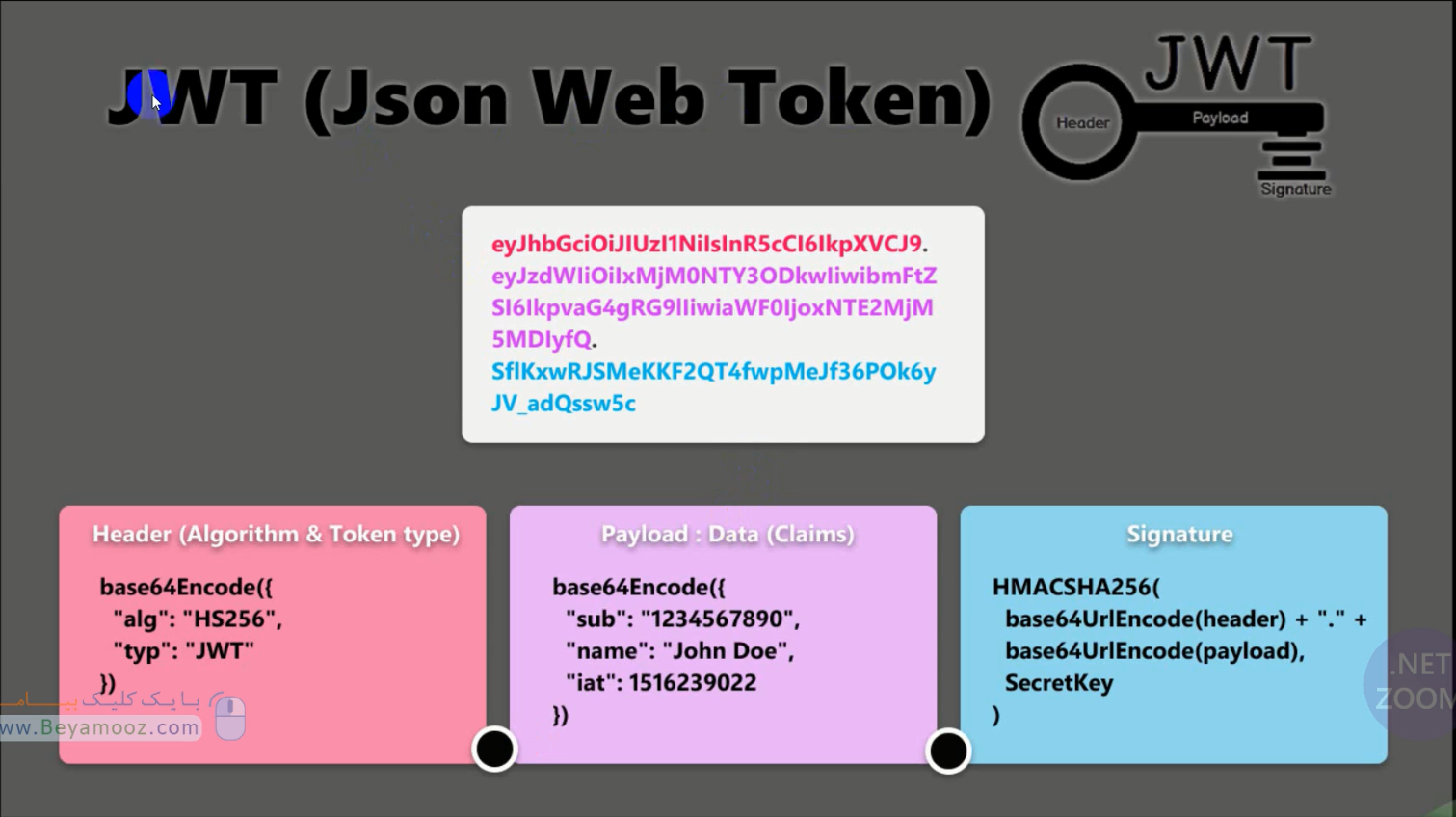
JWT :



Jwt در واقع یک توکن رمز نگاری شده است که دیتای داخل اون و در واقع مشخصاتی که برای احراز هویت کاربران لازمه به صورت جیسان ذخیره شده. که در تصویر بالا نمایش داده شده که شامل سه قسمت است. قسمت اول که قرمز رنگ هست بهش میگن header قسمت وسط که با رنگ بنفش مشخص شده و بهش میگن Payload که شامل تمام اطلاعات لازم برای احراز هویت هست و قسمت اخر Signature هست و در واقع امضای توکن هست و قسمت امنیتی توکن هست.

قسمت اول هم شامل مشخصات توکن هست اینکه چطور رمزنگاری شده با چه الگوریتی و ... و این سه قسمت با نقطه به هم متصل شده اند.

