# File online.js:

const socket =io(window.location.origin);

let room\_id;

let board;

let solution;

let timerInterval;

let startTime;

let incorrectAttempts = 0;

const MAX\_INCORRECT\_ATTEMPTS = 3;

let currentGameLevel = 'easy';

// Mức độ hiện tại của ván chơi

socket.on('connect', function() {

    console.log('Connected to server');

    socket.emit('join\_game'); // Gửi yêu cầu tham gia mặc định

    document.getElementById('waiting').style.display = 'block';

    document.getElementById('difficulty-menu').style.display = 'block'; // Hiển thị menu độ khó

});

socket.on('waiting', function(data) {

    document.getElementById('waiting').innerText = data.message;

    document.getElementById('game-area').style.display = 'none'; // Ẩn bảng khi chờ

});

socket.on('match\_found', function(data) {

    console.log('Match found!', data);

    room\_id = data.room\_id;

    board = data.board;

    solution = data.solution;

    document.getElementById('waiting').style.display = 'none';

    document.getElementById('game-area').style.display = 'block';

    document.getElementById('difficulty-menu').style.display = 'none'; // Ẩn menu độ khó

    initializeBoard();

    startTimer();

});

socket.on('board\_updated', function(data) {

    console.log('Board updated:', data);

    board = data.board;

    updateBoardDisplay(data);

});

socket.on('game\_over', function(data) {

    console.log('Game Over:', data);

    stopTimer();

    showModal(data.winner ? `Người thắng là: ${data.winner.name} (Score: ${data.winner.score})` : "Game Over", data.reason);

});

socket.on('opponent\_disconnected', function(data) {

    console.log('Opponent Disconnected:', data);

    stopTimer();

    showModal("Đối thủ đã ngắt kết nối", "Bạn thắng!");

});

function initializeBoard() {

    const gameBoard = document.getElementById('game-board');

    gameBoard.innerHTML = '';

    const table = document.createElement('table');

    for (let i = 0; i < 9; i++) {

        const row = document.createElement('tr');

        for (let j = 0; j < 9; j++) {

            const cell = document.createElement('td');

            const input = document.createElement('input');

            input.type = 'text';

            input.classList.add('cell');

            input.maxLength = 1;

            input.dataset.row = i;

            input.dataset.col = j;

            if (board[i][j] !== 0) {

                input.value = board[i][j];

                input.disabled = true;

            } else {

                input.addEventListener('input', handleInput);

                input.addEventListener('focus', () => highlightRelated(i, j));

                input.addEventListener('blur', clearHighlight);

            }

            cell.appendChild(input);

            row.appendChild(cell);

        }

        table.appendChild(row);

    }

    gameBoard.appendChild(table);

    displayLevelInfo(currentGameLevel);

    // Hiển thị độ khó khi khởi tạo bảng

}

function handleInput(event) {

    const input = event.target;

    const row = parseInt(input.dataset.row);

    const col = parseInt(input.dataset.col);

    let value = input.value ? parseInt(input.value) : 0;

    if (isNaN(value) || value < 0 || value > 9) {

        input.value = '';

        return;

    }

    socket.emit('update\_board', {

        room\_id: room\_id,

        row: row,

        col: col,

        value: value

    });

}

function updateBoardDisplay(data) {

    const input = document.querySelector(`input[data-row="${data.row}"][data-col="${data.col}"]`);

    if (!input) return;

    input.value = data.value !== 0 ? data.value : '';

    if (data.correct) {

        input.classList.remove('error'); // Loại bỏ class 'error' nếu có

    } else {

        input.classList.add('error');

        incorrectAttempts++;

        if (incorrectAttempts >= MAX\_INCORRECT\_ATTEMPTS) {

            showModal("Game Over", "Too many incorrect attempts");

            stopTimer();

        }

    }

}

function clearError(row, col) {

    const input = document.querySelector(`input[data-row="${row}"][data-col="${col}"]`);

    if (input) {

        input.classList.remove('error');

    }

}

function startTimer() {

    startTime = new Date();

    const timerElement = document.querySelector('.timer');

    timerInterval = setInterval(() => {

        const elapsedTime = Math.floor((new Date() - startTime) / 1000);

        timerElement.textContent = formatTime(elapsedTime);

    }, 1000);

}

function stopTimer() {

    clearInterval(timerInterval);

}

function formatTime(time) {

    const minutes = Math.floor(time / 60).toString().padStart(2, '0');

    const seconds = (time % 60).toString().padStart(2, '0');

    return `${minutes}:${seconds}`;

}

function showModal(title, message) {

    document.getElementById('result-title').textContent = title;

    document.getElementById('result-message').textContent = message;

    document.getElementById('result-modal').style.display = 'block';

}

function closeModal() {

    document.getElementById('result-modal').style.display = 'none';

