**Hands-On: Stage 4 - Final Check Case Study on Web API - Day 7 – Hands on**

1. **FSE Web API Movie Cruiser APP**

**AdminController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FinalCheck.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace FinalCheck.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    [Authorize(Roles = "Admin")]

    public class AdminController : ControllerBase

    {

        // GET: api/Admin

        [HttpGet]

        public IEnumerable<Movie> Get()

        {

            return MenuItemOperation.GetConnection();

        }

        // PUT: api/Admin/5

        [HttpPut("{id}")]

        public IActionResult Put(int id, [FromBody] Movie movieitem)

        {

            MenuItemOperation.Update(id, movieitem);

            return Ok();

        }

    }

}

**AnonymousUserController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FinalCheck.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace FinalCheck.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    [AllowAnonymous]

    public class AnonymousUserController : ControllerBase

    {

        // GET: api/AnonymousUser

        [HttpGet]

        public IEnumerable<Movie> 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 FinalCheck.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 FinalCheck.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace FinalCheck.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    [Authorize(Roles = "Customer")]

    public class CustomerController : ControllerBase

    {

        // GET: api/Customer

        [HttpGet]

        public IEnumerable<Movie> Get()

        {

            DateTime dt = DateTime.Now;

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

        }

        // GET: api/Customer/5

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

        public object Get(int userid)

        {

            int count = 0;

            List<Movie> list = new List<Movie>(MenuItemOperation.favoriteList(userid, ref count));

            return new { list, count };

        }

        // POST: api/Customer

        [HttpPost]

        public IActionResult Post([FromBody] List<Favorites> favorite)

        {

            MenuItemOperation.InsertIntoFavorites(favorite);

            return Ok();

        }

        // DELETE: api/ApiWithActions/5

        [HttpDelete("{favoriteid}")]

        public string Delete(int favoriteid)

        {

            return MenuItemOperation.Delete(favoriteid);

        }

    }

}

**UserDetailsController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FinalCheck.Models;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace FinalCheck.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();

        }

    }

}

**Favorites.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace FinalCheck.Models

{

    public class Favorites

    {

        public int Id { get; set; }

        public int MovieId { get; set; }

        public int UserId { get; set; }

    }

}

**Genre.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace FinalCheck.Models

{

    public class Genre

    {

        public int Id { get; set; }

        public string Name { get; set; }

    }

}

**Movie.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 FinalCheck

{

    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();

            });

        }

    }

}

**User.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace FinalCheck.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; }

    }

}

**MenuItemOperation.cs**

using FinalCheck.Models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Threading.Tasks;

namespace FinalCheck

{

    public class MenuItemOperation

    {

        public static string sqlDataSource = "Server=DESKTOP-B0SQG0N;Database=MovieCruiser;Trusted\_Connection=True;MultipleActiveResultSets=true;";

        public static IEnumerable<Movie> GetConnection()

        {

            List<Movie> Items = new List<Movie>();

            var list = new List<string> { "Science Fiction", "Superhero", "Romance", "Comedy","Adventure","Thriller"};

            using (SqlConnection con = new SqlConnection(sqlDataSource))

            {

                con.Open();

                using (SqlCommand cmd = new SqlCommand())

                {

                    cmd.Connection = con;

                    cmd.CommandText = "select \* from Movie";

                    SqlDataReader rd = cmd.ExecuteReader();

                    while (rd.Read())

                    {

                        Items.Add(new Movie

                        {

                            Id = Convert.ToInt32(rd["Id"]),

                            Title = rd["Title"].ToString(),

                            BoxOffice = rd["BoxOffice"].ToString(),

                            Active = Convert.ToBoolean(rd["Active"]),

                            DateOfLaunch = Convert.ToDateTime(rd["DateOfLaunch"]),

                            GenreId = Convert.ToInt32(rd["GenreId"]),

                            GenreType = list[Convert.ToInt32(rd["GenreId"]) - 1].ToString(),

                            HasTeaser = Convert.ToBoolean(rd["HasTeaser"]),

                        });

                    }

                    con.Close();

                }

                return Items;

            }

        }

        public static void Update(int id, Movie movieitem)

        {

            var list = new List<string> { "Science Fiction", "Superhero", "Romance", "Comedy", "Adventure", "Thriller" };

            using (SqlConnection con = new SqlConnection(sqlDataSource))

            {

                SqlCommand cmd = new SqlCommand("UpdateMovie", con);

