Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Segunda Fase do Trabalho Prático

Grupo 05:

46054 Bernardo Miguel Mira Camarate

46090 Cátia Marina Soares Abreu

44876 Pedro Henrique Muzachio Castanheira

Relatório para a Unidade Curricular de Sistemas de Informação 2 da Licenciatura em Engenharia Informática e de Computadores

Professor: Nuno Datia

22 – 01 – 2022

<< Esta página foi intencionalmente deixada em branco >>

Resumo

Neste trabalho pretende-se desenvolver uma aplicação de acesso a dados utilizando a plataforma .NET, mais especificamente a linguagem C#. Esta deve ser independente no modo de acesso a dados, podendo este ser feito pela tecnologia ADO.NET ou *Entity Framework*. Deve ainda respeitar as restrições impostas na construção do sistema de informação a usar, elaborado na primeira fase do trabalho, um sistema de informação para a gestão de manutenção de ativos físicos da empresa *Maintain4ver*.

Utilizamos processamento transacional, concretizado através de mecanismos disponíveis na plataforma .NET, organizamos o código de acesso a dados para permitir usar processamento transacional, utilizamos ADO.NET em modo "conectado", utilizamos *Entity Framework* para acessos a dados, garantimos a correta libertação de ligações e recursos, quando estes não estão a ser utilizadas.

Ao dispor do utilizador constam as seguintes funcionalidades: obter o código de uma equipa livre, dada uma descrição de intervenção; criar uma intervenção; criar uma equipa de manutenção; atualizar (adicionar ou remover) os elementos de uma equipa e associar as respectivas competências; listagem das intervenções de um determinado ano; a atribuição de uma equipa a intervenção (esta funcionalidade é feita automaticamente ao criar uma intervenção); trocar as competências de dois elementos de uma equipa.

Para realizar as funcionalidades descritas recorreu-se a funções e procedimentos armazenados já implementados em SQL na primeira fase do trabalho assim como a lógica aplicacional desenvolvida em C#.

Índice

1. Introdução 5

1.1 Objetivo, problema proposto e solução 5

2. Formulação do Problema 5

2.1 Aplicação de acesso a dados: ADO.NET e *Entity Framework* 5

2.2 Funcionalidades da aplicação 5

3. Solução Proposta 6

3.1 Considerações da solução proposta 6

3.2 Modo de utilização da aplicação 6

3.3 Implementação das funcionalidades da aplicação 6

3.3.1 Aceder às funcionalidades 2e a 2i, descritas na 1ª fase do trabalho 6

3.3.1.1 Obter o código de uma equipa livre, dada uma descrição de intervenção, capaz de resolver o problema 7

3.3.1.2 Criar uma intervenção 7

3.3.1.3 Implementar o mecanismo que permite criar uma equipa 7

3.3.1.4 Actualizar (adicionar ou remover) os elementos de uma equipa e associar as respectivas competências 7

3.3.1.5 Criar uma função para produzir a listagem das intervenções de um determinado ano 7

3.3.2 Implementar a funcionalidade 2f, descrita na fase 1 deste trabalho, sem usar qualquer procedimento armazenado e implementar a atribuição de uma intervenção usando conjuntamente as funcionalidades 8

3.3.3 Alteração da aplicação desenvolvida nos pontos anteriores, de forma a que esta use uma implementação de acesso a dados desenvolvida usando Entity Framework (EF) 8

3.3.4 Usando EF com *optimistic locking*, trocar as competências de dois elementos de uma equipa. 9

4. Avaliação experimental 10

4.1 Comparação das tecnologias ADO.NET e *Entity Framework* quanto à facilidade de programação e desempenho 10

5. Conclusões 12

6. Referências 13

Lista de Figuras

Figura 1 8
Figura 2 9
Figura 3 9
Figura 4 10

Lista de Tabelas

Tabela 1 11

1. Introdução

Neste capítulo, apresenta-se uma introdução da aplicação criada para interagir com a base de dados, elaborada durante a primeira fase do trabalho [ver pasta Scripts].

1.1 Objetivo, problema proposto e solução

O objetivo desta segunda fase do trabalho é desenvolver uma aplicação em C# que use diferentes implementações de acesso a dados. Esta aplicação deve permitir ao utilizador interagir com o sistema de informação para a gestão de manutenção de ativos físicos da empresa *Maintain4ver*, elaborado na primeira fase do trabalho.

A solução para a realização da aplicação é feita através de vários projetos, dos quais os principais são:TP2-SI2 e DAL.EF. A criação de vários projetos permite estruturar o código de modo a torná-lo reutilizável e usar os dois modos de acesso a dados na aplicação.

