- Audit Logging -

الغرض منها إن أقدر أعرف مين ال user اللي عمل create لل Entity دي أو آخر update حصله اللي عمله امتى وهكذا

① هنعمل Entity اسمها AuditableEntity وهنخلي ال Entities اللي عندي يـ inherit منها

```
Public class AuditableEntity
{
   Public String CreatedById {get; Set;} = String.Empty;
   Public DateTime CreatedOn {get; Set;} = DateTime.UtcNow;
   Public String? UpdatedById {get; Set;}
   Public DateTime? UpdatedOn {get; Set;}

   // Nav. Props.
   Public ApplicationUser CreatedBy {get; Set;} = default;
   Public ApplicationUser? UpdatedBy {get; Set;}
}
```

② هنروح نقف على ال Entities اللي عندي ونخليها تـ inherit منه

```
Public ~~class~~ Sealed class Poll : AuditableEntity
```

③ هنعمل ال Migration عادي ولما نيجي نعمل Updat-Database هيطلع error علشان ال CreatedByID أنا عامله required وفيه داتا في الداتابيز هيكون فيها بـ null فهنروح على الداتابيز ونمسح الداتا اللي عندي

```
Delete From Polls
```

وبعدين نظبط بقى ال Id علشان يرجع يعد من ال ١ تاني

```
DBCC CheckIdent ('Polls', RESEED, 0);
```

وبعدها هنعمل ال updat لل databas عادي

- Assign Audit values Automatically -

هنروح على ال ApplicationDbContext نعمل override على ال SaveChangesAsync

① هنعمل inject لل interface ← IHttpContextAccessor جوا ال class عندي

```
Public class ApplicationDbContext(DbContextOptions<ApplicationDbContext> options,
  IHttpContextAccessor httpContextAccessor) : IdentityDbContext<ApplicationUser>
                                                  (options)
{  Private readonly IHttpContextAccessor _httpContextAccessor = httpContextAccessor;

Public override Task<int> SaveChangesAsync(CancellationToken CancellationToken
                                           = default)
{
   ~~var CurrentUserId = _httpContextAccessor.HttpContext?.User.FindFirstValue~~
   ~~(ClaimTypes.NameIdentifier);~~
```

cont. →

```
var entries = ChangeTracker.Entries <AuditableEntity>();
Foreach (var EntityEntry in entries)
{
 var CurrentUserId = _httpContextAccessor.HttpContext? .User. FindfirstValue
                                        (ClaimTypes.NameIdentifier);
 if(EntityEntery.State == EntityState.Added)
 {
   EntityEntry.Property(x => x.CreatedById).CurrenValue = CurrentUserId;
 }
 else if (EntityEntry.State == EntityState.Modified)
 {
  EntityEntry.Property(x => x.UpdatedById).CurrentValue= CurrentUserId;
  EntityEntry.Property(x=>x.Updatedon).Currentvalue = DateTime.UtcNow;
 }
}
return base.SaveChangesAsync(CancellationToken);
}
```

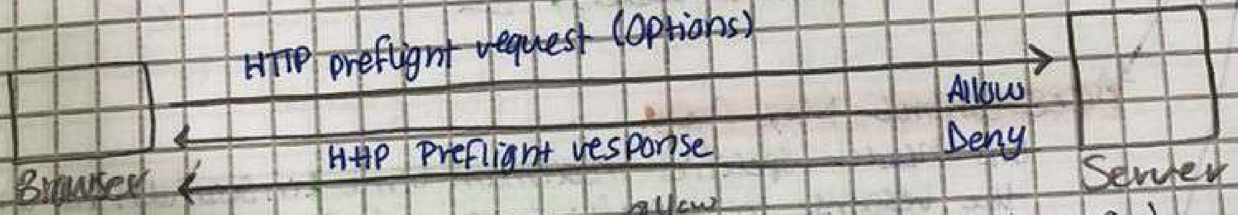
- CORS - → البرتكلات دي متعلقة بال browsers
يعني websites

→ CORS ⇒ Cross-origin Resource Sharing.

CORS is a Security Standard that enables Servers to indicate the
origins from which browsers are allowed to request resources. It was
created to refine the Same-origin policy (SOP), which browsers use to prevent
malicious applications from accessing sensitive data on domains they do not
control.
هي Security Standard بتسمح لل Server إنه يدي لل browser اللي بيبعت ال request
إذا كان مسموح له يبعت ال request ده.

→ SOP ⇒ Same-origin Policy.