                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@Id", id);

                cmd.Parameters.AddWithValue("@Title", movieitem.Title);

                cmd.Parameters.AddWithValue("@BoxOffice", movieitem.BoxOffice);

                cmd.Parameters.AddWithValue("@Active", movieitem.Active);

                cmd.Parameters.AddWithValue("@DateOfLaunch", movieitem.DateOfLaunch);

                cmd.Parameters.AddWithValue("@GenreId", movieitem.GenreId);

                cmd.Parameters.AddWithValue("@HasTeaser", movieitem.HasTeaser);

                con.Open();

                cmd.ExecuteNonQuery();

                con.Close();

            }

        }

        public static void Insert(User user)

        {

            SqlConnection con = new SqlConnection(sqlDataSource);

            SqlCommand sqlCmd = new SqlCommand();

            sqlCmd.CommandType = CommandType.Text;

            sqlCmd.CommandText = "INSERT INTO UserDetails(Id,UserName,FirstName,LastName,Password,ConfirmPassword) Values (@Id,@UserName,@FirstName,@LastName,@Password,@ConfirmPassword)";

            sqlCmd.Connection = con;

            sqlCmd.Parameters.AddWithValue("@Id", user.Id);

            sqlCmd.Parameters.AddWithValue("@UserName", user.UserName);

            sqlCmd.Parameters.AddWithValue("@FirstName", user.FirstName);

            sqlCmd.Parameters.AddWithValue("@LastName", user.LastName);

            sqlCmd.Parameters.AddWithValue("@Password", user.Password);

            sqlCmd.Parameters.AddWithValue("@ConfirmPassword", user.ConfirmPassword);

            con.Open();

            sqlCmd.ExecuteNonQuery();

            con.Close();

        }

        public static List<User> UserList()

        {

            List<User> users = new List<User>();

            using (SqlConnection con = new SqlConnection(sqlDataSource))

            {

                con.Open();

                using (SqlCommand cmd = new SqlCommand())

                {

                    cmd.Connection = con;

                    cmd.CommandText = "select \* from UserDetails";

                    SqlDataReader rd = cmd.ExecuteReader();

                    while (rd.Read())

                    {

                        users.Add(new User

                        {

                            Id = Convert.ToInt32(rd["Id"]),

                            UserName = rd["UserName"].ToString(),

                            FirstName = rd["FirstName"].ToString(),

                            LastName = rd["LastName"].ToString(),

                            Password = rd["Password"].ToString(),

                            ConfirmPassword = rd["ConfirmPassword"].ToString()

                        });

                    }

                    con.Close();

                }

            }

            return users;

        }

        public static void InsertIntoFavorites(List<Favorites> favorite)

        {

            SqlConnection con = new SqlConnection(sqlDataSource);

            SqlCommand sqlCmd = new SqlCommand();

            sqlCmd.CommandType = CommandType.Text;

            sqlCmd.CommandText = "INSERT INTO Favorites(Id,MovieId,UserId) Values (@Id,@MovieId,@UserId)";

            sqlCmd.Connection = con;

            con.Open();

            foreach (var i in favorite)

            {

                sqlCmd.Parameters.AddWithValue("@Id", i.Id);

                sqlCmd.Parameters.AddWithValue("@MovieId", i.MovieId);

                sqlCmd.Parameters.AddWithValue("@UserId", i.UserId);

                sqlCmd.ExecuteNonQuery();

                sqlCmd.Parameters.Clear();

            }

            con.Close();

        }

        public static List<Movie> favoriteList(int userid, ref int count)

        {

            List<Movie> Items = new List<Movie>();

            List<int> list = new List<int>();

            var l = new List<string> { "Science Fiction", "Superhero", "Romance", "Comedy", "Adventure", "Thriller" };

            using (SqlConnection con = new SqlConnection(sqlDataSource))

            {

                con.Open();

                using (SqlCommand cmd = new SqlCommand())

                {

                    cmd.Connection = con;

                    cmd.CommandText = "select MovieId from Favorites where Userid = @userid";

                    cmd.Parameters.AddWithValue("@userid", userid);

                    SqlDataReader rd = cmd.ExecuteReader();

                    while (rd.Read())

                    {

                        list.Add(Convert.ToInt32(rd["MovieId"]));

                    }

                    rd.Close();

                    foreach (var i in list)

                    {

                        cmd.CommandText = "select \* from Movie where Id = @i";

                        cmd.Parameters.AddWithValue("@i", i);

                        SqlDataReader rd1 = cmd.ExecuteReader();

                        while (rd1.Read())

                        {

                            count++;