قسمت اول : یک مثل جیسان است که نشون میده که الگوریتمی که این توکن باهاش رمزنگاری شده چیه که الان HS256 هست. و تایپش هم JWT هست. بعد این قسمت توسط base64 رمزنگاری شده و شده متنی که در توکن هست.

قسمت دوم : یک متن جیسانی هست که شامل تمام اطلاعات مورد نیاز برای احراز هویت کاربر هست اینکه ایدیش چیه نامش چیه و یه سری چیزای دیگه که با الگوریم base64 رمزنگاری شده و شده متنی که در توکن هست.

Claims : یک سری کلید و ولیو هست، کلید مقدار مثلا کلید نام و مقدار jwt.

معنی ترجمه ای claim میشه ادعا یعنی این کاربر چه ادعاهایی داره و شامل تمام اطلاعات لازم برای احراز هویت هست.

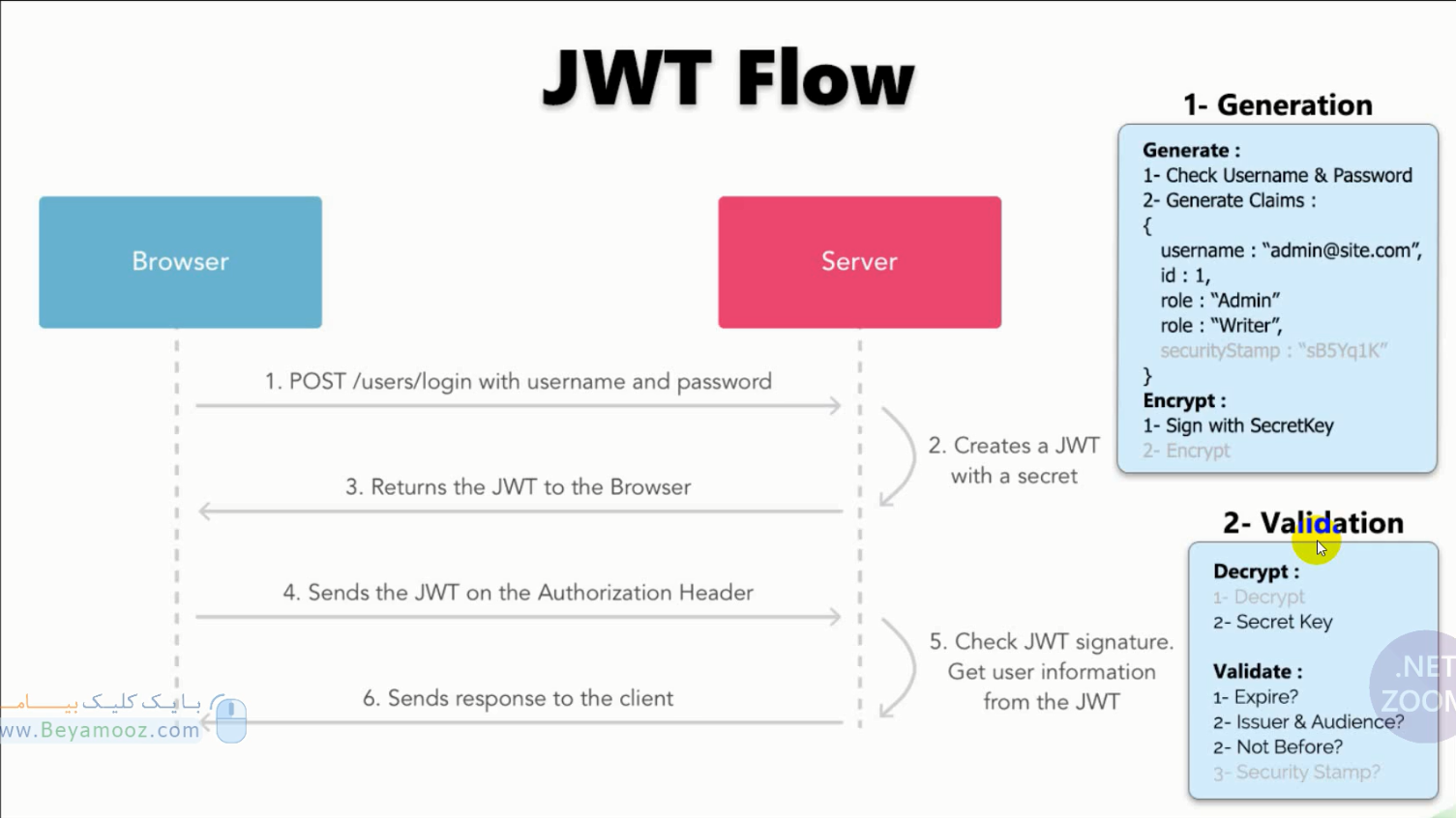
الگوریتم base 64 : یک الگوریتم استاندارد برای رمزگذاری و رمزگشایی هست. این رمزگذاری برای ایجاد امنیت نیست و با همین الگوریتم میتوانیم به راحتی توکن را به حالت اول در اورد.

