2019/2020

Uma imagem com objeto, relógio, parede, antena

Descrição gerada automaticamente

Trading Platform

Arquiteruras de Software

Fábio senra A82108

César borges A81644

Índice

[Introdução 4](#_Toc25874112)

[Modelo de Domínio 5](#_Toc25874113)

[Principais entidades da plataforma: 5](#_Toc25874114)

[Descrição do modelo de domínio 5](#_Toc25874115)

[Funcionalidades do sistema 6](#_Toc25874116)

[Diagrama de use cases 6](#_Toc25874117)

[Use Case – Compra de CFD 7](#_Toc25874118)

[Use Case – Venda de CFD 8](#_Toc25874119)

[Definição de Stop Loss/Top Profit 8](#_Toc25874120)

[Mockups da plataforma 9](#_Toc25874121)

[Menu Inicial 9](#_Toc25874122)

[Menu de registo 9](#_Toc25874123)

[Menu de utilizador 9](#_Toc25874124)

[Menu “Comprar CFD” 9](#_Toc25874125)

[Menu “Ver portfolio” 9](#_Toc25874126)

[Menu “Ver CFDs em posse” 10](#_Toc25874127)

[Menu de definição de Top Profit/Stop Loss 10](#_Toc25874128)

[Menu de CFD possuído 10](#_Toc25874129)

[Atributos de qualidade e cenários 11](#_Toc25874130)

[Padrão arquitetural 12](#_Toc25874131)

[Diagrama de classes 14](#_Toc25874132)

[Padrões de design 15](#_Toc25874133)

[Observer 15](#_Toc25874134)

[DAO 15](#_Toc25874135)

[Factory Method 16](#_Toc25874136)

[Template Method 16](#_Toc25874137)

[Singleton 17](#_Toc25874138)

[Mediator 17](#_Toc25874139)

[Diagramas de sequência 18](#_Toc25874140)

[Venda de CFD 18](#_Toc25874141)

[Compra de CFD 18](#_Toc25874142)

[Definição do Top Profit 19](#_Toc25874143)

[Definição do Stop Loss 19](#_Toc25874144)

[Diagrama de estados 20](#_Toc25874145)

[Conclusão 21](#_Toc25874146)

[2ª fase 22](#_Toc25874147)

[Funcionalidades suportadas 22](#_Toc25874148)

[Alterações efetuadas 22](#_Toc25874149)

[Padrão arquitetural 22](#_Toc25874150)

[Padrões de design 24](#_Toc25874151)

[Observer 24](#_Toc25874152)

[DAO 25](#_Toc25874153)

[Factory Method 25](#_Toc25874154)

[Template Method 26](#_Toc25874155)

[Singleton 26](#_Toc25874156)

[Mediator 26](#_Toc25874157)

[Figura 1 - Modelo de domínio 4](file:////Users/fabiosenra/Documents/MIEI-UM/AS/EESTrading/Relatório.docx#_Toc22837116)

[Figura 2 - Diagrama de Use Cases 5](#_Toc22837117)

[Figura 3 - Use Case - Compra de CFD 6](#_Toc22837118)

[Figura 4 - Use Case - Venda de CFD 7](#_Toc22837119)

[Figura 5 - Use Case - Definição de Stop Loss/Top Profit 7](#_Toc22837120)

[Figura 6 - Menu Inicial 8](#_Toc22837121)

[Figura 7 - Menu de registo 8](#_Toc22837122)

[Figura 8 - Menu de utilizador 8](#_Toc22837123)

[Figura 9 - Menu "Comprar CFD" 8](#_Toc22837124)

[Figura 10 - Menu "Ver portfolio" 8](#_Toc22837125)

[Figura 11 - Menu "Ver CFDs em posse" 9](#_Toc22837126)

[Figura 12 - Menu de definição de Top Profit/Stop Loss 9](#_Toc22837127)

[Figura 13 - Menu de CFD possuído 9](#_Toc22837128)

