<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Trading Signal Bot</title>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script src="https://cdn.jsdelivr.net/npm/chartjs-adapter-date-fns"></script>

<script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-annotation"></script>

<script src="https://cdn.tailwindcss.com"></script>

<script src="https://cdn.jsdelivr.net/npm/canvas-confetti@1.5.1/dist/confetti.browser.min.js"></script>

<script>

tailwind.config = {

darkMode: 'class',

theme: {

extend: {

colors: {

primary: '#5D5CDE',

'primary-dark': '#4a49b0',

buy: '#22c55e',

sell: '#ef4444',

neutral: '#6b7280',

},

animation: {

'spin-slow': 'spin 3s linear infinite',

}

}

}

}

</script>

<style>

.indicator-badge {

@apply inline-block px-2 py-1 text-xs font-medium rounded-full;

}

.indicator-badge.buy {

@apply bg-green-100 text-green-800 dark:bg-green-900 dark:text-green-100;

}

.indicator-badge.sell {

@apply bg-red-100 text-red-800 dark:bg-red-900 dark:text-red-100;

}

.indicator-badge.neutral {

@apply bg-gray-100 text-gray-800 dark:bg-gray-700 dark:text-gray-100;

}

.auth-input {

@apply w-full px-4 py-2 mt-2 text-base border rounded-md focus:outline-none focus:ring-2 focus:ring-primary dark:bg-gray-800 dark:border-gray-700;

}

.auth-btn {

@apply w-full px-4 py-2 mt-4 text-white transition-colors duration-200 transform bg-primary rounded-md hover:bg-primary-dark focus:outline-none focus:bg-primary-dark;

}

.captcha-container {

@apply mt-4 p-4 bg-gray-100 dark:bg-gray-800 rounded-md;

}

</style>

</head>

<body class="font-sans antialiased min-h-screen bg-gray-50 dark:bg-gray-900 text-gray-900 dark:text-gray-100 transition-colors duration-200">

<div class="container mx-auto px-4 py-6">

<!-- Auth Screen -->

<div id="authScreen" class="max-w-md mx-auto">

<div class="text-center mb-10">

<h1 class="text-3xl font-bold text-primary dark:text-primary mb-2">Trading Signal Bot</h1>

<p class="text-gray-600 dark:text-gray-400">Sign in to access real-time trading signals</p>

</div>

<div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-6">

<!-- Auth Tabs -->

<div class="flex border-b border-gray-200 dark:border-gray-700 mb-6">

<button id="loginTabBtn" class="px-4 py-2 font-medium text-primary border-b-2 border-primary">Login</button>

<button id="registerTabBtn" class="px-4 py-2 font-medium text-gray-500 dark:text-gray-400">Register</button>

</div>

<!-- Login Form -->

<form id="loginForm" class="space-y-4">

<div>

<label for="loginUsername" class="block text-sm font-medium">Username</label>

<input type="text" id="loginUsername" class="auth-input" placeholder="Enter your username" required>

</div>

<div>

<label for="loginPassword" class="block text-sm font-medium">Password</label>

<input type="password" id="loginPassword" class="auth-input" placeholder="Enter your password" required>

</div>

<div id="loginError" class="text-red-500 text-sm hidden"></div>

<button type="submit" class="auth-btn">Login</button>

</form>

<!-- Register Form -->

<form id="registerForm" class="space-y-4 hidden">

<div>

<label for="registerUsername" class="block text-sm font-medium">Username</label>

<input type="text" id="registerUsername" class="auth-input" placeholder="Choose a username" required>

</div>

<div>

<label for="registerPassword" class="block text-sm font-medium">Password</label>

<input type="password" id="registerPassword" class="auth-input" placeholder="Choose a password" required>

</div>

<div>

<label for="confirmPassword" class="block text-sm font-medium">Confirm Password</label>

<input type="password" id="confirmPassword" class="auth-input" placeholder="Confirm your password" required>

</div>

<!-- CAPTCHA -->

<div class="captcha-container">

<label class="block text-sm font-medium mb-2">CAPTCHA Verification</label>

<div id="captchaQuestion" class="text-lg font-medium text-center mb-2"></div>

<input type="number" id="captchaAnswer" class="auth-input" placeholder="Enter the answer" required>

</div>

<div id="registerError" class="text-red-500 text-sm hidden"></div>

<button type="submit" class="auth-btn">Register</button>

</form>

</div>

</div>

<!-- Main App Screen -->

<div id="appScreen" class="hidden">

<header class="mb-8 flex justify-between items-center">

<div>

<h1 class="text-3xl font-bold text-primary dark:text-primary mb-2">Trading Signal Bot</h1>

<p class="text-gray-600 dark:text-gray-400">Real-time market analysis and trading signals</p>

</div>

<div class="flex items-center space-x-4">

<span id="userDisplay" class="text-gray-700 dark:text-gray-300"></span>

<button id="logoutBtn" class="px-4 py-2 bg-gray-200 hover:bg-gray-300 dark:bg-gray-700 dark:hover:bg-gray-600 rounded-md">Logout</button>

</div>

</header>

<div class="grid grid-cols-1 lg:grid-cols-3 gap-6">

<!-- Chart and Signals Section -->

<div class="lg:col-span-2 space-y-6">

<!-- Market Chart -->

<div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-4">

<div class="flex justify-between items-center mb-4">

<h2 class="text-xl font-semibold">Market Chart (1m)</h2>

<div>

<button id="startButton" class="bg-primary hover:bg-primary-dark text-white px-4 py-2 rounded-md mr-2">Start</button>

<button id="pauseButton" class="bg-gray-500 hover:bg-gray-600 text-white px-4 py-2 rounded-md">Pause</button>

