# Data Ingestion UI Components

## Overview

This document provides comprehensive specifications for all Data Ingestion user interface components, including page layouts, component structures, state management, and integration patterns. The UI is built using React with TypeScript, shadcn/ui components, and follows modern design patterns for enterprise applications.

## Architecture Overview

### Component Hierarchy

DataIngestionLayout  
├── DataIngestionDashboard (Overview & Statistics)  
├── IngestionJobsPage (Active & Historical Jobs)  
├── BatchMonitoringPage (Real-time Batch Tracking)  
├── DataValidationPage (Pre-ingestion Validation)  
├── ErrorReportingPage (Error Analysis & Resolution)  
└── IngestionSettingsPage (Configuration & Preferences)

### Technology Stack

* **React 18+** with TypeScript
* **shadcn/ui** component library
* **Tanstack Query** for data fetching
* **Zustand** for state management
* **React Hook Form** for form handling
* **Zod** for validation
* **Recharts** for data visualization

## Page Components

### 1. Data Ingestion Layout

#### File: client/src/components/layout/DataIngestionLayout.tsx

import React from 'react';  
import { Outlet, useLocation } from 'wouter';  
import { Sidebar } from '@/components/ui/sidebar';  
import { Header } from '@/components/ui/header';  
import { Breadcrumb } from '@/components/ui/breadcrumb';  
import {   
 Database,   
 Upload,   
 Activity,   
 AlertTriangle,   
 Settings,  
 BarChart3  
} from 'lucide-react';  
  
interface NavigationItem {  
 title: string;  
 href: string;  
 icon: React.ComponentType<any>;  
 description: string;  
}  
  
const navigationItems: NavigationItem[] = [  
 {  
 title: 'Dashboard',  
 href: '/ingestion',  
 icon: BarChart3,  
 description: 'Overview and statistics'  
 },  
 {  
 title: 'Ingestion Jobs',  
 href: '/ingestion/jobs',  
 icon: Upload,  
 description: 'Manage data import operations'  
 },  
 {  
 title: 'Batch Monitoring',  
 href: '/ingestion/batches',  
 icon: Activity,  
 description: 'Real-time batch progress tracking'  
 },  
 {  
 title: 'Data Validation',  
 href: '/ingestion/validation',  
 icon: Database,  
 description: 'Pre-ingestion data validation'  
 },  
 {  
 title: 'Error Reports',  
 href: '/ingestion/errors',  
 icon: AlertTriangle,  
 description: 'Error analysis and resolution'  
 },  
 {  
 title: 'Settings',  
 href: '/ingestion/settings',  
 icon: Settings,  
 description: 'Configuration and preferences'  
 }  
];  
  
export function DataIngestionLayout() {  
 const [location] = useLocation();  
   
 const currentItem = navigationItems.find(item =>   
 location === item.href || location.startsWith(item.href + '/')  
 );  
  
 return (  
 <div className="flex h-screen bg-background">  
 <Sidebar   
 items={navigationItems}  
 currentPath={location}  
 title="Data Ingestion"  
 />  
   
 <div className="flex-1 flex flex-col overflow-hidden">  
 <Header title="Data Ingestion System" />  
   
 <div className="flex-1 overflow-auto p-6">  
 <div className="max-w-7xl mx-auto space-y-6">  
 <Breadcrumb   
 items={[  
 { label: 'Home', href: '/' },  
 { label: 'Data Ingestion', href: '/ingestion' },  
 ...(currentItem ? [{ label: currentItem.title, href: currentItem.href }] : [])  
 ]}  
 />  
   
 <Outlet />  
 </div>  
 </div>  
 </div>  
 </div>  
 );  
}

### 2. Data Ingestion Dashboard

#### File: client/src/pages/ingestion/DataIngestionDashboard.tsx

import React from 'react';  
import { useQuery } from '@tanstack/react-query';  
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from '@/components/ui/card';  
import { Badge } from '@/components/ui/badge';  
import { Progress } from '@/components/ui/progress';  
import { Button } from '@/components/ui/button';  
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';  
import {   
 BarChart,   
 Bar,   
 XAxis,   
 YAxis,   
 CartesianGrid,   
 Tooltip,   
 ResponsiveContainer,  
 LineChart,  
 Line,  
 PieChart,  
 Pie,  
 Cell  
} from 'recharts';  
import {   
 Database,   
 Upload,   
 CheckCircle,   
 XCircle,   
 Clock,  
 TrendingUp,  
 Server,  
 Shield  
} from 'lucide-react';  
  
interface IngestionStats {  
 totalBatches: number;  
 successfulBatches: number;  
 failedBatches: number;  
 totalAssets: number;  
 totalVulnerabilities: number;  
 totalControls: number;  
 totalPoams: number;  
 averageProcessingTime: number;  
 systemBreakdown: {  
 tenable: { batches: number; assets: number; vulnerabilities: number; };  
 xacta: { batches: number; systems: number; controls: number; poams: number; };  
 };  
 vulnerabilityBySeverity: {  
 critical: number;  
 high: number;  
 medium: number;  
 low: number;  
 info: number;  
 };  
 recentActivity: Array<{  
 date: string;  
 batches: number;  
 records: number;  
 successRate: number;  
 }>;  
}  
  
