# **MedMatch Platform - Complete Project Documentation v2**

## **🏥 \*\*Project Overview\*\***

MedMatch is a comprehensive internship matching platform designed specifically for medical students and healthcare clinics. The platform facilitates the connection between medical students seeking internship opportunities and clinics offering positions, with a robust admin system for platform management.

## **🎯 \*\*Core Purpose\*\***

The platform solves the challenge of matching medical students with appropriate internship positions by providing:

* A centralized application system for students
* Position management tools for clinics
* Document handling for application materials
* Administrative oversight and management capabilities

## **👥 \*\*User Roles & Permissions\*\***

### **\*\*Student Role\*\***

* **Primary Function**: Apply for internship positions
* **Capabilities**:
  + Create and manage personal profile with academic information
  + Browse available internship positions
  + Submit applications with cover letters
  + Upload and manage documents (CV, certificates, etc.)
  + Track application status and history
  + Update pending applications
* **Data Access**: Own profile, own applications, own documents only

### **\*\*Clinic Role\*\***

* **Primary Function**: Post positions and manage applications
* **Capabilities**:
  + Create and manage clinic profile with facility information
  + Post internship positions with detailed requirements
  + Manage position status (active/closed)
  + Review applications from students
  + Update application status (pending/accepted/rejected)
  + Access documents from applicants
  + Add notes to applications during review
* **Data Access**: Own profile, own positions, applications to own positions only

### **\*\*Admin Role\*\***

* **Primary Function**: Platform oversight and management
* **Capabilities**:
  + Access both student and clinic functionalities for testing/support
  + Promote other users to admin status
  + Full platform oversight capabilities
  + System management and troubleshooting
* **Data Access**: Own data only on regular endpoints (GDPR compliant)
* **Special Access**: Future admin panel for full user management

## **🏗️ \*\*Technical Architecture\*\***

### **\*\*Frontend Stack\*\***

* **Framework**: React with TypeScript
* **Routing**: React Router with protected routes
* **Authentication**: Auth0 integration
* **State Management**: React hooks and context
* **Styling**: Modern CSS with responsive design
* **Security**: Role-based route protection, forbidden page handling (frontend/src/components/ForbiddenPage.tsx)

### **\*\*Backend Stack\*\***

* **Framework**: NestJS with TypeScript
* **Database**: PostgreSQL with TypeORM (backend/src/app.module.ts)
* **Authentication**: JWT tokens via Auth0 (backend/src/auth/jwt-auth.guard.ts)
* **Authorization**: Role-based access control (backend/src/auth/roles.guard.ts, backend/src/auth/roles.decorator.ts)
* **File Handling**: Multer for document uploads (backend/src/controllers/documents.controller.ts)
* **Security**: Global guards, role decorators, ownership verification

### **\*\*Database Design\*\***

* **Users Table**: Core user information with Auth0 integration (backend/src/entities/user.entity.ts)
* **Student Profiles**: Extended academic information for students (backend/src/entities/student-profile.entity.ts)
* **Clinic Profiles**: Extended facility information for clinics (backend/src/entities/clinic-profile.entity.ts)
* **Internship Positions**: Job postings with requirements and details (backend/src/entities/internship-position.entity.ts)
* **Applications**: Student applications to positions with status tracking (backend/src/entities/application.entity.ts)
* **Documents**: File uploads linked to students (backend/src/entities/document.entity.ts)

### **\*\*Entity Relationships\*\***

* **User → Student/Clinic Profile**: One-to-one relationship based on role
* **Clinic → Internship Positions**: One-to-many (clinics can post multiple positions)
* **Student → Applications**: One-to-many (students can apply to multiple positions)
* **Position → Applications**: One-to-many (positions can receive multiple applications)
* **Student → Documents**: One-to-many (students can upload multiple documents)
* **Application Status Flow**: PENDING → ACCEPTED/REJECTED (with timestamps)

