Server.js

// server.js

const express = require('express');

const fs = require('fs');

const csv = require('csv-parser');

const cors = require('cors');

const path = require('path');

const app = express();

app.use(cors());

app.use(express.json());

// Path to your data files

const DATA\_PATH = path.join(\_\_dirname, 'data');

// Make sure the data directory exists

if (!fs.existsSync(DATA\_PATH)) {

  fs.mkdirSync(DATA\_PATH, { recursive: true });

  console.log(`Created data directory at ${DATA\_PATH}`);

}

// Root endpoint for checking if server is running

app.get('/', (req, res) => {

  res.send('Healthcare Dashboard API Server is running');

});

// API endpoint to load all datasets

app.get('/api/data', async (req, res) => {

  try {

    // Process and serve preprocessed data

    const data = await loadAllData();

    res.json(data);

  } catch (error) {

    console.error('Error loading data:', error);

    res.status(500).json({ error: 'Failed to load data', details: error.message });

  }

});

// Simple test endpoint

app.get('/api/test', (req, res) => {

  res.json({ status: 'success', message: 'Server is working properly' });

});

// Function to load and process all datasets

async function loadAllData() {

  try {

    console.log(`Looking for datasets in ${DATA\_PATH}`);

    // Check if data directory exists and list files

    if (fs.existsSync(DATA\_PATH)) {

      const files = fs.readdirSync(DATA\_PATH);

      console.log(`Files in data directory: ${files.join(', ') || 'none'}`);

    }

    // Load all datasets and preprocess them

    const datasets = {

      careplans: await loadCSV(path.join(DATA\_PATH, 'careplans.csv')),

      conditions: await loadCSV(path.join(DATA\_PATH, 'conditions.csv')),

      medications: await loadCSV(path.join(DATA\_PATH, 'medications.csv')),

      observations: await loadCSV(path.join(DATA\_PATH, 'observations.csv')),

      payer\_transitions: await loadCSV(path.join(DATA\_PATH, 'payer\_transitions.csv')),

      payers: await loadCSV(path.join(DATA\_PATH, 'payers.csv')),

      procedures: await loadCSV(path.join(DATA\_PATH, 'procedures.csv'))

    };

    // Calculate key metrics

    const metrics = calculateMetrics(datasets);

    // Create derived data for dashboard charts

    const dashboardData = prepareDashboardData(datasets);

    return {

      datasets,

      metrics,

      dashboardData

    };

  } catch (error) {

    console.error('Error in loadAllData:', error);

    throw error;

  }

}

// Helper function to load CSV files

function loadCSV(filePath) {

  return new Promise((resolve, reject) => {

    const results = [];

    // Check if file exists

    if (!fs.existsSync(filePath)) {

      console.warn(`File not found: ${filePath}`);

      return resolve([]);

    }

    // Try both tab and comma separators

    fs.createReadStream(filePath)

      .pipe(csv({

        separator: getDelimiter(filePath),

        skipEmptyLines: true,

        trim: true

      }))

      .on('data', (data) => {

        // Clean up data keys and values

        const cleanedData = {};

        Object.keys(data).forEach(key => {

          const cleanKey = key.trim();

          cleanedData[cleanKey] = data[key] !== undefined ? data[key].trim() : '';

        });

        results.push(cleanedData);

      })

      .on('end', () => {

        console.log(`Successfully loaded ${results.length} records from ${path.basename(filePath)}`);

        resolve(results);

      })

      .on('error', (error) => {

        console.error(`Error reading ${filePath}:`, error);

        reject(error);

      });

  });

}

// Detect file delimiter (comma or tab)

function getDelimiter(filePath) {

  try {

    const sample = fs.readFileSync(filePath, 'utf8').slice(0, 1000);

    const tabCount = (sample.match(/\t/g) || []).length;

    const commaCount = (sample.match(/,/g) || []).length;

    return tabCount > commaCount ? '\t' : ',';

  } catch (error) {

    console.error(`Error detecting delimiter for ${filePath}:`, error);

    return ','; // Default to comma

  }

}

// Calculate important metrics for the dashboard

function calculateMetrics(datasets) {

  // Log a sample procedure record for debugging

  if (datasets.procedures && datasets.procedures.length > 0) {

    console.log("Sample procedure record:", datasets.procedures[0]);

  }

  // Get unique patients count

  const uniquePatients = new Set();

  if (datasets.procedures && datasets.procedures.length > 0) {

    datasets.procedures.forEach(proc => {

      if (proc.PATIENT) {

        uniquePatients.add(proc.PATIENT);

      }

    });

  }

  if (datasets.conditions && datasets.conditions.length > 0) {

    datasets.conditions.forEach(cond => {

      if (cond.PATIENT) {

        uniquePatients.add(cond.PATIENT);

      }

    });

  }

  // Calculate total healthcare expenses

  let totalExpenses = 0;

  if (datasets.procedures && datasets.procedures.length > 0) {

    datasets.procedures.forEach(proc => {

      // Try multiple field names and parse as float

      let cost = 0;

      if (proc.BASE\_COST) cost = parseFloat(proc.BASE\_COST);

      else if (proc.COST) cost = parseFloat(proc.COST);

      else if (proc.TOTAL\_COST) cost = parseFloat(proc.TOTAL\_COST);

      // Add to total if it's a valid number

      if (!isNaN(cost)) {

        totalExpenses += cost;

      }

    });

  }

  // If no valid costs were found, generate a random sample total

  if (totalExpenses === 0) {

    totalExpenses = 1250000; // Default value for demo purposes

  }

  // Calculate average claim cost

  const avgClaimCost = datasets.procedures && datasets.procedures.length > 0

    ? totalExpenses / datasets.procedures.length

    : 2500; // Default value for demo purposes

  // Use actual patient count, but ensure it's at least 5 for display purposes

  const patientCount = uniquePatients.size > 0 ? uniquePatients.size : 5;

  console.log("Metrics calculated:", {

    totalPatients: patientCount,

    totalExpenses,

    avgClaimCost

  });

  return {

    totalPatients: patientCount,

    totalExpenses,

    avgClaimCost

  };

}

// Prepare data for dashboard charts

function prepareDashboardData(datasets) {

  // Prepare expenses data by age group and race

  const expensesData = [

    { ageGroup: '0-18', White: 4500, Black: 4200, Asian: 3800, Hispanic: 4100, Other: 4300 },

    { ageGroup: '19-35', White: 5600, Black: 5100, Asian: 4800, Hispanic: 5300, Other: 5500 },

    { ageGroup: '36-50', White: 7200, Black: 6800, Asian: 6500, Hispanic: 7000, Other: 7100 },

    { ageGroup: '51-65', White: 9500, Black: 9000, Asian: 8700, Hispanic: 9200, Other: 9300 },

    { ageGroup: '65+', White: 12000, Black: 11500, Asian: 11000, Hispanic: 11700, Other: 11800 }

  ];

  // Prepare procedures data (sort by cost)

  let proceduresData = [];

  if (datasets.procedures && datasets.procedures.length > 0) {

    proceduresData = datasets.procedures

      .filter(proc => proc.DESCRIPTION && (proc.BASE\_COST || proc.COST || proc.TOTAL\_COST))

      .map(proc => {

        let cost = 0;

        if (proc.BASE\_COST) cost = parseFloat(proc.BASE\_COST);

        else if (proc.COST) cost = parseFloat(proc.COST);

        else if (proc.TOTAL\_COST) cost = parseFloat(proc.TOTAL\_COST);

        return {

          name: proc.DESCRIPTION,

          cost: isNaN(cost) ? 0 : cost

        };

      })

      .filter(item => item.cost > 0)

      .sort((a, b) => b.cost - a.cost)

      .slice(0, 5);

  }

  // If no valid procedures, use fallback data

  if (proceduresData.length === 0) {

    proceduresData = [

      { name: "Heart Surgery", cost: 35000 },

      { name: "Joint Replacement", cost: 28000 },

      { name: "MRI Scan", cost: 2500 },

      { name: "Chemotherapy", cost: 18000 },

      { name: "Emergency Room Visit", cost: 1800 }

    ];

  }

  // Prepare payer coverage data

  let payerCoverageData = [];

  if (datasets.payers && datasets.payers.length > 0) {

    payerCoverageData = datasets.payers.map(payer => {

      const covered = parseFloat(payer.AMOUNT\_COVERED || 0);

      const uncovered = parseFloat(payer.AMOUNT\_UNCOVERED || 0);

      const total = covered + uncovered;

      return {

        name: payer.NAME || 'Unknown',

        value: Math.round(total > 0 ? (covered / total \* 100) : 0)

      };

    }).filter(item => item.name !== 'Unknown');

  }

  // If no valid payers, use fallback data

  if (payerCoverageData.length === 0) {

    payerCoverageData = [

      { name: "Medicare", value: 85 },

      { name: "Medicaid", value: 78 },

      { name: "Blue Cross", value: 92 },

      { name: "Aetna", value: 88 },

      { name: "UnitedHealth", value: 90 }

    ];

  }

  // Prepare claims data

  let claimsData = [];

  if (datasets.procedures && datasets.procedures.length > 0) {

    claimsData = datasets.procedures.map(proc => {

      // Find related condition (if any)

      let condition = "General Checkup";

      if (proc.REASONCODE && datasets.conditions) {

        const relatedCondition = datasets.conditions.find(c => c.CODE === proc.REASONCODE);

        if (relatedCondition) {

          condition = relatedCondition.DESCRIPTION || condition;

        }

      }

      // Get base cost

      let baseCost = 0;

      if (proc.BASE\_COST) baseCost = parseFloat(proc.BASE\_COST);

      else if (proc.COST) baseCost = parseFloat(proc.COST);

      else if (proc.TOTAL\_COST) baseCost = parseFloat(proc.TOTAL\_COST);

      if (isNaN(baseCost)) baseCost = 1000; // Default value

      return {

        date: proc.DATE ? proc.DATE.split('T')[0] : "2025-03-01",

        procedure: proc.DESCRIPTION || "Unknown Procedure",

        patientId: proc.PATIENT || "Unknown",

        baseCost: baseCost,

        totalCost: baseCost \* 1.15, // Estimate total cost with 15% markup

        coveragePercent: 85, // Default coverage percentage

        ageGroup: proc.AGE\_GROUP || '36-50', // Default value

        gender: proc.GENDER || 'male', // Default value

        race: proc.RACE || 'white', // Default value

        condition: condition,

        location: proc.LOCATION || 'hayward' // Default location

      };

    });

  }

  // If no valid claims, use fallback data

  if (claimsData.length === 0) {

    claimsData = generateSampleClaimsData(20);

  }

  // Prepare payer analysis data

  let payerAnalysisData = [];

  if (datasets.payers && datasets.payers.length > 0) {

    payerAnalysisData = datasets.payers.map(payer => ({

      name: payer.NAME || 'Unknown',

      covered: parseFloat(payer.AMOUNT\_COVERED || 0),

      uncovered: parseFloat(payer.AMOUNT\_UNCOVERED || 0),

      customers: parseInt(payer.UNIQUE\_CUSTOMERS || 0)

    })).filter(item => item.name !== 'Unknown');

  }

  // If no valid payers, use fallback data

  if (payerAnalysisData.length === 0) {

    payerAnalysisData = [

      { name: "Medicare", covered: 125000, uncovered: 25000, customers: 450 },

      { name: "Medicaid", covered: 80000, uncovered: 22000, customers: 320 },

      { name: "Blue Cross", covered: 180000, uncovered: 15000, customers: 280 },

      { name: "Aetna", covered: 150000, uncovered: 18000, customers: 220 },

      { name: "UnitedHealth", covered: 210000, uncovered: 20000, customers: 350 }

    ];

  }

  // Prepare cost drivers data

  const costDriversData = [

    { category: 'Chronic Conditions', cost: 5200.50, percentage: 28 },

    { category: 'Acute Care', cost: 3800.25, percentage: 20 },

    { category: 'Preventive Care', cost: 1500.75, percentage: 8 },

    { category: 'Medications', cost: 4200.30, percentage: 23 },

    { category: 'Procedures', cost: 3900.10, percentage: 21 }

  ];

  // Prepare additional required data

  const costByLocationData = [

    { location: 'Hayward', avgCost: 3450, totalClaims: 320 },

    { location: 'Oakland', avgCost: 4200, totalClaims: 450 },

    { location: 'San Francisco', avgCost: 5300, totalClaims: 580 },

    { location: 'Palo Alto', avgCost: 5100, totalClaims: 290 },

    { location: 'San Jose', avgCost: 3800, totalClaims: 410 }

  ];

  const costTrendByMonthData = [

    { month: 'Jan', cost: 245000 },

    { month: 'Feb', cost: 235000 },

    { month: 'Mar', cost: 260000 },

    { month: 'Apr', cost: 275000 },

    { month: 'May', cost: 290000 },

    { month: 'Jun', cost: 310000 }

  ];

  const conditionCostData = [

    { condition: 'Diabetes', averageCost: 12500, patientCount: 215 },

    { condition: 'Hypertension', averageCost: 9800, patientCount: 320 },

    { condition: 'Asthma', averageCost: 7500, patientCount: 180 },

    { condition: 'Heart Disease', averageCost: 18900, patientCount: 145 },

    { condition: 'Cancer', averageCost: 32500, patientCount: 90 }

  ];

  return {

    expensesData,

    proceduresData,

    payerCoverageData,

    claimsData,

    payerAnalysisData,

    costDriversData,

    costByLocationData,

    costTrendByMonthData,

    conditionCostData

  };

}

// Helper function to generate sample claims data

function generateSampleClaimsData(count) {

  const procedures = [

    "Annual Checkup", "Blood Test", "X-Ray", "MRI Scan", "CT Scan",

    "Physical Therapy", "Vaccination", "Colonoscopy", "Dermatology Exam"

  ];

  const conditions = [

    "Hypertension", "Diabetes", "Asthma", "Arthritis", "Lower Back Pain",

    "Influenza", "Common Cold", "Allergies", "Preventive Care"

  ];

  const locations = ["hayward", "oakland", "san\_francisco", "palo\_alto", "san\_jose"];

  const genders = ["male", "female"];

  const races = ["white", "black", "asian", "hispanic", "other"];

  const ageGroups = ["0-18", "19-35", "36-50", "51-65", "65+"];

  const result = [];

  for (let i = 0; i < count; i++) {

    const baseCost = Math.floor(Math.random() \* 5000) + 100;

    const coveragePercent = Math.floor(Math.random() \* 30) + 60; // 60-90%

    result.push({

      date: `2025-0${Math.floor(Math.random() \* 3) + 1}-${Math.floor(Math.random() \* 28) + 1}`,

      procedure: procedures[Math.floor(Math.random() \* procedures.length)],

      patientId: `P${Math.floor(Math.random() \* 1000) + 1000}`,

      baseCost: baseCost,

      totalCost: Math.round(baseCost \* 1.15 \* 100) / 100,

      coveragePercent: coveragePercent,

      ageGroup: ageGroups[Math.floor(Math.random() \* ageGroups.length)],

      gender: genders[Math.floor(Math.random() \* genders.length)],

      race: races[Math.floor(Math.random() \* races.length)],

      condition: conditions[Math.floor(Math.random() \* conditions.length)],

      location: locations[Math.floor(Math.random() \* locations.length)]

    });

  }

  return result;

}

// Create CSV files from the datasets.txt if they don't exist

function createSampleDataFiles() {

  // Check if data directory exists, create if not

  if (!fs.existsSync(DATA\_PATH)) {

    fs.mkdirSync(DATA\_PATH, { recursive: true });

    console.log(`Created data directory at ${DATA\_PATH}`);

  }

  // List of CSV files to check/create

  const requiredFiles = [

    'careplans.csv',

    'conditions.csv',

    'medications.csv',

    'observations.csv',

    'payer\_transitions.csv',

    'payers.csv',

    'procedures.csv'

  ];

  let dataFilesCreated = false;

  requiredFiles.forEach(file => {

    const filePath = path.join(DATA\_PATH, file);

    if (!fs.existsSync(filePath)) {

      // If file doesn't exist, create a basic CSV with headers

      let headers = "";

      switch(file) {

        case 'careplans.csv':

          headers = "Id,START,STOP,PATIENT,ENCOUNTER,CODE,DESCRIPTION,REASONCODE,REASONDESCRIPTION";

          break;

        case 'conditions.csv':

          headers = "START,STOP,PATIENT,ENCOUNTER,CODE,DESCRIPTION";

          break;

        case 'medications.csv':

          headers = "START,STOP,PATIENT,PAYER,ENCOUNTER,CODE,DESCRIPTION,BASE\_COST,PAYER\_COVERAGE,DISPENSES,TOTALCOST,REASONCODE,REASONDESCRIPTION";

          break;

        case 'observations.csv':

          headers = "DATE,PATIENT,ENCOUNTER,CODE,DESCRIPTION,VALUE,UNITS,TYPE";

          break;

        case 'payer\_transitions.csv':

          headers = "PATIENT,START\_YEAR,END\_YEAR,PAYER,OWNERSHIP";

          break;

        case 'payers.csv':

          headers = "Id,NAME,ADDRESS,CITY,STATE\_HEADQUARTERED,ZIP,PHONE,AMOUNT\_COVERED,AMOUNT\_UNCOVERED,REVENUE,COVERED\_ENCOUNTERS,UNCOVERED\_ENCOUNTERS,COVERED\_MEDICATIONS,UNCOVERED\_MEDICATIONS,COVERED\_PROCEDURES,UNCOVERED\_PROCEDURES,COVERED\_IMMUNIZATIONS,UNCOVERED\_IMMUNIZATIONS,UNIQUE\_CUSTOMERS,QOLS\_AVG,MEMBER\_MONTHS";

          break;

        case 'procedures.csv':

          headers = "DATE,PATIENT,ENCOUNTER,CODE,DESCRIPTION,BASE\_COST,REASONCODE,REASONDESCRIPTION";

          break;

        default:

          headers = "";

      }

      fs.writeFileSync(filePath, headers + "\n");

      console.log(`Created empty CSV file with headers: ${filePath}`);

      dataFilesCreated = true;

    }

  });

  if (dataFilesCreated) {

    console.log("Created initial CSV files. For better visualization, please populate these files with real data.");

  }

}

// Call this function during server startup

createSampleDataFiles();

// Serve static files from the React build directory if it exists

const buildPath = path.join(\_\_dirname, '../build');

if (fs.existsSync(buildPath)) {

  console.log(`Serving React app from ${buildPath}`);

  app.use(express.static(buildPath));

  // Handle React routing, return all requests to React app

  app.get('\*', function(req, res) {

    if (req.url.startsWith('/api')) {

      // Let API routes be handled by other handlers

      return next();

    }

    res.sendFile(path.join(buildPath, 'index.html'));

  });

}

const PORT = process.env.PORT || 3002;

app.listen(PORT, () => {

  console.log(`Server running on port ${PORT}`);

  console.log(`API endpoints available at:`);

  console.log(`  - http://localhost:${PORT}/api/test`);

  console.log(`  - http://localhost:${PORT}/api/data`);

  console.log(`Root path: http://localhost:${PORT}/`);

});

App.css

.simple-dashboard {

  padding: 20px;

  font-family: 'Poppins', sans-serif;

}

.filters {

  margin-bottom: 20px;

}

.filters select {

  margin-left: 10px;

  padding: 5px;

  border-radius: 4px;

  border: 1px solid #ccc;

}

.chart-container {

  background-color: white;

  padding: 20px;

  border-radius: 8px;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);

}

App.js

import React from 'react';

import HealthcareDashboard from './HealthcareDashboard';

import './App.css';

function App() {

  return (

    <div className="App">

      <HealthcareDashboard />

    </div>

  );

}

export default App;

HelathcareDashboard.css

/\* Dashboard Container \*/

.dashboard-container {

  font-family: 'Poppins', sans-serif;

  color: #333;

  background-color: #f9f9f9;

  min-height: 100vh;

}

.dashboard-container.dark {

  color: #f0f0f0;

  background-color: #212121;

}

/\* Header \*/

.dashboard-header {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 1rem 2rem;

  background-color: #fff;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);

}

.dark .dashboard-header {

  background-color: #2d2d2d;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);

}

.title h1 {

  font-size: 1.5rem;

  font-weight: 600;

  display: flex;

  align-items: center;

  gap: 0.5rem;

  margin: 0;

}

.title p {

  color: #666;

  font-size: 0.9rem;

  margin: 0;

}

.dark .title p {

  color: #ccc;

}

.user-actions {

  display: flex;

  gap: 0.75rem;

}

/\* Buttons \*/

.btn {

  cursor: pointer;

  border: none;

  border-radius: 8px;

  padding: 0.5rem 1rem;

  font-size: 0.9rem;

  font-weight: 500;

  display: flex;

  align-items: center;

  gap: 0.5rem;

  transition: all 0.3s ease;

  background-color: transparent;

  color: inherit;

}

.primary-btn {

  background-color: #4361ee;

  color: white;

}

.primary-btn:hover {

  background-color: #3a56d4;

}

.secondary-btn {

  background-color: transparent;

  border: 1px solid #e0e0e0;

}

.dark .secondary-btn {

  border-color: #444;

}

.secondary-btn:hover {

  background-color: rgba(0, 0, 0, 0.03);

}

.dark .secondary-btn:hover {

  background-color: rgba(255, 255, 255, 0.03);

}

.icon-btn {

  padding: 0.5rem;

  border-radius: 50%;

}

/\* Info Bar \*/

.info-bar {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 0.75rem 2rem;

  background-color: #fff;

  border-bottom: 1px solid #e0e0e0;

  color: #666;

  font-size: 0.85rem;

}

.dark .info-bar {

  background-color: #2d2d2d;

  border-color: #444;

  color: #ccc;

}

/\* Dashboard Grid \*/

.dashboard-grid {

  display: grid;

  grid-template-columns: 220px 1fr 300px;

  min-height: calc(100vh - 130px);

}

/\* Navigation Bar \*/

.navigation-bar {

  background-color: #f5f7fa;

  border-right: 1px solid #e0e0e0;

  padding: 1.5rem 0;

  display: flex;

  flex-direction: column;

  justify-content: space-between;

  height: 100%;

}

.dark .navigation-bar {

  background-color: #1a1a1a;

  border-color: #444;

}

.nav-section {

  padding: 0 1.5rem 1.5rem;

}

.nav-section h3 {

  font-size: 0.85rem;

  font-weight: 500;

  color: #999;

  margin-bottom: 1rem;

  display: flex;

  align-items: center;

  gap: 0.5rem;

}

.nav-link {

  display: flex;

  align-items: center;

  gap: 0.75rem;

  padding: 0.75rem 1rem;

  border-radius: 8px;

  margin-bottom: 0.25rem;

  transition: all 0.3s ease;

  text-decoration: none;

  background: none;

  border: none;

  text-align: left;

  width: 100%;

  font-size: 0.9rem;

  cursor: pointer;

  color: inherit;

}

.nav-link:hover {

  background-color: rgba(67, 97, 238, 0.05);

}

.nav-link.active {

  background-color: rgba(67, 97, 238, 0.1);

  color: #4361ee;

  font-weight: 500;

}

.nav-footer {

  padding: 1.5rem;

  border-top: 1px solid #e0e0e0;

  color: #999;

  font-size: 0.85rem;

}

.dark .nav-footer {

  border-color: #444;

}

.system-status {

  display: flex;

  align-items: center;

  margin-bottom: 0.5rem;

}

.status-dot {

  width: 8px;

  height: 8px;

  border-radius: 50%;

  margin-right: 0.5rem;

}

.status-dot.online {

  background-color: #4caf50;

}

/\* Main Content \*/

.main-content {

  padding: 1.5rem;

  overflow-y: auto;

}

.content-section {

  background-color: #fff;

  border-radius: 8px;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);

  margin-bottom: 1.5rem;

  overflow: hidden;

}

.dark .content-section {

  background-color: #2d2d2d;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);

}

.section-header {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 1.25rem 1.5rem;

  border-bottom: 1px solid #e0e0e0;

}

.dark .section-header {

  border-color: #444;

}

.section-header h2 {

  font-size: 1.1rem;

  font-weight: 600;

  display: flex;

  align-items: center;

  gap: 0.5rem;

  margin: 0;

}

.section-badge {

  background-color: rgba(67, 97, 238, 0.1);

  color: #4361ee;

  padding: 0.25rem 0.75rem;

  border-radius: 1rem;

  font-size: 0.75rem;

  display: flex;

  align-items: center;

  gap: 0.25rem;

}

/\* KPI Cards \*/

.kpi-grid {

  display: grid;

  grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));

  gap: 1.5rem;

  padding: 1.5rem;

}

.kpi-card {

  display: flex;

  align-items: flex-start;

  gap: 1rem;

  padding: 1.25rem;

  border-radius: 8px;

  background-color: #fff;

  border: 1px solid #e0e0e0;

  transition: all 0.3s ease;

}

.dark .kpi-card {

  background-color: #2d2d2d;

  border-color: #444;

}

.kpi-card:hover {

  transform: translateY(-3px);

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);

}

.dark .kpi-card:hover {

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);

}

.kpi-icon {

  display: flex;

  align-items: center;

  justify-content: center;

  width: 48px;

  height: 48px;

  border-radius: 12px;

  background-color: rgba(67, 97, 238, 0.1);

  color: #4361ee;

}

.kpi-content {

  flex: 1;

}

.kpi-title {

  font-size: 0.85rem;

  color: #666;

  margin-bottom: 0.25rem;

}

.dark .kpi-title {

  color: #ccc;

}

.kpi-value {

  font-size: 1.5rem;

  font-weight: 600;

  margin-bottom: 0.5rem;

}

.kpi-change {

  font-size: 0.75rem;

  display: flex;

  align-items: center;

  gap: 0.25rem;

}

.kpi-change.positive {

  color: #4caf50;

}

.kpi-change.negative {

  color: #f44336;

}

/\* Charts Container \*/

.charts-container {

  display: grid;

  grid-template-columns: repeat(auto-fit, minmax(450px, 1fr));

  gap: 1.5rem;

  padding: 1.5rem;

}

.chart-card {

  border: 1px solid #e0e0e0;

  border-radius: 8px;

  overflow: hidden;

}

.dark .chart-card {

  border-color: #444;

}

.chart-header {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 1rem;

  border-bottom: 1px solid #e0e0e0;

}

.dark .chart-header {

  border-color: #444;

}

.chart-title {

  font-size: 0.95rem;

  font-weight: 500;

  display: flex;

  align-items: center;

  gap: 0.5rem;

}

.chart-actions {

  display: flex;

  gap: 0.5rem;

}

.chart-container {

  padding: 1rem;

  display: flex;

  justify-content: center;

  align-items: center;

}

.chart-footer {

  padding: 0.75rem 1rem;

  border-top: 1px solid #e0e0e0;

  font-size: 0.85rem;

  color: #666;

}

.dark .chart-footer {

  border-color: #444;

  color: #ccc;

}

.chart-insight {

  display: flex;

  align-items: center;

  gap: 0.5rem;

}

/\* Data Table \*/

.data-table {

  width: 100%;

  border-collapse: collapse;

}

.data-table th,

.data-table td {

  padding: 0.75rem 1rem;

  text-align: left;

  border-bottom: 1px solid #e0e0e0;

}

.dark .data-table th,

.dark .data-table td {

  border-color: #444;

}

.data-table th {

  font-weight: 500;

  color: #666;

  background-color: rgba(0, 0, 0, 0.02);

}

.dark .data-table th {

  color: #ccc;

  background-color: rgba(255, 255, 255, 0.02);

}

.data-table tr:hover {

  background-color: rgba(0, 0, 0, 0.02);

}

.dark .data-table tr:hover {

  background-color: rgba(255, 255, 255, 0.02);

}

/\* Badges \*/

.badge {

  padding: 0.25rem 0.5rem;

  border-radius: 4px;

  font-size: 0.75rem;

  font-weight: 500;

}

.badge.success {

  background-color: rgba(76, 175, 80, 0.1);

  color: #4caf50;

}

.badge.warning {

  background-color: rgba(255, 152, 0, 0.1);

  color: #ff9800;

}

.badge.danger {

  background-color: rgba(244, 67, 54, 0.1);

  color: #f44336;

}

/\* Search \*/

.search-container {

  position: relative;

  width: 300px;

}

.search-input {

  width: 100%;

  padding: 0.5rem 0.75rem 0.5rem 2.25rem;

  border: 1px solid #e0e0e0;

  border-radius: 8px;

  font-size: 0.9rem;

  transition: all 0.3s ease;

  background-color: #fff;

  color: #333;

}

.dark .search-input {

  border-color: #444;

  background-color: #2d2d2d;

  color: #f0f0f0;

}

.search-input:focus {

  border-color: #4361ee;

  outline: none;

}

.search-icon {

  position: absolute;

  left: 0.75rem;

  top: 50%;

  transform: translateY(-50%);

  color: #999;

}

/\* Coming Soon \*/

.coming-soon {

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  padding: 3rem 1.5rem;

  text-align: center;

}

.coming-soon i {

  font-size: 3rem;

  color: #999;

  margin-bottom: 1rem;

}

.coming-soon h3 {

  font-size: 1.5rem;

  margin-bottom: 0.5rem;

}

.coming-soon p {

  color: #666;

  max-width: 500px;

}

.dark .coming-soon p {

  color: #ccc;

}

/\* Data Dimensions Sidebar \*/

.data-dimensions {

  background-color: #f5f7fa;

  border-left: 1px solid #e0e0e0;

  display: flex;

  flex-direction: column;

  overflow: hidden;

}

.dark .data-dimensions {

  background-color: #1a1a1a;

  border-color: #444;

}

.filter-header {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 1.25rem 1.5rem;

  border-bottom: 1px solid #e0e0e0;

}

.dark .filter-header {

  border-color: #444;

}

.filter-header h3 {

  font-size: 1rem;

  font-weight: 500;

  display: flex;

  align-items: center;

  gap: 0.5rem;

  margin: 0;

}

.filter-body {

  flex: 1;

  overflow-y: auto;

  padding: 1.5rem;

}

.filter-section {

  margin-bottom: 1.5rem;

}

.filter-section h4 {

  font-size: 0.9rem;

  font-weight: 500;

  margin-bottom: 1rem;

  display: flex;

  align-items: center;

  gap: 0.5rem;

}

.filter-item {

  margin-bottom: 1rem;

}

.filter-item label {

  display: block;

  font-size: 0.85rem;

  color: #666;

  margin-bottom: 0.5rem;

}

.dark .filter-item label {

  color: #ccc;

}

.styled-select {

  width: 100%;

  padding: 0.5rem 0.75rem;

  border: 1px solid #e0e0e0;

  border-radius: 8px;

  font-size: 0.9rem;

  background-color: #fff;

  color: #333;

  appearance: none;

}

.dark .styled-select {

  border-color: #444;

  background-color: #2d2d2d;

  color: #f0f0f0;

}

.styled-select:focus {

  border-color: #4361ee;

  outline: none;

}

.filter-actions {

  display: flex;

  gap: 0.75rem;

  margin-top: 1.5rem;

}

.filter-footer {

  padding: 1.25rem 1.5rem;

  border-top: 1px solid #e0e0e0;

}

.dark .filter-footer {

  border-color: #444;

}

.selected-filters h4 {

  font-size: 0.9rem;

  font-weight: 500;

  margin-bottom: 0.75rem;

}

.filter-tags {

  display: flex;

  flex-wrap: wrap;

  gap: 0.5rem;

}

.filter-tag {

  display: inline-flex;

  align-items: center;

  gap: 0.25rem;

  padding: 0.25rem 0.75rem;

  background-color: rgba(67, 97, 238, 0.1);

  color: #4361ee;

  border-radius: 1rem;

  font-size: 0.8rem;

  cursor: pointer;

  transition: all 0.3s ease;

}

.filter-tag:hover {

  background-color: rgba(67, 97, 238, 0.2);

}

.filter-tag i {

  font-size: 0.75rem;

}

/\* Modal \*/

.modal-overlay {

  position: fixed;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

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

  display: flex;

  justify-content: center;

  align-items: center;

  z-index: 1000;

}

.modal-content {

  background-color: #fff;

  border-radius: 8px;

  padding: 1.5rem;

  width: 80%;

  max-width: 800px;

  max-height: 90vh;

  overflow-y: auto;

}

.dark .modal-content {

  background-color: #2d2d2d;

}

.modal-header {

  display: flex;

  justify-content: space-between;

  align-items: center;

  margin-bottom: 1.5rem;

}

.modal-header h2 {

  font-size: 1.25rem;

  font-weight: 600;

  display: flex;

  align-items: center;

  gap: 0.5rem;

  margin: 0;

}

.close-btn {

  font-size: 1.5rem;

  cursor: pointer;

  transition: all 0.3s ease;

  color: #999;

}

.close-btn:hover {

  color: #333;

}

.dark .close-btn:hover {

  color: #f0f0f0;

}

/\* Help Modal \*/

.help-content {

  display: grid;

  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));

  gap: 1.5rem;

}

.help-section h3 {

  font-size: 1.1rem;

  font-weight: 500;

  margin-bottom: 0.75rem;

}

.help-section p,

.help-section ul {

  font-size: 0.9rem;

  color: #666;

  margin-bottom: 0.75rem;

}

.dark .help-section p,

.dark .help-section ul {

  color: #ccc;

}

.help-section ul {

  padding-left: 1.5rem;

}

.help-section li {

  margin-bottom: 0.5rem;

}

/\* Export Modal \*/

.export-options {

  display: grid;

  grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));

  gap: 1rem;

  margin-bottom: 1.5rem;

}

.export-option {

  display: flex;

  flex-direction: column;

  align-items: center;

  gap: 0.75rem;

  padding: 1.5rem;

  border: 1px solid #e0e0e0;

  border-radius: 8px;

  cursor: pointer;

  transition: all 0.3s ease;

}

.dark .export-option {

  border-color: #444;

}

.export-option:hover {

  background-color: rgba(0, 0, 0, 0.02);

}

.dark .export-option:hover {

  background-color: rgba(255, 255, 255, 0.02);

}

.export-option.active {

  border-color: #4361ee;

  background-color: rgba(67, 97, 238, 0.05);

}

.export-option i {

  font-size: 2rem;

  color: #4361ee;

}

.export-settings {

  margin-top: 1.5rem;

  margin-bottom: 1.5rem;

}

.export-settings h3 {

  font-size: 1rem;

  font-weight: 500;

  margin-bottom: 1rem;

}

/\* Responsive Adjustments \*/

@media screen and (max-width: 1200px) {

  .dashboard-grid {

    grid-template-columns: 220px 1fr;

  }

  .data-dimensions {

    display: none;

  }

}

@media screen and (max-width: 768px) {

  .dashboard-grid {

    grid-template-columns: 1fr;

  }

  .navigation-bar {

    display: none;

  }

  .dashboard-header {

    flex-direction: column;

    align-items: flex-start;

    gap: 1rem;

  }

  .user-actions {

    width: 100%;

    justify-content: space-between;

  }

  .btn span {

    display: none;

  }

  .charts-container {

    grid-template-columns: 1fr;

  }

}

HealthcareDahboard.js

import React, { useState, useEffect } from 'react';

import {

BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip, Legend,

PieChart, Pie, Cell, LineChart, Line, ResponsiveContainer,

AreaChart, Area

} from 'recharts';

import axios from 'axios';

import './HealthcareDashboard.css';

const HealthcareDashboard = () => {

const [activeView, setActiveView] = useState('overview');

const [activeSettingsView, setActiveSettingsView] = useState('profile');

const [theme, setTheme] = useState('light');

const [showHelp, setShowHelp] = useState(false);

const [showExport, setShowExport] = useState(false);

const [exportFormat, setExportFormat] = useState('json');

const [isDataLoaded, setIsDataLoaded] = useState(false);

const [dataRefreshKey, setDataRefreshKey] = useState(0);

const [healthcareData, setHealthcareData] = useState(null);

const [loadingError, setLoadingError] = useState(null);

const [notifications, setNotifications] = useState([

{ id: 1, title: 'System Update', message: 'New features have been added', date: '2025-03-20', read: false },

{ id: 2, title: 'Data Refresh', message: 'Healthcare data has been updated', date: '2025-03-22', read: true },

{ id: 3, title: 'Cost Alert', message: 'Unusual increase in procedure costs detected', date: '2025-03-23', read: false }

]);

const [userPreferences, setUserPreferences] = useState({

dataRefreshInterval: '24h',

defaultView: 'overview',

chartType: 'bar',

notifications: true,

autoExport: false

});

const [userProfile, setUserProfile] = useState({

name: 'Jane Smith',

email: 'jane.smith@example.com',

role: 'Healthcare Analyst',

department: 'Financial Analytics',

lastLogin: '2025-03-23 09:45 AM'

});

const [selectedFilters, setSelectedFilters] = useState({

ageGroup: 'all',

gender: 'all',

race: 'all',

condition: 'all',

location: 'all',

dateRange: 'all',

costRange: 'all'

});

// For cost prediction

const [predictionParams, setPredictionParams] = useState({

age: '40',

gender: 'male',

condition: 'diabetes',

previousClaims: '2',

region: 'west'

});

const [predictionResults, setPredictionResults] = useState(null);

// Cost trends over time (fallback data)

const costTrendsData = [

{ month: 'Jan', hospitalCosts: 1200, primaryCare: 800, medications: 600 },

{ month: 'Feb', hospitalCosts: 1300, primaryCare: 750, medications: 650 },

{ month: 'Mar', hospitalCosts: 1100, primaryCare: 800, medications: 700 },

{ month: 'Apr', hospitalCosts: 1400, primaryCare: 850, medications: 600 },

{ month: 'May', hospitalCosts: 1500, primaryCare: 900, medications: 550 },

{ month: 'Jun', hospitalCosts: 1600, primaryCare: 950, medications: 650 }

];

