آموزش Swagger :

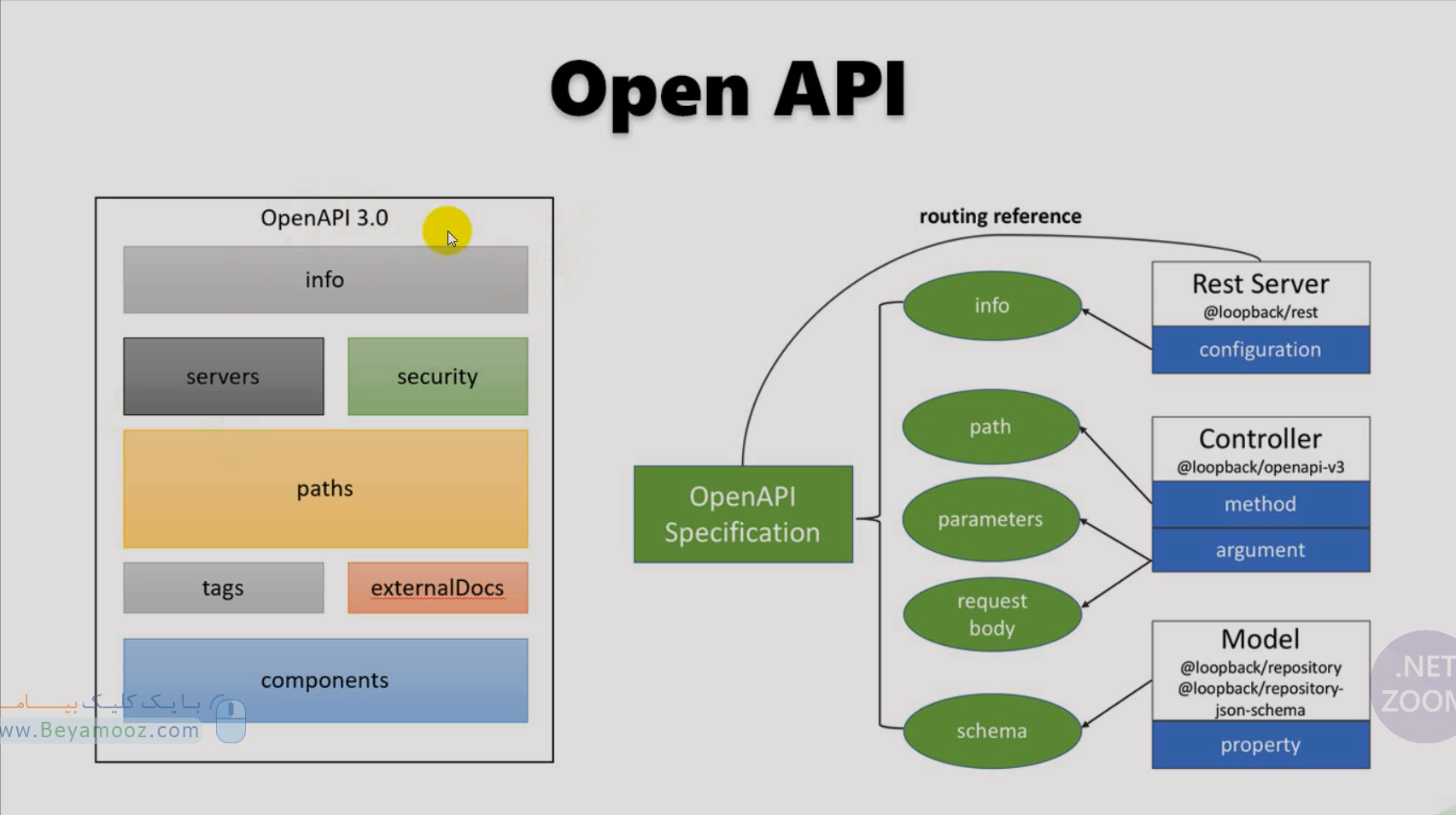
Swagger یک ابزار هست که از برای api هامون یک ui ساده و به همراه داکیومنت درست میکنه.