</div>

</div>

<div class="h-[400px] w-full">

<canvas id="priceChart"></canvas>

</div>

</div>

<!-- Latest Signals -->

<div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-4">

<h2 class="text-xl font-semibold mb-4">Trading Signals</h2>

<div id="signals" class="space-y-3 max-h-[300px] overflow-y-auto">

<div class="text-center text-gray-500 dark:text-gray-400">

Start the bot to generate signals

</div>

</div>

</div>

</div>

<!-- Settings and Strategy Section -->

<div class="space-y-6">

<!-- Bot Settings -->

<div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-4">

<h2 class="text-xl font-semibold mb-4">Bot Settings</h2>

<div class="space-y-4">

<div>

<label for="volatility" class="block text-sm font-medium mb-1">Market Volatility</label>

<input type="range" id="volatility" min="1" max="10" value="5" class="w-full h-2 bg-gray-200 rounded-lg appearance-none cursor-pointer dark:bg-gray-700">

<div class="flex justify-between text-xs text-gray-500 dark:text-gray-400 mt-1">

<span>Low</span>

<span>High</span>

</div>

</div>

<div>

<label for="market" class="block text-sm font-medium mb-1">Market Type</label>

<select id="market" class="w-full rounded-md border-gray-300 dark:border-gray-600 dark:bg-gray-700 shadow-sm text-base">

<option value="forex">Forex</option>

<option value="commodities">Commodities</option>

<option value="crypto">Cryptocurrency</option>

<option value="synthetic" selected>Synthetic Indices</option>

<option value="stocks">Global Stocks</option>

</select>

</div>

<div class="mt-4">

<label for="broker" class="block text-sm font-medium mb-1">Broker</label>

<select id="broker" class="w-full rounded-md border-gray-300 dark:border-gray-600 dark:bg-gray-700 shadow-sm text-base">

<option value="deriv">Deriv</option>

<option value="ig">IG</option>

<option value="fxcm">FXCM</option>

<option value="oanda">OANDA</option>

<option value="exness">Exness</option>

<option value="alpari">Alpari</option>

<option value="fbs">FBS</option>

<option value="xm">XM</option>

<option value="icmarkets">IC Markets</option>

<option value="fxtm">FXTM</option>

<option value="avatrade">AvaTrade</option>

<option value="pepperstone">Pepperstone</option>

<option value="hotforex">HotForex</option>

<option value="xtrader">XTrader</option>

<option value="axitrader">AxiTrader</option>

</select>

</div>

<div class="mt-4">

<label for="symbol" class="block text-sm font-medium mb-1">Symbol</label>

<select id="symbol" class="w-full rounded-md border-gray-300 dark:border-gray-600 dark:bg-gray-700 shadow-sm text-base">

<option value="volatility\_10">Volatility 10 Index</option>

<option value="volatility\_25">Volatility 25 Index</option>

<option value="volatility\_50">Volatility 50 Index</option>

<option value="volatility\_75">Volatility 75 Index</option>

<option value="volatility\_100">Volatility 100 Index</option>

<option value="crash\_1000">Crash 1000 Index</option>

<option value="boom\_1000">Boom 1000 Index</option>

<option value="step\_index">Step Index</option>

<option value="jump\_index">Jump Index</option>

</select>

</div>

<div class="mt-4">

<label for="trend" class="block text-sm font-medium mb-1">Market Trend</label>

<select id="trend" class="w-full rounded-md border-gray-300 dark:border-gray-600 dark:bg-gray-700 shadow-sm text-base">

<option value="sideways">Sideways</option>

<option value="bullish">Bullish</option>

<option value="bearish">Bearish</option>

<option value="random">Random</option>

</select>

</div>

<div>

<label for="indicators" class="block text-sm font-medium mb-1">Active Indicators</label>

<div class="space-y-2">

<div class="flex items-center">

<input type="checkbox" id="rsi" checked class="h-4 w-4 text-primary dark:text-primary border-gray-300 rounded">

<label for="rsi" class="ml-2 text-sm">RSI</label>

</div>

<div class="flex items-center">

<input type="checkbox" id="macd" checked class="h-4 w-4 text-primary dark:text-primary border-gray-300 rounded">

<label for="macd" class="ml-2 text-sm">MACD</label>

</div>

<div class="flex items-center">

<input type="checkbox" id="ema" checked class="h-4 w-4 text-primary dark:text-primary border-gray-300 rounded">

<label for="ema" class="ml-2 text-sm">EMA Crossover</label>

</div>

<div class="flex items-center">

<input type="checkbox" id="bb" checked class="h-4 w-4 text-primary dark:text-primary border-gray-300 rounded">

<label for="bb" class="ml-2 text-sm">Bollinger Bands</label>

</div>

</div>

</div>

</div>

</div>

<!-- Current Strategy -->

<div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-4">

<h2 class="text-xl font-semibold mb-4">Current Strategy</h2>

<div id="strategy" class="text-sm space-y-2">

<p>Click Start to generate a trading strategy</p>

</div>

<button id="generateStrategy" class="mt-4 bg-primary hover:bg-primary-dark text-white px-4 py-2 rounded-md w-full">Generate New Strategy</button>

</div>

<!-- Performance Metrics -->

<div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-4">

<h2 class="text-xl font-semibold mb-4">Performance</h2>

<div class="grid grid-cols-2 gap-4">

<div class="text-center">

<span class="block text-sm text-gray-600 dark:text-gray-400">Win Rate</span>

<span id="winRate" class="text-xl font-bold">0%</span>

</div>

<div class="text-center">

<span class="block text-sm text-gray-600 dark:text-gray-400">Profit Factor</span>

