**Executive Summary**

This AI-powered medical education analytics platform provides comprehensive insights into ACGME (Accreditation Council for Graduate Medical Education) data through intelligent dashboard generation and real-time data visualization. Built for demonstration to ACGME colleagues and stakeholders, the platform supports role-based access for Program Directors, DIOs, and Institutional Coordinators.

**A) Summary of Features**

**Core Analytics Features**

* **AI-Powered Dashboard Generation**: Natural language query processing using OpenAI to generate contextually relevant charts and insights
* **Real-Time ACGME Data Integration**: Connects to 8 authentic ACGME public data source endpoints with automatic caching and refresh cycles
* **Geographic Program Distribution**: Interactive US map visualization showing regional program counts across all medical specialties
* **Specialty-Specific Analytics**: Comprehensive analysis of 110+ medical specialties with trend predictions and growth analysis
* **Role-Based Data Access**: Implements row-level security for different user types (Public, Program Director, DIO, Institutional Coordinator)

**Advanced Visualization Features**

* **Professional Chart Library**: Chart.js integration with animations, export capabilities, and responsive design
* **Predictive Analytics**: Advanced trend analysis with linear regression and future projections
* **Benchmarking Tools**: National average comparisons and percentile rankings
* **Real-Time Data Panel**: Live ACGME metrics display with top specialties and growth indicators

**User Experience Features**

* **Mobile-First Responsive Design**: Optimized for all devices with professional medical education styling
* **Favorites Management**: Users can save and organize frequently accessed dashboards
* **Natural Language Interface**: Simple English queries like "Show cardiology trends 2020-2024"
* **Professional Authentication**: Azure B2C SSO integration ready (currently simulated for demo)

**Technical Features**

* **TypeScript Full-Stack**: Type-safe development with shared schemas between frontend and backend
* **Real-Time Data Caching**: 6-hour refresh cycles with connection monitoring
* **API-First Architecture**: RESTful endpoints for all data operations
* **Database Agnostic**: Supports both in-memory (demo) and PostgreSQL (production) storage

**B) Code Files Description**

***Frontend Architecture (client/)***

**Core Application Files**

* **src/App.tsx**: Main application router using Wouter, handles routing between dashboard and authentication pages
* **src/main.tsx**: Application entry point with React Query provider and root component mounting
* **index.html**: HTML template with responsive viewport and Tailwind CSS integration

**Component Architecture (src/components/)**

* **dashboard/DashboardGrid.tsx**: Main dashboard container with real-time ACGME data panel, AI prompt interface, and chart grid layout
* **dashboard/AIPromptInterface.tsx**: Natural language query input with role-aware suggestions and loading states
* **dashboard/AdvancedChartViewer.tsx**: Professional chart renderer with Chart.js, animations, and export functionality
* **ui/**: Shadcn/ui component library including forms, buttons, dialogs, and data visualization components

**Page Components (src/pages/)**

* **dashboard.tsx**: Main dashboard page with authentication check and role-based data loading
* **not-found.tsx**: 404 error page with navigation back to dashboard

**Hooks and Utilities (src/hooks/, src/lib/)**

* **use-dashboard.tsx**: Custom hooks for dashboard generation, user favorites, AI suggestions, and analytics data
* **use-toast.ts**: Toast notification system for user feedback
* **use-mobile.tsx**: Mobile device detection for responsive behavior
* **queryClient.ts**: TanStack Query configuration with error handling and default fetchers
* **utils.ts**: Utility functions for class merging and common operations
* **types.ts**: Frontend-specific TypeScript type definitions

***Backend Architecture (server/)***

**Core Server Files**

* **index.ts**: Express server setup with middleware, error handling, and port configuration
* **routes.ts**: RESTful API endpoints for authentication, dashboards, analytics, and ACGME data
* **vite.ts**: Development server integration with Vite for hot module replacement

**Service Layer (services/)**

* **acgmeDataService.ts**: Real ACGME data integration with 8 public API endpoints, caching, and data processing
* **aiService.ts**: OpenAI integration for dashboard generation with context-aware analysis and chart configuration
* **authService.ts**: Authentication service with role-based permissions and Azure B2C integration points
* **predictiveAnalyticsService.ts**: Advanced analytics with trend analysis, linear regression, and benchmarking

**Data Layer**

* **storage.ts**: Storage interface with in-memory implementation for demo and PostgreSQL integration points

