

**\*Título do Trabalho de Projecto  
Usando a Primeira Letra na Forma Maiúscula\***

|  |  |
| --- | --- |
|  | \*Fernando Pessoa\* |
|  | \*Ricardo Reis\* |

|  |  |
| --- | --- |
| Orientadores | \*Àlvaro de Campos\* |
|  | \*Alberto Caeiro, SoftCompany\* |

Relatório \*de progresso\* \*beta\* final\* realizado no âmbito de Projecto e Seminário,  
do curso de licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2014/2015

\*Maio\* de 2015

**Instituto Superior de Engenharia de Lisboa**  
Licenciatura em Engenharia Informática e de Computadores

**\*Título do Trabalho de Projecto  
Usando a Primeira Letra na Forma Maiúscula\***

|  |  |
| --- | --- |
| \*75463\* | \*Fernando António Nogueira Pessoa\* |
| \*72453\* | \*Ricardo Manuel Augusto dos Santos Reis\* |
|  |  |

|  |  |
| --- | --- |
| Orientadores: | \*Álvaro José Silva Veiga de Campos\* |
|  | \*Alberto Joaquim Alves Caeiro, SoftCompany\* |

Relatório \*de progresso\* \*beta\* final\* realizado no âmbito de Projecto e Seminário,  
do curso de licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2014/2015

\*Maio\* de 2015

# Resumo

O presente documento foi desenvolvido no âmbito da disciplina Sistemas de Informação II de Licenciatura do Curso de Engenharia Informática de Computadores. Trata-se de um relatório da Segunda Fase do Trabalho Prático que visa aplicar a matéria adquirida nas aulas sobre a linguagem SQL, mecanismos transacionais e camadas de acesso a dados com controlo transacional. Para tal, o projeto foi contextualizado num desenvolvimento de uma aplicação com acesso a um sistema de informação para a gestão de mercados financeiros.

Durante o seu desenvolvimento,

Breve descrição do projecto, dos resultados importantes e das conclusões: o objectivo é dar ao leitor uma visão global do projecto (não deve exceder uma página).

**Palavras-chave:**

**ACID: Atomicidade, Consistência, Isolamento e Durabilidade**

**ADO: ActivezX Data Object**

**GUI: Graphic User Interface**

**LINQ: Language-Integrated Query**

**ORM: Object Relational Mapping**

**UML: Unified Modeling Language**

# Abstract

Abstract text (1 page).

**Keywords:** sorted keyword list, delimited by ;.

**Índice**

[Resumo v](#_Toc28042015)

[Abstract vii](#_Toc28042016)

[Lista de Figuras xi](#_Toc28042017)

[1. Introdução 1](#_Toc28042018)

[1.1 Organização do documento 2](#_Toc28042019)

[2. Formulação do Problema 3](#_Toc28042020)

[2.1 Desenvolvimento da Segunda Fase – Questões surgidas 3](#_Toc28042021)

[2.2 Restrições de Interface do Utilizador 3](#_Toc28042022)

[2.3 Tecnologias usadas 4](#_Toc28042023)

[2.3.1 ADO.NET em modo “conectado” 4](#_Toc28042024)

[2.3.2 ADO.NET Entity Framework 5](#_Toc28042025)

[2.4 Contextualização 6](#_Toc28042026)

[3. Abordagens Gerais 7](#_Toc28042027)

[3.1 Abordagem dos problemas 7](#_Toc28042028)

[3.1.1 Interface do Utilizador 7](#_Toc28042029)

[3.1.2 Segurança de Informação sensível 7](#_Toc28042030)

[3.1.3 Estado da ligação 8](#_Toc28042031)

[3.2 Arquitetura Geral da Aplicação 8](#_Toc28042032)

[3.3 Sequência de Execução da Aplicação 9](#_Toc28042033)

[3.4 Restrição de Comandos 10](#_Toc28042034)

[4. Solução 1 – Aplicação com Menu de Configuração 11](#_Toc28042035)

[4.1 Restrições e constrições da solução 11](#_Toc28042036)

[4.2 Implementação da Aplicação 11](#_Toc28042037)

[4.3 Análise da Aplicação 13](#_Toc28042038)

[5. Solução Final – Duas Aplicações com Tecnologias Diferentes 14](#_Toc28042039)

[5.1 Implementação da Aplicação usando a Tecnologia ADO.NET em modo “conectado” 14](#_Toc28042040)

[5.2 Implementação da Aplicação usando a Tecnologia ADO.NET Entity Framework 16](#_Toc28042041)

[5.3 Análise das Aplicações 17](#_Toc28042042)

[5.3.1 Comparação das Tecnologias usadas 17](#_Toc28042043)

[A.1 Diagrama Sequencial da Solução 1 18](#_Toc28042044)

# 

# Lista de Figuras

[Figura 1 - Modelo Conectado e Desconectado 4](#_Toc28042059)

[Figura 2 - Arquitetura do ADO.NET Entity Framework [2] 5](#_Toc28042060)

[Figura 3 - Arquitetura geral implementada 9](#_Toc28042061)

[Figura 4 - Sequência de execução da aplicação 10](#_Toc28042062)

[Figura 5 - Comandos disponíveis para o utilizador 10](#_Toc28042063)

[Figura 6 - Arquitetura da Aplicação com um Menu de Configuração 12](#_Toc28042064)

