Integração de Sistemas de Informação

Trabalho de Grupo (TP1)

Joel Martins 17439

João Veloso 17729

Emanuel Carvalho 8820

Introdução

O presente relatório é baseado no trabalho prático da unidade curricular de “Integração de Sistemas de Informação”, inserida no plano de estudos do curso Engenharia de Sistemas Informáticos do 3º ano.

O relatório tem como objetivo de servir de documentação do trabalho prático.

Este trabalho prático consiste na implementação de operações CRUD sobre uma base de dados. Esta base de dados armazena dados relativamente à gestão do controlo da situação pandémica.

O objetivo deste trabalho passa por consolidar conceitos de Integração de Sistemas de Informação usando serviços web, desenhar arquiteturas de integração de sistemas, recorrendo a APIs

de interoperabilidade, explorar ferramentas de suporte ao desenvolvimento de serviços web, explorar tecnologias, frameworks ou paradigmas para implementação de serviços web (SOAP e RESTful) e assimilar conteúdos lecionados na UC.

Para o desenvolvimento deste projeto foi escolhida a framework .Net 5.0 e a linguagem C# para backend, para o frontend foi escolhido o Blazor WebAssembly, utilizando o componente MudBlazor. O Blazor permite que usemos a mesma linguagem tanto para backend como para frontend, e sendo a versão WebAssembly significa que os tempos de resposta serão mais rápidos dado que é uma aplicação que corre do lado do cliente.

Foi seguida uma aproximação DDD (Domain Driven Design), trata-se de uma forma de programar tendo em conta o foco do projeto, dando nomes a funções, classes e variáveis que sejam significativas para o “negócio”, desta forma todo o código fica mais percetível, sendo a sua leitura muito mais simplificada.

Tentamos seguir ao máximo uma boa arquitetura, ficando o projeto divido em várias camadas, incluindo o uso de DTOS (Data Transfer Objects), podendo desta forma passar para o lado do cliente os dados estritamente necessários.

Para gestão de base de dados foi escolhido o sistema PostgreSQL instalado em Docker.

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Índice

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

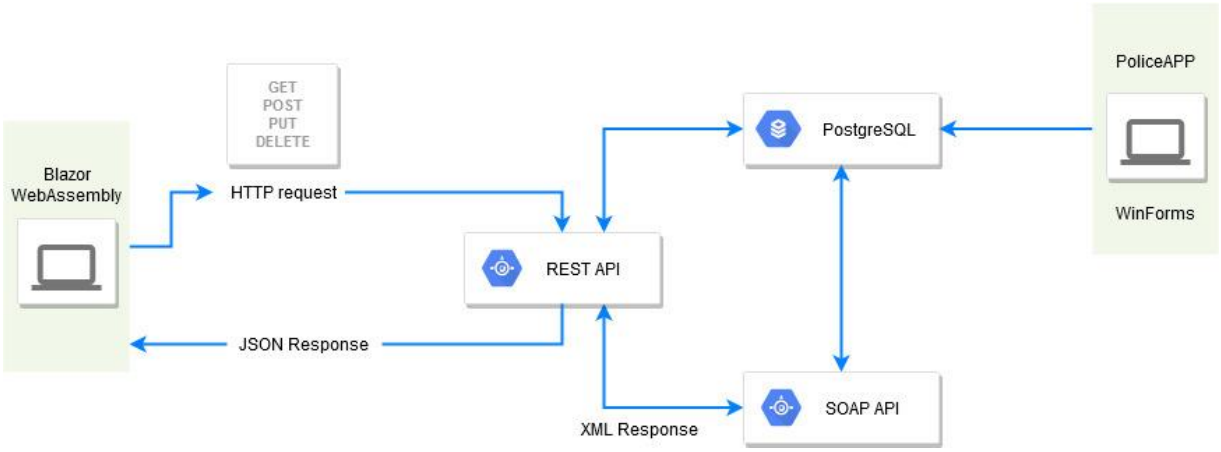
Enquadramento

Num desenvolvimento de software muitas vezes é preciso recorrer ao desenvolvimento de serviços SOAP e REST, assim qualquer aplicação, quer web, mobile ou desktop pode consumir.

Para este desenvolvimento necessitamos uma base de dados, sobre a qual iremos realizar operações CRUD.

Esta base de dados contém tabelas referentes à gestão pandémica vivida atualmente.

Em baixo o diagrama UML para exemplificar a nossa integração dos sistemas.



