--Authorization--

authentication: انت مين ؟
↳ Confirms users are who they Say They are.

authorization: تقدر تعمل ايه ؟
↳ Gives users Permission to access a resource.

--What is JWT ?!--

JWT ⇒ JSON web Token.

فكرة استخدام ال JWT ← اني في اول مرة يبعت لل Server ال credentials تاعتي
اي ال username وال Password
← ال Server خلاص عرف انا مين ← وهيعطيني ال Token تباعي
وكل مرة هبعتله مني علشان يعرف انا مين
← الميزة : ال Token بيكون ليه فترة وبعد Expire ← انا اللي بحددها
في ال configuration

JWT Token include
↳ Header — فيه ال algorithm ونوع ال Token
↳ Payload — تقدر تضيف فيه اي معلومات عايز ترجعها
↳ Signature — لل client تباعي

-- Add Identity Tables --

ال Identity في ال dotNet ← بيبني ال Schema الخاصة بكل ال Tables اللي محتاجها علشان
ت manage ال user وال roles بتاعتهم وال permissions
← مجموعة من ال Features الجاهزة

① install Package ↓
Microsoft.AspNetCore.Identity.EntityFrameworkCore

② هنروح على ال ApplicationDbContext وأخليها ت inherit من IdentityDbContext
بدل ال DbContext

③ هعمل Entity جديد وهسميها ApplicationUser هخليه ي extend ال IdentityUser
بإنه ي inherit منه وهضيف عليه ال props اللي انا عاوزاها ← كل حاجة

④ هنروح نعمل ال Configurations اللي انا عاوزاها لل ApplicationUser

⑤ هنخلي ال IdentityDbContext <ApplicationUser>
↳ Generic

⑦ وبعد كدا نعمل Migration وبعدين update Database

Entity: ApplicationUser

```
Public Sealed class ApplicationUser : IdentityUser
{
    Public String FirstName {get; Set;} = String.Empty;

    Public String LastName {get; Set;} = String.Empty;
}
```

UserConfiguration

```
Public class UserConfiguration: IEntityTypeConfiguration<ApplicationUser>
{
  Public void Configure(EntityTypeBuilder<ApplicationUser> builder)
  {
     builder.Property (x => x.FirstName).HasMaxlength (100);
     builder.Property (x => x.LastName).HasMaxlength (100);
  }
}
```

ApplicationDbContext class

```
Public class ApplicationDbContext (DbContextOptions<ApplicationDbContext> options)
    : IdentityDbContext<ApplicationUser> (options)
{
   ✓
}
```

Migration

```
Add-Migration AddIdentityTables.

Update-database.
```

لو عاوزة أعمل drop لأي Column من ال Columns الأساسية الي موجودة في ال IdentityUser هنستخدم Ignore Method

لا أفضل

--Identity Api EndPoints--

ال .Net Core عاملة ال EndPoints تابعت ال Identity جاهزة علي وعلشان
استخدمها هنروح في ال program تابعت

```
builder.Services.AddIdentityApiEndPoints<ApplicationUser>
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

هنروح على ال Middlewares بعد ال Authorization

```
app.MapIdentityApi<ApplicationUser>();
```

لكن مش هنقدر نـ Customize حاجة في ال Endpoints دي
فهنضطر نبني ال EndPoints دي بنفسنا .

--Add Auth Service--

ال Service هتكون مسئولة عن انها تاخد ال UserName وال Password وبعدها تبعتهم لل JWT Provider والتي هيبدأ بـ Generate Token ويبعته تاني لل Service اللي هتاخده تبعته بعد كدا لل EndPoint .

1. هنضيف [IAuthService] ← Interface و [AuthService] ← Class
2. هنعمل ال [AuthResponse] ← Record في ال Contracts
3. هنروح نعمل implement في ال AuthService هنتسابى نعمل inject
   ونضيف ال Dependency Injection لل IAuthService
4. هنعمل ال <AuthController>
5. هنضيف ال LoginRequest
6. نعمل ال LoginRequestValidator

علشان نعمل GenerateToken محتاجين ننزل Package
Microsoft.AspNetCore.Authentication.JwtBearer.

هنضيف Folder جديد هنسميه Authentication هنضيف فيه interface
هنسميه IJwtProvider و نعمل ال class الـ JwtProvider

← UserManager فيه ميثودز بتوفر عليا اوامر بسيطة لل Identity وهو Generic
بياخد ال IdentityUser تابعنا .