[Figura 7 - Diagrama dos objetos usados na aplicação que utiliza ADO.NET em modo "conectado" 15](#_Toc28042065)

[Figura 8 - Diagrama dos objetos usados na aplicação que utiliza ADO.NET Entity Framework 16](#_Toc28042066)

# Introdução

No âmbito da disciplina Sistemas de Informação II, foi proposto fazer um trabalho composto por duas fases cujos os objetivos são:

* Utilizar a linguagem SQL a um nível avançado, incluindo SQL/PSM:
* Compreender e usar os mecanismos transacionais (ACID) percebendo como podem ser usados para lidar com a concorrência;
* Construir camadas de acesso a dados com controlo transacional;

Nesta fase, segunda, pretende-se:

* Desenvolver uma aplicação em C#;
* Utilizar processamento transacional, concretizado através de mecanismos disponíveis na plataforma .NET;
* Utilizar ADO.NET em modo “conectado”;
* Utilizar ADO.NET Entity Framework para acessos a dados;
* Garantir a correta libertação de ligações e recursos, quando estes não estejam a ser utilizados;
* Garantir a correta implementação das restrições de integridade e/ou lógica de negócio;
* Organizar o código de acesso a dados usando padrões de desenho como *Data Mapper* e *Lazy Load*;

Para tal, o trabalho foi contextualizado na necessidade de desenvolver um sistema, para a empresa *Pilim*, de informação para a gestão de mercados financeiros.

Na primeira fase do trabalho, foi projetado e construído um sistema de armazenamento e organização de dados em SQL, de modo a que todos os dados do sistema sejam consistentes com as necessidades da empresa.

Na segunda fase, foi desenvolvido uma aplicação que acede ao sistema anteriormente criado. Para o âmbito da disciplina, foram desenvolvidas duas aplicações com tecnologias diferentes de acesso. Na primeira implementação foi usado o ADO.NET em modo “conectado” e na segunda ADO.NET Entity Framework.

## 1.1 Organização do documento

O restante relatório é constituído por seis capítulos principais.

O primeiro capitulo trata-se da formulação do problema, onde é relatado todos os problemas encontrados no inicio e durante a execução da segunda fase do trabalho.

O segundo capitulo demonstra a arquitetura geral onde as soluções encontradas se vão basear. Também responde a alguns problemas que foram postos no capitulo anterior, pois, em todas as soluções propostas, todas aplicam o mesmo tipo de abordagem.

O terceiro capitulo descreve a primeira solução implementada e justifica a razão para qual a solução foi abandonada.

O quarto capitulo apresenta a solução final implementada para a finalização desta fase.

O quinto capitulo demonstra todas as análises efetuadas durante a implementação da solução implementada, manifestando que testes foram efetuados para garantir a boa execução da aplicação e responde também algumas perguntas mencionadas no enunciado desta segunda fase do trabalho.

No sexto e último capitulo são colocadas todas as conclusões que os alunos obtiveram durante a execução desta fase.

# Formulação do Problema

Pretende-se que nesta segunda fase do trabalho se crie aplicações que usem diferentes *frameworks* de acesso a dados. As aplicações devem ter como metas a reutilização de código, fácil de manutenção e eficiência, e serem independentes do modo de acesso a dados.

Na secção 2.1, relata-se as questões que houve no inicio e na implementação das aplicações desenvolvidas. Na secção 2.2, é exposto as restrições que a interface do utilizador pode ter. Na secção 2.3, é explicado as tecnologias que são usadas para o desenvolvimento da aplicação. Na secção 2.4, é esclarecido o contexto do trabalho.

## 2.1 Desenvolvimento da Segunda Fase – Questões surgidas

Ao analisar o enunciado do trabalho, vários problemas surgiram para desenvolver as aplicações como:

* Vai ser necessário modificar alguma implementação realizada na fase anterior?
* Fazer uma aplicação em consola ou numa GUI?
* Para aceder aos dados, o utilizador introduz as suas credenciais? As credenciais estão guardadas num ficheiro proprietário? Ou as credenciais estão já guardadas no código (*hardcoded)*?
* No inicio da aplicação, testa-se o acesso aos dados?
* Faz-se uma aplicação que tenha um menu de configuração onde o utilizador consegue escolher que tipo de tecnologia a aplicação usa ou duas aplicações com tecnologias diferentes?

Estas foram umas de muitas questões que surgiram no inicio da implementação, no entanto outras foram surgindo durante a implementação

.

## 2.2 Restrições de Interface do Utilizador

Enquanto se planeava a interface que o utilizador iria ter com a aplicação para aceder aos dados do sistema, certos problemas surgiram, como:

* O utilizador interage diretamente com o sistema de armazenamento de dados, podendo construir as suas próprias *queries*?
* Deixamos o utilizador restringido a certos comandos sendo depois a responsabilidade de um programador futuro acrescentar outros comandos que o utilizador precise?

## 2.3 Tecnologias usadas

Para ir de encontro às metas e objetivos anteriormente delineados, foi necessário fazer um estudo das tecnologias que são necessárias usar, ADO.NET em modo “conectado” e ADO.NET Entity Framework.

Tradicionalmente, o processamento de dados baseia-se principalmente em um modelo de duas camadas baseado em conexão. À medida que o processamento de dados utiliza cada vez mais arquiteturas de várias camadas, os programadores estão a mudar para uma abordagem desconectada para fornecer melhor escabilidade para as suas aplicações [1].

