```
import React, { useState, useEffect } from 'react';
import { Network, Zap, TrendingUp, Settings, Info, Plus, Play, Award, Leaf } from

// Data Models
const initialNodes = [
  { id: 'ny', name: 'New York', region: 'US-East', lat: 40.7, lon: -74.0, energyP
  { id: 'sf', name: 'San Francisco', region: 'US-West', lat: 37.8, lon: -122.4, e
  { id: 'ld', name: 'London', region: 'EU-West', lat: 51.5, lon: -0.1, energyPric
  { id: 'fr', name: 'Frankfurt', region: 'EU-Central', lat: 50.1, lon: 8.7, energ
  { id: 'sg', name: 'Singapore', region: 'APAC-SE', lat: 1.3, lon: 103.8, energyP
  { id: 'tk', name: 'Tokyo', region: 'APAC-NE', lat: 35.7, lon: 139.7, energyPric
  { id: 'sy', name: 'Sydney', region: 'APAC-AU', lat: -33.9, lon: 151.2, energyPr
  { id: 'db', name: 'Dubai', region: 'ME', lat: 25.3, lon: 55.3, energyPricePerWh
];

const initialProviders = [
  { id: 'p1', name: 'HyperNet', region: 'Global', email: 'ops@hypernet.io', relia
  { id: 'p2', name: 'OceanLink', region: 'Trans-Atlantic', email: 'support@oceanl
  { id: 'p3', name: 'PhotonPath', region: 'APAC', email: 'contact@photonpath.com'
  { id: 'p4', name: 'VelocityGrid', region: 'Americas', email: 'info@velocitygrid
];

const generateInitialLinks = () => [
  { id: 'l1', from: 'ny', to: 'ld', latencyMs: 65, throughputGbps: 100, energyWhP
  { id: 'l2', from: 'ny', to: 'sf', latencyMs: 42, throughputGbps: 200, energyWhP
  { id: 'l3', from: 'sf', to: 'tk', latencyMs: 95, throughputGbps: 150, energyWhP
  { id: 'l4', from: 'sf', to: 'sg', latencyMs: 180, throughputGbps: 80, energyWhP
  { id: 'l5', from: 'ld', to: 'fr', latencyMs: 12, throughputGbps: 400, energyWhP
  { id: 'l6', from: 'ld', to: 'db', latencyMs: 85, throughputGbps: 120, energyWhP
  { id: 'l7', from: 'fr', to: 'sg', latencyMs: 155, throughputGbps: 100, energyWh
  { id: 'l8', from: 'fr', to: 'tk', latencyMs: 220, throughputGbps: 90, energyWhP
  { id: 'l9', from: 'tk', to: 'sg', latencyMs: 68, throughputGbps: 180, energyWhP
  { id: 'l10', from: 'tk', to: 'sy', latencyMs: 115, throughputGbps: 100, energyW
  { id: 'l11', from: 'sg', to: 'sy', latencyMs: 95, throughputGbps: 120, energyWh
  { id: 'l12', from: 'sg', to: 'db', latencyMs: 105, throughputGbps: 80, energyWh
  { id: 'l13', from: 'db', to: 'fr', latencyMs: 78, throughputGbps: 110, energyWh
  { id: 'l14', from: 'ny', to: 'fr', latencyMs: 88, throughputGbps: 150, energyWh
  { id: 'l15', from: 'ld', to: 'tk', latencyMs: 240, throughputGbps: 70, energyWh
  { id: 'l16', from: 'sf', to: 'sy', latencyMs: 190, throughputGbps: 85, energyWh
  { id: 'l17', from: 'ny', to: 'db', latencyMs: 145, throughputGbps: 90, energyWh
  { id: 'l18', from: 'tk', to: 'db', latencyMs: 125, throughputGbps: 75, energyWh
  { id: 'l19', from: 'sy', to: 'db', latencyMs: 170, throughputGbps: 60, energyWh
  { id: 'l20', from: 'fr', to: 'sy', latencyMs: 280, throughputGbps: 65, energyWh
];
```

```
const AetherionApp = () => {
  const [activeTab, setActiveTab] = useState('buyer');
  const [nodes, setNodes] = useState(initialNodes);
  const [links, setLinks] = useState(generateInitialLinks());
  const [providers, setProviders] = useState(initialProviders);

  // Buyer state
  const [fromNode, setFromNode] = useState('ny');
  const [toNode, setToNode] = useState('tk');
  const [alphaWeight, setAlphaWeight] = useState(0.5); // latency
  const [betaWeight, setBetaWeight] = useState(0.3); // energy
  const [gammaWeight, setGammaWeight] = useState(0.2); // cost
  const [computedRoute, setComputedRoute] = useState(null);
  const [isSimulating, setIsSimulating] = useState(true);

  // Provider state
  const [newLink, setNewLink] = useState({
    from: 'ny',
    to: 'ld',
    latencyMs: 50,
    throughputGbps: 100,
    energyWhPerGB: 2.0,
    carbonIntensity: 80,
    pricePerGB: 0.01,
    providerId: 'p1'
  });

  // Simulation - fluctuate congestion
  useEffect(() => {
    if (!isSimulating) return;

    const interval = setInterval(() => {
      setLinks(prevLinks => prevLinks.map(link => ({
        ...link,
        congestion: Math.max(0.05, Math.min(0.95, link.congestion + (Math.random(
      })));
    }, 3000);

    return () => clearInterval(interval);
  }, [isSimulating]);

  // Dijkstra pathfinding with custom scoring