قسمت سوم: این قسمت امضای توکن هست که میاد قسمت اول رو تبدیل به base64 میکنه بعد با نقطه جمع میکنه بعد با base64 قسمت دوم جمع میکنه و نهایتا با یک کلید مخفی یک hash ازش میسازه. که این الگورتیم HMACSH256 هست.

این امضا باعث میشه هر تغییری در قسمت اول و دوم ایجاد کردیم دیگه امضا با توکن نخونه و توکن نامعتبر بشه.

کل این قضیه باعث میشه یک هکر بیرونی نتونه محتوای توکن رو تغییر بده.

کلید سمت راست تصویر یک نمای کلی از jwt و بخش های ان در مقایسه با کلید واقعی به ما نشون میده.



تصویر بالا فرایند احراز هویت توسط jwt رو نشون میده.

تمامی روش های احراز هویت زیرمجموعه دوروش هستند :

یک Cookie Base Authentication : در این روش همون محتویات کاربر که در Claims ذخیره میشد توی کوکی ذخیره میشن.

دو Token Base Authentication : که jwt هم برهمین اساس است یک توکن درست میکنیم که کلاینت هربار توی هر درخواستش اون توکن رو برای ما ارسال میکنه و ما سمت سرور اون توکن رو دریافت میکنیم احراز هویت میکنیم و میبینیم ایا باید به این کاربر اجازه ی عملیات بدیم یا نه.

و تنها تفاوتشون در نحوه ارسال هست. یکی کوکی رو ارسال میکنه یکی توکن رو ارسال میکنه.

Authentication : یعنی احراز هویت و بحث لاگین کاربر رو بهش اشاره میکنه زمانی که کاربری میاد و لاگین میکنه همینکه میبینیم کی هست مشخصاتش چیه یوزرش چیه ایدیش چیه اینو بهش میگن احراز هویت.

Autorization : به تعیین سطح دسترسی هست. یعنی اینکه کاربر میدونیم کیه ولی میخایم تعیین سطح دسترسی کنیم ببینیم ایا این اکشن خاص رو که مثلا درخواست کرده ایا اجازه دسترسی بهش داره یا نه.

فرایند احراز هویت :

کلا شامل دو بخش میشه یکی قسمت ساختن توکن و بخش دوم اعتبار سنجی اون توکن هست.

کاربر میاد داخل صفحه بروزر و وارد صفحه لاگین میشه یوزرنیم و پسورد رو وارد میکنه و توسط متد پست ارسال میکنه سمت سرور، بعد سمت سرور میایم کاربر رو پیدا میکنیم و میبینیم که ایا وجود داره یا نه اگر وجود نداشت که هیچی اگر وجود داشت میایم توکنمون رو براش میسازیم. و در مرحله بعد توکن رو پاس میدیم سمت کلاینت.