OpenApi : یک قرارداده که Swagger یک پیاده سازی از این قرارداد هست. که به ما میگه ما باید خروجی اکشن هامون و کنترلرهامون به همراه تمامی مشخصاتشون ( مثل اینکه چه پارامترهای ورودی دارند یا اسم اکشن یا چه خروجی و تمامی اطلاعات کامل api هامون ) به صورت یک قالب Json ارائه کنیم.

و swagger که یک ابزار هست میاد از روی این خروجی استاندارد که در قالب فرمت json هست یک ظاهری رو تهیه میکنه.

نکته : به جز swagger ابزارهای دیگه ای هم هست که بر اساس openApi یک سری امکانات رو در اختیار ما میزارن.

طرز کار Open API :



داخل قسمت path میتونیم ببینیم چه کنترلرها و اکشن هایی در پروژه داریم و براساس چه url هایی در دسترس هست.

قسمت parameters مشخص میکنه که هر اکشن ما چه پارامترهایی رو میگیره. مثلا بادی یا فرم یا کوئری استریک یا اینکه خروجی چیه

قسمت info یه سری اطلاعات راجب سرور اینکه ورژن api چنده، یا مشخصات دیگه ای از کل پروژه بک اند در این قسمت مشخص میشه.

قسمت Schema مشخصه میکنه پارامتر هایی که در ورودی اکشن هامون مشخص میکنیم ساختارش به چه صورت هست. مثلا معادلش میشه همون Dto هامون.

قسمت Security متد احراز هویت در پروژه ما چی هست، outh هست jwt هست یا روش های دیگه و swagger هم از روی همین یک ui درست میکنه.

Swagger میاد از طریق رفلکشن میاد تمام اکشن ها و کنترلرهای ما به همراه تمامی پارامترهای ورودی و خروجی شون و بقیه چیزها میخونه و اون ها رو تبدیل به فایل جی سان میکنه و از روی اون جی سان یک یو ای میسازه.

پس swagger از دو بخش تشکیل شده یکی از روی اطلاعات کنترلر ها یک فایل جی سان در یک قالب استاندارد تهیه میکنه. و بخش دیگه ui رو از روی اون قالب میسازه.

اضافه کردن Swagger به پروژه :

اول پکیچ ها رو اضافه میکنیم:

Swashbuckle.AspNetCore

در ساده ترین حالت همین پکیچ برای swagger کافیه.

Swashbuckle.AspNetCore.Annotations

Swashbuckle.AspNetCore.Fillters

این دو پکیج هم برای سفارشی سازی بیشتر استفاده میشه.

حالا باید یک سری سرویس رو به Startup اضافه کنیم و دو تا میدیل ویر رو اجرا کنیم.

سرویس ها:

services.AddSwaggerGen(option => {

option.SwaggerDoc(“v1”, new Info {Version = “v1”, Title = “MyApi” });

});

میدیل ویرها :

app.UseSwagger();

app.UseSwaggerUI(options => {

options.SwaggerEndpoint (“ /swagger/v1/swagger.json”, “Doc-V1”);

});

میدیل ویر اول کارش ساختن اون فایل جیسان هست که در قالب استاندارد openApi هست رو میسازه.

و میدیل ویر دوم کارش ساختن UI از روی جیسان هست. توی این میدیل ویر ادرس جیسانی که قراره ui از روش ساخته بشه رو بهش میدیم. الان یک Swagger ساده داریم بدون هیچ چیز خاصی.

اگر بعد از Url سایتمون این ادرسی که بالا نوشتیم رو وارد کنیم اون فایل جیسانی که گفتیم رو خواهیم دید.

حال سفارش سازی Swagger :

اول از همه کد ها رو در Startup نمینویسیم که شلوغ بشه و به صورت اکستنشن مینویسیم و در startup صداش میزنیم :

Public class SwaggerConfigurationExtensions {

Public static void AddSwagger(this IserviceCollection services) {

Assert.NotNull (services, nameof (services));

این دستور چک میکنه که سرویس خالی نباشه.

services.AddSwaggerGen( option => {

var xmlDocPath = Path.Combine (AppContext.BaseDirectory, “MyApi.xml”);

این دستور فایل xml رو به swagger معرفی میکنه.

options.IncludeXmlComments (xmlDocPath, true);

با این دستور میگیم داکیومنت هایی که در فایل بالا هست رو به ما در خروجی swagger نمایش بده.