  const findOptimalRoute = (fromId, toId, alpha, beta, gamma) => {
    const distances = {};
    const previous = {};
    const unvisited = new Set();
```

```
nodes.forEach(node => {
  distances[node.id] = Infinity;
  previous[node.id] = null;
  unvisited.add(node.id);
});

distances[fromId] = 0;

// Build adjacency list
const adjacency = {};
links.forEach(link => {
  if (!adjacency[link.from]) adjacency[link.from] = [];
  if (!adjacency[link.to]) adjacency[link.to] = [];
  adjacency[link.from].push({ ...link, target: link.to });
  adjacency[link.to].push({ ...link, target: link.from, from: link.to, to: li
});

while (unvisited.size > 0) {
  let current = null;
  let minDist = Infinity;

  for (let nodeId of unvisited) {
    if (distances[nodeId] < minDist) {
      minDist = distances[nodeId];
      current = nodeId;
    }
  }

  if (current === null || current === toId) break;

  unvisited.delete(current);

  const neighbors = adjacency[current] || [];

  for (let link of neighbors) {
    if (!unvisited.has(link.target)) continue;

    const effectiveLatency = link.latencyMs * (1 + link.congestion);
    const node = nodes.find(n => n.id === link.from);
    const energyCost = link.energyWhPerGB * (node?.energyPricePerWh || 0.15)
    const carbonTax = link.carbonIntensity * 0.0001;
    const effectiveCost = link.pricePerGB + energyCost + carbonTax;

    // Scoring function (lower is better, so we invert for speed/efficiency)
    const score = alpha * effectiveLatency +
                 beta * link.energyWhPerGB * 100 +
                 gamma * effectiveCost * 1000;

    const alt = distances[current] + score;

    if (alt < distances[link.target]) {
      distances[link.target] = alt;
      previous[link.target] = { nodeId: current, link };
    }
  }
}

// Reconstruct path
const path = [];
const usedLinks = [];
let current = toId;

while (previous[current]) {
  path.unshift(current);
  usedLinks.unshift(previous[current].link);
  current = previous[current].nodeId;
}
path.unshift(fromId);

if (path.length === 1) return null;

// Calculate totals
let totalLatency = 0;
let totalCost = 0;
let totalEnergy = 0;
let totalCarbon = 0;

usedLinks.forEach(link => {
  const effectiveLatency = link.latencyMs * (1 + link.congestion);
  totalLatency += effectiveLatency;

  const node = nodes.find(n => n.id === link.from);
  const energyCost = link.energyWhPerGB * (node?.energyPricePerWh || 0.15) *
  const carbonTax = link.carbonIntensity * 0.0001;

  totalCost += link.pricePerGB + energyCost + carbonTax;
  totalEnergy += link.energyWhPerGB;
  totalCarbon += link.carbonIntensity;
});

return {
  path,
  usedLinks,
    totalLatency: Math.round(totalLatency),
    totalCostUSD: totalCost,
    energyWhPerGB: totalEnergy,
    carbonPerGB: totalCarbon,
    score: distances[toId]
  };
};

const handleComputeRoute = () => {
  const route = findOptimalRoute(fromNode, toNode, alphaWeight, betaWeight, gam
  setComputedRoute(route);
};

const handleAddLink = () => {
  const link = {
    ...newLink,
    id: `l${links.length + 1}`,
    congestion: Math.random() * 0.5
  };
  setLinks([...links, link]);
};

const getNodePosition = (node) => {
  const mapWidth = 800;
  const mapHeight = 400;
  const x = ((node.lon + 180) / 360) * mapWidth;
  const y = ((90 - node.lat) / 180) * mapHeight;
  return { x, y };
};

const getCongestionColor = (congestion) => {
  const r = Math.round(congestion * 255);
  const g = Math.round((1 - congestion) * 255);
  return `rgb(${r}, ${g}, 50)`;
};

// Analytics
const analytics = {
  avgLatency: Math.round(links.reduce((sum, l) => sum + l.latencyMs, 0) / links
  avgEnergy: (links.reduce((sum, l) => sum + l.energyWhPerGB, 0) / links.length
  avgCongestion: (links.reduce((sum, l) => sum + l.congestion, 0) / links.lengt
  fastestRoute: computedRoute ? `${nodes.find(n => n.id === computedRoute.path[
};

return (
  <div className="min-h-screen bg-gradient-to-br from-slate-900 via-purple-900
    {/* Header */}
```

```
<div className="max-w-7xl mx-auto mb-8">
  <div className="flex items-center justify-between mb-2">
    <div className="flex items-center gap-3">
      <Network className="w-10 h-10 text-cyan-400" />
      <div>