```
// AuthResponse : [record]
Public record AuthResponse(
    String Id,
    string? Email,
    String FirstName,
    String LastName,
    String Token,
    int ExpiresIn
);

// IAuthService : [Interface]
Public interface IAuthService
{
  Task<AuthResponse?> GetTokenAsync (String email, String password,
                    CancellationToken cancellationToken=default);
}
```

: Authentication داخل Folder

```
// IJWTProvider : [Interface]
Public interface IJWTProvider
{ // Tuble
  (String token, int expiresIn) GenerateToken(ApplicationUser user);
}

// JwtProvider [class]
Public class JwtProvider : IJwtProvider
{
  Public (String token, int expiresIn) GenerateToken (APPlicationUser user)
  {
    Claims [] claims = [
      new (JwtRegisteredClaimNames.Sub , user.Id),
      new ( "  "  "  "  .Email, user.Email! ),
      new ( "  "  "  "  .GivenName, user.FirstName),
      new ( "  "  "  "  .FamilyName, User.LastName),
      new ( "  "  "  "  .Jti , Guid.NewGuid().ToString())
      ];

    var SymmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8
                              .GetBytes(" Key السري "));

    var SingingCredentials = new SiningCredentials (SymmetricSecurityKey,
                              SecurityAlgorithms.HmacSha256);

    var expiresIn = 30;
```

→ Cont.

```
var token = new JwtSecurityToken(
    issuer: "SurveyBasketApp",
    audience: "SurveyBasketAPP users",
    claims: Claims,
    expires: DateTime.UTCNOW.AddMinutes(expiresIn),
    SingingCredentials: SingingCredentials
    );

return (token: new JwtSecurityTokenHandler().writeToken(token),
        expiresIn: expiresIn * 60);
}
}

// AuthService : [Class]
Public Class AuthService (UserManager<ApplicationUser> userManager,
    IJWTProvider jwtProvider) : IAuthService
{
  Private readonly UserManager<ApplicationUser> _userManager = userManager;
  Private readonly IJWTProvider _jwtProvider = jwtProvider;

  Public async Task<AuthResponse?> GetTokenAsync (String email,
            String password, CancellationToken cancellationToken = default)
  {
    // Select user
    var User = await _userManager.FindByEmailAsync(email);
    // Check user
    if (user is null)
        return null;
    //check Password                          return bool
    var isValidPassword = await _userManager.CheckPasswordAsync(user,
                                                password);

    if (! isValidPassword)
        return null;

    // Generate Token
    var (token, expiresIn) = _jwtProvider.GenerateToken(user);

    return new AuthResponse(user.Id, user.Email, user.FirstName,
                    user.LastName, token, expiresIn);
  }
}
```

LoginRequestValidator , LoginRequest في الـ Authentication في الـ Folder

```
// LoginRequest : [record]
Public record LoginRequest(
  String Email,
  String Password
);

// LoginRequestValidator : [Class]
Public class LoginRequestValidator : AbstractValidator<LoginRequest>
{
  Public LoginRequestValidator()
  {
    RuleFor(x => x.Email).NotEmpty();
    RuleFor(x => x.Password).NotEmpty();
  }
}

// AuthController:
[Route("[controller]")]
[APIController]

Pubic class AuthController(IAuthService authService) : ControllerBase
{
  Private readonly IAuthService _authService = authService;