export function DataIngestionDashboard() {  
 const { data: stats, isLoading, error } = useQuery<IngestionStats>({  
 queryKey: ['/api/ingestion/statistics'],  
 refetchInterval: 30000 // Refresh every 30 seconds  
 });  
  
 const { data: recentBatches } = useQuery({  
 queryKey: ['/api/ingestion/batches', { limit: 5, orderBy: 'started\_at', order: 'desc' }]  
 });  
  
 if (isLoading) {  
 return (  
 <div className="flex items-center justify-center h-64">  
 <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-primary"></div>  
 </div>  
 );  
 }  
  
 if (error) {  
 return (  
 <Card>  
 <CardContent className="pt-6">  
 <div className="text-center text-destructive">  
 Failed to load ingestion statistics  
 </div>  
 </CardContent>  
 </Card>  
 );  
 }  
  
 const successRate = stats ? (stats.successfulBatches / stats.totalBatches) \* 100 : 0;  
   
 const severityData = stats ? [  
 { name: 'Critical', value: stats.vulnerabilityBySeverity.critical, color: '#dc2626' },  
 { name: 'High', value: stats.vulnerabilityBySeverity.high, color: '#ea580c' },  
 { name: 'Medium', value: stats.vulnerabilityBySeverity.medium, color: '#d97706' },  
 { name: 'Low', value: stats.vulnerabilityBySeverity.low, color: '#65a30d' },  
 { name: 'Info', value: stats.vulnerabilityBySeverity.info, color: '#6b7280' }  
 ] : [];  
  
 return (  
 <div className="space-y-6">  
 {/\* Header \*/}  
 <div className="flex justify-between items-center">  
 <div>  
 <h1 className="text-3xl font-bold tracking-tight">Data Ingestion Dashboard</h1>  
 <p className="text-muted-foreground">  
 Monitor data import operations and system health  
 </p>  
 </div>  
 <Button onClick={() => window.location.href = '/ingestion/jobs'}>  
 <Upload className="mr-2 h-4 w-4" />  
 New Ingestion Job  
 </Button>  
 </div>  
  
 {/\* Key Metrics \*/}  
 <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-4">  
 <Card>  
 <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">  
 <CardTitle className="text-sm font-medium">Total Batches</CardTitle>  
 <Database className="h-4 w-4 text-muted-foreground" />  
 </CardHeader>  
 <CardContent>  
 <div className="text-2xl font-bold">{stats?.totalBatches || 0}</div>  
 <p className="text-xs text-muted-foreground">  
 {stats?.successfulBatches || 0} successful  
 </p>  
 </CardContent>  
 </Card>  
  
 <Card>  
 <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">  
 <CardTitle className="text-sm font-medium">Success Rate</CardTitle>  
 <CheckCircle className="h-4 w-4 text-muted-foreground" />  
 </CardHeader>  
 <CardContent>  
 <div className="text-2xl font-bold">{successRate.toFixed(1)}%</div>  
 <Progress value={successRate} className="mt-2" />  
 </CardContent>  
 </Card>  
  
 <Card>  
 <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">  
 <CardTitle className="text-sm font-medium">Total Assets</CardTitle>  
 <Server className="h-4 w-4 text-muted-foreground" />  
 </CardHeader>  
 <CardContent>  
 <div className="text-2xl font-bold">{stats?.totalAssets.toLocaleString() || 0}</div>  
 <p className="text-xs text-muted-foreground">  
 From {stats?.systemBreakdown.tenable.batches || 0} Tenable batches  
 </p>  
 </CardContent>  
 </Card>  
  
 <Card>  
 <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">  
 <CardTitle className="text-sm font-medium">Vulnerabilities</CardTitle>  
 <Shield className="h-4 w-4 text-muted-foreground" />  
 </CardHeader>  
 <CardContent>  
 <div className="text-2xl font-bold">{stats?.totalVulnerabilities.toLocaleString() || 0}</div>  
 <p className="text-xs text-muted-foreground">  
 {stats?.vulnerabilityBySeverity.critical || 0} critical findings  
 </p>  
 </CardContent>  
 </Card>  
 </div>  
  
 {/\* Charts and Analysis \*/}  
 <Tabs defaultValue="activity" className="space-y-4">  
 <TabsList>  
 <TabsTrigger value="activity">Recent Activity</TabsTrigger>  
 <TabsTrigger value="severity">Vulnerability Breakdown</TabsTrigger>  
 <TabsTrigger value="systems">System Sources</TabsTrigger>  
 </TabsList>  
  
 <TabsContent value="activity" className="space-y-4">  
 <Card>  
 <CardHeader>  
 <CardTitle>Ingestion Activity (Last 30 Days)</CardTitle>  
 <CardDescription>  
 Daily ingestion volume and success rates  
 </CardDescription>  
 </CardHeader>  
 <CardContent>  
 <ResponsiveContainer width="100%" height={300}>  
 <LineChart data={stats?.recentActivity || []}>  
 <CartesianGrid strokeDasharray="3 3" />  
 <XAxis dataKey="date" />  
 <YAxis yAxisId="left" />  
 <YAxis yAxisId="right" orientation="right" />  
 <Tooltip />  
 <Bar yAxisId="left" dataKey="records" fill="#3b82f6" />  
 <Line yAxisId="right" type="monotone" dataKey="successRate" stroke="#10b981" />  
 </LineChart>  
 </ResponsiveContainer>  
 </CardContent>  
 </Card>  
 </TabsContent>  
  
