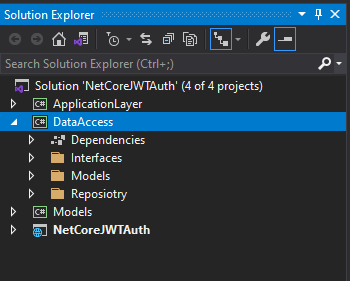
**.Net-Core-JWT-Authentication-Authorization-With-JWT-MiddleWare-Custom-AuthenticationFilter**

1. Create .Net core web api project
2. Create **DataAccess** .net core class library into solution



**===================================================================================**

1. Install below library into **DataAccess class** library using **NuGet package manager**

**Microsoft.EntityFrameworkCore.SqlServer**

**Microsoft.EntityFrameworkCore.SqlServer.Design**

**Microsoft.EntityFrameworkCore.Tools**

**===================================================================================**

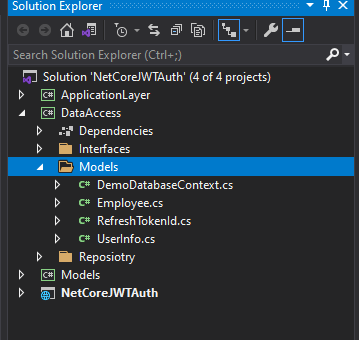
1. Create **Model**s folder in **DataAccess** class library

**===================================================================================**

1. Execute below query using **Package manager console** to generate **DBContext** class and **entities**

**PM> Scaffold-DbContext "Server=DESKTOP-3DF0FM6\SQLEXPRESS;Database=DemoDatabase;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models**

After executing this command it will generate below files in Models folder



**===================================================================================**

1. Open **DemoDatabaseContext.cs** file and comment the **OnConfiguring** method

**===================================================================================**

1. Add connection string and JWT setting in **appsettings.json** file

{

"ConnectionStrings": {

"DemoDBConnection": "Server=DESKTOP-3DF0FM6\\SQLEXPRESS;Database=DemoDatabase;Trusted\_Connection=True"

},

"JwtCongifuration": {

"SecretKey": "This is my sample key"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

**===================================================================================**

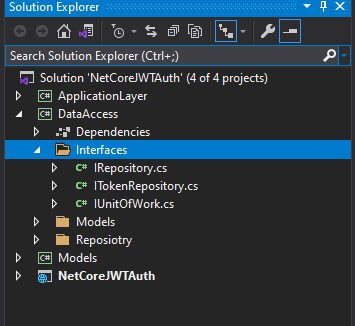
1. Open **startup.cs** file and add below into **ConfigureServices()** method

**services.AddDbContext<DemoDatabaseContext>(item => item.UseSqlServer**

**(Configuration.GetConnectionString("DemoDBConnection")));**

**===================================================================================**

1. Create **Interfaces** folder into **DataAccess** class library and add below interfaces into this folder-



public interface IRepository<T> where T : class

{

Task<IEnumerable<T>> Get();

Task<IEnumerable<T>> Get(Expression<Func<T, bool>> predicate);

T GetById(int id);

Task Add(T entity);

void Update(T entity);

void Delete(T entity);

int SaveChanges();

Task<int> SaveChangesAsync();

}

==================================================================

public interface IUnitOfWork : IDisposable

{

DbContext Context { get; }

//void Commit()

}

==================================================================

public interface ITokenRepository

{

Task<int> CreateRefreshTokenOnLogin(string refreshToken, DateTime expiryTime, int userId);

Task<int> UpdateRefreshTokenOnRefresh(string oldtoken, string newToken, DateTime expiryTime);

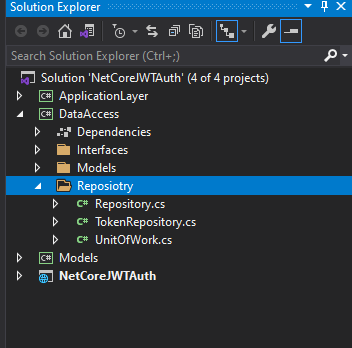
Task<int> RemoveRefreshToken(string refreshToken);