                            Items.Add(new Movie

                            {

                                Id = Convert.ToInt32(rd1["Id"]),

                                Title = rd1["Title"].ToString(),

                                BoxOffice = rd1["BoxOffice"].ToString(),

                                Active = Convert.ToBoolean(rd1["Active"]),

                                DateOfLaunch = Convert.ToDateTime(rd1["DateOfLaunch"]),

                                GenreId = Convert.ToInt32(rd1["GenreId"]),

                                GenreType = l[Convert.ToInt32(rd1["GenreId"]) - 1].ToString(),

                                HasTeaser = Convert.ToBoolean(rd1["HasTeaser"]),

                            });

                        }

                        cmd.Parameters.Clear();

                        rd1.Close();

                    }

                    con.Close();

                }

            }

            return Items;

        }

        public static string Delete(int Favoriteid)

        {

            SqlConnection con = new SqlConnection(sqlDataSource);

            SqlCommand sqlCmd = new SqlCommand();

            sqlCmd.CommandType = CommandType.Text;

            sqlCmd.CommandText = "delete from Favorites where Id =@favoriteid";

            sqlCmd.Connection = con;

            con.Open();

            sqlCmd.Parameters.AddWithValue("@favoriteid", Favoriteid);

            int i = sqlCmd.ExecuteNonQuery();

            if (i >= 1)

                return "record deleted";

            else

                return "no record";