Para garantir o bom funcionamento da aplicação, desenvolveram-se testes a algumas das funcionalidades da mesma, como é explicado no capítulo 4.

2. Formulação do Problema

Neste capítulo são referidos os detalhes da aplicação.

2.1 Aplicação de acesso a dados: ADO.NET e *Entity Framework*

A aplicação de acesso a dados utiliza o sistema de informação para a gestão de manutenção de ativos físicos da empresa *Maintain4ver*, realizado na primeira fase do trabalho. Nesta concretização devem ser implementadas as classes de acordo com as entidades e os atributos das mesmas que constam no sistema de informação. Não obstante, a aplicação deve ser constituída por uma camada abstrata que permita a reutilização de código, a fácil manutenção e a independência face à forma como se obtêm os dados.

Os dois modos de acesso a dados implementados são: o ADO.NET e o *Entity Framework*, uma ferramenta automática de acesso a dados sobre o modelo conceptual.

2.2 Funcionalidades da aplicação

A aplicação permite o acesso a algumas das funcionalidades já realizadas na fase anterior do trabalho, nomeadamente: obter o código de uma equipa livre, dada uma descrição de intervenção; criar uma intervenção; criar uma equipa de manutenção; atualizar (adicionar ou remover) os elementos de uma equipa e associar as respectivas competências; listagem das intervenções de um determinado ano; a atribuição de uma equipa a intervenção. Também permite trocar as competências de dois elementos de uma equipa, usando EF com *optimistic locking*.

3. Solução Proposta

Neste capítulo é proposta uma solução para a realização da aplicação. Na secção 3.1 são apresentadas as considerações da solução. Na secção 3.2 encontra-se o modo como o utilizador deve utilizar a aplicação. Na secção 3.3 são explicadas as decisões tomadas para realizar as funcionalidades propostas.

3.1 Considerações da solução proposta

O modelo desenvolvido para a aplicação foi feito tendo em conta o modelo Entidade-Associação e o modelo conceptual realizado na primeira fase do trabalho. Para tornar a aplicação genérica e se poder reutilizar o código implementou-se classes concretas através de interfaces. Desta forma, para outras utilizações, basta implementar as classes concretas conforme o pretendido.

Foram definidas interfaces de modo a criar "contratos" que as classes que realmente implementam o código de interação com os dados têm de seguir. Desta maneira surgiu a classe *Context* que contém a conexão à base de dados. E todas as classes *Mappers* que fazem a interação com o modelo físico. Definimos também o *model* de forma a facilitar o mapeamento em memória. Todas as classes *Mappers* derivam de *AbstractMapper*, que contém *Create*, *Delete*, *Read*, *ReadAll*, etc

De modo a tornar mais eficiente a aplicação recorreu-se a *proxies*, cuja principal função é a implementação dos *Lazy Loads*. A informação só é carregada para memória quando é necessário.

```
ctx.SaveChanges();
```

No nosso projeto decidimos utilizar o padrão *unit of work*.

Como a aplicação foi realizada para ser independente da tecnologia utilizada para aceder à base de dados (ADO.NET e *Entity Framework*), alguns métodos colocados nas interfaces não são corretamente implementados numa das versões(ADO.NET), mas são necessários nas interfaces para que estas sirvam para as duas implementações.

Algumas funcionalidades, por falta de tempo, não foram implementadas da maneira mais eficiente.

3.2 Modo de utilização da aplicação

Para ser mais fácil e rápido a utilizar a nossa aplicação, decidimos organizar um guião, que ajuda a correr a nossa aplicação. Este guião tem valores sugeridos, uma vez que há algumas restrições na base da dados, i.e: introduza um *ssn* do *supervisor*, este valor tem de ser um valor válido, não pode ser um qualquer *ssn*, uma vez que esse *supervisor* tem de ser um *Employee* que exista, por isso no nosso guião sugerímos um *ssn* de exemplo e que existe na nossa base de dados.

3.3 Implementação das funcionalidades da aplicação

Nas secções 3.3.1 a 3.3.4 são apresentadas as soluções adotadas para a implementação de cada uma das funcionalidades propostas.

3.3.1 Aceder às funcionalidades 2e a 2i, descritas na 1ª fase do trabalho

3.3.1.1 Obter o código de uma equipa livre, dada uma descrição de intervenção, capaz de resolver o problema

De forma a obtermos o código de uma equipa livre, começamos por perguntar ao utilizador qual a descrição necessária da equipa para ser capaz de resolver o problema da intervenção. Depois de obtidos os dados, através do *InterventionMapper* chamamos a função *get_code_from_team* feito na primeira fase do trabalho.