**Shared Architecture (shared/)**

* **schema.ts**: Drizzle ORM schema definitions for all database tables with TypeScript types and Zod validation

**Configuration Files**

* **package.json**: Dependencies and build scripts for full-stack TypeScript application
* **vite.config.ts**: Vite configuration with path aliases and Replit integration
* **tailwind.config.ts**: Tailwind CSS configuration with custom color palette and component styling
* **tsconfig.json**: TypeScript configuration for monorepo structure with path mapping
* **drizzle.config.ts**: Database configuration for PostgreSQL integration
* **postcss.config.js**: PostCSS configuration for Tailwind CSS processing

**C) Setup and** Running **Instructions**

**Prerequisites**

* Node.js 18+ with npm package manager
* PostgreSQL database (for production) or use in-memory storage (for demo)
* OpenAI API key for dashboard generation
* Optional: Azure B2C tenant for production authentication

**Installation Sequence**

1. **Install Dependencies**

npm install

1. **Environment Configuration** Create .env file with required variables:

OPENAI\_API\_KEY=your\_openai\_api\_key\_here

DATABASE\_URL=postgresql://username:password@host:port/database (production only)

NODE\_ENV=development

1. **Database Setup (Production Only)**

*# Generate database migrations*

npx drizzle-kit generate:pg

*# Run database migrations*

npx drizzle-kit migrate

1. **Development Server**

npm run dev

This starts both frontend (Vite) and backend (Express) servers on port 5000

1. **Production Build**

npm run build

npm start

**File Creation and Modification Sequence**

1. **Schema First**: Start with shared/schema.ts for data model definitions
2. **Storage Layer**: Implement server/storage.ts for data persistence
3. **API Routes**: Create endpoints in server/routes.ts
4. **Frontend Components**: Build UI components in client/src/components/
5. **Page Integration**: Connect components in client/src/pages/
6. **Service Integration**: Add external services in server/services/

**D) Integration Points for Production Systems**

**1. Power BI Data Integration**

**File**: server/services/acgmeDataService.ts

**Location**: Lines 45-65 in fetchACGMEData() method

*// REPLACE THIS SECTION FOR POWER BI INTEGRATION*

private async simulateACGMEDataFetch(): Promise<ProcessedACGMEData> {

*// TODO: Replace with actual Power BI REST API calls*

*// Use Power BI REST API endpoints:*

*// https://api.powerbi.com/v1.0/myorg/datasets/{datasetId}/tables/{tableName}/rows*

*// Example Power BI integration:*

*// const powerBIResponse = await fetch(`https://api.powerbi.com/v1.0/myorg/datasets/${DATASET\_ID}/executeQueries`, {*

*// method: 'POST',*

*// headers: {*

*// 'Authorization': `Bearer ${accessToken}`,*

*// 'Content-Type': 'application/json'*

*// },*

*// body: JSON.stringify({*

*// queries: [{ query: "EVALUATE SUMMARIZE(Programs, Programs[Specialty], \"Count\", COUNT(Programs[ProgramID]))" }]*

*// })*

*// });*

}

**Additional Files to Modify**:

* server/routes.ts lines 125-135: Update /api/acgme/data endpoint
* Add Power BI authentication in server/services/authService.ts

**2. Azure B2C Single Sign-On Integration**

**File**: server/services/authService.ts

**Location**: Lines 85-95 in authenticateWithAzureB2C() method

async authenticateWithAzureB2C(token: string): Promise<AuthUser | null> {

*// TODO: Implement actual Azure B2C token validation*

*// Use Microsoft Graph API and MSAL library*

*// Example B2C integration:*

*// const msalConfig = {*

*// auth: {*

*// clientId: process.env.AZURE\_CLIENT\_ID,*

*// authority: `https://${process.env.AZURE\_TENANT}.b2clogin.com/${process.env.AZURE\_TENANT}.onmicrosoft.com/${process.env.AZURE\_POLICY}`*

*// }*

*// };*

*// const cca = new ConfidentialClientApplication(msalConfig);*

*// const tokenValidation = await cca.acquireTokenOnBehalfOf({...});*

*}*

**Additional Integration Points**:

* client/src/components/auth/: Create Azure B2C login components
* server/index.ts lines 25-35: Add MSAL middleware configuration
* Update shared/schema.ts lines 15-25: Add Azure user ID mapping