## **🔐 \*\*Security Implementation\*\***

### **\*\*Authentication System\*\***

* **Auth0 Integration**: Complete OAuth2 flow with JWT tokens
* **Token Validation**: Backend validates all requests via JWT guard (backend/src/auth/jwt-auth.guard.ts)
* **User Creation**: Automatic user creation on first login (backend/src/controllers/users.controller.ts)
* **Role Assignment**: Immutable role selection during registration (backend/src/controllers/users.controller.ts)

### **\*\*Authorization System (RBAC)\*\***

* **Global Guards**: All controllers protected by JWT + Roles guards (backend/src/auth/roles.guard.ts)
* **Role Decorators**: @Roles() decorators on all endpoints (backend/src/auth/roles.decorator.ts)
* **Ownership Verification**: Users can only access their own data
* **Admin Privileges**: Admins inherit access to all role endpoints
* **Data Privacy**: &quot;Own data only&quot; approach for GDPR compliance

### **\*\*Security Features\*\***

* **Role Immutability**: Users cannot change roles after initial selection
* **Secure Admin Creation**: Only via environment secret or existing admin (backend/src/controllers/users.controller.ts)
* **File Upload Security**: Type validation, size limits, secure naming (backend/src/controllers/documents.controller.ts)
* **Access Control**: Document ownership and application-based access
* **Route Protection**: Frontend prevents unauthorized navigation

## **🛠️ \*\*API Structure\*\***

### **\*\*Controllers & Endpoints\*\***

* **Users Controller**: User management, role assignment, admin creation (backend/src/controllers/users.controller.ts)
* **Students Controller**: Student profiles, applications management (backend/src/controllers/students.controller.ts)
* **Clinics Controller**: Clinic profiles, position management (backend/src/controllers/clinics.controller.ts)
* **Positions Controller**: Internship position CRUD operations (backend/src/controllers/positions.controller.ts)
* **Applications Controller**: Application lifecycle management (backend/src/controllers/applications.controller.ts)
* **Documents Controller**: File upload, download, and management (backend/src/controllers/documents.controller.ts)

### **\*\*API Security Pattern\*\***

All endpoints follow this security pattern:

* **JWT Authentication**: Verify valid token
* **Role Authorization**: Check user has required role
* **Ownership Verification**: Ensure user can only access their own data
* **Data Validation**: Validate input parameters and body
* **Error Handling**: Proper HTTP status codes and error messages

### **\*\*Endpoint Access Matrix\*\***

## **🔄 \*\*Authentication Flow\*\***

### **\*\*Login Process\*\***

* **Frontend**: User clicks login, redirected to Auth0
* **Auth0**: User authenticates, returns to app with code
* **Token Exchange**: Frontend exchanges code for JWT token
* **Backend Sync**: API call creates/updates user record (backend/src/controllers/users.controller.ts)
* **Role Check**: Backend verifies user role and permissions (backend/src/auth/roles.guard.ts)
* **Session Start**: User gains access to role-appropriate features

### **\*\*Request Flow\*\***

* **Frontend**: Makes API request with JWT token
* **JWT Guard**: Validates token and extracts user info (backend/src/auth/jwt-auth.guard.ts)
* **Roles Guard**: Checks user role against endpoint requirements (backend/src/auth/roles.guard.ts)
* **Controller**: Processes request with verified user context
* **Data Filter**: Returns only data user is authorized to see

## **📁 \*\*File Management System\*\***

### **\*\*Document Handling\*\***

* **Upload Security**: File type validation, size limits, secure naming (backend/src/controllers/documents.controller.ts)
* **Storage**: Local filesystem with configurable upload directory (./uploads/)
* **Access Control**: Document ownership verification
* **Download Security**: Proper MIME type handling and access checks
* **Cleanup**: Automatic file removal on database operation failures

### **\*\*Document Types\*\***

* **CV**: Student curriculum vitae
* **CERTIFICATE**: Academic or professional certificates
* **OTHER**: Additional supporting documents

### **\*\*Access Rules\*\***

* **Students**: Can upload, view, and delete their own documents
* **Clinics**: Can view documents from students who applied to their positions
* **Admins**: Same rules as regular users (own documents only)

