<!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>