Options.EnableAnnotations ();

این دستور باعث میشه بتونیم یک سری اتریبیوت رو بالای سر اکشن هامون بزاریم که طرز کار اون اکشن رو بیشتر مشخص کنیم.

options.DescribeAllEnumAsStrings ();

این باعث میشه اگر پروپرتی هامون از نوع enum باشه میاد اون رو به صورت string نمایش میده.

مثلا gender به صورت male و famale نمایش میده اگر این دستور نبود به صورت عدد 1و 2 نمایش میداد.

options.DescribeAllParametersInCamelCase ();

این دستور نام پارامترهای ورودی رو تبدیل به کمل کیس میکنه.

options.DescribeStringEnumsInCamelCase ();

این دستور enum هامون که به صورت string نمایش میده به کمل کیس تبدیل میکنه.

options.IgnoreObsoleteActions ();

این دستور اکشن های که بالای سرشون اتریبیوت obsolete داره رو ignore کن و نمایشش نده.چ

options.IgnoreObsoleteProperties ();

این هم با دستور بالایی هست.

options.SwaggerDoc (“v1”, new Info {Version =”v1”, Title =” API V1” });

این دستور برای ساخت ورژن درست میشه.

option.SwaggerDoc(“v1”, new Info {Version = “V1”, Title =”MyApi” });

options.OperationFilter <RemoveVersionParameters> ();

این میاد پارامتر ورژن رو که باید داخل هر اکشن دستی وارد میکردیم اون رو حذف میکنه.

options.DocumentFilter <SetVersionInPaths> ();

این میاد میگه به جای اینکه بری ورژن رو از تسک باکس بخونی باید اون رو از ورژن داکیومنتمون بخون همون داکیومنتی که وقتی داشتیم ورژن رو میساختیم مشخص کردیم.

options.DocInclusionPredicate ((docName, apiDesc) => {

if (!apiDesc.TryGetMethodInfo (out MethodInfo methodInfo)) return false;

var versions = methodInfo.DeclaringType

.GetCustomAttributes <ApiVersionAttribute> (true)

.SelectMany (attr => attr.Versions);

Return versions.Any (v => $”v{v.ToString()}” == docName );

});

این میگه اون v1 رو از url هم حذف کن. قرار نیست توی url هم ببینیمش. این کار هم با رفلکشن انجام میده میگیم بیا هر اکشنی که دریافت میکنی رو اتریبیوت هاش رو بگیر بعد اگر ورژنش رو دربیار اگر ورژنش مطابقت داشت با ورژنمون نمایشش نده

});

}

یک مد هم برای میدیل ویر ها مینویسیم :

Public static void UseSwaggerAndUI (this IapplicationBuilder app) {

app.UseSwagger ();

app.UseSwaggerUI (options => {

options.SwaggerEndpoing(“/swagger/v1/swagger.json”, “Doc-V1”);

});

}

داکیومنت نویسی برای اکشن ها و نمایش در swagger :

ما میتونیم بالای هر اکشنی یک داکیومنت مخصوص به اون اکشن بنویسیم و با Swagger اون رو نمایش بدیم برای این اینکار:

اول بالای هر اکشنی که خاصتیم با نوشتن /// و زدن دکمه tab حالت داکیومنت نویسی بالای سر اکشن فعال میشه و ما توضیحات لازم رو مینویسیم، مثلا بالای اکشنی که توکن جنریت میکنه :

/// <summary>

/// This method generate JWT Token

/// </summary>

/// <param name =”username”> The UserName of User </param>

/// <param name=”password”> The Password of User </param>

/// <param name=”cancellation Token”></param>

/// <returns></returns>

حالا روی پروژه کلیک راست میکنیم ( لایه ای که api ها داخلش نوشته شده ) وارد Propertise میشیم وارد تب build میشیم و قسمت XML documentation file رو فعال میکنیم.

این باعث میشه یک فایل xml ای ساخته بشه که شامل داکیومنت هایی هست که ما ایجاد کردیم.

