File app.py:

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

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

import random, uuid, threading, time

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

app.config['SECRET\_KEY'] = 'your-secret-key'

socketio = SocketIO(app, cors\_allowed\_origins="\*")

waiting\_players = []

rooms = {}

TURN\_TIME\_LIMIT = 30  # Giới hạn thời gian mỗi lượt (giây)

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

    for x in range(9):

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

            return False

        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 != row or j + start\_col != col):

                return False

    return True

def solve\_sudoku(board):

    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():

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

    solve\_sudoku(board)

    return board

def generate\_sudoku(level):

    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():

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

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

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

@socketio.on('join\_game')

def handle\_join\_game(data):

    player\_id = request.sid

    level = data.get('level', 'medium')

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

    if len(waiting\_players) == 0:

        waiting\_players.append((player\_id, level, game\_board, solution\_board))

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

    else:

        opponent\_id, \_, game\_board, solution\_board = waiting\_players.pop(0)

        room\_id = str(uuid.uuid4())

        rooms[room\_id] = {

            'players': [player\_id, opponent\_id],

            'scores': {player\_id: 0, opponent\_id: 0},

            'board': game\_board,

            'solution': solution\_board,

            'turn': 0,

            'timer': None

        }

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

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

        emit('match\_found', {'room\_id': room\_id, 'board': game\_board, 'turn': 0}, room=room\_id)

        start\_turn\_timer(room\_id)

def start\_turn\_timer(room\_id):

    def timer\_func():

        time.sleep(TURN\_TIME\_LIMIT)

        if room\_id in rooms:

            current\_turn = rooms[room\_id]['turn']

            rooms[room\_id]['turn'] = (current\_turn + 1) % 2

            socketio.emit('turn\_timeout', {'turn': rooms[room\_id]['turn']}, room=room\_id)

            start\_turn\_timer(room\_id)

    t = threading.Thread(target=timer\_func)

    t.daemon = True

    rooms[room\_id]['timer'] = t

    t.start()

@socketio.on('update\_board')

def handle\_update\_board(data):

    room\_id, row, col, value = data['room\_id'], data['row'], data['col'], data['value']

    player\_id = request.sid

    if room\_id in rooms:

        if isinstance(value, int) and 0 <= value <= 9 and is\_valid(rooms[room\_id]['board'], row, col, value):

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

            emit('board\_updated', {'board': rooms[room\_id]['board']}, room=room\_id)

@socketio.on('end\_turn')

def handle\_end\_turn(data):

    room\_id = data['room\_id']

    if room\_id in rooms:

        rooms[room\_id]['turn'] = (rooms[room\_id]['turn'] + 1) % 2

        emit('turn\_changed', {'turn': rooms[room\_id]['turn']}, room=room\_id)

        start\_turn\_timer(room\_id)

@socketio.on('game\_completed')

def handle\_game\_completed(data):

    room\_id = data['room\_id']

    player\_id = request.sid

    if room\_id in rooms:

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

        scores[player\_id] += 1

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

        emit('game\_over', {'winner': winner\_id, 'score': scores[winner\_id]}, room=room\_id)

@socketio.on('disconnect')

def handle\_disconnect():

    player\_id = request.sid

    for i, (pid, \*\_rest) in enumerate(waiting\_players):

        if pid == player\_id:

            del waiting\_players[i]

            break

    for room\_id, room in list(rooms.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)

            if not room['players']:

                del rooms[room\_id]

            break

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

    socketio.run(app, host='0.0.0.0', port=5000, debug=True)

file script.js:

let currentLevel = 'medium';

let selectedCell = null;

let history = [];

let startTime = null;

let timerInterval = null;

let solutionBoard = null; // Lưu bảng giải pháp

let playerBoard = []; // Khai báo bảng người chơi

function startGame(level) {

    currentLevel = level;

    document.getElementById('difficulty').textContent = {

        'easy': 'Dễ',

        'medium': 'Trung bình',

        'hard': 'Khó'

    }[level];

    document.getElementById('stars').textContent = {

        'easy': '★',

        'medium': '★★',

        'hard': '★★★'

