

**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](#_Toc93592346)

[2. Modelo ER 4](#_Toc93592347)

[3. Correções da Fase 1 4](#_Toc93592348)

[4. Modelo de desenho da aplicação 5](#_Toc93592349)

[5. ADO.NET e Entiry FrameWork 5](#_Toc93592350)

[6. Desenho da Aplicação 6](#_Toc93592351)

[*a.* Componente *Fase2\_Project* 6](#_Toc93592352)

[*b.* Componente *BusinessLayer* 7](#_Toc93592353)

[*c.* Componente *DataLayer* 8](#_Toc93592354)

[*d.* Componente *EntityFrameworkManager* 9](#_Toc93592355)

[*e.* Componente *Model* 10](#_Toc93592356)

Lista de Figuras

Figura 1 - Diagrama ER 4

Figura 2 - Componente Fase2\_Project 6

Figura 3 - Componente BusinessLayer 7

Figura 4 - Componente DataLayer 8

Figura 5 - Componente EntityFrameworkManager 9

Figura 6 - Componente Model 10

# 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 ultilizador, é executada uma das instruções e pedido ao SGBD a informação necessária para a funcionalidade do programa.

# Modelo ER

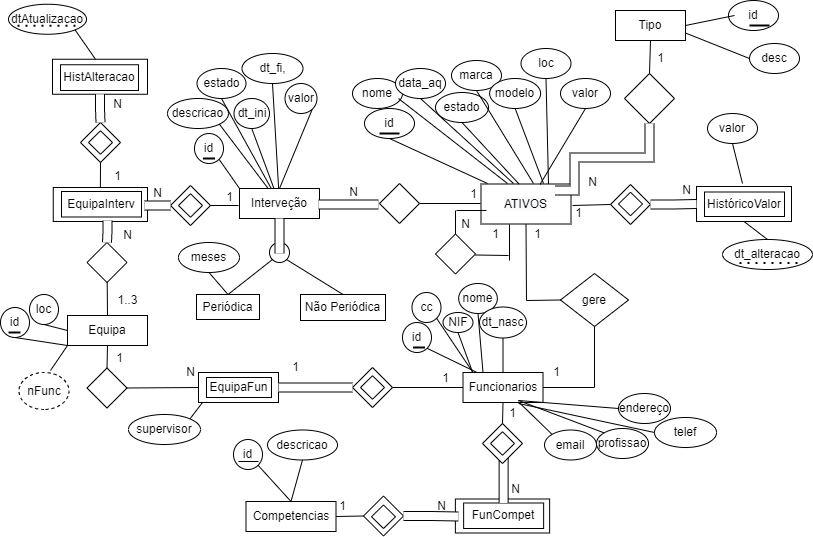


Figura 1 - Diagrama ER

# Correções da Fase 1

Com a implementação da aplicação, foram encontrados erros na resolução da primeira fase do projeto.

* Função obter equipa Livre -> Query corrigida porque não retornava sempre a equipa correcta.
* *Trigger [trg\_DeleteEquipaIntervencao]* -> Alteramos de “*For Delete”* para “*Instead of Delete”* porque existia dependencia da chave para a tabela “*dbo.HistAlteracaoEqInterv”*

# 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 contêm 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;

Devido à estrutura do problema e das opções implementadas, não foi implementada o padrão de desenho por Virtual Proxy.

# ADO.NET e Entiry FrameWork

As diferentes frameworks possibilitam duas maneiras diferentes de acesso aos dados:

* 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.
* Entity Framework -> 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.