 <TabsContent value="severity" className="space-y-4">  
 <Card>  
 <CardHeader>  
 <CardTitle>Vulnerability Severity Distribution</CardTitle>  
 <CardDescription>  
 Breakdown of vulnerabilities by severity level  
 </CardDescription>  
 </CardHeader>  
 <CardContent>  
 <ResponsiveContainer width="100%" height={300}>  
 <PieChart>  
 <Pie  
 data={severityData}  
 cx="50%"  
 cy="50%"  
 labelLine={false}  
 label={({ name, percent }) => `${name} ${(percent \* 100).toFixed(0)}%`}  
 outerRadius={80}  
 fill="#8884d8"  
 dataKey="value"  
 >  
 {severityData.map((entry, index) => (  
 <Cell key={`cell-${index}`} fill={entry.color} />  
 ))}  
 </Pie>  
 <Tooltip />  
 </PieChart>  
 </ResponsiveContainer>  
 </CardContent>  
 </Card>  
 </TabsContent>  
  
 <TabsContent value="systems" className="space-y-4">  
 <div className="grid gap-4 md:grid-cols-2">  
 <Card>  
 <CardHeader>  
 <CardTitle>Tenable Integration</CardTitle>  
 <CardDescription>Vulnerability scanner data</CardDescription>  
 </CardHeader>  
 <CardContent className="space-y-2">  
 <div className="flex justify-between">  
 <span>Batches Processed:</span>  
 <span className="font-semibold">{stats?.systemBreakdown.tenable.batches || 0}</span>  
 </div>  
 <div className="flex justify-between">  
 <span>Assets Imported:</span>  
 <span className="font-semibold">{stats?.systemBreakdown.tenable.assets.toLocaleString() || 0}</span>  
 </div>  
 <div className="flex justify-between">  
 <span>Vulnerabilities Found:</span>  
 <span className="font-semibold">{stats?.systemBreakdown.tenable.vulnerabilities.toLocaleString() || 0}</span>  
 </div>  
 </CardContent>  
 </Card>  
  
 <Card>  
 <CardHeader>  
 <CardTitle>Xacta Integration</CardTitle>  
 <CardDescription>Compliance management data</CardDescription>  
 </CardHeader>  
 <CardContent className="space-y-2">  
 <div className="flex justify-between">  
 <span>Batches Processed:</span>  
 <span className="font-semibold">{stats?.systemBreakdown.xacta.batches || 0}</span>  
 </div>  
 <div className="flex justify-between">  
 <span>Systems Imported:</span>  
 <span className="font-semibold">{stats?.systemBreakdown.xacta.systems.toLocaleString() || 0}</span>  
 </div>  
 <div className="flex justify-between">  
 <span>Controls Assessed:</span>  
 <span className="font-semibold">{stats?.systemBreakdown.xacta.controls.toLocaleString() || 0}</span>  
 </div>  
 <div className="flex justify-between">  
 <span>POAMs Tracked:</span>  
 <span className="font-semibold">{stats?.systemBreakdown.xacta.poams.toLocaleString() || 0}</span>  
 </div>  
 </CardContent>  
 </Card>  
 </div>  
 </TabsContent>  
 </Tabs>  
  
 {/\* Recent Batches \*/}  
 <Card>  
 <CardHeader>  
 <CardTitle>Recent Ingestion Batches</CardTitle>  
 <CardDescription>Latest data import operations</CardDescription>  
 </CardHeader>  
 <CardContent>  
 <div className="space-y-3">  
 {recentBatches?.batches?.map((batch: any) => (  
 <div key={batch.batchId} className="flex items-center justify-between p-3 border rounded-lg">  
 <div className="flex items-center space-x-3">  
 <div className={`h-3 w-3 rounded-full ${  
 batch.status === 'completed' ? 'bg-green-500' :  
 batch.status === 'failed' ? 'bg-red-500' :  
 'bg-yellow-500'  
 }`} />  
 <div>  
 <p className="font-medium">{batch.sourceSystem} - {batch.batchType}</p>  
 <p className="text-sm text-muted-foreground">  
 {batch.successfulRecords}/{batch.totalRecords} records processed  
 </p>  
 </div>  
 </div>  
 <div className="text-right">  
 <Badge variant={  
 batch.status === 'completed' ? 'default' :  
 batch.status === 'failed' ? 'destructive' :  
 'secondary'  
 }>  
 {batch.status}  
 </Badge>  
 <p className="text-sm text-muted-foreground mt-1">  
 {new Date(batch.startedAt).toLocaleString()}  
 </p>  
 </div>  
 </div>  
 )) || (  
 <div className="text-center text-muted-foreground py-4">  
 No recent batches found  
 </div>  
 )}  
 </div>  
 </CardContent>  
 </Card>  
 </div>  
 );  
}

