✅ Lab Title: JWT Authentication & Role-based Authorization in ASP.NET Core Web API

🧰 Prerequisites

- Visual Studio or VS Code
- .NET SDK (7.0 or later)
- SQL Server / SQLite
- Postman (for testing APIs)

📁 Step 1: Create a new ASP.NET Core Web API Project

```
dotnet new webapi -n JwtAuthDemo
cd JwtAuthDemo
```

🧩 Step 2: Install Required NuGet Packages

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Tools
```

🧱 Step 3: Create the Data Models

Models/User.cs

```
public class User
{
    public int Id { get; set; }
    public string Username { get; set; }
    public string PasswordHash { get; set; }
    public string Role { get; set; } // e.g., "User", "Admin"
}
```

🗃️ Step 4: Create the DbContext

Data/AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using JwtAuthDemo.Models;

public class AppDbContext : DbContext
{
```

```
    public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options) { }

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

🛠️ Step 5: Configure EF Core in `Program.cs`

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

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConn
ection")));
```

Add Connection String in `appsettings.json`

```
"ConnectionStrings": {
  "DefaultConnection":
"Server=.;Database=JwtAuthDemoDb;Trusted_Connection=True;"
}
```

🔐 Step 6: Add JWT Authentication Configuration

In `Program.cs`:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;

var key = "ThisIsASecretKeyForJwt"; // store this in a secure place

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(key))
        };
    });

builder.Services.AddAuthorization();
```

Also add:

```
app.UseAuthentication();
app.UseAuthorization();
```

📦 Step 7: Create DTOs for Registration and Login

DTOs/RegisterDto.cs

```
public class RegisterDto
{
    public string Username { get; set; }
    public string Password { get; set; }
    public string Role { get; set; } // "User" or "Admin"
}
```

DTOs/LoginDto.cs

```
public class LoginDto
{
    public string Username { get; set; }
    public string Password { get; set; }
}
```

🔑 Step 8: Create AuthController with Register & Login

Controllers/AuthController.cs

```
using Microsoft.AspNetCore.Mvc;
using JwtAuthDemo.Models;
using JwtAuthDemo.DTOs;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

[ApiController]
[Route("api/[controller]")]
public class AuthController : ControllerBase
{
    private readonly AppDbContext _context;
    private readonly string _key = "ThisIsASecretKeyForJwt";

    public AuthController(AppDbContext context)
    {
        _context = context;
    }

    [HttpPost("register")]
    public IActionResult Register(RegisterDto dto)
    {
        if (_context.Users.Any(u => u.Username == dto.Username))
            return BadRequest("User already exists");

        var user = new User
        {
            Username = dto.Username,
            PasswordHash = BCrypt.Net.BCrypt.HashPassword(dto.Password),
            Role = dto.Role
        };

        _context.Users.Add(user);
        _context.SaveChanges();
```

```
        return Ok("User registered successfully");
    }

    [HttpPost("login")]
    public IActionResult Login(LoginDto dto)
    {
        var user = _context.Users.FirstOrDefault(u => u.Username ==
dto.Username);
        if (user == null || !BCrypt.Net.BCrypt.Verify(dto.Password,
user.PasswordHash))
            return Unauthorized();

        var tokenHandler = new JwtSecurityTokenHandler();
        var tokenKey = Encoding.UTF8.GetBytes(_key);
        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.Name, user.Username),
            new Claim(ClaimTypes.Role, user.Role)
        };

        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(claims),
            Expires = DateTime.UtcNow.AddHours(1),
            SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(tokenKey), SecurityAlgorithms.HmacSha256Signature)
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);
        var tokenString = tokenHandler.WriteToken(token);

        return Ok(new { Token = tokenString });
    }
}
```

🚪 Step 9: Create Test API Endpoints with Different Access Levels

Controllers/TestController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

[ApiController]
[Route("api/[controller]")]
public class TestController : ControllerBase
{
    [HttpGet("anonymous")]
    public IActionResult AnonymousAccess() => Ok("Hello Anonymous!");

    [Authorize]
    [HttpGet("user")]
    public IActionResult AuthenticatedUser() => Ok("Hello Authenticated
User!");

    [Authorize(Roles = "Admin")]
    [HttpGet("admin")]
```

```
    public IActionResult AdminOnly() => Ok("Hello Admin!");
}
```

🧪 Step 10: Run Migrations and Update Database

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

🧪 Step 11: Test in Postman

1. **Register a User**
   - `POST /api/auth/register`
   - Body:

     ```
     json
     CopyEdit
     {
       "username": "admin1",
       "password": "Admin@123",
       "role": "Admin"
     }
     ```

2. **Login to get JWT Token**
   - `POST /api/auth/login`
   - Save the returned token.
3. **Call endpoints**
   - `GET /api/test/anonymous` → No token required
   - `GET /api/test/user` → Add `Authorization: Bearer <token>` (any logged-in user)
   - `GET /api/test/admin` → Token must be for user with `"role": "Admin"`

📘 Summary

- Built JWT Auth from scratch using EF Core.
- Configured role-based access.
- Used secure password hashing (BCrypt).
- Tested with Postman.