**Project Title:** ProfileBook

**Name:** Mohammad Oves Sheikh

**Date:** 08 Sept 2025

**Batch Name:** WIPRO NGA - .Net Full Stack Angular - FY26

**Instructor Name:** Jyoti Patil Mam

# Table of Contents Pg No.

1. Problem Definition and Objectives ------ 3

2. Frontend & Backend Architecture ------ 4

3. Component Breakdown & API Design ------ 5

4. Database Design & Storage Optimization ------ 8

5. Implementation Details ------ 12

6. Testing Strategy ------ 13

7. Deployment & Configuration ------ 14

8. Conclusion ------ 15

**1. Problem Definition and Objectives**

ProfileBook is an online social media platform that connects people virtually. Users can post content, message other users, and report users. Admins manage users, approve posts, and handle reported users.

**Project Goals and Objectives**

- **Primary Goal:** Develop a full-stack social media platform with comprehensive admin management

- **User Management:** Implement secure user registration, authentication, and profile management

- **Content Management:** Create post creation, approval workflow, and content moderation system

- **Administrative Control:** Build admin dashboard for user management, content approval, and reporting

**- Group Functionality:** Enable users to create and manage groups with messaging capabilities

**2. Frontend & Backend Architecture**

**Technology Stack Overview**

**Frontend Architecture**

**- Framework: Angular 17 with Standalone Components**

**- UI Framework: Bootstrap 5.3 with custom SCSS styling**

**- State Management: RxJS with BehaviorSubject pattern**

**- Routing: Angular Router with lazy loading and route guards**

**- Authentication: JWT-based token management with localStorage**

**- HTTP Client: Angular HttpClient with interceptors**

**Backend Architecture**

**- Framework: ASP.NET Core 8.0 Web API**

**- Database: SQL Server with Entity Framework Core**

**- Authentication: ASP.NET Core Identity with JWT Bearer tokens**

**- Architecture Pattern: Service-Repository pattern with dependency injection**

**- API Documentation: Swagger/OpenAPI integration**

**- Security: CORS policy, JWT validation, role-based authorization**

**3. Component Breakdown & API Design**

**Frontend Component Architecture**

**Core Components (8 Main Components)**

1. **Authentication Components**

   - `LoginComponent` - User login with JWT token handling

   - `RegisterComponent` - User registration with validation

2. **User Components**

   - `FeedComponent` - Main social media feed with posts

   - `ProfileComponent` - User profile management

   - `MessagesComponent` - Direct messaging system

   - `SearchComponent` - User and content search functionality

   - `MyPostsComponent` - Personal post management

   - `NotificationsComponent` - User notification center

3. **Admin Components**

   - `AdminDashboardComponent` - Administrative overview

   - `PostApprovalComponent` - Content moderation

   - `UserManagementComponent` - User administration

   - `ReportManagementComponent` - Report handling

   - `GroupManagementComponent` - Group administration

**Service Layer (8 Services)**

- `AuthService` - Authentication and user session management

- `PostService` - Post CRUD operations and interactions

- `MessageService` - Direct messaging functionality

- `UserService` - User profile and management operations

- `AdminService` - Administrative operations

- `NotificationService` - User notification management

- `ReportService` - Report creation and management

- `UserGroupsService` - Group functionality and messaging

**API Design & Endpoints**

**Authentication Endpoints**

POST /api/auth/register - User registration

POST /api/auth/login - User authentication

POST /api/auth/logout - User logout

**User Management Endpoints**

GET /api/users - Get all users (Admin only)

GET /api/users/{id} - Get user by ID

PUT /api/users/{id} - Update user profile

DELETE /api/users/{id} - Delete user (Admin only)

GET /api/users/search - Search users

**Post Management Endpoints**

GET /api/posts - Get all posts

POST /api/posts - Create new post

PUT /api/posts/{id} - Update post

DELETE /api/posts/{id} - Delete post

POST /api/posts/{id}/like - Like/unlike post

GET /api/posts/{id}/comments - Get post comments

POST /api/posts/{id}/comments - Add comment

**Admin Endpoints**

GET /api/admin/posts/pending - Get pending posts for approval

PUT /api/admin/posts/{id}/approve - Approve post

GET /api/admin/reports - Get all reports

PUT /api/admin/reports/{id} - Update report status

**4. Database Design & Storage Optimization**

**Entity-Relationship Diagram (ERD)**

**Core Entities**

1. **AspNetUsers** (Identity Framework)

   - Primary Key: Id (nvarchar(450))

