**MICROSERVICES ASSIGNMENT**

**## Question 1: Implement JWT Authentication in ASP.NET Core Web API**

#### Install NuGet Packages

bash

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

#### appsettings.json

json

{

"Jwt": {

"Key": "ThisIsASecretKeyForJwtToken",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

}

}

#### Program.cs

builder.Services.AddAuthentication("Bearer")

.AddJwtBearer("Bearer", options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = builder.Configuration["Jwt:Issuer"],

ValidAudience = builder.Configuration["Jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))

};

});

builder.Services.AddAuthorization();

#### AuthController.cs

[ApiController]

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

public class AuthController : ControllerBase

{

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

if (IsValidUser(model))

{

var token = GenerateJwtToken(model.Username);

return Ok(new { Token = token });

}

return Unauthorized();

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username)

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("ThisIsASecretKeyForJwtToken"));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

issuer: "MyAuthServer",

audience: "MyApiUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(60),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

**## Question 2: Secure an API Endpoint Using JWT**

#### SecureController.cs

[ApiController]

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

public class SecureController : ControllerBase

{

[HttpGet("data")]

[Authorize]

public IActionResult GetSecureData()

{

return Ok("This is protected data.");

}

}

**## Question 3: Add Role-Based Authorization**

#### Modify Token Generation

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, "Admin")

};

#### AdminController.cs

[ApiController]

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

public class AdminController : ControllerBase

{

[HttpGet("dashboard")]

[Authorize(Roles = "Admin")]

public IActionResult GetAdminDashboard()

{

return Ok("Welcome to the admin dashboard.");

}

}

**## Question 4: Validate JWT Token Expiry and Handle Unauthorized Access**

#### Program.cs (Add to `AddJwtBearer`)

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

if (context.Exception.GetType() == typeof(SecurityTokenExpiredException))

{

context.Response.Headers.Add("Token-Expired", "true");

}

return Task.CompletedTask;

}

};