EASTERN INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

Practice Assignment – Quarter 4, 2024-2025
Course Name: Backend Development
Course Code: CSW 306
Student's Full Name:
Student ID:

Practice Assignment 5

LIBRARY MANAGEMENT SYSTEM

Authentication & Authorization (JWT)

Learning Objectives

By the end of this lab, you should be able to:

- Authenticate users using credentials and issue JWTs.
- Protect API routes with `[Authorize]`.
- Implement role-based authorization (e.g., Admin, User).
- Understand how claims and tokens work in stateless APIs.

Guide Setup

1. Step 1 – Add JWT Settings to appsettings.json (MUST BE FIRST)

```
"JwtSettings": {
  "SecretKey": "your-strong-random-key-should-be-32+chars",
  "Issuer": "YourCompany.AuthServer",
  "Audience": "YourCompany.ClientApp",
  "ExpiryMinutes": 60
}
```

➔ This configuration must be added **before** configuring JWT in Program.cs.

---

2. Step 2 – Install JWT Authentication NuGet Package

In your NuGet package manager console:

Microsoft.AspNetCore.Authentication.JwtBearer

Ensure the package version matches your .NET Core project version.

3. Step 3 – Configure JWT in Program.cs

```
var builder = WebApplication.CreateBuilder(args);
// Load JWT settings from configuration
var jwtSettings = builder.Configuration.GetSection("JwtSettings");
var key = jwtSettings["SecretKey"];
var issuer = jwtSettings["Issuer"];
var audience = jwtSettings["Audience"];
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = issuer,
            ValidAudience = audience,
      IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key))
        };
    });
builder.Services.AddAuthorization();
var app = builder.Build();
app.UseAuthentication();
app.UseAuthorization();
```

Exercise 1: Create the User Model and Load from Database

Create a User entity if you haven't already, and retrieve users from your database using AppDbContext. If you have this model in previous, you do not need to create new user model.

Example:

```
public class User
{
    ...............
    public string Password { get; set; } // 🔒 In real systems, this must be hashed
    ...................
}
```

Exercise 2: Implement Login with JWT in AuthController

DTO – LoginRequest.cs

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

Controller – AuthController.cs

```
[ApiController]
[Route("api/[controller]")]
public class AuthController : ControllerBase
{
    private readonly AppDbContext _context;
    private readonly IConfiguration _configuration;

    public AuthController(AppDbContext context, IConfiguration configuration)
    {
        _context = context;
        _configuration = configuration;
    }
    [HttpPost("login")]
    public IActionResult Login([FromBody] LoginRequest request)
    {
        var user = _context.Users.FirstOrDefault(u =>
            u.Username == request.Username && u.Password == request.Password);
        if (user == null)
            return Unauthorized();
        var token = GenerateJwtToken(user);
        return Ok(new { token });
    }
    private string GenerateJwtToken(User user)
    {
        var claims = new[]
        {
            new Claim(ClaimTypes.Name, user.Username),
            new Claim(ClaimTypes.Role, user.Role)
        };
        var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["JwtSettings:Secret
Key"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            issuer: _configuration["JwtSettings:Issuer"],
            audience: _configuration["JwtSettings:Audience"],
            claims: claims,
            expires: DateTime.Now.AddMinutes(60),
            signingCredentials: creds
        );
        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

Exercise 3: Create a Protected Endpoint

Endpoint: GET /api/users/me

```
[ApiController]
[Route("api/[controller]")]
public class UsersController : ControllerBase
{
    [HttpGet("me")]
    [Authorize]
    public IActionResult GetMe()
    {
        var username = User.Identity?.Name;
        var role = User.FindFirst(ClaimTypes.Role)?.Value;
        return Ok(new { username, role });
    }
}
```
🔐 This endpoint requires a valid JWT in the Authorization header.

Exercise 4: Role-Based Authorization

Endpoint: GET /api/admin/dashboard

```
[HttpGet("dashboard")]
[Authorize(Roles = "Admin")]
public IActionResult AdminDashboard()
{
    return Ok("Welcome, Admin!");
}
```
Users without the **Admin** role should receive **403 Forbidden**.

Exercise 5: Test with Swagger / Postman

1. Test login with valid and invalid credentials:
   - POST /api/auth/login
2. Copy the returned JWT token.
3. Use the token in:
   - GET /api/users/me (requires any valid user)
   - GET /api/admin/dashboard (requires Admin role)

Set the Authorization header in format: Authorization: Bearer <your_token_here>

Submission Checklist

Make sure the following are included in your submission:

- POST /api/auth/login returns a valid JWT on success.
- GET /api/users/me requires authentication via [Authorize].
- GET /api/admin/dashboard requires Admin role via [Authorize(Roles = "Admin")].
- JWT secret and settings stored in appsettings.json.
- Clean and testable code, verified via Swagger or Postman.
- User data retrieved from the database (not hardcoded list).

⚠️ Important Notes

- **Passwords should be hashed** in real applications. For practice, plaintext is acceptable but insecure in production.
- Use **environment variables or secrets manager** in production for storing JWT secrets.