<span id="profitFactor" class="text-xl font-bold">0.00</span>

</div>

<div class="text-center">

<span class="block text-sm text-gray-600 dark:text-gray-400">Total Signals</span>

<span id="totalSignals" class="text-xl font-bold">0</span>

</div>

<div class="text-center">

<span class="block text-sm text-gray-600 dark:text-gray-400">Accuracy</span>

<span id="accuracy" class="text-xl font-bold">0%</span>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

<script>

// Dark mode support

if (window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches) {

document.documentElement.classList.add('dark');

}

window.matchMedia('(prefers-color-scheme: dark)').addEventListener('change', event => {

if (event.matches) {

document.documentElement.classList.add('dark');

} else {

document.documentElement.classList.remove('dark');

}

});

// Authentication System

const AuthSystem = (() => {

// LocalStorage keys

const USERS\_KEY = 'trading\_bot\_users';

const CURRENT\_USER\_KEY = 'trading\_bot\_current\_user';

// Get users from local storage

const getUsers = () => {

const usersJson = localStorage.getItem(USERS\_KEY);

return usersJson ? JSON.parse(usersJson) : {};

};

// Save users to local storage

const saveUsers = (users) => {

localStorage.setItem(USERS\_KEY, JSON.stringify(users));

};

// Register a new user

const registerUser = (username, password) => {

const users = getUsers();

// Check if user already exists

if (users[username]) {

return { success: false, message: 'Username already exists' };

}

// Add new user

users[username] = {

password,

createdAt: new Date().toISOString(),

settings: {

preferredMarket: 'synthetic',

preferredBroker: 'deriv'

}

};

saveUsers(users);

return { success: true };

};

// Login user

const loginUser = (username, password) => {

const users = getUsers();

// Check if user exists and password is correct

if (!users[username] || users[username].password !== password) {

return { success: false, message: 'Invalid username or password' };

}

// Set current user

localStorage.setItem(CURRENT\_USER\_KEY, username);

return { success: true, username };

};

// Logout user

const logoutUser = () => {

localStorage.removeItem(CURRENT\_USER\_KEY);

};

// Check if user is logged in

const isLoggedIn = () => {

return !!localStorage.getItem(CURRENT\_USER\_KEY);

};

// Get current user

const getCurrentUser = () => {

return localStorage.getItem(CURRENT\_USER\_KEY);

};

// Generate a simple math CAPTCHA

const generateCaptcha = () => {

const operations = ['+', '-', '\*'];

const operation = operations[Math.floor(Math.random() \* operations.length)];

let num1, num2, answer;

switch (operation) {

case '+':

num1 = Math.floor(Math.random() \* 10) + 1;

num2 = Math.floor(Math.random() \* 10) + 1;

answer = num1 + num2;

break;

case '-':

num1 = Math.floor(Math.random() \* 10) + 10; // Ensure positive answer

num2 = Math.floor(Math.random() \* num1);

answer = num1 - num2;

break;

case '\*':

num1 = Math.floor(Math.random() \* 5) + 1;

num2 = Math.floor(Math.random() \* 5) + 1;

answer = num1 \* num2;

break;

}

return {

question: `${num1} ${operation} ${num2} = ?`,

answer

};

};

return {

registerUser,

loginUser,

logoutUser,

isLoggedIn,

getCurrentUser,

generateCaptcha

};

})();

// Auth UI Management

document.addEventListener('DOMContentLoaded', () => {

// DOM elements

const authScreen = document.getElementById('authScreen');

const appScreen = document.getElementById('appScreen');

const loginForm = document.getElementById('loginForm');

const registerForm = document.getElementById('registerForm');

const loginTabBtn = document.getElementById('loginTabBtn');

const registerTabBtn = document.getElementById('registerTabBtn');

const loginError = document.getElementById('loginError');

const registerError = document.getElementById('registerError');

const logoutBtn = document.getElementById('logoutBtn');

const userDisplay = document.getElementById('userDisplay');

const captchaQuestion = document.getElementById('captchaQuestion');

// CAPTCHA state

let currentCaptchaAnswer = null;

// Function to show/hide screens based on login state

const updateScreenVisibility = () => {

if (AuthSystem.isLoggedIn()) {

authScreen.classList.add('hidden');

appScreen.classList.remove('hidden');

// Update user display

const username = AuthSystem.getCurrentUser();

userDisplay.textContent = `Welcome, ${username}!`;

// Initialize the trading bot after login

if (window.TradingBot && typeof window.TradingBot.init === 'function') {

window.TradingBot.init();

}

} else {

authScreen.classList.remove('hidden');

appScreen.classList.add('hidden');

}

};

// Generate new CAPTCHA

const refreshCaptcha = () => {

const captcha = AuthSystem.generateCaptcha();

captchaQuestion.textContent = captcha.question;

currentCaptchaAnswer = captcha.answer;

};

// Tab switching

loginTabBtn.addEventListener('click', () => {

loginTabBtn.classList.add('text-primary', 'border-b-2', 'border-primary');

loginTabBtn.classList.remove('text-gray-500', 'dark:text-gray-400');

registerTabBtn.classList.add('text-gray-500', 'dark:text-gray-400');

registerTabBtn.classList.remove('text-primary', 'border-b-2', 'border-primary');

loginForm.classList.remove('hidden');

registerForm.classList.add('hidden');

});

registerTabBtn.addEventListener('click', () => {

registerTabBtn.classList.add('text-primary', 'border-b-2', 'border-primary');

registerTabBtn.classList.remove('text-gray-500', 'dark:text-gray-400');

loginTabBtn.classList.add('text-gray-500', 'dark:text-gray-400');

loginTabBtn.classList.remove('text-primary', 'border-b-2', 'border-primary');

registerForm.classList.remove('hidden');

loginForm.classList.add('hidden');