Task<UserInfo> GetUserFromRefreshToken(string refreshToken);

}

**===================================================================================**

1. Create **Repository** folder into **DataAccess** class library and add below **classes** into this folder-



public class UnitOfWork : IUnitOfWork

{

public DbContext Context { get; }

public UnitOfWork(DemoDatabaseContext context)

{

Context = context;

}

public void Dispose()

{

Context.Dispose();

}

}

=============================================================================

public class Repository<T> : IRepository<T> where T : class

{

private readonly IUnitOfWork \_unitOfWork;

protected DbSet<T> dbSet;

protected DbContext dbContext;

public Repository(IUnitOfWork unitOfWork)

{

//\_unitOfWork = unitOfWork;

\_unitOfWork = unitOfWork;

if (\_unitOfWork.Context != null)

{

dbContext = \_unitOfWork.Context;

dbSet = dbContext.Set<T>();

}

}

public async Task<IEnumerable<T>> Get()

{

//return \_unitOfWork.Context.Set<T>().AsEnumerable<T>();

return await dbSet.ToListAsync();

}

public async Task<IEnumerable<T>> Get(System.Linq.Expressions.Expression<Func<T, bool>> predicate)

{

//return \_unitOfWork.Context.Set<T>().Where(predicate).AsEnumerable<T>();

return await dbSet.Where(predicate).ToListAsync();

}

public T GetById(int id)

{

return dbSet.Find(id);

}

public async Task Add(T entity)

{

// \_unitOfWork.Context.Set<T>().Add(entity);

await dbSet.AddAsync(entity);

}

public void Update(T entity)

{

// \_unitOfWork.Context.Set<T>().Update(entity);

dbSet.Update(entity);

}

public void Delete(T entity)

{

//T existing = \_unitOfWork.Context.Set<T>().Find(entity);

//if (existing != null) \_unitOfWork.Context.Set<T>().Remove(existing);

//T existing = dbSet.Find(entity);

//if (existing != null)

//{

// dbSet.Remove(existing);

//}

dbSet.Remove(entity);

}

public int SaveChanges()

{

return \_unitOfWork.Context.SaveChanges();

}

public async Task<int> SaveChangesAsync()

{

return await \_unitOfWork.Context.SaveChangesAsync();

}

}

==================================================================================

public class TokenRepository : Repository<RefreshTokenId>, ITokenRepository

{

public TokenRepository(IUnitOfWork unitOfWork) : base(unitOfWork)

{

if (unitOfWork.Context != null)

{

dbSet = unitOfWork.Context.Set<RefreshTokenId>();

}

}

public async Task<int> CreateRefreshTokenOnLogin(string refreshToken, DateTime expiryTime, int userId)

{

var tokenData = new RefreshTokenId

{

RefreshTokenValue = refreshToken,

ExpiryTime = expiryTime,

UserId = userId

};

dbContext.Set<RefreshTokenId>().Add(tokenData);

return await dbContext.SaveChangesAsync();

}

public async Task<UserInfo> GetUserFromRefreshToken(string refreshToken)

{

var result = await (from r in dbContext.Set<RefreshTokenId>().AsNoTracking()

join u in dbContext.Set<UserInfo>().AsNoTracking()

on r.UserId equals u.UserId

where r.RefreshTokenValue == refreshToken

select new UserInfo

{

UserId = u.UserId,

UserName = u.UserName

}).FirstOrDefaultAsync();

return result;

}

public async Task<int> RemoveRefreshToken(string refreshToken)

{

var refreshTokenObj = await dbContext.Set<RefreshTokenId>().Where(x => x.RefreshTokenValue == refreshToken).FirstOrDefaultAsync();

if (refreshTokenObj != null)

{

dbContext.Remove(refreshTokenObj);

return await dbContext.SaveChangesAsync();

}

return 0;

}

public async Task<int> UpdateRefreshTokenOnRefresh(string oldtoken, string newToken, DateTime expiryTime)