## **🚀 \*\*Development Setup\*\***

### **\*\*Project Structure\*\***

project-root/

├── frontend/ # React application

│ ├── src/

│ │ ├── components/

│ │ ├── pages/

│ │ └── App.js

│ └── .env

└── backend/ # NestJS API

├── src/

│ ├── controllers/

│ ├── entities/

│ ├── auth/

│ └── main.ts

└── .env

### **\*\*Key Configuration Files\*\***

* **Backend Main**: Application bootstrap and module configuration (backend/src/main.ts, backend/src/app.module.ts)
* **Database Config**: TypeORM connection and entity registration (backend/src/app.module.ts)
* **Auth Config**: JWT strategy and Auth0 integration
* **Guards Setup**: Global security guard registration (backend/src/app.module.ts)

### **\*\*Environment Variables\*\***

* **Frontend**: Auth0 domain, client ID, API audience
* **Backend**: Database connection, JWT secret, Auth0 configuration
* **Security**: Admin creation secret, file upload limits

## **🔧 \*\*Recent Security Fixes (Completed)\*\***

### **\*\*Role-Based Access Control Implementation\*\***

* **Frontend Route Protection**: Added ProtectedRoute component with role checking
* **Backend Authorization**: Implemented @Roles() decorators on all controllers
* **Global Security Guards**: JWT + Roles guards active on all endpoints (backend/src/app.module.ts)
* **Data Ownership**: Users can only access their own data
* **Admin System**: Proper admin inheritance with GDPR compliance

### **\*\*Security Vulnerabilities Removed\*\***

* **Insecure Role Switching**: Removed endpoints that allowed easy role changes (backend/src/controllers/users.controller.ts)
* **Public Data Access**: Removed endpoints that exposed user data without auth
* **Admin Key Exploits**: Replaced simple key systems with secure admin creation (backend/src/controllers/users.controller.ts)
* **URL Manipulation**: Frontend now prevents unauthorized route access
* **Data Leakage**: Backend ensures users only see their own data

### **\*\*Current Security Status\*\***

✅ **Authentication**: Complete Auth0 integration with JWT validation✅ **Authorization**: Full RBAC system with role decorators✅ **Data Protection**: Own-data-only access with ownership verification✅ **Admin Security**: Secure admin creation and privilege inheritance✅ **File Security**: Complete upload/download access control✅ **Route Protection**: Frontend and backend route security

## **📋 \*\*Known Issues & Future Plans\*\***

### **\*\*Current State\*\***

* **Backend**: Fully secured with comprehensive RBAC system
* **Frontend**: Basic role-based routing implemented
* **Database**: All entities and relationships properly defined
* **File System**: Secure document management operational

### **\*\*Future Enhancements\*\***

* **Admin Panel**: Dedicated admin interface for user management
* **Email Notifications**: Application status change notifications
* **Advanced Search**: Position filtering and search capabilities
* **Reporting**: Analytics dashboard for platform usage
* **Mobile App**: React Native mobile application

### **\*\*Testing Priorities\*\***

* **Security Testing**: Verify all role-based restrictions work correctly
* **File Upload Testing**: Test document upload/download security
* **Application Flow**: End-to-end application process testing
* **Admin Functions**: Test admin privilege inheritance
* **Error Handling**: Verify proper error responses and edge cases

## **🎯 \*\*Development Guidelines\*\***

### **\*\*Security First Approach\*\***

* Always implement authentication before authorization
* Use @Roles() decorators on all new endpoints
* Verify data ownership in all data access operations
* Log admin actions for audit trails
* Follow &quot;own data only&quot; principle for GDPR compliance

### **\*\*Code Organization\*\***

* Keep entities in separate files with proper relationships
* Use TypeScript interfaces for request/response types
* Implement proper error handling with meaningful messages
* Follow NestJS best practices for dependency injection
* Maintain consistent naming conventions across frontend/backend

This documentation provides complete context for continuing development of the MedMatch platform with full understanding of the current implementation, security measures, and architectural decisions.