        <h1 className="text-4xl font-bold bg-gradient-to-r from-cyan-400 to
          Aetherion
        </h1>
        <p className="text-gray-400 text-sm">Network Latency Exchange</p>
      </div>
    </div>
    <button
      onClick={() => setIsSimulating(!isSimulating)}
      className={`px-4 py-2 rounded-lg flex items-center gap-2 ${isSimulati
    >
      <Play className="w-4 h-4" />
      {isSimulating ? 'Live' : 'Paused'}
    </button>
  </div>

  {/* Tabs */}
  <div className="flex gap-2 mt-6">
    {[
      { id: 'buyer', label: 'Route Builder', icon: Zap },
      { id: 'provider', label: 'Provider', icon: Plus },
      { id: 'analytics', label: 'Analytics', icon: TrendingUp },
      { id: 'readme', label: 'About', icon: Info }
    ].map(tab => (
      <button
        key={tab.id}
        onClick={() => setActiveTab(tab.id)}
        className={`px-4 py-2 rounded-t-lg flex items-center gap-2 transiti
          activeTab === tab.id ? 'bg-slate-800 text-cyan-400' : 'bg-slate-8
        }`}
      >
        <tab.icon className="w-4 h-4" />
        {tab.label}
      </button>
    ))}
  </div>
</div>

<div className="max-w-7xl mx-auto">
  {/* Buyer Dashboard */}
  {activeTab === 'buyer' && (
    <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
      {/* Controls */}
```

```
<div className="space-y-6">
  <div className="bg-slate-800/50 backdrop-blur rounded-lg p-6 border
    <h2 className="text-xl font-semibold mb-4 flex items-center gap-2
      <Settings className="w-5 h-5 text-cyan-400" />
      Route Configuration
    </h2>

    <div className="space-y-4">
      <div>
        <label className="block text-sm text-gray-400 mb-2">Source No
        <select
          value={fromNode}
          onChange={e => setFromNode(e.target.value)}
          className="w-full bg-slate-700 border border-slate-600 roun
        >
          {nodes.map(node => (
            <option key={node.id} value={node.id}>{node.name}</option
          ))}
        </select>
      </div>

      <div>
        <label className="block text-sm text-gray-400 mb-2">Destinati
        <select
          value={toNode}
          onChange={e => setToNode(e.target.value)}
          className="w-full bg-slate-700 border border-slate-600 roun
        >
          {nodes.map(node => (
            <option key={node.id} value={node.id}>{node.name}</option
          ))}
        </select>
      </div>

      <div>
        <label className="block text-sm text-gray-400 mb-2">
          Priority: Latency {(alphaWeight * 100).toFixed(0)}%
        </label>
        <input
          type="range"
          min="0"
          max="1"
          step="0.1"
          value={alphaWeight}
          onChange={e => setAlphaWeight(parseFloat(e.target.value))}
          className="w-full"
        />
        </div>

        <div>
          <label className="block text-sm text-gray-400 mb-2">
            Priority: Energy {(betaWeight * 100).toFixed(0)}%
          </label>
          <input
            type="range"
            min="0"
            max="1"
            step="0.1"
            value={betaWeight}
            onChange={e => setBetaWeight(parseFloat(e.target.value))}
            className="w-full"
          />
        </div>

        <div>
          <label className="block text-sm text-gray-400 mb-2">
            Priority: Cost {(gammaWeight * 100).toFixed(0)}%
          </label>
          <input
            type="range"
            min="0"
            max="1"
            step="0.1"
            value={gammaWeight}
            onChange={e => setGammaWeight(parseFloat(e.target.value))}
            className="w-full"
          />
        </div>

        <button
          onClick={handleComputeRoute}
          className="w-full bg-gradient-to-r from-cyan-600 to-purple-60
        >
          Compute Optimal Route
        </button>
      </div>
    </div>

    {/* Results */}
    {computedRoute && (
      <div className="bg-slate-800/50 backdrop-blur rounded-lg p-6 bord
        <h2 className="text-xl font-semibold mb-4 flex items-center gap
          <Award className="w-5 h-5 text-green-400" />
          Route Solution
      </h2>

      <div className="space-y-3">