  [HttpPost("")]
  Public async Task<IActionResult> LoginAsync(LoginRequest request,
                                  CancellationToken CancellationToken)
  {
    var authResult = await _authService.GetTokenAsync(request.Email,
                    request.Password, cancellationToken);

    return authResult is null? BadRequest("Invalid email/Password"):
                                OK(authResult);
  }
}
```

Service وبنعمل Dependency injection في الـ

Services.AddScoped<IAuthService, AuthService>();

- وبعدين JWT Configuration

```
Private Static IServiceCollection AddAuthConfig (this IServiceCollection
                                                  services )
{
    Services.AddSingleton<IJwtProvider, JwtProvider>();

    Services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<ApplicationDbContext>();

    Services.AddAuthentication(options =>
    {
        options.DefaultAuthenticationScheme = JwtBearerDefaults.AuthenticationScheme;

        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(o =>
    {
        o.SaveToken = true;
        o.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            IssuerSigningKey =                    ,
            ValidIssuer = "                    ",
            ValidateAudience = "
        };
    });

    return Services;
}
```

appSettings

Ex

Public
{
    var
    {
        My
        log
        En
        On
    };
}

- User Sec

Manage

- Read J

"JWT"

-- Application Options --

- appSettings Files -

هو المكان الي بنضيف فيه ال configurations الي ال application محتاجها

← ال Hierarchy عندي ; ال json. appSettings هيقرا منها ، لو لقى نفس ال key موجود جوا الفايل تباع appSettings.env.json هيقرا منه ، لو لقى جوا ال Secrets تفاصيل ال Environment variables. هيقرأ منها ، الموجودة جوا ال variables هيقرا منها.

- → Environment variables
- → Secrets.json
- → appSettings.env.json
- → appSettings.json

Ex

```
Public IActionResult Test ()
{
   var Config = new
   {
      MyKey = _Configuration["ConnectionStrings:DefaultConnection"];
      logLevel = _Configuration["Logging:LogLevel:Default"],
      Env = _Configuration["ASPNETCORE_ENVIRONMENT"],
      OneDrive = _Configuration["OneDrive"] -> From Machine
   }; return OK(config);
}
```

- User Secrets -

هو فايل مش بيتنقل معايا لما اعمل Push للكود على ال claude أو git .
لا مش بيكون موجود جوا ال Solution

← بنروح على الفايل BasketSurvey ونضغط Right Click ونختار Manage User Secrets

---

- Read Jwt Configurations -

related Settings

هنروح ال appSettings ونضيف Section جديد ؛

```
"Jwt": {   -> Section

   "Issuer":     ✓  BasketSurveyApp ,
   "Audience":        "    "    Users,
   "ExpiryMinutes": 30
```

وهنضيف ال key جوا ال Secrets ;

```
"Jwt": {
}, "Key": "  ✓  ",
```

هنروح نيجا لاي مكان في الكود عشان الداتا دي وكليه نعرفها عادي بس جيا (configuration)

مثلا في ال Dependency Injection

```
IssuerSigningKey = new SymmetricSecurityKey ( Encoding.UTF8
                    .GetBytes( configuration["jwt:key"] )),
ValidIssuer = Configuration["Jwt:Issuer"],
ValidAudience = Configuration["Jwt:Audience"]
                ↳ Configuration.GetValue( ) عادي كتبها
```

- Options Pattern -

↳ the options Pattern uses classes to provide strongly typed access to groups of related settings. when configuration settings are isolated by scenario into separate classes, the app adheres to two important software principles:

1. Encapsulation:
   ↳ classes that depend on configuration settings depend only on the configuration settings they use.
2. Separation of concerns:
   ↳ Settings for different parts of the app aren't dependent or coupled to one another.

→ Must be non-abstract.
→ Has Public read-write properties of the type that have corresponding items in config are bound

- Options Pattern in Action: -

Folder Authentication في Class اسمه JwtOptions

```
Public class JwtOptions
{
    Public Static String SectionName = "jwt";
    // keys: appsettings
    Public String Key {get; init;} = String.Empty;
    Public String Issuer {get; init;} = String.Empty;
    Public String Audience {get; init;} = String.Empty;
    Public int ExpiryMinutes {get; init;}
}
```

③ بعد كدا هنروح للـ Controller بتاعنا نعمل inject للـ IOptions مكان الـ IConfiguration علشان دا اللي هنقرأ منه بعد كدا.