و آدرس مخصوص خودمون رو هم وارد میکنیم. و build رو که بزنیم در این ادرس یک فایل xml درست شده که داکیومنتی که نوشتی رو داخلش میتونیم ببینم.

حالا لازمه که این فایل رو به Swagger معرفی کنیم. و اگر پروژه رو اجرا کنیم این داکیومنت رو خواهیم دید.

اگر بخایم ورژنیگ رو به پروژه اضافه کنیم اول از همه باید ورژن رو بالای کنترلر اضافه کنیم:

[ApiVersion(“1”)]

[Route (“api/v{version:apiVersion}/[controller]”)]

و ورژنینگ رو داخل سرویس کالکشن اضافه کنیم:

Public static void AddCustomApiVersioning (this IServiceCollection services) {

services.AddApiVersioning (options => {

options.AssumeDefaultVersionWhenUnspecified = true;

options.DefaultApiVersion = new ApiVersion (1, 0);

options.ReportApiVersions = true;

ApiVersion.TryParse (“1.0”, out var version10);

ApiVersion.TryParse (“1”, out var version1);

Var a = version10 == version1;

});

حالا اگر پروژه رو اجرا کنیم دقیقا چیزی که داخل روت بالای کنترلر نوشتیم جلوی اکشن هامون نوشته شده و باید ورژن رو هم به اکشن هامون بدیم.

حالا میخایم ورژن اتوماتیک بشه و نخایم داخل اکشن هامون ورژن رو پاس بدیم باید داخل سرویس swagger یک سری دستورات اضافه کنیم.

جلسه دوم Swagger :

اضافه کردن jwt به swagger :

ابتدا باید به option مربوط به AddSwaggerGen یک تعریف امنیتی اضافه کنیم :

option.AddSecurityDefinition (“Bearer”, new ApiKeyScheme {

Description = “JWT Auhorization header using the Bearer scheme, Example: \ “Authorization : Bearer {token}\””

Name = “Authorization”,

In = “header “ });

تعریف امنیتی یعنی اینکه سیستم امنیتی ما که به شکلی هست ( کوکی، jwt ، oauth ) اون رو باید اول تعریفش کنیم.

پارامتر اول نام هست که Bearer گذاشتیم. و بعد نوع سیستم احراز هویت رو مشخص میکنیم.

گفتیم که باید توی هدر ارسال بشه و نامش هم authorization هست. اگر الان پروژه رو اجرا کنیم یک دکمه authorization به swagger اضافه شده.

option.AddSecurityRequirement (new Dictionary <string, IEnumrable <string >> {

{“Bearer “, new string [] {}} });

اینجا یک نیاز امنیتی رو تعریف میکنیم. میگیم که همه Api نیاز امنیتی رو به صورت همگانی ست میکنیم یعنی به ازای همه اکشن ها احراز هویتشون باید به سیستم امنیتی که تعریف کردیم متصل باشه.

الان اگر پروژه رو اجرا کنیم همه اکشن ها به سیستم امنیتی متصل هستند و میدونن که باید به چه روشی احراز هویت بشن و یک ایکون قفل هم کنار هر اکشنی ظاهر میشه.

حالا اگر اکشنی که اتریبیتوت الوانینیموس نداره رو فراخوانی کنیم خطای احراز هویت بهمون میده.

حالا اکشن هایی که اتریبیوت الونینیموس هم دارند اون علامت قفل رو کنار خودشون دارند و باید این رو اصلاح کنیم: برای این کار دستور بالا رو پاک میکنیم و دستور زیر رو به جاش مینویسیم :

option.OperationFilter <UnauthorizedResponsesOpeationFilter> (false, “Bearer”);

این میاد قفل رو فقط به اکشن هایی که نیاز به احراز هویت دارند میده و فقط اون اکشن ها رو به سیستم احراز هویت متصل میکنه.

کلاس UnauthorizedResponsesOptinsFilter از IOpeationFilter ارث بری کرده که این کلاس والد یک متد داره به نام Apply که رو روی api ای که توسط swagger شناسایی میشه میاد و یک بار اعمال میشه.

