**ASP.NET Core 8 Web API +MongoDB.EntityFrameworkCore.**

**Cookie-based authentication + role-based authorization** -

1. **Install packages needed :**

Microsoft.AspNetCore.Authentication.Cookies ( for cookie based authentication)

BCrypt.Net-Next ( password hashing/verification )

1. **Configure cookie auth + roles in Program.cs :**

// ✅ Cookie Authentication

builder.Services

.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie(options =>

{

options.Cookie.Name = "StudyNotion.Auth";

options.Cookie.HttpOnly = true;

options.Cookie.SecurePolicy = CookieSecurePolicy.Always;

options.Cookie.SameSite = SameSiteMode.Lax;

options.SlidingExpiration = true;

options.ExpireTimeSpan = TimeSpan.FromDays(14);

// API-friendly: return status codes instead of redirecting to HTML pages

options.Events = new CookieAuthenticationEvents

{

OnRedirectToLogin = ctx => { ctx.Response.StatusCode = StatusCodes.Status401Unauthorized; return Task.CompletedTask; },

OnRedirectToAccessDenied = ctx => { ctx.Response.StatusCode = StatusCodes.Status403Forbidden; return Task.CompletedTask; }

};

});

// ✅ Authorization (policies optional; roles can be used via attributes)

builder.Services.AddAuthorization(options =>

{

options.AddPolicy("AdminOnly", p => p.RequireRole("Admin"));

options.AddPolicy("InstructorOnly", p => p.RequireRole("Instructor"));

});

1. **Create Auth DTOs (never accept/return the entity directly)**

using System.ComponentModel.DataAnnotations;

using StudyNotionServer.Data;

namespace StudyNotionServer.ServiceLayer.Models

{

public class RegisterUserRequest

{

[Required(ErrorMessage ="Firstname must be mentioned")]

public string FirstName { get; set; }

[Required(ErrorMessage = "Lastname must be mentioned")]

public string LastName { get; set; }

[Required(ErrorMessage = "Email must be mentioned")]

public string Email { get; set; }

[Required(ErrorMessage = "Password must be mentioned")]

public string Password { get; set; }

[Required(ErrorMessage = "Confirm-Password must be mentioned")]

public string ConfirmPassword { get; set; }

public string Image { get; set; }

public string? ProfileId { get; set; } // optional during registration

[Required(ErrorMessage = "AccountType must be mentioned")]

public string AccountType { get; set; } // "Admin" | "Student" | "Instructor"

}

public class RegisterUserResponse

{

public bool Success { get; set; }

public string Message { get; set; }

public User? RegisteredUser { get; set; }

}

public class LoginUserRequest

{

[Required]

[EmailAddress]

[RegularExpression(@"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$")]

public string Email { get; set; }

[Required]

[RegularExpression(@"^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[!@#$%^&\*()\_+])[A-Za-z\d!@#$%^&\*()\_+]{8,}$")]

public string Password { get; set; }

}

public class LoginUserResponse

{

public bool Success { get; set; }

public string Message { get; set; }

public User? user { get; set; }

}

}

1. **Implement AuthController (register, login, logout, me)**

using System.Net;

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.AspNetCore.Authentication;

using System.Security.Claims;

using Microsoft.AspNetCore.Mvc;

using StudyNotionServer.Data;

using StudyNotionServer.ServiceLayer;

using StudyNotionServer.ServiceLayer.Models;

using Microsoft.AspNetCore.Authorization;

namespace StudyNotionServer.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class AuthController : ControllerBase

{

private readonly IStudyNotionS \_serviceLayer;

private readonly ILogger<AuthController> \_logger;

public AuthController(IStudyNotionS serviceLayer,ILogger<AuthController> logger) {

\_serviceLayer = serviceLayer;

\_logger = logger;

}

//POST: /api/auth/register

[HttpPost("register")]

public async Task<IActionResult> Register([FromBody] RegisterUserRequest request)

{

try

{

RegisterUserResponse response = await \_serviceLayer.RegisterUser(request);

if (response != null)

{

if (response.Success)

{

return Ok(new{

success = response.Success,

message = "successfully registered the user",

data = response.RegisteredUser

});

}

else

{

if(! response.Success && string.Equals(response.Message.ToLower(), "invalid account type"))

{

return BadRequest(new

{

success = false,

message = "invalid account type"

});

}

else

{

return Ok(new

{

success = response.Success,

message = response.Message

});

}

}

}

else

{

return StatusCode(StatusCodes.Status500InternalServerError,new

{

success = false,

message = "Internal server error"

});

}

}

catch (Exception ex)

{

\_logger.LogError($"Exception occured in Register method of AuthController. Exception - {ex.ToString()}");

return StatusCode(StatusCodes.Status500InternalServerError, new

{

success = false,

message = "exception occured"

});

}

}

//POST: /api/auth/login

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] LoginUserRequest request)