    }[level];

    document.getElementById('difficulty-menu').style.display = 'none';

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

    document.getElementById('toolbar').style.display = 'flex';

    startTime = Date.now();

    clearInterval(timerInterval);

    timerInterval = setInterval(updateTimer, 1000);

    let board = '<table>';

    for (let i = 0; i < 9; i++) {

        board += '<tr>';

        for (let j = 0; j < 9; j++) {

            // Thêm data-row và data-col để biết vị trí ô

            board += `<td><input type="text" maxlength="1" class="cell" data-row="${i}" data-col="${j}" oninput="handleInput(this)" onfocus="selectCell(this)"></td>`;

        }

        board += '</tr>';

    }

    board += '</table>';

    document.getElementById('game-board').innerHTML = board;

    fetch(`/generate\_sudoku?level=${level}`)

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

        .then(data => {

            fillBoard(data.board);

            solutionBoard = data.solution; // Lưu bảng giải pháp

        });

}

function fillBoard(board) {

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

    cells.forEach((cell, index) => {

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

        const col = index % 9;

        const value = board[row][col];

        cell.value = value !== 0 ? value : '';

        cell.dataset.row = row; // Thêm data-row

        cell.dataset.col = col; // Thêm data-col

        if (value !== 0) {

            cell.disabled = true;

        } else {

            cell.classList.add('user-filled');

        }

        playerBoard[row][col] = value; // Khởi tạo playerBoard

    });

}

function selectCell(cell) {

    if (selectedCell) {

        selectedCell.classList.remove('selected');

    }

    selectedCell = cell;

    cell.classList.add('selected');

}

function isValid(board, row, col, num) {

    for (let x = 0; x < 9; x++) {

        if (board[row][x] == num && x != col) {

            return false;

        }

    }

    for (let x = 0; x < 9; x++) {

        if (board[x][col] == num && x != row) {

            return false;

        }

    }

    let startRow = 3 \* Math.floor(row / 3);

    let startCol = 3 \* Math.floor(col / 3);

    for (let i = 0; i < 3; i++) {

        for (let j = 0; j < 3; j++) {

            if (board[i + startRow][j + startCol] == num && (i + startRow != row) && (j + startCol != col)) {

                return false;

            }

        }

    }

    return true;

}

function handleInput(cell) {

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

    if (value && (value < 1 || value > 9)) {

        alert("Please enter a number between 1 and 9.");

        cell.value = '';

        return;

    }

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

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

    // Kiểm tra tính hợp lệ trước khi cập nhật bảng

    if (!isValid(playerBoard, row, col, value)) {

        cell.classList.add('invalid'); // Thêm class 'invalid' để tô màu ô

    } else {

        cell.classList.remove('invalid'); // Xóa class 'invalid' nếu có

        playerBoard[row][col] = value; // Cập nhật bảng người chơi

    }

    history.push({ cell, oldValue: cell.dataset.oldValue || '', newValue: value });

    cell.dataset.oldValue = value;

    cell.classList.add('user-filled');

    checkBoard(); // Kiểm tra bảng sau mỗi lần nhập

}

function updateTimer() {

    const elapsed = Math.floor((Date.now() - startTime) / 1000);

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

    const seconds = String(elapsed % 60).padStart(2, '0');

    document.querySelector('.timer').textContent = `${minutes}:${seconds}`;

}

function eraseCell() {

    if (selectedCell && !selectedCell.disabled) {

        history.push({ cell: selectedCell, oldValue: selectedCell.value, newValue: '' });

        selectedCell.value = '';

        selectedCell.dataset.oldValue = '';

    }

}

function undo() {

    if (history.length === 0) return;

    const lastAction = history.pop();

    lastAction.cell.value = lastAction.oldValue;

    lastAction.cell.dataset.oldValue = lastAction.oldValue;

}

function checkBoard() {

    let isComplete = true;

    // Tạo bảng người chơi từ các ô

    for (let i = 0; i < 9; i++) {

        for (let j = 0; j < 9; j++) {

            if (playerBoard[i][j] === 0) {

                isComplete = false;

                break;

            }

        }

        if (!isComplete) break;