João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Serviços SOAP e REST

Com o objetivo de ajudar a efetuar operações CRUD (Create, Read, Update e Delete) sobre uma base de dados em SQL, foram desenvolvidos serviços para oito diferentes tabelas, nomeadamente: team, product, requisition, requisitionproduct, person, personcontact, personcovid e visit. Para além dos serviços acima citados, foi usado um serviço externo através de uma API que permite consultar os números atuais em Portugal relativos à situação pandémica. Para ter acesso às dashboards que consome um serviço chamado auth0 por token.

CR(ead)UD

Foram implementados serviços que permitem a leitura na base de dados e são exportados em JSON no caso do REST ou em XML no caso do SOAP.

No caso do REST, por exemplo:

```
/// <summary>
/// Método GET que devolve todos os produtos atualmente activos
/// </summary>
/// <returns></returns>

[SwaggerOperation("Get all active products", null, Tags = new[] { "3. Products" })]
[SwaggerResponse(StatusCodes.Status200OK, Description = "Method successfully executed.", Type = typeof(List<ProductDTO>))]
[SwaggerResponse(StatusCodes.Status400BadRequest, Description = "The endpoint or data structure is not in line with expectations.", Type = typeof(BadRequestResult))]
[SwaggerResponse(StatusCodes.Status401Unauthorized, Description = "Api key authentication was not provided or it is not valid.", Type = typeof(UnauthorizedResult))]
[SwaggerResponse(StatusCodes.Status403Forbidden, Description = "You do not have permissions to perform the operation.", Type = typeof(StatusCodeResult))]
[SwaggerResponse(StatusCodes.Status404NotFound, Description = "The requested resource was not found.", Type = typeof(NotFoundResult))]
[SwaggerResponse(StatusCodes.Status500InternalServerError, Description = "An unexpected API error has occurred.", Type = typeof(StatusCodeResult))]
// GET: api/<ProductController>/GetAllActive
[HttpGet("GetAllActiveProducts")]
0 references | astrinh0, 2 days ago | 3 authors, 5 changes
public async Task<ActionResult<List<ProductDTO>>> GetAllActive()
{
    return Ok(await _productService.GetAllActive());
}
```

Neste serviço de exemplo, chamamos o método GetAllActive pelo endpoint api/ProductController/GetAllActive que nos permite receber os produtos ativos que estão na tabela referente a Product, utilizando sempre o verbo GET como é percetível na imagem.

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

No caso do SOAP, por exemplo:

```
1 reference | XerO-PT, 15 days ago | 1 author, 1 change
public Task<Person> GetPerson(int id)
{
    try
    {
        DataContext db = new DataContext();

        return db.Persons.FirstOrDefaultAsync(x => x.Id == id);
    }
    catch (Exception)
    {
        return null;
        throw;
    }
}
```

Neste exemplo, é enviado por parâmetro o id da pessoa (um valor do tipo inteiro), e é devolvido todos os dados existentes referentes à pessoa com o id inserido naquela linha ou nulo caso não encontre.

C(reate)RUD

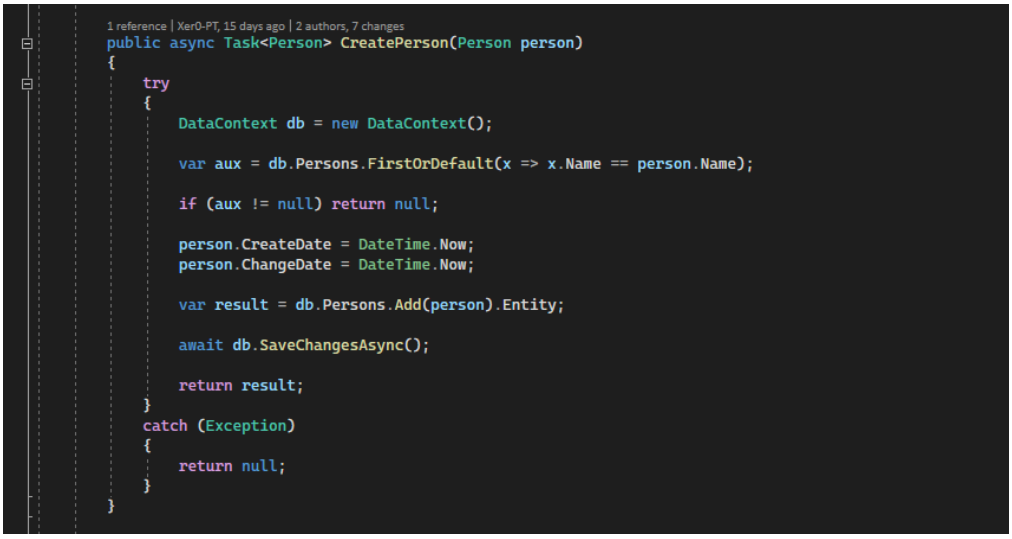
No caso do create, foram criados serviços para inserção na base de dados.