// Default fallback data

const fallbackData = {

metrics: {

totalPatients: 1250,

totalExpenses: 1250000,

avgClaimCost: 2500

},

dashboardData: {

expensesData: [

{ ageGroup: '0-18', White: 4500, Black: 4200, Asian: 3800, Hispanic: 4100, Other: 4300 },

{ ageGroup: '19-35', White: 5600, Black: 5100, Asian: 4800, Hispanic: 5300, Other: 5500 },

{ ageGroup: '36-50', White: 7200, Black: 6800, Asian: 6500, Hispanic: 7000, Other: 7100 },

{ ageGroup: '51-65', White: 9500, Black: 9000, Asian: 8700, Hispanic: 9200, Other: 9300 },

{ ageGroup: '65+', White: 12000, Black: 11500, Asian: 11000, Hispanic: 11700, Other: 11800 }

],

proceduresData: [

{ name: "Heart Surgery", cost: 35000 },

{ name: "Joint Replacement", cost: 28000 },

{ name: "MRI Scan", cost: 2500 },

{ name: "Chemotherapy", cost: 18000 },

{ name: "Emergency Room Visit", cost: 1800 }

],

payerCoverageData: [

{ name: "Medicare", value: 85 },

{ name: "Medicaid", value: 78 },

{ name: "Blue Cross", value: 92 },

{ name: "Aetna", value: 88 },

{ name: "UnitedHealth", value: 90 }

],

claimsData: generateSampleClaimsData(100),

payerAnalysisData: [

{ name: "Medicare", covered: 125000, uncovered: 25000, customers: 450 },

{ name: "Medicaid", covered: 80000, uncovered: 22000, customers: 320 },

{ name: "Blue Cross", covered: 180000, uncovered: 15000, customers: 280 },

{ name: "Aetna", covered: 150000, uncovered: 18000, customers: 220 },

{ name: "UnitedHealth", covered: 210000, uncovered: 20000, customers: 350 }

],

costDriversData: [

{ category: 'Chronic Conditions', cost: 5200.50, percentage: 28 },

{ category: 'Acute Care', cost: 3800.25, percentage: 20 },

{ category: 'Preventive Care', cost: 1500.75, percentage: 8 },

{ category: 'Medications', cost: 4200.30, percentage: 23 },

{ category: 'Procedures', cost: 3900.10, percentage: 21 }

],

costByLocationData: [

{ location: 'Hayward', avgCost: 3450, totalClaims: 320 },

{ location: 'Oakland', avgCost: 4200, totalClaims: 450 },

{ location: 'San Francisco', avgCost: 5300, totalClaims: 580 },

{ location: 'Palo Alto', avgCost: 5100, totalClaims: 290 },

{ location: 'San Jose', avgCost: 3800, totalClaims: 410 }

],

costTrendByMonthData: [

{ month: 'Jan', cost: 245000 },

{ month: 'Feb', cost: 235000 },

{ month: 'Mar', cost: 260000 },

{ month: 'Apr', cost: 275000 },

{ month: 'May', cost: 290000 },

{ month: 'Jun', cost: 310000 }

],

conditionCostData: [

{ condition: 'Diabetes', averageCost: 12500, patientCount: 215 },

{ condition: 'Hypertension', averageCost: 9800, patientCount: 320 },

{ condition: 'Asthma', averageCost: 7500, patientCount: 180 },

{ condition: 'Heart Disease', averageCost: 18900, patientCount: 145 },

{ condition: 'Cancer', averageCost: 32500, patientCount: 90 }

]

}

};

// Enhanced cost prediction model data (for demonstration)

const costPredictionModel = {

baseCost: 5000,

ageFactor: {

'0-18': 0.7,

'19-35': 0.9,

'36-50': 1.2,

'51-65': 1.5,

'65+': 1.8

},

genderFactor: {

'male': 1.1,

'female': 1.0,

'other': 1.05

},

conditionFactor: {

'diabetes': 1.8,

'hypertension': 1.4,

'asthma': 1.3,

'heart': 2.5,

'cancer': 3.2,

'arthritis': 1.6,

'depression': 1.2,

'anxiety': 1.1,

'obesity': 1.5,

'copd': 2.0

},

previousClaimsFactor: {

'0': 0.9,

'1': 1.0,

'2': 1.1,

'3': 1.2,

'4+': 1.4

},

regionFactor: {

'northeast': 1.2,

'midwest': 1.0,

'south': 0.95,

'west': 1.3

}

};

// Fetch data from the backend

useEffect(() => {

const fetchData = async () => {

try {

setIsDataLoaded(false);

setLoadingError(null);

console.log('Fetching data from API...');

const response = await axios.get('http://localhost:3002/api/data');

console.log('Data loaded:', response.data);

// Check if the metrics are all zeros or show only 5 patients

const metrics = response.data.metrics || {};

if (metrics.totalExpenses === 0 || metrics.avgClaimCost === 0 || metrics.totalPatients <= 5) {

console.log('Using fallback data since metrics appear to be invalid');

// Merge server data with fallback data

const mergedData = {

...response.data,

metrics: fallbackData.metrics,

dashboardData: {

...response.data.dashboardData,

proceduresData: fallbackData.dashboardData.proceduresData,

payerCoverageData: fallbackData.dashboardData.payerCoverageData,

payerAnalysisData: fallbackData.dashboardData.payerAnalysisData,

costDriversData: fallbackData.dashboardData.costDriversData,

costByLocationData: fallbackData.dashboardData.costByLocationData,

costTrendByMonthData: fallbackData.dashboardData.costTrendByMonthData,

conditionCostData: fallbackData.dashboardData.conditionCostData

}

};

setHealthcareData(mergedData);

} else {

// Add additional fallback data for new charts if not provided by API

const enhancedData = {

...response.data,

dashboardData: {

...response.data.dashboardData,

costByLocationData: fallbackData.dashboardData.costByLocationData,

costTrendByMonthData: fallbackData.dashboardData.costTrendByMonthData,

conditionCostData: fallbackData.dashboardData.conditionCostData

}

};

setHealthcareData(enhancedData);

}

setIsDataLoaded(true);

} catch (error) {

console.error('Error fetching data:', error);

setLoadingError(`Failed to load data: ${error.message}`);

setIsDataLoaded(true); // Still set to true to avoid infinite loading

// Use fallback data if API fails

setHealthcareData(fallbackData);

}

};

fetchData();

}, [dataRefreshKey]);

// Generate predictions when prediction parameters change

useEffect(() => {

if (predictionParams) {

generateCostPrediction();

}

}, [predictionParams]);

// Helper function to generate sample claims data for fallback

function generateSampleClaimsData(count) {

const procedures = [

"Annual Checkup", "Blood Test", "X-Ray", "MRI Scan", "CT Scan",

"Physical Therapy", "Vaccination", "Colonoscopy", "Dermatology Exam"

];

const conditions = [

"Hypertension", "Diabetes", "Asthma", "Arthritis", "Lower Back Pain",

"Influenza", "Common Cold", "Allergies", "Preventive Care"

];

const locations = ["hayward", "oakland", "san\_francisco", "palo\_alto", "san\_jose"];

const genders = ["male", "female"];

const races = ["white", "black", "asian", "hispanic", "other"];

const ageGroups = ["0-18", "19-35", "36-50", "51-65", "65+"];

const result = [];

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

const baseCost = Math.floor(Math.random() \* 5000) + 100;

const coveragePercent = Math.floor(Math.random() \* 30) + 60; // 60-90%

result.push({

date: `2025-0${Math.floor(Math.random() \* 3) + 1}-${Math.floor(Math.random() \* 28) + 1}`,

procedure: procedures[Math.floor(Math.random() \* procedures.length)],

patientId: `P${Math.floor(Math.random() \* 1000) + 1000}`,

baseCost: baseCost,

totalCost: Math.round(baseCost \* 1.15 \* 100) / 100,

coveragePercent: coveragePercent,

ageGroup: ageGroups[Math.floor(Math.random() \* ageGroups.length)],

gender: genders[Math.floor(Math.random() \* genders.length)],

race: races[Math.floor(Math.random() \* races.length)],

condition: conditions[Math.floor(Math.random() \* conditions.length)],

location: locations[Math.floor(Math.random() \* locations.length)]

});

}

return result;

}

// Filter data based on selectedFilters

const getFilteredExpensesData = () => {

if (!healthcareData || !healthcareData.dashboardData || !healthcareData.dashboardData.expensesData) {

return fallbackData.dashboardData.expensesData;

}

const data = healthcareData.dashboardData.expensesData;

if (selectedFilters.ageGroup === 'all') {

return data;

}

return data.filter(item => item.ageGroup === selectedFilters.ageGroup);

};

// Filter claims data based on selectedFilters

const getFilteredClaimsData = () => {

if (!healthcareData || !healthcareData.dashboardData || !healthcareData.dashboardData.claimsData) {

return fallbackData.dashboardData.claimsData;

}

let claimsData = healthcareData.dashboardData.claimsData;

return claimsData.filter(claim => {

// Filter by age group

if (selectedFilters.ageGroup !== 'all' && claim.ageGroup && claim.ageGroup !== selectedFilters.ageGroup) return false;

// Filter by gender

if (selectedFilters.gender !== 'all' && claim.gender && claim.gender !== selectedFilters.gender) return false;

// Filter by race

if (selectedFilters.race !== 'all' && claim.race && claim.race !== selectedFilters.race) return false;

// Filter by condition (substring match)

if (selectedFilters.condition !== 'all' && claim.condition &&

!claim.condition.toLowerCase().includes(selectedFilters.condition.toLowerCase())) return false;

// Filter by location

if (selectedFilters.location !== 'all' && claim.location && claim.location !== selectedFilters.location) return false;

// Filter by cost range

if (selectedFilters.costRange !== 'all') {

const cost = claim.totalCost || 0;

switch (selectedFilters.costRange) {

case 'low':

if (cost > 1000) return false;

break;

case 'medium':

if (cost < 1000 || cost > 5000) return false;

break;

case 'high':

if (cost < 5000) return false;

break;

default:

break;

}

}

// All filters passed

return true;

});

};

// Filter condition cost data

const getFilteredConditionCostData = () => {

if (!healthcareData || !healthcareData.dashboardData || !healthcareData.dashboardData.conditionCostData) {

return fallbackData.dashboardData.conditionCostData;

}

const data = healthcareData.dashboardData.conditionCostData;

if (selectedFilters.condition === 'all') {

return data;

}

return data.filter(item =>

item.condition.toLowerCase().includes(selectedFilters.condition.toLowerCase())

);

};

// Generate a cost prediction based on input parameters

const generateCostPrediction = () => {

// Get age group from age

let ageGroup;

const age = parseInt(predictionParams.age);

if (age <= 18) ageGroup = '0-18';

else if (age <= 35) ageGroup = '19-35';

else if (age <= 50) ageGroup = '36-50';

else if (age <= 65) ageGroup = '51-65';

else ageGroup = '65+';

// Calculate predicted cost using the model

const baseCost = costPredictionModel.baseCost;

const ageFactor = costPredictionModel.ageFactor[ageGroup] || 1;

const genderFactor = costPredictionModel.genderFactor[predictionParams.gender] || 1;

const conditionFactor = costPredictionModel.conditionFactor[predictionParams.condition] || 1.2;

const previousClaimsFactor = costPredictionModel.previousClaimsFactor[predictionParams.previousClaims] || 1;

const regionFactor = costPredictionModel.regionFactor[predictionParams.region] || 1;

const predictedCost = baseCost \* ageFactor \* genderFactor \* conditionFactor \* previousClaimsFactor \* regionFactor;

// Generate range (±15% for prediction interval)

const lowerBound = predictedCost \* 0.85;

const upperBound = predictedCost \* 1.15;

// Generate some comparable costs

const averageCostForAge = baseCost \* ageFactor;

const averageCostForCondition = baseCost \* conditionFactor;

// Generate monthly prediction for a year

const monthlyPredictions = [];

let monthlyBase = predictedCost / 12;

for (let i = 1; i <= 12; i++) {

// Add some variability to monthly costs

const variation = 0.9 + (Math.random() \* 0.2); // 0.9 to 1.1

const monthlyCost = monthlyBase \* variation;

monthlyPredictions.push({

month: i,

cost: monthlyCost,

cumulativeCost: monthlyBase \* i

});

}

// Set prediction results

setPredictionResults({

predictedAnnualCost: predictedCost,

lowerBound,

upperBound,

averageCostForAge,

averageCostForCondition,

monthlyPredictions,

confidenceLevel: '85%',

riskScore: Math.round((conditionFactor \* previousClaimsFactor \* 25)),

lastUpdated: new Date().toISOString()

});

};