// Generate CAPTCHA when switching to register tab

refreshCaptcha();

});

// Login form submission

loginForm.addEventListener('submit', (e) => {

e.preventDefault();

const username = document.getElementById('loginUsername').value;

const password = document.getElementById('loginPassword').value;

loginError.classList.add('hidden');

const result = AuthSystem.loginUser(username, password);

if (result.success) {

updateScreenVisibility();

loginForm.reset();

// Show confetti for successful login

confetti({

particleCount: 100,

spread: 70,

origin: { y: 0.6 }

});

} else {

loginError.textContent = result.message;

loginError.classList.remove('hidden');

}

});

// Register form submission

registerForm.addEventListener('submit', (e) => {

e.preventDefault();

const username = document.getElementById('registerUsername').value;

const password = document.getElementById('registerPassword').value;

const confirmPassword = document.getElementById('confirmPassword').value;

const captchaAnswer = parseInt(document.getElementById('captchaAnswer').value);

registerError.classList.add('hidden');

// Validate inputs

if (password !== confirmPassword) {

registerError.textContent = 'Passwords do not match';

registerError.classList.remove('hidden');

return;

}

if (captchaAnswer !== currentCaptchaAnswer) {

registerError.textContent = 'CAPTCHA answer is incorrect';

registerError.classList.remove('hidden');

refreshCaptcha();

return;

}

const result = AuthSystem.registerUser(username, password);

if (result.success) {

// Auto-login after successful registration

AuthSystem.loginUser(username, password);

updateScreenVisibility();

registerForm.reset();

// Show confetti for successful registration

confetti({

particleCount: 150,

spread: 100,

origin: { y: 0.6 }

});

} else {

registerError.textContent = result.message;

registerError.classList.remove('hidden');

refreshCaptcha();

}

});

// Logout button

logoutBtn.addEventListener('click', () => {

AuthSystem.logoutUser();

updateScreenVisibility();

});

// Check login status on page load

updateScreenVisibility();

// Generate initial CAPTCHA

refreshCaptcha();

});

// Trading Bot Logic

const TradingBot = (() => {

// State

let marketData = [];

let signals = [];

let isRunning = false;

let intervalId = null;

let strategy = null;

let performance = {

winCount: 0,

lossCount: 0,

totalProfit: 0,

totalLoss: 0,

signalCount: 0,

correctSignals: 0

};

// Chart

let priceChart = null;

// Constants

const MAX\_DATA\_POINTS = 100;

const CANDLE\_INTERVAL = 1000; // 1 second represents 1 minute in simulation

// Initialize

const init = () => {

initChart();

updateSymbols(); // Initialize symbols based on default market/broker

initEventListeners();

generateInitialData();

generateNewStrategy();

};

// Initialize the price chart

const initChart = () => {

const ctx = document.getElementById('priceChart').getContext('2d');

priceChart = new Chart(ctx, {

type: 'line',

data: {

datasets: [{

label: 'Price',

data: [],

borderColor: '#5D5CDE',

borderWidth: 2,

tension: 0.1,

pointRadius: 0,

pointHoverRadius: 5,

}]

},

options: {

responsive: true,

maintainAspectRatio: false,

scales: {

x: {

type: 'time',

time: {

unit: 'minute',

tooltipFormat: 'HH:mm:ss',

displayFormats: {

minute: 'HH:mm'

}

},

grid: {

display: false

}

},

y: {

grid: {

color: 'rgba(200, 200, 200, 0.1)',

},

ticks: {

callback: (value) => `$${value.toFixed(2)}`

}

}

},

interaction: {

intersect: false,

mode: 'index',

},

plugins: {

tooltip: {

callbacks: {

label: (context) => `Price: $${context.raw.y.toFixed(2)}`

}

},

annotation: {

annotations: []

}

}

}

});

};

// Initialize event listeners

const initEventListeners = () => {

document.getElementById('startButton').addEventListener('click', startBot);

document.getElementById('pauseButton').addEventListener('click', pauseBot);

document.getElementById('generateStrategy').addEventListener('click', generateNewStrategy);

// Settings listeners

document.getElementById('volatility').addEventListener('input', updateMarketSettings);

document.getElementById('trend').addEventListener('change', updateMarketSettings);

document.getElementById('market').addEventListener('change', updateMarketSettings);

document.getElementById('broker').addEventListener('change', updateSymbols);

document.getElementById('symbol').addEventListener('change', updateMarketSettings);

// Indicator checkboxes

document.querySelectorAll('input[type=checkbox]').forEach(checkbox => {

checkbox.addEventListener('change', generateNewStrategy);

});

};

// Update symbols based on selected broker and market type

const updateSymbols = () => {

const market = document.getElementById('market').value;

const broker = document.getElementById('broker').value;

const symbolSelect = document.getElementById('symbol');

// Clear existing options

symbolSelect.innerHTML = '';

// Add new options based on market and broker

if (market === 'synthetic') {

if (broker === 'deriv') {

addOption(symbolSelect, 'volatility\_10', 'Volatility 10 Index');

addOption(symbolSelect, 'volatility\_25', 'Volatility 25 Index');

addOption(symbolSelect, 'volatility\_50', 'Volatility 50 Index');

addOption(symbolSelect, 'volatility\_75', 'Volatility 75 Index');

addOption(symbolSelect, 'volatility\_100', 'Volatility 100 Index');

addOption(symbolSelect, 'crash\_1000', 'Crash 1000 Index');

addOption(symbolSelect, 'boom\_1000', 'Boom 1000 Index');

addOption(symbolSelect, 'step\_index', 'Step Index');

addOption(symbolSelect, 'jump\_index', 'Jump Index');

