در اینجا یک \*\*پروژه کامل Authentication با JWT در ASP.NET Core\*\* را به صورت کاملاً ساختارمند و قابل اجرا برای شما آماده کرده‌ام. این پروژه شامل تمام بخش‌های لازم است:

---

## 📁 ساختار کلی پروژه

```

/AuthSystem

│

├── Models/

│ └── User.cs

│

├── DTOs/

│ ├── RegisterRequest.cs

│ └── LoginRequest.cs

│

├── Services/

│ ├── IUserService.cs

│ ├── UserService.cs

│ ├── IJwtService.cs

│ └── JwtService.cs

│

├── Controllers/

│ └── AuthController.cs

│

├── Data/

│ └── AppDbContext.cs

│

├── Program.cs

├── appsettings.json

└── .csproj

```

---

## ✅ 1. Models/User.cs

```csharp

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

public class User

{

public Guid Id { get; set; } = Guid.NewGuid();

[Required]

[StringLength(100)]

public string Username { get; set; }

public string PasswordHash { get; set; }

public UserRole Role { get; set; }

[EmailAddress]

public string Email { get; set; }

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

public DateTime? LastLoginAt { get; set; }

public string SecurityStamp { get; set; } = Guid.NewGuid().ToString();

public bool IsActive { get; set; } = true;

public bool IsDeleted { get; set; } = false;

}

public enum UserRole

{

Admin,

Operator,

Guest

}

```

---

## ✅ 2. DTOs/RegisterRequest.cs

```csharp

using System.ComponentModel.DataAnnotations;

public class RegisterRequest

{

[Required(ErrorMessage = "وارد کردن نام کاربری الزامی است")]

[StringLength(100, ErrorMessage = "نام کاربری نباید بیش از 100 کاراکتر باشد")]

public string Username { get; set; }

[Required(ErrorMessage = "وارد کردن رمز عبور الزامی است")]

[StringLength(100, MinimumLength = 6, ErrorMessage = "رمز عبور باید حداقل 6 کاراکتر باشد")]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required(ErrorMessage = "وارد کردن نقش کاربر الزامی است")]

public UserRole Role { get; set; }

[EmailAddress(ErrorMessage = "ایمیل وارد شده معتبر نیست")]

public string Email { get; set; }

}

```

---

## ✅ 3. DTOs/LoginRequest.cs

```csharp

using System.ComponentModel.DataAnnotations;

public class LoginRequest

{

[Required(ErrorMessage = "وارد کردن نام کاربری الزامی است")]

[StringLength(100, ErrorMessage = "نام کاربری نباید بیش از 100 کاراکتر باشد")]

public string Username { get; set; }

[Required(ErrorMessage = "وارد کردن رمز عبور الزامی است")]

[DataType(DataType.Password)]

public string Password { get; set; }

}

```

---

## ✅ 4. Data/AppDbContext.cs

```csharp

using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext

{

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

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

{

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<User>()

.HasIndex(u => u.Username)

.IsUnique();

base.OnModelCreating(modelBuilder);

}

}

```

---

## ✅ 5. Services/IUserService.cs + UserService.cs

### `IUserService.cs`

```csharp

using System.Threading.Tasks;

public interface IUserService

{

Task<User> RegisterAsync(RegisterRequest request);

Task<string> LoginAsync(LoginRequest request);

}

```

### `UserService.cs`

```csharp

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using BCrypt.Net;

public class UserService : IUserService

{

private readonly AppDbContext \_context;

private readonly IJwtService \_jwtService;

public UserService(AppDbContext context, IJwtService jwtService)

{

\_context = context;

\_jwtService = jwtService;

}

public async Task<User> RegisterAsync(RegisterRequest request)

{

if (await \_context.Users.AnyAsync(u => u.Username == request.Username))

throw new Exception("نام کاربری تکراری است");

var user = new User

{

Username = request.Username,

PasswordHash = BCrypt.HashPassword(request.Password),

Role = request.Role,

Email = request.Email

};

await \_context.Users.AddAsync(user);

await \_context.SaveChangesAsync();

return user;

}

public async Task<string> LoginAsync(LoginRequest request)

{

var user = await \_context.Users

.FirstOrDefaultAsync(u => u.Username == request.Username && !u.IsDeleted && u.IsActive);

if (user == null || !BCrypt.Verify(request.Password, user.PasswordHash))

return null;

user.LastLoginAt = DateTime.UtcNow;

await \_context.SaveChangesAsync();

return \_jwtService.GenerateToken(user);

}

}