### 3. Ingestion Jobs Management

#### File: client/src/pages/ingestion/IngestionJobsPage.tsx

import React, { useState } from 'react';  
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';  
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from '@/components/ui/card';  
import { Button } from '@/components/ui/button';  
import { Input } from '@/components/ui/input';  
import { Label } from '@/components/ui/label';  
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from '@/components/ui/select';  
import { Textarea } from '@/components/ui/textarea';  
import { Badge } from '@/components/ui/badge';  
import { Dialog, DialogContent, DialogDescription, DialogHeader, DialogTitle, DialogTrigger } from '@/components/ui/dialog';  
import { Table, TableBody, TableCell, TableHead, TableHeader, TableRow } from '@/components/ui/table';  
import { useToast } from '@/hooks/use-toast';  
import { Upload, Play, Pause, Trash2, Eye, Plus } from 'lucide-react';  
  
interface IngestionJob {  
 id: number;  
 batchId: string;  
 sourceSystem: string;  
 batchType: string;  
 fileName?: string;  
 totalRecords?: number;  
 successfulRecords: number;  
 failedRecords: number;  
 status: string;  
 startedAt: string;  
 completedAt?: string;  
 createdBy: number;  
}  
  
export function IngestionJobsPage() {  
 const [selectedSourceSystem, setSelectedSourceSystem] = useState<string>('');  
 const [isCreateDialogOpen, setIsCreateDialogOpen] = useState(false);  
 const { toast } = useToast();  
 const queryClient = useQueryClient();  
  
 const { data: jobs, isLoading } = useQuery<{ jobs: IngestionJob[] }>({  
 queryKey: ['/api/ingestion/batches', {   
 sourceSystem: selectedSourceSystem || undefined,  
 limit: 50   
 }]  
 });  
  
 const createJobMutation = useMutation({  
 mutationFn: async (data: FormData) => {  
 const response = await fetch('/api/ingestion/upload', {  
 method: 'POST',  
 body: data  
 });  
 if (!response.ok) throw new Error('Upload failed');  
 return response.json();  
 },  
 onSuccess: () => {  
 toast({  
 title: 'Success',  
 description: 'Ingestion job started successfully'  
 });  
 queryClient.invalidateQueries({ queryKey: ['/api/ingestion/batches'] });  
 setIsCreateDialogOpen(false);  
 },  
 onError: (error: Error) => {  
 toast({  
 title: 'Error',  
 description: error.message,  
 variant: 'destructive'  
 });  
 }  
 });  
  
 const deleteJobMutation = useMutation({  
 mutationFn: async (batchId: string) => {  
 const response = await fetch(`/api/ingestion/batch/${batchId}`, {  
 method: 'DELETE'  
 });  
 if (!response.ok) throw new Error('Delete failed');  
 return response.json();  
 },  
 onSuccess: () => {  
 toast({  
 title: 'Success',  
 description: 'Ingestion job deleted successfully'  
 });  
 queryClient.invalidateQueries({ queryKey: ['/api/ingestion/batches'] });  
 }  
 });  
  
 const handleFileUpload = (event: React.FormEvent<HTMLFormElement>) => {  
 event.preventDefault();  
 const formData = new FormData(event.currentTarget);  
 createJobMutation.mutate(formData);  
 };  
  
 const getStatusColor = (status: string) => {  
 switch (status) {  
 case 'completed': return 'default';  
 case 'failed': return 'destructive';  
 case 'in\_progress': return 'secondary';  
 default: return 'outline';  
 }  
 };  
  
 return (  
 <div className="space-y-6">  
 {/\* Header \*/}  
 <div className="flex justify-between items-center">  
 <div>  
 <h1 className="text-3xl font-bold tracking-tight">Ingestion Jobs</h1>  
 <p className="text-muted-foreground">  
 Manage data import operations and monitor progress  
 </p>  
 </div>  
   
 <Dialog open={isCreateDialogOpen} onOpenChange={setIsCreateDialogOpen}>  
 <DialogTrigger asChild>  
 <Button>  
 <Plus className="mr-2 h-4 w-4" />  
 New Job  
 </Button>  
 </DialogTrigger>  
 <DialogContent className="max-w-md">  
 <DialogHeader>  
 <DialogTitle>Create Ingestion Job</DialogTitle>  
 <DialogDescription>  
 Upload a data file to start a new ingestion job  
 </DialogDescription>  
 </DialogHeader>  
   
 <form onSubmit={handleFileUpload} className="space-y-4">  
 <div className="space-y-2">  
 <Label htmlFor="sourceSystem">Source System</Label>  
 <Select name="sourceSystem" required>  
 <SelectTrigger>  
 <SelectValue placeholder="Select source system" />  
 </SelectTrigger>  
 <SelectContent>  
 <SelectItem value="tenable">Tenable</SelectItem>  
 <SelectItem value="xacta">Xacta</SelectItem>  
 </SelectContent>  
 </Select>  
 </div>  
  