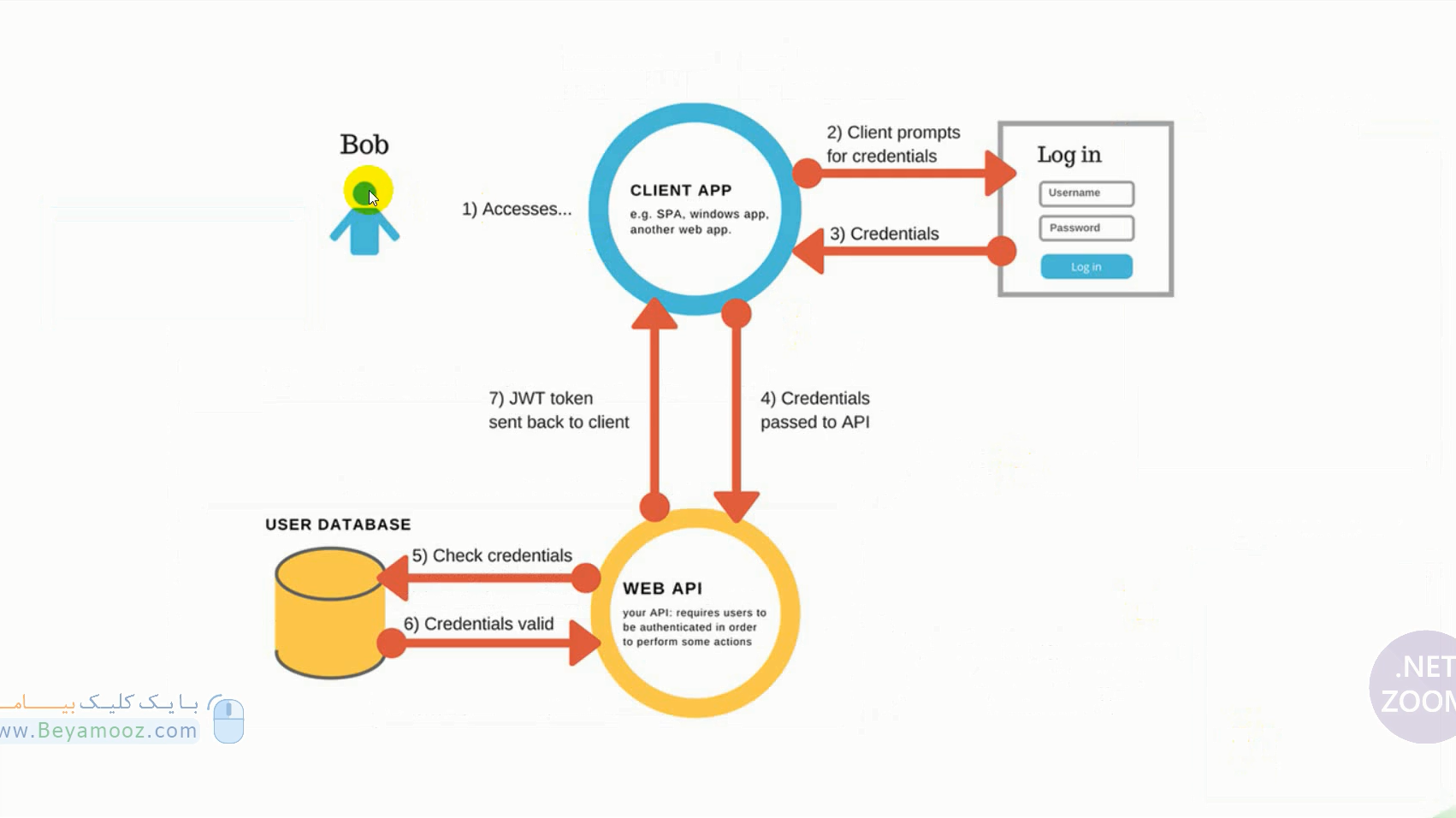
کاربر باید توکن رو سمت خودش نگه داره و با هر درخواستی اون توکن رو برای ما ارسال میکنه و ما سمت سرور بررسی میکنیم که اون کاربر کیه و اعتبار سنجی میکنیم و جواب رو بهش میدیم.

مراحل Generate :

1. یوزر نیم و پسورد رو میگیریم و چک میکنیم این کاربر کیه
2. کاربر رو اوردیم و میخایم Claims هاش رو بسازیم. مثل اینکه ایدیش چیه پسوردش چیه رولش چیه
3. و نهایتا توسط Secretkey که سمت سرور هست اون امضای مخصوص برای توکن رو درست میکنیم.

مراحل Validation :

1. توکن رو دریافت میکنیم و با Secretkey چکش میکنیم. یعنی سمت سرور میاد header و payload توسط اون Secretkey که داره یه بار هش میکنه و هش رو با این Secretkey مربوط به توکنی که ارسال شده مقایسه میکنه. اگر یکی بود که یعنی اوکیه اگر یکی نبود یعنی اینکه توکن دست کاری شده. این کار اتوماتیک انجام میشه.
2. توکن رو بررسی میکنیم که ایا این توکن منقضی شده یا نه.
3. میام Issuer و Audience رو بررسی میکنیم این دو مقدار داخل payload قرار میگیره و نشان دهنده اینه که چه کسی یا چه سروری این توکن رو ارسال کرده و برای چه نوع مصرف کننده ای این توکن ساخته شده.
4. NotBefore چک میکنیم. یعنی از چه زمانی به بعد میتونیم از این توکن استفاده کنیم. مثلا از 5 ساعت دیگه به بعد میتونی از این توکن استفاده کنی.



تو تصویر بالا کاربر ما اسمش باب هست وارد اپلیکیشنش میشه که هرچیزی میتونه باشه یک صفحه لاگین بهش نشون میدیم. بعد از وارد کردن اطلاعات ارسال میشه به سرور با دیتابیس چک میکنیم اگر که اون کاربر وجود داشت اون کاربر رو واکشی میکنیم و توکنش رو میسازیم و برمیگردونیم سمت کاربر.

