<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no" />

<title>Supply Chain Logistics Optimization</title>

<style>

/\* Reset and base \*/

html, body {

margin: 0; padding: 0; height: 100%;

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: #f0f4ff;

color: #1e293b;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

user-select: none;

}

#app {

max-width: 350px;

max-height: 600px;

margin: 10px auto;

background: white;

border-radius: 14px;

box-shadow: 0 8px 24px rgb(0 0 0 / 0.1);

padding: 15px 20px;

display: flex;

flex-direction: column;

}

h1 {

font-weight: 700;

color: #2563eb;

font-size: 1.5rem;

margin-bottom: 5px;

text-align: center;

}

h2 {

font-weight: 600;

font-size: 1.1rem;

margin: 15px 0 7px 0;

color: #1e293b;

border-bottom: 2px solid #3b82f6;

padding-bottom: 4px;

}

label {

font-weight: 600;

font-size: 0.9rem;

margin-bottom: 3px;

display: block;

}

input[type=number], input[type=text] {

width: 100%;

padding: 6px 8px;

font-size: 1rem;

border-radius: 6px;

border: 1.5px solid #cbd5e1;

margin-bottom: 10px;

box-sizing: border-box;

transition: border-color 0.25s;

}

input[type=number]:focus, input[type=text]:focus {

outline: none;

border-color: #2563eb;

box-shadow: 0 0 6px #93c5fd;

}

button {

background-color: #2563eb;

border: none;

color: white;

padding: 10px 0;

border-radius: 10px;

font-weight: 700;

font-size: 1rem;

cursor: pointer;

margin: 10px 0 0 0;

transition: background-color 0.3s;

}

button:active {

background-color: #1e40af;

}

#forecastOutput, #routeOutput {

background: #e0e7ff;

border-radius: 12px;

padding: 10px;

min-height: 40px;

font-family: monospace;

font-size: 0.9rem;

line-height: 1.3;

color: #1e293b;

}

#canvasContainer {

margin-top: 10px;

border-radius: 14px;

overflow: hidden;

background: white;

flex-grow: 1;

display: flex;

justify-content: center;

align-items: center;

}

canvas {

max-width: 100%;

border-radius: 14px;

background: #f9fafb;

}

/\* Scroll overflow for forecast and route output if needed \*/

#forecastOutput, #routeOutput {

max-height: 100px;

overflow-y: auto;

}

/\* Responsive for small screens \*/

@media (max-width: 400px) {

#app {

max-width: 95vw;

max-height: 95vh;

padding: 10px 12px;

}

h1 {

font-size: 1.3rem;

}

button {

font-size: 0.9rem;

padding: 8px 0;

}

}

</style>

</head>

<body>

<div id="app" role="main" aria-label="Supply Chain Logistics Optimization Application">

<h1>Supply Chain Logistics Optimization</h1>

<section aria-label="Demand Data Input">

<h2>Demand Data Input</h2>

<label for="demandCount">Number of demand points:</label>

<input type="number" id="demandCount" min="2" max="12" value="6" aria-describedby="demandCountDesc"/>

<div id="demandCountDesc" style="font-size:0.8rem; color:#64748b; margin-bottom:8px;">(2 to 12 delivery points)</div>

<button id="generateDemandBtn" aria-label="Generate Random Demand Data">Generate Demand Data</button>

</section>

<section aria-label="Demand Forecasting">

<h2>Demand Forecasting (Simple Moving Average)</h2>

<div id="forecastOutput" aria-live="polite" aria-atomic="true" role="region">Press "Generate Demand Data" to see forecast.</div>

</section>

<section aria-label="Route Optimization">

<h2>Route Optimization</h2>

<button id="optimizeRouteBtn" aria-label="Optimize Delivery Route">Optimize Route</button>

<div id="routeOutput" aria-live="polite" aria-atomic="true" role="region">Optimized route will appear here after clicking the button.</div>

</section>

<section aria-label="Route Map Visualization">

<h2>Route Map</h2>

<div id="canvasContainer">

<canvas id="routeCanvas" width="320" height="280" aria-label="Route Map Visualization"></canvas>

</div>

</section>

</div>

<script>

(() => {

// Utility functions

function randomInt(min, max) {

return Math.floor(Math.random() \* (max - min + 1)) + min;

}

// Generate demand points with coordinates and demand values

function generateDemandPoints(count) {

const points = [];

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

points.push({

id: i,

x: Math.random() \* 300 + 10,

y: Math.random() \* 260 + 10,

demand: randomInt(5, 20)

});

}

return points;

}

// Simple moving average forecasting: predict next demand as average of last n points

function simpleMovingAverage(data, windowSize = 3) {

const forecasts = [];

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

if (i < windowSize - 1) {

forecasts.push(null);

} else {

let sum = 0;

for (let j = i - windowSize + 1; j <= i; j++) {

sum += data[j];

}

forecasts.push(sum / windowSize);

}

}

return forecasts;

}

// Calculate Euclidean distance between two points

function distance(p1, p2) {

return Math.sqrt((p1.x - p2.x) \*\* 2 + (p1.y - p2.y) \*\* 2);

}

// Calculate total length of route (circular)

function routeDistance(points, route) {

let dist = 0;

for (let i = 0; i < route.length - 1; i++) {

dist += distance(points[route[i]], points[route[i + 1]]);

}

// Return to start

dist += distance(points[route[route.length - 1]], points[route[0]]);

return dist;

}

// Genetic Algorithm for TSP (simple implementation)

function geneticTSP(points, populationSize = 100, generations = 200) {

// Create initial population (random routes)

function createRoute() {

const route = [];

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

route.push(i);

}

for (let i = route.length - 1; i > 0; i--) {

const j = Math.floor(Math.random() \* (i + 1));