{

var refreshTokenObj = await dbContext.Set<RefreshTokenId>().Where(x => x.RefreshTokenValue == oldtoken).FirstOrDefaultAsync();

if (refreshTokenObj != null)

{

refreshTokenObj.RefreshTokenValue = newToken;

refreshTokenObj.ExpiryTime = expiryTime;

dbContext.Set<RefreshTokenId>().Update(refreshTokenObj);

return await dbContext.SaveChangesAsync();

}

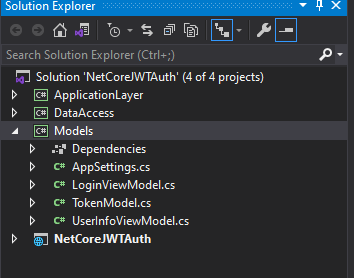
return 0;

}

}

================================================================================

1. Create **Models ->** .net core class library into solution



=================================================================================

1. Add **Model** classes in **Models** Project as below

public class AppSettings

{

public string SecretKey { get; set; }

}

public class LoginViewModel

{

public string UserName { get; set; }

public string Password { get; set; }

}

public class TokenModel

{

public string accessToken { get; set; }

public string refreshToken { get; set; }

public DateTime accessTokenExpiryTime { get; set; }

}

public class UserInfoViewModel

{

public int UserId { get; set; }

public string UserName { get; set; }

public string Password { get; set; }

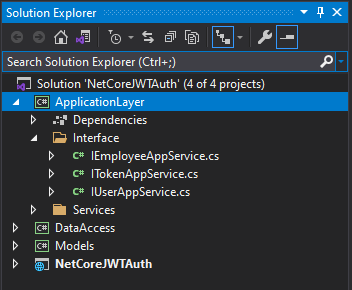
public string FirstName { get; set; }

public string LastName { get; set; }

}

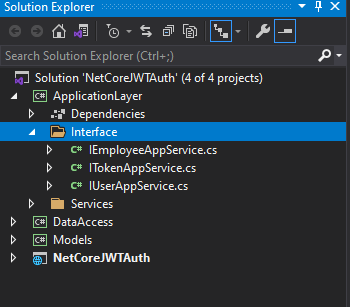
==================================================================================

1. Create **ApplicationLayer ->** .net core class library into solution



====================================================================================

1. Create **Interface** folder into **ApplicationLayer** project and add below interface into this folder-



public interface IEmployeeAppService

{

Task<IEnumerable<Employee>> GetEmployeeList();

Task<IEnumerable<Employee>> GetEmployeeById(Expression<Func<Employee, bool>> predicate);

Employee GetEmployeeById(int id);

Task<int> AddEmployee(Employee employee);

Task<int> UpdateEmployee(Employee employee);

Task<int> DeleteEmployee(int id);

}

==================================================================================

public interface ITokenAppService

{

Task<TokenModel> Login(string userName, string password);

Task<TokenModel> CreateTokenOnRefreshToken(string refreshToken);

Task<int> RemoveRefreshToken(string refreshToken);

}

=================================================================================

public interface IUserAppService

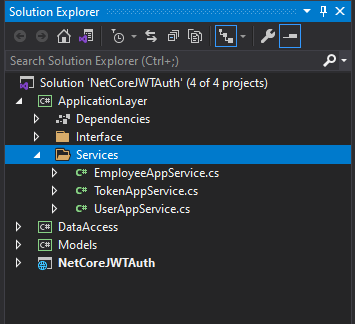
{

Task<IEnumerable<UserInfo>> GetUserDetailsByExpression(Expression<Func<UserInfo, bool>> predicate);

}

===============================================================================

1. Create **Services** folder into **Application** class library and add below service into this folder-



public class EmployeeAppService : IEmployeeAppService

{

public readonly IRepository<Employee> \_employeeGenericRepository;

public EmployeeAppService(IRepository<Employee> genericRepository)

{

\_employeeGenericRepository = genericRepository;

}

public async Task<IEnumerable<Employee>> GetEmployeeList()

{

var result = await \_employeeGenericRepository.Get().ConfigureAwait(false);

if (result == null)