پیاده سازی jwt :

پکیج system .IdentityModel.Tokens.Jwt رو نصب میکنیم.

قسمت Generate :

Public Class JwtService

{

Public string Generate (User user)

{

Var secretKey = Encoding.UTF8.GetBytes (“MySecretKey”);

Var SigningCredentials = new SigningCredentials (new

SymmetricSecurityKey (secretKey), SecurityAlgorithms.HmacSha256Signature);

Var claims = GetClaims (user);

Var descriptor = new SecurityTokenDescriptor

{

Issuer = “MyWebsite” ,

Audience = “MyWebsite” ,

IssuedAt = DateTime.Now ,

NotBefore = DateTime.Now.AddMinutes (5) ,

Expires = DateTime.Now.AddHours (1) ,

SigningCredentials = signingCredentials,

Subject = new ClaimsIdentity (claims );

};

Var tokenHandler = new JwtSecurityTokenHandler();

Var securityToken = tokenHandler.CreateToken (descriptor);

Var jwt = tokenHandler.WriteToken (securityToken);

Return jwt;

}

Private Ienumerable <Claim) GetClaims (User user)

{

Var list = new List <Claim>

{

New Claim (ClaimTypes.Name, user.UserName),

New Claim (ClaimTypes.NameIdentifier, User.Id.ToString())

};

Foreach( var role in user.Roles){

List.Add (new Claim (ClaimTypes.Role, role.Name));

}

Descriptor رو هرجا دیدیم یعنی توضیح دهنده مشخصات همون چیزی که ازش استفاده میکنیم.

پروپرتی Issuer مشخص میکنه که صادر کننده توکن کی باید باشه. که هر متنی میتونیم قرار بدیم.

پروپرتی Audience به معنی مصرف کننده هست و اینکه مصرف کننده این توکن قرار کی باشه.

پروپرتی IssuedAt به معنی اینکه این توکن چه زمانی صادر شده.

پروپرتی NotBefore از چه زمانی به بعد قابل توکن قابل استفاده هست.

پروپرتی Expires تاریخ انقضای توکن هست.

پروپرتی SigningCredentials با این پروپرتی امضای مورد نیاز توکن رو میسازیم. که باید امضا رو درست کنیم و بدیم به این پروپرتی.

متغیر signingCredentials یک نمونه از SigningCredentials میسازه که این نمونه دوتا مقدار ورودی از ما میخاد. یک کلید ( از semmetricSecurityKey استفاده میکنیم که یک کلاس رمزنگاری متقارن برای ما به وجود میاره). و دو الگوریتم که ما از الگوریتم HmacSha256Signature برای رمز نگاری استفاده میکنیم.

نکته : SecretKey حتما باید بزرگتر مساوی 16 کارکتر باشه. اگر نباشه ارور میده بهمون.

پروپرتی Subject این در واقع همون Claims های کاربر هست که باید بهش بدیم.

مرحله بعد ساخت jwt از روی Descriptor هست.

حالا در جایی که میخایم ازش استفاده کنیم:

Public async Task <string> Token (string username, string password, CancellationToken cancellationToken) {

UserRepository.GetByUserNameAndPass (username, password, cancellationToken);

If (user == null ) { throw new BadRequestException(“user not found “); }

Var jwt = JwtService.Generate (user);

Return jwt;

اگر در مورد پیاده سازی Jwt در اینترنت سرچ کنیم نمونه کدهای مختلفی رو خواهیم دید و در واقع همه اونها یکی هست و سبکشون یکی هست و یه تفاوت هایی در هدر یا پیلود شاید داشته باشند.

و نکته بعدی این هست که وقتی توکن رو دیکد میکنیم ( مثلا در سایت jwt میتونیم این کار رو انجام بدیم) میبینیم که نام Claim ها تغییر کرده مثلا username به unique\_name تغییر نام پیدا کرده.

به این دلیل هست که کلاس JwtSecurityTokenHandler میاد نام توکن هایی که معادل jwt داره رو به اون نام ها تغییر میده.