# Desenho da Aplicação

# Componente *Fase2\_Project*

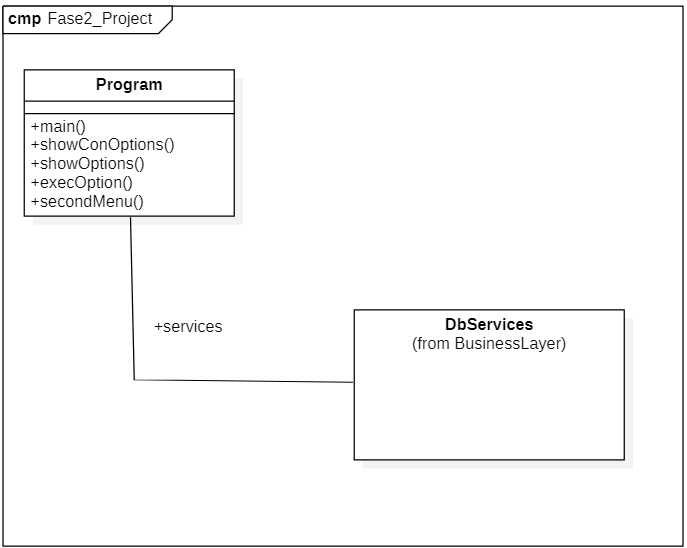


Figura 2 - Componente Fase2\_Project

# Componente *BusinessLayer*

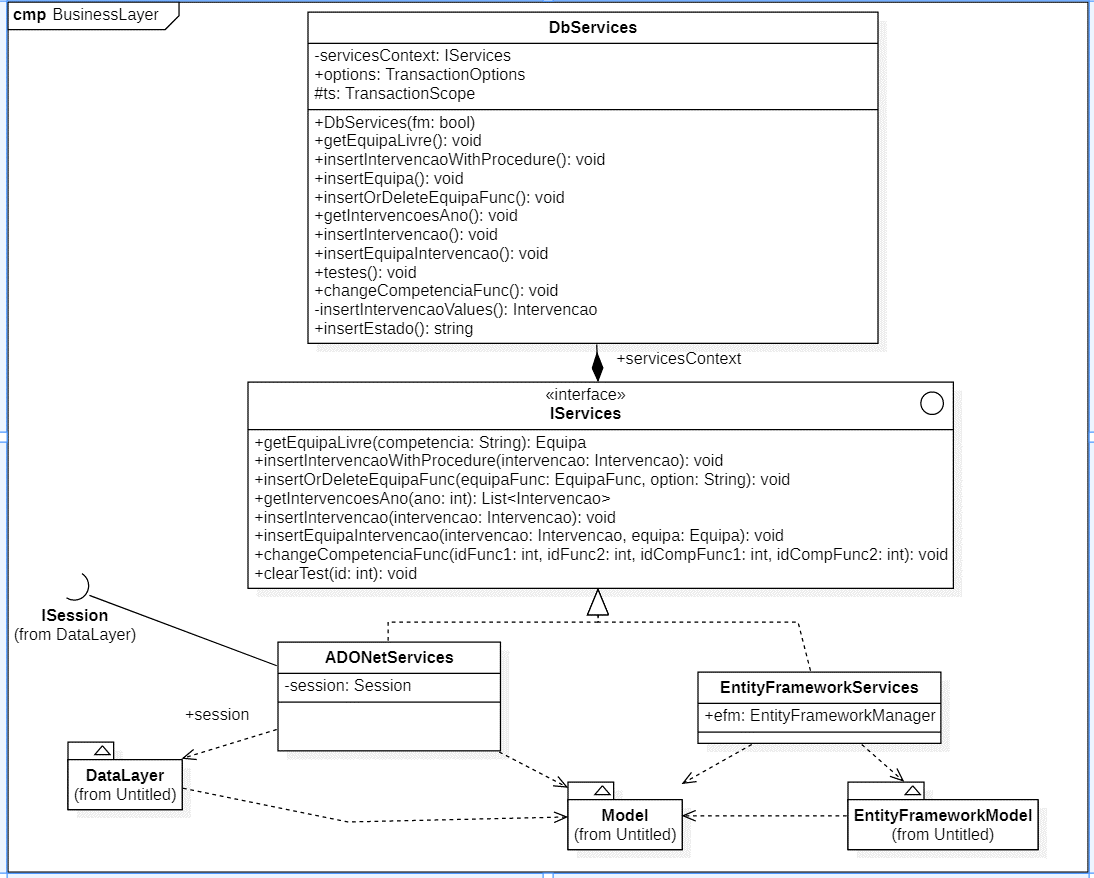


Figura 3 - Componente BusinessLayer

# Componente *DataLayer*

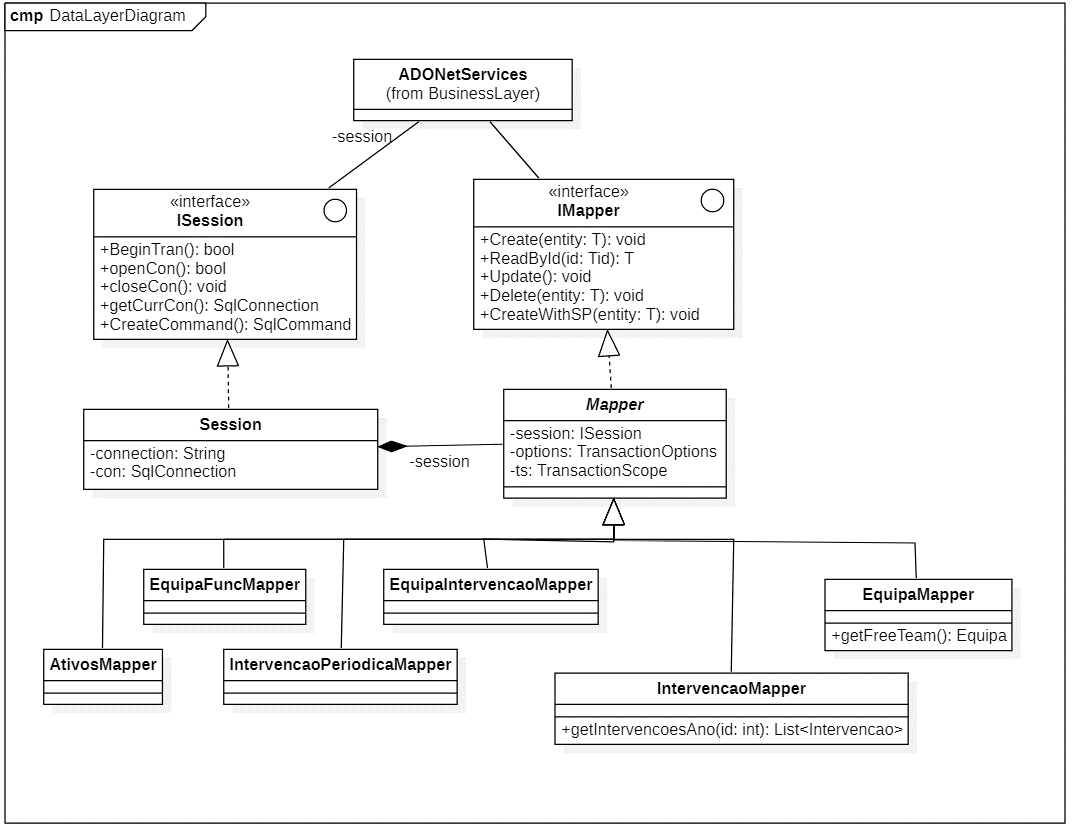


Figura 4 - Componente DataLayer

Para ligação com a base de dados 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.

Foi também criada uma Interface *IMapper* com os métodos *CRUD* (*Create*, *Read*, *Update* e *Delete*), sendo estes os básicos de um sistema que trabalha com *BDs*. Cada *Mapper* irá implementar esta interface