### 2.3.1 ADO.NET em modo “conectado”

O modo conectado do ADO.NET é um modo que foi explicitamente desenhado para manipulação de dados, apenas leitura e um acesso rápido a dados. Como é ilustrado na Figura 1, este modo é composto por um objeto *Connection*, que providencia conectividade à fonte de dados, um objeto *Command*, que permite o acesso aos comandos da base de dados para retornar e modificar dados, executar procedimentos e enviar ou recuperar informações de parâmetros, um objeto *Data Reader*, apresenta uma *Stream* de alta performance de dados da fonte de dados, e um objeto *Data Adapter* que promove uma ligação ponte entre a fonte e dados e um objeto *Data Set* [1].

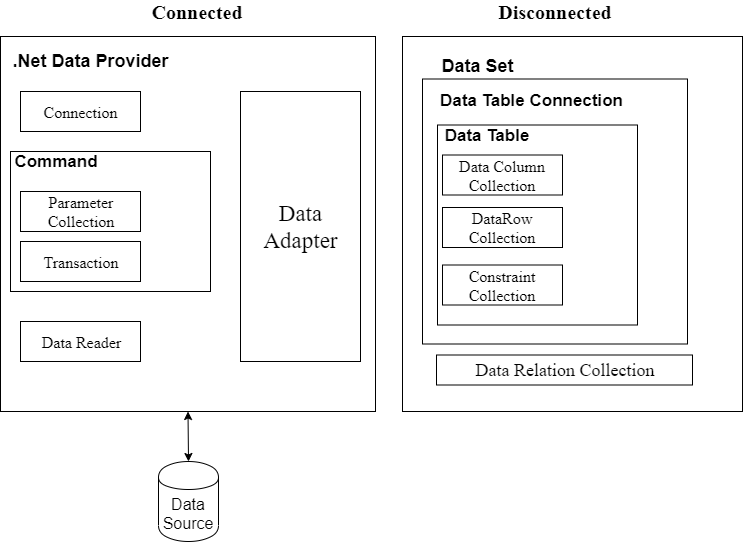


Figura - Modelo Conectado e Desconectado

No entanto, como pedido no enunciado, só vai ser utilizado o ADO.NET em modo “conectado”, o que implica que não se vai usar objetos do modo “desconectado”. Assim, não vai ser utilizado objetos *Data Adapter*.

### 2.3.2 ADO.NET Entity Framework

O Entity Framework é uma ferramenta da Microsoft, pertencente ao pacote ADO.NET, que permite ao programador trabalhar com banco de dados relacional de maneira abstrata [2].

Esta ferramenta utiliza uma técnica para diminuir a complexidade e o constrangimento da programação orientada a objetos com bases de dados relacionais denominada por ORM (*Object Relational Mapping*) ou mapeamento de objeto relacional. A partir desta técnica é possível apresentar tabelas como classes e os registos de cada tabela como registos nas classes correspondentes.

Com esta ferramenta, é possível desenvolver aplicações com uma estrutura em camadas e, assim, o programador pode focar mais nas regras do negócio, abstraindo da camada da base de dados, como é representado na Figura 2.

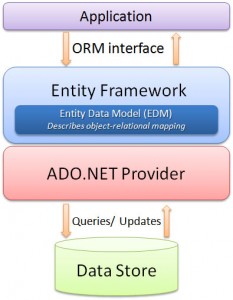


Figura - Arquitetura do ADO.NET Entity Framework [2]

Para trabalhar com o Entity Framework, existem 3 possíveis abordagens, *Database-First*, *Model-First* e *Code-First*.

Na primeira abordagem, a base de dados foi criada previamente e as entidades são surgidas baseadas na base de dados. Na segunda abordagem, é criado um modelo conceitual primeiro e através deste é gerado um script para criar a base de dados. Na terceira abordagem, são criadas classes primeiro e através destas toda a lógica da entidade é criado, permitindo que com este modelo seja gerado a base de dados.

## 2.4 Contextualização

O presente trabalho contextualiza-se na necessidade de desenvolver um sistema de informação para a gestão de mercados financeiros para a empresa *Pilim*.

Este sistema é constituído por um mercado financeiro. Um mercado financeiro pode ser constituído por um conjunto de instrumentos. Tanto os instrumentos como os mercados financeiros têm um registo diário com registos próprios e fundamentais para a sua avaliação quantitativa e qualitativa. Existe também um conjunto de informação denominada por triplos onde identifica um instrumento existente ou inexistente na base dados com informação do seu valor numa certa altura temporal.

O sistema também contém um armazenamento de clientes que têm contactos, email e/ou telefónico. Cada cliente pode ter também um ou mais portfólios. Cada portfólio pode ter um certo número de ações ou posições de um ou vários instrumentos.

Nesta segunda fase do trabalho, é preciso desenvolver uma aplicação para aceder a estes dados, obedecendo às necessidades e regras que a empresa impõe.

# Abordagens Gerais

Para ir ao encontro das metas e objetivos, foi pensado desenvolver uma única aplicação com um menu de configuração ou duas aplicações com configurações diferentes. Estas configurações definem que tipo de ferramentas e tecnologias vão ser usadas, ADO.NET em modo “conectado” e ADO.NET Entity Framework.

Apesar de serem soluções um pouco diferentes, as suas abordagens e arquiteturas são muito semelhantes, apenas diferenciando pela sua maneira de aplicar. Esta arquitetura demonstra onde as duas soluções se vão basear.

