Projeto de Arquiteturas de Software

Plataforma de Trading

Relatório de Desenvolvimento

José André Martins Pereira
(a82880@alunos.uminho.pt)

Ricardo André Gomes Petronilho
(a81744@alunos.uminho.pt)

26 de Outubro de 2019

Resumo

No Mestrado integrado de Engenharia Informática, na Unidade Curricular de Arquiteturas de Software foi-nos proposta a elaboração de um projeto de desenvolvimento de uma plataforma de trading.

Conteúdo

Capítulo 1

Introdução

No Mestrado integrado de Engenharia Informática, na unidade curricular de Arquiteturas de Software foi-nos proposta a elaboração de um projeto para o desenvolvimento de uma plataforma de trading.
A plataforma de trading tem como objetivo a realização de contratos de diferença (Contrats for Differences – CFD), podendo estes ser de venda ou compra de ativos.
Um **CFD de compra**, também designado CFD long deverá ocorrer quando o utilizador tem a expectativa que o preço atual de um determinado ativo vai aumentar no futuro e fechar o contrato, nessa mesma altura, para obter a diferença.
Um **CFD de venda**, também designado **CFD long** deverá ocorrer quando o utilizador tem a expectativa que o preço atual de um determinado ativo vai diminuir no futuro e fechar o contrato, nessa mesma altura, para obter a diferença.
Ao longo deste relatório, especifica-se o processo de desenvolvimento da arquitetura para esta plataforma de trading.

Capítulo 2

Desenvolvimento da Arquitetura

2.1 Funcionalidades

- Os traders/investidores podem criar uma conta com um plafond inicial para negociação/investimento.
- Os traders/investidores podem adicionar/remover fundos ao seu plafond.
- Os traders/investidores podem abrir posições (CFDs) sobre os ativos disponíveis, posições de venda e compra.
- Os traders/investidores podem encerrar um contrato por ação:
  Direta – encerra a posição (CFDs) no momento.
  Indireta - definir para as posições de venda e compra abertas limites de perda e ganho - Take profit (TP) e Stop Loss (SL)).
- Os traders/investidores pode desativar o encerramento por ação indireta (remover os limites de SL ou TP).
- Os traders/investidores deverão poder monitorizar em 'tempo real' o seu portfolio atual (e de outros caso os mesmos consintam) de CFDs.

2.2 Mockups

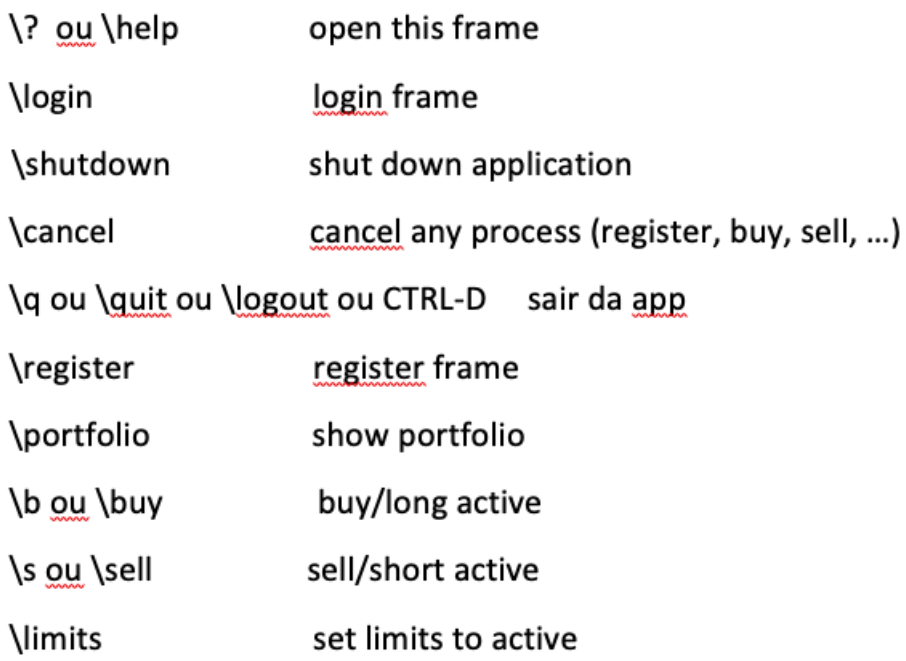

Figura 2.1: escrever barra help lista todas as opções/funcionalidades da aplicação.

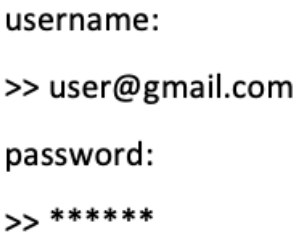

Figura 2.2: Login do utilizador.

username:

>> user@gmail.com

password:

>> ******

confirm password:

>> ******

initial investment (€):

>> 200.00

Figura 2.3: Registo do utilizador.

```
Plafond: 1 000 €
Total balance: 200,50 €
Id | Organization | Type | Units | Initial price (€) | Current Price (€)  | Invested | Balance (%) | Balance (€) | value
_____________________________________________________________________________________________________________________
0 | APPL.         | Buy  | 5     | 236.00 (B)        | 550.00 (S)         | 1180.00  | +0.48%      | +30.00      | 580.00
1 | BIT.          | Sell | 5     | 25.00 (S)         | 550.00 (B)         | 1500.00  | +0.48%      | +30.00      | 580.00
>>
```

Figura 2.4: Consultar portfolio.

[Buy/long active] Organization:

>> APPL

[Buy/long active] Current price (€): 236.00 / 237.00

[Buy/long active] Units (empty if not usable):

>> 75

[Buy/long active] Amount (€) (empty if not usable):

>>

[Buy/long active] CFD balance: +1750 (€)

Figura 2.5: Fazer um CFD de compra.