برای اینکه این ترجمه اتفاق نیفته میتونیم دستور زیر رو قبل از تعریف توکن هندلر اضافه میکنیم:

JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear ();

JwtSecurityTokenHandler.DefaultMapInboundClaims = false;

JwtSecurityTokenHandler.DefaultOutboundClaimTypeMap.Clear ();

قسمت Validation :

اول یک کلاس برای کانفیک کردن jwtvalidation درست میکنیم :

Public static Class ServiceCollectionExtensions

{

Public static void AddJwtAuthentication (this IserviceCollection services) {

Services.AddAuthentication (options => {

Options.DefaultAuthenticationScheme = JwtBearerDefaults.AuthenticationScheme;

Option.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

Option.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;

}).AddJwtBearer ( options => {

ClockSkew = TimeSpan.Zero,

RequireSignedTokens = true,

ValidateIssuerSigningKey = true,

IssuerSigningKey = new symmetricSecurityKey (secretkey),

RequireExpirationTime = true,

ValidateLifetime = true,

ValidateAudience = true,

ValidAudience = “” ,

ValidateIssuer = true,

ValidIssuer = “” ,

};

Optoins.RequireHttpsMetadata = false;

Options.SaveToken = true;

Options.TokenValidationParameters = validationParameters; });

}

Shema : روش احراز هویت هست.

TokenValidationParameters یعنی توکن های لازم برای اعتبار سنجی توکن. که در بالا مقدار دهیش کردیم و در اخر بهش مقدار دادیم.

ClockSkew یک ترلانس زمانی برای توکن ما ایجاد میکنه که روی دوتا مقدار تاثیر داره یکی ExpireTime و NotBefore مقدار دیفالت این کلاک 5 دقیقه هست. اگر این رو صفر نکنیم و مقدار دیفالت رو داشته باشه حتی بعد از 5 دقیقه منقضی شدن هم به ما این اجازه رو میده که بتونیم از توکنمون استفاده کنیم. برای NotBefore هم به این صورت هست که اگر مثلا مقدارش روی 5 دقیقه باشه این توکن رو همین الان هم میتونیم استفاده کنیم و در واقع 5 دقیقه ClockSkew اون رو خنثی میکنه.

RequireSignedTokens میگه حتما توکن هایی که ارسال میشن حتما امضا داشته باشن.

ValidateIssuerSigningKey یعنی امضای توکن مورد اعتبار سنجی قرار بگیره حتما.

IssuerSigningKey امضای توکن رو چک میکنه.

RequireExpirationTime تاریخ انقضای توکن رو چک میکنه.

ValidateLifetime این هم با همین تاریخ انقضا مربوط هست.

ValidateAudience این دستور مصرف کننده توکن رو بررسی میکنه.

ValidAudience هم نام مصرف کننده رو وارد میکنیم.

validateIssuer صادر کننده توکن رو اعتبار سنجی میکنه.

validIssuer نام صادر کننده رو وارد میکنیم.

اعتبار سنجی توکن ما تقریبا انجام شده.

و در نهایا میدیل ویر رو اضافه میکنیم :

App.UseAuthentication ();

و در بالای اکشن هامون اگر اتریبیوت Authorize رو بزاریم فقط کسایی میتونن به اون اکشن دسترسی داشته باشن که احراز هویت شده باشند.

داخل اتریبیوت میتونیم role هایی که میتونن به این اکشن دسترسی داشته باشند رو مشخص کنیم.

[Authorize (Roles = “Write”)]

همچنین میتونیم این سطح دسترسی رو برای کنترلر بزاریم به جای اینکه برای همه اکشن ها بزاریم.

کافیه همین دستور رو بالای سر کنترلر بزاریم. حالا یک اکشن اگر لازم بود که سطح دسترسی نداشته باشه کافیه دستور زیر رو بالای سرش بنویسیم :

[AllowAnonymous]

همچنین میتونیم بگیم تمامی اکشن های پروژه ما به احراز هویت نیاز دارند مگر اینکه خلافش ثابت بشه.

خلافش با دستور بالا ثابت میشه. میریم داخل استارتاپ :

Services.AddMvc (Options => {

Options.Filters.Add (new AuthorizeFilter ());

}).SetCompatibilityVersion (CompatibilityVersion.Version\_2\_1);

این دستور باعث میشه احراز هویت رو برای تمامی اکشن های پروژه ما اعمال میکنه.

مثلا همه اکشن ها به احراز هویت نیاز دارند یکی از اکشن ها به role مدیر نیاز داره با همون اتریبیوتی که رول هم داره اون اکشن رو احراز هویت میکنیم.