[Figura 14 - Cenário Manutenibilidade 10](file:////Users/fabiosenra/Documents/MIEI-UM/AS/EESTrading/Relatório.docx#_Toc22837129)

[Figura 15 - Cenário Usabilidade 11](#_Toc22837130)

[Figura 16 - Diagrama de sequencia da venda de CFD 13](#_Toc22837131)

[Figura 17 - Diagrama de sequência da compra de CFD 13](#_Toc22837132)

[Figura 18 - Diagrama de sequência da Definição do Top Profit 14](#_Toc22837133)

[Figura 19 - Diagrama de sequência da Definição do Stop Loss 14](#_Toc22837134)

[Figura 20 - Diagrama de estados 15](file:////Users/fabiosenra/Documents/MIEI-UM/AS/EESTrading/Relatório.docx#_Toc22837135)

# Introdução

O projeto atual tem como principal a construção de uma arquitetura de software para uma plataforma de trocas. A plataforma deve permitir aos seus utilizadores várias formas de negociação de contratos de diferenças sobre vários tipos de ativos financeiros (ações, commodities, índices, moeda). Os valores destes contratos devem regular-se pelas subidas e descidas que ocorrem na bolsa de valores real, através da utilização de uma API online.

É pretendido ainda que se realizem alguns diagramas em UML de maneira a ilustrar algumas das decisões tomadas ao longo do trabalho.

Começar-se-á pelo modelo de domínio, a definição das funcionalidades principais, quais os atributos de qualidade adotados, a elaboração de alguns use cases, os mockups da aplicação, os atributos de qualidades e os seus cenários, o diagrama de classes, em seguida as funcionalidades especificadas em diagrama de sequência e acabando com o diagrama de estados que revela o fluxo que existe na aplicação.

# Modelo de Domínio

Neste capítulo apresentar-se-á as principais entidades do sistema em questão e ainda as relações que essas entidades estabelecem entre si. Tudo isto será baseado em algumas plataformas já existentes no mercado e a forma como estas trabalham.

## Principais entidades da plataforma:

* Contrato de diferenças (CFD)
* Investidor
* Trader
* Saldo
* Portfolio
* Ativos financeiros (Ações, Commodities, Índices, Moedas)

## Descrição do modelo de domínio

A entidade principal do sistema será o contrato de diferenças, uma vez que, este englobará todas as outras entidades existentes. Por sua vez, os investidores e os Traders serão quem cria os contratos abrindo posições de compra ou venda, cada um destes tem ainda o seu saldo que lhes permite comprar contratos.

À medida que se vão realizando contratos de diferenças é importante guardar um Portfolio para que todas essas transações que foram efetuadas no passado sejam guardadas para que os utilizadores possam saber tudo o que já realizaram na plataforma. De referir ainda que, todos estes contratos são realizados sobre um tipo de ativo financeiros podendo este ser uma ação, um commodity (Ouro, Petróleo, Prata), um índice ou uma moeda.

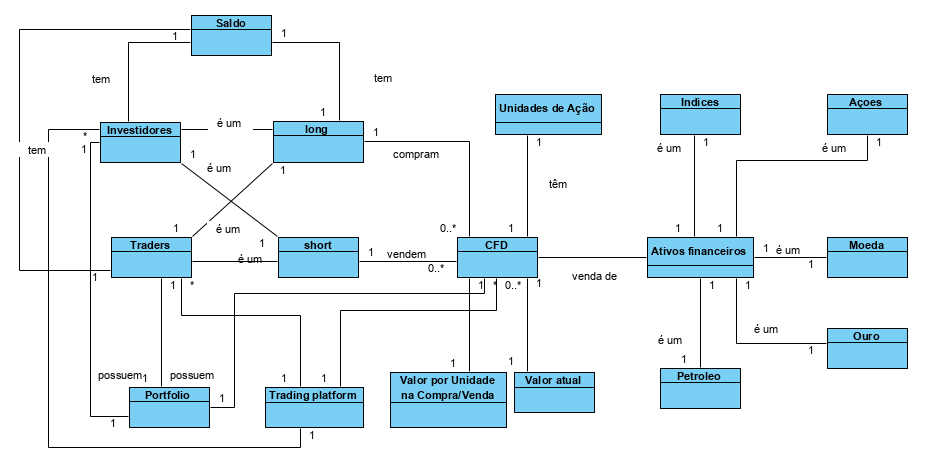


Figura - Modelo de domínio

# Funcionalidades do sistema

Depois de algum estudo sobre as outras plataformas já existentes encontraram-se as seguintes funcionalidades:

* **Registo:** permite que um utilizador crie uma conta para depois usufruir das funcionalidades da aplicação. Para este efeito deve apenas introduzir o nome de utilizador e a password que deseja.
* **Autenticação**: permite a entrada dos utilizadores nas suas contas. Para realizar a autenticação basta apenas fornecer o seu nome de utilizador e a respetiva password.
* **Compra de ativo financeiro**: o utilizador deve indicar a quantidade monetária que deseja desse mesmo ativo financeiro.
* **Ver portfolio**: os utilizadores podem ver todos os CFD que já adquiriram. É possível visualizar a quantidade monetária investida em cada ativo financeiro de cada contrato e o valor atual com os juros aplicados pela plataforma.
* **Definição de Top Profit/Stop Loss**: os utilizadores devem conseguir estabelecer limites de perda/ganho em cada contrato, esta ação pode ser realizado aquando da compra do CFD ou posteriormente. De referir que o valor a inserir para o TopProfit não pode ser inferior ao valor inicial e o valor a inserir para o StopLoss também não pode ser superior ao valor inicial do CFD.
* **Depositar/levantar dinheiro**: o saldo de cada utilizador pode ser alterado pelos mesmos.
* **Ver contratos já concluídos:** o utilizador pode ver todos os contratos que já concluiu e o lucro que obteve desses mesmo contratos.

# Diagrama de use cases

Todas as funcionalidades supracitadas deram posteriormente origem aos use cases do sistema.

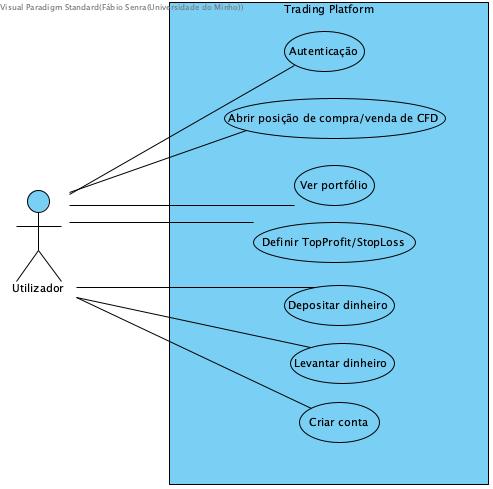


Figura - Diagrama de Use Cases

Na especificação dos use cases foi decidido que apenas seriam especificados os use cases mais importantes do sistema.