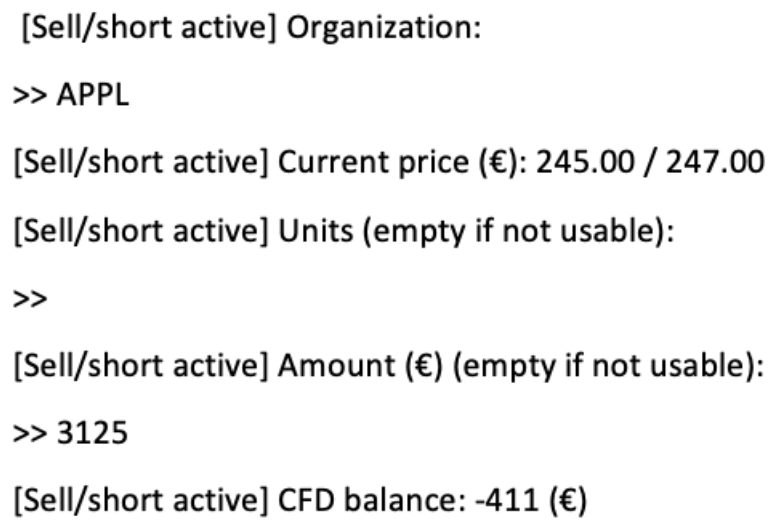

Figura 2.6: Fazer um CFD de venda.

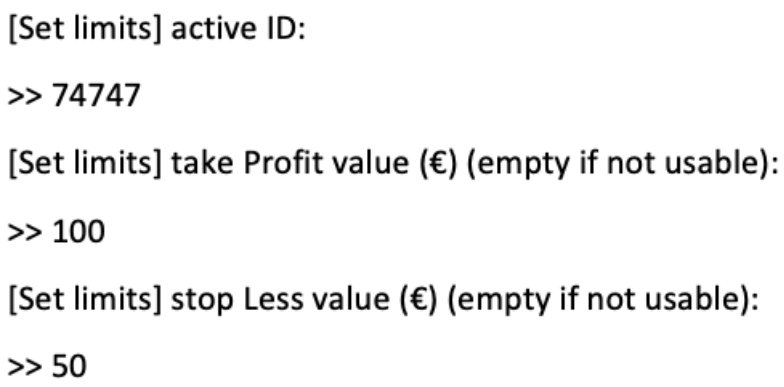

Figura 2.7: Definir limites de stop less e take profit.

2.3 Modelo Domínio

Devido às dimensões do modelo domínio, vamos apresentar o mesmo por partes. O primeiro subsistema que se vai abordar é o intitulado de recursos humanos, ou seja, responsável pelos utilizadores deste sistema.
Todo o utilizador contém um username e password, pois são os dados comuns aos diferentes tipos, no entanto, achou-se importante distinguir dois tipos de utilizador do sistema, o administrador e trader, este último tem ainda associado um plafond, ou seja, o valor em carteira na aplicação.
O trader pode assumir dois comportamentos no sistema, o de comprador e vendedor de ativos.
Por fim, e não menos importante, o utilizador pode consultar a lista de ativos existentes e a respetiva informação (valor atual, designação, ...), bem como o seu portfolio, ou seja, o conjunto de contratos realizados pelo mesmo, sendo que a associação do portfolio aos CFDs não é visível na imagem abaixo.