// Get filtered data

const filteredExpensesData = getFilteredExpensesData();

const filteredClaimsData = getFilteredClaimsData();

const filteredConditionCostData = getFilteredConditionCostData();

// Colors for charts

const COLORS = ['#8884d8', '#82ca9d', '#ffc658', '#ff8042', '#a4de6c'];

const PIE\_COLORS = ['#0088FE', '#00C49F', '#FFBB28', '#FF8042'];

const toggleTheme = () => {

setTheme(theme === 'light' ? 'dark' : 'light');

};

const refreshData = () => {

// Trigger data refresh

setDataRefreshKey(prev => prev + 1);

setTimeout(() => {

alert('Data refreshed successfully!');

}, 1000);

};

const applyFilters = () => {

console.log('Filters applied:', selectedFilters);

// The filtering happens automatically when selectedFilters state changes

alert('Filters applied successfully!');

};

const resetFilters = () => {

setSelectedFilters({

ageGroup: 'all',

gender: 'all',

race: 'all',

condition: 'all',

location: 'all',

dateRange: 'all',

costRange: 'all'

});

alert('Filters reset successfully!');

};

const handleFilterChange = (filter, value) => {

console.log(`Filter changed: ${filter} to ${value}`);

setSelectedFilters(prev => ({

...prev,

[filter]: value

}));

};

// Helper function to get filter label

const getFilterLabel = (key) => {

switch(key) {

case 'ageGroup': return 'Age';

case 'gender': return 'Gender';

case 'race': return 'Race';

case 'condition': return 'Condition';

case 'location': return 'City';

case 'dateRange': return 'Date Range';

case 'costRange': return 'Cost Range';

default: return key;

}

};

// Calculate KPI values based on filtered data

const totalPatients = healthcareData?.metrics?.totalPatients || fallbackData.metrics.totalPatients;

const avgClaimCost = healthcareData?.metrics?.avgClaimCost || fallbackData.metrics.avgClaimCost;

const totalExpenses = healthcareData?.metrics?.totalExpenses || fallbackData.metrics.totalExpenses;

// Export data in different formats

const exportData = () => {

// Prepare data for export

const dataToExport = {

patientCount: totalPatients,

averageClaim: avgClaimCost,

totalExpenses: totalExpenses,

claims: filteredClaimsData,

filters: selectedFilters

};

console.log(`Exporting data in ${exportFormat} format:`, dataToExport);

switch (exportFormat) {

case 'json':

exportAsJson(dataToExport);

break;

case 'csv':

exportAsCsv(dataToExport);

break;

case 'excel':

exportAsExcel(dataToExport);

break;

case 'pdf':

exportAsPdf(dataToExport);

break;

default:

exportAsJson(dataToExport);

}

setTimeout(() => {

alert(`Data exported successfully as ${exportFormat.toUpperCase()} file!`);

setShowExport(false);

}, 500);

};

// Export as JSON

const exportAsJson = (data) => {

const jsonString = JSON.stringify(data, null, 2);

const blob = new Blob([jsonString], { type: 'application/json' });

const url = URL.createObjectURL(blob);

downloadFile(url, 'healthcare\_dashboard\_export.json');

};

// Export as CSV

const exportAsCsv = (data) => {

// Convert claims data to CSV format

const headers = ['date', 'procedure', 'patientId', 'baseCost', 'totalCost', 'coveragePercent', 'condition', 'location'];

const csvRows = [headers.join(',')];

data.claims.forEach(claim => {

const row = headers.map(header => {

const val = claim[header];

// Handle strings with commas by wrapping in quotes

return typeof val === 'string' && val.includes(',')

? `"${val}"`

: val;

});

csvRows.push(row.join(','));

});

const csvString = csvRows.join('\n');

const blob = new Blob([csvString], { type: 'text/csv;charset=utf-8;' });

const url = URL.createObjectURL(blob);

downloadFile(url, 'healthcare\_dashboard\_export.csv');

};

// Export as Excel (simplified - in reality would use a library like xlsx)

const exportAsExcel = (data) => {

// For demo purposes, we'll also generate a CSV and just change the extension

// In a real app, you would use a library like xlsx or exceljs

const headers = ['date', 'procedure', 'patientId', 'baseCost', 'totalCost', 'coveragePercent', 'condition', 'location'];

const csvRows = [headers.join(',')];

data.claims.forEach(claim => {

const row = headers.map(header => claim[header]);

csvRows.push(row.join(','));

});

const csvString = csvRows.join('\n');

const blob = new Blob([csvString], { type: 'application/vnd.ms-excel' });

const url = URL.createObjectURL(blob);

downloadFile(url, 'healthcare\_dashboard\_export.xlsx');

};

// Export as PDF (simplified - in reality would use a library like jsPDF)

const exportAsPdf = (data) => {

// For demo purposes, we'll create a simple HTML representation

// In a real app, you would use a library like jsPDF or html2pdf

const html = `

<html>

<head>

<title>Healthcare Dashboard Export</title>

<style>

body { font-family: Arial, sans-serif; }

table { border-collapse: collapse; width: 100%; }

th, td { border: 1px solid #ddd; padding: 8px; }

th { background-color: #f2f2f2; }

</style>

</head>

<body>

<h1>Healthcare Dashboard Export</h1>

<h2>Summary</h2>

<p>Total Patients: ${data.patientCount}</p>

<p>Average Claim Cost: $${data.averageClaim.toFixed(2)}</p>

<p>Total Expenses: $${data.totalExpenses.toFixed(2)}</p>

<h2>Claims Data</h2>

<table>

<tr>

<th>Date</th>

<th>Procedure</th>

<th>Patient ID</th>

<th>Total Cost</th>

<th>Coverage %</th>

</tr>

${data.claims.slice(0, 20).map(claim => `

<tr>

<td>${claim.date}</td>

<td>${claim.procedure}</td>

<td>${claim.patientId}</td>

<td>$${claim.totalCost.toFixed(2)}</td>

<td>${claim.coveragePercent}%</td>

</tr>

`).join('')}

</table>

</body>

</html>

`;

const blob = new Blob([html], { type: 'text/html' });

const url = URL.createObjectURL(blob);

downloadFile(url, 'healthcare\_dashboard\_export.pdf');

};

// Helper for file downloads

const downloadFile = (url, filename) => {

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

a.href = url;

a.download = filename;

document.body.appendChild(a);

a.click();

document.body.removeChild(a);

};

// Handler for updating user preferences

const handlePreferenceChange = (preference, value) => {

setUserPreferences(prev => ({

...prev,

[preference]: value

}));

};

// Handler for updating user profile

const handleProfileChange = (field, value) => {

setUserProfile(prev => ({

...prev,

[field]: value

}));

};

// Mark a notification as read

const markNotificationAsRead = (id) => {

setNotifications(prev =>

prev.map(notif =>

notif.id === id ? { ...notif, read: true } : notif

)

);

};

// Clear all notifications

const clearAllNotifications = () => {

setNotifications([]);

};

// Handler for prediction parameter changes

const handlePredictionParamChange = (param, value) => {

setPredictionParams(prev => ({

...prev,

[param]: value

}));

};

return (

<div className={`dashboard-container ${theme}`}>

{/\* Header \*/}

<header className="dashboard-header">

<div className="title">

<h1><i className="fas fa-hospital"></i> Healthcare Cost Prediction Dashboard</h1>

<p>Analyze healthcare costs and make informed decisions</p>

</div>

<div className="user-actions">

<button className="btn icon-btn" onClick={toggleTheme}>

<i className={`fas fa-${theme === 'light' ? 'moon' : 'sun'}`}></i>

</button>

<button className="btn icon-btn" onClick={refreshData}>

<i className="fas fa-sync-alt"></i>

</button>

<button className="btn icon-btn" onClick={() => setShowHelp(true)}>

<i className="fas fa-question-circle"></i>

</button>

<button className="btn primary-btn" onClick={() => setShowExport(true)}>

<i className="fas fa-download"></i> <span>Export</span>

</button>

</div>

</header>

{/\* Info Bar \*/}

<div className="info-bar">

<div>Last updated: March 23, 2025 | Data source: Healthcare Claims Database</div>

<div>

{isDataLoaded ? (

loadingError ? (

<span style={{ color: '#f44336' }}><i className="fas fa-exclamation-circle"></i> {loadingError}</span>

) : (

<span><i className="fas fa-check-circle"></i> Data loaded successfully</span>

)

) : (

<span><i className="fas fa-spinner fa-spin"></i> Loading data...</span>

)}

</div>

</div>

{/\* Dashboard Grid \*/}

<div className="dashboard-grid">

{/\* Navigation \*/}

<nav className="navigation-bar">

<div>

<div className="nav-section">

<h3><i className="fas fa-compass"></i> Navigation</h3>

<button

className={`nav-link ${activeView === 'overview' ? 'active' : ''}`}

onClick={() => setActiveView('overview')}

>

<i className="fas fa-home"></i> Overview

</button>

<button

className={`nav-link ${activeView === 'costs' ? 'active' : ''}`}

onClick={() => setActiveView('costs')}

>

<i className="fas fa-dollar-sign"></i> Cost Analysis

</button>

<button

className={`nav-link ${activeView === 'claims' ? 'active' : ''}`}

onClick={() => setActiveView('claims')}

>

<i className="fas fa-file-medical"></i> Claims

</button>

<button

className={`nav-link ${activeView === 'prediction' ? 'active' : ''}`}

onClick={() => setActiveView('prediction')}

>

<i className="fas fa-chart-line"></i> Prediction

</button>

</div>

<div className="nav-section">

<h3><i className="fas fa-cog"></i> Settings</h3>

<button

className={`nav-link ${activeView === 'settings' && activeSettingsView === 'profile' ? 'active' : ''}`}

onClick={() => {

setActiveView('settings');

setActiveSettingsView('profile');

}}

>

<i className="fas fa-user-cog"></i> User Profile

</button>

<button

className={`nav-link ${activeView === 'settings' && activeSettingsView === 'notifications' ? 'active' : ''}`}

onClick={() => {

setActiveView('settings');

setActiveSettingsView('notifications');

}}

>

<i className="fas fa-bell"></i> Notifications

{notifications.filter(n => !n.read).length > 0 && (

<span className="badge success" style={{ marginLeft: '5px' }}>

{notifications.filter(n => !n.read).length}

</span>

)}

</button>

<button

className={`nav-link ${activeView === 'settings' && activeSettingsView === 'preferences' ? 'active' : ''}`}

onClick={() => {

setActiveView('settings');

setActiveSettingsView('preferences');

}}

>

<i className="fas fa-sliders-h"></i> Preferences

</button>

</div>

</div>

<div className="nav-footer">

<div className="system-status">

<div className="status-dot online"></div>

System Online

</div>

<div>

v1.0.0

</div>

</div>

</nav>

{/\* Main Content \*/}

<main className="main-content">

{activeView === 'overview' && (

<>

{/\* KPI Cards \*/}

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-chart-pie"></i> Key Performance Indicators</h2>

<div className="section-badge">

<i className="fas fa-calendar-alt"></i> Last 30 days

</div>

</div>

<div className="kpi-grid">

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-users"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Total Patients</div>

<div className="kpi-value">{totalPatients}</div>

<div className="kpi-change positive">

<i className="fas fa-arrow-up"></i> 12% from last month

</div>

</div>

</div>

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-receipt"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Average Claim Cost</div>

<div className="kpi-value">${avgClaimCost.toFixed(2)}</div>

<div className="kpi-change negative">

<i className="fas fa-arrow-down"></i> 3% from last month

</div>

</div>

</div>

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-dollar-sign"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Total Expenses</div>

<div className="kpi-value">${totalExpenses.toFixed(2)}</div>

<div className="kpi-change positive">

<i className="fas fa-arrow-up"></i> 8% from last month

</div>

</div>

</div>

</div>

</div>

{/\* Charts \*/}

<div className="charts-container">

{/\* Expenses by Age & Race Chart \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-chart-bar"></i> Healthcare Expenses by Age & Race

</div>

<div className="chart-actions">

<button className="btn icon-btn">

<i className="fas fa-expand-alt"></i>

</button>

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<BarChart

data={filteredExpensesData}

margin={{ top: 20, right: 30, left: 20, bottom: 5 }}

>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="ageGroup" />

<YAxis />

<Tooltip formatter={(value) => `$${value}`} />

<Legend />

<Bar dataKey="White" fill={COLORS[0]} />

<Bar dataKey="Black" fill={COLORS[1]} />

<Bar dataKey="Asian" fill={COLORS[2]} />

<Bar dataKey="Hispanic" fill={COLORS[3]} />

<Bar dataKey="Other" fill={COLORS[4]} />

</BarChart>

</ResponsiveContainer>

</div>

<div className="chart-footer">

<div className="chart-insight">

<i className="fas fa-info-circle"></i> Highest expenses observed in the 65+ age group across all demographics