{

throw new Exception("error");

}

return result;

}

public async Task<IEnumerable<Employee>> GetEmployeeById(Expression<Func<Employee, bool>> predicate)

{

var result = await \_employeeGenericRepository.Get(predicate).ConfigureAwait(false);

if (result == null)

{

throw new Exception("error");

}

return result;

}

public Employee GetEmployeeById(int id)

{

if (id <= 0)

{

throw new Exception("error");

}

var result = \_employeeGenericRepository.GetById(id);

if (result == null)

{

throw new Exception("error");

}

return result;

}

public async Task<int> AddEmployee(Employee employee)

{

if (employee == null)

{

throw new Exception("error");

}

await this.\_employeeGenericRepository.Add(employee);

var result = await \_employeeGenericRepository.SaveChangesAsync();

if (result <= 0)

{

throw new Exception("error");

}

return result;

}

public async Task<int> UpdateEmployee(Employee employee)

{

if (employee.EmpId <= 0)

{

throw new Exception("error");

}

this.\_employeeGenericRepository.Update(employee);

var result = await \_employeeGenericRepository.SaveChangesAsync();

if (result <= 0)

{

throw new Exception("error");

}

return result;

}

public async Task<int> DeleteEmployee(int id)

{

if (id <= 0)

{

throw new Exception("error");

}

var employee = \_employeeGenericRepository.GetById(id);

if (employee == null)

{

throw new Exception("error");

}

\_employeeGenericRepository.Delete(employee);

var result = await \_employeeGenericRepository.SaveChangesAsync();

if (result <= 0)

{

throw new Exception("error");

}

return result;

}

}

=============================================================================

public class TokenAppService : ITokenAppService

{

private readonly AppSettings \_appSettings;

private readonly IUserAppService \_userAppService;

private readonly ITokenRepository \_tokenRepository;

public TokenAppService(IOptions<AppSettings> appSettings,

ITokenRepository tokenRepository,

IUserAppService userAppService)

{

\_appSettings = appSettings.Value;

\_tokenRepository = tokenRepository;

\_userAppService = userAppService;

}

public async Task<TokenModel> Login(string userName, string password)

{

var userList = await \_userAppService.GetUserDetailsByExpression(x => x.UserName == userName);

var user = userList.FirstOrDefault();

if (user == null)

{

throw new Exception("error");

}

var token = GetTokenModel(user.UserId);

DateTime refreshTokenExpirytime = DateTime.UtcNow.AddHours(2);

var result = await \_tokenRepository.CreateRefreshTokenOnLogin(token.refreshToken, refreshTokenExpirytime, user.UserId).ConfigureAwait(false);

if (result > 0)

{

return token;

}

return null;

}

public async Task<TokenModel> CreateTokenOnRefreshToken(string refreshToken)

{

var oldRefreshToken = refreshToken;

var user = await \_tokenRepository.GetUserFromRefreshToken(oldRefreshToken).ConfigureAwait(false);

if (user != null)

{

var token = GetTokenModel(user.UserId);

DateTime refreshTokenExpiryTime = DateTime.UtcNow.AddHours(2);

//Update refresh token

var result = await \_tokenRepository.UpdateRefreshTokenOnRefresh(oldRefreshToken, token.refreshToken, refreshTokenExpiryTime).ConfigureAwait(false);

if (result > 0)

{

return token;

}

else

{

return null;

}

}

return null;

}

public async Task<int> RemoveRefreshToken(string refreshToken)

{

var result = await \_tokenRepository.RemoveRefreshToken(refreshToken);

if(result > 0)

{

return result;

}

else

{

return 0;

}

}

private TokenModel GetTokenModel(int userId)

{

string refreshtoken = Guid.NewGuid().ToString();

DateTime accessTokenExpirtTime = DateTime.UtcNow.AddDays(1);

var token = new TokenModel

{

accessToken = GenerateJwtToken(userId),

refreshToken = refreshtoken,

accessTokenExpiryTime= accessTokenExpirtTime

};

return token;