# Componente *EntityFrameworkManager*

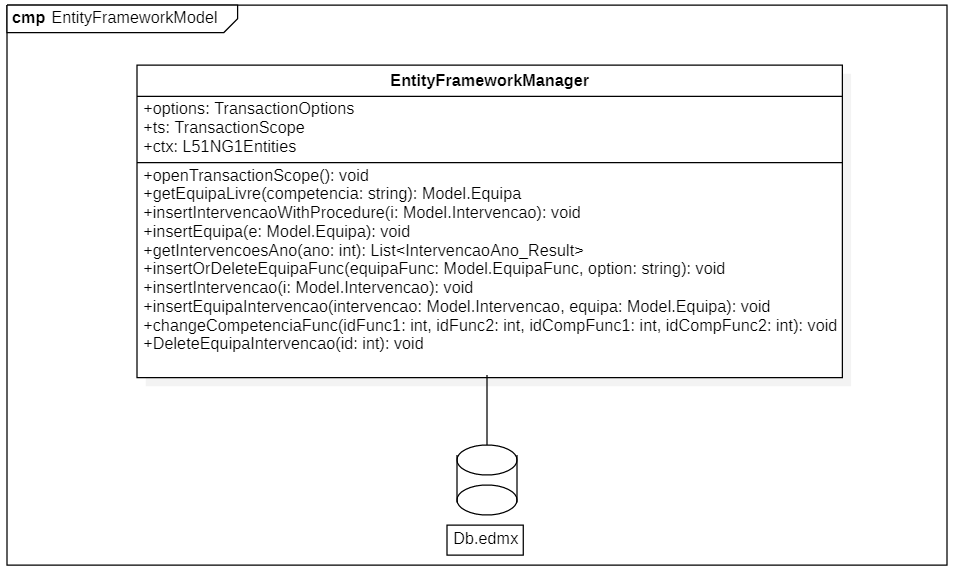


Figura 5 - Componente EntityFrameworkManager

# Componente *Model*

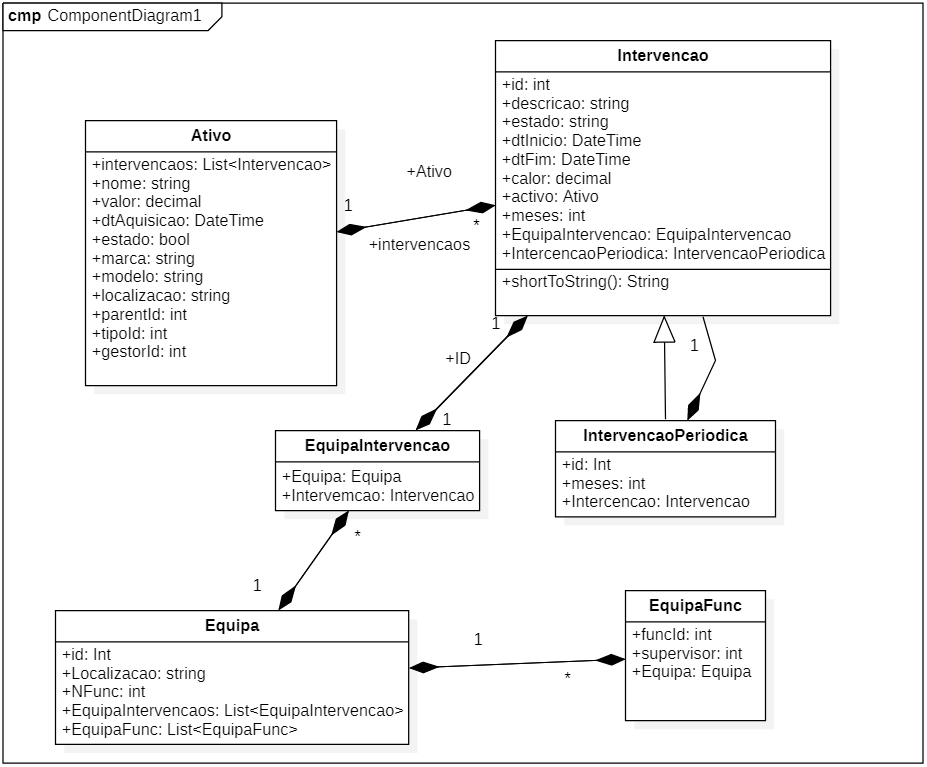


Figura 6 - Componente Model

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.