[route[i], route[j]] = [route[j], route[i]];

}

return route;

}

let population = [];

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

population.push(createRoute());

}

// Fitness = inverse of route distance

function fitness(route) {

return 1 / routeDistance(points, route);

}

// Selection by tournament

function select(pop) {

const tournamentSize = 5;

let best = null;

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

const candidate = pop[randomInt(0, pop.length - 1)];

if (best === null || fitness(candidate) > fitness(best)) {

best = candidate;

}

}

return best;

}

// Order crossover

function crossover(parent1, parent2) {

const start = randomInt(0, parent1.length - 2);

const end = randomInt(start + 1, parent1.length - 1);

const child = Array(parent1.length).fill(null);

for (let i = start; i <= end; i++) {

child[i] = parent1[i];

}

let p2Index = 0;

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

if (child[i] === null) {

while (child.includes(parent2[p2Index])) {

p2Index++;

}

child[i] = parent2[p2Index];

p2Index++;

}

}

return child;

}

// Mutation: swap two cities

function mutate(route, mutationRate = 0.1) {

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

if (Math.random() < mutationRate) {

const j = randomInt(0, route.length - 1);

[route[i], route[j]] = [route[j], route[i]];

}

}

}

// GA main loop

for (let g = 0; g < generations; g++) {

const newPopulation = [];

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

const parent1 = select(population);

const parent2 = select(population);

let child = crossover(parent1, parent2);

mutate(child);

newPopulation.push(child);

}

population = newPopulation;

}

// Find best route

let bestRoute = population[0];

let bestFit = fitness(bestRoute);

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

const fit = fitness(population[i]);

if (fit > bestFit) {

bestFit = fit;

bestRoute = population[i];

}

}

return bestRoute;

}

// Drawing route + points on canvas

function drawRoute(points, route) {

const canvas = document.getElementById('routeCanvas');

const ctx = canvas.getContext('2d');

ctx.clearRect(0, 0, canvas.width, canvas.height);

// Clear background white with radius corners effect

ctx.fillStyle = '#f9fafb';

ctx.fillRect(0, 0, canvas.width, canvas.height);

// Draw edges

ctx.strokeStyle = '#2563eb';

ctx.lineWidth = 3;

ctx.lineJoin = 'round';

ctx.beginPath();

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

const curr = points[route[i]];

const next = points[route[(i + 1) % route.length]];

if (i === 0) ctx.moveTo(curr.x, curr.y);

ctx.lineTo(next.x, next.y);

}

ctx.stroke();

// Draw points

points.forEach(p => {

// Outer circle for demand size (scaled)

const radius = 7 + p.demand \* 0.6;

const gradient = ctx.createRadialGradient(p.x, p.y, radius \* 0.4, p.x, p.y, radius);

gradient.addColorStop(0, '#60a5fa');

gradient.addColorStop(1, '#1e40af');

ctx.fillStyle = gradient;

ctx.beginPath();

ctx.arc(p.x, p.y, radius, 0, 2 \* Math.PI);

ctx.fill();

// Inner white circle

ctx.fillStyle = 'white';

ctx.beginPath();

ctx.arc(p.x, p.y, radius \* 0.45, 0, 2 \* Math.PI);

ctx.fill();

// Demand value text

ctx.fillStyle = '#1e293b';

ctx.font = 'bold 12px Segoe UI, Tahoma, Geneva, Verdana, sans-serif';

ctx.textAlign = 'center';

ctx.textBaseline = 'middle';

ctx.fillText(p.demand, p.x, p.y);

});

}

// Application state variables

let demandPoints = [];

let forecastValues = [];

let optimizedRoute = [];

// DOM Elements

const demandCountInput = document.getElementById('demandCount');

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

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

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

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

// Generate demand data and forecast

function generateDemand() {

const count = Math.min(12, Math.max(2, parseInt(demandCountInput.value) || 6));

demandCountInput.value = count;

demandPoints = generateDemandPoints(count);

// Extract pure demand values array

const demandValues = demandPoints.map(p => p.demand);

// Calculate simple moving average forecast

forecastValues = simpleMovingAverage(demandValues, 3);

// Prepare forecast output string for display

let forecastText = 'Point | Demand | 3-Point Moving Avg Forecast\n';

forecastText += '-------------------------------------\n';

demandPoints.forEach((p, i) => {

let fVal = forecastValues[i];

const displayVal = fVal === null ? '-' : fVal.toFixed(2);

forecastText += `${p.id + 1}\t | ${p.demand}\t | ${displayVal}\n`;

});

forecastOutput.textContent = forecastText;

// Clear any previous route info and canvas

routeOutput.textContent = 'Optimized route will appear here after clicking the button.';

optimizedRoute = [];

drawRoute(demandPoints, []);

}

// Optimize route and display

function optimizeRoute() {

if (demandPoints.length === 0) {

routeOutput.textContent = 'Generate demand data first!';

return;

}

optimizedRoute = geneticTSP(demandPoints, 200, 400);

// Show route order by point IDs (1-based)

const orderStr = optimizedRoute.map(i => i + 1).join(' → ') + ' → ' + (optimizedRoute[0] + 1);

const dist = routeDistance(demandPoints, optimizedRoute).toFixed(2);

routeOutput.textContent = `Optimized Route Order:\n${orderStr}\nTotal Distance: ${dist}`;

// Draw on canvas

drawRoute(demandPoints, optimizedRoute);

}

// Event listeners

generateDemandBtn.addEventListener('click', generateDemand);

optimizeRouteBtn.addEventListener('click', optimizeRoute);

// Initialize on page load

generateDemand();

})();

</script>

</body>

</html>

</content>

</create\_file>