}

function eraseCell() {

    const focusedElement = document.activeElement;

    if (focusedElement.tagName === 'INPUT' && !focusedElement.disabled) {

        const row = parseInt(focusedElement.dataset.row);

        const col = parseInt(focusedElement.dataset.col);

        focusedElement.value = '';

        socket.emit('update\_board', {

            room\_id: room\_id,

            row: row,

            col: col,

            value: 0

        });

        clearError(row, col); // Đảm bảo xóa class 'error'

    }

}

function undo() {

    // Implement undo functionality (if needed)

    alert("Undo is not implemented yet.");

}

function highlightRelated(row, col) {

    clearHighlight(); // Clear previous highlights

    const table = document.querySelector('table');

    const cells = table.querySelectorAll('input.cell');

    const selectedValue = parseInt(cells[row \* 9 + col].value) || 0;

    // Highlight the selected cell

    cells[row \* 9 + col].style.border = '2px solid #007bff';

    cells.forEach((cell, index) => {

        const cellRow = Math.floor(index / 9);

        const cellCol = index % 9;

        const cellValue = parseInt(cell.value) || 0;

        if (cellValue === selectedValue && selectedValue !== 0) {

            cell.classList.add('highlighted');

        }

        if (cellRow === row || cellCol === col) {

            cell.classList.add('highlighted-row-col');

        }

    });

}

function clearHighlight() {

    const cells = document.querySelectorAll('input.cell');

    cells.forEach(cell => {

        cell.classList.remove('highlighted');

        cell.classList.remove('highlighted-row-col');

        cell.style.border = ''; // Reset border

    });

}

function requestNewGame(level) {

    currentGameLevel = level;

    socket.emit('request\_new\_game', {

        level: level

    });

    // Gửi yêu cầu độ khó

    document.getElementById('waiting').style.display = 'block'; // Hiển thị thông báo chờ

    document.getElementById('game-area').style.display = 'none';

    // Ẩn bảng

    document.getElementById('difficulty-menu').style.display = 'none'; // Ẩn menu độ khó

    displayLevelInfo(level);

    // Hiển thị độ khó đã chọn

}

function displayLevelInfo(level) {

    const difficultyElement = document.querySelector('.level');

    const starsElement = document.querySelector('.share');

    // Tạm dùng .share để hiển thị sao

    if (difficultyElement && starsElement) {

        difficultyElement.textContent = level.charAt(0).toUpperCase() + level.slice(1);

        // Viết hoa chữ cái đầu

        let stars = '';

        for (let i = 0; i < (level === 'easy' ? 1 : level === 'medium' ? 2 : 3); i++) {

            stars += ' ★ ';

        }

        starsElement.textContent = stars;

    }

}

# File style.css:

/\* ==================================

    Tổng quan và bố cục trang

    ================================== \*/

    body {

        font-family: Arial, sans-serif;

        display: flex;

        justify-content: center;

        align-items: center;

        min-height: 100vh; /\* Sử dụng min-height để nội dung không bị cắt \*/

        background-color: #f0f8ff;

        margin: 0;

        padding: 0; /\* Thêm padding để có khoảng trắng xung quanh nội dung nếu cần \*/

    }

    .container {

        text-align: center;

        width: 80%; /\* Giữ kích thước tương đối \*/

        max-width: 600px; /\* Nhưng giới hạn chiều rộng tối đa \*/

        margin: auto; /\* Căn giữa ngang \*/

    }

    /\* ==================================

        Header (Tiêu đề, Timer, Level)

        ================================== \*/

    .header {

        display: flex;

        justify-content: space-between;

        align-items: center;

        margin-bottom: 20px;

        width: 100%; /\* Đảm bảo header chiếm hết chiều rộng của container \*/

    }

    .level,

    .timer,

    .share {

        font-size: 1.125rem; /\* 18px / 16px = 1.125rem (sử dụng rem) \*/

    }

    .level span {

        margin-right: 0.625rem; /\* 10px / 16px = 0.625rem (sử dụng rem) \*/

    }

    .share i {

        cursor: pointer;

    }

    /\* ==================================

        Menu (Nút chọn chế độ chơi/độ khó)

        ================================== \*/

    .menu button {

        padding: 0.625rem 1.25rem; /\* 10px / 16px = 0.625rem, 20px / 16px = 1.25rem (sử dụng rem) \*/

        margin: 0.3125rem; /\* 5px / 16px = 0.3125rem (sử dụng rem) \*/

        font-size: 1rem; /\* 16px / 16px = 1rem (sử dụng rem) \*/

        cursor: pointer;

        background-color: #4CAF50;

        color: white;

        border: none;

        border-radius: 0.3125rem; /\* 5px / 16px = 0.3125rem (sử dụng rem) \*/

        transition: background-color 0.3s ease;

    }

    .menu button:hover {

        background-color: #45a049;

