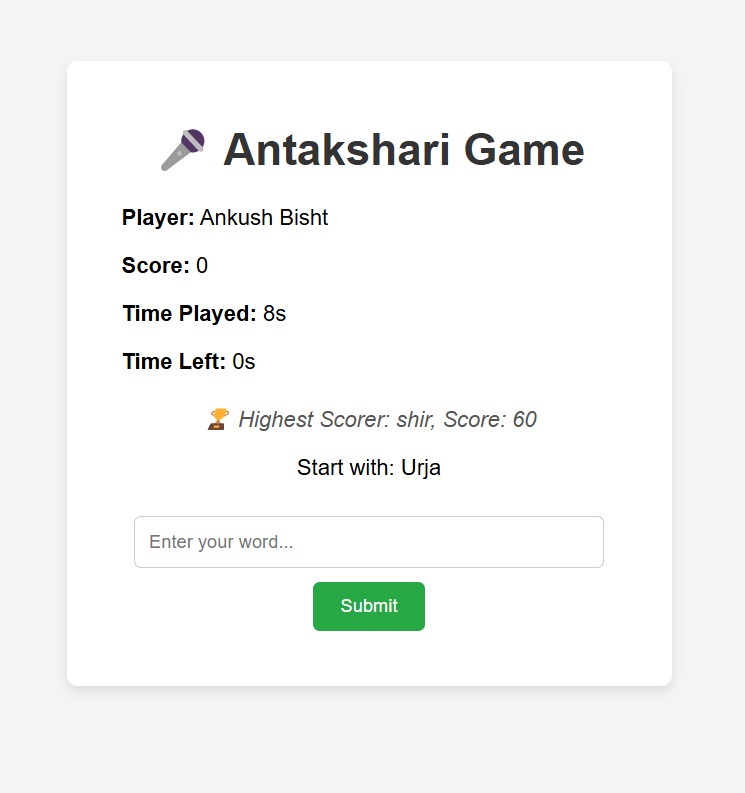
    console.error("❌ Connection error:", err);