SOP is a browser Security feature that restricts how resources can be accessed
by different web applications. this Policy requires that a resource must be
from the origin as the web application that is attempting to access it.
لما بنطلب ال resource لو ال resource دا موجود تحت نفس ال domain نفس ال origin
يعني ال browser هيسمح لي إني اتعامل مع ال resource دا عادي
بس لو كانت ال resource دا على domain تاني وانا كنت اوصلها بـ origin تاني
ال browser هيروح يشوف الإذن لو انا مسموح ليا اتعامل ولا لأ.



هو هيرد على ال request بـ allow or deny

-Preflight Request Headers-

→ Request Headers:

↳ Access-Control-Request-Method:
this header contains the HTTP method that will be used when the browser makes the actual request. it tells the Server what to expect when the real request is made.

↳ Access-Control-Request-Headers:
this is a list of headers that will be sent with the actual request, including custom request headers.

→ هو كدا عرفه معلومات ⟨ ال origin مين
نوع ال HTTP Method اللي مع ال request
ال Headers اللي مع ال request.

Response Headers:

Access-Control-Allow-Origin:
this header tells the browser which origins can access its resources.
It can be Set to the value of the origin itself Ex: https://SurveyBasket.com

or it can use a wildcard (*) instead.
ساع تستعمل الـ request من URL معين أو (*) يعني أي URL.

Access-Control-Allow-Credentials:
this header is a boolean value that indicates whether the browser should include credentials (like Cookies or HTTP authentication credentials) when making a Cross-origin request.
بيكون ضمن value من 2 ← True ← الـ Server هيضيف الـ browser يبعت الـ credential مع الـ request
or False

Access-Control-Allow-Methods:
this header is used to respons to a Preflight request to indicate which request Method are allowed in main request.

Access-Controls-Allow-Headers:
which headers
الاتنين دول بيحددوا الـ methods والـ headers اللي مسموح بيهم

Access-Control-Max-Age:
this header specifies maximum time (in Seconds) for which the response to a preflight request can be cached.
السيرفر بيعرف الـ browser أقصى مدة ممكنة علشان يقدر يـ cache بيانات الـ Preflight الـ request.

Access-Control-Expose-Headers:
this header Specifie which headers can be exposed to the browser. Any header that is not specified here will not be accessible by Client-Side JavaScript Code.
السيرفر بيعرف الـ browser إيه هي الـ headers اللي يقدر يستخدمها عن طريق الـ Java

Configure CoRs -

① هنروح على الـ DependencyInjection ←

```
Services. AddCors ( options =>
                    options . AddPolicy ("AllowAll", builder =>
                                          builder
                                                 . AllowAnyOrigin()
                                                 . AllowAnyMethod()
                                                 . AllowAnyHeader()
                    ));
```

كدا معناه إنه هيسمح لأي origin وأي method وأي header

② بعد كدا هنروح على الـ Program ونضيف الـ Middleware بس لازم تكون قبل الـ AutheriZation ←

```
app.UseCors( "AllowAll");
```

طب لو عاوزة أغير الـ Cors وأخليها لـ origin معينة ←

```
Services. AddCors (options =>
                    options. AddPolicy ("MyPolicy", builder =>
                                          builder
                                                 . AllowAny Method()
                                                 . AllowAnyHeader()
                                                 . WithOrigins("            ")
                    ));
```

نحط هنا الـ host بتاع الـ react أو أي حاجة أنا عاوزاه

وهنغير في الـ Program ←

```
app. UseCors ("MyPolicy");
```

refactoring Code:

بدل ما أغير في مرحلة الـ development أو أضيف الـ origin في الجزء بتاع
AppSettings. development

```
{
 "AllowedOrigins": [
   "          ",
   "          "
 ]
}
```

بعد كدا هنروح على الـ DI ونضيف ←

```
var allowedOrigins = Configuration.GetSection ("AllowedOrigins"). Get<string[]>();
```

بعد كدا هنروح نباصي الـ allowed origins في الـ cors عادي.