Na REST API, por exemplo:

```
[SwaggerOperation("Add a product", null, Tags = new[] { "3. Products" })]
[SwaggerResponse(StatusCodes.Status200OK, Description = "Method successfully executed.", Type = typeof(bool))]
[SwaggerResponse(StatusCodes.Status400BadRequest, Description = "The endpoint or data structure is not in line with expectations.", Type = typeof(BadRequestResult))]
[SwaggerResponse(StatusCodes.Status401Unauthorized, Description = "Api key authentication was not provided or it is not valid.", Type = typeof(UnauthorizedResult))]
[SwaggerResponse(StatusCodes.Status403Forbidden, Description = "You do not have permissions to perform the operation.", Type = typeof(StatusCodeResult))]
[SwaggerResponse(StatusCodes.Status404NotFound, Description = "The requested resource was not found.", Type = typeof(NotFoundResult))]
[SwaggerResponse(StatusCodes.Status500InternalServerError, Description = "An unexpected API error has occurred.", Type = typeof(StatusCodeResult))]
// POST api/<ProductController>/AddProduct
[HttpPost("AddProduct")]
0 references | astrinh0, 2 days ago | 3 authors, 5 changes
public async Task<ActionResult<bool>> AddProduct([FromBody] ProductDTO product)
{
    var result = await _productService.AddProduct(product);

    if (result == false) { return BadRequest(new { message = "Product already exist with that name." }); }

    else { return Ok(new { message = "Product successfully created." }); }

}
```

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Na SOAP API, por exemplo:



```
1 reference | Xer0-PT, 15 days ago | 2 authors, 7 changes
public async Task<Person> CreatePerson(Person person)
{
    try
    {
        DataContext db = new DataContext();

        var aux = db.Persons.FirstOrDefault(x => x.Name == person.Name);

        if (aux != null) return null;

        person.CreateDate = DateTime.Now;
        person.ChangeDate = DateTime.Now;

        var result = db.Persons.Add(person).Entity;

        await db.SaveChangesAsync();

        return result;
    }
    catch (Exception)
    {
        return null;
    }
}
```

Este serviço permite fazer um insert duma row na base de dados, no caso do REST, faz a inserção na tabela Product caso seja com sucesso, retorna um true, caso seja negativo, retorna um false.

No SOAP, faz a inserção na tabela Person, caso insira devolve a pessoa previamente criada, se apanhar uma exceção resolve um null.

CRU(pdate)D

Foram implementados os serviços necessários para que o utilizador possa proceder ao update de registos existentes, podendo por exemplo, alterar a quantidade de stock de um dado produto ou alterar o nome de uma pessoa.