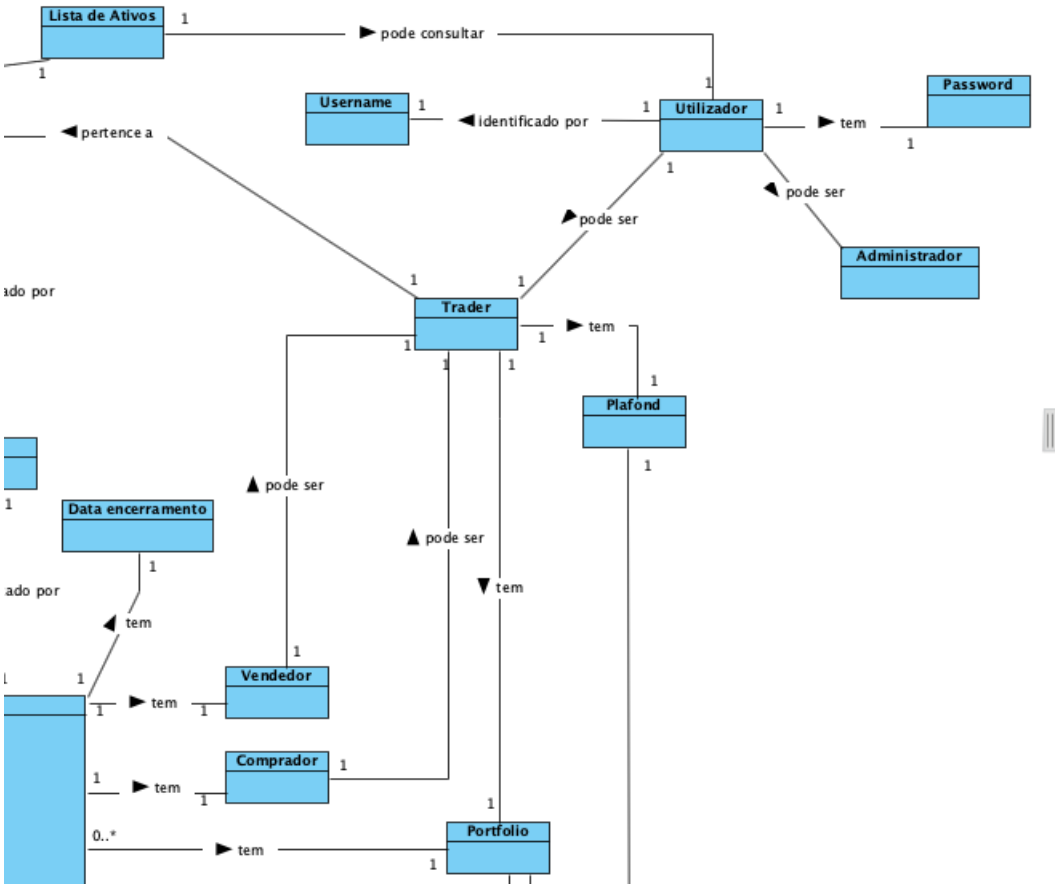

Figura 2.8: Subsistema dos recursos humanos.

O ativo tal como se pode verificar na figura abaixo, é identificado por um **id** (*APPL.*), contém uma **designação**, normalmente associado à organização (*APPLE*), dois tipos de valor, **valor de compra** e **venda**, e pode ser dividído em dois tipos:

- Ação
- Commodities

O ativos do tipo **ação** está associado a uma organização enquanto que o **commodities** está a um recurso natural. Também é importante referir que vai existir uma *Lista de ativos*, onde estão as informações de cada ativo.

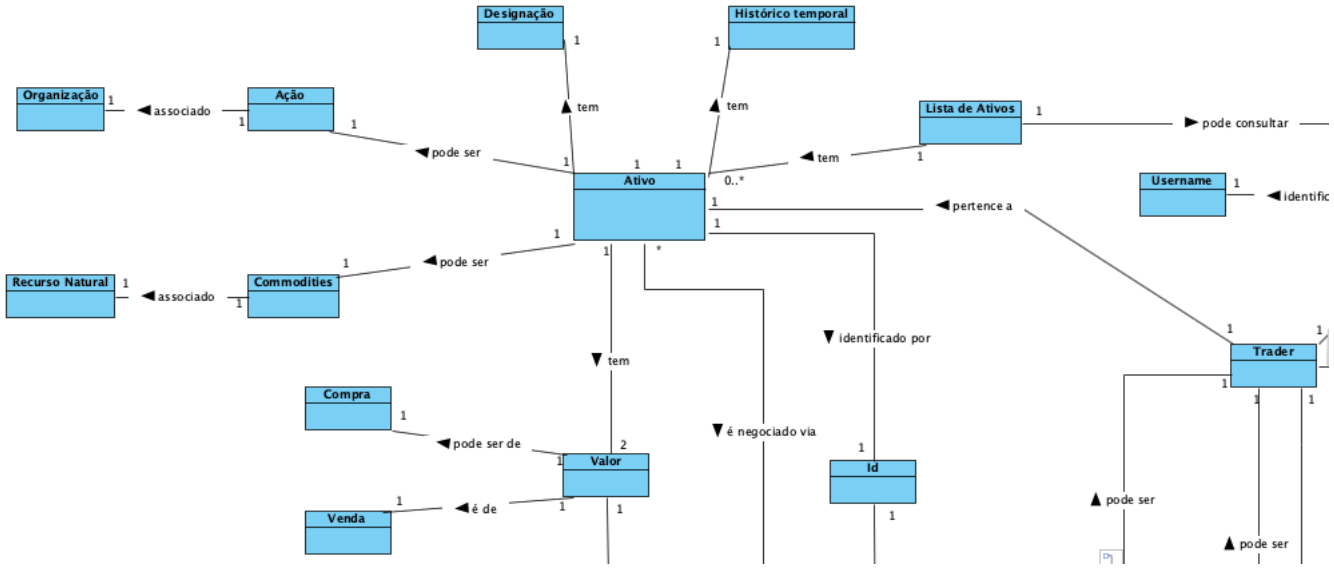

Figura 2.9: Subsistema do ativo.

Tal como se pode verificar na figura abaixo, o CFD é um contrato de diferenças de uma determinada quantidade de ativos associado a uma organização ou recurso natural, sendo este identificado por um **id**.
Deste modo, existem dois tipos de **CFDs**, o **Long** e **Short**, que são respetivamente o **CFD de venda** e **compra**.
Tal como o **ativo** o **CFD** também tem um valor de venda e compra, mas acresce ainda os campos de **valor investido** (total do valor dos ativos) e um **profit**, que são o agregado dos valores dos ativos.
Existem duas formas de encerramento do **CFD**, por ação direta ou indireta. A ação direta, tal como o nome sugere, o encerramento é deliberado pelo utilizar no instante que a executar, a ação indireta é realizada através do estabelecimento de limites de **Stop less** e **Take profit**, tal como se pode verificar na figura abaixo.
Por fim, e não menos importante, através do ato de encerramento descrito anteriormente, o CFD tem dois estados possíveis, **aberto** e **encerrado**.