```

---

## ✅ 6. Services/IJwtService.cs + JwtService.cs

### `IJwtService.cs`

```csharp

public interface IJwtService

{

string GenerateToken(User user);

}

```

### `JwtService.cs`

```csharp

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

public class JwtService : IJwtService

{

private readonly string \_secretKey;

private readonly int \_expireHours;

public JwtService(IConfiguration configuration)

{

\_secretKey = configuration["JwtSettings:SecretKey"];

\_expireHours = int.Parse(configuration["JwtSettings:ExpireHours"] ?? "2");

}

public string GenerateToken(User user)

{

var key = Encoding.ASCII.GetBytes(\_secretKey);

var claims = new[]

{

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

new Claim(ClaimTypes.Name, user.Username),

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

};

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(claims),

Expires = DateTime.UtcNow.AddHours(\_expireHours),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var tokenHandler = new JwtSecurityTokenHandler();

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

}

```

---

## ✅ 7. Controllers/AuthController.cs

```csharp

using Microsoft.AspNetCore.Mvc;

using System.Threading.Tasks;

[ApiController]

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

public class AuthController : ControllerBase

{

private readonly IUserService \_userService;

public AuthController(IUserService userService)

{

\_userService = userService;

}

[HttpPost("register")]

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

{

if (!ModelState.IsValid)

return BadRequest(ModelState);

try

{

var user = await \_userService.RegisterAsync(request);

return Ok(new { user.Id, user.Username, user.Role });

}

catch (Exception ex)

{

return BadRequest(new { message = ex.Message });

}

}

[HttpPost("login")]

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

{

if (!ModelState.IsValid)

return BadRequest(ModelState);

var token = await \_userService.LoginAsync(request);

if (token == null)

return Unauthorized("نام کاربری یا رمز عبور اشتباه است");

return Ok(new { token });

}

}

```

---

## ✅ 8. Program.cs

```csharp

using Microsoft.AspNetCore.Builder;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Configuration;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Serilog;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Host.UseSerilog((ctx, lc) => lc

.WriteTo.Console()

.ReadFrom.Configuration(ctx.Configuration));

builder.Services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped<IUserService, UserService>();

builder.Services.AddScoped<IJwtService, JwtService>();

var key = Encoding.ASCII.GetBytes(builder.Configuration["JwtSettings:SecretKey"]);

builder.Services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(x =>

{

x.RequireHttpsMetadata = false;

x.SaveToken = true;

x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(key),

ValidateIssuer = false,

ValidateAudience = false,

ClockSkew = TimeSpan.Zero

};

});

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

```

---

## ✅ 9. appsettings.json

```json

{

"ConnectionStrings": {

"DefaultConnection": "Server=.;Database=AuthDb;Trusted\_Connection=True;"

},

"JwtSettings": {

"SecretKey": "کلید\_خیلی\_خیلی\_طولانی\_و\_امن",

"ExpireHours": "2"

},

"Serilog": {

"MinimumLevel": "Information",

"WriteTo": [

{ "Name": "Console" }

]

}

}

```

---

## ✅ 10. Install NuGet Packages

```bash

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

dotnet add package Swashbuckle.AspNetCore

dotnet add package Serilog.AspNetCore

dotnet add package BCrypt.Net-Next

```

---

## ✅ 11. Migration

```bash

dotnet ef migrations add InitialCreate

dotnet ef database update

```

---

## ✅ 12. Seed Data — کاربر Admin اولیه

در `Program.cs` بعد از `app.Build()`:

```csharp

using (var scope = app.Services.CreateScope())

{

var context = scope.ServiceProvider.GetRequiredService<AppDbContext>();

await context.Database.MigrateAsync();

if (!await context.Users.AnyAsync(u => u.Role == UserRole.Admin))

{

var admin = new User

{

Username = "admin",

PasswordHash = BCrypt.HashPassword("Admin@123"),

Role = UserRole.Admin,

Email = "admin@example.com",

IsActive = true

};

context.Users.Add(admin);

await context.SaveChangesAsync();

}

}

```

---

## ✅ 13. تست API با Swagger

بعد از اجرای پروژه، به آدرس زیر بروید:

```

https://localhost:5001/swagger

```

و دو endpoint زیر را تست کنید:

- `POST /api/auth/register`

- `POST /api/auth/login`

---

## ✅ 14. پیشنهادات توسعه

| عنصر | پیشنهاد |

|------|---------|

| FluentValidation | اعتبارسنجی دقیق‌تر |

| Rate Limiting | جلوگیری از حملات Brute Force |

| Refresh Token | تمدید توکن بدون لاگین مجدد |

| Global Query Filter | فیلتر خودکار `IsDeleted` |

| Audit Log | ثبت زمان ورود/خروج کاربر |

---

## ✅ خلاصه

این پروژه:

- شامل تمام بخش‌های Authentication با JWT است

- از EF Core و SQL Server استفاده می‌کند

- شامل Register و Login است

- به همراه Swagger و Logging

- قابل توسعه برای محیط Production

---

اگر مایل باشید، این پروژه را به صورت ZIP آماده کنم و برایتان آپلود کنم.

فقط کافی است بفرمایید:

✅ \*\*بله، فایل زیپ شده را می‌خواهم\*\*

یا

✅ \*\*به من کد تست واحد و FluentValidation هم اضافه کن\*\*

منتظر پاسختون هستم!