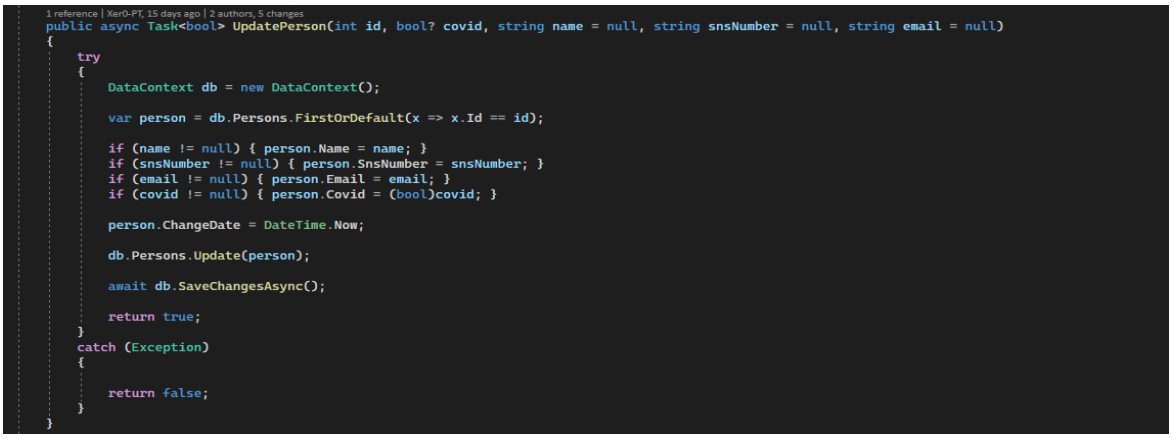
Por exemplo em REST:

```
/// <summary>
/// Método PUT que atualiza determinada Equipa
/// </summary>
/// <param name="team"></param>
/// <returns></returns>
[SwaggerOperation("Update a team", null, Tags = new[] { "5. Teams" })]
[SwaggerResponse(StatusCodes.Status200OK, Description = "Method successfully executed.", Type = typeof(bool))]
[SwaggerResponse(StatusCodes.Status400BadRequest, Description = "The endpoint or data structure is not in line with expectations.", Type = typeof(BadRequestResult))]
[SwaggerResponse(StatusCodes.Status401Unauthorized, Description = "Api key authentication was not provided or it is not valid.", Type = typeof(UnauthorizedResult))]
[SwaggerResponse(StatusCodes.Status403Forbidden, Description = "You do not have permissions to perform the operation.", Type = typeof(StatusCodeResult))]
[SwaggerResponse(StatusCodes.Status404NotFound, Description = "The requested resource was not found.", Type = typeof(NotFoundResult))]
[SwaggerResponse(StatusCodes.Status500InternalServerError, Description = "An unexpected API error has occurred.", Type = typeof(StatusCodeResult))]
// PUT api/<teamController>/UpdateTeam
[HttpPut("UpdateTeam")]
0 references | astrinh0, 2 days ago | 3 authors, 4 changes
public async Task<ActionResult<bool>> UpdateTeam([FromBody] TeamDTO team)
{
    var result = await _teamService.UpdateTeam(team);

    if (result == false) { return BadRequest(new { message = "Team doesn't exist." }); }

    else { return Ok(new { message = "Team successfully updated." }); }
}
```

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Recebemos um DTO (Data Transfer Object) e procuramos na base de dados pelo id. Substituimos os campos e retornámos um true caso seja com sucesso, false caso o resultado seja negativo, utilizando o verbo PUT.

No SOAP:

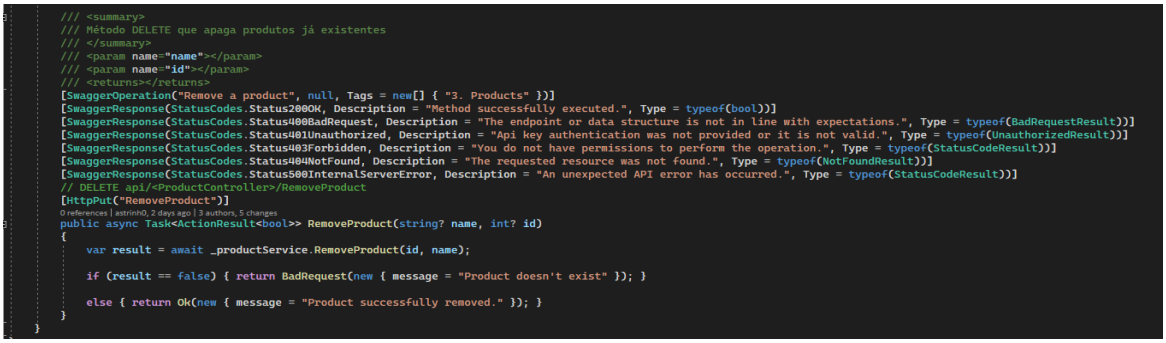


```
1 reference | Xer0-PT, 15 days ago | 2 authors, 5 changes
public async Task<bool> UpdatePerson(int id, bool? covid, string name = null, string snsNumber = null, string email = null)
{
    try
    {
        DataContext db = new DataContext();

        var person = db.Persons.FirstOrDefault(x => x.Id == id);

        if (name != null) { person.Name = name; }
        if (snsNumber != null) { person.SnsNumber = snsNumber; }
        if (email != null) { person.Email = email; }
        if (covid != null) { person.Covid = (bool)covid; }

        person.ChangeDate = DateTime.Now;

        db.Persons.Update(person);

        await db.SaveChangesAsync();

        return true;
    }
    catch (Exception)
    {

        return false;
    }
}
```

Neste caso enviamos o id obrigatório e os campos que queremos alterar. Procura por id na base de dados se o encontrar, substitui os campos que foram enviados menos o id. Retorna true caso seja com sucesso e false caso falhe.

