Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

**10**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 1 | Implement the microservice for user profile management, incorporating registration, login, and profile updates using ASP.NET Core Identity. After implementation, test the microservice endpoints using Postman to ensure functionality and seamless interaction. |

Submitted On:

13 June 2024

\_\_\_\_\_\_\_\_\_\_\_\_

(Date: DD/MM/YY)

**Task 1**

Implement the microservice for user profile management, incorporating registration, login, and profile updates using ASP.NET Core Identity. After implementation, test the microservice endpoints using Postman to ensure functionality and seamless interaction.

**ApplicationDbContext**

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Task9.Models;

namespace Task9.DbContext

{

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

}

}

**Controller**

using Microsoft.AspNetCore.Mvc;

namespace Task9.Controllers

{

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Task9.Models;

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

[ApiController]

public class AuthController : ControllerBase

{

private readonly UserManager<ApplicationUser> \_userManager;

private readonly SignInManager<ApplicationUser> \_signInManager;

private readonly IConfiguration \_configuration;

public AuthController(UserManager<ApplicationUser> userManager, SignInManager<ApplicationUser> signInManager, IConfiguration configuration)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_configuration = configuration;

}

[HttpPost("register")]

public async Task<IActionResult> Register([FromBody] RegisterModel model)

{

var user = new ApplicationUser { UserName = model.Username, Email = model.Email, FirstName = model.FirstName, LastName = model.LastName };

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

return Ok(new { message = "User registered successfully" });

}

return BadRequest(result.Errors);

}

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] LoginModel model)

{

var result = await \_signInManager.PasswordSignInAsync(model.Username, model.Password, false, false);

if (result.Succeeded)

{

var appUser = await \_userManager.FindByNameAsync(model.Username);

var token = GenerateJwtToken(appUser);

return Ok(new { token });

}

return Unauthorized();

}

private string GenerateJwtToken(ApplicationUser user)

{

var claims = new[]

{

new Claim(JwtRegisteredClaimNames.Sub, user.UserName),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),

new Claim(ClaimTypes.NameIdentifier, user.Id)

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]));

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

var token = new JwtSecurityToken(

issuer: \_configuration["Jwt:Issuer"],

audience: \_configuration["Jwt:Audience"],

claims: claims,

expires: DateTime.Now.AddMinutes(30),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

[HttpPut("updateProfile")]

public async Task<IActionResult> UpdateProfile([FromBody] ProfileUpdateModel model)

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

var user = await \_userManager.FindByIdAsync(userId);

if (user == null)

{

return NotFound();

}

user.FirstName = model.FirstName;

user.LastName = model.LastName;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

return Ok(new { message = "Profile updated successfully" });

}

return BadRequest(result.Errors);

}

}

}

**Models**

using Microsoft.AspNetCore.Identity;

namespace Task9.Models

{

public class ApplicationUser : IdentityUser

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

}

namespace Task9.Models

{

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

namespace Task9.Models

{

public class ProfileUpdateModel

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

}

namespace Task9.Models

{

public class RegisterModel

{

public string Username { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

}

}

**Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.IdentityModel.Tokens;

using System.Text;

using Task9.DbContext;

using Task9.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

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

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

builder.Services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(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.AddControllers();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

**appsetting.json**

{

"ConnectionStrings": {

"DefaultConnection": "data source=(localdb)\\local;initial catalog=CCLab;user id=local;password=12345;encrypt=False;MultipleActiveResultSets=True;App=EntityFramework"

},

"Jwt": {

"Key": "supersecretkey1234567890123456789012345",

"Issuer": "mzain.com",

"Audience": "mzain.com"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generatedA close up of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