} else if (broker === 'alpari') {

addOption(symbolSelect, 'random\_index', 'Random Index');

addOption(symbolSelect, 'random\_volatility', 'Random Volatility');

addOption(symbolSelect, 'random\_100', 'Random 100 Index');

} else if (broker === 'fbs') {

addOption(symbolSelect, 'fbs\_vix', 'FBS VIX');

addOption(symbolSelect, 'fbs\_btcvix', 'FBS BTC Volatility');

} else if (broker === 'xm') {

addOption(symbolSelect, 'xm\_synth50', 'XM Synthetic 50');

addOption(symbolSelect, 'xm\_vix75', 'XM VIX 75');

} else {

addOption(symbolSelect, 'random\_index', 'Random Index');

addOption(symbolSelect, 'random\_volatility', 'Random Volatility');

}

} else if (market === 'forex') {

addOption(symbolSelect, 'eurusd', 'EUR/USD');

addOption(symbolSelect, 'gbpusd', 'GBP/USD');

addOption(symbolSelect, 'usdjpy', 'USD/JPY');

addOption(symbolSelect, 'audusd', 'AUD/USD');

addOption(symbolSelect, 'usdcad', 'USD/CAD');

addOption(symbolSelect, 'eurgbp', 'EUR/GBP');

// Add more pairs based on broker

if (['icmarkets', 'fxtm', 'xm', 'exness'].includes(broker)) {

addOption(symbolSelect, 'eurjpy', 'EUR/JPY');

addOption(symbolSelect, 'gbpjpy', 'GBP/JPY');

addOption(symbolSelect, 'nzdusd', 'NZD/USD');

addOption(symbolSelect, 'eurchf', 'EUR/CHF');

}

} else if (market === 'crypto') {

addOption(symbolSelect, 'btcusd', 'BTC/USD');

addOption(symbolSelect, 'ethusd', 'ETH/USD');

addOption(symbolSelect, 'ltcusd', 'LTC/USD');

addOption(symbolSelect, 'xrpusd', 'XRP/USD');

if (['xm', 'icmarkets', 'fxtm'].includes(broker)) {

addOption(symbolSelect, 'solusd', 'SOL/USD');

addOption(symbolSelect, 'avaxusd', 'AVAX/USD');

addOption(symbolSelect, 'dogeusd', 'DOGE/USD');

addOption(symbolSelect, 'adausd', 'ADA/USD');

}

} else if (market === 'commodities') {

addOption(symbolSelect, 'xauusd', 'Gold (XAU/USD)');

addOption(symbolSelect, 'xagusd', 'Silver (XAG/USD)');

addOption(symbolSelect, 'oil', 'Crude Oil');

if (['fxtm', 'avatrade', 'icmarkets'].includes(broker)) {

addOption(symbolSelect, 'natural\_gas', 'Natural Gas');

addOption(symbolSelect, 'copper', 'Copper');

addOption(symbolSelect, 'platinum', 'Platinum');

}

} else if (market === 'stocks') {

addOption(symbolSelect, 'aapl', 'Apple (AAPL)');

addOption(symbolSelect, 'msft', 'Microsoft (MSFT)');

addOption(symbolSelect, 'amzn', 'Amazon (AMZN)');

addOption(symbolSelect, 'googl', 'Google (GOOGL)');

addOption(symbolSelect, 'tsla', 'Tesla (TSLA)');

if (['avatrade', 'ig', 'xtrader'].includes(broker)) {

addOption(symbolSelect, 'meta', 'Meta (META)');

addOption(symbolSelect, 'nvda', 'NVIDIA (NVDA)');

addOption(symbolSelect, 'nflx', 'Netflix (NFLX)');

addOption(symbolSelect, 'ba', 'Boeing (BA)');

}

}

// Trigger market settings update with new symbol

updateMarketSettings();

};

// Helper function to add options to select element

const addOption = (selectElement, value, text) => {

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

option.value = value;

option.textContent = text;

selectElement.appendChild(option);

};

// Generate initial market data

const generateInitialData = () => {

const basePrice = 100;

const now = new Date();

for (let i = 0; i < 30; i++) {

const time = new Date(now.getTime() - (30 - i) \* CANDLE\_INTERVAL);

marketData.push({

time,

price: basePrice + Math.random() \* 5 - 2.5

});

}

updateChart();

};

// Generate new price data

const generateNewPrice = () => {

const volatility = parseInt(document.getElementById('volatility').value) / 10;

const trend = document.getElementById('trend').value;

const market = document.getElementById('market').value;

const symbol = document.getElementById('symbol').value;

let lastPrice = marketData[marketData.length - 1].price;

let trendFactor = 0;

let marketVolatilityMultiplier = 1;

// Adjust trend factor based on the selected trend

switch (trend) {

case 'bullish':

trendFactor = 0.1;

break;

case 'bearish':

trendFactor = -0.1;

break;

case 'random':

trendFactor = (Math.random() - 0.5) \* 0.2;

break;

default: // sideways

trendFactor = 0;

}

// Adjust volatility based on market type and specific symbol

if (market === 'synthetic') {

if (symbol.includes('volatility')) {

// Extract volatility level from symbol (e.g., volatility\_75 -> 75)

const volatilityLevel = parseInt(symbol.split('\_')[1]) || 10;

marketVolatilityMultiplier = volatilityLevel / 25; // Scale based on volatility level

} else if (symbol.includes('crash')) {

// Crash indices have higher downside volatility with sudden drops

marketVolatilityMultiplier = 1.5;

if (Math.random() < 0.05) { // 5% chance of a crash event

trendFactor -= 0.5; // Significant downward bias during crash events

}

} else if (symbol.includes('boom')) {

// Boom indices have higher upside volatility with sudden rises

marketVolatilityMultiplier = 1.5;

if (Math.random() < 0.05) { // 5% chance of a boom event

trendFactor += 0.5; // Significant upward bias during boom events

}

} else if (symbol.includes('step')) {

// Step index has discrete price movements

const stepSize = 0.2;

return lastPrice + (Math.round((Math.random() - 0.5 + trendFactor \* 0.5) / stepSize) \* stepSize);