 <div className="space-y-2">  
 <Label htmlFor="batchType">Data Type</Label>  
 <Select name="batchType" required>  
 <SelectTrigger>  
 <SelectValue placeholder="Select data type" />  
 </SelectTrigger>  
 <SelectContent>  
 <SelectItem value="assets">Assets</SelectItem>  
 <SelectItem value="vulnerabilities">Vulnerabilities</SelectItem>  
 <SelectItem value="systems">Systems</SelectItem>  
 <SelectItem value="controls">Controls</SelectItem>  
 <SelectItem value="poams">POAMs</SelectItem>  
 </SelectContent>  
 </Select>  
 </div>  
  
 <div className="space-y-2">  
 <Label htmlFor="file">Data File</Label>  
 <Input  
 id="file"  
 name="file"  
 type="file"  
 accept=".json,.csv,.xml"  
 required  
 />  
 <p className="text-xs text-muted-foreground">  
 Supported formats: JSON, CSV, XML (Max 50MB)  
 </p>  
 </div>  
  
 <div className="space-y-2">  
 <Label htmlFor="description">Description (Optional)</Label>  
 <Textarea  
 id="description"  
 name="description"  
 placeholder="Brief description of this ingestion job"  
 rows={3}  
 />  
 </div>  
  
 <div className="flex justify-end space-x-2">  
 <Button   
 type="button"   
 variant="outline"   
 onClick={() => setIsCreateDialogOpen(false)}  
 >  
 Cancel  
 </Button>  
 <Button   
 type="submit"   
 disabled={createJobMutation.isPending}  
 >  
 {createJobMutation.isPending ? (  
 <>  
 <div className="animate-spin rounded-full h-4 w-4 border-b-2 border-white mr-2"></div>  
 Creating...  
 </>  
 ) : (  
 <>  
 <Upload className="mr-2 h-4 w-4" />  
 Create Job  
 </>  
 )}  
 </Button>  
 </div>  
 </form>  
 </DialogContent>  
 </Dialog>  
 </div>  
  
 {/\* Filters \*/}  
 <Card>  
 <CardHeader>  
 <CardTitle>Filters</CardTitle>  
 </CardHeader>  
 <CardContent>  
 <div className="flex space-x-4">  
 <div className="flex-1">  
 <Label htmlFor="sourceFilter">Source System</Label>  
 <Select value={selectedSourceSystem} onValueChange={setSelectedSourceSystem}>  
 <SelectTrigger>  
 <SelectValue placeholder="All source systems" />  
 </SelectTrigger>  
 <SelectContent>  
 <SelectItem value="">All Systems</SelectItem>  
 <SelectItem value="tenable">Tenable</SelectItem>  
 <SelectItem value="xacta">Xacta</SelectItem>  
 </SelectContent>  
 </Select>  
 </div>  
 </div>  
 </CardContent>  
 </Card>  
  
 {/\* Jobs Table \*/}  
 <Card>  
 <CardHeader>  
 <CardTitle>Ingestion Jobs</CardTitle>  
 <CardDescription>  
 All data import operations and their current status  
 </CardDescription>  
 </CardHeader>  
 <CardContent>  
 {isLoading ? (  
 <div className="flex items-center justify-center h-32">  
 <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-primary"></div>  
 </div>  
 ) : (  
 <Table>  
 <TableHeader>  
 <TableRow>  
 <TableHead>Job Details</TableHead>  
 <TableHead>Progress</TableHead>  
 <TableHead>Status</TableHead>  
 <TableHead>Created</TableHead>  
 <TableHead>Actions</TableHead>  
 </TableRow>  
 </TableHeader>  
 <TableBody>  
 {jobs?.jobs?.map((job) => (  
 <TableRow key={job.batchId}>  
 <TableCell>  
 <div>  
 <p className="font-medium">  
 {job.sourceSystem} - {job.batchType}  
 </p>  
 {job.fileName && (  
 <p className="text-sm text-muted-foreground">  
 {job.fileName}  
 </p>  
 )}  
 <p className="text-xs text-muted-foreground">  
 ID: {job.batchId.substring(0, 8)}...  
 </p>  
 </div>  
 </TableCell>  
 <TableCell>  
 <div>  
 <p className="text-sm">  
 {job.successfulRecords}/{job.totalRecords || 0} records  
 </p>  
 {job.failedRecords > 0 && (  
 <p className="text-sm text-destructive">  
 {job.failedRecords} failed  
 </p>  
 )}  
 </div>  
 </TableCell>  
 <TableCell>  
 <Badge variant={getStatusColor(job.status)}>  
 {job.status}  
 </Badge>  
 </TableCell>  
 <TableCell>  
 <p className="text-sm">  
 {new Date(job.startedAt).toLocaleDateString()}  
 </p>  
 <p className="text-xs text-muted-foreground">  
 {new Date(job.startedAt).toLocaleTimeString()}  
 </p>  
 </TableCell>  
 <TableCell>  
 <div className="flex space-x-2">  
 <Button  
 size="sm"  
 variant="outline"  
 onClick={() => window.location.href = `/ingestion/batch/${job.batchId}`}  
 >  
 <Eye className="h-4 w-4" />  
 </Button>  
   
 {(job.status === 'completed' || job.status === 'failed') && (  
 <Button  
 size="sm"  
 variant="outline"  
 onClick={() => deleteJobMutation.mutate(job.batchId)}  
 disabled={deleteJobMutation.isPending}  
 >  
 <Trash2 className="h-4 w-4" />  
 </Button>  
 )}  
 </div>  
 </TableCell>  
 </TableRow>  
 ))}  
 </TableBody>  
 </Table>  
 )}  
 </CardContent>  
 </Card>  
 </div>  
 );  
}