Neste caso, fazemos uso do método *ExecuteScalar*, pois este método executa a *query* e retorna a primeira coluna da primeira linha do conjuto de resultados obtidos, ou seja, vai retornar o código da equipa livre dada a descrição em causa.

No início deste método nós fazemos *context.createCommand()* com a finalidade de obter um objeto *SqlCommand* associado à S*qlConnection* em questão (que está definida no ficheiro *app.config)* o mesmo processo é feito para todos os comandos executados em baixo.

3.3.1.2 Criar uma intervenção

Da mesma forma que realizamos a alínea anterior, perguntamos ao utilizador todos os dados necessários para chamar-mos o procedimento *p_criaInter*, e com o auxílio da classe *InterventionMapper* chamamos o procedimento que permite criar uma intervenção.

3.3.1.3 Implementar o mecanismo que permite criar uma equipa

De forma a criar uma equipa, perguntamos ao utilizador os membros que pretende adicionar. Visto que o nosso procedimento armazenado recebe uma tabela com os membros a adicionar, temos um método auxiliar que transforma uma lista de *ssn's* numa tabela. Depois chamamos o procedimento *p_create_team* que cria uma equipa tendo em conta as restrições do enunciado da primeira fase do trabalho.

3.3.1.4 Actualizar (adicionar ou remover) os elementos de uma equipa e associar as respectivas competências

Após recolhida toda a informação do utilizador tendo como objetivo executar a atualização de elementos de uma equipa, passamos então chamar o método *UpdateTeam* do *TeamMemberMapper* que está incumbido de realizar o procedimento *update_team_members*. Este procedimento armazenado na primeira parte do trabalho que é reponsável por toda a lógica de dar *update* dos membros da equipa em causa.

3.3.1.5 Criar uma função para produzir a listagem das intervenções de um determinado ano

Depois de o utilizador passar como parâmetro o ano em causa o método *ListInterventationYear* é responsável por: criar o comando; selecionar a função, desenvolvida em *sql* na primeira fase, *interByYear*; executar o comando definido com auxílio do método *ExecuteReader*, que devolve um *SqlDataReader* e este sim vai ser percorrido de forma a popular a lista de intervenções efetuadas nesse ano. Por fim a lista de intervenções é devolvida ao método *GetInterventationByYear* que a percorre e apresenta a descrição de cada intervenção.

3.3.2 Implementar a funcionalidade 2f, descrita na fase 1 deste trabalho, sem usar qualquer procedimento armazenado e implementar a atribuição de uma intervenção usando conjuntamente as funcionalidades

Tendo em conta que a nossa intervenção (*p_criaInter*) já atribuiu uma equipa à intervenção e atualiza o estado desta, optámos por realizar a alínea 1b) e 1c) pedidas nesta fase do trabalho, juntas. Assim sendo, quando criamos uma intervenção, perguntamos ao utilizador a descrição da competência desejada para esta intervenção, se existir alguma equipa livre conforme a descrição pedida, atribuímos essa equipa à intervenção. Depois inserimos na tabela *scheduling* todos os dados necessários.

3.3.3 Alteração da aplicação desenvolvida nos pontos anteriores, de forma a que esta use uma implementação de acesso a dados desenvolvida usando Entity Framework (EF)

Para utilizar o *Entity Framework* (EF) como modo de acesso a dados, criou-se um projeto dll com um item ADO.NET *Entity Data Model*. E através do *wizard* configurou-se a base de dados a usar e assim gerar as entidades de domínio, *stored procedures* e funções existentes em SQL para .NET.

Para obter o código de uma equipa de manutenção que esteja livre, baseada numa descrição fornecida pelo utilizador, chamamos a função gerada pelo EF, como demonstrado na figura 1, depois é so mostrar a informação obtida ao utilizador. No fim chamando *ctx.SaveChanges*(). Procedemos da mesma forma para listar as intervenções de um determinado ano, que também se trata de uma função.

```
using (var ctx = new L51DG5Entities())
{

    IQueryable<get_code_from_team_Result> team_result = ctx.get_code_from_team(skill);
    var list = team_result.Select(team => new
    {
        team.team_code,
        team.location,
        team.n_elements,
        team.supervisor
    }).ToList();
    int? team_code = null;
    list.ForEach(team =>
    {
        (imprimir)
    });
    ctx.SaveChanges();
}
```

Figura 1

Para a criação de uma intervenção temos duas opções, uma chamando o procedimento armazenado *p_criaInter* e outra fazendo manualmente de uma forma semelhante ao procedimento.