CRUD(elete)

No caso do delete acabámos por não o implementar, porque por norma, não é correto apagar dados permanamente. Utilizámos uma flag activo para definir se está na nossa plataforma ou não, de forma, a nunca perder dados. Segue uma imagem de exemplo:

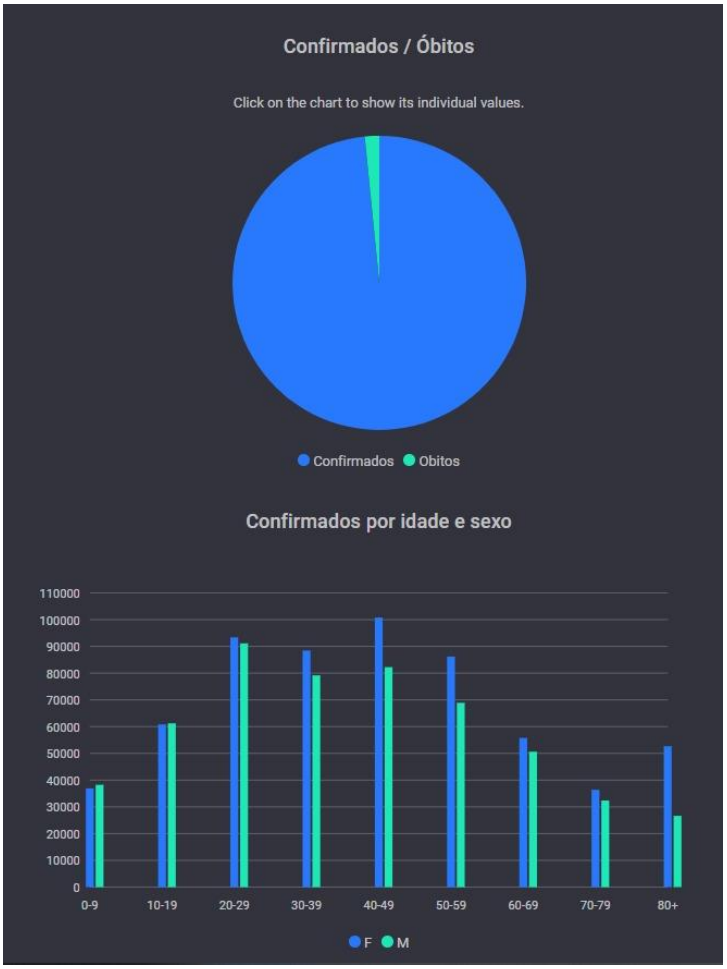


```
    /// <summary>
    /// Método DELETE que apaga produtos já existentes
    /// </summary>
    /// <param name="name"></param>
    /// <param name="id"></param>
    /// <returns></returns>
    [SwaggerOperation("Remove a product", null, Tags = new[] { "3. Products" })]
    [SwaggerResponse(StatusCodes.Status200OK, Description = "Method successfully executed.", Type = typeof(bool))]
    [SwaggerResponse(StatusCodes.Status400BadRequest, Description = "The endpoint or data structure is not in line with expectations.", Type = typeof(BadRequestResult))]
    [SwaggerResponse(StatusCodes.Status401Unauthorized, Description = "Api key authentication was not provided or it is not valid.", Type = typeof(UnauthorizedResult))]
    [SwaggerResponse(StatusCodes.Status403Forbidden, Description = "You do not have permissions to perform the operation.", Type = typeof(StatusCodeResult))]
    [SwaggerResponse(StatusCodes.Status404NotFound, Description = "The requested resource was not found.", Type = typeof(NotFoundResult))]
    [SwaggerResponse(StatusCodes.Status500InternalServerError, Description = "An unexpected API error has occurred.", Type = typeof(StatusCodeResult))]
    // DELETE api/<ProductController>/RemoveProduct
    [HttpPut("RemoveProduct")]
    0 references | astrinhO, 2 days ago | 3 authors, 5 changes
    public async Task<ActionResult<bool>> RemoveProduct(string? name, int? id)
    {
        var result = await _productService.RemoveProduct(id, name);

        if (result == false) { return BadRequest(new { message = "Product doesn't exist" }); }

        else { return Ok(new { message = "Product successfully removed." }); }
    }
}
```

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

REST API EXTERNA

Neste ponto utilizámos a seguinte API : [Covid REST API] para construir uma dashboard sobre os dados atuais da pandemia em Portugal como é demonstrado na seguinte imagem:



João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Serviço Auth0

Para a autenticação, optámos por utilizar o Auth0 em vez do OAuth porque considerámos mais fácil a integração com o nosso sistema e acaba por ser uma integração.

