Learning Management System (LMS) — Requirements Specification

Project Name: LMS (Assessment & Lesson Upload Platform)
Platform / Stack: .NET Core MVC (ASP.NET Core), C#
ORM: Entity Framework Core — Code-first (migrations)
Authentication: JWT bearer tokens with Refresh Token mechanism
Authorization: Role-Based Access Control (RBAC) implemented using Microsoft Identity (Identity Core)
Architecture Patterns: Controller → Service → Repository (CSR) with Generic Repository pattern
API Style: RESTful MVC controllers (JSON endpoints + optional server-rendered views)
File Storage: Local file system (dev) / Azure Blob Storage or AWS S3 (prod)

Stakeholders

- Product Owner / Client
- Admin Users
- Instructors (Content Creators)
- Students (Consumers)
- DevOps / Infrastructure Team
- QA / Testers

Goals & High-Level Features

1. Upload and manage **lessons** (documents, videos, attachments).
2. Upload and manage **assessments** (PDF, DOCX, quizzes as files + metadata).
3. RBAC with three roles: **Admin**, **Instructor**, **Student**.
4. Secure authentication using JWT access tokens and refresh tokens.
5. Maintain clean architecture (CSR pattern + Generic Repository).
6. File validation, size limits, optional virus scanning.
7. Unit and integration tests for all major flows.

Roles & Permissions (RBAC)

Role	Upload Lessons	Edit Lessons	Delete Lessons	Upload Assessments	Grade Assessments	Manage Users/Roles
Admin						
Instructor		(own)	(own)		(own course)	
Student				Submit only		

Functional Requirements

1. Authentication & Authorization

- Microsoft Identity for user & role management.
- **Login** → Access Token + Refresh Token.
- **Refresh Token** → New access token on expiry.
- **Logout** → Revoke refresh token.
- Admin can create Instructor & Student accounts.
- Claims-based authorization for granular control.

2. File Uploads & Content Management

- Upload **Lessons**: PDFs, DOCX, PPTX, MP4, images.
- Upload **Assessments**: Files + metadata (due date, evaluation type).
- Versioning support (optional).
- Download/stream access-controlled content.
- File size/type validation.

3. Course & Enrollment

- Admins/Instructors create courses & assign instructors.
- Students enroll in courses.

4. Assessment Lifecycle

- Create assessments.
- Students submit solutions.
- Instructors grade & give feedback.
- Notifications for submissions/grades.

5. Search & Filter

- Search by title, tag, instructor, course, date.
- Pagination & sorting.

Non-Functional Requirements

- **Security:** HTTPS only, hashed refresh tokens, CSRF protection.
- **Scalability:** Cloud storage for large files.
- **Performance:** Pagination, DB indexes.
- **Maintainability:** CSR pattern, SOLID, DI for services.
- **Observability:** Structured logging (Serilog), metrics.
- **Testing:** Unit + integration tests.

Data Model (Entities)

- **User** — IdentityUser extended (Id, DisplayName, etc.)
- **Role** — IdentityRole
- **Course** — Id, Title, Description, InstructorId
- **Lesson** — Id, CourseId, Title, Description, ContentType, FilePath/BlobUri, UploadedBy, UploadedAt, Version
- **Assessment** — Id, CourseId, Title, Description, FilePath/BlobUri, DueDate, CreatedBy
- **Submission** — Id, AssessmentId, StudentId, FilePath, SubmittedAt, Grade, Feedback
- **RefreshToken** — Id, UserId, TokenHash, ExpiresAt, RevokedAt, ReplacedByTokenId
- **AuditLog** — Id, UserId, Action, EntityType, EntityId, Timestamp, Metadata

---

Authentication Flow (JWT + Refresh Tokens)

1. **Login** → Validate credentials, issue:
   - Short-lived Access Token (JWT)
   - Long-lived Refresh Token (stored hashed in DB)
2. **Access Resource** → `Authorization: Bearer <token>`
3. **Token Expiry** → `/auth/refresh` endpoint
4. **Logout/Revoke** → Invalidate refresh token

Architecture & Layers

Solution Structure: LMS.Core → Entities, Interfaces LMS.Data → DbContext, EF migrations, Repositories LMS.Services → Business logic, DTOs LMS.Web → MVC Controllers, Startup configs LMS.Common → Shared utilities, mapping LMS.Tests → Unit & integration tests

Patterns Used: - Controller (thin) - Service (business logic) - Repository (data access) - Generic Repository for CRUD - Specific repositories for domain queries

---

Example API Endpoints

Auth

- `POST /api/auth/login`
- `POST /api/auth/refresh`
- `POST /api/auth/logout`

Users & Roles

- `POST /api/users` (Admin create)
- `GET /api/users` (Admin list)

Courses

- `GET /api/courses`
- `POST /api/courses` (Admin/Instructor)

Lessons

- `POST /api/courses/{id}/lessons`
- `GET /api/courses/{id}/lessons`

Assessments

- `POST /api/courses/{id}/assessments`
- `POST /api/assessments/{id}/submissions`
- `POST /api/assessments/{id}/grade`

Validation, Logging & Error Handling

- FluentValidation or DataAnnotations for inputs.
- Centralized exception handling middleware.
- Serilog structured logging.

Testing & Acceptance Criteria

- Unit tests for services/repositories.
- Integration tests for auth, file upload, grading.
- Acceptance:

- File upload works with validation.
- RBAC enforced for all endpoints.

Deployment & DevOps

- Deploy to Azure App Service / AWS ECS.
- Use Azure Key Vault / AWS Secrets Manager.
- CI/CD with automated testing.

Implementation Checklist

- [ ] Scaffold solution projects
- [ ] Configure Identity & seed roles
- [ ] Implement EF Core DbContext & migrations
- [ ] Create GenericRepository & specific repositories
- [ ] Implement services with DI
- [ ] Configure JWT + RefreshToken service
- [ ] Build controllers & endpoints
- [ ] Add file storage adapter
- [ ] Add Swagger docs
- [ ] Write tests
- [ ] Configure CI/CD