## Use Case – Compra de CFD

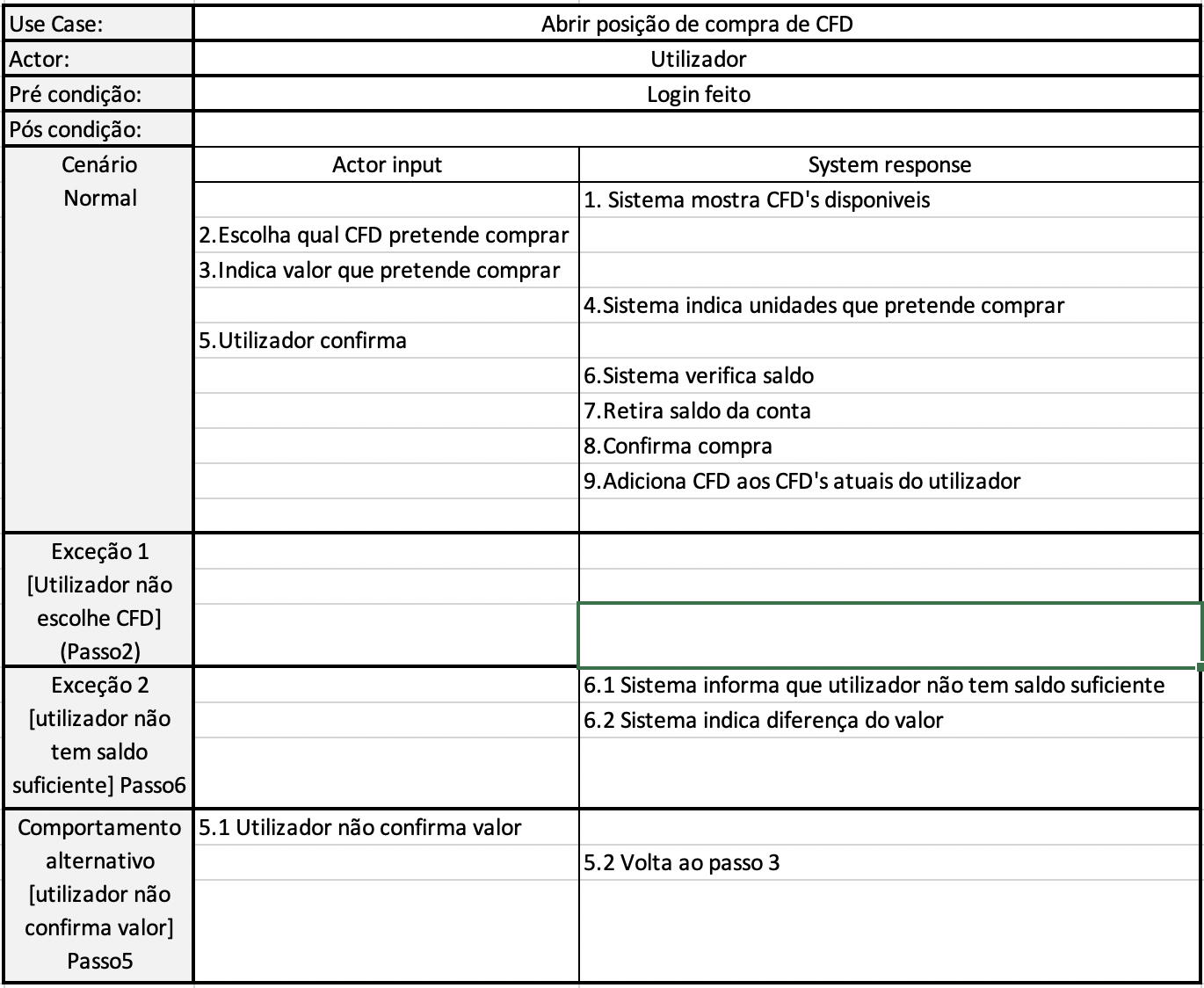


Figura - Use Case - Compra de CFD

## Use Case – Venda de CFD

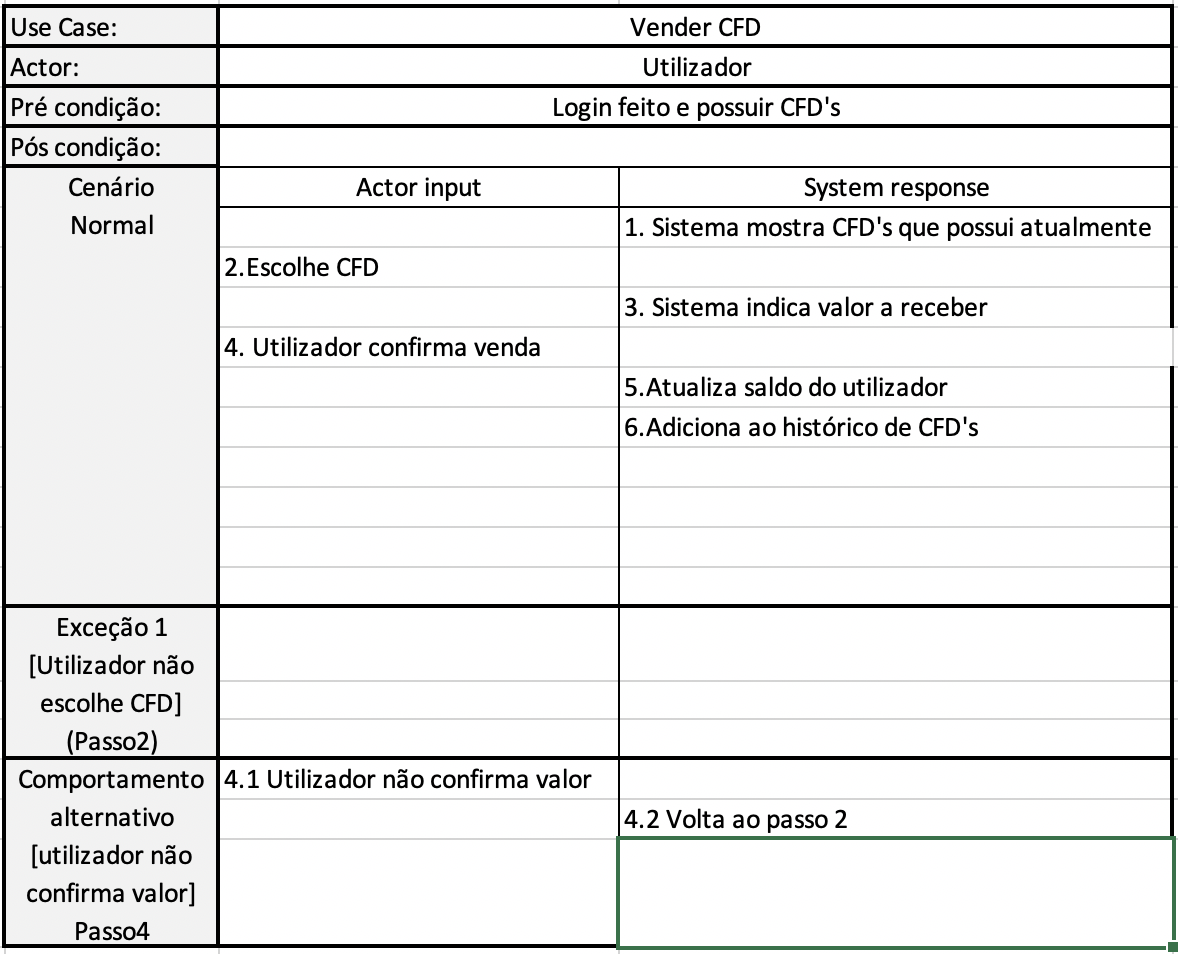


Figura - Use Case - Venda de CFD

## Definição de Stop Loss/Top Profit

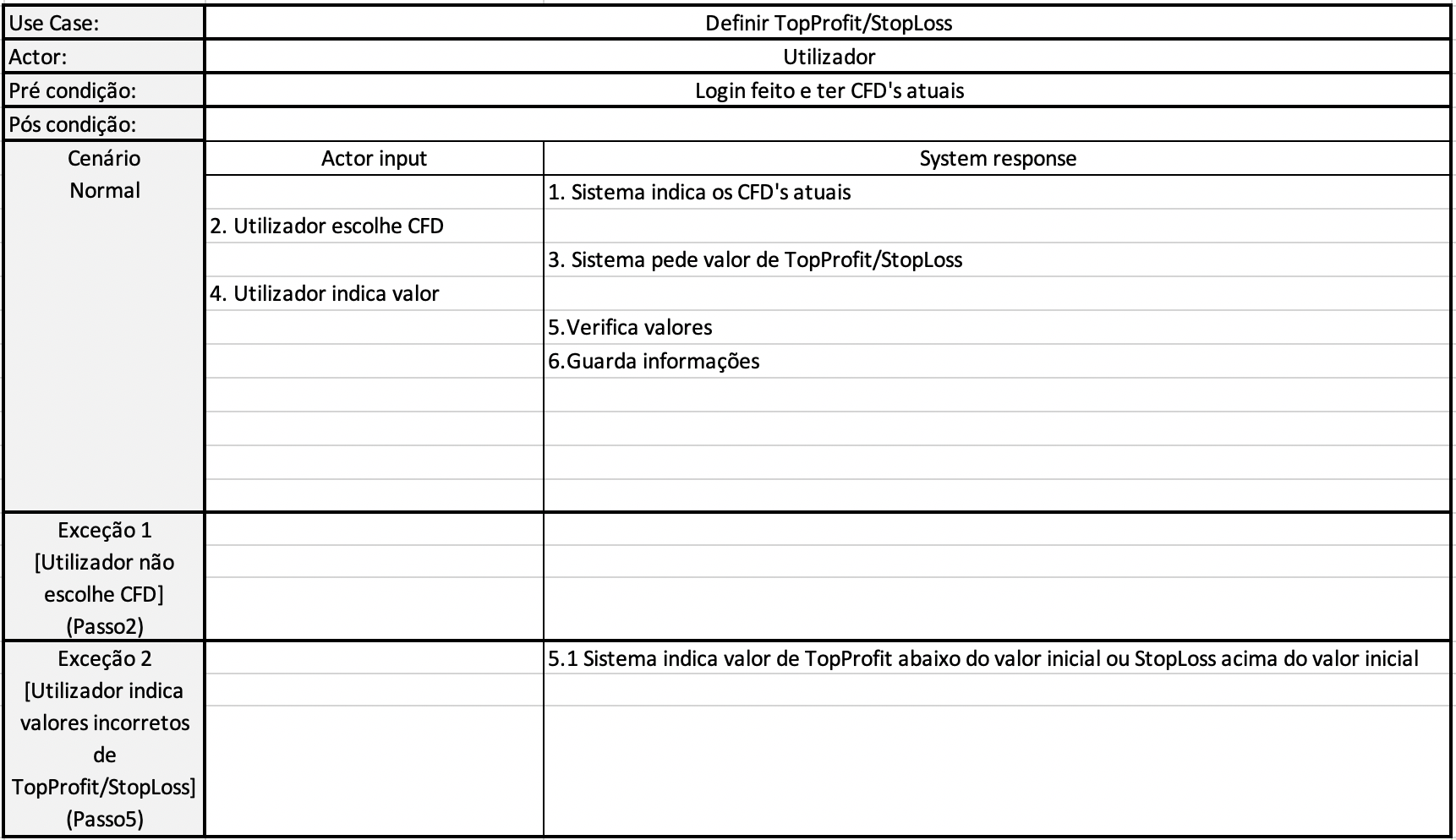


Figura 5 - Use Case - Definição de Stop Loss/Top Profit

# Mockups da plataforma

Nesta secção mostrar-se-ão as interfaces que os utilizadores vão ver no uso das várias funcionalidades do sistema.