} else if (symbol.includes('jump')) {

// Jump index has occasional large jumps

if (Math.random() < 0.02) { // 2% chance of a jump

const jumpDirection = Math.random() > 0.5 ? 1 : -1;

return lastPrice + (jumpDirection \* (Math.random() \* 3 + 1));

}

}

} else if (market === 'forex') {

// Forex markets typically have lower volatility

marketVolatilityMultiplier = 0.5;

// Major pairs (EUR/USD, GBP/USD) tend to be less volatile than crosses

if (symbol === 'eurusd' || symbol === 'gbpusd') {

marketVolatilityMultiplier \*= 0.8;

}

} else if (market === 'crypto') {

// Cryptocurrencies have higher volatility

marketVolatilityMultiplier = 2.5;

// Bitcoin is often less volatile than altcoins

if (symbol === 'btcusd') {

marketVolatilityMultiplier \*= 0.7;

}

} else if (market === 'commodities') {

// Gold is typically less volatile than other commodities

if (symbol === 'xauusd') {

marketVolatilityMultiplier = 0.8;

} else {

marketVolatilityMultiplier = 1.2;

}

}

// Apply market-specific random walk with trend and volatility adjustments

const baseChange = (Math.random() - 0.5) \* volatility \* marketVolatilityMultiplier;

const change = baseChange + trendFactor;

let newPrice = lastPrice + change;

// Ensure price doesn't go below zero

newPrice = Math.max(newPrice, 0.01);

return newPrice;

};

// Update the chart with new data

const updateChart = () => {

if (!priceChart) return;

// Create data points for the chart

const chartData = marketData.map(candle => ({

x: candle.time,

y: candle.price

}));

// Update the chart

priceChart.data.datasets[0].data = chartData;

// Add signal annotations

priceChart.options.plugins.annotation.annotations = signals

.slice(-10) // Only show recent signals

.map((signal, index) => {

const signalData = marketData.find(d => d.time.getTime() === signal.time.getTime());

if (!signalData) return null;

return {

type: 'point',

xValue: signal.time,

yValue: signalData.price,

backgroundColor: signal.type === 'buy' ? 'rgba(34, 197, 94, 0.8)' : 'rgba(239, 68, 68, 0.8)',

borderColor: signal.type === 'buy' ? 'rgba(34, 197, 94, 1)' : 'rgba(239, 68, 68, 1)',

borderWidth: 2,

radius: 6,

label: {

content: signal.type === 'buy' ? '↑' : '↓',

enabled: true,

color: 'white',

font: {

weight: 'bold'

}

}

};

})

.filter(Boolean);

priceChart.update();

};

// Calculate technical indicators

const calculateIndicators = (data) => {

if (data.length < 30) return null;

const prices = data.map(d => d.price);

const result = {

price: prices[prices.length - 1],

signals: {}

};

// RSI (Relative Strength Index)

if (document.getElementById('rsi').checked) {

const rsiPeriod = 14;

const rsiValues = calculateRSI(prices, rsiPeriod);

const currentRSI = rsiValues[rsiValues.length - 1];

result.rsi = currentRSI;

// RSI signal logic

if (currentRSI < 30) {

result.signals.rsi = 'buy';

} else if (currentRSI > 70) {

result.signals.rsi = 'sell';

} else {

result.signals.rsi = 'neutral';

}

}

// MACD (Moving Average Convergence Divergence)

if (document.getElementById('macd').checked) {

const fastPeriod = 12;

const slowPeriod = 26;

const signalPeriod = 9;

const macdResult = calculateMACD(prices, fastPeriod, slowPeriod, signalPeriod);

const macdLine = macdResult.macdLine[macdResult.macdLine.length - 1];

const signalLine = macdResult.signalLine[macdResult.signalLine.length - 1];

result.macd = {

macdLine,

signalLine,

histogram: macdLine - signalLine

};

// MACD signal logic

if (macdLine > signalLine && macdLine > 0) {

result.signals.macd = 'buy';

} else if (macdLine < signalLine && macdLine < 0) {

result.signals.macd = 'sell';

} else {

result.signals.macd = 'neutral';

}

}

// EMA Crossover

if (document.getElementById('ema').checked) {

const shortPeriod = 9;

const longPeriod = 21;

const shortEMA = calculateEMA(prices, shortPeriod);

const longEMA = calculateEMA(prices, longPeriod);

const currentShortEMA = shortEMA[shortEMA.length - 1];

const currentLongEMA = longEMA[longEMA.length - 1];

const prevShortEMA = shortEMA[shortEMA.length - 2];

const prevLongEMA = longEMA[longEMA.length - 2];

result.ema = {

short: currentShortEMA,

long: currentLongEMA

};

// EMA crossover signal logic

if (prevShortEMA <= prevLongEMA && currentShortEMA > currentLongEMA) {

result.signals.ema = 'buy';

} else if (prevShortEMA >= prevLongEMA && currentShortEMA < currentLongEMA) {

result.signals.ema = 'sell';

} else {

result.signals.ema = 'neutral';

}

}

// Bollinger Bands

if (document.getElementById('bb').checked) {

const bbPeriod = 20;

const deviation = 2;

const sma = calculateSMA(prices, bbPeriod);

const currentSMA = sma[sma.length - 1];