   - Extended with: FirstName, LastName, Bio, ProfilePicture, CreatedAt, UpdatedAt, IsActive

2. **Posts**

   - Primary Key: PostId (int, Identity)

   - Foreign Keys: UserId, ApprovedByUserId

   - Fields: Content, ImageUrl, Status, CreatedAt, UpdatedAt

3. **Groups**

   - Primary Key: GroupId (int, Identity)

   - Foreign Key: CreatedByUserId

   - Fields: GroupName, Description, CreatedAt, UpdatedAt, IsActive

4. **UserGroups** (Many-to-Many)

   - Primary Key: UserGroupId (int, Identity)

   - Foreign Keys: UserId, GroupId

   - Composite Unique Index: (UserId, GroupId)

5. **Messages**

   - Primary Key: MessageId (int, Identity)

   - Foreign Keys: SenderId, ReceiverId

   - Fields: MessageContent, TimeStamp, IsRead

6. **Notifications**

   - Primary Key: NotificationId (int, Identity)

   - Foreign Keys: UserId, PostId (nullable)

   - Fields: Title, Message, Type, IsRead, CreatedAt

**Database Relationships**

- **One-to-Many:** User → Posts, User → Messages (Sent/Received), User → Groups (Created)

- **Many-to-Many:** Users ↔ Groups (via UserGroups)

- **Self-Referencing:** Posts → Comments, Users → Followers

**Storage Optimization Techniques**

1. **Indexing Strategy**

   - Clustered indexes on primary keys

   - Non-clustered indexes on foreign keys

   - Composite indexes for frequently queried combinations

   - Unique indexes for business constraints

2. **Query Optimization**

   - Entity Framework query optimization with Include() for eager loading

   - Pagination implementation for large datasets

   - Stored procedures for complex queries

   - Database connection pooling

3. **Performance Features**

   - Default value constraints with GETUTCDATE()

   - Cascade delete rules for data integrity

   - Proper normalization to 3NF

   - Connection string optimization with MultipleActiveResultSets

**5. Implementation Details**

**Authentication & Security**

- **JWT Implementation:** 24-hour token expiration with HMAC SHA256 signing

- **Role-Based Authorization:** Admin and User roles with route guards

- **Password Security:** ASP.NET Core Identity with configurable password policies

- **CORS Configuration:** Specific origin allowance for frontend-backend communication

**Key Features Implemented**

1. **User Registration & Login** - Complete authentication flow

2. **Admin CRUD Operations** - Full administrative control panel

3. **Post Management** - Create, approve, edit, delete posts with image support

4. **Group Functionality** - Create groups, manage members, group messaging

5. **Real-time Notifications** - Event-driven notification system

6. **Responsive Design** - Bootstrap-based responsive UI with custom styling

**6. Testing Strategy**

**Backend Testing (NUnit Framework)**

- **Unit Tests:** 39 comprehensive tests covering all controllers and services

- **Test Coverage:** AuthController, AdminController, PostsController, UsersController, GroupsController

- **Service Testing:** TokenService validation and JWT generation testing

- **Database Testing:** In-memory Entity Framework for isolated testing

- **Mock Implementation:** Moq framework for dependency mocking

**### Testing Infrastructure**

- **Framework:** NUnit 3.14.0 with modern Assert.That() syntax

- **Mocking:** Moq 4.20.69 for UserManager and service dependencies

- **Database:** Entity Framework InMemory for unit test isolation

- **Coverage:** Authentication, authorization, CRUD operations, error handling

**7. Deployment & Configuration**

**Configuration Files**

- **appsettings.json:** Database connection, JWT configuration, logging settings

- **Program.cs:** Service registration, middleware configuration, CORS setup

- **Environment Configuration:** Development and production environment separation

**GitHub Deployment**

- **Repository Structure:** Organized solution with API, Frontend, and Tests projects

- **Version Control:** Git with proper .gitignore for sensitive files

- **Documentation:** Comprehensive README with setup instructions

**8. Conclusion**

**Project Achievements**

ProfileBook successfully delivers a comprehensive social media platform meeting all evaluation criteria:

**Database & Schema** - Complete relational database with proper relationships

**Admin Login CRUD** - Full administrative control panel

**User Registration & Login** - Secure authentication system

**Angular Frontend Template** - Professional responsive design

**Code Sanitization -** Clean, well-structured codebase

**CRUD Functionalities** - Complete data operations

**Responsiveness & Validation** - Bootstrap responsive design with validation

**Session Handling** - JWT-based session management

**Frontend-Backend Integration** - Seamless API integration

**Testing** - Comprehensive NUnit test suite