A interface do utilizador resume-se em menus criados na linha de comandos, em cada menu será permitido ao utilizador escolhe uma ação pretendida e, caso seja necessário, será pedido inputs do deste.

## Menu Inicial

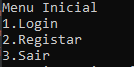


Figura - Menu Inicial

## Menu de registo



Figura - Menu de registo

## Menu de utilizador

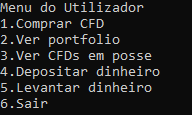


Figura - Menu de utilizador

## Menu “Comprar CFD”

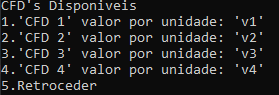


Figura - Menu "Comprar CFD"

## Menu “Ver portfolio”

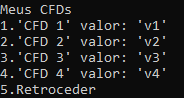


Figura - Menu "Ver portfolio"

## Menu “Ver CFDs em posse”



Figura - Menu "Ver CFDs em posse"

## Menu de definição de Top Profit/Stop Loss

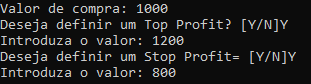


Figura - Menu de definição de Top Profit/Stop Loss

## Menu de CFD possuído



Figura - Menu de CFD possuído

# Atributos de qualidade e cenários

Como principais atributos de qualidade foram escolhidos a **modificabilidade** e **portabilidade**.