        <div className="flex justify-between items-center">
          <span className="text-gray-400">Path</span>
          <span className="text-cyan-400 font-mono text-sm">
            {computedRoute.path.map(id => nodes.find(n => n.id === id
          </span>
        </div>

        <div className="grid grid-cols-2 gap-4 mt-4">
          <div className="bg-slate-700/50 rounded p-3">
            <div className="text-gray-400 text-xs mb-1">Total Latency
            <div className="text-2xl font-bold text-cyan-400">{comput
          </div>

          <div className="bg-slate-700/50 rounded p-3">
            <div className="text-gray-400 text-xs mb-1">Total Cost</d
            <div className="text-2xl font-bold text-green-400">${comp
          </div>

          <div className="bg-slate-700/50 rounded p-3">
            <div className="text-gray-400 text-xs mb-1">Energy</div>
            <div className="text-2xl font-bold text-yellow-400">{comp
          </div>

          <div className="bg-slate-700/50 rounded p-3">
            <div className="text-gray-400 text-xs mb-1">Carbon</div>
            <div className="text-2xl font-bold text-purple-400">{comp
          </div>
        </div>
      </div>
    </div>
  )}
</div>

{/* Map */}
<div className="bg-slate-800/50 backdrop-blur rounded-lg p-6 border b
  <h2 className="text-xl font-semibold mb-4">Global Network Map</h2>
  <svg viewBox="0 0 800 400" className="w-full bg-slate-900/50 rounde
    {/* Links */}
    {links.map(link => {
      const fromPos = getNodePosition(nodes.find(n => n.id === link.f
      const toPos = getNodePosition(nodes.find(n => n.id === link.to)
      const isInRoute = computedRoute?.usedLinks.some(l => l.id === l

      return (
        <line
          key={link.id}
          x1={fromPos.x}
          y1={fromPos.y}
          x2={toPos.x}
          y2={toPos.y}
          stroke={isInRoute ? '#22d3ee' : getCongestionColor(link.con
          strokeWidth={isInRoute ? 3 : 1.5}
          opacity={isInRoute ? 1 : 0.4}
        />
      );
    })}

    {/* Nodes */}
    {nodes.map(node => {
      const pos = getNodePosition(node);
      const isInRoute = computedRoute?.path.includes(node.id);

      return (
        <g key={node.id}>
          <circle
            cx={pos.x}
            cy={pos.y}
            r={isInRoute ? 8 : 6}
            fill={isInRoute ? '#22d3ee' : '#8b5cf6'}
            stroke="#fff"
            strokeWidth={isInRoute ? 2 : 1}
          />
          <text
            x={pos.x}
            y={pos.y - 12}
            textAnchor="middle"
            fill="#fff"
            fontSize="10"
            fontWeight={isInRoute ? 'bold' : 'normal'}
          >
            {node.name}
          </text>
        </g>
      );
    })}
  </svg>

  <div className="mt-4 flex items-center gap-4 text-xs">
    <div className="flex items-center gap-2">
      <div className="w-3 h-3 rounded-full" style={{ backgroundColor:
      <span className="text-gray-400">Low Congestion</span>
```

```
          </div>
          <div className="flex items-center gap-2">
            <div className="w-3 h-3 rounded-full" style={{ backgroundColor:
            <span className="text-gray-400">High Congestion</span>
          </div>
          <div className="flex items-center gap-2">
            <div className="w-3 h-3 rounded-full bg-cyan-400"></div>
            <span className="text-gray-400">Selected Route</span>
          </div>
        </div>
      </div>
    </div>
  )}

  {/* Provider Dashboard */}
  {activeTab === 'provider' && (
    <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
      <div className="bg-slate-800/50 backdrop-blur rounded-lg p-6 border b
        <h2 className="text-xl font-semibold mb-4">Register New Link</h2>

        <div className="space-y-4">
          <div className="grid grid-cols-2 gap-4">
            <div>
              <label className="block text-sm text-gray-400 mb-2">From</lab
              <select
                value={newLink.from}
                onChange={e => setNewLink({ ...newLink, from: e.target.valu
                className="w-full bg-slate-700 border border-slate-600 roun
              >
                {nodes.map(n => <option key={n.id} value={n.id}>{n.name}</o