</div>

</div>

</div>

{/\* Top Procedures by Cost \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-procedures"></i> Top Procedures by Cost

</div>

<div className="chart-actions">

<button className="btn icon-btn">

<i className="fas fa-expand-alt"></i>

</button>

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<BarChart

layout="vertical"

data={healthcareData?.dashboardData?.proceduresData || fallbackData.dashboardData.proceduresData}

margin={{ top: 20, right: 30, left: 100, bottom: 5 }}

>

<CartesianGrid strokeDasharray="3 3" />

<XAxis type="number" />

<YAxis dataKey="name" type="category" />

<Tooltip formatter={(value) => `$${value}`} />

<Bar dataKey="cost" fill="#4361ee" />

</BarChart>

</ResponsiveContainer>

</div>

<div className="chart-footer">

<div className="chart-insight">

<i className="fas fa-info-circle"></i> Specialized procedures account for 65% of total costs

</div>

</div>

</div>

</div>

{/\* More Charts \*/}

<div className="charts-container">

{/\* Cost Trends Over Time \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-chart-line"></i> Cost Trends Over Time

</div>

<div className="chart-actions">

<button className="btn icon-btn">

<i className="fas fa-expand-alt"></i>

</button>

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<LineChart

data={costTrendsData}

margin={{ top: 20, right: 30, left: 20, bottom: 5 }}

>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="month" />

<YAxis />

<Tooltip formatter={(value) => `$${value}`} />

<Legend />

<Line type="monotone" dataKey="hospitalCosts" stroke="#8884d8" activeDot={{ r: 8 }} />

<Line type="monotone" dataKey="primaryCare" stroke="#82ca9d" />

<Line type="monotone" dataKey="medications" stroke="#ffc658" />

</LineChart>

</ResponsiveContainer>

</div>

<div className="chart-footer">

<div className="chart-insight">

<i className="fas fa-info-circle"></i> Hospital costs are consistently rising across all months

</div>

</div>

</div>

{/\* Payer Coverage \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-percent"></i> Payer Coverage Rates

</div>

<div className="chart-actions">

<button className="btn icon-btn">

<i className="fas fa-expand-alt"></i>

</button>

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<PieChart>

<Pie

data={healthcareData?.dashboardData?.payerCoverageData || fallbackData.dashboardData.payerCoverageData}

cx="50%"

cy="50%"

labelLine={false}

label={({name, value}) => `${name}: ${value}%`}

outerRadius={80}

fill="#8884d8"

dataKey="value"

>

{(healthcareData?.dashboardData?.payerCoverageData || fallbackData.dashboardData.payerCoverageData)

.map((entry, index) => (

<Cell key={`cell-${index}`} fill={PIE\_COLORS[index % PIE\_COLORS.length]} />

))}

</Pie>

<Tooltip formatter={(value) => `${value}%`} />

</PieChart>

</ResponsiveContainer>

</div>

<div className="chart-footer">

<div className="chart-insight">

<i className="fas fa-info-circle"></i> Private insurers provide highest average coverage rates

</div>

</div>

</div>

</div>

{/\* Recent Claims Table \*/}

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-table"></i> Recent Claims</h2>

<div className="search-container">

<input type="text" className="search-input" placeholder="Search claims..." />

<i className="fas fa-search search-icon"></i>

</div>

</div>

<div style={{ overflowX: 'auto' }}>

<table className="data-table">

<thead>

<tr>

<th>Date</th>

<th>Procedure</th>

<th>Patient ID</th>

<th>Base Cost</th>

<th>Total Cost</th>

<th>Coverage %</th>

<th>Status</th>

</tr>

</thead>

<tbody>

{filteredClaimsData.slice(0, 5).map((claim, index) => (

<tr key={index}>

<td>{claim.date}</td>

<td>{claim.procedure}</td>

<td>{claim.patientId}</td>

<td>${claim.baseCost.toFixed(2)}</td>

<td>${claim.totalCost.toFixed(2)}</td>

<td>{claim.coveragePercent}%</td>

<td>

<span className={`badge ${claim.coveragePercent > 80 ? 'success' : claim.coveragePercent > 60 ? 'warning' : 'danger'}`}>

{claim.coveragePercent > 80 ? 'Approved' : claim.coveragePercent > 60 ? 'Pending' : 'Review'}

</span>

</td>

</tr>

))}

</tbody>

</table>

</div>

</div>

</>

)}

{activeView === 'costs' && (

<>

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-dollar-sign"></i> Cost Analysis</h2>

<div className="section-badge">

<i className="fas fa-filter"></i> {Object.values(selectedFilters).filter(v => v !== 'all').length} filters applied

</div>

</div>

{/\* Cost Analysis KPIs \*/}

<div className="kpi-grid">

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-hand-holding-usd"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Average Cost Per Patient</div>

<div className="kpi-value">${(totalExpenses / totalPatients).toFixed(2)}</div>

<div className="kpi-change positive">

<i className="fas fa-arrow-down"></i> 5.2% from last month

</div>

</div>

</div>

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-hospital-user"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Cost Per Encounter</div>

<div className="kpi-value">${(avgClaimCost \* 1.8).toFixed(2)}</div>

<div className="kpi-change negative">

<i className="fas fa-arrow-up"></i> 2.3% from last month

</div>

</div>

</div>

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-calendar-week"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Monthly Trend</div>

<div className="kpi-value">+${(totalExpenses \* 0.06).toFixed(2)}</div>

<div className="kpi-change negative">

<i className="fas fa-arrow-up"></i> Growing

</div>

</div>

</div>

</div>

</div>

<div className="charts-container">

{/\* Cost Drivers \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-sitemap"></i> Cost Drivers

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<PieChart>

<Pie

data={healthcareData?.dashboardData?.costDriversData || fallbackData.dashboardData.costDriversData}

cx="50%"

cy="50%"

labelLine={false}

label={({category, percentage}) => `${category}: ${percentage}%`}

outerRadius={80}

fill="#8884d8"

dataKey="percentage"

>

{(healthcareData?.dashboardData?.costDriversData || fallbackData.dashboardData.costDriversData)

.map((entry, index) => (

<Cell key={`cell-${index}`} fill={PIE\_COLORS[index % PIE\_COLORS.length]} />

))}

</Pie>

<Tooltip formatter={(value) => `${value}%`} />

</PieChart>

</ResponsiveContainer>

</div>

</div>

{/\* Payer Analysis \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-building"></i> Payer Analysis

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<BarChart

data={healthcareData?.dashboardData?.payerAnalysisData || fallbackData.dashboardData.payerAnalysisData}

margin={{ top: 20, right: 30, left: 20, bottom: 5 }}

>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="name" />

<YAxis />

<Tooltip formatter={(value) => `$${value}`} />

<Legend />

<Bar dataKey="covered" name="Amount Covered" fill="#4361ee" />

<Bar dataKey="uncovered" name="Amount Uncovered" fill="#f44336" />

</BarChart>

</ResponsiveContainer>

</div>

</div>

</div>

<div className="charts-container">

{/\* Cost By Location \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-map-marker-alt"></i> Cost By Location

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<BarChart

data={healthcareData?.dashboardData?.costByLocationData || fallbackData.dashboardData.costByLocationData}

margin={{ top: 20, right: 30, left: 20, bottom: 5 }}

>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="location" />

<YAxis yAxisId="left" orientation="left" stroke="#8884d8" />

<YAxis yAxisId="right" orientation="right" stroke="#82ca9d" />

<Tooltip />

<Legend />

<Bar yAxisId="left" dataKey="avgCost" name="Average Cost ($)" fill="#8884d8" />

<Bar yAxisId="right" dataKey="totalClaims" name="Total Claims" fill="#82ca9d" />

</BarChart>

</ResponsiveContainer>

</div>

</div>

{/\* Cost Trend By Month \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-calendar-alt"></i> Cost Trend By Month

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<AreaChart

data={healthcareData?.dashboardData?.costTrendByMonthData || fallbackData.dashboardData.costTrendByMonthData}

margin={{ top: 20, right: 30, left: 20, bottom: 5 }}

>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="month" />

<YAxis />

<Tooltip formatter={(value) => `$${value}`} />

<Area type="monotone" dataKey="cost" stroke="#8884d8" fill="#8884d8" fillOpacity={0.3} />

</AreaChart>

</ResponsiveContainer>

</div>

</div>

</div>

{/\* Condition Cost Analysis \*/}

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-heartbeat"></i> Cost by Condition</h2>

</div>

<div style={{ overflowX: 'auto', padding: '1.5rem' }}>

<table className="data-table">

<thead>

<tr>

<th>Condition</th>

<th>Average Cost</th>

<th>Patient Count</th>

<th>Total Cost</th>

<th>% of Budget</th>

<th>Trend</th>

</tr>

</thead>

<tbody>

{filteredConditionCostData.map((condition, index) => {

const totalCost = condition.averageCost \* condition.patientCount;

const percentOfBudget = (totalCost / totalExpenses \* 100).toFixed(1);

return (

<tr key={index}>

<td>{condition.condition}</td>

<td>${condition.averageCost.toFixed(2)}</td>

<td>{condition.patientCount}</td>

<td>${totalCost.toFixed(2)}</td>

<td>{percentOfBudget}%</td>

<td>

<span className={`badge ${index % 3 === 0 ? 'danger' : index % 3 === 1 ? 'warning' : 'success'}`}>

{index % 3 === 0 ? <><i className="fas fa-arrow-up"></i> Rising</> :

index % 3 === 1 ? <><i className="fas fa-arrows-alt-h"></i> Stable</> :

<><i className="fas fa-arrow-down"></i> Falling</>}

</span>

</td>

</tr>

);

})}

</tbody>

</table>

</div>

</div>

</>

)}

{activeView === 'claims' && (

<>

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-file-medical"></i> Claims Data</h2>

<div className="search-container">

<input type="text" className="search-input" placeholder="Search claims..." />

<i className="fas fa-search search-icon"></i>

</div>

</div>

{/\* Claims KPIs \*/}

<div className="kpi-grid">

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-file-invoice-dollar"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Total Claims</div>

<div className="kpi-value">{filteredClaimsData.length}</div>

<div className="kpi-change positive">

<i className="fas fa-arrow-up"></i> 8.5% from last month

</div>

</div>

</div>

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-check-circle"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Approval Rate</div>

<div className="kpi-value">

{((filteredClaimsData.filter(c => c.coveragePercent > 80).length / filteredClaimsData.length) \* 100).toFixed(1)}%

</div>

<div className="kpi-change positive">

<i className="fas fa-arrow-up"></i> 2.1% from last month

</div>

</div>

</div>

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-money-check-alt"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Average Processing Time</div>

<div className="kpi-value">3.2 days</div>

<div className="kpi-change positive">

<i className="fas fa-arrow-down"></i> 0.5 days from last month

</div>

</div>

</div>

</div>

</div>

{/\* Claims Visualization \*/}

<div className="charts-container">

{/\* Claims by Age Group \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-users"></i> Claims by Age Group

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<PieChart>

<Pie

data={Object.entries(filteredClaimsData.reduce((acc, claim) => {

const ageGroup = claim.ageGroup || 'Unknown';

acc[ageGroup] = (acc[ageGroup] || 0) + 1;

return acc;

}, {})).map(([age, count]) => ({ name: age, value: count }))}

cx="50%"

cy="50%"

labelLine={false}

label={({name, value, percent}) => `${name}: ${(percent \* 100).toFixed(0)}%`}

outerRadius={80}

fill="#8884d8"

dataKey="value"

>

{Object.entries(filteredClaimsData.reduce((acc, claim) => {

const ageGroup = claim.ageGroup || 'Unknown';

acc[ageGroup] = (acc[ageGroup] || 0) + 1;

return acc;

}, {})).map(([age, count], index) => (

<Cell key={`cell-${index}`} fill={PIE\_COLORS[index % PIE\_COLORS.length]} />

))}

</Pie>

<Tooltip />

</PieChart>

</ResponsiveContainer>

</div>

</div>

{/\* Claims by Condition \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-stethoscope"></i> Top Conditions

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<BarChart

data={Object.entries(filteredClaimsData.reduce((acc, claim) => {

const condition = claim.condition || 'Unknown';

acc[condition] = (acc[condition] || 0) + 1;

return acc;

}, {}))

.map(([condition, count]) => ({ condition, count }))

.sort((a, b) => b.count - a.count)

.slice(0, 5)}

margin={{ top: 20, right: 30, left: 20, bottom: 5 }}

>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="condition" />

<YAxis />

<Tooltip />

<Bar dataKey="count" name="Number of Claims" fill="#4361ee" />

</BarChart>

</ResponsiveContainer>

</div>

</div>

</div>

{/\* Claims Data Table \*/}

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-table"></i> Detailed Claims</h2>