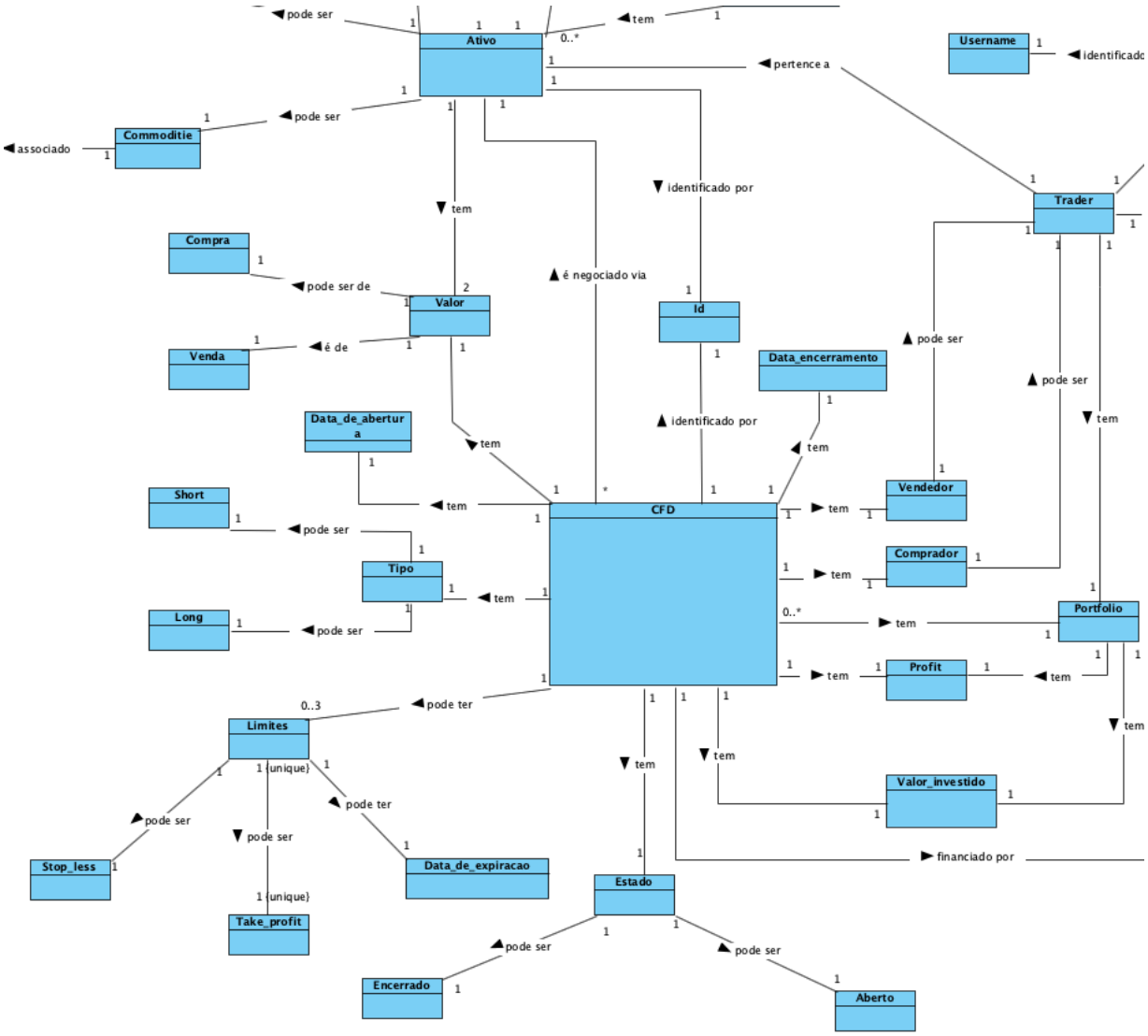

Figura 2.10: Subsistema do CFD.

2.4 Diagrama de use cases

Tal como se pode observar a figura abaixo, o diagrama de use cases é um reflexo das funcionalidades já listadas anteriormente, tendo este dois atores, o **trader** (cliente da aplicação) e o **administrador**.
Das funcionalidades presentes, as mais importantes em relação ao **cliente** são **Visualizar a lista de ativos**, que tal como o nome indica vai listar os ativos e a respetiva informação dos preços atuais de venda e compra e a organização a que estão associados, o **Abrir CFD**, responsável por começar um contrato.
No entanto, a funcionalidade mais importante de todo o sistema é o **Encerrar CFD**, sendo este responsável por calcular todos os valores finais do contrato.
Por fim, a funcionalidade mais importante do administrador é a **inicialização do trading system**, pois tal como já foi referenciado anteriormente, o sistema é capaz de executar encerramentos por ação indireta/automáticos através de limites. No entanto, isto só é garantido com um servidor que está *full time* a verificar os **CFDs** que ultrapassam os limites de **Stop Less** e **Take Profit**.

Figura 2.11: Diagrama de use cases.

2.5 Atributos de qualidade

O sistema ser **funcional** é o principal objetivo, no entanto são as **características não funcionais** ou **atributos de qualidade** que influenciam totalmente a conceção da solução arquitetural. Para a mesma funcionalidade podemos ter diferentes arquiteturas. Um sistema lento pode fornecer a mesma funcionaliade que outro mais rápido, neste caso o **desempenho** é o atributo de qualidade que influenciou as diferenças na solução.

A implementação interna da nossa aplicação, neste momento, realiza acessos a uma única base de dados, quando utilizada por vários utilizadores simultâneamente pode existir um pico de acessos à mesma.
Uma das soluções possíveis para resolver este declínio de desempenho é definir um limite temporal de *queries* para cada utilizador.
A seguinte figura ilustra um cenário em que um **Trader** consulta o seu porfolio para verificar os contratos (CFD's) em aberto.