    }

    /\* ==================================

        Bảng Sudoku

        ================================== \*/

    table {

        border-collapse: collapse;

        margin: 1.25rem auto; /\* 20px / 16px = 1.25rem (sử dụng rem) \*/

        border: 0.125rem solid #333; /\* 2px / 16px = 0.125rem (sử dụng rem) \*/

    }

    td {

        width: 3.125rem; /\* 50px / 16px = 3.125rem (sử dụng rem) \*/

        height: 3.125rem; /\* 50px / 16px = 3.125rem (sử dụng rem) \*/

        border: 0.0625rem solid #ccc; /\* 1px / 16px = 0.0625rem (sử dụng rem) \*/

        position: relative;

    }

    /\* Quy tắc bổ sung để tạo viền đậm cho các khối 3x3 \*/

    td:nth-child(3n) {

        border-right: 0.125rem solid #333; /\* 2px / 16px = 0.125rem (sử dụng rem) \*/

    }

    td:nth-child(3n+1) {

        border-left: 0.125rem solid #333; /\* 2px / 16px = 0.125rem (sử dụng rem) \*/

    }

    tr:nth-child(3n) td {

        border-bottom: 0.125rem solid #333; /\* 2px / 16px = 0.125rem (sử dụng rem) \*/

    }

    tr:nth-child(3n+1) td {

        border-top: 0.125rem solid #333; /\* 2px / 16px = 0.125rem (sử dụng rem) \*/

    }

    /\* ==================================

        Ô Input

        ================================== \*/

    .cell {

        width: 100%;

        height: 100%;

        text-align: center;

        font-size: 1.25rem; /\* 20px / 16px = 1.25rem (sử dụng rem) \*/

        border: none;

        outline: none;

        background: transparent;

        transition: background-color 0.3s ease, border-color 0.3s ease;

    }

    .cell:disabled {

        color: #000;

        font-weight: bold;

    }

    .cell:focus {

        border: 0.125rem solid #007bff; /\* 2px / 16px = 0.125rem (sử dụng rem) \*/

    }

    /\* ==================================

        Hiệu ứng Highlight

        ================================== \*/

    .highlighted {

        background-color: #fffacd;

    }

    .highlighted-row-col {

        background-color: #f0ffff;

    }

    /\* ==================================

        Hiệu ứng lỗi

        ================================== \*/

    .error {

        background-color: #ffcccc;

    }

    /\* ==================================

        Thanh công cụ (Xóa, Hoàn tác)

        ================================== \*/

    .toolbar {

        margin-top: 1.25rem; /\* 20px / 16px = 1.25rem (sử dụng rem) \*/

    }

    .toolbar button {

        padding: 0.625rem; /\* 10px / 16px = 0.625rem (sử dụng rem) \*/

        margin: 0 0.3125rem; /\* 0 5px (sử dụng rem) \*/

        border: none;

        border-radius: 0.3125rem; /\* 5px / 16px = 0.3125rem (sử dụng rem) \*/

        cursor: pointer;

        background-color: #eee;

        transition: background-color 0.3s ease;

    }

    .toolbar button:hover {

        background-color: #ddd;

    }

    /\* ==================================

        Modal (Thông báo kết quả)

        ================================== \*/

    .modal {

        display: none;

        position: fixed;

        z-index: 1;

        left: 0;

        top: 0;

        width: 100%;

        height: 100%;

        overflow: auto;

        background-color: rgba(0, 0, 0, 0.4);

    }

    .modal-content {

        background-color: #fefefe;

        margin: 15% auto;

        padding: 1.25rem; /\* 20px / 16px = 1.25rem (sử dụng rem) \*/

        border: 0.0625rem solid #888; /\* 1px / 16px = 0.0625rem (sử dụng rem) \*/

        width: 80%;

        max-width: 25rem; /\* 400px / 16px = 25rem (sử dụng rem) \*/

        border-radius: 0.625rem; /\* 10px / 16px = 0.625rem (sử dụng rem) \*/

    }

    .close {

        color: #aaa;

        float: right;

        font-size: 1.75rem; /\* 28px / 16px = 1.75rem (sử dụng rem) \*/

        font-weight: bold;

        cursor: pointer;

    }

    .close:hover,

    .close:focus {

        color: black;

        text-decoration: none;

        cursor: pointer;

    }

    /\* ==================================

        Thông báo chờ đợi (Online)

        ================================== \*/

    #waiting {

        font-size: 1.25rem; /\* 20px / 16px = 1.25rem (sử dụng rem) \*/

        margin-top: 1.25rem; /\* 20px / 16px = 1.25rem (sử dụng rem) \*/

    }

    /\* ==================================

        Hiển thị kết quả (Số lần nhập đúng)

        ================================== \*/

    #result-cells {

        display: flex;

        justify-content: center;

        margin-top: 1.25rem; /\* 20px / 16px = 1.25rem (sử dụng rem) \*/

    }

    .result-cell {

        width: 1.875rem; /\* 30px / 16px = 1.875rem (sử dụng rem) \*/

        height: 1.875rem; /\* 30px / 16px = 1.875rem (sử dụng rem) \*/

        text-align: center;

        line-height: 1.875rem; /\* 30px / 16px = 1.875rem (sử dụng rem) \*/

        margin: 0 0.3125rem; /\* 0 5px (sử dụng rem) \*/

        border: 0.0625rem solid #ccc; /\* 1px / 16px = 0.0625rem (sử dụng rem) \*/

    }

    /\* ==================================

        Responsive Design (Cho di động)

