Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

9

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 1 | Create a microservice for user profile management, including registration, login, and profile updates, using ASP.NET Core Identity. |
| 2 | Build a microservice for managing inventory and stock levels of products using ASP.NET. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Submitted On:

**\_\_\_\_\_\_\_\_\_\_\_**

Task 01: Create a microservice for user profile management, including registration, login, and profile updates, using ASP.NET Core Identity.

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", // This should be at least 32 characters long

"Issuer": "hamza.com",

"Audience": "hamza.com"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

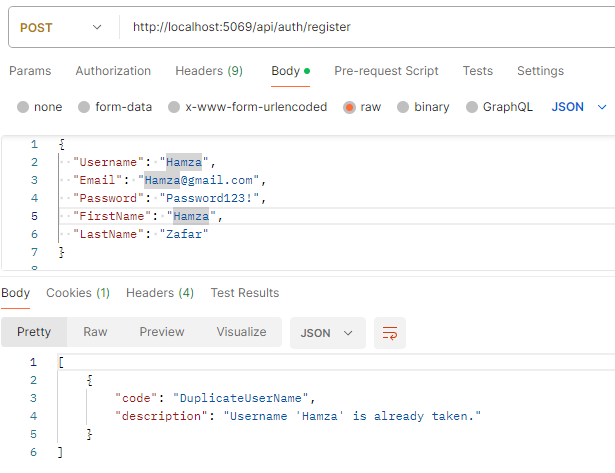
}

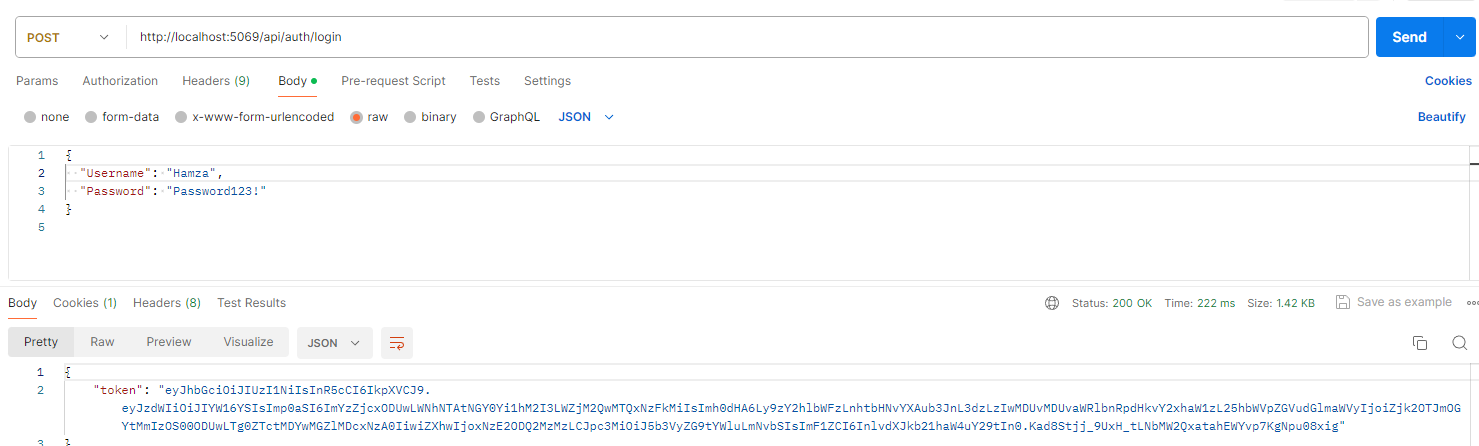
},

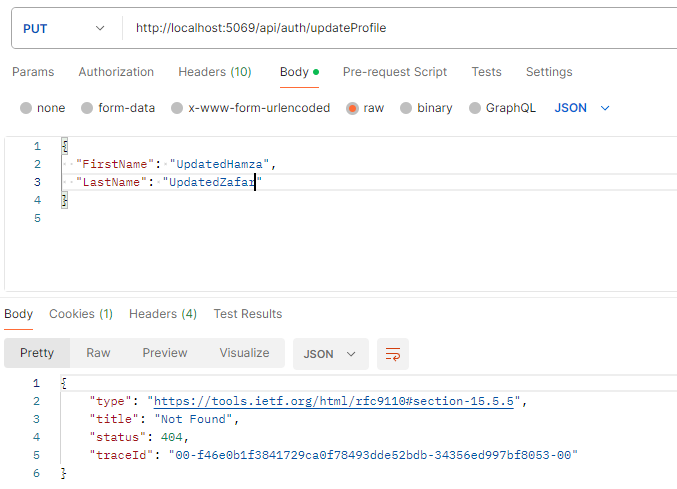
"AllowedHosts": "\*"