حالا کدهای ما نیاز داره به صورت حرفه ای تر ویرایش کنیم :

یک سری اطلاعات ثابت هست که در جاهای مختلف تکرار شده این مقادیر استاتیک رو میبریم داخل appsetting:

ابتدا یک کلاس درست میکنیم به نام SiteSettings :

Public Class SiteSettings {

Public int ElmahPath {get; set; }

Public string WebSiteUrl {get; set; }

Public JwtSettings JwtSettings {get; set; }

Public IdentitySettings IdentitySettings {get; set; }

}

Public Class JwtSettings {

Public string SecretKey {get; set; }

Public string Issuer {get; set; }

Public string Audience {get; set; }

Public int NotBeforeTime {get; set; }

Public int ExpirationDay {get; set; }

}

حالا appsetting :

“SiteSettings” : {

“ElmahPath” = “/elmah-errors”;

“JwtSettings” : {

“SecretKey” : “mysecretkey123456789”,

“Issuer” : “mywebsite”,

“Audience” : “mywebsite”,

“NotBeforeTime” : 0,

“ExpirationDay” : 1 } }

حالا میخایم مقادیر رو بخونیم و بریزم داخل کلاس siteSetting :

داخل استارتاپ میاد شی Configuration رو در اختیار ما قرار میده که شامل تمام اطلاعات appsettings و زیرمجموعه هاش هست. که با صدا زدن configuration و سکشن مورد نظر در appsettings مقادیر رو فراخوانی کنیم.

Private readonly SiteSettings \_siteSetting;

Public IConfiguration Configuration {get; }

Public Startup (IConfiguration configuration) {

Configuration = configuration;

\_siteSetting = configuration.GetSection (nameof (SiteSettings)).Get <SiteSettings>(); }

حالا جاهایی که داریم مقادیر استاتک مقدار دهی میکنیم میام از مقادیر \_siteSetting استفاده میکنیم.

حالا در متد ConfigureServices :

Services.Configure <siteSettings> (configuration.GetSection (nameof (SiteSettings )));

این متد باعث میشه شی siteSettings رو توسط تزریق وابستگی توی سازنده های پروژه هامون دریافت کنیم.

مثلا در سرویس jwt به این شی نیاز داریم و میخایم ازش استفاده کنیم :

Private readonly SiteSettings \_settings;

Public JwtService (IOptionsSnapshot <SiteSettings> settings) {

\_settings = settings; }

Jwt در حالت عادی هیچ امنیتی در مقابل خونده شدن اطاعاتش نداره و به راحتی میتونیم محتواش رو رمزگاشی کنیم. میخایم بدنه توکنمون یک بار دیگه رمزگذاری بشه و برای این کار از EWT استفاده میکنیم.

برای اینکار وارد سرویس Jwt میشیم متد Generate اونجایی که داریم توکنمون رو ایجاد میکنیم رمزنگاری اضافه تر رو هم انجام بدیم.

Var encryptionKey = Encoding.UTF8.GetBytes (\_siteSetting.JwtSettings.Encryptkey);

Var encryptingCredentials = new EncryptingCredentials (new SymmetricSecurityKey (encuryptionkey), SecurityAlgorithms.Aes128KW, SecurityAlgoritms.Aes128CbcHmacSha256);

حالا باید پروپرتی Encryptkey رو به کلاس siteSetting اضافه کنیم و همچنین در appsettings هم باید این کلید رو اضافه کنیم و مقدار دهی کنیم.

این کلید حتما باید 16 کارکتر باشه نه کمتر نه بیشتر.

و در متغیر descriptor که مربوط به jwt هست میام فیلد EncryptingCredentials رو پر میکنیم.

حالا موقع validation هم باید یک بار توکن رو با همین کلید رمزگشایی کنیم برای این کار داخل متغیر validationParameters کد زیر رو اضافه میکنیم :

Var encryptionKey = Encoding.UTF8.GetBytes (jwtSettings.SecretKey);

TokenDecryptionKey = new SymmetricSecurityKey (encryptionKey);

و اگر توکنی که ایجاد شده رو مثلا در سایت jwt وارد کنیم دیگه نمیتونیم مقادیر رو بیرون بکشه.

نکته : هدر رمزگذاری نمیشه با ewt.

Security Stamp : مهر امنیتی

در jwt زمانی که میخایم Claim ها رو درست کنیم به دو روش میتونیم کار کنیم :

1. یک کلایم درست میکنیم که فقط userId داخلش قرار میگیره و در هر درخواستی که توکن رو داریم ارسال میکنیم میریم سمت دیتابیس و همه اطلاعات کاربر رو میاریم و بقیه ماجرا. خوبی این روش این است که در هر درخواست اخرین ویرایش های اطلاعات کاربر رو به دست میاریم. و بدی این روش این هست که تعداد تراکنش هایی که به دیتابیس میزنیم زیاد میشود. یک مزیت دیگر این روش دیتای کمتری در توکن رد و بدل میشه. چه زمانی از این روش استفاده میکنیم : یک این تعداد تراکنش هایی که به بانک باید بزنیم مشکلی برای پروژه ما ایجاد نکنه و پروژه رو کند نکنه و دوم اینکه به دیتابیس user هامون دسترسی داشته باشیم در پروژه.
2. یک کلایم درست کنیم که همه اطلاعات کاربرکه بعدا لازم میشه مثل نام، نام خانوادگی، نقش های کاربر و همه اینها رو در کلایم ثبت میکنیم و در درخواست هایی که توکن رو ارسال میکنیم دیگه به دیتابیس نمیزنیم و همه اطلاعات لازم رو داریم. به این روش SelfContent یا خودشامل.