Auth0 é uma plataforma de autenticação e autorização adaptável e fácil de implementar.

Implementamos um método de autenticação por token em header, com a seguinte configuração:

```
services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options =>
{
    options.Authority = $"https://{Configuration["Auth0:Domain"]}/";
    options.Audience = Configuration["Auth0:Audience"];
});
```

Na imagem a acima, vemos dois campos Authority e Audience, esses dois campos têm que ser configurados da seguinte forma:

Domain: Corresponde ao link do nosso repositório

Audience: Corresponde ao que definirmos no nosso repositório

Desta forma é possível a nossa API consegue comprar o token que enviámos com o auth0 token.

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Clients

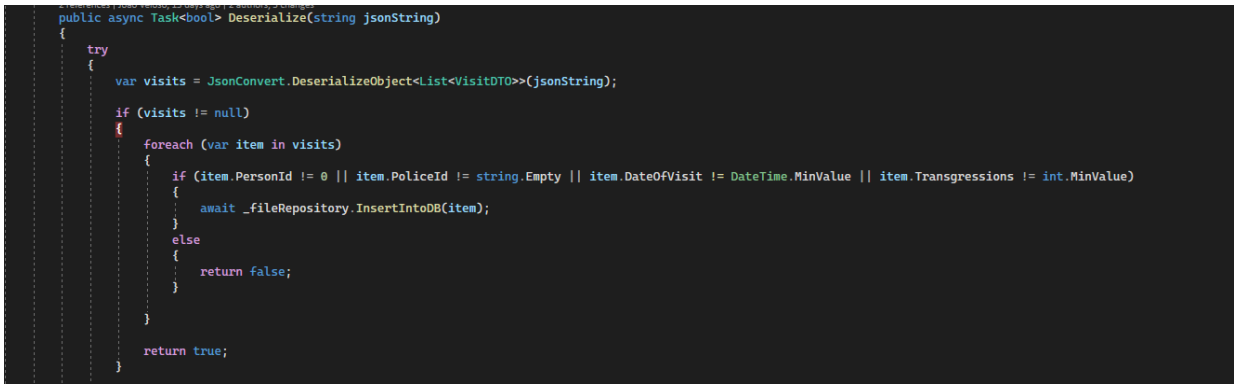
No âmbito de consumir e fazer testes nestes serviços foi desenvolvido uma solução cliente, um frontend em Blazor WebAssembly para consumir o SOAP e o REST. Também desenvolvemos uma aplicação em Windows Forms para upload de ficheiros JSON e XML.

PoliceApp

Esta aplicação tem como objetivo, focar no que foi aprendido nas aulas, na parte da serialização de JSON e XML.

JSON

Na serialização do JSON usamos a função JsonConvert.DeserializeObject para desserializar diretamente para o modelo como se verifica na próxima imagem:

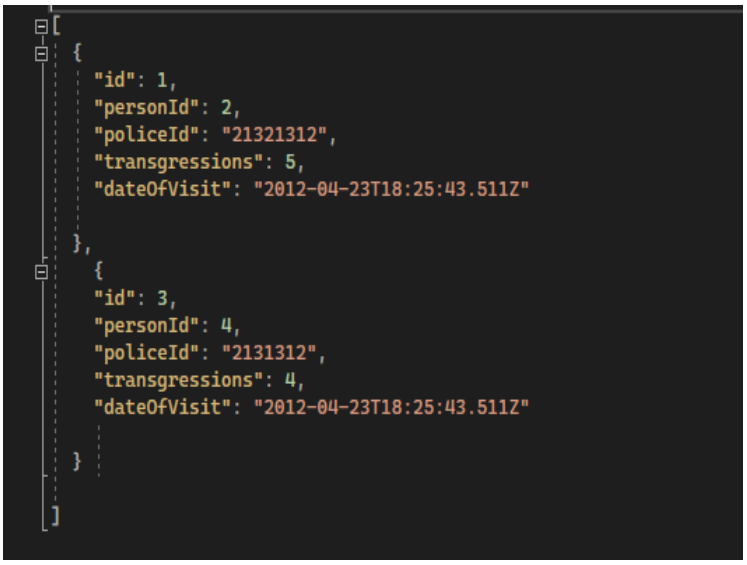


```
public async Task<bool> Deserialize(string jsonString)
{
    try
    {
        var visits = JsonConvert.DeserializeObject<List<VisitDTO>>(jsonString);

        if (visits != null)
        {
            foreach (var item in visits)
            {
                if (item.PersonId != 0 || item.PoliceId != string.Empty || item.DateOfVisit != DateTime.MinValue || item.Transgressions != int.MinValue)
                {
                    await _fileRepository.InsertIntoDB(item);
                }
                else
                {
                    return false;
                }

            }

            return true;
        }
```

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Na imagem anterior, verificámos que desserializa para uma lista de VisitDTO e verifica se cada campo está bem preenchido e de seguida insere na base dados objeto a objeto, caso seja com sucesso retorna true, caso seja negativo, retorna false.