## 3.1 Abordagem dos problemas

Nesta secção, é explicado que tipos de abordagens são implementadas para resolver alguns problemas expostos no capitulo anterior. Estas são abordadas da mesma maneira em ambas as soluções.

### 3.1.1 Interface do Utilizador

O tipo de interação que um utilizador tem com a aplicação é um dos componentes mais importantes que uma aplicação pode ter, pois é partir desta interface que a aplicação executa as suas funções. Se uma aplicação não tiver uma interação humana, então deixa de ser aplicação.

Entende-se como interação humana uma inserção de uma ação na aplicação ou uma resposta da mesma. Este tipo de interação pode ser feito de várias maneiras, sendo duas delas expostas nos problemas encontrados no inicio do desenvolvimento desta fase. A aplicação interage com o utilizador a partir de uma GUI ou de uma consola? Apesar de uma GUI ser mais apelativo e intuitivo para o utilizador, foi decidido que o utilizador interaja a partir de uma consola, pois, não havia tempo, nem experiência, nem mão de obra para implementar uma GUI, sendo mais fácil e fazível implementar uma consola a tempo da entrega desta fase.

### 3.1.2 Segurança de Informação sensível

Como foi referido no capitulo anterior, esta aplicação tem acesso a dados a clientes e também algumas credenciais com conteúdo sensível e privado. De modo a proteger o sistema, privacidade e informação restrita, foi implementado certas restrições no desenvolvimento das soluções.

A primeira restrição são as credenciais de acesso à base de dados. Estas credenciais são as chaves para aceder aos dados do sistema e, como tal, são de teor sensível e privado. Por isso, estas credenciais têm que ser obrigatoriamente inseridas pelo utilizador.

A segunda restrição é o modo de interação que o utilizador tem com a consola. Ao desenvolver uma aplicação, o programador nunca sabe que tipo de utilizador vai estar a interagir com a aplicação. Pode ser um utilizador bem formado com boas ou más intenções ou mesmo um utilizador sem experiência informática. Por isso cabe ao programador oferecer uma interação simples e restrita para evitar fuga de informação acidental ou intencional. Com isto, em ambas as soluções, o utilizador tem apenas acesso a um certo número de comandos com uma interação fácil e explicita.

### 3.1.3 Estado da ligação

Ao desenvolver qualquer tipo de aplicação ou dispositivo que requer uma ligação externa a dados ou informação, é preciso ter a certeza que não existem problemas nas comunicações feitas, pois, estas podem induzir em erro com falsa informação à aplicação ou à própria base de dados. Por isso, é necessário testar a ligação, verificando se as credenciais estão corretas e se a base de dados está a funcionar corretamente.

## 3.2 Arquitetura Geral da Aplicação

Nesta secção, será descrito a arquitetura que ambas as soluções implementam.

A arquitetura implementada, é constituída por três camadas principais, a aplicação, onde está presença toda a lógica de negócio, a fornecedor de dados, que fornece à aplicação dados e permite que a aplicação forneça dados para a camada de baixo, e a fonte de dados, onde é armazenado todos os dados.

Em cada camada principal, existe sempre subcamadas que garantem o bom funcionamento interno para qual a camada principal é responsável e, também, outras subcamadas que garantem a receção e a passagem de dados para camadas superiores ou inferiores. A Figura 3 demonstra a existência de uma dessas subcamadas na camada da aplicação. A subcamada *Pilim APP*, é o ponto de acesso à aplicação no qual a *Virtual Machine* chama a função *Main*. Esta subcamada, depois, chama o objeto responsável por correr a lógica de negócio e envia informação ao fornecedor de dados. A partir dessa informação, o fornecedor de dados envia e recolhe dados à fonte de dados.

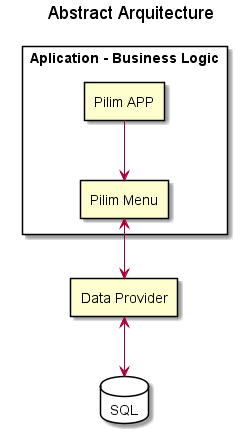


Figura - Arquitetura geral implementada

## 3.3 Sequência de Execução da Aplicação

Nesta secção é exposto, no geral, como a aplicação se comporta durante o seu tempo de execução em ambas as soluções.

Na Figura 4, é demonstrado a sequência da aplicação. Esta inicia-se com o *Login* abrindo a consola para inserir as credenciais do utilizador, passando ao teste da conexão, caso seja bem-sucedida entra no menu ficando lá até escolher a opção *Exit*. No *Menu* é possível executar um diverso de opções que caso sejam válidas de efetuar são executadas mostrando os resultados da mesma. Caso o comando não seja válido é emitida uma mensagem de erro voltando ao *Menu*.