}

private string GenerateJwtToken(int userId)

{

var tokenHandler = new JwtSecurityTokenHandler();

var securityKey = Encoding.ASCII.GetBytes(\_appSettings.SecretKey);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

//new Claim(ClaimTypes.Name, userId.ToString())

new Claim("id", userId.ToString())

}),

Expires = DateTime.UtcNow.AddHours(1),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(securityKey), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

var finaltoken = tokenHandler.WriteToken(token);

return finaltoken;

}

}

=================================================================================

public class UserAppService: IUserAppService

{

public readonly IRepository<UserInfo> \_userGenericRepository;

public UserAppService(IRepository<UserInfo> genericRepository)

{

\_userGenericRepository = genericRepository;

}

public async Task<IEnumerable<UserInfo>> GetUserDetailsByExpression(Expression<Func<UserInfo, bool>> predicate)

{

var result = await \_userGenericRepository.Get(predicate);

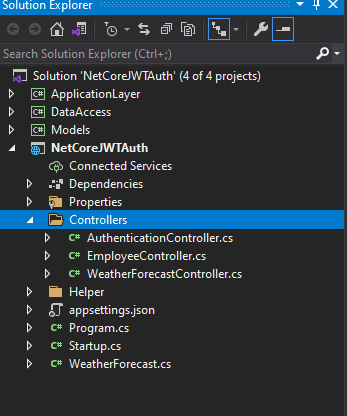
return result;

}

}

===============================================================================

1. Open **.Net core project** which we have created **in step 1**, Add **EmployeeController.cs and AuthorizationController.cs**



**AuthorizationController.cs**

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

[ApiController]

public class AuthenticationController : ControllerBase

{

private readonly ITokenAppService \_tokenAppService;

public AuthenticationController(ITokenAppService tokenAppService)

{

\_tokenAppService = tokenAppService;

}

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] LoginViewModel loginViewModel)

{

var token = await \_tokenAppService.Login(loginViewModel.UserName, loginViewModel.Password);

return Ok(token);

}

}

===============================================================================

**EmployeeController.cs**

**[Authorize]**

**[EnableCors("OIPAPolicy")]**

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

[ApiController]

public class EmployeeController : ControllerBase

{

public readonly IEmployeeAppService \_employeeService;

public EmployeeController(IEmployeeAppService employeeService)

{

\_employeeService = employeeService;

}

[HttpGet("getEmployeeList")]

public async Task<IActionResult> Get()

{

var result = await \_employeeService.GetEmployeeList();

return Ok(result);

}

[HttpGet("getEmployeeById/{id}")]

public async Task<IActionResult> GetEmployeeById(int id)

{

var result = await \_employeeService.GetEmployeeById(x => x.EmpId == id);

return Ok(result);

}

[HttpPost("saveEmployee")]

public async Task<IActionResult> SaveEmployee([FromBody] Employee employee)

{

var result = await \_employeeService.AddEmployee(employee);

return Ok(result);

}

[HttpPut("updateEmployee")]

public async Task<IActionResult> UpdateEmployee([FromBody] Employee employee)

{

var result = await \_employeeService.UpdateEmployee(employee);

return Ok(result);

}

[HttpDelete("deleteEmployee/{id}")]

public async Task<IActionResult> DeleteEmployee(int id)

{

var result = await \_employeeService.DeleteEmployee(id);

return Ok(result);

}

}

============================================================================

1. Open **startup.cs** file and inject dependency injection into **ConfigureServices** method. Also **add JWT Middler ware and Enable cors code**

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)

{

//Fetching appsettings Value

**var appSettingSections = Configuration.GetSection("JwtCongifuration");**

**services.Configure<AppSettings>(appSettingSections);**

**var appSettings = appSettingSections.Get<AppSettings>();**

services.AddControllers();