### 4. Batch Monitoring Page

#### File: client/src/pages/ingestion/BatchMonitoringPage.tsx

import React, { useState } from 'react';  
import { useParams } from 'wouter';  
import { useQuery } from '@tanstack/react-query';  
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from '@/components/ui/card';  
import { Badge } from '@/components/ui/badge';  
import { Progress } from '@/components/ui/progress';  
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';  
import { Alert, AlertDescription } from '@/components/ui/alert';  
import { Table, TableBody, TableCell, TableHead, TableHeader, TableRow } from '@/components/ui/table';  
import { CheckCircle, XCircle, Clock, AlertTriangle, Database, FileText } from 'lucide-react';  
  
interface BatchDetails {  
 batch: {  
 id: number;  
 batchId: string;  
 sourceSystem: string;  
 batchType: string;  
 fileName?: string;  
 totalRecords?: number;  
 successfulRecords: number;  
 failedRecords: number;  
 status: string;  
 startedAt: string;  
 completedAt?: string;  
 errorDetails?: string;  
 metadata?: any;  
 processingTimeSeconds?: number;  
 };  
 errors?: Array<{  
 id: number;  
 tableName: string;  
 recordIdentifier: string;  
 errorType: string;  
 errorMessage: string;  
 createdAt: string;  
 }>;  
}  
  
export function BatchMonitoringPage() {  
 const { batchId } = useParams<{ batchId: string }>();  
   
 const { data: batchDetails, isLoading } = useQuery<BatchDetails>({  
 queryKey: [`/api/ingestion/batch/${batchId}/status`],  
 refetchInterval: (data) => {  
 // Stop refetching if batch is completed or failed  
 return data?.batch.status === 'in\_progress' ? 5000 : false;  
 }  
 });  
  
 if (isLoading) {  
 return (  
 <div className="flex items-center justify-center h-64">  
 <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-primary"></div>  
 </div>  
 );  
 }  
  
 if (!batchDetails) {  
 return (  
 <Alert>  
 <AlertTriangle className="h-4 w-4" />  
 <AlertDescription>  
 Batch not found or you don't have permission to view it.  
 </AlertDescription>  
 </Alert>  
 );  
 }  
  
 const { batch, errors } = batchDetails;  
 const progressPercentage = batch.totalRecords ?   
 ((batch.successfulRecords + batch.failedRecords) / batch.totalRecords) \* 100 : 0;  
  
 const getStatusIcon = (status: string) => {  
 switch (status) {  
 case 'completed': return <CheckCircle className="h-5 w-5 text-green-500" />;  
 case 'failed': return <XCircle className="h-5 w-5 text-red-500" />;  
 case 'in\_progress': return <Clock className="h-5 w-5 text-yellow-500" />;  
 default: return <Clock className="h-5 w-5 text-gray-500" />;  
 }  
 };  
  
 return (  
 <div className="space-y-6">  
 {/\* Header \*/}  
 <div className="flex items-center space-x-3">  
 {getStatusIcon(batch.status)}  
 <div>  
 <h1 className="text-3xl font-bold tracking-tight">  
 Batch Monitoring  
 </h1>  
 <p className="text-muted-foreground">  
 {batch.sourceSystem} - {batch.batchType} Import  
 </p>  
 </div>  
 </div>  
  
 {/\* Status Overview \*/}  
 <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-4">  
 <Card>  
 <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">  
 <CardTitle className="text-sm font-medium">Status</CardTitle>  
 <Database className="h-4 w-4 text-muted-foreground" />  
 </CardHeader>  
 <CardContent>  
 <Badge variant={  
 batch.status === 'completed' ? 'default' :  
 batch.status === 'failed' ? 'destructive' :  
 'secondary'  
 }>  
 {batch.status}  
 </Badge>  
 </CardContent>  
 </Card>  
  
 <Card>  
 <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">  
 <CardTitle className="text-sm font-medium">Progress</CardTitle>  
 <CheckCircle className="h-4 w-4 text-muted-foreground" />  
 </CardHeader>  
 <CardContent>  
 <div className="text-2xl font-bold">  
 {batch.totalRecords ?   
 `${Math.round(progressPercentage)}%` :   
 'Unknown'  
 }  
 </div>  
 {batch.totalRecords && (  
 <Progress value={progressPercentage} className="mt-2" />  
 )}  
 </CardContent>  
 </Card>  
  
 <Card>  
 <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">  
 <CardTitle className="text-sm font-medium">Records Processed</CardTitle>  
 <FileText className="h-4 w-4 text-muted-foreground" />  
 </CardHeader>  
 <CardContent>  
 <div className="text-2xl font-bold">  
 {(batch.successfulRecords + batch.failedRecords).toLocaleString()}  
 </div>  
 <p className="text-xs text-muted-foreground">  
 of {batch.totalRecords?.toLocaleString() || 'unknown'} total  
 </p>  
 </CardContent>  
 </Card>  
  