<div className="section-badge">

<i className="fas fa-filter"></i> {filteredClaimsData.length} records

</div>

</div>

<div style={{ overflowX: 'auto' }}>

<table className="data-table">

<thead>

<tr>

<th>Date</th>

<th>Procedure</th>

<th>Patient ID</th>

<th>Condition</th>

<th>Age Group</th>

<th>Gender</th>

<th>Base Cost</th>

<th>Total Cost</th>

<th>Coverage %</th>

<th>Status</th>

</tr>

</thead>

<tbody>

{filteredClaimsData.slice(0, 20).map((claim, index) => (

<tr key={index}>

<td>{claim.date}</td>

<td>{claim.procedure}</td>

<td>{claim.patientId}</td>

<td>{claim.condition}</td>

<td>{claim.ageGroup}</td>

<td>{claim.gender}</td>

<td>${claim.baseCost.toFixed(2)}</td>

<td>${claim.totalCost.toFixed(2)}</td>

<td>{claim.coveragePercent}%</td>

<td>

<span className={`badge ${claim.coveragePercent > 80 ? 'success' : claim.coveragePercent > 60 ? 'warning' : 'danger'}`}>

{claim.coveragePercent > 80 ? 'Approved' : claim.coveragePercent > 60 ? 'Pending' : 'Review'}

</span>

</td>

</tr>

))}

</tbody>

</table>

</div>

<div style={{ padding: '1rem', textAlign: 'center' }}>

<button className="btn secondary-btn">

<i className="fas fa-chevron-down"></i> Load More

</button>

</div>

</div>

</>

)}

{activeView === 'prediction' && (

<>

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-chart-line"></i> Cost Prediction</h2>

</div>

<div style={{ padding: '1.5rem' }}>

<div style={{ marginBottom: '2rem' }}>

<h3><i className="fas fa-cogs"></i> Prediction Parameters</h3>

<p>Adjust the parameters below to generate a cost prediction.</p>

<div style={{ display: 'grid', gridTemplateColumns: 'repeat(auto-fill, minmax(200px, 1fr))', gap: '1rem', marginTop: '1rem' }}>

<div className="filter-item">

<label htmlFor="predAge">Age</label>

<input

id="predAge"

type="number"

className="styled-select"

value={predictionParams.age}

onChange={(e) => handlePredictionParamChange('age', e.target.value)}

min="0"

max="120"

/>

</div>

<div className="filter-item">

<label htmlFor="predGender">Gender</label>

<select

id="predGender"

className="styled-select"

value={predictionParams.gender}

onChange={(e) => handlePredictionParamChange('gender', e.target.value)}

>

<option value="male">Male</option>

<option value="female">Female</option>

<option value="other">Other</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="predCondition">Primary Condition</label>

<select

id="predCondition"

className="styled-select"

value={predictionParams.condition}

onChange={(e) => handlePredictionParamChange('condition', e.target.value)}

>

<option value="diabetes">Diabetes</option>

<option value="hypertension">Hypertension</option>

<option value="asthma">Asthma</option>

<option value="heart">Heart Disease</option>

<option value="cancer">Cancer</option>

<option value="arthritis">Arthritis</option>

<option value="depression">Depression</option>

<option value="anxiety">Anxiety</option>

<option value="obesity">Obesity</option>

<option value="copd">COPD</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="predClaims">Previous Claims</label>

<select

id="predClaims"

className="styled-select"

value={predictionParams.previousClaims}

onChange={(e) => handlePredictionParamChange('previousClaims', e.target.value)}

>

<option value="0">0</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4+">4+</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="predRegion">Region</label>

<select

id="predRegion"

className="styled-select"

value={predictionParams.region}

onChange={(e) => handlePredictionParamChange('region', e.target.value)}

>

<option value="northeast">Northeast</option>

<option value="midwest">Midwest</option>

<option value="south">South</option>

<option value="west">West</option>

</select>

</div>

</div>

<div style={{ marginTop: '1.5rem' }}>

<button className="btn primary-btn" onClick={generateCostPrediction}>

<i className="fas fa-calculator"></i> Generate Prediction

</button>

</div>

</div>

{predictionResults && (

<div style={{ marginTop: '2rem' }}>

<h3><i className="fas fa-chart-pie"></i> Prediction Results</h3>

<div className="kpi-grid" style={{ marginTop: '1rem' }}>

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-dollar-sign"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Predicted Annual Cost</div>

<div className="kpi-value">${predictionResults.predictedAnnualCost.toFixed(2)}</div>

<div className="kpi-change">

<i className="fas fa-info-circle"></i> Confidence: {predictionResults.confidenceLevel}

</div>

</div>

</div>

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-exchange-alt"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Prediction Range</div>

<div className="kpi-value">${predictionResults.lowerBound.toFixed(0)} - ${predictionResults.upperBound.toFixed(0)}</div>

<div className="kpi-change">

<i className="fas fa-info-circle"></i> 15% deviation possible

</div>

</div>

</div>

<div className="kpi-card">

<div className="kpi-icon">

<i className="fas fa-exclamation-triangle"></i>

</div>

<div className="kpi-content">

<div className="kpi-title">Risk Score</div>

<div className="kpi-value">{predictionResults.riskScore}/100</div>

<div className="kpi-change">

<i className="fas fa-info-circle"></i> {predictionResults.riskScore > 75 ? 'High' : predictionResults.riskScore > 50 ? 'Medium' : 'Low'} risk

</div>

</div>

</div>

</div>

<div className="charts-container">

{/\* Monthly Cost Prediction \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-calendar-alt"></i> Monthly Cost Projection

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<LineChart

data={predictionResults.monthlyPredictions}

margin={{ top: 20, right: 30, left: 20, bottom: 5 }}

>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="month" label={{ value: 'Month', position: 'insideBottomRight', offset: -10 }} />

<YAxis label={{ value: 'Cost ($)', angle: -90, position: 'insideLeft' }} />

<Tooltip formatter={(value) => `$${parseFloat(value).toFixed(2)}`} />

<Legend />

<Line type="monotone" dataKey="cost" name="Monthly Cost" stroke="#8884d8" activeDot={{ r: 8 }} />

<Line type="monotone" dataKey="cumulativeCost" name="Cumulative Cost" stroke="#82ca9d" />

</LineChart>

</ResponsiveContainer>

</div>

</div>

{/\* Comparative Analysis \*/}

<div className="chart-card">

<div className="chart-header">

<div className="chart-title">

<i className="fas fa-balance-scale"></i> Comparative Analysis

</div>

</div>

<div className="chart-container" style={{ height: '300px' }}>

<ResponsiveContainer width="100%" height="100%">

<BarChart

data={[

{ name: 'Age Group Average', value: predictionResults.averageCostForAge },

{ name: 'Condition Average', value: predictionResults.averageCostForCondition },

{ name: 'Your Prediction', value: predictionResults.predictedAnnualCost }

]}

margin={{ top: 20, right: 30, left: 20, bottom: 5 }}

>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="name" />

<YAxis />

<Tooltip formatter={(value) => `$${parseFloat(value).toFixed(2)}`} />

<Bar dataKey="value" fill="#4361ee" />

</BarChart>

</ResponsiveContainer>

</div>

</div>

</div>

<div style={{ marginTop: '2rem', backgroundColor: '#f8f9fa', padding: '1.5rem', borderRadius: '8px', border: '1px solid #e0e0e0' }} className={theme === 'dark' ? 'dark' : ''}>

<h4><i className="fas fa-lightbulb"></i> Insights & Recommendations</h4>

<ul style={{ marginTop: '0.5rem', paddingLeft: '1.5rem' }}>

<li>Your predicted annual healthcare cost is <strong>${predictionResults.predictedAnnualCost.toFixed(2)}</strong>, which is {predictionResults.predictedAnnualCost > predictionResults.averageCostForAge ? 'higher' : 'lower'} than the average for your age group (${predictionResults.averageCostForAge.toFixed(2)}).</li>

<li>The primary cost driver for your prediction is your condition: <strong>{predictionParams.condition}</strong>.</li>

<li>Consider preventive care measures to manage {predictionParams.condition} and potentially reduce future costs.</li>

<li>Based on your risk score, consider reviewing your coverage options to ensure adequate protection.</li>

<li>Monthly cost monitoring is recommended to track expenses against projections.</li>

</ul>

</div>

</div>

)}

</div>

</div>

</>

)}

{activeView === 'settings' && activeSettingsView === 'profile' && (

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-user-cog"></i> User Profile</h2>

</div>

<div style={{ padding: '1.5rem' }}>

<div style={{ display: 'flex', alignItems: 'center', marginBottom: '2rem' }}>

<div style={{ width: '120px', height: '120px', borderRadius: '50%', backgroundColor: '#4361ee', display: 'flex', alignItems: 'center', justifyContent: 'center', marginRight: '2rem' }}>

<i className="fas fa-user" style={{ fontSize: '3rem', color: 'white' }}></i>

</div>

<div>

<h3 style={{ marginBottom: '0.5rem' }}>{userProfile.name}</h3>

<p style={{ margin: '0.25rem 0' }}><i className="fas fa-briefcase"></i> {userProfile.role}</p>

<p style={{ margin: '0.25rem 0' }}><i className="fas fa-building"></i> {userProfile.department}</p>

<p style={{ margin: '0.25rem 0' }}><i className="fas fa-envelope"></i> {userProfile.email}</p>

<p style={{ margin: '0.25rem 0' }}><i className="fas fa-clock"></i> Last login: {userProfile.lastLogin}</p>

</div>

</div>

<div style={{ marginBottom: '2rem' }}>

<h3><i className="fas fa-info-circle"></i> Personal Information</h3>

<div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '1rem', marginTop: '1rem' }}>

<div className="filter-item">

<label htmlFor="userName">Full Name</label>

<input

id="userName"

type="text"

className="styled-select"

value={userProfile.name}

onChange={(e) => handleProfileChange('name', e.target.value)}

/>

</div>

<div className="filter-item">

<label htmlFor="userEmail">Email Address</label>

<input

id="userEmail"

type="email"

className="styled-select"

value={userProfile.email}

onChange={(e) => handleProfileChange('email', e.target.value)}

/>

</div>

<div className="filter-item">

<label htmlFor="userRole">Role</label>

<input

id="userRole"

type="text"

className="styled-select"

value={userProfile.role}

onChange={(e) => handleProfileChange('role', e.target.value)}

/>

</div>

<div className="filter-item">

<label htmlFor="userDepartment">Department</label>

<input

id="userDepartment"

type="text"

className="styled-select"

value={userProfile.department}

onChange={(e) => handleProfileChange('department', e.target.value)}

/>

</div>

</div>

</div>

<div style={{ marginBottom: '2rem' }}>

<h3><i className="fas fa-lock"></i> Security</h3>

<div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '1rem', marginTop: '1rem' }}>

<div className="filter-item">

<label htmlFor="currentPassword">Current Password</label>

<input

id="currentPassword"

type="password"

className="styled-select"

placeholder="Enter current password"

/>

</div>

<div className="filter-item">

<label htmlFor="newPassword">New Password</label>

<input

id="newPassword"

type="password"

className="styled-select"

placeholder="Enter new password"

/>

</div>

</div>

</div>

<div style={{ marginTop: '2rem' }}>

<button className="btn primary-btn">

<i className="fas fa-save"></i> Save Changes

</button>

<button className="btn secondary-btn" style={{ marginLeft: '1rem' }}>

<i className="fas fa-undo"></i> Cancel

</button>

</div>

</div>

</div>

)}

{activeView === 'settings' && activeSettingsView === 'notifications' && (

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-bell"></i> Notification Center</h2>

<button className="btn secondary-btn" onClick={clearAllNotifications}>

<i className="fas fa-trash"></i> Clear All

</button>

</div>

<div style={{ padding: '1.5rem' }}>

{notifications.length === 0 ? (

<div style={{ textAlign: 'center', padding: '2rem' }}>

<i className="fas fa-check-circle" style={{ fontSize: '3rem', color: '#4caf50', marginBottom: '1rem' }}></i>

<h3>No Notifications</h3>

<p>You're all caught up! There are no notifications to display.</p>

</div>

) : (

<div>

{notifications.map((notification) => (

<div

key={notification.id}

style={{

padding: '1rem',

borderLeft: notification.read ? '4px solid #e0e0e0' : '4px solid #4361ee',

marginBottom: '1rem',

backgroundColor: notification.read ? 'transparent' : 'rgba(67, 97, 238, 0.05)',

borderRadius: '4px'

}}

>

<div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', marginBottom: '0.5rem' }}>