// Calculate standard deviation

const currentPrices = prices.slice(-bbPeriod);

const std = calculateStandardDeviation(currentPrices);

const upperBand = currentSMA + (deviation \* std);

const lowerBand = currentSMA - (deviation \* std);

result.bb = {

sma: currentSMA,

upper: upperBand,

lower: lowerBand

};

// Bollinger Bands signal logic

const currentPrice = prices[prices.length - 1];

if (currentPrice < lowerBand) {

result.signals.bb = 'buy';

} else if (currentPrice > upperBand) {

result.signals.bb = 'sell';

} else {

result.signals.bb = 'neutral';

}

}

return result;

};

// Generate a new trading strategy

const generateNewStrategy = () => {

const activeIndicators = [];

if (document.getElementById('rsi').checked) activeIndicators.push('rsi');

if (document.getElementById('macd').checked) activeIndicators.push('macd');

if (document.getElementById('ema').checked) activeIndicators.push('ema');

if (document.getElementById('bb').checked) activeIndicators.push('bb');

if (activeIndicators.length === 0) {

strategy = null;

document.getElementById('strategy').innerHTML = '<p>Please select at least one indicator</p>';

return;

}

// Randomly generate strategy weights

const weights = {};

let totalWeight = 0;

activeIndicators.forEach(indicator => {

weights[indicator] = Math.random() \* 0.8 + 0.2; // Weight between 0.2 and 1.0

totalWeight += weights[indicator];

});

// Normalize weights

activeIndicators.forEach(indicator => {

weights[indicator] = weights[indicator] / totalWeight;

});

// Generate confirmation requirements

const confirmationCount = Math.floor(Math.random() \* activeIndicators.length) + 1;

strategy = {

indicators: activeIndicators,

weights,

confirmationCount,

description: generateStrategyDescription(activeIndicators, weights, confirmationCount)

};

document.getElementById('strategy').innerHTML = strategy.description;

};

// Generate a description of the strategy

const generateStrategyDescription = (indicators, weights, confirmationCount) => {

const weightsSorted = Object.entries(weights)

.sort((a, b) => b[1] - a[1])

.map(([indicator, weight]) => {

const percentage = Math.round(weight \* 100);

return `<div class="flex justify-between">

<span class="capitalize">${indicator.toUpperCase()}</span>

<span class="font-medium">${percentage}%</span>

</div>`;

});

const indicatorList = weightsSorted.join('');

return `

<p class="font-medium">Strategy Parameters:</p>

<div class="mt-2 space-y-1">

${indicatorList}

</div>

<p class="mt-3">Requires <span class="font-medium">${confirmationCount}</span> indicator(s) to confirm a signal</p>

`;

};

// Apply the trading strategy to generate signals

const applyStrategy = (indicators) => {

if (!strategy || !indicators) return null;

const { signals } = indicators;

const { weights, confirmationCount } = strategy;

let buyScore = 0;

let sellScore = 0;

let confirmationBuy = 0;

let confirmationSell = 0;

// Calculate scores based on indicator signals and their weights

Object.entries(signals).forEach(([indicator, signal]) => {

if (signal === 'buy') {

buyScore += weights[indicator] || 0;

confirmationBuy++;

} else if (signal === 'sell') {

sellScore += weights[indicator] || 0;

confirmationSell++;

}

});

// Generate a signal if confirmation threshold is met

if (confirmationBuy >= confirmationCount && buyScore > sellScore) {

return {

type: 'buy',

confidence: Math.min(buyScore \* 100, 100).toFixed(0),

indicators: Object.entries(signals)

.filter(([\_, signal]) => signal === 'buy')

.map(([indicator]) => indicator.toUpperCase())

};

} else if (confirmationSell >= confirmationCount && sellScore > buyScore) {

return {

type: 'sell',

confidence: Math.min(sellScore \* 100, 100).toFixed(0),

indicators: Object.entries(signals)

.filter(([\_, signal]) => signal === 'sell')

.map(([indicator]) => indicator.toUpperCase())

};

}

return null;

};

// Update market settings

const updateMarketSettings = () => {

// If the bot is running, apply the new settings immediately

if (isRunning) {

// Just let the next tick use the new settings

}

};

// Start the trading bot

const startBot = () => {

if (isRunning) return;

isRunning = true;

// If we don't have a strategy, generate one

if (!strategy) {

generateNewStrategy();

}

intervalId = setInterval(() => {

// Generate a new price

const newPrice = generateNewPrice();

const now = new Date();

// Get market and symbol information

const market = document.getElementById('market').value;

const symbol = document.getElementById('symbol').value;

// Update chart title with current market and symbol

const chartTitle = document.querySelector('.bg-white.dark\\:bg-gray-800 h2');

const symbolDisplay = symbol.split('\_').map(word => word.charAt(0).toUpperCase() + word.slice(1)).join(' ');

chartTitle.textContent = `${symbolDisplay} (1m)`;

// Add to market data

marketData.push({

time: now,

price: newPrice,

market,

symbol

});

// Limit the number of data points

if (marketData.length > MAX\_DATA\_POINTS) {

marketData.shift();

}

// Calculate indicators

const indicators = calculateIndicators(marketData);

// Apply strategy to generate signals

if (indicators) {

const signal = applyStrategy(indicators);

// If we have a signal, add it to the signals array

if (signal) {

signal.time = now;

signal.price = newPrice;

signals.push(signal);

// Add to the signals display

addSignalToDisplay(signal);

// Update performance

updatePerformance(signal);

}

}

// Update the chart

updateChart();

}, CANDLE\_INTERVAL);

};

// Pause the trading bot

const pauseBot = () => {

if (!isRunning) return;

isRunning = false;

clearInterval(intervalId);

};

// Add a signal to the display

const addSignalToDisplay = (signal) => {

const signalsContainer = document.getElementById('signals');

