1.install Microsoft.AspNetCore.Authentication.JwtBearer

Validate

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(JwtBearer
Defaults.AuthenticationScheme, config => {

                config.TokenValidationParameters = new TokenValidationParameters
                {

                    ValidateIssuer = true,
                    ValidIssuer = configuration["Security:Tokens:Issuer"],
                    ValidateAudience = true,
                    ValidAudience = configuration["Security:Tokens:Audience"],
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["Security:Tokens:Key"])),

                };

            });
```

Issue

```
public string IssueCandidateJwtToken(Candidate candidate)
        {
            var claims = new[]
          {
                  new Claim(ClaimTypes.Name,candidate.FullName),
                  new Claim(ClaimTypes.Email,candidate.Email),
                  new Claim("Candidate","true")

              };

            var key = new

SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["Security:Tokens:Key"]));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
            var token = new
JwtSecurityToken(configuration["Security:Tokens:Issuer"],configuration["Security:Tokens:A
udience"],
            claims,
            expires: DateTime.Now.AddMinutes(30),
            signingCredentials: creds);
            var tokenText = new JwtSecurityTokenHandler().WriteToken(token);
            return tokenText;
    }
```

Cookies or Headers for Authentication?

First, some background. Authentication tokens (such as JWTs) are typically transmitted in the HTTP Authorization header, like this:

```
GET /foo
Authorization: Bearer [token]
```

Tokens can also be transmitted via browser cookies. Which transport method you choose (headers or cookies) depends on your application and use case. For mobile applications, headers are the way to go.

For web applications, we recommend using HttpOnly cookies instead of HTML5 storage/headers, for better security against XSS attacks. It's important to note that using cookies means that you need to protect your forms against CSRF attacks (by using ASP.NET Core's AntiForgery features, for example).

Validating Tokens in ASP.NET Core

First, you'll need to create a `SecurityKey` from your secret key. For this example, I'm creating a symmetrical key to sign and validate JWTs with HMAC-SHA256. You can do this in your `Startup.cs` file:

```
// secretKey contains a secret passphrase only your server knows
var secretKey = "mysupersecret_secretkey!123";
var signingKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(secretKey));
```

Validating JWTs in Headers

In your `Startup` class, you can use the `UseJwtBearerAuthentication` method in the `Microsoft.AspNetCore.Authentication.JwtBearer` package to require a valid JWT for your protected MVC or Web API routes:

```
var tokenValidationParameters = new TokenValidationParameters
{
    // The signing key must match!
    ValidateIssuerSigningKey = true,
    IssuerSigningKey = signingKey,

    // Validate the JWT Issuer (iss) claim
    ValidateIssuer = true,
    ValidIssuer = "ExampleIssuer",

    // Validate the JWT Audience (aud) claim
```

```
    ValidateAudience = true,
    ValidAudience = "ExampleAudience",

    // Validate the token expiry
    ValidateLifetime = true,

    // If you want to allow a certain amount of clock drift, set that here:
    ClockSkew = TimeSpan.Zero
};

app.UseJwtBearerAuthentication(new JwtBearerOptions
{
    AutomaticAuthenticate = true,
    AutomaticChallenge = true,
    TokenValidationParameters = tokenValidationParameters
});
```

With this middleware added to your application pipeline, any routes protected with `[Authorize]` will require a JWT that passes the following validation requirements:

- The signature matches your server's secret key
- The expiration date (`exp` claim) has not passed
- The not-before date (`nbf` claim) *has* passed
- The Issuer (`iss`) claim matches "ExampleIssuer"
- The Audience (`aud`) claim matches "ExampleAudience"

If there is not a valid JWT in the Authorization header, or it fails these validation steps, the request will be rejected. If you're not familiar with the JWT spec, the Issuer and Audience claims are optional. They are being used here to identify the application (issuer) and the client (audience).

How to Secure JWT

There are a lot of libraries out there that will help you create and verify JWT, but when using JWT's there still some things that you can do to limit your security risk.

- Always verify the signature before you trust any information in the JWT. This should be a given, but we have recently seen security vulnerabilities in other company's JWT frameworks. One gotcha that we have seen recently is around the JWT spec that allows you to set signature algorithm to 'none'. This should be ignored if you expect the JWT to be signed. Put another way, if you are passing a secret signing key to the method that verifies the signature and the signature algorithm is set to 'none', it should fail verification.
- Secure the secret signing key used for calculating and verifying the signature. The secret signing key should only be accessible by the issuer and the consumer; it should not be accessible outside of these two parties.

- Do not contain any sensitive data in a JWT. These tokens are usually signed to protect against manipulation (not encrypted) so the data in the claims can be easily decoded and read. For example, you wouldn't want to include a user's address in a JWT; you would want to store a link to the user's record or another identifier that is opaque and have your application look up the information. If you do need to store sensitive information in a JWT, check out JSON Web Encryption (JWE).
- If you worried about replay attacks, include a nonce (`jti` claim), expiration time (`exp` claim), and creation time (`iat` claim) in the claims. These are well defined in the JWT Spec(https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-25)