# GitHub Copilot Agent Prompt: File Upload System with MongoDB and MinIO

## Project Overview

Update my existing project to add a complete file upload system with form submission, using MongoDB for metadata storage and MinIO for file storage.

## Requirements

### Core Functionality

- Create a user-friendly file upload interface
- Implement form submission with file handling
- Store file metadata in MongoDB
- Store actual files in MinIO object storage
- Add proper error handling and validation
- Include upload progress indicators
- Support multiple file types (images, documents, etc.)

### Technical Stack Integration

- **Database**: MongoDB for storing file metadata (filename, size, upload date, user info, etc.)
- **File Storage**: MinIO for actual file storage
- **Frontend**: Create responsive UI with file drag-and-drop functionality
- **Backend**: RESTful API endpoints for file operations

### Specific Implementation Tasks

#### Frontend Requirements:

1. **File Upload Page/Component**:
   - Drag and drop file upload area
   - File type validation on client side
   - Upload progress bar
   - Preview for uploaded files (especially images)
   - Form fields for additional metadata (title, description, tags, etc.)

- Submit button with loading states

2. **UI/UX Features**:
   - Responsive design for mobile and desktop
   - Error messages for validation failures
   - Success notifications
   - File size limits display
   - Supported file formats list

**Backend Requirements:**

1. **API Endpoints**:
   - `POST /api/upload` - Handle file upload and form submission
   - `GET /api/files` - List uploaded files with metadata
   - `GET /api/files/:id` - Get specific file metadata
   - `DELETE /api/files/:id` - Delete file from both MinIO and MongoDB
   - `GET /api/files/:id/download` - Download file from MinIO

2. **Database Schema** (MongoDB):
   - File metadata collection with fields:
     - `filename` (original name)
     - `mimetype`
     - `size`
     - `uploadDate`
     - `minioKey` (MinIO object key)
     - `userId` (if authentication exists)
     - `description`
     - `tags`
     - `status` (uploaded, processing, error)

3. **MinIO Integration**:
   - Configure MinIO client connection
   - Create bucket if not exists
   - Generate unique object keys
   - Handle file upload to MinIO

- Implement file deletion from MinIO

**Configuration & Setup:**

1. **Environment Variables**:
   - MinIO credentials and endpoint
   - MongoDB connection string
   - File upload limits
   - Allowed file types
2. **Dependencies to Add**:
   - MinIO client library
   - File upload middleware (multer or similar)
   - MongoDB driver/ODM
   - File validation libraries

**Security & Validation:**

- File type validation (server-side)
- File size limits
- Sanitize uploaded file names
- Virus scanning consideration
- Rate limiting for uploads
- Authentication integration (if applicable)

**Error Handling:**

- Network errors during upload
- MinIO connection failures
- MongoDB operation errors
- File validation failures
- Storage quota exceeded

## Implementation Guidelines

### Code Structure:

- Create modular, reusable components

- Follow existing project architecture patterns
- Add proper TypeScript types if applicable
- Include comprehensive error handling
- Add logging for debugging

### Testing:

- Unit tests for file validation
- Integration tests for upload flow
- Test error scenarios
- Test file deletion functionality

### Performance Considerations:

- Implement file chunking for large files
- Add compression for applicable file types
- Consider CDN integration for file serving
- Implement proper caching strategies

## Expected Deliverables

1. **Frontend Components**:
   - File upload page/component
   - File list/gallery component
   - File management interface
2. **Backend Services**:
   - File upload service
   - MongoDB service layer
   - MinIO service layer
   - API routes
3. **Configuration**:
   - Environment setup
   - Database migrations/setup
   - MinIO bucket configuration
4. **Documentation**:
   - API documentation

- Setup instructions
- Usage examples

## Additional Features (Optional):

- Image resizing/optimization
- File versioning
- Batch file upload
- File sharing capabilities
- Admin panel for file management
- File metadata search functionality

Please analyze my current project structure and implement this file upload system following the existing code patterns and architecture. Ensure all code is production-ready with proper error handling and security considerations.