- Multi CORS Policies -

← أقدر أخلي EndPoint معينة بس اللي تقدر الـ Cors تـ apply عليها بإني أروح على الـ EndPoint التانية اللي مش محتاجة أ apply عليها الـ Cors و أكتب
[DisableCors]

← ينفع يكون عندي أكتر من Policy في الـ Cors

```
Options.AddPolicy(" 1 ", —
        .
        .);
options.AddPolicy(" 2 ", —
        );
```

فأقدر أحدد بعد كدا أي Policy منهم تتنفذ على أي EndPoint من اللي عندي ← هروح على الـ EndPoint وأكتب

```
[EnableCors("   ")]
```
تكتب اسم الـ Policy اللي عاوزاه

- Default CORS Policy -

in DI

دي اللي هستعمل فيها

```
var allowedOrigins = ___;
Services.AddCors(options =>
        options.AddDefaultPolicy(builder =>
            builder
                .AllowAnyMethod()
                .AllowAnyHeader()
                .WithOrigins(allowedOrigins)
        ));
```

in Program

```
app.UseCors();
```
هو هيفهم إنك هتستعمل الـ default.

- Error Handling -

هنضيف Folder نسميه Abstractions ونضيف جواه class نسميه Result
وهنضيف record هنسميه Error

```
Public record Error (String Code, String Description)
{
  Public Static readonly Error None = new(String.Empty ,String.Empty);
}
```

كدا ال Shape لل Errors.

```
Public class Result (
{
  // ctor
  Public Result (bool isSuccess , Error error)
  {  لو هو Failure والerror بيساوي none   او   لو هو Sucess وال error مش بيساوي none
    if( (isSuccess && error != Error.None) || (!isSuccess && error == Error.None))
        throw new InvalidOperationException();

    IsSuccess = isSuccess;
    Error = error;
  }
```

NOTE
الي فات
بنضبط ال class Result

```
  //Props
  Public bool IsSuccess {get;}
  Public bool Isfailure => !IsSuccess;
  Public Error Error {get;} = default;

  //Methods to return Data
  Public static Result Success () => new (true , Error.None);
  Public Static Result ~~Failure() => new (false ,~~
                       }Failure (Error error) => new (false , error);

  Public Static Result<Tvalue> Success<Tvalue>(Tvalue value) => new(value,
                                                   true , Error.None);
Public static Result<Tvalue> Failure<Tvalue> (Error error) => new(default, False, error);
}
                    حطه علشان ده ممكن يرجع values في ال return
Public Class Result<Tvalue> : Result
{
  Private readonly Tvalue? _value;
  Public Result (Tvalue? value , bool isSucces , Error error): base(isSucess, error)
  {

    _value = value;
  }
     علشان نقدر نعمل access على (value) نفسها
Public Tvalue value => IsSuccess ? _value! : throw  new
InvalidOperationException ("Failure [illegible] Cannot have [illegible]"); }
```

- Update Auth Service -

↳ to Use Result class.

① هنضيف Folder جديد اسمه Errors هنعمل فيه class هنسميها UserErrors ، وبرده هنعمل class زي الـ Entity اللي موجود عندي.

```
        static
Public  class  UserErrors
       x
{
    Public Static readonly Error InvalidCredentials =
        new( "User.InvalidCredentials", "Invalid Email / Password" );
}
```

② هنروح الـ IAuthService نغير الـ return بتاع الميثود :

```
Task<Result<AuthResponse>> GetTokenAsync( ✓      );
```

③ هنروح على الـ AuthService نخلي الـ implement

⟸ هكتب الحاجات اللي هتتغير بس

```
Public async Task<Result<AuthResponse>> GetTokenAsync(  ✓  )
{
    ✓
    if(user is null)
        return Result.~~[illegible]~~ Failure<AuthResponse>(UserErrors.InvalidCredi

    ✓
    if(! isvalidPassword)
        return Result.Failure<AuthResponse>(userErrors.InvalidCredentials);

    ✓

    var response = new AuthResponse(user.Id, user.Email, user.FirstName,
          user.LastName, token, expiresIn, RefreshToken, refreshTokenExpiration);

    return Result.Success(response);
}
```

- Update Login EndPoint -

```
Public async Task<IActionResult> loginAsync(  ✓   )
{
    ✓
return authResult.IsSuccess? Ok(authResult.value) : BadRequest(
                                                  authResult.Error);
```

-Update Poll Service-

① هكريت الـ class بتاع الـ PollErrors

② هعدل في الـ IPollService

```
Task<Result<PollResponse>> GetAsync ( ✓ );
```

③ هعدل في PollService

```
Public async Task<Result<PollResponse>> GetAsync (  ✓  )
{
  var Poll = await Context.Polls.FindAsync (id, CancellationToken);

  return Poll is not null ?
                        Result.Success(Poll.Adapt<PollResponse>())
                        : Result.Failure<PollResponse> (PollErrors.PollNotFound);
```