Em baixo, segue o exemplo do JSON:



João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

XML

Na serialização do XML usamos um XPathDocument para o documento, criamos um navigator e apontamos o navigator para o primeiro nodo de visit, fazemos um foreach para percorrer todos esses nodos e selecionamos os campos que necessitámos para inserir na base de dados. Retorna true com sucesso e false caso falhe. Segue a imagem em baixo de exemplo:

```
public async Task<bool> Deserialize(string path)
{
    var docNavigator = new XPathDocument(path);
    var nav = docNavigator.CreateNavigator();



    foreach (XPathNavigator item in nav.Select("//visit"))
    {
        var transgressions = item.SelectSingleNode("@transgressions").ValueAsInt;
        var personId = item.SelectSingleNode("personId").ValueAsInt;
        var policeId = item.SelectSingleNode("policeId").Value;
        var dateOfVisit = item.SelectSingleNode("dateOfVisit").ValueAsDateTime;

        var aux = new VisitDTO
        {
            Transgressions = transgressions,
            PersonId = personId,
            PoliceId = policeId,
            DateOfVisit = dateOfVisit
        };

        if (aux.PersonId != 0 || aux.PoliceId != string.Empty || aux.DateOfVisit != DateTime.MinValue)
        {
            await _fileRepository.InsertIntoDB(aux);
        }
        else
        {
            return false;
        }
```

Em baixo, segue um exemplo do XML:

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE visits SYSTEM "xml_dtd.dtd">
<visits>
    <visit transgressions="6">
        <policeId>BCL001</policeId>
        <personId>1</personId>
        <dateOfVisit>2009-03-13T22:16:00</dateOfVisit>
    </visit>
    <visit transgressions="5">
        <policeId>BCL123</policeId>
        <personId>2</personId>
        <dateOfVisit>2009-03-13T22:16:00</dateOfVisit>
    </visit>
    <visit transgressions="1">
        <policeId>BCL666</policeId>
        <personId>3</personId>
        <dateOfVisit>2009-03-13T22:16:00</dateOfVisit>
    </visit>
</visits>
```

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

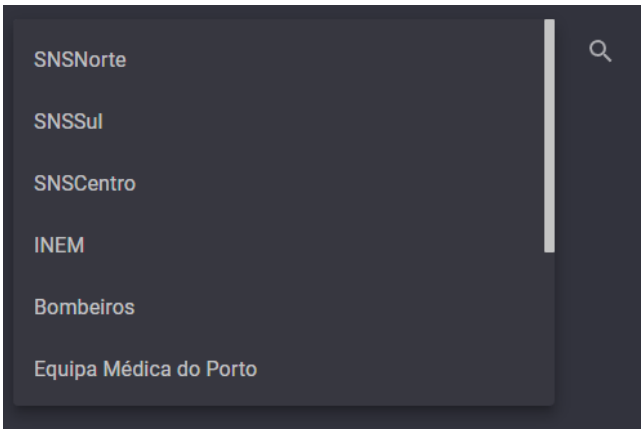
E o DTD:

```
<!--elements-->
<!ELEMENT visits (visit+)>
<!ELEMENT visit (policeId, personId, dateOfVisit)>
<!ELEMENT policeId (#PCDATA)>
<!ELEMENT personId (#PCDATA)>
<!ELEMENT dateOfVisit (#PCDATA)>