بعد داخل این متد میگیم تمام اتریبیوت هایی که اکشن داره رو در بیار.

بعد گفتیم ایا توی این اتریبیوت ها فیلتر به نام AllowAnonymousFolter هست یا نه. اگر بود هیچ سفارش سازی خاصی نمیخاد روش انجام بدیم.

بعد بیا ببین ایا AuthorizeFilter روش اعمال شده یا نه، یعنی همون احراز هویت. اگر اعمال نشده بود نیازی به احراز هویت نداره ولی اگر اعمال شده بود بیا احراز هویت رو به اکشن وصل کن.

اضافه کردن مکانیزیم احراز هویت oauth :

در حالت عادی برای احراز هویت ما ابتدا باید یوز و پس خود را وارد کنیم و توکن را بگیرم، بعد دکمه احراز هویت را بزنیم و توکن خود را وارد کنیم تا بتوانیم از اکشن هایی که نیاز به احراز هویت دارند استفاده کنیم.

یک روش دیگر که میتوانیم پیاده سازی کنیم این است که یک مودال باز شود یوز و پس خود را وارد کنیم بعد اتوماتیک توکن ساخته میشد، ذخیره میشده و در درخواست های بعدی ارسال میکرد. برای این کار از oauth استفاده میکنیم.

یک استاندارد هست که نوع توکن از جنس jwt هست. که مشخص میکنه طبق چه استانداردی ورودی ها گرفته بشه و چه خروجی هایی داده بشه.

ابتدا تعریف امنیتی که برای AddSwaggerGen نوشتیم رو حذف میکنیم و دستورات زیر رو اضافه میکنیم :

option.AddSecurityDefinition (“Bearer”, new OAuth25scheme {

Flow = “password”,

TokenUrl = “https://localhost:5001/api/v1/users/Token”,

});

اول یک اسم از ما میخواد که bearer گذاشتیم ( دلبخواه هست)

استفاده از oauth روشهای مختلفی داره که ما از روش password استفاده میکنیم. این فلو به این صورت هست که یک یوز و پس رو در ورودی ازمون میگیره و در خروجی یک متن خاص بهمون میده.

قسمت بعدی باید اکشنی که توکن رو جنریت میکنه مشخص کنیم. اگر پروژه رو اجرا کنیم و دکمه احراز هویت رو بزنیم دوتا قسمت اضافه شده که از ما یوز و پس رو میخاد. اگر پیاده سازی کامل باشه وقتی که یوز و پس رو وارد میکنیم و authorize رو میزنیم این اطلاعات رو میفرسته سمت سرور توکن صادر میشه، توکن میاد توی swagger ذخیره میشه و میتونیم راحت اکشن هامون استفاده کنیم.

وقتی که دکمه swagger رو میزنیم به صورت یک فرم یک سری اطلاعات ارسال میشه سمت سرور :

Grant\_type : password این نوع توکنی که استفاده شده رو مشخص میکنه

Username : admin یوزرنیمی که ارسال کردیم

Password : 123445 پسوردی که ارسال کردیم

Client\_id : clientId ممکنه ما چندین کلاینت داشته باشیم یکی که همین swagger هست یکی که صفحه وب سایت هست که با فرانت سروکار داره و ممکنه چندین کلاینت دیگه داشته باشیم. برای اینکه بفهمیم این توکن از کدام کلاینت ارسال شده از این فیلد متوجه میشه.

Client\_secret : clientSecret وقتی که کاربر به عنوان یک کلاینت میخواد که احراز هویت بشه برای اینکه مطمعن بشیم از طرف همون کلاینتی که وارد کرده میخواد احراز هویت بشه این فیلد رو هم باید وارد کنه.

اصلاح متد توکن برای پیاده سازی این روش :

[Route (“[action]”), HttpPost, HttpGet]

[AllowAnonymous]

Public virtual async Task<string> Token ([FromForm] TokenRequest tokenRequest, CancellationToke cancellation) {

If (!tokenRequest.grant\_type.Equals (“password” , stringComparison.OrdinalIgnoreCase )) {

Throw new notFound ();

و بقیه ساخت توکن به صورت قبل