Os procedimentos armazenados *p_create_team* e *update_team_members* criados em SQL, foram gerados pelo EF, no entanto como em SQL estes procedimentos recebiam uma tabela de *EMPLOYEES* que o utilizador desejava adicionar a uma equipa/para criar uma equipa. E o EF ao gerar estes procedimentos não contabiliza que recebe uma tabela como parâmetro. Portanto, teve de ser implementada manualmente, à semelhança da criada para o ADO.NET.

3.3.4 Usando EF com *optimistic locking*, trocar as competências de dois elementos de uma equipa.

O controlo de concorrência otimista por omissão esta não ativado, logo tivemos que ativá-lo, definindo *Concurrency mode* nas propriedades em questão.

Começamos por trocar as competências dos dois elementos escolhidos pelo utilizador, como mostra na figura 2.

```
using (var ctx = new L51DG5Entities())
{

    EMPLOYEE employee1 = ctx.EMPLOYEEs.Find(Convert.ToInt32(ssn1));
    EMPLOYEE employee2 = ctx.EMPLOYEEs.Find(Convert.ToInt32(ssn2));

    ICollection<SKILL> s1 = employee1.SKILLs;
    ICollection<SKILL> s2 = employee2.SKILLs;
    //swap
    employee1.SKILLs = s2;
    employee2.SKILLs = s1;
```

Figura 2

Depois lidamos com erros de concorrência, como se pode verificar na figura 3, com *optimistic locking*.

```
bool fail;
do
{
    fail = false;
    try
    {
        ctx.Entry(employee1).State = EntityState.Modified;
        ctx.Entry(employee2).State = EntityState.Modified;
        ctx.SaveChanges();
    }
    catch (DbUpdateConcurrencyException e)
    {
        fail = true;
        Console.WriteLine(e.Message);
        // esmagar as alterações na BD
        var entry = e.Entries.Single();
        var dbValues = entry.GetDatabaseValues();
        entry.OriginalValues.SetValues(dbValues);
        ctx.SaveChanges();
    }
} while (fail);
```

Figura 3

4. Avaliação experimental

Comparamos o desempenho das tecnologias EF e ADO.NET na implementação da alínea 1 c) que é descrita na secção 3.3.2.

4.1 Comparação das tecnologias ADO.NET e *Entity Framework* quanto à facilidade de programação e desempenho

Para comparar o ADO.NET e o *Entity Framework* quanto ao desempenho, fizemos algumas medições de tempo. Executámos os métodos em ADO.NET e depois em *Entity Framework*, ambos na base de dados do ISEL.

De forma a comparar utilizamos o seguinte método:

```
public void GetCodeFromTeam()
{
    Console.WriteLine("Get a code of an available team");
    Console.WriteLine("");
    Console.Write("Please Enter a description.");
    var description = Console.ReadLine();

    //Benchmark time
    var startTime = Environment.TickCount;

    InterventionMapper mapper = new InterventionMapper(ctx);
    int? intervention_code = mapper.GetCodeFromTeam(Convert.ToString(description));

    //Benchmark time
    var endTime = Environment.TickCount;
    var timespan = TimeSpan.FromMilliseconds((float)(endTime - startTime));
    Console.WriteLine("Duration = " + timespan);

    Console.WriteLine("Intervention code: " + intervention_code);
}
```

Figura 4

Na tabela 1, está os resultados das nossas medições, e podemos observar, como esperado, que os métodos em ADO.NET são ligeiramente mais rápidos. Isto deve-se ao facto do *Entity Framework*, gerar automaticamente os métodos e disponibilizar algumas funcionalidades que não utilizamos logo cria um ligeiro *overhead*. Mas a nível de tempo de desenvolvimento de *software*, o *Entity Framework* é muito mais rápido, pois é mais fácil o manuseamento de dados com estes a estarem acessíveis na forma de "contentores" de informação que pode ser facilmente iterável.

Método	ADO.NET	Entity Framework
Get Code From Team	0.36 s	3.17 s
Create Intervention With Procedure	1.17 s	1.18 s
Create Team	1.11 s	6.327 s
Get Intervention By Year	0.113 s	1.07 s
Create Intervention	0.391 s	1.35 s

Tabela 1

5. Conclusões

Neste trabalho construiu-se uma aplicação utilizando a plataforma .NET para interagir com o sistema de informação de uma empresa de manutenção de ativos físico. Posteriormente foram desenvolvidas duas soluções de acesso e manipulação de dados, mais concretamente uma em ADO.NET e outra fazendo uso do *Entity Framework*. Com estas duas realizações percebemos que apesar da versão que faz uso do *Entity Framework* ser mais fácil de implementar esta também é ligeiramente mais lenta que a versão que faz uso do ADO.NET.

A solução apresentada assenta nos princípios de boas práticas de realização do mesmo, aprendidas na unidade curricular de Sistemas de Informação 2. A solução adotada foi a que pareceu mais correta para as funcionalidades da aplicação.

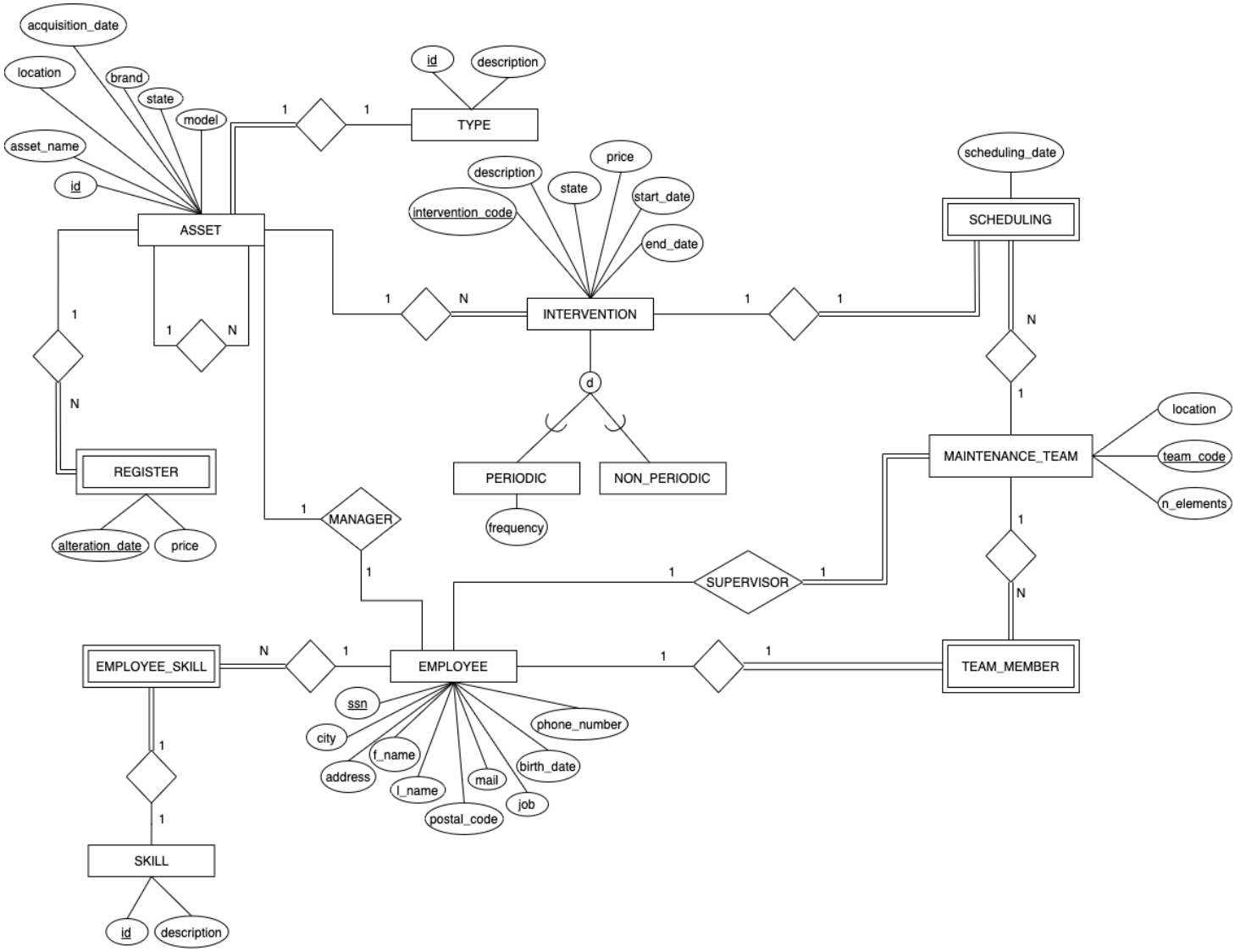
6. Referências

[1] Fundamentals of Database Systems (7th ed.), Ramez Elmasri, Shamkant B. Navathe Pearson Education, 2017

[2] SI2, ADO.NET. ISEL, 2021

[3] Walter Vieira. SI2, Acesso a dados com Microsoft Entity Framework. ISEL, 2021

A. Diagrama do modelo Entidade-Associação



B. *Entity Framework* UML