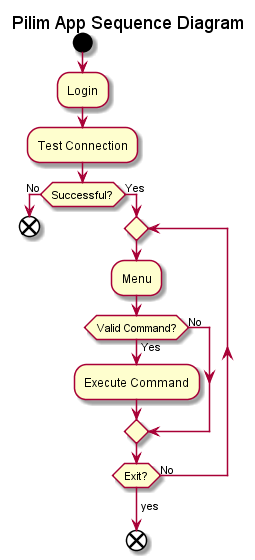


Figura - Sequência de execução da aplicação

## 3.4 Restrição de Comandos

Como referido anteriormente, na implementação da aplicação, o utilizador vai estar restrito a certos comandos, não podendo usufruir ao máximo toda a informação presente na base de dados. No entanto, no futuro, pode existir a necessidade de implementar certos comandos que anteriormente não foram implementados. Para isso acontecer, é necessário implementar um modo de implementação de comandos que seja igual para qualquer comando. Este modo tem que ser simples e não pode modificar qualquer tipo de implementação anteriormente feita. Para tal, basta acrescentar uma opção no enumerável *Option*, criar e implementar a função e adicioná-la ao dicionário *\_\_methods*;

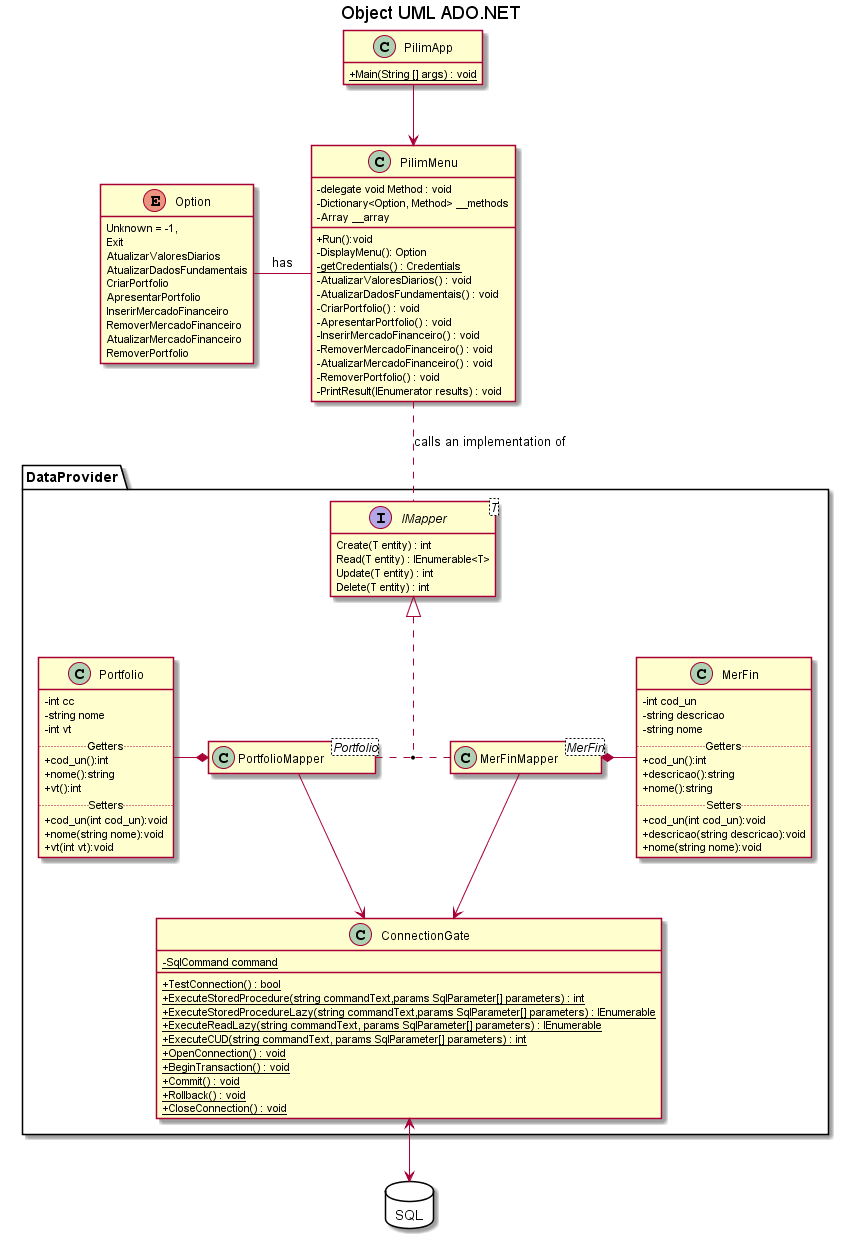


Figura - Comandos disponíveis para o utilizador

# Solução 1 – Aplicação com Menu de Configuração

Esta solução, serviu para ir de encontro às metas e objetivos que foram propostos no inicio desta fase. Foi pensado que o utilizador poderia escolher que tipo de tecnologia a aplicação usaria para o acesso à fonte de dados. A partir desta, era possível demonstrar o uso das tecnologias e das ferramentas que o enunciado pede para usar.

## Restrições e constrições da solução

Enquanto esta solução foi pensada, houve certas restrições e constrições que surgiram.

Primeiro, a solução tem que ser modular e a sua manutenção seja simples e fácil, pois para a conclusão desta fase do trabalho, só é pedido que um certo número de comandos seja executado. Quer dizer que, se o utilizador quiser utilizar a aplicação e precisa de usar um comando não existente, a implementação do mesmo tem que ser simples sem que seja necessário modificar qualquer implementação anteriormente feita.

Segundo, os objetos que representam as entidades na base de dados têm que ser os mesmos para as ambas tecnologias, pois a escolha de diferentes tecnologias não pode nem deve influenciar a constituição de uma entidade. Estas têm que ser independentes da tecnologia usada.