④ هنروح الـ PollsController

```
Public Async Task<IActionResult> Get ( ✓ )
{
  var result = await _PollService.GetAsync (id, CancellationToken);
  return result.IsSuccess ? Ok(result.value) : NotFound(result.Error);
}
```

another Service

① في الـ IPollService

```
Task <Result> UpdateAsync ( ✓ , PollRequest Poll , ✓ );
```

لأن الـ update مش بيرجع value

② هعدل في الـ PollService

```
Public async Task<Result> update Async(     ✓      ).
{
   var CurrentPoll = await Context.Polls.FindAsync (id, CancellationToken);

   if (Current Poll is null)
       return Result.Failure (PollErrors.PollNotFound);

       ✓

   return Result.Success();
}
```

④ هنروح على الـ PollsController

```
Public async Task<IActionResult> update ( ✓ )
{
  var result = await _PollService.UpdateAsync (id, request, ✓ );

  return result.IsSuccess ? NoContent() : NotFound (result.Error);
}
```

- Use Problems - → RFC Standard

← هنطلع شكل الـ Error بالظبط NotFound أو BadRequest

← في الـ Poll Controller هنروح نعدل على Get

```
Public async Task <IActionResult> Get ( ✓ )
{
    ✓

return result.IsSuccess
        ? Ok (result.value)
        : Problem (statusCode: StatusCodes.Status404NotFound,
                   title: result.Error.Code,
                   detail: result.Error.Description );
}
```

- Use OneOf -

ممكن نعمل نفس اللي عملناه بالـ class result باستخدام Package جاهزة اسمها OneOf

① هننزل Package ← OneOf

هنشتغل على EndPoint واحدة اللي هي بتاعت الـ login في الـ AuthService

① ← IAuthService

```
Task oneOf <AuthResponse, Error> GetTokenAsync ( ✓ );
```

بنقوله هترجع واحد من الاتنين إما الـ AuthResponse أو Error

② هنروح على الـ AuthService

```
Public async Task <oneOf <AuthResponse, Error>> GetTokenAsync ( ✓ )
{
```

هنعدل في الأجزاء الخاصة بالـ return

```
    ✓
  if (User is null)
      return UserErrors.InvalidCredintials;
    ✓
  if (!isValidPassword)
      return UserErrors.InvalidCredintials;

    ✓
  return new AuthResponse (user.Id,  ✓  );
}
```

③ هنروح على الـ AuthController عند الـ LoginAsync

```
return authResult.Match (authResponse => Ok (authResponse),
       error => Problem (StatusCode  ✓  ));
```

- Problems -

على الكود القديم بتاع loginAsync في الـ AuthController (Result implemented) لو جربنا واحنا بن Test الـ EndPoint نغير الـ email او الـ Password هنلاحظ الـ error message مختلفة واحنا عاوزين الاتنين يطلعوا بنفس شكل الـ error هنطبق مع الـ email

```
Public async Task <IActionResult> LoginAsync ( ✓ )
{
    ✓
return  authResult.IsSuccess
        ? OK( authResult.value)
        : Problem (
                StatusCode : StatusCodes.Status400BadRequest ,
                title : "BadRequest",
                extensions : new Dictionary <string , object?>
                {
                  {  "errors" , new[] { authResult.Error }
                  }
                }
        );
}
```

- Add to Problem Extension Method -

علشان منقعدش كل شوية نكتب الكود الطويل اللي فوق كل مرة في الـ Controller هنعمل ExtensionMethod اسمها ToProblem

① هنعمل class اسمها ResultExtension في الـ AbstractionMethod

```
Public static class ResultExtensions
{
  Public static ObjectResult ToProblem (this Result result , int Statuscode)
  {
     if ( result.IsSuccess)
          throw new InvalidOperationException (" Cannot convert Success
                         result to aproblem");

        var Problem = Results.Problem (statusCode : statusCode);
        var ProblemDetails = Problem.GetType().GetProperty ( nameof (
           ProblemDetails))!.GetValue (Problem) as ProblemDetails ;

        ProblemDetails!.Extensions = new Dictionary <String , Object?>
        {
           { "errors" , new[] { result.Error }
        }
        return new ObjectResult (ProblemDetails);
  }
```

reflection استخدمنا [illegible] access [illegible] Problem Details [illegible] protected [illegible]

8

بعد كدا هنروح على الـ AuthController هناك" هنروح للـ loginAsync

```
return  authResult.IsSuccess
        ? OK(authResult.value)
        : authResult.ToProblem(StatusCodes.Status400BadRequest);
```