        ================================== \*/

    @media screen and (max-width: 600px) {

        table {

            width: 90%;

        }

        td {

            width: 2.5rem; /\* 40px / 16px = 2.5rem (sử dụng rem) \*/

            height: 2.5rem; /\* 40px / 16px = 2.5rem (sử dụng rem) \*/

        }

        .cell {

            font-size: 1rem; /\* 16px / 16px = 1rem (sử dụng rem) \*/

        }

    }

# File script.js:

let board = [];

let solution = [];

let timerInterval;

let startTime;

let incorrectAttempts = 0;

const MAX\_INCORRECT\_ATTEMPTS = 3;

let currentLevel = 'easy'; // Mức độ mặc định

let history = []; // Mảng lưu trữ các bước đi để hoàn tác

// document.addEventListener('DOMContentLoaded', () => {  // Comment hoặc xóa dòng này

//     startGame('easy'); // Bắt đầu với mức độ dễ khi trang tải xong

// });

function startGame(level) {

    currentLevel = level;

    incorrectAttempts = 0;

    history = []; // Reset lịch sử khi bắt đầu game mới

    document.getElementById('difficulty-menu').style.display = 'none'; // Ẩn menu độ khó

    document.getElementById('game-board').innerHTML = ''; // Xóa bảng cũ

    fetch('/generate\_sudoku?level=' + level) // Gọi API để tạo bảng

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

        .then(data => {

            board = data.board;

            solution = data.solution;

            initializeBoard(); // Khởi tạo bảng trên trang

            startTimer(); // Bắt đầu đếm giờ

            displayLevelInfo(level); // Hiển thị thông tin độ khó

        });

}

function startGame(level) {

    currentLevel = level;

    incorrectAttempts = 0;

    history = []; // Reset lịch sử khi bắt đầu game mới

    document.getElementById('difficulty-menu').style.display = 'none'; // Ẩn menu độ khó

    document.getElementById('game-board').innerHTML = ''; // Xóa bảng cũ

    fetch('/generate\_sudoku?level=' + level) // Gọi API để tạo bảng

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

        .then(data => {

            board = data.board;

            solution = data.solution;

            initializeBoard(); // Khởi tạo bảng trên trang

            startTimer(); // Bắt đầu đếm giờ

            displayLevelInfo(level); // Hiển thị thông tin độ khó

        });

}

function initializeBoard() {

    const gameBoard = document.getElementById('game-board');

    const table = document.createElement('table');

    for (let i = 0; i < 9; i++) {

        const row = document.createElement('tr');

        for (let j = 0; j < 9; j++) {

            const cell = document.createElement('td');

            const input = document.createElement('input');

            input.type = 'text';

            input.classList.add('cell');

            input.maxLength = 1; // Chỉ cho phép nhập 1 ký tự

            input.dataset.row = i; // Lưu chỉ số hàng

            input.dataset.col = j; // Lưu chỉ số cột

            if (board[i][j] !== 0) {

                input.value = board[i][j];

                input.disabled = true; // Ô số ban đầu không cho sửa

            } else {

                input.addEventListener('input', handleInput); // Xử lý khi nhập

                input.addEventListener('focus', () => highlightRelated(i, j)); // Highlight khi chọn ô

                input.addEventListener('blur', clearHighlight); // Xóa highlight khi bỏ chọn ô

            }

            cell.appendChild(input);

            row.appendChild(cell);

        }

        table.appendChild(row);

    }

    gameBoard.appendChild(table);

}

function handleInput(event) {

    const input = event.target;

    const row = parseInt(input.dataset.row);

    const col = parseInt(input.dataset.col);

    let value = input.value ? parseInt(input.value) : 0; // Chuyển đổi sang số

    // Kiểm tra đầu vào

    if (isNaN(value) || value < 0 || value > 9) {

        input.value = ''; // Xóa giá trị không hợp lệ

        return;

    }

    // Lưu bước đi vào lịch sử

    history.push({ row, col, prevValue: board[row][col], newValue: value });

    board[row][col] = value; // Cập nhật bảng

    if (solution[row][col] === value) {

        input.classList.remove('error'); // Xóa hiệu ứng lỗi

        if (checkWinCondition()) {

            stopTimer();

            showModal('Chúc mừng!', `Bạn đã hoàn thành trong ${document.querySelector('.timer').textContent}`);

        }

    } else {

        input.classList.add('error'); // Thêm hiệu ứng lỗi

        incorrectAttempts++;

        if (incorrectAttempts >= MAX\_INCORRECT\_ATTEMPTS) {

            stopTimer();

            showModal('Game Over', 'Bạn đã nhập sai quá 3 lần.');