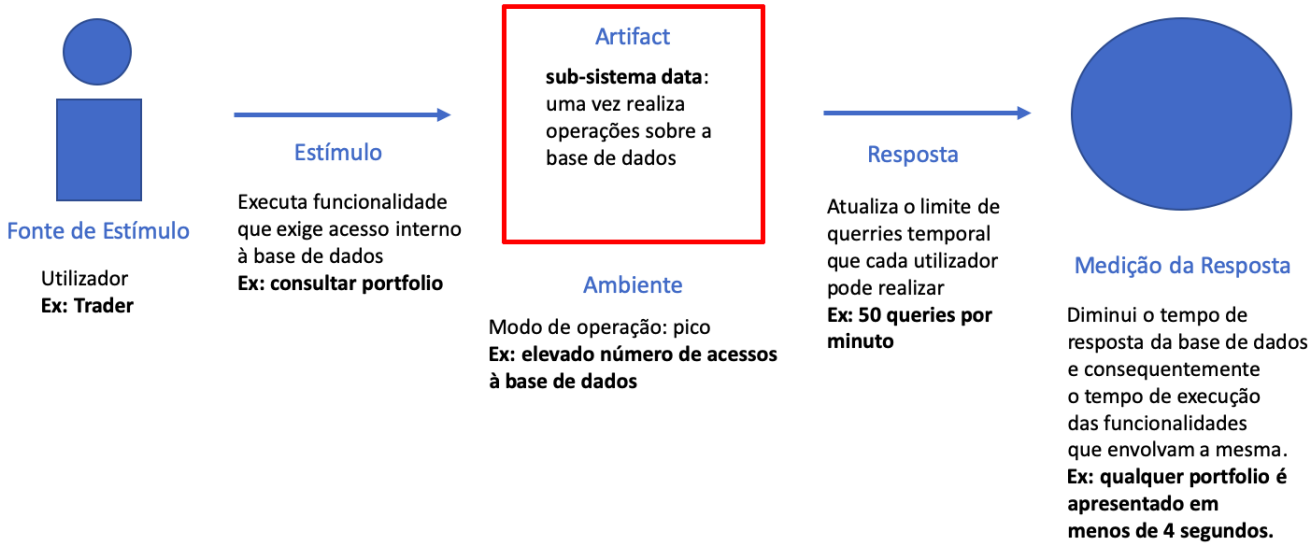

Figura 2.12: Cenário de performance.

2.6 Diagrama de classes

O desenvolvimento do diagrama de classes foi baseado na arquitetura *MVC*, sendo que dividiu-se o sistema em dois grandes *packages*: **business** e **data**.
No entanto, para reforçar a arquitetura, no *package* **business**, foram criados dois *sub-packages*, pois a equipa achou clara a distinção entre o subsistema dos utilizadores, denominado por **recursos humanos** e o subsistema da lógica **trading**. Importante referir que para cada *package* foi definida uma interface e uma classe *Facade* que implementa a mesma com todos os métodos.

Em relação ao *package* **data**, tal como se pode verificar na figura abaixo, caracteriaza-se pela comunicação com os dados, ou seja, a base de dados, bem como a **API** utilizada para obtenção dos valores reais dos ativos, nomeadamente a classe **AtivoDAO**. As restantes classes internas a este package são referentes aos **CFDs** e **Utilizadores**. Importante referir que os métodos presentes no **Facade** são nomeadamente inserção, remoção e listagem de dados.

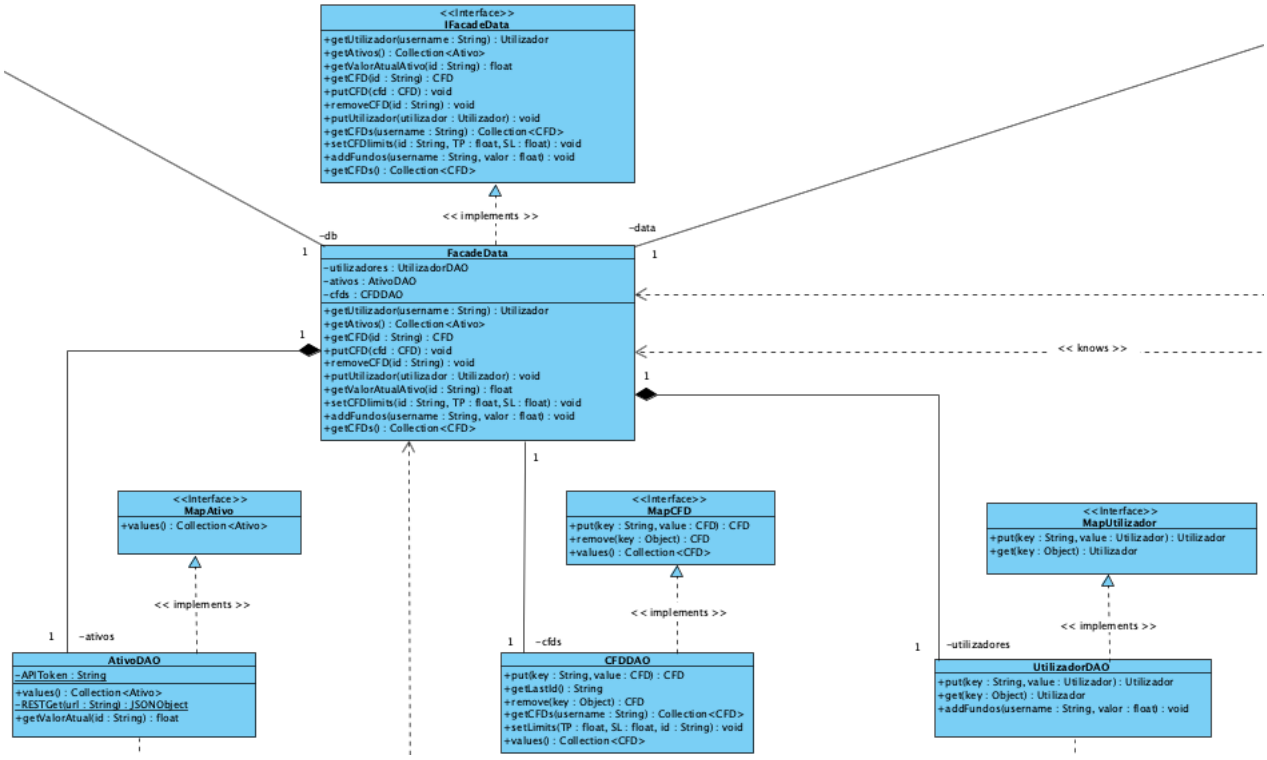

Figura 2.13: Diagrama de classes, package data.