Terceiro, a abordagem de execução de comandos tem que ser o mais generalista possível, pois, a verificação do uso da tecnologia para poder executar um comando especifico requer um custo de performance. Este custo pode ser reduzido se utilizar a arquitetura da linguagem em favor da aplicação.

## Implementação da Aplicação

Para implementar a aplicação, reparou-se que era necessário um certo número de interfaces, para garantir que a implementação da aplicação seja o mais abrangente possível. Por isso certas interfaces, como *IConfig* e *IEntities*, foram criadas.

As configurações ADO.NET em modo “conectado” e ADO.NET Entity Framework implementam a interface *IConfig*, e o tipo de configuração é perguntado ao utilizador logo no inicio do programa. Com isto, é instanciado uma implementação de *IConfig* que está guardado no *Data Provider*. Assim quando o *Menu* manda o *Data Provider* para executar um comando, este pede à configuração presente para executar esse mesmo comando, deste que este seja de acesso à base de dados.

A arquitetura desta solução, como referido anteriormente, é baseada numa arquitetura geral composta por camadas. Na Figura 5 é possível verificar a arquitetura desta solução e que a sua base é proveniente da arquitetura geral. As configurações ADO.NET em modo “conectado” e ADO.NET Entity Framework estão representados na figura como *Mapper* e *DBSet* respetivamente.

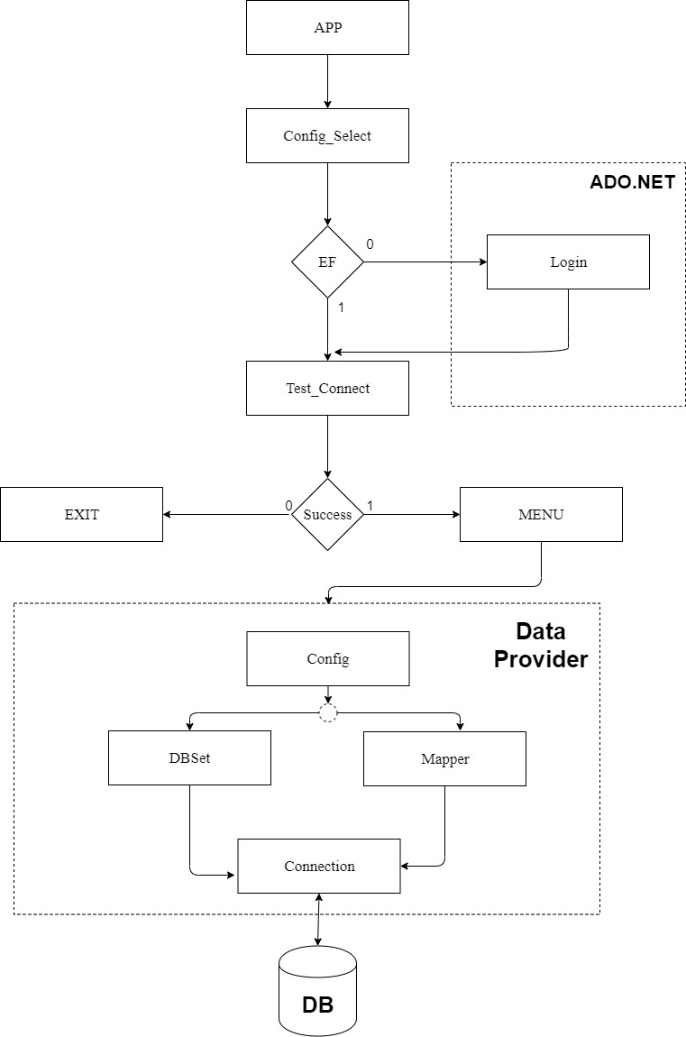


Figura - Arquitetura da Aplicação com um Menu de Configuração

## Análise da Aplicação

Apesar desta implementação tenha sido executada com sucesso até a meio da implementação da configuração ADO.NET Entity Framework, notou-se que foi necessário o uso de certas tecnologias que influenciam a performance da aplicação, como a reflexão. Este tipo de tecnologia foi utilizado para se poder aceder a um mapa de *Data Mappers* que por sua vez executam um comando de acesso a base de dados.

Os problemas de implementação começaram a surgir quando se percebeu que ia ser necessário o uso do *LINQ* *queries* juntamente com a reflexão. Como se reparou que este tipo de abordagem ia aumentar a complexidade da solução e que por sua vez até podia ser impossível de implementar, foi decidido abandonar esta solução e passou-se a implementar a solução com duas aplicações em que cada uma utiliza uma tecnologia de acesso a dados diferente, baseando-se na arquitetura geral definida no inicio da segunda fase do trabalho.

# Solução Final – Duas Aplicações com Tecnologias Diferentes

Como referido anteriormente, o abandono da primeira solução significa que foi necessário migrar algumas classes já criadas, visto que o único componente diferente implementado foi a camada *Data Provider*.

Como nesta solução cada aplicação especificamente utiliza uma tecnologia diferente uma da outra, já não se tem tantas constrições nem restrições como anteriormente encontramos, no entanto, a modularidade, a escabilidade e a implementação de novos comandos têm que permanecer a mesma.

## 5.1 Implementação da Aplicação usando a Tecnologia ADO.NET em modo “conectado”

Para implementar a aplicação com a tecnologia ADO.NET em modo “conectado”, foi necessário criar um ponto de entrada para a aplicação, *Pilim App*, no qual interage com um *Menu* que irá chamar e interagir com o *Data Provider* através de um conjunto de opções possíveis.