        }

    }

}

function checkWinCondition() {

    for (let i = 0; i < 9; i++) {

        for (let j = 0; j < 9; j++) {

            if (board[i][j] !== solution[i][j]) {

                return false; // Chưa hoàn thành

            }

        }

    }

    return true; // Đã hoàn thành

}

function startTimer() {

    startTime = new Date();

    const timerElement = document.querySelector('.timer');

    timerInterval = setInterval(() => {

        const elapsedTime = Math.floor((new Date() - startTime) / 1000);

        timerElement.textContent = formatTime(elapsedTime);

    }, 1000);

}

function stopTimer() {

    clearInterval(timerInterval);

}

function formatTime(time) {

    const minutes = Math.floor(time / 60).toString().padStart(2, '0');

    const seconds = (time % 60).toString().padStart(2, '0');

    return `${minutes}:${seconds}`;

}

function showModal(title, message) {

    document.getElementById('result-title').textContent = title;

    document.getElementById('result-message').textContent = message;

    document.getElementById('result-modal').style.display = 'block';

}

function closeModal() {

    document.getElementById('result-modal').style.display = 'none';

}

function eraseCell() {

    const focusedElement = document.activeElement;

    if (focusedElement.tagName === 'INPUT' && !focusedElement.disabled) {

        const row = parseInt(focusedElement.dataset.row);

        const col = parseInt(focusedElement.dataset.col);

        const prevValue = board[row][col]; // Lưu giá trị trước khi xóa

        focusedElement.value = '';

        board[row][col] = 0;

        history.push({ row, col, prevValue: prevValue, newValue: 0 }); // Lưu vào lịch sử

        focusedElement.classList.remove('error'); // Xóa hiệu ứng lỗi

    }

}

function undo() {

    if (history.length > 0) {

        const lastMove = history.pop();

        const input = document.querySelector(`input[data-row="${lastMove.row}"][data-col="${lastMove.col}"]`);

        if (input) {

            input.value = lastMove.prevValue !== 0 ? lastMove.prevValue : '';

            board[lastMove.row][lastMove.col] = lastMove.prevValue;

            input.classList.remove('error'); // Xóa hiệu ứng lỗi

        }

    } else {

        alert('Không còn bước nào để hoàn tác.');

    }

}

function highlightRelated(row, col) {

    clearHighlight(); // Xóa highlight cũ

    const table = document.querySelector('table');

    const cells = table.querySelectorAll('input.cell');

    const selectedValue = parseInt(cells[row \* 9 + col].value) || 0;

    // Highlight ô được chọn

    cells[row \* 9 + col].style.border = '2px solid #007bff';

    cells.forEach((cell, index) => {

        const cellRow = Math.floor(index / 9);

        const cellCol = index % 9;

        const cellValue = parseInt(cell.value) || 0;

        if (cellValue === selectedValue && selectedValue !== 0) {

            cell.classList.add('highlighted'); // Highlight ô cùng số

        }

        if (cellRow === row || cellCol === col) {

            cell.classList.add('highlighted-row-col'); // Highlight hàng và cột

        }

    });

}

function clearHighlight() {

    const cells = document.querySelectorAll('input.cell');

    cells.forEach(cell => {

        cell.classList.remove('highlighted');

        cell.classList.remove('highlighted-row-col');

        cell.style.border = ''; // Reset viền

    });

}

function displayLevelInfo(level) {

    const difficultyElement = document.getElementById('difficulty');

    const starsElement = document.getElementById('stars');

    difficultyElement.textContent = level.charAt(0).toUpperCase() + level.slice(1); // Viết hoa chữ cái đầu

    let stars = '';

    for (let i = 0; i < (level === 'easy' ? 1 : level === 'medium' ? 2 : 3); i++) {

        stars += '★';

    }

    starsElement.textContent = stars;

}

# File static.css:

/\* ==================================

    Tổng quan và bố cục trang

    ================================== \*/

    body {

        font-family: Arial, sans-serif;

        display: flex;

        justify-content: center;

        align-items: center;

        height: 100vh;

        background-color: #d3e3f5; /\* Changed to match style.css \*/

        margin: 0;

    }

    .container {

        text-align: center;

    }

    /\* ==================================

        Header (Tiêu đề, Timer, Level)

        ================================== \*/

    .header {

        display: flex;

        justify-content: space-between;

        align-items: center;

        margin-bottom: 20px;

    }

    .level,

    .timer,

    .share {

        font-size: 18px;

    }

    .level span {

        margin-right: 10px;

    }

    .share i {

        cursor: pointer;

    }

    /\* ==================================

        Menu (Nút chọn chế độ chơi/độ khó)

        ================================== \*/

    .menu button {

        padding: 10px 20px; /\* Changed to match style.css \*/

        margin: 5px;

        font-size: 16px;

        cursor: pointer;

        background-color: #4CAF50;

        color: white;

        border: none;

        border-radius: 5px;