{

try

{

LoginUserResponse response = await \_serviceLayer.LoginUser(request);

if (response != null)

{

if (response.Success && response.user != null)

{

// Build claims

var claims = new List<Claim>

{

new Claim(ClaimTypes.NameIdentifier,response.user.Id),

new Claim(ClaimTypes.Name, $"{response.user.FirstName} {response.user.LastName}"),

new Claim(ClaimTypes.Email, response.user.Email),

new Claim(ClaimTypes.Role, response.user.AccountType.ToString())

};

var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);

var principal = new ClaimsPrincipal(identity);

var props = new AuthenticationProperties

{

IsPersistent = true,

ExpiresUtc = DateTimeOffset.UtcNow.AddDays(14)

};

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, principal, props);

return Ok(new

{

success = response.Success,

message = "login successful",

});

}

else

{

return Ok(new

{

success = response.Success,

message = response.Message

});

}

}

else

{

return StatusCode(StatusCodes.Status500InternalServerError, new

{

success = false,

message = "error occured"

});

}

}

catch (Exception ex)

{

\_logger.LogError($"Exception occured in Login method of AuthController. Exception - {ex.ToString()}");

return StatusCode(StatusCodes.Status500InternalServerError, new

{

success = false,

message = "exception occured"

});

}

}

[Authorize]

[HttpPost("logout")]

public async Task<IActionResult> Logout()

{

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

return Ok(new {success = true, message = "logout successful" });

}

}

}

1. **Protect your other controllers/actions**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

[Route("api/[controller]")]

public class AdminController : ControllerBase

{

[Authorize(Roles = "Admin")]

[HttpGet("dashboard")]

public IActionResult Dashboard() => Ok(new { message = "Admin dashboard" });

}

[Authorize(Policy = "InstructorOnly")]

[HttpPost("create-course")]

public IActionResult CreateCourse() => Ok();

**Create Collections in the database :   
  
Note -**

In SQL EF Core, migrations (Add-Migration, Update-Database) generate schema and tables for you.

But in MongoDB, there is no schema migration — Mongo is schemaless.

EF Core Mongo provider (MongoDB.EntityFrameworkCore) does not auto-create empty collections at app start.

MongoDB will only create a collection when you insert the first document into it.

So your DbSets exist, but until you actually call AddAsync() + SaveChangesAsync(), nothing appears in Atlas.  
 **Step 1: Ensure your DbContext has all models**

using Microsoft.EntityFrameworkCore;

namespace StudyNotionServer.Data

{

public class StudyNotionDbContext : DbContext

{

public DbSet<User> Users { get; init; }

public DbSet<Profile> Profiles { get; init; }

public DbSet<Course> Courses { get; init; }

public DbSet<CourseProgress> CourseProgresses { get; init; }

public DbSet<Section> Section { get; init; }

public DbSet<SubSection> SubSection { get; init; }

public DbSet<RatingAndReview> RatingAndReview { get; init; }

public DbSet<Tag> Tag { get; init; }

public DbSet<OTP> OTP { get; init; }

public StudyNotionDbContext(DbContextOptions<StudyNotionDbContext> options) : base(options)

{

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<User>();

modelBuilder.Entity<Profile>();

modelBuilder.Entity<Course>();

modelBuilder.Entity<CourseProgress>();

modelBuilder.Entity<Section>();

modelBuilder.Entity<SubSection>();

modelBuilder.Entity<RatingAndReview>();

modelBuilder.Entity<Tag>();

modelBuilder.Entity<OTP>();

}

}

}

**Step 2: Create a DataSeeder service**

This class will insert some initial data into MongoDB.

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using StudyNotionServer.Data;

namespace StudyNotionServer.Seed

{

public static class DbSeeder

{

public static async Task SeedAsync(IServiceProvider serviceProvider)

{

using var scope = serviceProvider.CreateScope();

var dbContext = scope.ServiceProvider.GetRequiredService<StudyNotionDbContext>();

// Ensure Mongo collections are created

await dbContext.Database.EnsureCreatedAsync();

// --------------------------

// 1. Profiles

// --------------------------

if (!await dbContext.Profiles.AnyAsync())

{

var profile1 = new Profile

{

Gender = "Male",

DateOfBirth = "1995-01-01",

About = "Passionate learner",

phoneNumber = 987654321

};

var profile2 = new Profile

{

Gender = "Female",

DateOfBirth = "1990-05-05",

About = "Expert instructor",

phoneNumber = 123456789

};

await dbContext.Profiles.AddRangeAsync(profile1, profile2);

await dbContext.SaveChangesAsync();

}

var profiles = await dbContext.Profiles.ToListAsync();

// --------------------------

// 2. Users

// --------------------------

if (!await dbContext.Users.AnyAsync())