O *Menu* contém várias opções de tudo o que é possível efetuar na base de dados, como criar um *Portfolio*, atualizar os valores diários e os dados fundamentais de um mercado financeiro e inserir novos mercados. Foi implementado desta maneira para que seja fácil de criar novas opções para o menu, tornando-o independente das suas opções.

O *Data Provider* contém uma *Connection Gate* que estabelece a conexão com a base de dados e irá interagir com a mesma executando os comandos pedidos pelo utilizador, podendo executar *Stored Procedures*, *commit*, *rollback*, *create*, *update*, *delete*, *read*. Foi criada uma classe que representa o *Portfólio* e o *Mercado* *Financeiro* como objetos contendo as suas propriedades e atributos.

Para a interação com a aplicação criou-se uma interface *IMapper* que está preparado para executar operações *create*, *read*, *update* e *delete* a qualquer tipo de entidade. *PortfolioMapper* e *MerFinMapper* implementam *IMapper* efetuando essas mesmas operações aos *portfolios* e *mercados financeiros* que recebem respetivamente. Esta implementação é ilustrada na Figura 7, onde é demonstrado um UML dos objetos utilizados.

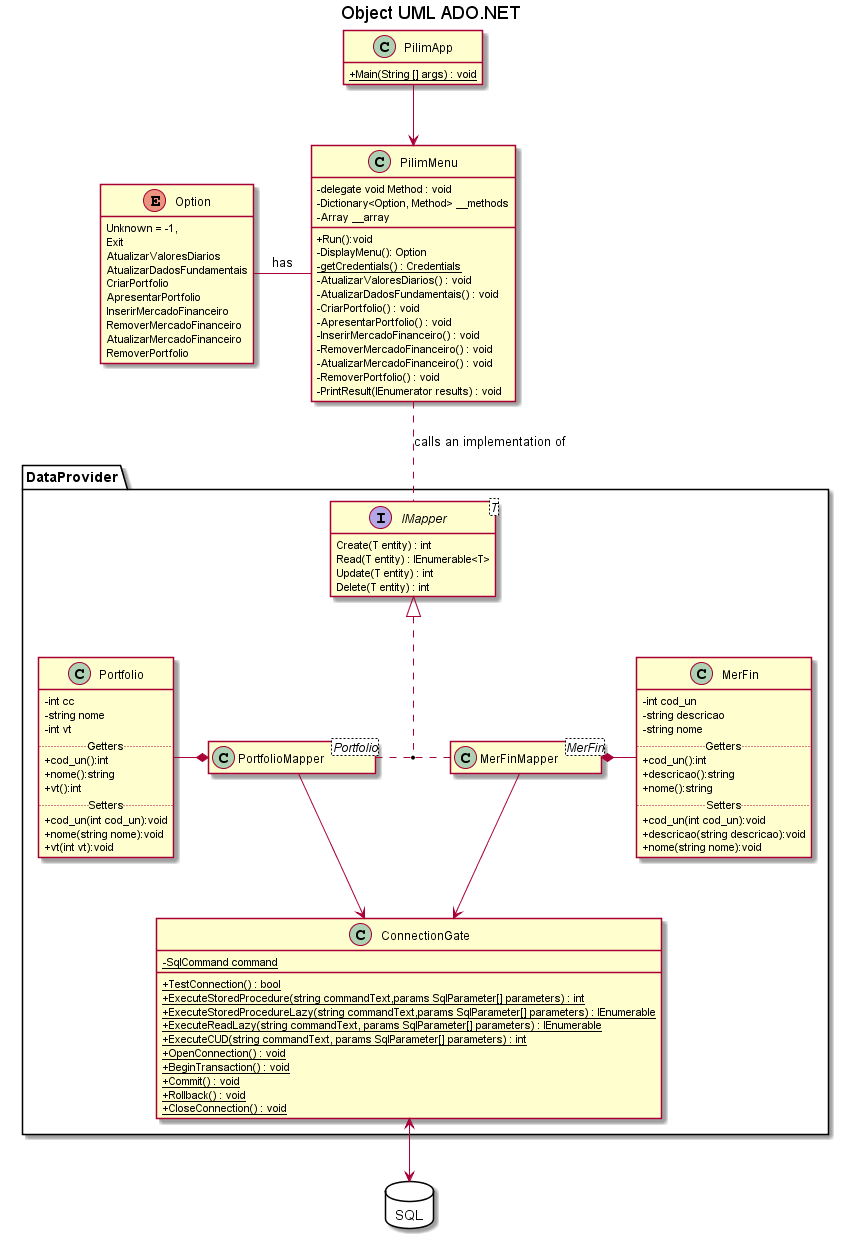


Figura - Diagrama dos objetos usados na aplicação que utiliza ADO.NET em modo "conectado"

## Implementação da Aplicação usando a Tecnologia ADO.NET Entity Framework

Para implementar a aplicação com a tecnologia ADO.NET Entity Framework, tal como na aplicação anterior, a camada da aplicação mantem-se a mesma, como se pode comparar a partir das Figuras 7 e 8.