    }

    .menu button:hover {

        background-color: #45a049;

    }

    /\* ==================================

        Bảng Sudoku

        ================================== \*/

    table {

        border-collapse: collapse;

        margin: 20px auto;

        border: 2px solid #000;

    }

    td {

        width: 50px;

        height: 50px;

        border: 1px solid #ccc;

        position: relative;

    }

    /\* Viền đậm cho khối 3x3 \*/

    td:nth-child(3n) {

        border-right: 2px solid #000;

    }

    td:nth-child(3n+1) {

        border-left: 2px solid #000;

    }

    tr:nth-child(3n) td {

        border-bottom: 2px solid #000;

    }

    tr:nth-child(3n+1) td {

        border-top: 2px solid #000;

    }

    /\* ==================================

        Ô Input

        ================================== \*/

    .cell {

        width: 100%;

        height: 100%;

        text-align: center;

        font-size: 20px;

        border: none;

        outline: none;

        background: transparent;

    }

    .cell:disabled {

        color: #000;

        font-weight: bold;

    }

    .cell.user-filled {

        color: #1e90ff;

    }

    .cell.selected {

        border: 2px solid #000;

    }

    /\* ==================================

        Thanh công cụ (Xóa, Hoàn tác)

        ================================== \*/

    .toolbar {

        display: flex;

        justify-content: center;

        margin-top: 20px;

    }

    .toolbar button {

        background-color: #f0f0f0;

        border: 1px solid #ccc;

        padding: 10px;

        margin: 0 5px;

        cursor: pointer;

        border-radius: 5px;

    }

    .toolbar button:hover {

        background-color: #e0e0e0;

    }

    .toolbar i {

        font-size: 18px;

    }

    /\* ==================================

        Modal (Thông báo kết quả)

        ================================== \*/

    .modal {

        position: fixed;

        top: 0;

        left: 0;

        width: 100%;

        height: 100%;

        background-color: rgba(0, 0, 0, 0.5);

        display: flex;

        justify-content: center;

        align-items: center;

    }

    .modal-content {

        background-color: white;

        padding: 20px;

        border-radius: 10px;

        text-align: center;

        box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);

    }

    .modal-content h2 {

        margin: 0 0 10px;

    }

    .modal-content p {

        margin: 0 0 20px;

    }

    .modal-content button {

        padding: 10px 20px;

        margin: 5px;

        font-size: 16px;

        cursor: pointer;

        background-color: #4CAF50;

        color: white;

        border: none;

        border-radius: 5px;

    }

    .modal-content button:hover {

        background-color: #45a049;

    }

    /\* ==================================

        Responsive Design (Nếu cần)

        ================================== \*/

    /\* Bạn có thể thêm các quy tắc responsive ở đây nếu cần thiết  \*/

# File main.py:

import eventlet

eventlet.monkey\_patch()  # Phải gọi monkey\_patch() đầu tiên

from flask import Flask, render\_template, jsonify, request

from flask\_socketio import SocketIO, join\_room, leave\_room, rooms, emit

import random

import uuid

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

app.config['SECRET\_KEY'] = 'your-secret-key'  # Đảm bảo bạn có một secret key an toàn

socketio = SocketIO(app)  # Loại bỏ cors\_allowed\_origins (xem giải thích bên dưới)

# Sử dụng Flask-CORS nếu cần thiết (xem giải thích bên dưới)

# from flask\_cors import CORS

# CORS(app)

rooms\_data = {}  # Quản lý thông tin phòng chi tiết

def is\_valid(board, row, col, num):

    """Kiểm tra tính hợp lệ của số trong bảng Sudoku."""

    for x in range(9):

        if board[row][x] == num and x != col:

            return False

    for x in range(9):

        if board[x][col] == num and x != row:

            return False

    start\_row, start\_col = 3 \* (row // 3), 3 \* (col // 3)

    for i in range(3):

        for j in range(3):

            if board[i + start\_row][j + start\_col] == num and (i + start\_row, j + start\_col) != (row, col):

                return False

    return True

def solve\_sudoku(board):

    """Giải bảng Sudoku (đệ quy, quay lui)."""

    for row in range(9):

        for col in range(9):

            if board[row][col] == 0:

                for num in range(1, 10):

                    if is\_valid(board, row, col, num):

                        board[row][col] = num

                        if solve\_sudoku(board):

                            return True

                        board[row][col] = 0

                return False

    return True

def generate\_full\_board():

    """Tạo một bảng Sudoku hoàn chỉnh."""

    board = [[0 for \_ in range(9)] for \_ in range(9)]

    solve\_sudoku(board)

    return board

def generate\_sudoku(level):

    """Tạo bảng Sudoku với độ khó khác nhau."""