A parte do diagrama de classes presente na figura abaixo, representa o *package* **recursos humanos** denominado no diagrama por **RH**, que está inserido no *package* **business**. Neste *package* foi criado um **FacadeRH**, pois tal como já foi referido decidiu-se implementar uma classe **Facade** para cada *package*. Nesta classe estão definidos os métodos de **registo**, **login** dos utilizadores, bem como a respetiva classe **Utilizador** e as suas sub-classes.

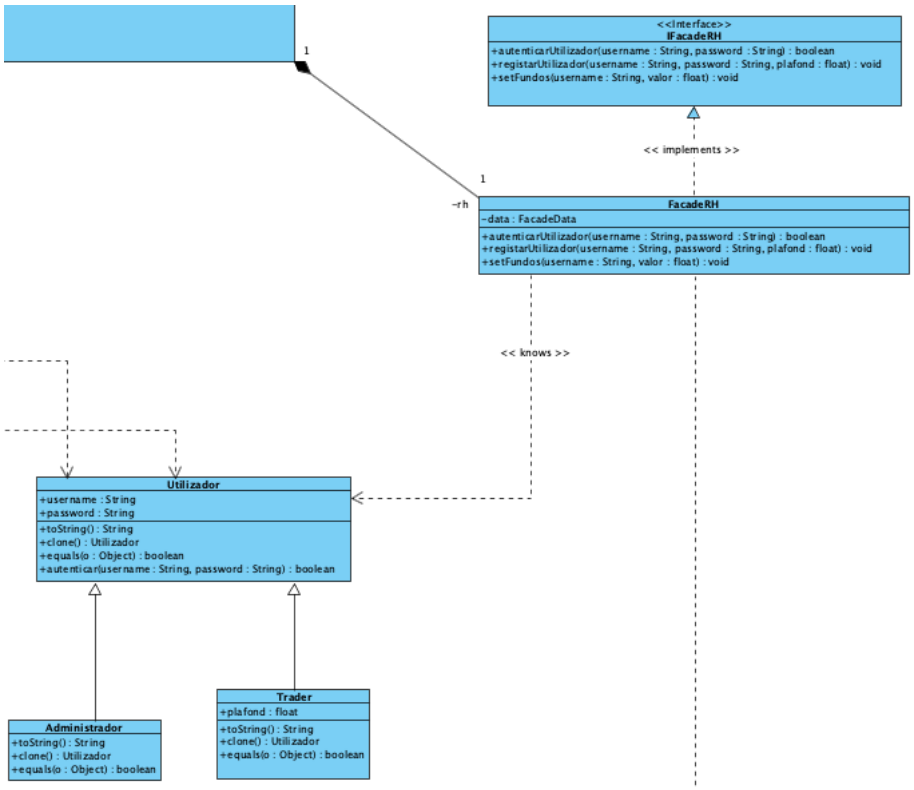

Figura 2.14: Diagrama de classes, package business.RH.

Do mesmo modo, que o *package* referido anteriormente, este incluí a interface **IFacadeTrading** e a classe **FacadeTrading**, onde estao definidos os métodos deste *package*. O objetivo deste *package* é concentrar as funcionalidades sobre os **CFDs** e os **Ativos**, tal como se pode verificar na figura abaixo.