 <Card>  
 <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">  
 <CardTitle className="text-sm font-medium">Success Rate</CardTitle>  
 <CheckCircle className="h-4 w-4 text-muted-foreground" />  
 </CardHeader>  
 <CardContent>  
 <div className="text-2xl font-bold">  
 {batch.successfulRecords + batch.failedRecords > 0 ?   
 `${Math.round((batch.successfulRecords / (batch.successfulRecords + batch.failedRecords)) \* 100)}%` :  
 '0%'  
 }  
 </div>  
 <p className="text-xs text-muted-foreground">  
 {batch.successfulRecords} successful  
 </p>  
 </CardContent>  
 </Card>  
 </div>  
  
 {/\* Detailed Information \*/}  
 <Tabs defaultValue="details" className="space-y-4">  
 <TabsList>  
 <TabsTrigger value="details">Batch Details</TabsTrigger>  
 <TabsTrigger value="errors">  
 Errors ({errors?.length || 0})  
 </TabsTrigger>  
 <TabsTrigger value="metadata">Metadata</TabsTrigger>  
 </TabsList>  
  
 <TabsContent value="details" className="space-y-4">  
 <Card>  
 <CardHeader>  
 <CardTitle>Batch Information</CardTitle>  
 </CardHeader>  
 <CardContent className="space-y-4">  
 <div className="grid gap-4 md:grid-cols-2">  
 <div>  
 <Label className="text-sm font-medium">Batch ID</Label>  
 <p className="text-sm text-muted-foreground font-mono">  
 {batch.batchId}  
 </p>  
 </div>  
   
 <div>  
 <Label className="text-sm font-medium">Source System</Label>  
 <p className="text-sm text-muted-foreground">  
 {batch.sourceSystem}  
 </p>  
 </div>  
   
 <div>  
 <Label className="text-sm font-medium">Data Type</Label>  
 <p className="text-sm text-muted-foreground">  
 {batch.batchType}  
 </p>  
 </div>  
   
 {batch.fileName && (  
 <div>  
 <Label className="text-sm font-medium">File Name</Label>  
 <p className="text-sm text-muted-foreground">  
 {batch.fileName}  
 </p>  
 </div>  
 )}  
   
 <div>  
 <Label className="text-sm font-medium">Started At</Label>  
 <p className="text-sm text-muted-foreground">  
 {new Date(batch.startedAt).toLocaleString()}  
 </p>  
 </div>  
   
 {batch.completedAt && (  
 <div>  
 <Label className="text-sm font-medium">Completed At</Label>  
 <p className="text-sm text-muted-foreground">  
 {new Date(batch.completedAt).toLocaleString()}  
 </p>  
 </div>  
 )}  
   
 {batch.processingTimeSeconds && (  
 <div>  
 <Label className="text-sm font-medium">Processing Time</Label>  
 <p className="text-sm text-muted-foreground">  
 {Math.round(batch.processingTimeSeconds)} seconds  
 </p>  
 </div>  
 )}  
 </div>  
  
 {batch.errorDetails && (  
 <Alert>  
 <AlertTriangle className="h-4 w-4" />  
 <AlertDescription>  
 {batch.errorDetails}  
 </AlertDescription>  
 </Alert>  
 )}  
 </CardContent>  
 </Card>  
 </TabsContent>  
  
 <TabsContent value="errors" className="space-y-4">  
 <Card>  
 <CardHeader>  
 <CardTitle>Processing Errors</CardTitle>  
 <CardDescription>  
 Records that failed to process during ingestion  
 </CardDescription>  
 </CardHeader>  
 <CardContent>  
 {errors && errors.length > 0 ? (  
 <Table>  
 <TableHeader>  
 <TableRow>  
 <TableHead>Table</TableHead>  
 <TableHead>Record ID</TableHead>  
 <TableHead>Error Type</TableHead>  
 <TableHead>Error Message</TableHead>  
 <TableHead>Time</TableHead>  
 </TableRow>  
 </TableHeader>  
 <TableBody>  
 {errors.map((error) => (  
 <TableRow key={error.id}>  
 <TableCell className="font-mono text-sm">  
 {error.tableName}  
 </TableCell>  
 <TableCell className="font-mono text-sm">  
 {error.recordIdentifier}  
 </TableCell>  
 <TableCell>  
 <Badge variant="outline">  
 {error.errorType}  
 </Badge>  
 </TableCell>  
 <TableCell className="max-w-md">  
 <p className="text-sm truncate" title={error.errorMessage}>  
 {error.errorMessage}  
 </p>  
 </TableCell>  
 <TableCell className="text-sm text-muted-foreground">  
 {new Date(error.createdAt).toLocaleTimeString()}  
 </TableCell>  
 </TableRow>  
 ))}  
 </TableBody>  
 </Table>  
 ) : (  
 <div className="text-center text-muted-foreground py-8">  
 No errors recorded for this batch  
 </div>  
 )}  
 </CardContent>  
 </Card>  
 </TabsContent>  
  