    board = generate\_full\_board()

    solution\_board = [row[:] for row in board]

    game\_board = [row[:] for row in board]

    cells\_to\_remove = {'easy': 40, 'medium': 50, 'hard': 60}.get(level, 50)

    while cells\_to\_remove > 0:

        row, col = random.randint(0, 8), random.randint(0, 8)

        if game\_board[row][col] != 0:

            game\_board[row][col] = 0

            cells\_to\_remove -= 1

    return game\_board, solution\_board

@app.route('/')

def index():

    return render\_template('index.html')

@app.route('/practice')

def practice():

    return render\_template('practice.html')

@app.route('/online')

def online():

    return render\_template('online.html')

@app.route('/generate\_sudoku')

def generate\_sudoku\_route():

    """API để tạo bảng Sudoku (cho chế độ offline)."""

    level = request.args.get('level', 'medium')

    game\_board, solution\_board = generate\_sudoku(level)

    return jsonify({'board': game\_board, 'solution': solution\_board})

@socketio.on('connect')

def handle\_connect():

    print('Client connected')

@socketio.on('join\_game')

def handle\_join\_game():

    """Xử lý khi người chơi tham gia vào hàng đợi hoặc phòng."""

    player\_id = request.sid

    print(f"Player {player\_id} joined")

    available\_room\_id = next(

        (room\_id for room\_id, room\_data in rooms\_data.items() if len(room\_data['players']) < 2),

        None

    )

    if available\_room\_id:

        rooms\_data[available\_room\_id]['players'].append(player\_id)

        rooms\_data[available\_room\_id]['scores'][player\_id] = 0

        join\_room(available\_room\_id, sid=player\_id)

        game\_board = rooms\_data[available\_room\_id]['board']

        solution\_board = rooms\_data[available\_room\_id]['solution']

        emit('match\_found', {

            'room\_id': available\_room\_id,

            'board': game\_board,

            'solution': solution\_board

        }, room=available\_room\_id)

    else:

        room\_id = str(uuid.uuid4())  # Tạo ID phòng duy nhất

        game\_board, solution\_board = generate\_sudoku('medium')  # Tạo bảng và lời giải

        rooms\_data[room\_id] = {

            'players': [player\_id],

            'board': game\_board,

            'solution': solution\_board,  # Thêm solution vào đây

            'level': 'medium',  # Mặc định là medium

            'scores': {player\_id: 0}

        }

        join\_room(room\_id, sid=player\_id)

        emit('waiting', {'message': 'Đang chờ người chơi khác...'}, room=room\_id)

@socketio.on('rejoin\_game')

def handle\_rejoin\_game(data):

    """Xử lý khi người chơi cố gắng kết nối lại vào phòng."""

    room\_id = data.get('room\_id')

    player\_id = request.sid

    if room\_id in rooms\_data:

        room = rooms\_data[room\_id]

        if player\_id not in room['players']:

            room['players'].append(player\_id)

            room['scores'][player\_id] = 0  # Hoặc khôi phục điểm số nếu cần

            join\_room(room\_id, sid=player\_id)

            emit('match\_found', {

                'room\_id': room\_id,

                'board': room['board'],

                'solution': room['solution']

            }, room=player\_id)

            emit('message', {'text': f'Người chơi {player\_id} đã quay lại phòng.'}, room=room\_id, skip\_sid=player\_id)

        else:

            # Người chơi đã ở trong phòng (trường hợp không nên xảy ra nếu logic client đúng)

            join\_room(room\_id, sid=player\_id)

            emit('match\_found', {

                'room\_id': room\_id,

                'board': room['board'],

                'solution': room['solution']

            }, room=player\_id)

    else:

        emit('error', {'message': 'Phòng không tồn tại hoặc đã đóng.'}, room=player\_id)

@socketio.on('update\_board')

def handle\_update\_board(data):

    """Xử lý khi người chơi cập nhật bảng."""

    room\_id = data['room\_id']

    row = data['row']

    col = data['col']

    value = data['value']

    player\_id = request.sid

    if room\_id in rooms\_data:

        rooms\_data[room\_id]['board'][row][col] = value

        correct = (value != 0 and value == rooms\_data[room\_id]['solution'][row][col])

        if correct:

            rooms\_data[room\_id]['scores'][player\_id] += 1

        emit('board\_updated', {

            'row': row,

            'col': col,

            'value': value,

            'correct': correct

        }, room=room\_id)

        check\_game\_over(room\_id)

def check\_game\_over(room\_id):

    """Kiểm tra xem trò chơi đã kết thúc chưa."""

    if room\_id not in rooms\_data:

        return

    board = rooms\_data[room\_id]['board']

    solution = rooms\_data[room\_id]['solution']

    for i in range(9):

        for j in range(9):

            if board[i][j] != solution[i][j] and board[i][j] != 0:

                return  # Chưa kết thúc

    # Xác định người thắng

    scores = rooms\_data[room\_id]['scores']

    winner\_id = max(scores, key=scores.get)

    winner\_score = scores[winner\_id]

    winner\_name = f"Player {rooms\_data[room\_id]['players'].index(winner\_id) + 1}"

    emit('game\_over', {

        'winner': {

            'name': winner\_name,

            'score': winner\_score

        }

    }, room=room\_id)

