Project Overview

The **Blog System with Role-based Authentication** is a web application that allows users to create, manage, and comment on blog posts. It supports multiple user roles, including **admin**, **editor**, and **reader**, with different permissions for managing posts and interacting with content. This system uses **ASP.NET Core MVC**, **ASP.NET Core Web API**, **Entity Framework Core**, **ASP.NET Core Identity**, and **Role-based Authorization** to implement user authentication and management.

The application provides a simple, yet extensible blog platform where users can create articles, add comments, and view posts while respecting different access levels based on user roles.

Key Features

1. **User Authentication and Role Management**:
    - Users can register, log in, and manage their profiles using **ASP.NET Core Identity**.
    - The system supports **role-based authentication**:
        - **Admins**: Can create, edit, and delete blog posts.
        - **Editors**: Can create and edit blog posts but cannot delete them.
        - **Readers**: Can only read posts and comment on them.
2. **Blog Post Management**:
    - **Admins** and **Editors** can create, edit, and delete blog posts.
    - Blog posts can include a title, content, categories, and tags.
    - **Readers** can view blog posts and leave comments but cannot modify them.
3. **Commenting System**:
    - Users (readers, admins, and editors) can leave comments on blog posts.
    - Admins can moderate and delete inappropriate comments.
    - Comments are linked to specific blog posts.
4. **Role-based Authorization**:
    - Different user roles (admin, editor, reader) have varying levels of access to different parts of the application.
    - The application implements role-based access control (RBAC) to secure endpoints and views.
5. **API Endpoints for Blog Management**:
    - The system exposes RESTful APIs to manage blog posts and comments.

- The API allows for operations such as creating, retrieving, updating, and deleting posts and comments.

6. **Search and Filtering**:
    - Users can search for posts by keywords, tags, or categories.
    - Admins and editors can filter posts by status (published, draft, or archived).

Technologies Used

- **ASP.NET Core MVC**: For building the user interface of the blog system.
- **ASP.NET Core Web API**: To handle blog-related backend services (e.g., post management, comments, authentication).
- **Entity Framework Core**: For database interaction and managing blog data (posts, comments, users).
- **ASP.NET Core Identity**: For managing user registration, authentication, and role-based access control.
- **Role-based Authorization**: Implemented to differentiate between admin, editor, and reader roles for different levels of access.

System Architecture

1. **Frontend (ASP.NET Core MVC)**:
    - The MVC architecture handles user interactions, including blog post creation, editing, and viewing, as well as commenting on posts.
    - The front-end renders pages for blog posts, comments, and role-specific views (e.g., admin dashboard).
2. **Backend (ASP.NET Core Web API)**:
    - The API manages CRUD operations for blog posts and comments, exposing endpoints for interaction from the front-end.
    - The API also handles user authentication and authorization.
3. **Database (Entity Framework Core)**:
    - EF Core manages interactions with the database, handling data for blog posts, comments, and users.
4. **Role-based Authentication (ASP.NET Core Identity)**:

- ASP.NET Identity is used to register users, authenticate them, and assign them roles (admin, editor, reader).
- Authorization logic is implemented to control access based on the user's role.

User Stories

1. User Registration and Login

- **As a new user**, I want to register for an account so I can interact with the blog system.
- **As a registered user**, I want to log in securely using my credentials.
- **As an admin**, I want to manage user roles and assign users as admin, editor, or reader.

2. Blog Post Management

- **As an admin**, I want to create, edit, and delete blog posts to manage content.
- **As an editor**, I want to create and edit blog posts but cannot delete them.
- **As a reader**, I want to read blog posts but cannot modify or delete them.
- **As a reader**, I want to see the author's name, tags, and categories for each blog post.

3. Commenting on Blog Posts

- **As a reader**, I want to comment on blog posts, adding my thoughts or feedback.
- **As an admin**, I want to moderate comments and delete inappropriate ones.
- **As an admin or editor**, I want to respond to user comments to engage with the audience.

4. Search and Filtering

- **As a user**, I want to search for blog posts by title, tag, or category.
- **As an admin**, I want to filter posts based on their status (draft, published, archived).

Database Design

Entities:

1. **User**:
    - Id (Primary Key)
    - Username
    - Email
    - PasswordHash

  - Role (Admin, Editor, Reader)

2. **BlogPost**:
   - Id (Primary Key)
   - Title
   - Content
   - AuthorId (Foreign Key referencing User)
   - CreatedAt
   - UpdatedAt
   - Status (Published, Draft, Archived)
   - Tags (Many-to-Many relationship with Tags)
   - CategoryId (Foreign Key referencing Category)
3. **Category**:
   - Id (Primary Key)
   - Name
4. **Tag**:
   - Id (Primary Key)
   - Name
5. **Comment**:
   - Id (Primary Key)
   - Content
   - CreatedAt
   - PostId (Foreign Key referencing BlogPost)
   - AuthorId (Foreign Key referencing User)

APIs and Endpoints

- **POST** /api/auth/register: Register a new user.
- **POST** /api/auth/login: Log in a user.
- **GET** /api/posts: Retrieve all blog posts (with optional filters for categories and status).
- **GET** /api/posts/{id}: Retrieve a specific blog post by ID.

- **POST** /api/posts: Create a new blog post (for admin and editor).
- **PUT** /api/posts/{id}: Update an existing blog post (for admin and editor).
- **DELETE** /api/posts/{id}: Delete a blog post (only for admin).
- **POST** /api/comments: Create a comment on a blog post.
- **GET** /api/comments/{postId}: Retrieve all comments for a specific post.

Role-based Authorization

- **Admin Role**: Full control over the system (can create, edit, delete posts, manage users, and moderate comments).
- **Editor Role**: Can create and edit posts but cannot delete them.
- **Reader Role**: Can view posts, comment, and read other users' comments, but cannot modify posts.

Authorization in Controller:

- **[Authorize(Roles = "Admin")]**: Used to restrict access to admin-specific functionality (e.g., managing users).
- **[Authorize(Roles = "Admin, Editor")]**: Used to allow both admin and editor roles to create and manage posts.