              </select>
            </div>

            <div>
              <label className="block text-sm text-gray-400 mb-2">To</label
              <select
                value={newLink.to}
                onChange={e => setNewLink({ ...newLink, to: e.target.value
                className="w-full bg-slate-700 border border-slate-600 roun
              >
                {nodes.map(n => <option key={n.id} value={n.id}>{n.name}</o
              </select>
            </div>
          </div>

          <div>
            <label className="block text-sm text-gray-400 mb-2">Latency (ms
      <input
        type="number"
        value={newLink.latencyMs}
        onChange={e => setNewLink({ ...newLink, latencyMs: parseFloat
        className="w-full bg-slate-700 border border-slate-600 rounde
      />
    </div>

    <div>
      <label className="block text-sm text-gray-400 mb-2">Throughput
      <input
        type="number"
        value={newLink.throughputGbps}
        onChange={e => setNewLink({ ...newLink, throughputGbps: parse
        className="w-full bg-slate-700 border border-slate-600 rounde
      />
    </div>

    <div>
      <label className="block text-sm text-gray-400 mb-2">Energy (Wh/
      <input
        type="number"
        step="0.1"
        value={newLink.energyWhPerGB}
        onChange={e => setNewLink({ ...newLink, energyWhPerGB: parseF
        className="w-full bg-slate-700 border border-slate-600 rounde
      />
    </div>

    <div>
      <label className="block text-sm text-gray-400 mb-2">Price ($/GB
      <input
        type="number"
        step="0.001"
        value={newLink.pricePerGB}
        onChange={e => setNewLink({ ...newLink, pricePerGB: parseFloa
        className="w-full bg-slate-700 border border-slate-600 rounde
      />
    </div>

    <div>
      <label className="block text-sm text-gray-400 mb-2">Provider</l
      <select
        value={newLink.providerId}
        onChange={e => setNewLink({ ...newLink, providerId: e.target.
        className="w-full bg-slate-700 border border-slate-600 rounde
      >
```

```
        {providers.map(p => <option key={p.id} value={p.id}>{p.name}<
      </select>
    </div>

    <button
      onClick={handleAddLink}
      className="w-full bg-gradient-to-r from-purple-600 to-pink-600
    >
      Add Network Link
    </button>
  </div>
</div>

<div className="bg-slate-800/50 backdrop-blur rounded-lg p-6 border b
  <h2 className="text-xl font-semibold mb-4">Active Providers</h2>
  <div className="space-y-3">
    {providers.map(provider => {
      const providerLinks = links.filter(l => l.providerId === provid
      const avgLatency = providerLinks.length > 0
        ? Math.round(providerLinks.reduce((sum, l) => sum + l.latency
        : 0;

      return (
        <div key={provider.id} className="bg-slate-700/50 rounded p-4
          <div className="flex justify-between items-start mb-2">
            <div>
              <div className="font-semibold">{provider.name}</div>
              <div className="text-xs text-gray-400">{provider.region
            </div>
            <div className="text-right">
              <div className="text-xs text-gray-400">Reliability</div
              <div className="text-green-400 font-semibold">{(provide
            </div>
          </div>
          <div className="grid grid-cols-2 gap-2 text-xs mt-3">
            <div>
              <span className="text-gray-400">Links:</span>
              <span className="ml-2 text-cyan-400">{providerLinks.len
            </div>
            <div>
              <span className="text-gray-400">Avg Latency:</span>
              <span className="ml-2 text-cyan-400">{avgLatency}ms</sp
            </div>
          </div>
        </div>
      );
    })}
```

```
          </div>
        </div>
      </div>
    )}

    {/* Analytics */}
    {activeTab === 'analytics' && (
      <div>
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6
          <div className="bg-gradient-to-br from-cyan-600/20 to-cyan-800/20 b
            <div className="flex items-center justify-between mb-2">
              <Zap className="w-8 h-8 text-cyan-400" />
            </div>
            <div className="text-3xl font-bold text-cyan-400">{analytics.avgL