// Get market and symbol info

const market = document.getElementById('market').value;

const symbol = document.getElementById('symbol').value;

const symbolDisplay = symbol.split('\_').map(word => word.charAt(0).toUpperCase() + word.slice(1)).join(' ');

// Create the signal element

const signalElement = document.createElement('div');

signalElement.className = `p-3 border rounded-md ${signal.type === 'buy' ? 'border-green-200 bg-green-50 dark:border-green-900 dark:bg-green-900/20' : 'border-red-200 bg-red-50 dark:border-red-900 dark:bg-red-900/20'}`;

const time = signal.time.toLocaleTimeString();

const price = signal.price.toFixed(2);

const indicators = signal.indicators.join(', ');

signalElement.innerHTML = `

<div class="flex justify-between items-center">

<div>

<span class="indicator-badge ${signal.type}">${signal.type.toUpperCase()}</span>

<span class="ml-2 text-sm">@$${price}</span>

</div>

<span class="text-xs text-gray-500 dark:text-gray-400">${time}</span>

</div>

<div class="mt-1 text-sm">

<span class="text-gray-700 dark:text-gray-300">Symbol: ${symbolDisplay}</span>

</div>

<div class="mt-1 text-sm">

<span class="text-gray-700 dark:text-gray-300">Indicators: ${indicators}</span>

</div>

<div class="mt-1 text-xs">

<span class="text-gray-600 dark:text-gray-400">Confidence: ${signal.confidence}%</span>

</div>

`;

// Add to the signals container

if (signalsContainer.querySelector('.text-center')) {

signalsContainer.innerHTML = '';

}

signalsContainer.prepend(signalElement);

// Limit the number of displayed signals

const maxDisplayedSignals = 10;

const signalElements = signalsContainer.querySelectorAll('div');

if (signalElements.length > maxDisplayedSignals) {

for (let i = maxDisplayedSignals; i < signalElements.length; i++) {

signalElements[i].remove();

}

}

};

// Update performance metrics

const updatePerformance = (signal) => {

// Simulated outcome

const outcome = Math.random() > 0.5; // Random outcome for demonstration

performance.signalCount++;

if (outcome) {

performance.correctSignals++;

performance.winCount++;

performance.totalProfit += Math.random() \* 2 + 0.5; // Random profit between 0.5 and 2.5

} else {

performance.lossCount++;

performance.totalLoss += Math.random() \* 1.5 + 0.2; // Random loss between 0.2 and 1.7

}

// Update the display

const winRate = performance.winCount / performance.signalCount \* 100;

const profitFactor = performance.totalLoss > 0 ? performance.totalProfit / performance.totalLoss : performance.totalProfit;

const accuracy = performance.correctSignals / performance.signalCount \* 100;

document.getElementById('winRate').textContent = `${winRate.toFixed(1)}%`;

document.getElementById('profitFactor').textContent = profitFactor.toFixed(2);

document.getElementById('totalSignals').textContent = performance.signalCount;

document.getElementById('accuracy').textContent = `${accuracy.toFixed(1)}%`;

};

// Technical indicator calculation functions

const calculateSMA = (data, period) => {

const result = [];

for (let i = period - 1; i < data.length; i++) {

const sum = data.slice(i - period + 1, i + 1).reduce((a, b) => a + b, 0);

result.push(sum / period);

}

return result;

};

const calculateEMA = (data, period) => {

const result = [];

const k = 2 / (period + 1);

// First EMA is just SMA

result.push(data.slice(0, period).reduce((a, b) => a + b, 0) / period);

// Calculate remaining EMAs

for (let i = period; i < data.length; i++) {

result.push(data[i] \* k + result[result.length - 1] \* (1 - k));

}

return result;

};

const calculateRSI = (data, period) => {

const gains = [];

const losses = [];

// Calculate gains and losses

for (let i = 1; i < data.length; i++) {

const difference = data[i] - data[i - 1];

gains.push(difference > 0 ? difference : 0);

losses.push(difference < 0 ? Math.abs(difference) : 0);

}

const result = [];

// Calculate RSI

for (let i = period; i <= gains.length; i++) {

const gainSlice = gains.slice(i - period, i);

const lossSlice = losses.slice(i - period, i);

const avgGain = gainSlice.reduce((a, b) => a + b, 0) / period;

const avgLoss = lossSlice.reduce((a, b) => a + b, 0) / period;

if (avgLoss === 0) {

result.push(100);

} else {

const rs = avgGain / avgLoss;

result.push(100 - (100 / (1 + rs)));

}

}

return result;

};

const calculateMACD = (data, fastPeriod, slowPeriod, signalPeriod) => {

const fastEMA = calculateEMA(data, fastPeriod);

const slowEMA = calculateEMA(data, slowPeriod);

// Calculate MACD line (fast EMA - slow EMA)

const macdLine = [];

// Adjust for the length difference

const diff = slowPeriod - fastPeriod;

for (let i = 0; i < fastEMA.length; i++) {

if (i >= diff) {

macdLine.push(fastEMA[i] - slowEMA[i - diff]);

}

}

// Calculate signal line (EMA of MACD line)

const signalLine = calculateEMA(macdLine, signalPeriod);

return {

macdLine,

signalLine

};

};

const calculateStandardDeviation = (data) => {

const mean = data.reduce((a, b) => a + b, 0) / data.length;

const squaredDiffs = data.map(value => Math.pow(value - mean, 2));

const variance = squaredDiffs.reduce((a, b) => a + b, 0) / data.length;

return Math.sqrt(variance);

};

// Public API

return {

init

};

})();

// Initialize the trading bot on page load

window.TradingBot = TradingBot;

</script>

</body>

</html>