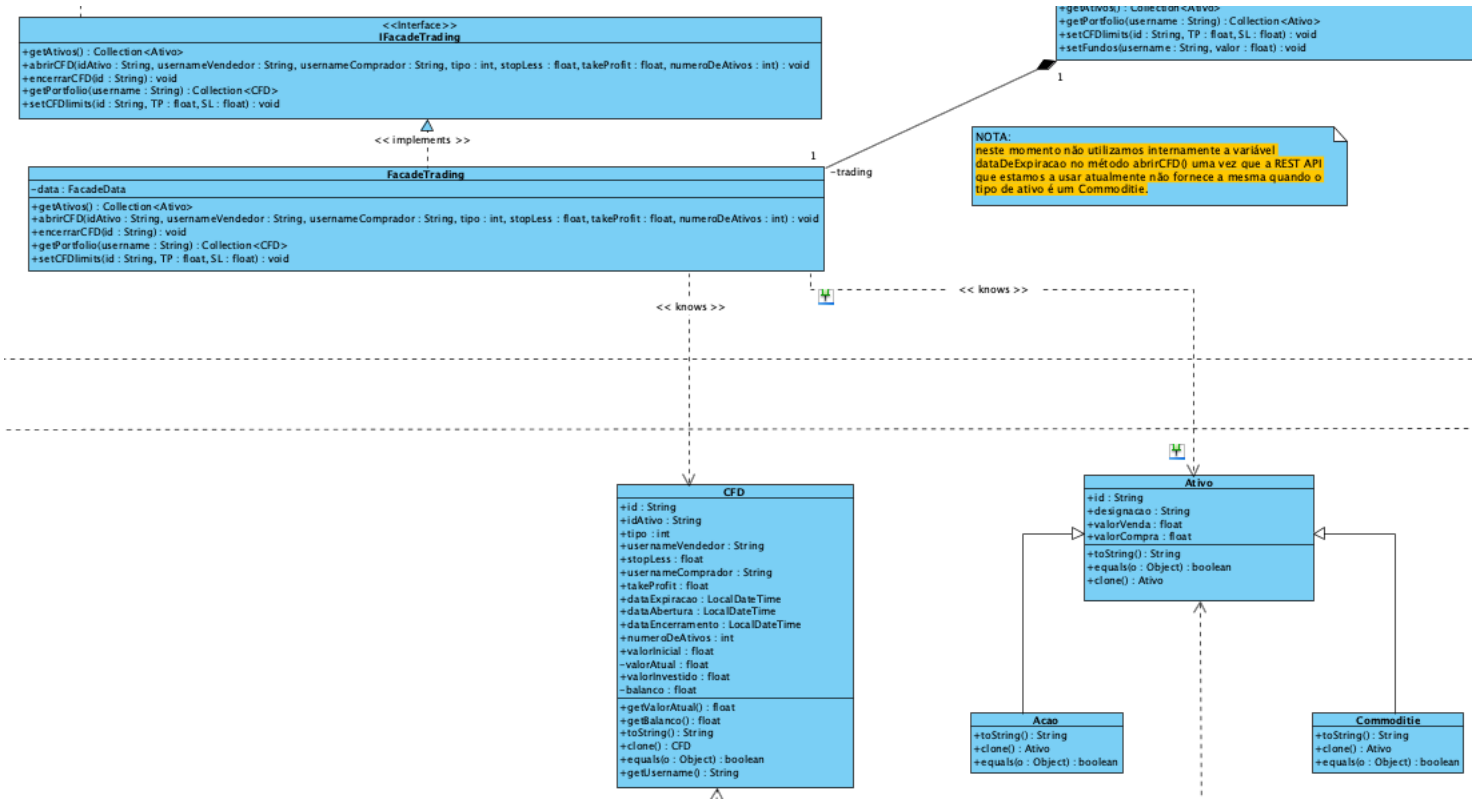

Figura 2.15: Diagrama de classes, package business.Trading.

Tal como foi dito anteriormente e observando a figura abaixo, os *packages* apresentados acima, estão inseridos no *package* **business**. Igualmente aos anteriores, este *package* também tem uma interface, denominada **IFacadeBusiness** e a classe **Facade** designada **FacadeBusiness**, onde estão implementados todos os métodos dos *packages* **Trading** e **RH** e os atributos dos **Facades** dos mesmos (setas vermelhas), bem como o **FacadeData** para aceder aos dados.

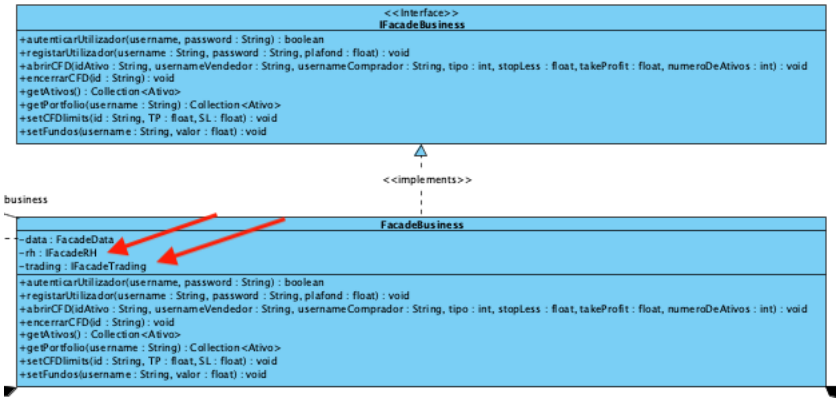

Figura 2.16: Diagrama de classes, package business.

Tal como foi explicado inicialmente o utilizador pode encerrar um CFD automáticamente, definindo limites de **Top Less** e **Take profit**, o que significa que, a qualquer momento isso pode acontecer, mesmo quando o cliente não estiver autenticado na aplicação. Então a equipa decidiu que era necessário um servidor, para

verificar os encerramentos automáticos dos **CFDs**, sendo este implementado na classe **TSServer**.

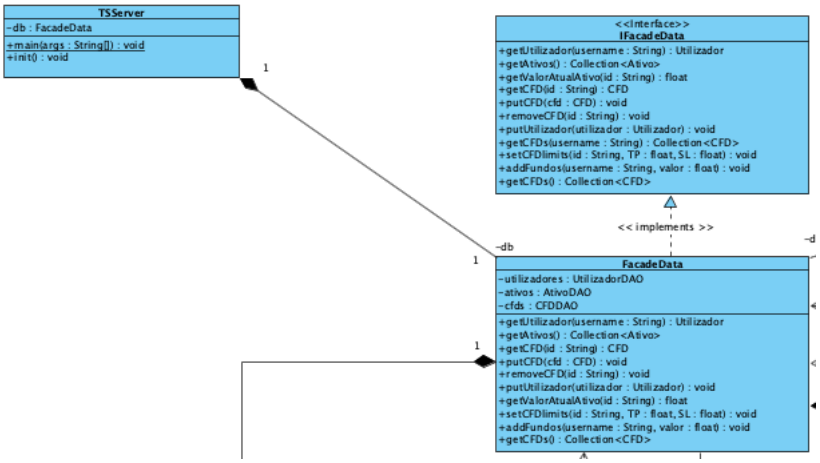

Figura 2.17: Diagrama de classes, classe TSServer.

Por fim, a classe cental da arquitetura denominada por **TSClient**, onde estará desenvolvida a view do programa, bem como todas as funcionalidades do mesmo. Os atributos desta classe, são os **Facades** dos *packages*.