{

var student = new User

{

FirstName = "John",

LastName = "Doe",

Email = "student@studynotion.com",

PasswordHash = "hashedpassword",

AccountType = accountType.Student,

Image = "https://placehold.co/100x100",

ProfileId = profiles[0].Id

};

var instructor = new User

{

FirstName = "Jane",

LastName = "Smith",

Email = "instructor@studynotion.com",

PasswordHash = "hashedpassword",

AccountType = accountType.Instructor,

Image = "https://placehold.co/100x100",

ProfileId = profiles[1].Id

};

await dbContext.Users.AddRangeAsync(student, instructor);

await dbContext.SaveChangesAsync();

}

var users = await dbContext.Users.ToListAsync();

var instructorUser = users.First(u => u.AccountType == accountType.Instructor);

var studentUser = users.First(u => u.AccountType == accountType.Student);

// --------------------------

// 3. Tags

// --------------------------

if (!await dbContext.Tag.AnyAsync())

{

var tag = new Tag

{

Name = "Web Development",

Description = "Learn to build modern web apps"

};

await dbContext.Tag.AddAsync(tag);

await dbContext.SaveChangesAsync();

}

var tagDb = await dbContext.Tag.FirstAsync();

// --------------------------

// 4. Sections + SubSections

// --------------------------

if (!await dbContext.Section.AnyAsync())

{

var subsection1 = new SubSection

{

Title = "Intro to ASP.NET Core",

timeDuration = "10 min",

Description = "Basic overview of ASP.NET Core",

VideoUrl = "https://video.com/intro"

};

var subsection2 = new SubSection

{

Title = "Setting up environment",

timeDuration = "15 min",

Description = "Install .NET SDK and tools",

VideoUrl = "https://video.com/setup"

};

var section = new Section

{

SectionName = "Getting Started",

SubSections = new List<SubSection> { subsection1, subsection2 }

};

await dbContext.Section.AddAsync(section);

await dbContext.SaveChangesAsync();

}

var sectionDb = await dbContext.Section.Include(s => s.SubSections).FirstAsync();

// --------------------------

// 5. Course

// --------------------------

if (!await dbContext.Courses.AnyAsync())

{

var course = new Course

{

Title = "ASP.NET Core Basics",

Description = "Learn how to build APIs with ASP.NET Core.",

WhatYouWillLearn = "Controllers, Routing, Dependency Injection",

Price = 49,

Thumbnail = "https://placehold.co/600x400",

InstructorId = instructorUser.Id,

SectionId = sectionDb.Id,

TagId = tagDb.Id,

EnrolledStudentsIds = new List<string> { studentUser.Id }

};

await dbContext.Courses.AddAsync(course);

await dbContext.SaveChangesAsync();

// Assign Course to tag

tagDb.CourseId = course.Id;

await dbContext.SaveChangesAsync();

}

var courseDb = await dbContext.Courses.FirstAsync();

// --------------------------

// 6. Course Progress

// --------------------------

if (!await dbContext.CourseProgresses.AnyAsync())

{

var progress = new CourseProgress

{

CourseId = courseDb.Id,

CompletedVideosIds = new List<string> { sectionDb.SubSections.First().Id },

SubSections = sectionDb.SubSections

};

await dbContext.CourseProgresses.AddAsync(progress);

await dbContext.SaveChangesAsync();

}

// --------------------------

// 7. Rating & Review

// --------------------------

if (!await dbContext.RatingAndReview.AnyAsync())

{

var review = new RatingAndReview

{

Rating = 5,

Review = "Great course, highly recommended!",

UserId = studentUser.Id

};

await dbContext.RatingAndReview.AddAsync(review);

await dbContext.SaveChangesAsync();

}

// --------------------------

// 8. OTP

// --------------------------

if (!await dbContext.OTP.AnyAsync())

{

var otp = new OTP

{

Email = "student@studynotion.com",

otp = "123456",

CreatedAt = DateTime.UtcNow

};

await dbContext.OTP.AddAsync(otp);

await dbContext.SaveChangesAsync();

}

}

}

}

**Step 3: Call the “SeedAsync” method of “DbSeeder” class from controller**

using Microsoft.AspNetCore.Mvc;

using StudyNotionServer.Seed;

namespace StudyNotionServer.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class SeederController : ControllerBase

{

private readonly IServiceProvider \_serviceProvider;

public SeederController(IServiceProvider serviceProvider)

{

\_serviceProvider = serviceProvider;

}

[HttpPost("run")]

public async Task<IActionResult> RunSeeder()

{

try

{

await DbSeeder.SeedAsync(\_serviceProvider);

return Ok(new { message = "Seeding completed successfully!" });

}

catch (Exception ex)

{

return StatusCode(500, new { message = "Seeding failed", error = ex.Message });

}

}

}

}

This will create and insert Profiles, Users, Tags, Sections, SubSections, Courses, CourseProgress, RatingAndReview, OTP into MongoDB.  
Then you can check them in MongoDB Atlas → Browse Collections.