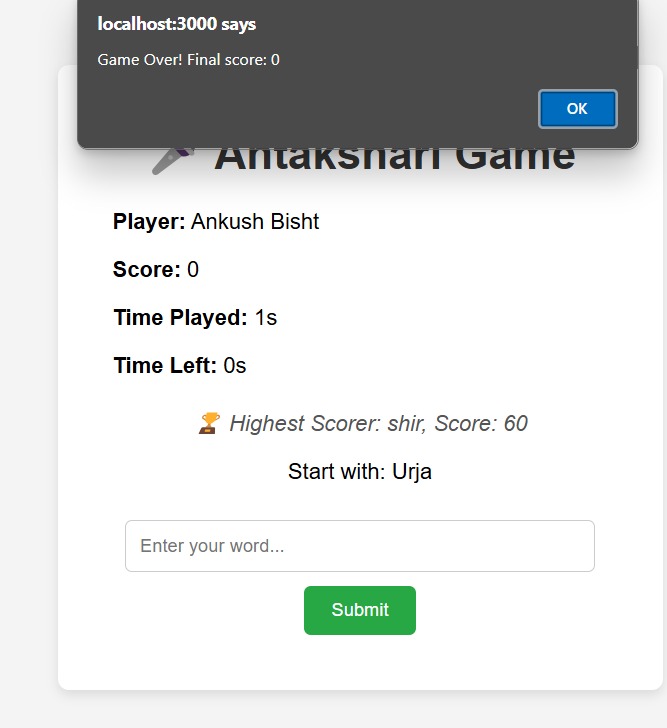
  }

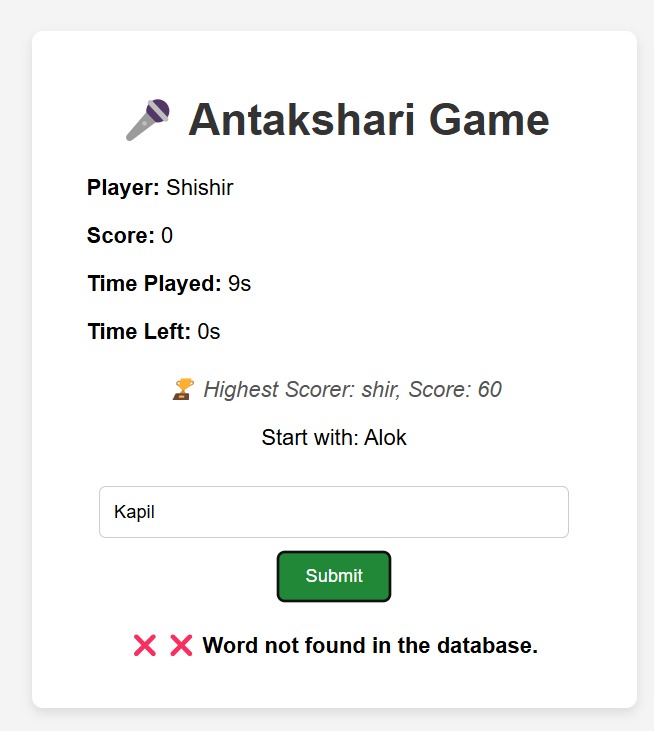
}

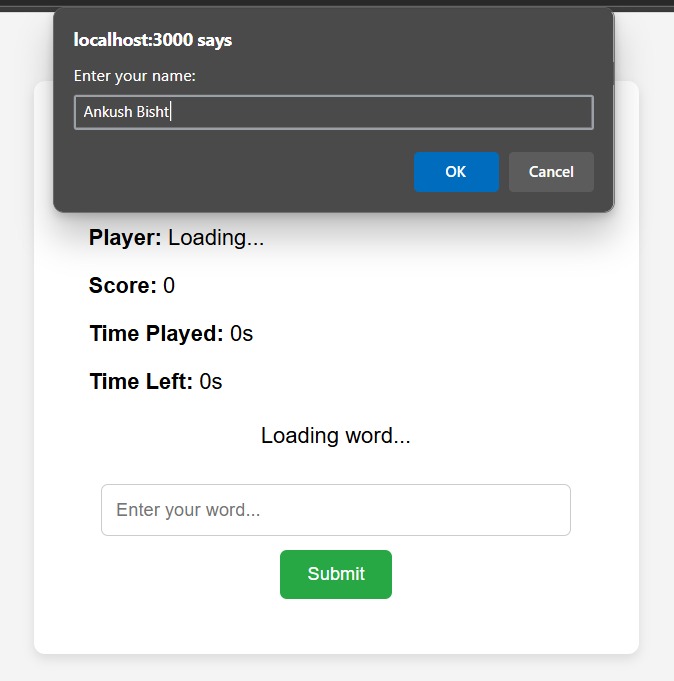
run();

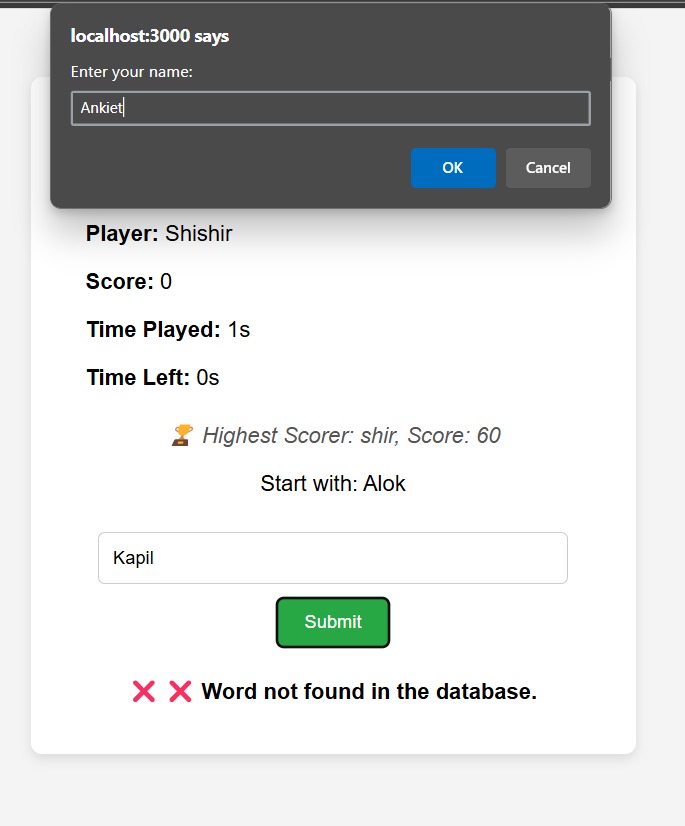
**SCREEN SHOT**

****

****

****

****

****