<!--attributes-->
<!ATTLIST visit transgressions CDATA #REQUIRED>
```

Blazor WebAssembly

O Blazor permite-nos desenvolver uma aplicação para frontend utilizando a linguagem C# e HTML. Desta forma ficamos com o backend e frontend desenvolvido na mesma linguagem. Optamos pela versão WebAssembly porque como a aplicação é transferida para o browser do cliente, todo o processamento fica também no lado do cliente, obtendo desta forma uma aplicação muito mais rápida devido aos tempos de resposta serem mais curtos. Apesar de ser uma tecnologia recente e de não ter tantos seguidores como por exemplo Angular, acreditamos que virá a ter bastante sucesso.

Na imagem seguinte, temos o *view model* da página em que podem ser consultadas as requisições por equipa.



Para esta view model, temos o código da próxima imagem onde podemos ver a integração da linguagem C# com o componente MudBlazor, que se distingue pelas tags do tipo HTML.

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Este excerto de código traduz-se para o monitor como um ecrã de loading enquanto os dados são carregados, posteriormente é mostrada uma típica *dropdown list*, vista na última imagem, com as equipas existentes e um botão para fazer a pesquisa. Após o utilizador pressionar o botão de pesquisa, se esta não retornar conteúdo é mostrado ao utilizador um aviso de que a equipa selecionada não tem requisições.

```
@if (IsLoading)
{
    <Loading isVisible="IsLoading" />
}
else
{
    if (TeamsList.Count != 0)
    {
        <MudItem xs="14" sm="6" md="6">
            <MudSelect T="int" Label="Teams" Strict="true" Variant="Variant.Outlined" FullWidth="true" @bind-Value="selectedTeamId">
                @foreach (var team in TeamsList)
                {
                    <MudSelectItem T="int" Value="team.Id">@team.Name</MudSelectItem>
                }
            </MudSelect>
        </MudItem>
        <MudIconButton Icon="@Icons.Material.Filled.Search" OnClick="Search" />
        <br />
    }
    if (searched)
    {
        if (RequisitionsList.Count == 0)
        {

            <MudItem>
                <MudAlert Severity="Severity.Warning">This team has no requisitions.</MudAlert>
            </MudItem>
        }
```

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Na seguinte imagem, é demonstrado o código que está a carregar os dados para a *view model* e a interagir com as ações do utilizador. Que como se pode observar é feito com a linguagem C#.

```
0 references | Xer0-PT, 21 days ago | 1 author, 2 changes
protected override async Task OnInitializedAsync()
{
    await Task.Delay(1);
    TeamsList = await TeamService.GetAllTeams();
    IsLoading = false;
}

1 reference | Xer0-PT, 28 days ago | 1 author, 1 change
public void ShowRequisitionDetails()
{
    var parameters = new DialogParameters();
    parameters.Add("SelectedRequisition", SelectedRequisition);

    DialogService.Show<RequisitionDetailsDialog>("", parameters);
}

1 reference | Xer0-PT, 21 days ago | 1 author, 2 changes
public async void Search()
{
    searched = false;
    RequisitionsList.Clear();

    if (selectedTeamId > 0)
    {
        searched = true;
        IsLoading = true;
        await Task.Delay(1);
        RequisitionsList = await RequisitionService.GetRequisitionsByTeam(selectedTeamId);
        IsLoading = false;
    }
    StateHasChanged();
}
```

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Problemas e adversidades

Ao longo do desenvolvimento deste projeto foram a encontradas algumas adversidades que nos custaram algum tempo, pelo que passamos a enumerar algumas:

- Foi escolhido o Blazor WebAssembly como aplicação para o frontend, o que mais tarde se veio a verificar um problema na integração com o serviço SOAP. Conseguimos resolver este problema integrando o serviço SOAP com o serviço REST, desta forma a aplicação de frontend faz o pedido ao serviço REST que por sua vez remete este pedido para o serviço SOAP, sendo que o único inconveniente que fica percetível é um maior tempo de resposta.

- Inicialmente queríamos fazer a integração de um serviço de login externo, por exemplo Microsoft, para acesso à dashboard, mas como já nos encontrávamos perto da data de entrega, escolhemos a integração com o serviço Auth0, que tem uma implementação fácil e rápida.

- No final tentamos ainda fazer a publicação da aplicação para o serviço Azure, mas encontramos alguns entraves e não nos foi possível finalizar antes da entrega.

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820

Conclusão

Cumprimos todos os objetivos que nos foi proposto no enunciado, integração de REST com frontend, SOAP com frontend, elaboração de dashboards, desserialização de XML e JSON e autenticação.

A realização deste trabalho permitiu-nos enraizar conceitos lecionados em aula. Permitindo-nos perceber melhor o desenvolvimento de serviços REST e SOAP tal como a integração de sistemas.

Com aprendizagem adquirida neste trabalho, conseguimos ter noções de como funciona as integrações de vários sistemas que é um tema muito utilizado no mundo profissional atualmente.

Permitiu nos também apronfundar o nosso conhecimento em C# visto que foi a linguagem que trabalhamos.

João Veloso 17729
Joel Martins 17439
Emanuel Carvalho 8820