    }

    // Nếu chưa điền hết, không kiểm tra

    if (!isComplete) return;

    // So sánh với bảng giải pháp

    let isCorrect = true;

    for (let i = 0; i < 9; i++) {

        for (let j = 0; j < 9; j++) {

            if (playerBoard[i][j] !== solutionBoard[i][j]) {

                isCorrect = false;

                break;

            }

        }

        if (!isCorrect) break;

    }

    // Hiển thị kết quả

    showResult(isCorrect);

}

function showResult(isCorrect) {

    clearInterval(timerInterval); // Dừng timer

    const modal = document.getElementById('result-modal');

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

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

    if (isCorrect) {

        title.textContent = 'Chúc mừng!';

        message.textContent = `Bạn đã hoàn thành bảng Sudoku ${currentLevel} trong ${document.querySelector('.timer').textContent}!`;

    } else {

        title.textContent = 'Sai rồi!';

        message.textContent = 'Có một số ô không đúng. Hãy thử lại nhé!';

    }

    modal.style.display = 'flex';

}

function closeModal() {

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

}

File online.js:

// online.js

const socket = io('http://127.0.0.1:5000/online');

let selectedCell = null;

let playerBoard = Array.from({ length: 9 }, () => Array(9).fill(0));

let solutionBoard = null;

let roomId = null;

let isPlayerTurn = false;

let turnTimer = null;

let turnTimeLeft = 30;

// ====== SOCKET EVENTS ======

socket.on('connect', () => {

  console.log('Connected to server');

  socket.emit('join\_game');

});

socket.on('waiting', () => {

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

});

socket.on('match\_found', ({ room\_id, board, solution, turn }) => {

  roomId = room\_id;

  solutionBoard = solution;

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

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

  renderBoard(board);

  updateTurn(turn);

});

socket.on('update\_turn', (turn) => {

  updateTurn(turn);

});

socket.on('board\_updated', ({ board }) => {

  renderBoard(board);

});

socket.on('game\_over', ({ winner }) => {

  stopTurnTimer();

  alert(`Game Over! ${winner} wins!`);

});

// ====== GAME LOGIC ======

function renderBoard(board) {

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

  let html = '<table>';

  for (let i = 0; i < 9; i++) {

    html += '<tr>';

    for (let j = 0; j < 9; j++) {

      const value = board[i][j] !== 0 ? board[i][j] : '';

      const disabled = value ? 'disabled' : '';

      html += `<td><input type="text" maxlength="1" class="cell" data-row="${i}" data-col="${j}" value="${value}" ${disabled}></td>`;

      playerBoard[i][j] = board[i][j];

    }

    html += '</tr>';

  }

  html += '</table>';

  container.innerHTML = html;

  document.querySelectorAll('.cell').forEach(cell => {

    cell.addEventListener('input', onCellInput);

    cell.addEventListener('focus', () => selectCell(cell));

  });

}

function selectCell(cell) {

  if (!isPlayerTurn || cell.disabled) return;

  if (selectedCell) selectedCell.classList.remove('selected');

  selectedCell = cell;

  cell.classList.add('selected');

}

function onCellInput(e) {

  const cell = e.target;

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

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

  const value = parseInt(cell.value);

  if (!isPlayerTurn || isNaN(value) || value < 1 || value > 9) {

    cell.value = '';

    return;

  }

  if (!isValid(playerBoard, row, col, value)) {

    cell.classList.add('invalid');

    setTimeout(() => cell.classList.remove('invalid'), 1000);

    cell.value = '';

    return;

  }

  playerBoard[row][col] = value;

  socket.emit('update\_board', { room\_id: roomId, row, col, value });

  stopTurnTimer();

}

function isValid(board, row, col, num) {

  for (let i = 0; i < 9; i++) {

    if (board[row][i] === num || board[i][col] === num) return false;

  }

  const startRow = Math.floor(row / 3) \* 3;

  const startCol = Math.floor(col / 3) \* 3;

  for (let i = 0; i < 3; i++) {

    for (let j = 0; j < 3; j++) {

      if (board[startRow + i][startCol + j] === num) return false;

    }

  }

  return true;