//Add Connection String

**services.AddDbContext<DemoDatabaseContext>(item => item.UseSqlServer**

**(Configuration.GetConnectionString("DemoDBConnection")));**

//Add Cors Policy

**services.AddCors(option => option.AddPolicy("OIPAPolicy",**

**builder =>**

**{**

**builder.AllowAnyOrigin()**

**.AllowAnyHeader()**

**.AllowAnyMethod();**

**})**

**);**

**//Inject Dependency Injection**

**services.AddScoped<IUnitOfWork, UnitOfWork>();**

**services.AddScoped(typeof(IRepository<>), typeof(Repository<>));**

**services.AddScoped<IEmployeeAppService, EmployeeAppService>();**

**services.AddScoped<IUserAppService, UserAppService>();**

**services.AddScoped<ITokenRepository, TokenRepository>();**

**services.AddScoped<ITokenAppService, TokenAppService>();**

}

// 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.UseRouting();

//Add JWT middleware to set user in context/session ->

//It will fetch user from token we are passing to request in web api

**app.UseMiddleware<JwtMiddleware>();**

**app.UseAuthorization();**

**app.UseCors("OIPAPolicy");**

app.UseHttpsRedirection();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

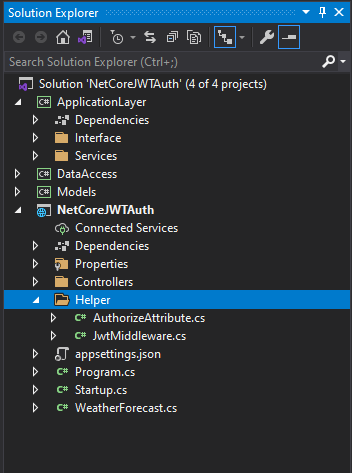
});

}

}

**===================================================================================**

1. Add **Helper** folder in .Net core project and **add JwtMiddleware.cs and AuthorizeAttribute.cs** file in **Helper** folder



using ApplicationLayer.Interface;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Options;

using Microsoft.IdentityModel.Tokens;

using Models;

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NetCoreJWTAuth.Helper

{

public class JwtMiddleware

{

private readonly RequestDelegate \_next;

private readonly AppSettings \_appSettings;

public JwtMiddleware(RequestDelegate next, IOptions<AppSettings> appSettings){

\_next = next;

\_appSettings = appSettings.Value;

}

public async Task Invoke(HttpContext context, IUserAppService userService)

{

var token = context.Request.Headers["Authorization"].FirstOrDefault()?.Split(" ").Last();

if (token != null)

await attachAccountToContext(context, userService, token);

await \_next(context);

}

private async Task attachAccountToContext(HttpContext context, IUserAppService userService, string token)

{

try

{

var tokenHandler = new JwtSecurityTokenHandler();

var securityKey = Encoding.ASCII.GetBytes(\_appSettings.SecretKey);

tokenHandler.ValidateToken(token, new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(securityKey),

ValidateIssuer = false,

ValidateAudience = false,

// set clockskew to zero so tokens expire exactly at token expiration time (instead of 5 minutes later)

ClockSkew = TimeSpan.Zero

}, out SecurityToken validatedToken);

var jwtToken = (JwtSecurityToken)validatedToken;

var userId = jwtToken.Claims.First(x => x.Type == "id").Value;

if (userId != null)

{

// attach account to context on successful jwt validation

var userData = await userService.GetUserDetailsByExpression(x => x.UserId == int.Parse(userId));

context.Items["User"] = userData.FirstOrDefault();

}

}

catch (Exception ex)

{

throw new Exception(ex.ToString());

}

}

}

}

================================================================================

**Does not add code in namespace name in authorizationAttribute class, otherwise AuthorizeAttribute does not gets call**

**namespace NetCoreJWTAuth.Helper**

**{}**

Direclty write the code as below

using DataAccess.Models;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]

public class AuthorizeAttribute : Attribute, IAuthorizationFilter

{

public void OnAuthorization(AuthorizationFilterContext context)

{

var account = (UserInfo)context.HttpContext.Items["User"];

if (account == null)

{

// not logged in

context.Result = new JsonResult(new { message = "Unauthorized" }) { StatusCode = StatusCodes.Status401Unauthorized };

}

}

}