```
Public class AuthController (IAuthService authService, IOptions<JwtOptions>
            Jwt options ) : Controller Base

{
  Private readonly IAuthService _authService = authService;
  Private readonly JwtOptions _jwtOptions = jwtOptions.value;

//testEndPoint:
 [HttpGet("Test")]
 Public IActionResult Test()
 {
   return ok (_jwtOptions.Audience);
 }
```

_ كدا هو هيرجع response 200 عادي بس مش هيرجع الـ value اللي كاهستنيه نرجعها علشان محتاجين نزودها في الـ addAuthConfig اللي موجودة في الـ DI:

④ هنروح على الـ Dependency injection في جزء الـ AddAuthConfig

```
Services.Configure<JwtOptions>(Configuration.GetSection(JwtOptions.
                                              SectionName));
```

← علشان اكتب اسم الـ Section عن طول على "JWT" زي ما موجود في الـ appsettings

← كدا عرفناه ان جوا الـ Configuration بتاعت الـ application بتاعنا فيه هيعمل Bind لـ Section اللي احنا عرفنا الـ class اللي حددنا اسمه

- Bind Jwt Configurations -

علشان نعمل الـ Configuration اللي في الـ DI ونقدر نقراها من الـ appSettings

← هنروح في الـ AddAuthConfig

```
var jwtSettings = configuration.GetSection(JwtOptions.SectionName).Get
                                                    <JwtOptions>();
```

هنروح نغير الـ validators :

```
IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(
                                            jwtSettings?.key!)),
ValidIssuer = jwtSettings?.Issuer,
ValidAudience = jwtSettings?.Audience
```

- Options Interfaces -

→ IOptions <TOptions> → دا النوع اللي نستخدمه في أغلب الحالات.

- reading Configurations once with application Start.
- registered as a Singleton.
- Can be injected into any Service lifetime.

- هو بـ initiate منه instant بياخد الـ values يقرأها أول ما الـ application بيشتغل وملوش صوح نقرأها تاني
- يعني أول حاجة تقرأها تم أول ما الـ application اشتغل.
- نقدر نستخدم الـ interface في أي نوع Service سواء Singleton, Scoped

→ IOptionsSnapshot <TOptions>

- reading configurations with each request.
- registered as a Scoped.
- Can't be injected into a Singleton Service.

- بيستخدم نفس الـ instant في الـ request الواحد والـ request الجديد هيكريت instant جديد

→ IOptionsMonitor<TOptions>

- reading Configurations with each file update.
- registered as a Singleton.
- Can be injected into any Service lifetime.

أما الـ application بيشتغل يقرا الـ values على الـ values حصلها update يرجع يقراها من تاني

- Options Interfaces in Action: -

✓

- Options Pattern Validations -

DataAnnotation ← [Required] على الـ Issuer, Key, Audience في الـ JwtOptions Class والـ ExpiryMinutes هنحط عليها [Range (1, int.MaxValue)]

وهنضيف على الـ AddAuthConf

```
Services.AddOptions<JwtOptions>()
    .BindConfiguration(JwtOptions.SectionName)
    .ValidateDataAnnotations()
    .ValidateOnStart();
```

ValidateOnStart: دا علشان لو فيه مشكلة في الـ validation تظهر الـ exception من الأول قبل ما الـ app يرن.

هكا هي

```
Services.Configure<JwtOptions>(Configuration.GetSection(JwtOptions.SectionName));
```