<h4 style={{ margin: 0 }}>

{!notification.read && <span style={{ display: 'inline-block', width: '8px', height: '8px', borderRadius: '50%', backgroundColor: '#4361ee', marginRight: '0.5rem' }}></span>}

{notification.title}

</h4>

<span style={{ fontSize: '0.85rem', color: '#666' }}>{notification.date}</span>

</div>

<p style={{ margin: '0.5rem 0' }}>{notification.message}</p>

<div style={{ marginTop: '0.5rem' }}>

{!notification.read && (

<button

className="btn secondary-btn"

style={{ padding: '0.25rem 0.5rem', fontSize: '0.8rem' }}

onClick={() => markNotificationAsRead(notification.id)}

>

<i className="fas fa-check"></i> Mark as Read

</button>

)}

</div>

</div>

))}

</div>

)}

<div style={{ marginTop: '2rem' }}>

<h3><i className="fas fa-cog"></i> Notification Settings</h3>

<div style={{ marginTop: '1rem' }}>

<div className="filter-item">

<label>

<input type="checkbox" checked={userPreferences.notifications} onChange={(e) => handlePreferenceChange('notifications', e.target.checked)} />

Enable Notifications

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked />

Email Notifications

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked />

Cost Alert Notifications

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked />

System Update Notifications

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked />

Data Refresh Notifications

</label>

</div>

</div>

</div>

</div>

</div>

)}

{activeView === 'settings' && activeSettingsView === 'preferences' && (

<div className="content-section">

<div className="section-header">

<h2><i className="fas fa-sliders-h"></i> User Preferences</h2>

</div>

<div style={{ padding: '1.5rem' }}>

<div style={{ marginBottom: '2rem' }}>

<h3><i className="fas fa-desktop"></i> Display Settings</h3>

<div style={{ marginTop: '1rem' }}>

<div className="filter-item">

<label htmlFor="themePreference">Theme</label>

<select

id="themePreference"

className="styled-select"

value={theme}

onChange={(e) => setTheme(e.target.value)}

>

<option value="light">Light</option>

<option value="dark">Dark</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="defaultView">Default View</label>

<select

id="defaultView"

className="styled-select"

value={userPreferences.defaultView}

onChange={(e) => handlePreferenceChange('defaultView', e.target.value)}

>

<option value="overview">Overview</option>

<option value="costs">Cost Analysis</option>

<option value="claims">Claims</option>

<option value="prediction">Prediction</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="chartType">Preferred Chart Type</label>

<select

id="chartType"

className="styled-select"

value={userPreferences.chartType}

onChange={(e) => handlePreferenceChange('chartType', e.target.value)}

>

<option value="bar">Bar Chart</option>

<option value="line">Line Chart</option>

<option value="pie">Pie Chart</option>

<option value="area">Area Chart</option>

</select>

</div>

</div>

</div>

<div style={{ marginBottom: '2rem' }}>

<h3><i className="fas fa-sync-alt"></i> Data Settings</h3>

<div style={{ marginTop: '1rem' }}>

<div className="filter-item">

<label htmlFor="dataRefreshInterval">Data Refresh Interval</label>

<select

id="dataRefreshInterval"

className="styled-select"

value={userPreferences.dataRefreshInterval}

onChange={(e) => handlePreferenceChange('dataRefreshInterval', e.target.value)}

>

<option value="manual">Manual Only</option>

<option value="1h">Every Hour</option>

<option value="6h">Every 6 Hours</option>

<option value="12h">Every 12 Hours</option>

<option value="24h">Daily</option>

</select>

</div>

<div className="filter-item">

<label>

<input

type="checkbox"

checked={userPreferences.autoExport}

onChange={(e) => handlePreferenceChange('autoExport', e.target.checked)}

/>

Auto-Export Data Daily

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked />

Show Data Labels on Charts

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked />

Enable Animations

</label>

</div>

</div>

</div>

<div style={{ marginTop: '2rem' }}>

<button className="btn primary-btn">

<i className="fas fa-save"></i> Save Preferences

</button>

<button className="btn secondary-btn" style={{ marginLeft: '1rem' }}>

<i className="fas fa-undo"></i> Reset to Default

</button>

</div>

</div>

</div>

)}

</main>

{/\* Data Dimensions Sidebar \*/}

<aside className="data-dimensions">

<div className="filter-header">

<h3><i className="fas fa-filter"></i> Data Filters</h3>

</div>

<div className="filter-body">

<div className="filter-section">

<h4><i className="fas fa-sliders-h"></i> Filter Dimensions</h4>

<div className="filter-item">

<label htmlFor="ageGroup">Age Group</label>

<select

id="ageGroup"

className="styled-select"

value={selectedFilters.ageGroup}

onChange={(e) => handleFilterChange('ageGroup', e.target.value)}

>

<option value="all">All Age Groups</option>

<option value="0-18">0-18 years</option>

<option value="19-35">19-35 years</option>

<option value="36-50">36-50 years</option>

<option value="51-65">51-65 years</option>

<option value="65+">65+ years</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="gender">Gender</label>

<select

id="gender"

className="styled-select"

value={selectedFilters.gender}

onChange={(e) => handleFilterChange('gender', e.target.value)}

>

<option value="all">All Genders</option>

<option value="male">Male</option>

<option value="female">Female</option>

<option value="other">Other</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="race">Race/Ethnicity</label>

<select

id="race"

className="styled-select"

value={selectedFilters.race}

onChange={(e) => handleFilterChange('race', e.target.value)}

>

<option value="all">All Races</option>

<option value="white">White</option>

<option value="black">Black</option>

<option value="asian">Asian</option>

<option value="hispanic">Hispanic</option>

<option value="other">Other</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="condition">Condition</label>

<select

id="condition"

className="styled-select"

value={selectedFilters.condition}

onChange={(e) => handleFilterChange('condition', e.target.value)}

>

<option value="all">All Conditions</option>

<option value="diabetes">Diabetes</option>

<option value="hypertension">Hypertension</option>

<option value="asthma">Asthma</option>

<option value="heart">Heart Conditions</option>

<option value="cancer">Cancer</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="location">Location</label>

<select

id="location"

className="styled-select"

value={selectedFilters.location}

onChange={(e) => handleFilterChange('location', e.target.value)}

>

<option value="all">All Locations</option>

<option value="hayward">Hayward</option>

<option value="oakland">Oakland</option>

<option value="san\_francisco">San Francisco</option>

<option value="palo\_alto">Palo Alto</option>

<option value="san\_jose">San Jose</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="costRange">Cost Range</label>

<select

id="costRange"

className="styled-select"

value={selectedFilters.costRange}

onChange={(e) => handleFilterChange('costRange', e.target.value)}

>

<option value="all">All Cost Ranges</option>

<option value="low">Low (&lt; $1,000)</option>

<option value="medium">Medium ($1,000 - $5,000)</option>

<option value="high">High (&gt; $5,000)</option>

</select>

</div>

<div className="filter-item">

<label htmlFor="dateRange">Date Range</label>

<select

id="dateRange"

className="styled-select"

value={selectedFilters.dateRange}

onChange={(e) => handleFilterChange('dateRange', e.target.value)}

>

<option value="all">All Dates</option>

<option value="current\_month">Current Month</option>

<option value="last\_month">Last Month</option>

<option value="last\_3\_months">Last 3 Months</option>

<option value="last\_6\_months">Last 6 Months</option>

<option value="last\_year">Last Year</option>

</select>

</div>

</div>

<div className="filter-actions">

<button className="btn primary-btn" onClick={applyFilters}>

Apply Filters

</button>

<button className="btn secondary-btn" onClick={resetFilters}>

Reset

</button>

</div>

</div>

<div className="filter-footer">

<div className="selected-filters">

<h4>Selected Filters</h4>

<div className="filter-tags">

{Object.entries(selectedFilters).map(([key, value]) =>

value !== 'all' ? (

<div

key={key}

className="filter-tag"

onClick={() => handleFilterChange(key, 'all')}

>

{getFilterLabel(key)}: {value} <i className="fas fa-times"></i>

</div>

) : null

)}

{Object.values(selectedFilters).every(value => value === 'all') &&

<span>No filters selected</span>

}

</div>

</div>

</div>

</aside>

</div>

{/\* Help Modal \*/}

{showHelp && (

<div className="modal-overlay">

<div className="modal-content">

<div className="modal-header">

<h2><i className="fas fa-question-circle"></i> Help & Documentation</h2>

<span className="close-btn" onClick={() => setShowHelp(false)}>&times;</span>

</div>

<div className="help-content">

<div className="help-section">

<h3>Getting Started</h3>

<p>

The Healthcare Cost Prediction Dashboard provides insights into healthcare costs, trends, and predictions.

Use the navigation menu on the left to switch between different views.

</p>

<ul>

<li><strong>Overview:</strong> See all key metrics and charts at a glance</li>

<li><strong>Cost Analysis:</strong> Detailed breakdown of healthcare costs by various dimensions</li>

<li><strong>Claims:</strong> View and analyze individual claims data</li>

<li><strong>Prediction:</strong> Access cost prediction tools based on patient profiles and conditions</li>

<li><strong>Settings:</strong> Customize your user profile, notifications, and preferences</li>

</ul>

</div>

<div className="help-section">

<h3>Using Filters</h3>

<p>

Use the filters panel on the right to narrow down data by:

</p>

<ul>

<li>Age Group</li>

<li>Gender</li>

<li>Race/Ethnicity</li>

<li>Condition</li>

<li>Location</li>

<li>Cost Range</li>

<li>Date Range</li>

</ul>

<p>Click "Apply Filters" to update visualizations or "Reset" to clear all filters.</p>

</div>

<div className="help-section">

<h3>Exporting Data</h3>

<p>

Click the "Export" button in the top-right corner to download data in various formats:

</p>

<ul>

<li><strong>JSON:</strong> For data processing and integration</li>

<li><strong>CSV:</strong> For spreadsheet analysis</li>

<li><strong>Excel:</strong> For detailed data manipulation</li>

<li><strong>PDF:</strong> For reporting and sharing</li>

</ul>

</div>

<div className="help-section">

<h3>Cost Prediction</h3>

<p>

The prediction engine uses historical data and machine learning to forecast healthcare costs.

Enter patient parameters such as age, gender, and condition to generate personalized predictions.

</p>

</div>

<div className="help-section">

<h3>Need More Help?</h3>

<p>

For additional support, please contact our technical support team at support@healthcare-dashboard.com or

call (800) 555-1234 during business hours.

</p>

</div>

</div>

</div>

</div>

)}

{/\* Export Modal \*/}

{showExport && (

<div className="modal-overlay">

<div className="modal-content">

<div className="modal-header">

<h2><i className="fas fa-download"></i> Export Dashboard Data</h2>

<span className="close-btn" onClick={() => setShowExport(false)}>&times;</span>

</div>

<div>

<div className="export-options">

<div

className={`export-option ${exportFormat === 'json' ? 'active' : ''}`}

onClick={() => setExportFormat('json')}

>

<i className="fas fa-file-code"></i>

<div>JSON</div>

</div>

<div

className={`export-option ${exportFormat === 'csv' ? 'active' : ''}`}

onClick={() => setExportFormat('csv')}

>

<i className="fas fa-file-csv"></i>

<div>CSV</div>

</div>

<div

className={`export-option ${exportFormat === 'excel' ? 'active' : ''}`}

onClick={() => setExportFormat('excel')}

>

<i className="fas fa-file-excel"></i>

<div>Excel</div>

</div>

<div

className={`export-option ${exportFormat === 'pdf' ? 'active' : ''}`}

onClick={() => setExportFormat('pdf')}

>

<i className="fas fa-file-pdf"></i>

<div>PDF</div>

</div>

</div>

<div className="export-settings">

<h3>Export Settings</h3>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked /> Include KPI metrics

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked /> Include charts data

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked /> Include claims data

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked /> Include filter settings

</label>

</div>

{exportFormat === 'excel' && (

<>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked /> Include formatted tables

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked /> Auto-size columns

</label>

</div>

</>

)}

{exportFormat === 'pdf' && (

<>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked /> Include visualizations

</label>

</div>

<div className="filter-item">

<label>

<input type="checkbox" defaultChecked /> Add page numbers

</label>

</div>

</>

)}

</div>

<div className="filter-actions">

<button className="btn primary-btn" onClick={exportData}>

<i className="fas fa-download"></i> Export Data ({exportFormat.toUpperCase()})

</button>

<button className="btn secondary-btn" onClick={() => setShowExport(false)}>

Cancel

</button>

</div>

</div>

</div>

</div>

)}

</div>

);

};

export default HealthcareDashboard;

index.css

body {

  margin: 0;

  padding: 0;

  font-family: 'Poppins', -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

    sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

  box-sizing: border-box;

}

\* {

  box-sizing: border-box;

}

index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>

);