TASK 03

Tic-Tac-Toe Web application

To build a tic-tac-toe web application, you can use HTML, CSS, and JavaScript. By implementing functions to handle user clicks, track game state, and check for winning conditions, you can create an interactive and engaging tic-tac-toe game. With these technologies and functionalities, users can play against each other or against an AI opponent, aiming to get three markers in a row to win the game. With coding

Certainly! Below is a basic implementation of a Tic-Tac-Toe web application using HTML, CSS, and JavaScript. This implementation allows two players to play against each other. For simplicity, this example does not include an AI opponent.

HTML (index.html)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tic-Tac-Toe</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Tic-Tac-Toe</h1>
        <div class="board" id="board">
            <div class="cell" data-index="0"></div>
            <div class="cell" data-index="1"></div>
            <div class="cell" data-index="2"></div>
            <div class="cell" data-index="3"></div>
            <div class="cell" data-index="4"></div>
            <div class="cell" data-index="5"></div>
            <div class="cell" data-index="6"></div>
            <div class="cell" data-index="7"></div>
            <div class="cell" data-index="8"></div>
        </div>
        <button id="restartButton">Restart Game</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (style.css)

```css
body {
    font-family: Arial, sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    background-color: #f0f0f0;
}

.container {
    text-align: center;
}
```

```
.board {
    display: grid;
    grid-template-columns: repeat(3, 100px);
    gap: 10px;
    margin: 20px auto;
}

.cell {
    width: 100px;
    height: 100px;
    background-color: #fff;
    display: flex;
    justify-content: center;
    align-items: center;
    font-size: 2em;
    cursor: pointer;
    border: 2px solid #000;
}

#restartButton {
    margin-top: 20px;
    padding: 10px 20px;
    font-size: 1em;
    cursor: pointer;
}
```

JavaScript (script.js)

```
const board = document.getElementById('board');
const cells = document.querySelectorAll('.cell');
const restartButton = document.getElementById('restartButton');

let currentPlayer = 'X';
let gameState = ['', '', '', '', '', '', '', '', ''];
const winningConditions = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6]
];

cells.forEach(cell => cell.addEventListener('click', handleCellClick));
restartButton.addEventListener('click', restartGame);

function handleCellClick(event) {
    const clickedCell = event.target;
    const clickedCellIndex = parseInt(clickedCell.getAttribute('data-index'));

    if (gameState[clickedCellIndex] !== '' || checkWin()) {
        return;
    }

    gameState[clickedCellIndex] = currentPlayer;
    clickedCell.textContent = currentPlayer;

    if (checkWin()) {
        alert(`${currentPlayer} wins!`);
        return;
    }

    if (gameState.every(cell => cell !== '')) {
        alert('Draw!');
        return;
    }

    currentPlayer = currentPlayer === 'X' ? 'O' : 'X';
}

function checkWin() {
    return winningConditions.some(condition => {
        return condition.every(index => gameState[index] === currentPlayer);
    });
}

function restartGame() {
    gameState = ['', '', '', '', '', '', '', '', ''];
    cells.forEach(cell => cell.textContent = '');
    currentPlayer = 'X';
}
```

Explanation

1.HTML (index.html):

This sets up the structure of the webpage with a container for the game board and a restart button.
Each cell in the game board is a div with a data-index attribute to track its position.

2.CSS (style.css):

Styles the game board, cells, and the restart button.
Uses a grid layout for the board to neatly arrange the cells.

3.JavaScript (script.js):

Manages the game state and logic.
Handles user clicks on cells to mark them with the current player's symbol ('X' or 'O').
Checks for winning conditions after each move.
Provides a function to restart the game, resetting the state and the board.

You can extend this basic implementation by adding more features, such as an AI opponent or a more sophisticated UI. This basic setup, however, should provide a solid foundation for a functional Tic-Tac-Toe game.