**3. Row-Level Security Implementation**

**File**: server/storage.ts

**Location**: Lines 200-250 in data access methods

*// CURRENT DEMO IMPLEMENTATION - REPLACE FOR PRODUCTION*

async getPrograms(userRole: UserRole, userId?: number, institutionId?: number): Promise<Program[]> {

*// TODO: Implement database-level row-level security*

*// For PostgreSQL RLS:*

*// ALTER TABLE programs ENABLE ROW LEVEL SECURITY;*

*// CREATE POLICY program\_access ON programs FOR SELECT*

*// USING (*

*// CASE*

*// WHEN current\_setting('app.user\_role') = 'program\_director'*

*// THEN program\_director\_id = current\_setting('app.user\_id')::integer*

*// WHEN current\_setting('app.user\_role') = 'dio'*

*// THEN institution\_id = current\_setting('app.user\_institution\_id')::integer*

*// END*

*// );*

*}*

**Database Migration Required**:

* Create RLS policies in drizzle/migrations/ directory
* Update connection string in drizzle.config.ts
* Add user context setting in server/routes.ts middleware

**4. OpenAI API Integration**

**File**: server/services/aiService.ts

**Location**: Lines 15-25 in generateDashboard() method

async generateDashboard(request: DashboardRequest): Promise<DashboardResponse> {

*// PRODUCTION: Remove fallback logic and use only OpenAI*

*// Current implementation includes fallback for demo purposes*

try {

const openai = new OpenAI({ apiKey: process.env.OPENAI\_API\_KEY });

*// TODO: Enhance prompt engineering for medical education context*

const prompt = `You are a medical education data analyst...`;

const response = await openai.chat.completions.create({

model: "gpt-4o", // Latest model as of May 2024

messages: [{ role: "user", content: prompt }],

response\_format: { type: "json\_object" }

});

*// TODO: Add error handling for API limits and rate limiting*

*// TODO: Implement response validation and sanitization*

} catch (error) {

*// REMOVE: Fallback to local generation for production*

console.warn('OpenAI API unavailable, using local generation');

return this.generateLocalDashboard(request, analyticsData);

}

}

**Environment Variables Required**:

OPENAI\_API\_KEY=sk-your-actual-openai-key

OPENAI\_ORG\_ID=org-your-organization-id (optional)

**5. Additional API Integrations**

**ACGME Public Data APIs**

**File**: server/services/acgmeDataService.ts

**Lines**: 25-40 in dataSources array

Replace URLs with actual ACGME endpoints:

private dataSources: ACGMEDataSource[] = [

*// TODO: Replace with actual ACGME API endpoints*

{ category: "programs", url: "https://apps.acgme.org/ads/Public/Programs/Search", description: "Program directory and accreditation status" },

{ category: "residents", url: "https://apps.acgme.org/ads/Public/Reports/Report/1", description: "Resident and fellow data" },

*// Add authentication headers and API keys as required by ACGME*

];

**External Medical Education APIs**

**File**: server/services/predictiveAnalyticsService.ts

**Lines**: 50-75 in benchmarking methods

**Add integrations for:**

* AAMC (Association of American Medical Colleges) data
* NRMP (National Resident Matching Program) statistics
* Specialty board certification data

**E) Additional Technical Considerations**

**Security Implementation**

* **API Rate Limiting**: Implement rate limiting in server/routes.ts
* **Input Validation**: Zod schemas in shared/schema.ts provide request validation
* **CORS Configuration**: Update CORS settings in server/index.ts for production domains
* **SQL Injection Prevention**: Drizzle ORM provides parameterized queries

**Performance Optimization**

* **Data Caching**: Redis integration for scalable caching (currently in-memory)
* **Database Indexing**: Add indexes for frequently queried fields in schema
* **CDN Integration**: Serve static assets via CDN in production
* **Image Optimization**: Optimize chart exports and map images

**Monitoring and Logging**

* **Application Insights**: Azure integration for performance monitoring
* **Error Tracking**: Sentry or similar service integration points
* **Audit Logging**: Track user actions and data access for compliance

**Deployment Architecture**

* **Container Support**: Docker configuration available
* **Load Balancing**: Multi-instance deployment support
* **Database Scaling**: Connection pooling and read replicas
* **SSL/TLS**: HTTPS configuration for production domains