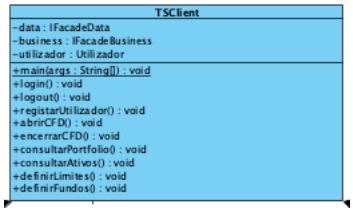

Figura 2.18: Diagrama de classes, TSClient.

2.7 Diagramas de comportamento

Nesta fase implementou-se o diagrama de sequência de implementação para a funcionalidade automática do sistema que consiste em verificar os contratos abertos (CFD's) de cada utilizador e caso os limites de Stop Less ou Take Profit sejam alcançados, é feito o encerramento do contrato (CFD).

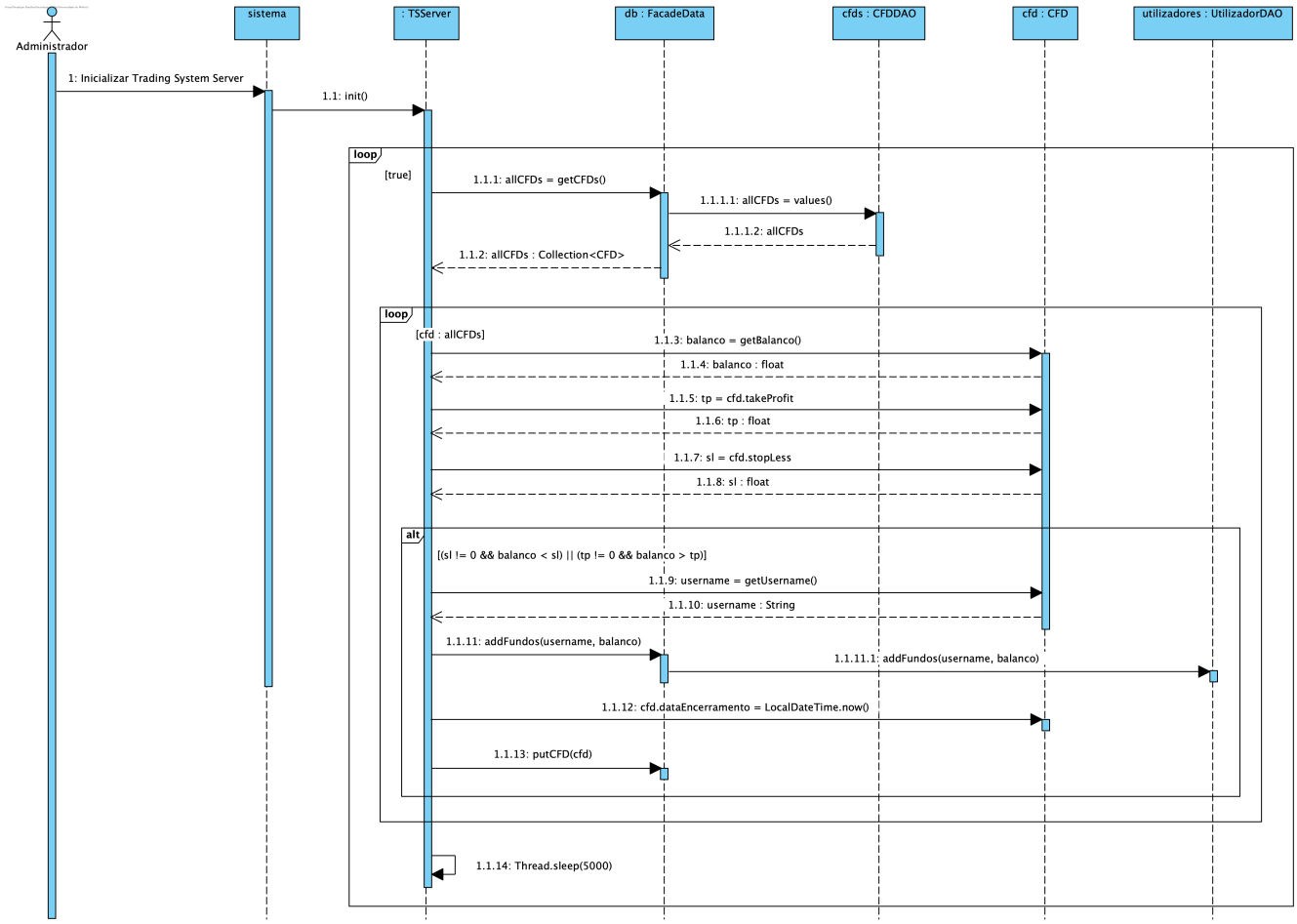

Figura 2.19: Diagrama de sequência de implementação.

No diagrama estão já especificadas as classes e variáveis envolventes tornando o futuro processo de implementação do método bastante facilitado uma vez que o raciocínio e fluxo de execução (comportamento) da funcionalidade está detalhada.

Note-se que a última execução do loop exterior é o método **Thread.sleep(5000)** uma vez que esta funcionalidade automática é realizada de 5 em 5 segundos, o intervalo pode ser ajustado conforme os resultados pertendidos.

Capítulo 3

Conclusão

Ao realizar esta fase do projeto a equipa apercebeu-se da importância de seguir o processo de desenvolvimento de software pela ordem indicada pelos professores docentes: identificação das funcionalidades pertendidas, modelação de domínio, diagrama de use case, cenários de atributos de qualidade, digrama de estrutura e diagramas de comportamento. Esta sequência de conceção incentiva a tomada de decisão prévia à implementação do código final o que aumenta a eficiência e correção da arquitetura a definir, desta forma quando é feita a implementação a probabilidade de ocorrência de erros de conceção ou alteração da arquitetura é reduzida.