            <div className="text-sm text-gray-400">Avg Network Latency</div>
          </div>

          <div className="bg-gradient-to-br from-green-600/20 to-green-800/20
            <div className="flex items-center justify-between mb-2">
              <Leaf className="w-8 h-8 text-green-400" />
            </div>
            <div className="text-3xl font-bold text-green-400">{analytics.avg
            <div className="text-sm text-gray-400">Avg Wh/GB</div>
          </div>

          <div className="bg-gradient-to-br from-yellow-600/20 to-yellow-800/
            <div className="flex items-center justify-between mb-2">
              <TrendingUp className="w-8 h-8 text-yellow-400" />
            </div>
            <div className="text-3xl font-bold text-yellow-400">{(analytics.a
            <div className="text-sm text-gray-400">Avg Congestion</div>
          </div>

          <div className="bg-gradient-to-br from-purple-600/20 to-purple-800/
            <div className="flex items-center justify-between mb-2">
              <Network className="w-8 h-8 text-purple-400" />
            </div>
            <div className="text-3xl font-bold text-purple-400">{links.length
            <div className="text-sm text-gray-400">Active Links</div>
          </div>
        </div>

        <div className="bg-slate-800/50 backdrop-blur rounded-lg p-6 border b
          <h2 className="text-xl font-semibold mb-4">Network Performance</h2>
          <div className="space-y-2">
            {links.slice(0, 10).map(link => {
              const fromNode = nodes.find(n => n.id === link.from);
          const toNode = nodes.find(n => n.id === link.to);
          const score = 100 - (link.latencyMs / 3 + link.congestion * 50)

          return (
            <div key={link.id} className="bg-slate-700/50 rounded p-3">
              <div className="flex justify-between items-center mb-2">
                <span className="text-sm">{fromNode?.name} → {toNode?.nam
                <span className="text-xs text-cyan-400">{link.latencyMs}m
              </div>
              <div className="w-full bg-slate-600 rounded-full h-2">
                <div
                  className="bg-gradient-to-r from-cyan-400 to-purple-400
                  style={{ width: `${Math.max(0, Math.min(100, score))}%`
                ></div>
              </div>
            </div>
          );
        })}
      </div>
    </div>
  </div>
)}

{/* README */}
{activeTab === 'readme' && (
  <div className="bg-slate-800/50 backdrop-blur rounded-lg p-8 border bor
    <h2 className="text-3xl font-bold mb-6 bg-gradient-to-r from-cyan-400
      About Aetherion
    </h2>

    <div className="space-y-6 text-gray-300">
      <section>
        <h3 className="text-xl font-semibold text-cyan-400 mb-2">What is
        <p>Aetherion is a network latency exchange - a marketplace where
      </section>

      <section>
        <h3 className="text-xl font-semibold text-cyan-400 mb-2">Why Late
        <p>Every network route has multiple dimensions of cost. Physical
      </section>

      <section>
        <h3 className="text-xl font-semibold text-cyan-400 mb-2">How Buye
        <p>Navigate to the Route Builder, select source and destination n
      </section>

      <section>
```

```
              <h3 className="text-xl font-semibold text-cyan-400 mb-2">How Prov
              <p>Network infrastructure providers can register new connections
            </section>

            <section>
              <h3 className="text-xl font-semibold text-cyan-400 mb-2">Pricing
              <p className="mb-2">Routes are scored using:</p>
              <code className="block bg-slate-900 p-3 rounded text-sm font-mono
                score = α×latency + β×energy + γ×cost
              </code>
              <p className="mt-2">Where effective costs include bandwidth prici
            </section>

            <section>
              <h3 className="text-xl font-semibold text-cyan-400 mb-2">Technica
              <p>This is a demonstration with simulated data. All network metri
            </section>
          </div>
        </div>
      )}
    </div>
  </div>
  );
};

export default AetherionApp;
```