**ASP.NET CORE WEB API HANDSON DAY - 77**

**AdminController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using PracticeCheck.Models;

namespace PracticeCheck.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    [Authorize]

    [Authorize(Roles ="Admin")]

    public class AdminController : ControllerBase

    {

        // GET: api/Admin

        [HttpGet]

        public IEnumerable<MenuItem> Get()

        {

            return MenuItemOperation.GetConnection();

        }

        // PUT: api/Admin/5

        [HttpPut("{id}")]

        public IActionResult Put(int id, [FromBody] MenuItem menuitem)

        {

           MenuItemOperation.Update(id,menuitem);

            return Ok();

        }

    }

}

**AnonymousUserController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using PracticeCheck.Models;

namespace PracticeCheck.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    [AllowAnonymous]

    public class AnonymousUserController : ControllerBase

    {

        // GET: api/AnonynousUser

        [HttpGet]

        public IEnumerable<MenuItem> Get()

        {

           return MenuItemOperation.GetConnection();

        }

    }

}

**AuthController.cs**

using System;

using System.Collections.Generic;

using System.IdentityModel.Tokens.Jwt;

using System.Linq;

using System.Security.Claims;

using System.Text;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

namespace PracticeCheck.Controllers

{

    //[Route("api/[controller]")]

    [ApiController]

    public class AuthController : ControllerBase

    {

        [HttpGet]

        [Route("api/Auth/{id}")]

        public IActionResult Get(int id)

        {

            if(id == -1)

                return Ok(GenerateJSONWebToken(""));

            if(id == 1)

                return Ok(GenerateJSONWebToken("Admin"));

            else

                return Ok(GenerateJSONWebToken("Customer"));

        }

        private string GenerateJSONWebToken(string userRole)

        {

            var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret"));

            var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

            var claims = new List<Claim>

            {

                new Claim(ClaimTypes.Role, userRole),

            };

            var token = new JwtSecurityToken(

                        issuer: "mySystem",

                        audience: "myUsers",

                        claims: claims,

                        expires: DateTime.Now.AddMinutes(5),

                        signingCredentials: credentials);

            return new JwtSecurityTokenHandler().WriteToken(token);

        }

    }

}

**CustomerController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using PracticeCheck.Models;

namespace PracticeCheck.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    //[Authorize(Roles = "Customer")]

    public class CustomerController : ControllerBase

    {

        // GET: api/Customer

        [HttpGet]

        public IEnumerable<MenuItem> Get()

        {

            DateTime dt = DateTime.Now;

            return MenuItemOperation.GetConnection().Where(p => p.Active == true && p.DateOfLaunch <= dt);

        }

        // GET: api/Customer/5

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

        public object Get(int userid)

        {

            int totalprice=0;

            List<MenuItem> list = new List<MenuItem>(MenuItemOperation.CartList(userid, ref totalprice));

            return new {list,totalprice };

        }

        // POST: api/Customer

        [HttpPost]

        public IActionResult Post([FromBody] List<Cart> cart)

        {

            MenuItemOperation.InsertIntoCart(cart);

            return Ok();

        }

        // DELETE: api/ApiWithActions/5

        [HttpDelete("{cartid}")]

        public string Delete(int cartid)

        {

            return MenuItemOperation.Delete(cartid);

        }

    }

}

**UserDetailsController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using PracticeCheck.Models;

namespace PracticeCheck.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class UserDetailsController : ControllerBase

    {

        // GET: api/User/5

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

        public string Get(int id,[FromBody]string password)

        {

            List<User> list = MenuItemOperation.UserList();

            bool user = list.Any(p => p.Id == id && p.Password == password);

              if (user == true)

                  return "true";

              return "falseSubmission";

        }

        // POST: api/User

        [HttpPost]

        public IActionResult Post([FromBody] User user)

        {

            MenuItemOperation.Insert(user);

            return Ok();

        }

    }

}

**Cart.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace PracticeCheck.Models

{

    public class Cart

    {

        public int Id { get; set; }

        public int MenuItemId { get; set; }

        public int UserId { get; set; }

    }

}

**Category.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace PracticeCheck.Models

{

    public class Category

    {

        public int Id { get; set; }

        public string Name { get; set; }

    }

}

**MenuItem.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace PracticeCheck.Models

{

    public class MenuItem

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public int Price { get; set; }

        public bool Active { get; set; }

        public DateTime DateOfLaunch { get; set; }

        [ForeignKey("Category")]

        public int CategoryId { get; set; }

        public string CategoryName { get; set; }

        public bool FreeDelivery { get; set; }

    }

}

**User.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace PracticeCheck.Models

{

    public class User

    {

        public int Id { get; set; }

        public string UserName { get; set; }

        public string FirstName { get; set; }

        public string LastName { get; set; }

        public string Password { get; set; }

        public string ConfirmPassword { get; set; }

    }

}

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

namespace PracticeCheck

{

    public class Program

    {

        public static void Main(string[] args)

        {

            CreateHostBuilder(args).Build().Run();

        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>

            Host.CreateDefaultBuilder(args)

                .ConfigureWebHostDefaults(webBuilder =>

                {

                    webBuilder.UseStartup<Startup>();

                });

    }

}

**Startup.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.HttpsPolicy;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

namespace PracticeCheck

{

    public class Startup

    {

        public Startup(IConfiguration configuration)

        {

            Configuration = configuration;

        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.

        public void ConfigureServices(IServiceCollection services)

        {

            services.AddControllers();

            services.AddSwaggerGen(c =>

            {

                c.SwaggerDoc("v1", new OpenApiInfo

                {

                    Title = "Swagger Demo",

                    Version = "v1",

                    Description = "TBD",

                    TermsOfService = new Uri("https://www.example.com"),

                    Contact = new OpenApiContact() { Name = "John Doe", Email = "john@xyzmail.com", Url = new Uri("https://www.example.com") },

                    License = new OpenApiLicense() { Name = "License Terms", Url = new Uri("https://www.example.com") }

                });

            });

            string securityKey = "mysuperdupersecret";

            var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(securityKey));

            services.AddAuthentication(x =>

            {

                x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

                x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

                x.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme;

            })

          .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>

          {

              x.TokenValidationParameters = new TokenValidationParameters

              {

                    //what to validate

                  ValidateIssuer = true,

                  ValidateAudience = true,

                  ValidateLifetime = true,

                  ValidateIssuerSigningKey = true,

                  ClockSkew = TimeSpan.FromSeconds(1),

                    //setup validate data

                  ValidIssuer = "mySystem",

                  ValidAudience = "myUsers",

                  IssuerSigningKey = symmetricSecurityKey

              };

          });

        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

        {

            if (env.IsDevelopment())

            {

                app.UseDeveloperExceptionPage();

            }

            app.UseSwagger();

            app.UseSwaggerUI(c =>

            {

                // specifying the Swagger JSON endpoint.

                c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

            });

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthentication();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>

            {

                endpoints.MapControllers();

            });

        }

    }

}

**OUTPUT:**

Graphical user interface, application

Description automatically generatedGraphical user interface, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated



