The Challenge

You will build an **Enterprise Document Management System** that allows organizations to securely store, manage, and share documents with fine-grained access control.

Business Context

A growing company needs a secure system to manage their internal documents. The system should:

- Allow employees to upload and organize documents
- Control who can access which documents
- Track all user activities for audit purposes
- Scale to support hundreds of users
- Run on Windows Server with IIS

Core Requirements

1. Authentication & Authorization (Mocked for the Challenge)

Use a mocked authentication flow. No user registration, password hashing, refresh tokens, or external IdP integration are required.

Endpoints

- POST /auth/login
  Accepts { email, password }. If email matches one of the provided mock users, return a mock token and user payload.
- GET /auth/me (optional)
  Returns the user derived from the provided token.
- POST /auth/logout (optional)
  Client-side token disposal (no server state required).

User Roles & Permissions The system must support four role levels:

- **Viewer**: Can view public and shared documents
- **Contributor**: Can create, edit, and delete their own documents
- **Manager**: Can manage documents from their team, assign access
- **Admin**: Full system access, user management, audit logs

Mock Users

Provide preset users with roles.

Token format

- Return a mock token (either a signed JWT with a local secret or a simple base64 opaque token).
- The token payload must include: sub, email, role, and optionally exp.

2. Document Management

Document CRUD Operations

- Upload documents (support PDF, DOCX, TXT files)
- View list of all accessible documents
- View document details
- Download documents
- Edit document metadata (title, description, tags)
- Delete documents
- Document size limit: 10MB per file

Document Metadata Each document should store:

- Title (required)
- Description (optional)
- Tags (multiple tags allowed)
- Uploaded by (user)
- Created date
- Last modified date
- Access type (Public, Private, Restricted)
- File name and size

Document Access Control

- **Public**: All authenticated users can access
- **Private**: Only the document owner can access
- **Restricted**: Only specific users or roles can access (require sharing mechanism)

Sharing Functionality

- Document owner can share with specific users
- When sharing, specify permission level (Read or Write)
- Shared documents should appear in recipient's document list
- Owner can revoke sharing access

Search Functionality

- Users should be able to search documents by:
  - Title
  - Description
  - Tags
  - Content type
- Search should be case-insensitive
- Return results with relevance ranking

3. Database Design

Create a SQL Server database with proper schema design for:

Tables Required:

- Documents (document metadata)
- DocumentShares (tracks which users have access to specific documents)
- DocumentTags (tags associated with documents, many-to-many relationship)
- AuditLogs (tracks all user actions for compliance)

Database Requirements:

- Create appropriate indexes for performance optimization
- Implement foreign key constraints
- Add data validation at database level where appropriate
- Create a migration script
- Users are managed by the provided authentication service

4. API Design

Design and implement a RESTful API. The API should cover document CRUD, sharing, user management (admin), and auditing

API Requirements:

- Use ASP.NET
- Implement proper HTTP status codes (200, 201, 400, 401, 403, 404, 409, 500)
- Return JSON responses
- Implement request validation (validate all input)
- Apply rate limiting (100 requests per minute per user)
- Include API versioning (v1)
- Add comprehensive error handling with meaningful error messages
- Implement pagination for list endpoints (default page size: 20)
- Document API

5. Frontend Application

Build a web-based frontend using JavaScript (vanilla JS or a framework/library of your choice - eg React).

Pages Required:

1. **Login Page**
    - Email and password fields
    - Login button that calls the provided Auth Service endpoint
2. **Dashboard** (Main page after login)
    - Welcome message with user name
    - List of accessible documents in a table
    - Search bar
    - "Upload Document" button
    - Pagination controls
    - Display: Document title, owner, date, access type, actions (view, download, delete)
3. **Document Upload Page**

    - Form to upload document
    - Fields: Title, Description, File, Tags, Access Type
    - Validation feedback
    - Cancel and Submit buttons

Frontend Requirements:

- Use ES6+ JavaScript
- Handle authentication (store auth token in localStorage)
- Make authenticated API calls
- Display loading states during API calls
- Show success/error messages to users
- Handle token expiration and redirect to login
- Implement client-side validation
- Clean, professional UI

6. Deployment

IIS

Configure your application for Windows Server IIS deployment.

Requirements:

- Create `web.config` with proper settings for ASP.NET
- Configure static file serving
- Set up HTTPS configuration
- Configure connection strings for production
- Provide deployment instructions
- Handle reverse proxy configuration (if needed)

Azure

- In your README, describe how you would deploy it to Azure

7. Security Requirements

Implement security best practices:

API Security:

- Implement CORS policies
- Validate and sanitize all input
- Prevent SQL injection attacks
- Prevent XSS (Cross-Site Scripting) attacks
- Set proper HTTP security headers

File Upload Security:

- Validate file types (whitelist approach)
- Restrict file size
- Sanitize file names

8. Performance & Optimization

Optimize your application for performance:

Database:

- Implement proper indexing
- Use pagination for large datasets
- Optimize queries (avoid N+1 queries)
- Use async/await throughout

API:

- Implement response caching where appropriate
- Use async operations
- Stream large file downloads
- Optimize JSON serialization

Frontend:

- Minimize API calls
- Implement loading states
- Use efficient DOM manipulation

9. Logging & Monitoring

Implement comprehensive logging:

Application Logging:

- Log all API requests
- Log errors with stack traces
- Log performance metrics for critical operations

Audit Trail:

- Log all document operations (create, update, delete, download)
- Log all access attempts
- Store: User, action, timestamp, entity affected, IP address

10. Testing (Bonus - Architect Level)

Write tests for your application:

Unit Tests:

- Test business logic
- Test utility functions
- Test data validation
- Target: 70% code coverage minimum

Integration Tests:

- Test API endpoints
- Test document operations
- Test database operations

Technical Stack

You are required to use the following technologies:

- **ASP.NET**
- **SQL Server** (LocalDB or Express for local development)
- **JavaScript** (ES6+, any framework or vanilla JS)
- **IIS Deployment**

Quality & Architecture Expectations (non-functional)

- Follow **.NET naming and style conventions**; enforce via **.editorconfig**.
- Ensure **clear separation of concerns**.
- Follow **SOLID principles**.
- Use **DTOs** for requests/responses; avoid exposing persistence entities directly.
- Apply patterns **only when they add value**.
- Maintain **code quality**: meaningful naming, **DRY**, **KISS**, **YAGNI**, no commented or dead code.
- Centralize **cross-cutting concerns** (validation, error handling, logging) via **middleware, filters, or handlers**.

Tip: Reference this section in your README (Architecture Decisions)

Deliverables

Please provide the following in your submission:

1. **Complete Source Code**
   - Well-organized project structure - No need to run/execute
   - All source files including frontend
   - Database migration scripts
   - Configuration files
2. **Documentation**
   - README.md with:
     - Project description
     - Setup instructions
     - How to run the application
     - Database setup steps
     - API documentation or reference to Swagger

        - Auth Mock Notes: available endpoints, list of users/roles, token format (opaque or JWT) and how to switch between users/roles.
    - Database Schema Documentation:
        - ER diagram or table structure
        - Relationships between tables
        - Indexes created
    - Deployment Guide:
        - IIS deployment instructions
        - Describe how the system could be deployed to Azure (no deployment required)
        - Environment configuration

3. **Architecture Documentation** (Required for Architect Level)
    - System architecture diagram
    - Database schema diagram
    - API architecture overview
    - Security considerations document
    - Describe how your application uses the mocked Auth.
4. **Working Application**
    - Application should run locally
    - All features should work as specified
5. **Git Repository**
    - Commit history with meaningful messages
    - .gitignore file
    - Link to repository or ZIP file

Evaluation Criteria

Senior Developer (Focus Areas)

Code Quality (25%)

- Clean, readable, maintainable code
- Consistent coding style
- Proper naming conventions
- Error handling
- Code organization and structure

Functionality (30%)

- All core features implemented and working
- Document management CRUD operations
- Search and sharing functionality
- Basic frontend implementation

Database Design (15%)

- Proper schema design
- Relationships and constraints
- Indexes for performance

- Migration scripts

Security (15%)

- Input validation
- SQL injection prevention
- Proper use of Auth mocked token

Documentation (10%)

- README with setup instructions
- API documentation
- Code comments where appropriate

IIS/Azure Setup (5%)

- IIS deployment configuration
- Azure deployment explanation

Architect Level (Additional Focus Areas)

Architecture (20%)

- Proper separation of concerns
- Design patterns used appropriately
- Scalability considerations
- Microservices-friendly design (optional)

Performance (15%)

- Query optimization
- Caching implementation
- Efficient algorithms
- Load handling

Testing (10%)

- Unit tests with good coverage
- Integration tests
- Test quality and organization

Advanced Features (5%)

- Infrastructure as Code
- CI/CD pipeline
- Monitoring and observability

Getting Started

Local Development Setup

1. Install prerequisites
2. Create a new ASP.NET project
3. Create database models
4. Build API endpoints
5. Create frontend application
6. Test your implementation
7. Configure for IIS
8. Write documentation

Testing Credentials

Use these credentials for initial testing:

- **Admin**: admin@company.com / Admin@123
- This user is included in the mock users list

Important Notes

- **No External Libraries Beyond Required**: Use standard libraries and packages. Avoid heavy frameworks unless necessary.
- **Focus on Quality Over Quantity**: It's better to have fewer features implemented well than many features poorly implemented.
- **Ask Questions**: If anything is unclear, document your assumptions.
- **Time Management**: If you run out of time, prioritize core features over bonus features.
- **Security First**: Security vulnerabilities will significantly impact your evaluation.
- **Showcase Your Skills**: Include code comments, documentation, and architectural decisions that demonstrate your expertise.

What We're Looking For

We want to understand:

- Your problem-solving approach
- Your technical depth and breadth
- Your code quality standards
- Your architectural thinking
- Your ability to balance requirements with constraints
- Your professionalism and attention to detail

Submission

Please submit your solution via:

- GitHub repository (preferred) with link
- Or ZIP file containing all deliverables

Include in your submission:

- A brief summary of your approach
- Any assumptions you made
- What you would improve given more time
- Any challenges you encountered

Good luck! We look forward to reviewing your work.