 <TabsContent value="metadata" className="space-y-4">  
 <Card>  
 <CardHeader>  
 <CardTitle>Batch Metadata</CardTitle>  
 <CardDescription>  
 Additional information about this ingestion job  
 </CardDescription>  
 </CardHeader>  
 <CardContent>  
 {batch.metadata ? (  
 <pre className="bg-muted p-4 rounded-lg overflow-auto text-sm">  
 {JSON.stringify(batch.metadata, null, 2)}  
 </pre>  
 ) : (  
 <div className="text-center text-muted-foreground py-8">  
 No metadata available for this batch  
 </div>  
 )}  
 </CardContent>  
 </Card>  
 </TabsContent>  
 </Tabs>  
 </div>  
 );  
}

## State Management

### Ingestion Store (Zustand)

#### File: client/src/stores/ingestionStore.ts

import { create } from 'zustand';  
import { subscribeWithSelector } from 'zustand/middleware';  
  
interface IngestionState {  
 // Active ingestion jobs  
 activeJobs: Map<string, any>;  
   
 // Real-time statistics  
 statistics: {  
 totalBatches: number;  
 activeBatches: number;  
 successRate: number;  
 lastUpdated: Date | null;  
 };  
   
 // UI state  
 selectedSourceSystem: string;  
 isMonitoring: boolean;  
   
 // Actions  
 addActiveJob: (job: any) => void;  
 removeActiveJob: (batchId: string) => void;  
 updateJobStatus: (batchId: string, status: any) => void;  
 updateStatistics: (stats: any) => void;  
 setSelectedSourceSystem: (system: string) => void;  
 startMonitoring: () => void;  
 stopMonitoring: () => void;  
}  
  
export const useIngestionStore = create<IngestionState>()(  
 subscribeWithSelector((set, get) => ({  
 activeJobs: new Map(),  
 statistics: {  
 totalBatches: 0,  
 activeBatches: 0,  
 successRate: 0,  
 lastUpdated: null  
 },  
 selectedSourceSystem: '',  
 isMonitoring: false,  
   
 addActiveJob: (job) => set((state) => {  
 const newJobs = new Map(state.activeJobs);  
 newJobs.set(job.batchId, job);  
 return { activeJobs: newJobs };  
 }),  
   
 removeActiveJob: (batchId) => set((state) => {  
 const newJobs = new Map(state.activeJobs);  
 newJobs.delete(batchId);  
 return { activeJobs: newJobs };  
 }),  
   
 updateJobStatus: (batchId, status) => set((state) => {  
 const newJobs = new Map(state.activeJobs);  
 const existingJob = newJobs.get(batchId);  
 if (existingJob) {  
 newJobs.set(batchId, { ...existingJob, ...status });  
 }  
 return { activeJobs: newJobs };  
 }),  
   
 updateStatistics: (stats) => set({  
 statistics: {  
 ...stats,  
 lastUpdated: new Date()  
 }  
 }),  
   
 setSelectedSourceSystem: (system) => set({ selectedSourceSystem: system }),  
   
 startMonitoring: () => set({ isMonitoring: true }),  
   
 stopMonitoring: () => set({ isMonitoring: false })  
 }))  
);  
  
// Subscribe to active jobs changes to update statistics  
useIngestionStore.subscribe(  
 (state) => state.activeJobs,  
 (activeJobs) => {  
 const activeBatches = activeJobs.size;  
 // Update statistics based on active jobs  
 useIngestionStore.getState().updateStatistics({  
 ...useIngestionStore.getState().statistics,  
 activeBatches  
 });  
 }  
);

## Integration Guide

### Adding to Your Application

1. **Install Dependencies**:

npm install @tanstack/react-query zustand react-hook-form @hookform/resolvers zod recharts

1. **Add Routes to App.tsx**:

import { DataIngestionLayout } from '@/components/layout/DataIngestionLayout';  
import { DataIngestionDashboard } from '@/pages/ingestion/DataIngestionDashboard';  
import { IngestionJobsPage } from '@/pages/ingestion/IngestionJobsPage';  
import { BatchMonitoringPage } from '@/pages/ingestion/BatchMonitoringPage';  
  
// Add routes  
<Route path="/ingestion" component={DataIngestionLayout}>  
 <Route path="" component={DataIngestionDashboard} />  
 <Route path="/jobs" component={IngestionJobsPage} />  
 <Route path="/batch/:batchId" component={BatchMonitoringPage} />  
</Route>

1. **Configure Query Client**:

import { QueryClient, QueryClientProvider } from '@tanstack/react-query';  
  
const queryClient = new QueryClient({  
 defaultOptions: {  
 queries: {  
 staleTime: 5 \* 60 \* 1000, // 5 minutes  
 retry: 3  
 }  
 }  
});

## Responsive Design

All components are built with responsive design principles: - **Mobile**: Single column layout with stacked cards - **Tablet**: Two-column grid with collapsible sidebar - **Desktop**: Full multi-column layout with persistent sidebar

## Accessibility Features

* **Keyboard Navigation**: All interactive elements are keyboard accessible
* **Screen Reader Support**: Proper ARIA labels and semantic HTML
* **Color Contrast**: Meets WCAG 2.1 AA standards
* **Focus Management**: Clear focus indicators and logical tab order

This comprehensive UI component specification provides everything needed to implement a professional, enterprise-grade data ingestion interface with modern React patterns and best practices.