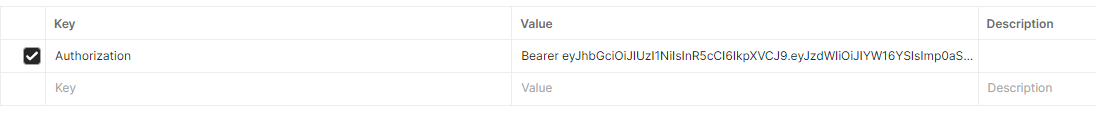
}

Output:









Task 02: Build a microservice for managing inventory and stock levels of products using ASP.NET.

Solution:

Web Service:

Database:

create database DD\_Lab\_9

create table Products(

Id int not null primary key,

Name varchar(255),

Description varchar(255),

Price decimal(10,2));

Model:

namespace Task9\_02.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public decimal Price { get; set; }

}

}

ProductDbContext:

using Microsoft.EntityFrameworkCore;

using Task9\_02.Models;

namespace Task9\_02.DbContexts

{

public class ProductDbContext : DbContext

{

public ProductDbContext(DbContextOptions options) : base(options) { }

public DbSet<Product> products { get; set; }

}

}

Connection String:

"ConnectionStrings": {

"DB": "Data Source=(localdb)\\local; Initial Catalog=DD\_Lab\_9; User Id=local; Password=12345; encrypt=False; MultipleActiveResultSets=True; App=EntityFramework"

}

Program.cs:

builder.Services.AddDbContext<EmployeeContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("DB")));

builder.Services.AddTransient<IProductRepository, ProductRepository>();

ProductRepository:

using Task9\_02.Models;

namespace Task9\_02.Repository

{

public interface IProductRepository

{

Product GetProductById(int id);

IEnumerable<Product> GetProducts();

void DeleteProduct(int id);

void InsertProduct(Product product);

void UpdateProduct(Product product);

void Save();

}

}

using Microsoft.EntityFrameworkCore;

using Task9\_02.DbContexts;

using Task9\_02.Models;

namespace Task9\_02.Repository

{

public class ProductRepository : IProductRepository

{

private readonly ProductDbContext \_dbContext;

public ProductRepository(ProductDbContext dbContext)

{

\_dbContext = dbContext;

}

public void DeleteProduct(int productId)

{

var product = \_dbContext.Products.Find(productId);

\_dbContext.Products.Remove(product);

Save();

}

public Product GetProductById(int productId)

{

return \_dbContext.Products.Find(productId);

}

public IEnumerable<Product> GetProducts()

{

return \_dbContext.Products.ToList();

}

public void InsertProduct(Product product)

{

\_dbContext.Add(product);

Save();

}

public void UpdateProduct(Product product)

{

\_dbContext.Entry(product).State = EntityState.Modified;

Save();

}

public void Save()

{

\_dbContext.SaveChanges();

}

}

}

ProductController:

using Microsoft.AspNetCore.Mvc;

using System.Transactions;

using Task9\_02.Models;

using Task9\_02.Repository;

namespace Task9\_02.Controllers

{

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

[ApiController]

public class ProductsController : ControllerBase

{

private readonly IProductRepository \_productRepository;

public ProductsController(IProductRepository productRepository)

{

\_productRepository = productRepository;

}

[HttpGet]

public IActionResult Get()

{

var products = \_productRepository.GetProducts();

return new OkObjectResult(products);

}

[HttpGet("{id}", Name = "Get")]

public IActionResult Get(int id)

{

var product = \_productRepository.GetProductById (id);

return new OkObjectResult(product);

}

[HttpPost]

public IActionResult Post([FromBody] Product product)

{

using (var scope = new TransactionScope())

{

\_productRepository.InsertProduct(product); scope.Complete();

return CreatedAtAction(nameof(Get), new { id = product.Id },

product);

}

}

[HttpPut]

public IActionResult Put([FromBody] Product product)

{

if (product != null)

{

using (var scope = new TransactionScope())

{

\_productRepository.UpdateProduct(product); scope.Complete();

return new OkResult();

}

}

return new NoContentResult();

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

\_productRepository.DeleteProduct(id); return new OkResult();

}

}

}

Output:

