



**Área Departamental de Engenharia de Electrónica e  
Telecomunicações e de Computadores**

**Gestão de Ativos (Fase 2)**

Autores: A46971 Diogo Jorge de Castro Neto Fernandes  
A47612 Tiago José Leal Ribeiro

Relatório para a Unidade Curricular de Programação da  
Licenciatura em Engenharia Informática e de Computadores

Professor: Engenheiro Afonso Remédios

18/01/2022

## Resumo

Um Sistema de Gestão de Base de Dados (SGBD) visa a organização estruturada (modelo relacional) de dados de um determinado contexto (negócio), gerindo o seu armazenamento, manipulação e pesquisa dos dados, funcionando como uma interface entre aplicações e os dados necessários para a sua execução.

Foi proposta a implementação de um programa de acesso a dados com base em diferentes frameworks, ADO.NET e Entity Framework. Para tal, foi utilizada a base de dados trabalhada anteriormente, com o objectivo de simular a interação entre cliente e o SGBD, mantendo a independência do modo de acesso a dados.

De modo a alcançar o objectivo pretendido, foi necessário a compreensão dos diferentes modos de acesso a dados, a utilização correta das ligações com o SGBD tal como a importância das transações.

## **Abstract**

A Relational Data Base Management System (RDBMS) aims at the structured data organization (relational model) for a particular context (business), managing its storage, manipulation, and data querying, functioning as an interface between application data and its execution.

It was proposed to implement a data access program based on different frameworks, ADO.NET and Entity Framework. Was used a previously worked database, with the objective of simulating the interaction between the client and the DBMS, maintaining the independence of the data access mode.

To achieve the intended objective, it is necessary to understand the different ways of accessing data, the correct use of connections with the DBMS as well as the importance of transactions.

# Índice

1. Introdução .....	3
2. Modelo ER .....	4
3. Correções da Fase 1 .....	4
4. Modelo de desenho da aplicação .....	5
5. Desenho da Aplicação .....	6
a. Componente <i>Fase2_Project</i> .....	6
b. Componente <i>BusinessLayer</i> .....	8
c. Componente <i>DataLayer</i> .....	9
d. Componente <i>EntityFrameworkManager</i> .....	10
e. Componente <i>Model</i> .....	11
f. Componente <i>ModelInterfaces</i> .....	12
6. Testes de Performance entre ADO.Net e EF .....	12
7. Alteração de duas Competências entre funcionários com Optimistic Locking .....	13
8. Conclusão .....	14

## Lista de Figuras

Figura 1 - Diagrama ER.....	4
Figura 2 - Componente Fase2_Project .....	6
Figura 3 - Componente BusinessLayer .....	8
Figura 4 - Componente DataLayer.....	9
Figura 5 - Componente EntityFrameworkManager .....	10
Figura 6 - Componente Model.....	11
Figura 7- Resultado teste performance.....	12
Figura 8- Diagrama de troca competências.....	13
Figura 9-Mensagem de erro <i>OptimisticConcurrencyException</i> .....	14

## **1. Introdução**

Foi desenvolvida uma aplicação, com base na linguagem C#, com o objetivo de realizar todas as instruções pretendidas, acedendo à Base de dados (BD) de forma eficiente, garantindo todas as funcionalidades anteriormente trabalhadas. Com isto, foram utilizadas duas frameworks de acesso a dados: .NET e Entity Framework.

Tendo em conta que a aplicação tem de aceder às bases de dados da empresa “Maintain4ver” implementada anteriormente, todas as funcionalidades desta vão estar disponíveis na aplicação, onde vão ser apresentadas na consola. Após a escolha da framework pretendida por parte do utilizador, é executada uma das instruções e pedido ao SGBD a informação necessária para a funcionalidade do programa.



## 4. Modelo de desenho da aplicação

Com o objetivo de realizar uma aplicação eficiente, com reutilização de código e processos, simultaneamente, garantido o mínimo de acessos à base de dados, foram aplicados diferentes padrões de desenho para uma melhor organização e estrutura de dados.

Um padrão de desenho é uma solução geral para um problema recorrente. Este é uma descrição, ou molde, que pode ser utilizada na solução de diferentes problemas. Os padrões aceleram o processo de desenvolvimento, fornecendo testes e otimizações que poderiam passar despercebidas caso não se fizesse uso de um. Os padrões utilizados foram os seguintes:

- *Data Mapper* - Camada que separa a data dos objetos da memória da base de dados ;
- *Lazy Load* - Objeto que não contém toda a informação necessária, mas sabe como obtê-la;
- *Data Transfer Object* - Objeto que transporta a informação entre processos para reduzir o número de métodos;
- *Virtual Proxy* - Objeto que representa outro objeto;



## 5. Desenho da Aplicação

### *a. Componente Fase2\_Project*

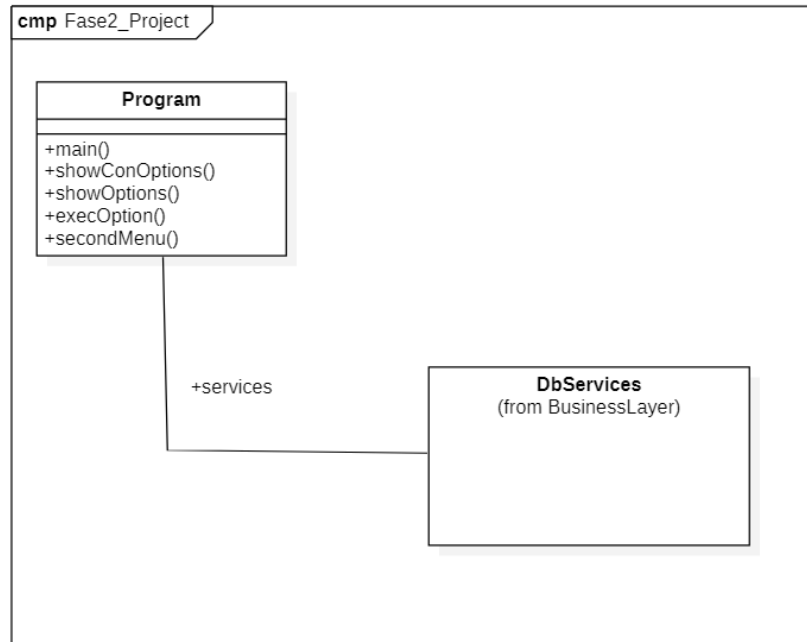


Figura 2 - Componente Fase2\_Project

Este componente foi criado com o intuito de criar uma *user interface* para que seja possível ao utilizador ter uma interação com o programa.

No primeiro menu, poderá ser escolhida o tipo de acesso à base de dados:

- 1) ADO.NET -> utiliza uma conexão ao servidor:
  - Esta framework realiza os comandos necessários ligando-se à base de dados para ler ou manipular os dados desta. As classes desta framework permitem a ligação com a BD, consulta e alteração de dados;
- 2) Entity Framework (EF)-> utiliza um mapeamento objeto relacional:
  - A EF consiste na abstração da base de dados e trabalhar com a informação na forma de objetos e propriedades. A framework utiliza um modelo da BD para a leitura ou manipulação de dados. Depois é este modelo que irá fazer a ligação com a BD;
- 3) Testes comparativos entre os dois tipos de ligação -> Apresenta testes comparativos de desempenho das tecnologias EF e ADO, na implementação da 1c.

No segundo menu, estão disponíveis todas as opções de acordo com as alíneas pedidas no enunciado da fase 2:

- 1) Obter equipa livre para uma intervenção -> acede à funcionalidade 2e da fase 1;
- 2) Inserir Intervenção com *procedure* -> acede à funcionalidade 2f da fase 1;
- 3) Inserir Equipa -> acede à funcionalidade 2g da fase 1;
- 4) Atualizar Elementos a uma Equipa -> acede à funcionalidade 2h da fase 1;
- 5) Intervenções num ano -> acede à funcionalidade 2i da fase 1;
- 6) Inserir Intervenção -> acede à funcionalidade 1b da fase 2;
- 7) Atribuir intervenção a uma equipa livre -> acede à funcionalidade 1c da fase 2;
- 8) Trocar competências entre 2 funcionários -> alínea 4 da fase 2.

## b. Componente *BusinessLayer*

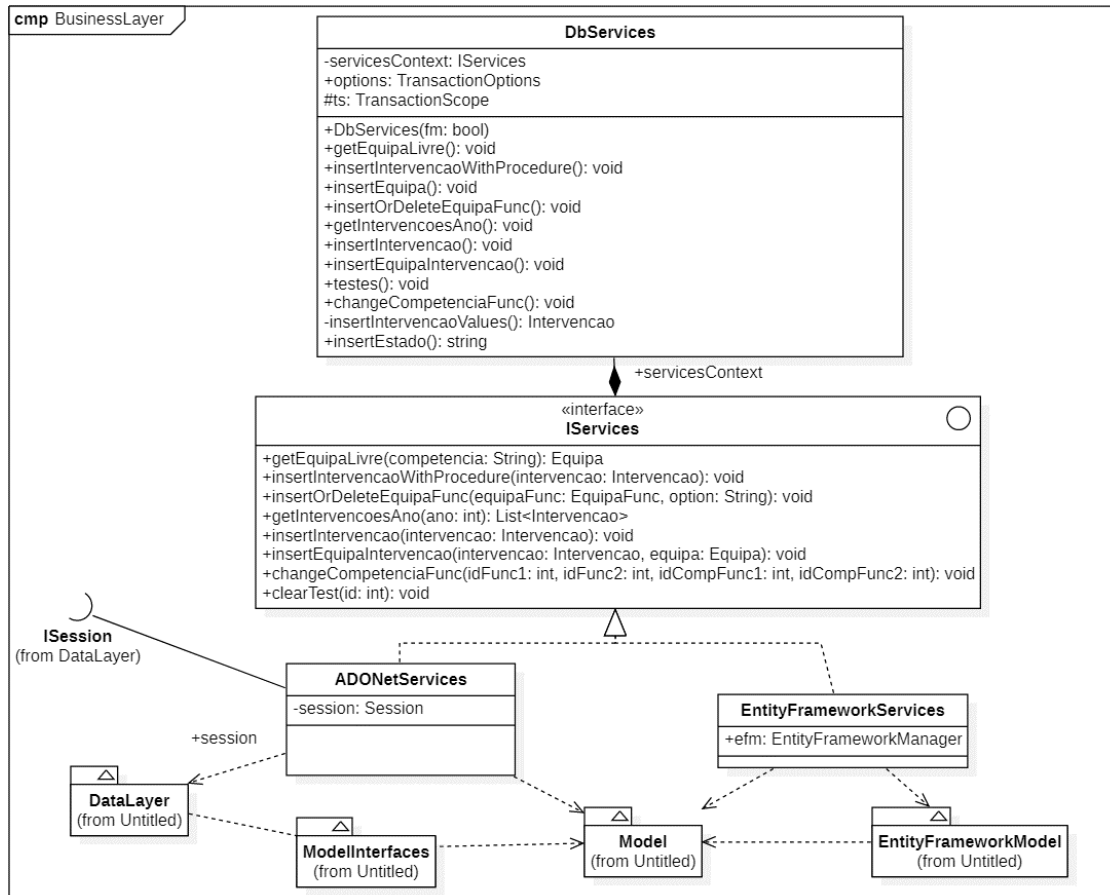


Figura 3 - Componente *BusinessLayer*

Neste componente foram criadas todas as classes essenciais para o modelo de negócio.

A classe **Db services** está directamente ligada à classe *Program* do componente *Fase2\_Project* e contém toda a logica de negócio. Aqui serão pedidos dados ao utilizador quando há necessidade de criar novos objectos para inserção na base de dados ou pedir elementos para manipular dados na base de dados. Foram também usados nesta class, os objectos para controlo transaccional: **Transaction Scope** e **Transaction Options**. Foi utilizado uma transação do tipo **Required**, que, caso esta esteja dentro de outra transação e for abortada, a transação parent também vai ser abortada e não prossegue. O isolation level necessário foi **Read Commit** devido ao facto de estarmos a fazer leituras e assim evitar dirty reads. Estes objectos são criados no construtor da classe, tal como, é criado um objecto implementado pela Interface **IServices** com o modelo de acesso à BD: **ADONetSevices** ou **EntityFrameworkServices**.

A classe **ADONetSevices** usará uma Interface **ISession**, responsável pela conexão à base de dados, passando o objecto ao componente *DataLayer*.

Já a classe *EntityFrameworkManager*, já tem incorporado um modo de acesso à base de dados (*DbContext*).

### c. Componente *DataLayer*

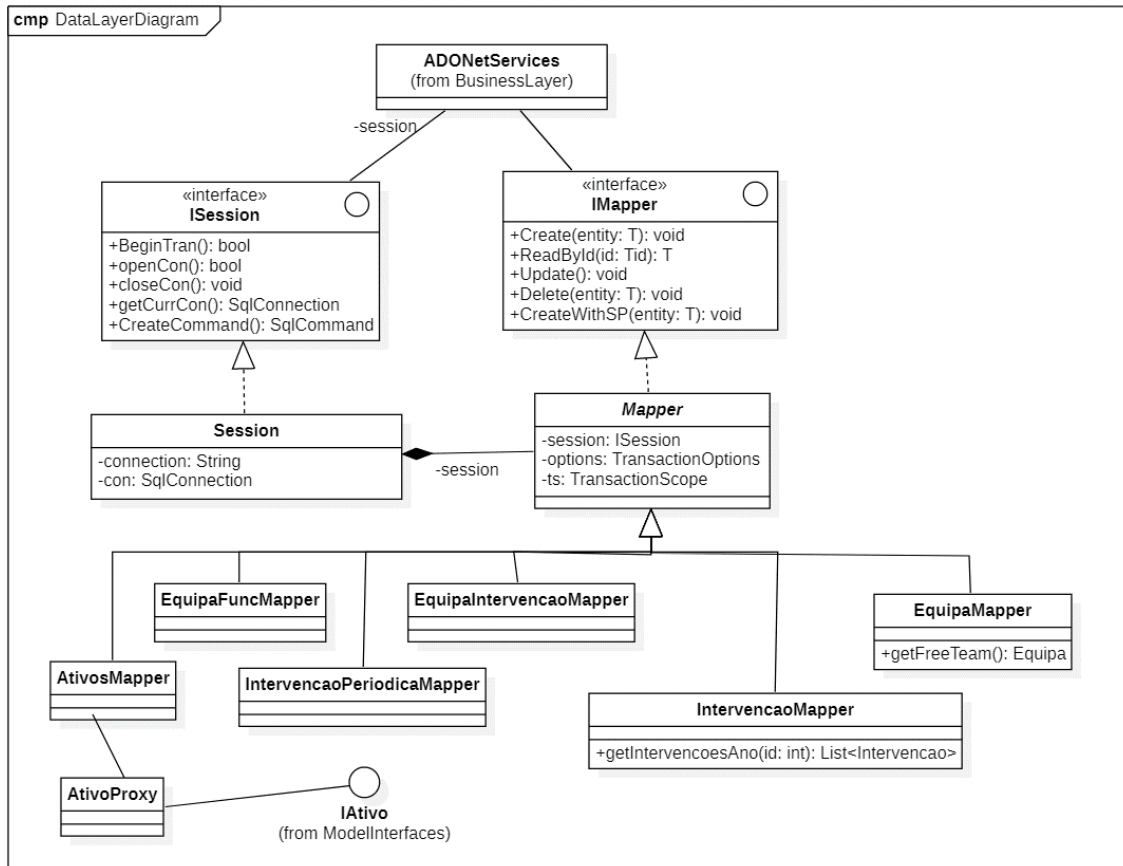


Figura 4 - Componente *DataLayer*

Neste componente, estão todas as classes necessárias para acesso à BD e manipulação de dados.

Para ligação com a BD foi criada uma interface *ISession* implementada por *Session*. É através desta classe que é feita uma conexão à base de dados, desde a leitura de comandos ao processo das transações. Sempre que é chamado o método *CreateCommand()*, é aberta uma conexão à BD e retornado um *SqlCommand* utilizado para executar as *queries*. Como a interface *ISession* implementa *IDisposable*, o fecho da conexão é salvaguardado pelo fim da instrução *using* nas classes dos *Mappers*.

A Interface *IMapper* contém os métodos básicos de um sistema que trabalha com BDs: *CRUD* (*Create*, *Read*, *Update* e *Delete*). A classe *Mapper* é uma classe abstrata que implementa *IMapper* e terá no construtor a sessão criada em *ADONetServices*. Cada *mapper* estendido desta classe, corresponde a cada tabela criada na BD. Estes *mappers* serão responsáveis por executar cada operação na sua tabela correspondente.

Também criamos objetos do tipo *Transaction Scope* e *Transaction Options* nos *mappers* de modo a ver melhor controlo transaccional e se uma destas sub-operações falhar, não comprometer as instruções sobre a BD criadas pela classe pai (*DbSevices*). Outra razão para aplicarmos este controlo transaccional, deve-se ao facto de os *mapper* poderem ser chamados de forma independente para além da classe *DbServices*.

Neste componente demonstrámos a aplicação dos modelos de desenho *Virtual Proxy* e *Lazy Load* na classe *AtivoProxy*. O *Mapper* das intervenções tem uma função para obter os Ativos que é passada no construtor deste objecto e, quando necessário, o objecto tem forma de obter os dados quando são pedidos.

#### d. Componente *EntityFrameworkManager*

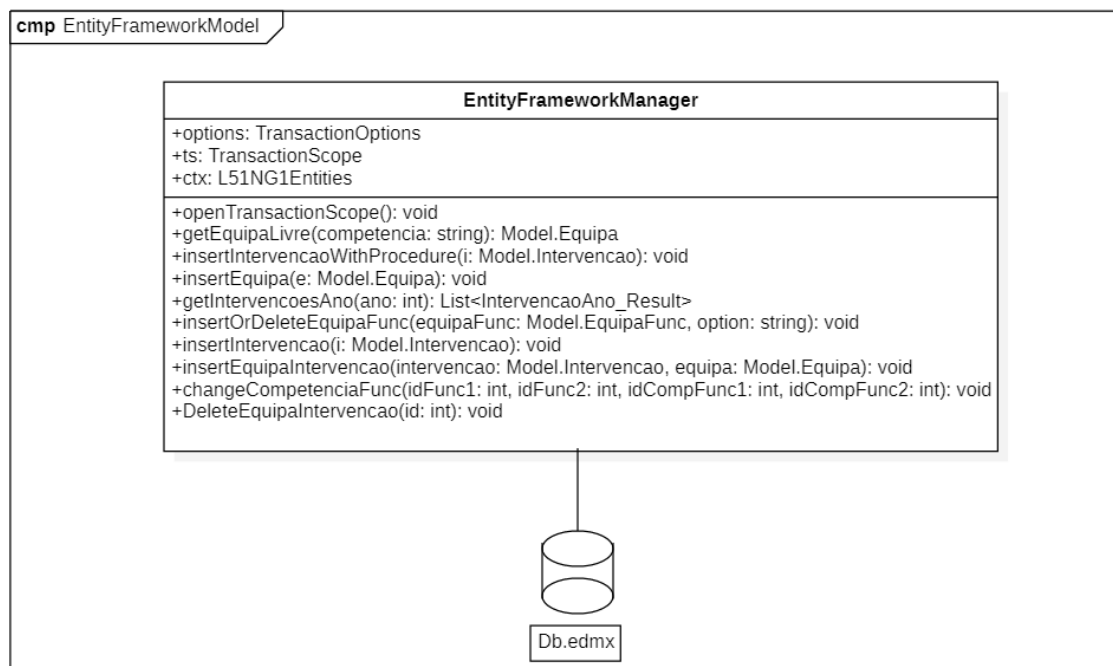


Figura 5 - Componente *EntityFrameworkManager*

A execução de operações na BD através da *EntityFrameWork* é gerida pela classe *EntityFrameworkManager*, funcionando como uma espécie de interface entre a camada de negócio, nomeadamente a classe *EntityFrameworkServices*, e a referida *EntityFrameWork*. Em todas as operações é realizado o acesso ao modelo da *EF*, previamente configurado através do IDE *VisualStudio*, sendo esse acesso efetuado através do respetivo contexto - *L51NG1Entities*. Em cada função é criado um novo *TransactionScope*, representado numa sub-transação da transação anteriormente aberta na camada de negócio. O nível de isolamento é sempre *Read Committed*. De salientar que as operações devolvem entidades em correspondência com o modelo criado por nós na componente *Model* e não do modelo da *EF*.

## e. Componente Model

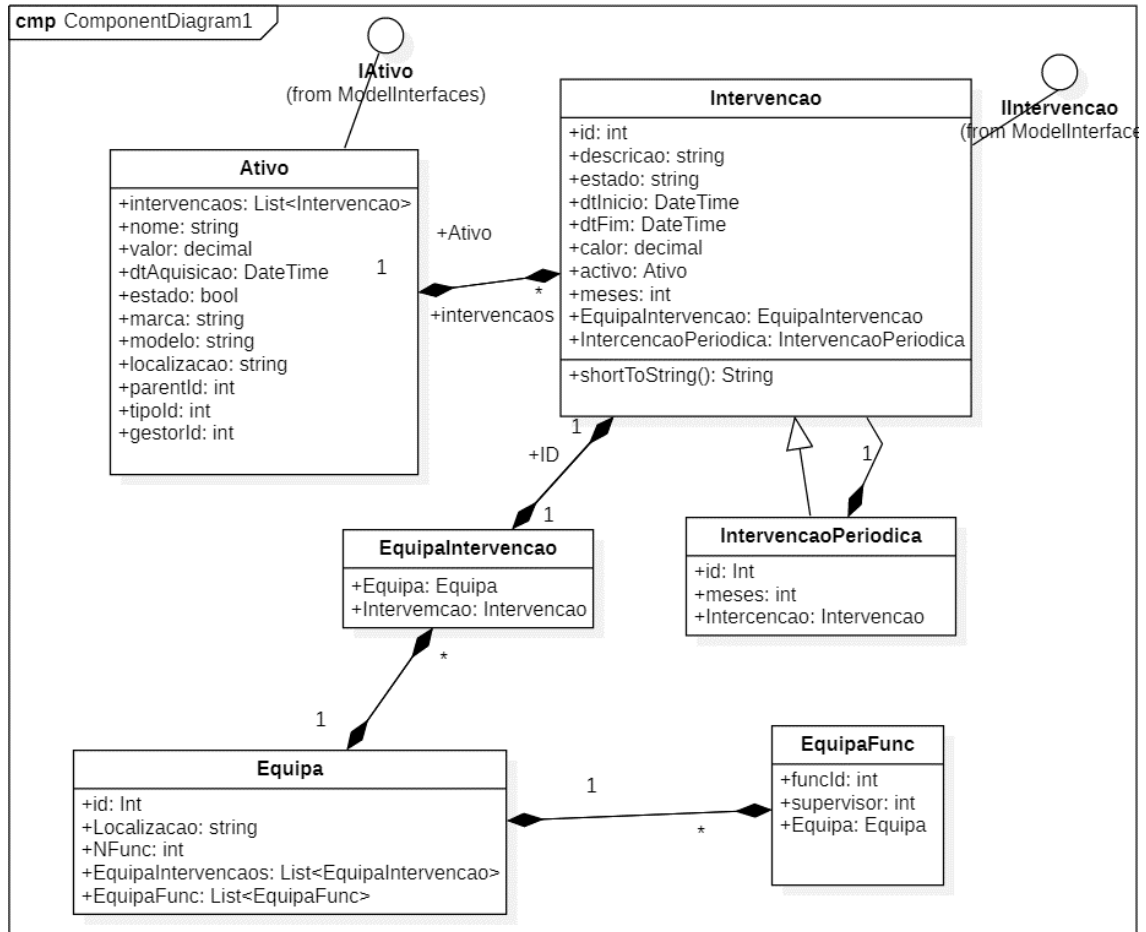


Figura 6 - Componente Model

**Nota:** Neste trabalho, não criámos todos os objetos que não tinham utilização nas alíneas do enunciado deste trabalho.

Neste componente foram criados os objetos de acordo com cada tabela existente na base de dados do projeto. Estes são uma representação do modelo de dados mantendo as mesmas propriedades incluindo objetos inseridos dentro de outros de maneira a suportar as suas relações.

Em alguns casos foi utilizado o modelo de desenho *Lazy Load* que cria os objectos entre relações com a dados disponíveis no momento de uma transacção. Este modelo reduz o numero de acessos às bases de dados tal como o tempo despendido em cada

operação. Caso esses dados sejam necessários, o programa tem uma referência para obtê-los.

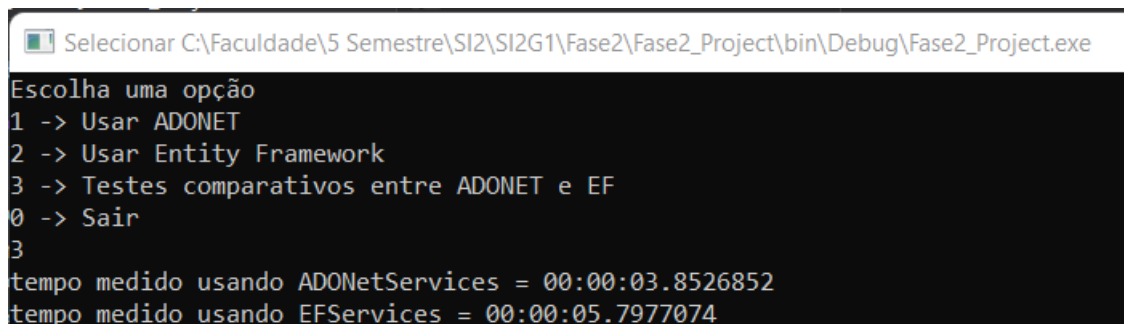
### ***f. Componente *ModelInterfaces****

Este componente foi criado por ter referências de outras classes para podermos conseguir aplicar os padrões de desenho virtual proxy e Lazy Load. Apenas criamos as interfaces dos modelos Intervenção e Ativo para efeitos de demonstração.

## **6. Testes de Performance entre ADO.Net e EF**

Para testar a performance entre o modelo EF e o ADO.NET de maneira a entender qual dos dois modelos apresenta melhor desempenho foi medido o tempo de execução de uma função em ambos os modelos.

Utilizamos a função testes () para medir o tempo para executar 100 vezes as instruções de *insertIntervencao*, *getEquipaLivre*, *insertEquipaIntervencao* e *clearTest*.



```
Selecionar C:\Faculdade\5 Semestre\SI2\SI2G1\Fase2\Fase2_Project\bin\Debug\Fase2_Project.exe
Escolha uma opção
1 -> Usar ADONET
2 -> Usar Entity Framework
3 -> Testes comparativos entre ADONET e EF
0 -> Sair
3
tempo medido usando ADONetServices = 00:00:03.8526852
tempo medido usando EFServices = 00:00:05.7977074
```

Figura 7- Resultado teste performance

Podemos ver que os acessos usando ADO.Net são mais rápidos que a EF. Uma das razões para tal acontecer é a EF gerar código automaticamente para aceder à base de dados. Outra razão é porque a EF utiliza como *background* o modelo usado pelo ADO.Net

## 7. Alteração de duas Competências entre funcionários com Optimistic Locking

Para realização desta alinea foi necessário alterar no modelo da *EF* as propriedades das entidades a serem acedidas, nomeadamente o *idFunc* e *idComp*, de forma a suportarem deteção de concorrência. Para isso foi alterado o seu *ConcurrencyMode* para *Fixed*.

De salientar que este teste foi possível realizar porque o nível de isolamento definido na transação é *Read Committed* para corresponder com uma abordagem *Optimistic Locking*. Foram usados dois contextos dentro da mesma transação para manipular as leituras e escritas. A implementação usada para forçar concorrência é demonstrada na figura seguinte.

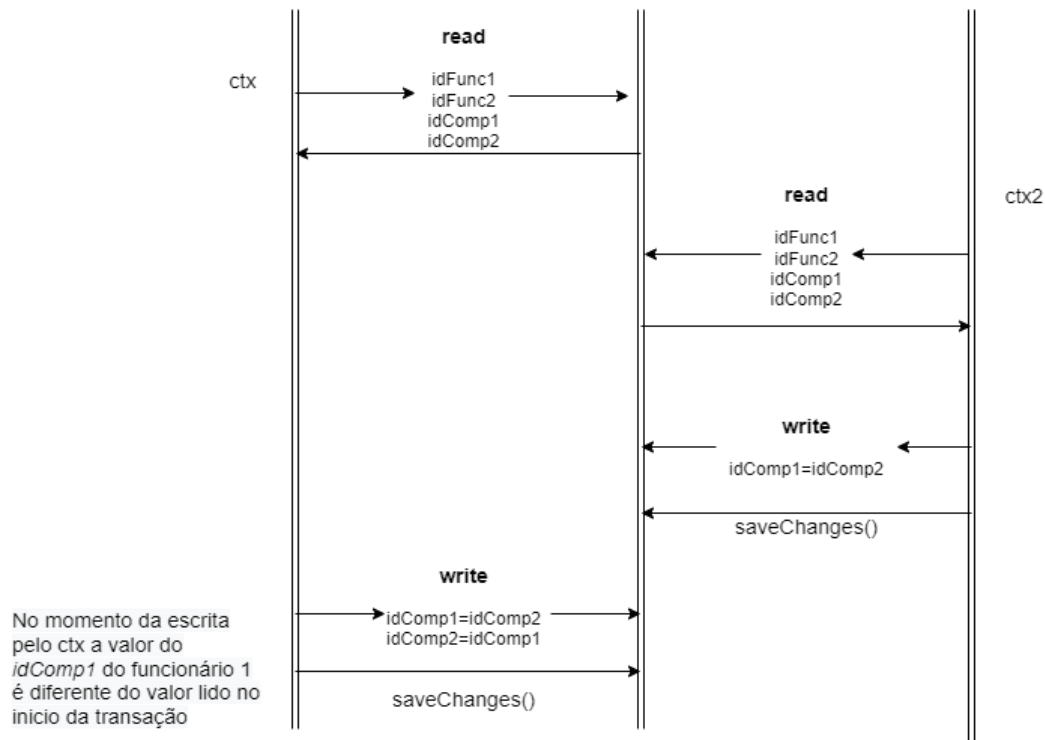


Figura 8- Diagrama de troca competências



A mensagem de erro devolvida é bem representativa da ocorrência de concorrência e da inviabilização da operação no momento da escrita. A exceção é lançada pelo objeto *DdContext* sendo esta do tipo *OptimisticConcurrencyException* que deriva de *UpdateException*.

```
7 -> Atribuir intervenção a uma equipa livre
8 -> Trocar competências entre 2 funcionários
9 -> Teste com getInteracao e obter Ativo via proxy
0 -> Sair para o menu anterior
3
Inserir id do primeiro funcionario
1
Inserir id do segundo funcionario
6
Inserir id da competencia do primeiro funcionario
2
Inserir id da competencia do segundo funcionario
3
Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or
deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding a
nd handling optimistic concurrency exceptions.
System.Data.Entity.Infrastructure.DbUpdateException
System.Data.Entity.Core.OptimisticConcurrencyException
```

Figura 9-Mensagem de erro *OptimisticConcurrencyException*

## 8. Conclusão

Na realização da segunda fase do trabalho prático obtivemos conhecimentos sobre modelos de acessos a dados, sendo estes o *ADO.NET* (em modo conectado) e o *EntityFramework*, tal como padrões de desenho diferentes de auxílio para o código. Com estes conhecimentos desenvolvemos uma aplicação em *C#* com a implementação de ambos os modelos. A necessidade de trabalhar com *mappers* deu a entender as suas funcionalidades, inclusive o suporte dado pelos mesmos. Foi também importante garantir o controlo transacional durante a realização de operações CRUD com a BD.