مهر امنیتی : در واقع زمانی که داریم برای کاربر توکن رو صادر میکنیم یک مهر امنیتی مختص همون کاربر به توکن اضافه میکنیم. در قالب یک کلیم. حالا اگر تغییری در اطلاعات کاربر ایجاد بشه این مهر امنیتی تغییر میکنه و زمانیکه کاربر درخواستی داره مهرامنیتی که در توکن وجود داره با مهر امنیتی که برای کاربر در دیتابیس وجود داره مقایسه میکنیم اگر برابر بودند کاربر به درخواستش دسترسی داره.

برای این کار اول یک فیلد داخل جدول user درست میکنیم :

Public Guid SecurityStamp {get; set; }

و در کانستراکتور :

Public User() { SecurityStamp = Guid.NewGuid (); }

یک سرویس هم درست میکنیم برای update کردن SecurityStamp :

Public Task UpdateSecurityStamp (User user, CancellationToken cancellationToken ) {

user.SecurityStamp = Guid.NewGuid ();

return UpdateAsync (user, cancellationToken); }

در متد GetClaim :

private Ienumerable <Claim> GetClaims (User user) {

var securityStampClaimType =

new ClaimsIdentityOptions ().SecurityStampClaimType;

var list = new List <Claim> {

new Claim (securityStampClaimType, user.SecurityStamp.ToString ()) };

حالا موقع Validation : میریم داخل متد AddJwtAuthentication همون جایی که ولیدیش jwt پیاده سازی شده :

options.Events = new JwtBearerEvents {

OnAuthenticationFailed = context => {

If (context.Exception != null)

throw new AppException (ApiResultStatusCode.UnAuthorized, “Authentication failed” , Context.Exception, null);

Return Task.CompletedTask;

}

OnTokenValidated = async context => {

Var userService = context.HttpContext.RequestServices.GetRequiredService <IuserRepository> ();

Var claimsIdentity = context.Principal.Identity as ClaimsIdentity;

If ( claimsIdentity.Claims?.Any () != true)

context.Fail (“This token has no claims.”);

var securityStamp = claimsIdentity.FindFirstValue (

new ClaimsIdentityOptions ().SecurityStampClaim

if (!securityStamp.HasValue ())

context.Fail (“This token has no security stamp”);

var userId = claimsIdentity.GetUserId <int> ();

var user = await userService.GetByIdAsync (context.HttpContext.RequestAborted, userId);

if(user.SecurityStamp != Guid.Parse (secuityStamp))

context.Fail (“Token security stamp is not valid.”);

if(!user.IsActive)

context.Fail (“User is not active.”);

await userRepository.UpdateLastLoginDateAsync (user, context.HttpContext.RequestAborted);

}

OnMessageReceived = context => {

}

OnChallenge = context => {

If (context.AuthenticateFailure != null )

throw new AppException (ApiResultStatusCode.UnAuthorized, “Authenticate failure.”, context.AuthenticateFailure, null);

throw new AppException (ApiResultStatusCode.UnAuthorized, “You are unauthorized to access this resource.”); }

}

رخ داد OnMessageReceived زمانی یک که ریکوئست حاوی توکن به سمت سرور میاد اجرا میشه

رخ داد OnChallenge زمانی که یک اکشنی رو فراخوانی میکنیم که نیاز به احراز هویت داره ولی همراه با ریکوئست توکنی ارسال نکردیم اجرا میشه.

رخ داد OnTokenValidated زمانی که توکن اعتبار سنجی میشه اجرا میشه و میتونیم داخل این رخ داد سفارشی سازی های لازم رو انجام بدیم.

رخ داد OnAuthenticationFailed زمانی که به هر دلیلی توکن failed بشه این رخ داد اجرا میشه.

متد UpdateLastLoginDateAsync هم در ریپازیتوری user میسازیم که با این متد میتونیم اخرین ورود کاربر رو ثبت کنیم:

Public Task UpdateLastLoginDateAsync (User user, CancellationToken cancellationToken) {

user.LastLoginDate = DateTimeOffset.Now;

return UpdateAsync (user, cancellationToken); }