        }

    }

}

**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 FinalCheck

{

    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 FinalCheck

{

    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 =>

            {

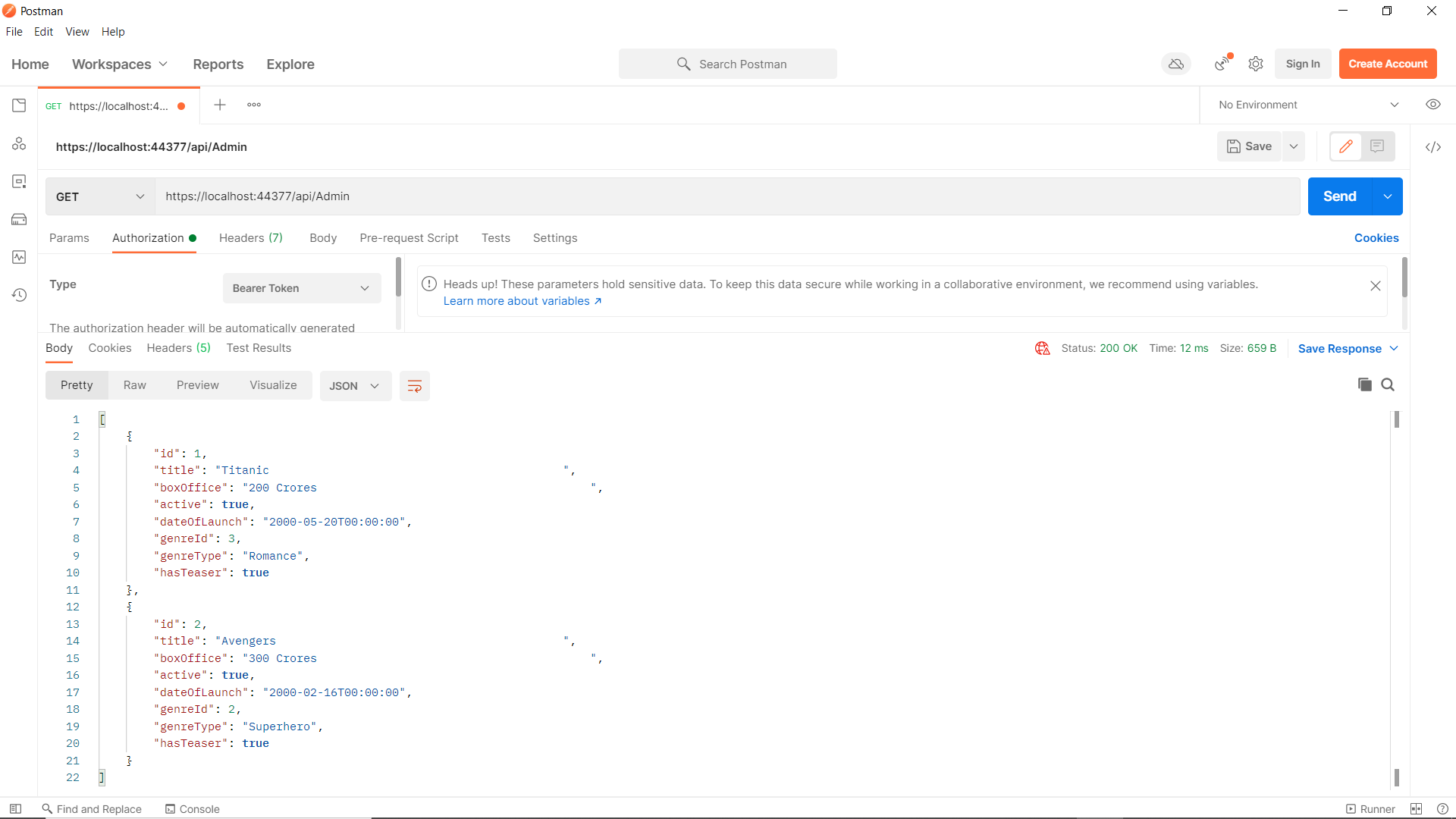
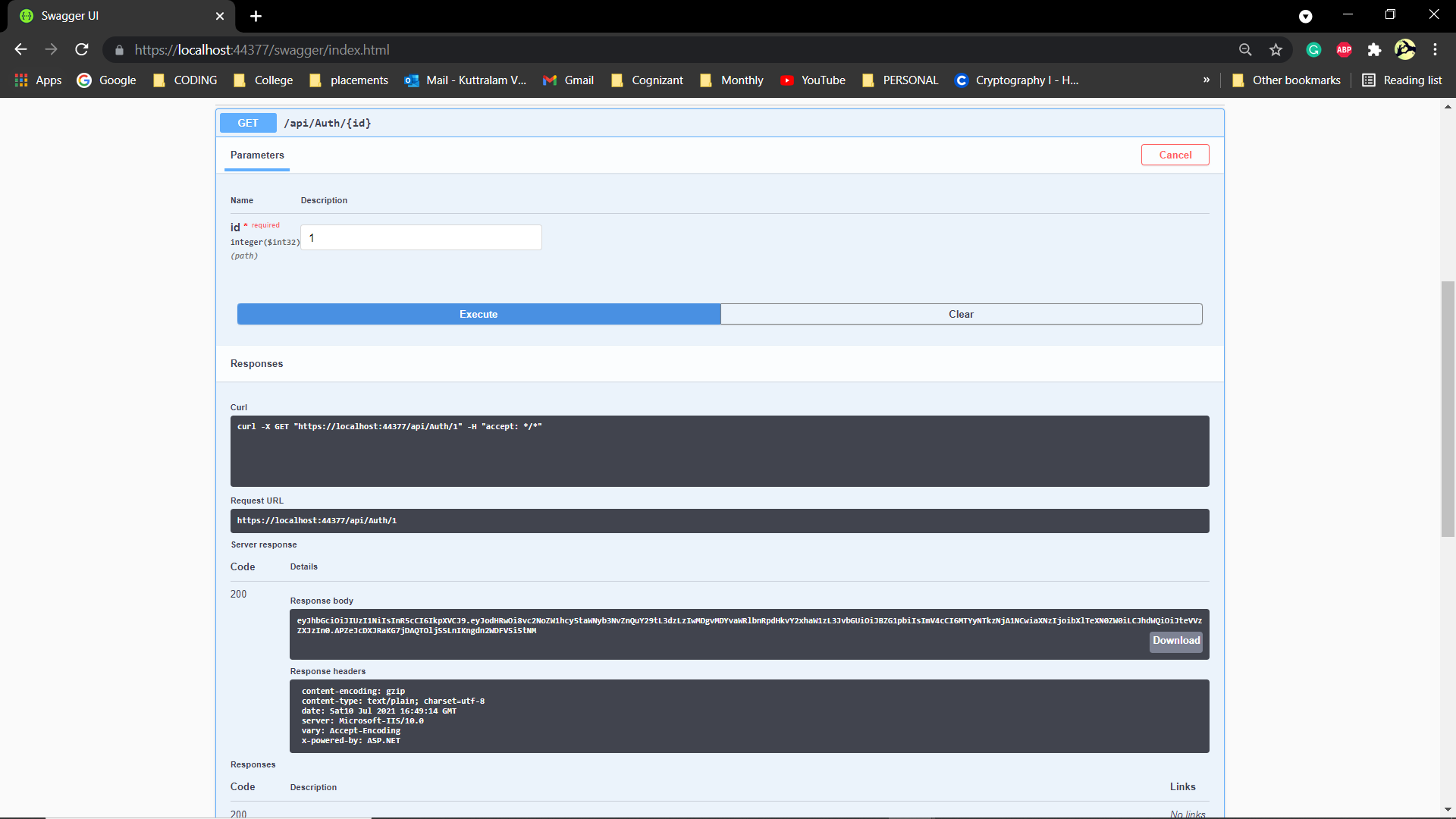
                endpoints.MapControllers();

            });

        }

    }

}

**OUTPUT:**