O *Data Provider* que irá interagir com o *Menu* consoante a opção do mesmo for escolhida pelo utilizador. Este contem *PilimEnteties*, uma extensão de *DBContext*, que contem um *DBSet<T>* para cada tabela da nossa base de dados, pois será esta a classe que irá interagir com a base de dados. É nesta classe que são criadas algumas funções para conseguir executar *Stored Procedures*,que estão armazenadas na base de dados. Há opções do *Menu* que usufruem estas funções tais como: listar portfolio, inserir registo diário automático, atualiza dados instrumentos e atualiza valor diário.

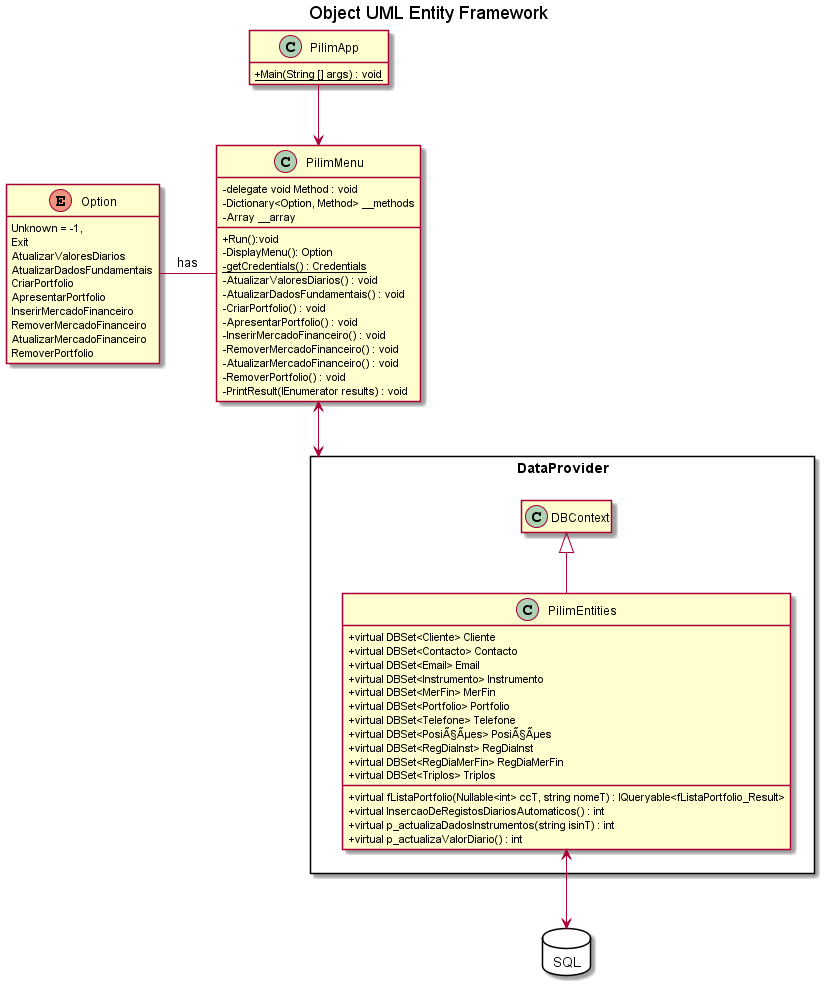


Figura - Diagrama dos objetos usados na aplicação que utiliza ADO.NET Entity Framework

## Análise das Aplicações

Após a implementação das aplicações e dos seus testes, foi notado algumas diferenças entre as duas tecnologias.

A primeira diferença notada durante a implementação das aplicações foi a maneira como as tecnologias são implementadas. O ADO.NET em modo “conectado” como é constituído por *Data Mappers*, requer a criação de objetos que implementem *IMapper*, enquanto que com a tecnologia ADO.NET Entity Framework, no melhor dos casos, é possível executar um comando a partir de uma *LINQ query*. Por isso, em termos de implementação de comandos, o ADO.NET Framework é mais simples e rápido de utilizar.

A segunda diferença notada foi durante os testes das aplicações, a aplicação ADO.NET Entity Framework demorava um algum tempo a apresentar valores pela primeira vez, pois o *DBSet* dessa tabela precisava de ser mapeada primeiro, enquanto que na outra aplicação, o tempo de apresentação era quase sempre o mesmo, fora o tempo perdido por causa da conexão, independentemente do número de vezes que se executava o comando.

### 5.3.1 Comparação das Tecnologias usadas

A maior vantagem do ADO.NET Entity Framework é que oferece um maior nível de abstração para trabalhar com a informação, isolando o desenvolvimento da aplicação do modelo de dados.

O mapeamento entre objetos e bases de dados é especificado declarativamente em vez de estar no código, se precisar de fazer uma mudança no esquema da base de dados, podemos minimizar o impacto no código a ser modificado, podendo na mesma fazer *queries* especificas ou operações de dados sem perder a abstração

O ADO.NET em modo “conectado” dá uma sintaxe comum para criar conexões e executar *queries* e processar resultados, mas caso estivéssemos a fazer *queries* em Oracle e depois passássemos para *SQL Server* teríamos que mudar algumas *queries* (pois variam consoante a base de dados utilizada). Esta tecnologia também permite *asyncpatter* o que pode ser bastante útil em alguns casos como beneficiar a responsividade do lado do cliente.

No ADO.NET Entity Framework, as *queries* são escritas em LINQ ou Entity SQL e traduzidos em *runtime* pelo *provider* para uma base de dados em particular.

# A.1 Diagrama Sequencial da Solução 1