}

function updateTurn(turn) {

  isPlayerTurn = turn === socket.id;

  document.getElementById('turn-indicator').textContent = isPlayerTurn ? 'Your Turn' : 'Opponent Turn';

  if (isPlayerTurn) {

    startTurnTimer();

  } else {

    stopTurnTimer();

  }

}

function startTurnTimer() {

  turnTimeLeft = 30;

  document.getElementById('turn-timer').textContent = `Time left: ${turnTimeLeft}s`;

  stopTurnTimer();

  turnTimer = setInterval(() => {

    turnTimeLeft--;

    document.getElementById('turn-timer').textContent = `Time left: ${turnTimeLeft}s`;

    if (turnTimeLeft <= 0) {

      stopTurnTimer();

      socket.emit('turn\_timeout', { room\_id: roomId });

    }

  }, 1000);

}

function stopTurnTimer() {

  if (turnTimer) clearInterval(turnTimer);

}

File index.html:

<!DOCTYPE html>

<html lang="vi">

<head>

    <meta charset="UTF-8">

    <title>Sudoku Online</title>

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

</head>

<body>

    <div class="container">

        <h1>Chào mừng đến với Sudoku Online</h1>

        <div class="menu">

            <button onclick="window.location.href='/practice'">Luyện tập</button>

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

        </div>

    </div>

</body>

</html>

File online.html:

<!DOCTYPE html>

<html lang="vi">

<head>

    <meta charset="UTF-8">

    <title>Sudoku Online</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f0f0f0;

            padding: 20px;

            text-align: center;

        }

        table {

            border-collapse: collapse;

            margin: auto;

        }

        td {

            border: 1px solid #999;

            width: 40px;

            height: 40px;

        }

        .cell {

            width: 100%;

            height: 100%;

            text-align: center;

            font-size: 18px;

        }

        .cell:disabled {

            background-color: #eee;

        }

        .selected {

            border: 2px solid blue;

        }

        .invalid {

            background-color: #f88;

        }

        .user-filled {

            background-color: #fffdd0;

        }

        .timer {

            font-size: 20px;

            margin: 10px;

        }

        #result-modal {

            display: none;

            position: fixed;

            top: 0; left: 0;

            width: 100%; height: 100%;

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

            justify-content: center;

            align-items: center;

        }

        #result-content {

            background: white;

            padding: 20px;

            border-radius: 10px;

            width: 300px;

        }

    </style>

</head>

<body>

    <h1>Chế độ Chơi Online</h1>

    <div class="timer">00:00</div>

    <div id="waiting">

        <p>Đang tìm đối thủ...</p>

    </div>

    <div id="game-area" style="display:none;">

        <div id="game-board"></div>

        <br>

        <button onclick="eraseCell()">Xoá ô</button>

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

        <button onclick="checkBoard()">Kiểm tra</button>

        <div id="result-cells" style="margin-top: 10px;"></div>

    </div>

    <div id="result-modal">

        <div id="result-content">

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

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

            <button onclick="closeModal()">Đóng</button>

        </div>

    </div>

    <script src="/socket.io/socket.io.js"></script>

    <script src="/static/online.js"></script>

</body>

</html>

File practice.html:

<!DOCTYPE html>

<html lang="vi">

<head>

    <meta charset="UTF-8">

    <title>Luyện tập Sudoku</title>

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

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css">

</head>

<body>

    <div class="container">

        <div class="header">

            <div class="level">

                <span id="difficulty">Trung bình</span>

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

            </div>

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

            <div class="share">

                <i class="fas fa-share-alt"></i>

            </div>

        </div>

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

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

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

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

        </div>

        <div id="game-board" style="display:none;"></div>

        <div class="toolbar" style="display:none;" id="toolbar">

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

            <button title="Hoàn tác" onclick="undo()"><i class="fas fa-undo"></i></button>

        </div>

        <!-- Modal kết quả -->

        <div id="result-modal" class="modal" style="display:none;">

            <div class="modal-content">

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

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

                <button onclick="closeModal()">Đóng</button>

                <button onclick="startGame(currentLevel)">Chơi lại</button>

            </div>

        </div>

    </div>

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

</body>

</html>