@socketio.on('request\_new\_game')

def handle\_request\_new\_game(data):

    """Xử lý khi người chơi yêu cầu ván chơi mới với độ khó."""

    room\_id = data['room\_id']

    level = data['level']

    if room\_id in rooms\_data:

        rooms\_data[room\_id]['level'] = level

        game\_board, solution\_board = generate\_sudoku(level)

        rooms\_data[room\_id]['board'] = game\_board

        rooms\_data[room\_id]['solution'] = solution\_board

        # Reset điểm số

        for player in rooms\_data[room\_id]['players']:

            rooms\_data[room\_id]['scores'][player] = 0

        emit('match\_found', {

            'room\_id': room\_id,

            'board': game\_board,

            'solution': solution\_board

        }, room=room\_id)

@socketio.on('disconnect')

def handle\_disconnect():

    """Xử lý khi người chơi ngắt kết nối."""

    player\_id = request.sid

    for room\_id, room in list(rooms\_data.items()):

        if player\_id in room['players']:

            room['players'].remove(player\_id)

            emit('opponent\_disconnected', {'message': 'Đối thủ đã ngắt kết nối.'}, room=room\_id, skip\_sid=player\_id)

            del room['scores'][player\_id]

            if len(room['players']) < 2:

                emit('waiting', {'message': 'Đang chờ người chơi khác...'}, room=room\_id)

            if len(room['players']) == 0:

                del rooms\_data[room\_id]

            break

if \_\_name\_\_ == '\_\_main\_\_':

    import eventlet.wsgi

    eventlet.wsgi.server(eventlet.listen(('', 5000)), app)  # Chạy với eventlet

# File online.html:

<!DOCTYPE html>

 <html lang="en">

 <head>

  <meta charset="UTF-8">

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

  <title>Sudoku Online</title>

  <link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}">

  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css">

  <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.1/socket.io.js"></script>

  <script src="{{ url\_for('static', filename='js/online.js') }}"></script>

 </head>

 <body>

  <div class="container">

   <div class="header">

    <h1>Sudoku Online</h1>

    <div class="level">

     Độ khó: <span id="difficulty"></span>

     <span id="stars"></span>

    </div>

    <div class="timer">00:00</div>

   </div>

   <div id="difficulty-menu" class="menu">

    <button onclick="requestNewGame('easy')">Dễ</button>

    <button onclick="requestNewGame('medium')">Trung bình</button>

    <button onclick="requestNewGame('hard')">Khó</button>

   </div>

   <div id="waiting">Đang kết nối...</div>

   <div id="game-area">

    <table id="game-board"></table>

    <div class="toolbar">

     <button onclick="eraseCell()"><i class="fas fa-eraser"></i></button>

     <button onclick="undo()">Hoàn tác</button>

    </div>

   </div>

   <div id="result-modal" class="modal">

    <div class="modal-content">

     <span class="close" onclick="closeModal()">&times;</span>

     <h2 id="result-title"></h2>

     <p id="result-message"></p>

     <button onclick="closeModal()">Chơi lại</button>

    </div>

   </div>

  </div>

 </body>

 </html>

# File practice.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Sudoku Practice</title>

    <link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css">

    <script src="{{ url\_for('static', filename='js/script.js') }}"></script>

</head>

<body>

    <div class="container">

        <div class="header">

            <h1>Sudoku</h1>

            <div class="level">

                Độ khó: <span id="difficulty"></span>

                <span id="stars"></span>

            </div>

            <div class="timer">00:00</div>

        </div>

        <div id="difficulty-menu" class="menu">

            <button onclick="startGame('easy')">Dễ</button>

            <button onclick="startGame('medium')">Trung bình</button>

            <button onclick="startGame('hard')">Khó</button>

        </div>

        <table id="game-board"></table>

        <div class="toolbar">

            <button onclick="eraseCell()"><i class="fas fa-eraser"></i></button>

            <button onclick="undo()">Hoàn tác</button>

        </div>

        <div id="result-modal" class="modal">

            <div class="modal-content">

                <span class="close" onclick="closeModal()">&times;</span>

                <h2 id="result-title"></h2>

                <p id="result-message"></p>

                <button onclick="closeModal()">Chơi lại</button>

            </div>

        </div>

    </div>

</body>

</html>

# File index.html

<!DOCTYPE html>

 <html lang="en">

 <head>

  <meta charset="UTF-8">

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

  <title>Sudoku</title>

  <link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}">

 </head>

 <body>

  <div class="container">

   <h1>Sudoku</h1>

   <div class="menu">

    <button onclick="window.location.href='/practice'">Chơi Offline</button>

    <button onclick="window.location.href='/online'">Chơi Online</button>

   </div>

  </div>

 </body>

 </html>