Uma das principais que o código deve ter é modificabilidade para permitir que qualquer mudança, acréscimo ou remoção de funcionalidades seja feita da forma mais simples possível. Tendo em conta que todos os produtos de software estão em constante mudança é imprescindível que o código tenha esta caraterística.

Outro atributo de qualidade considerado bastante importante é a portabilidade, esta permite que um módulo, com determinadas funcionalidades, criado para uma determinada arquitetura possa também ser útil noutra arquitetura, tornando esse módulo de código portável.

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura - Cenário Modificabilidade

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura - Cenário Portabilidade

# Padrão arquitetural

Para o produto a desenvolver e para os atributos de qualidade considerados foi escolhido o modelo arquitetural em camadas.

As 2 partes de um modelo em 2 camadas são: a camada de negócio (núcleo) e a camada que interage com ele constituída pela base de dados, pelos scrappers e pela interface de utilizador.

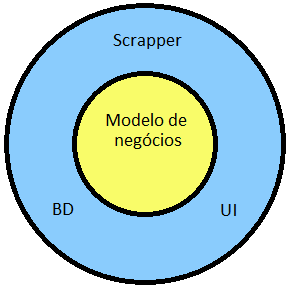


Figura - Arquitetura

O software executado na camada exterior pode ser substituído sem causar diferença para o sistema assim como atualizações e correções de defeitos podem ser feitas sem prejudicar a camada central.

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura - Diagrama de packages

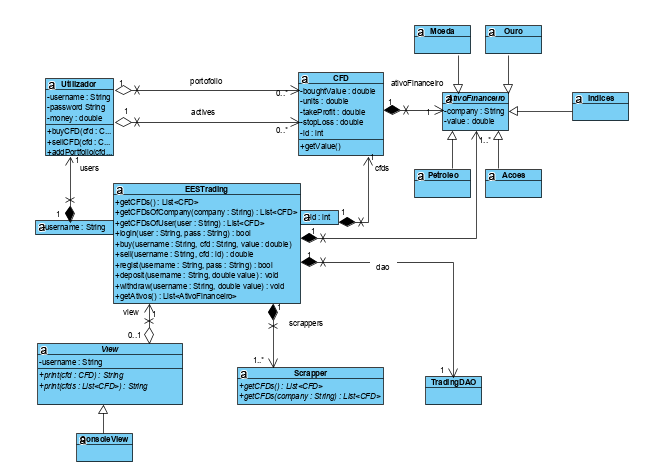
No total o sistema tem 4 módulos, sendo estes a View, Business, Scrapper e Data. O módulo View é responsável pela parte que o utilizador vê, isto é, a parte gráfica da aplicação, em seguida o Business que é responsável por toda a lógica da plataforma e onde estão todas as principais entidades do sistema, o módulo Scrapper que tem como função ir buscar (através de uma API online) as informações relativas aos ativos financeiros (Nome da empresa, valor do ativo financeiro,...) e por fim o módulo Data que é onde está toda a persistência da plataforma.

# Diagrama de classes

Depois de uma análise de todos os diagramas construídos, chegou-se ao seguinte diagrama de classes, ainda sendo uma previsão do que se pretende implementar podendo ainda sofrer alterações no futuro.

Será disponibilizado a cada utilizador uma vista, isto é um *View*, sendo este o único ponto onde o utilizador consegue comunicar com a aplicação, todas as funcionalidades do sistema são fornecidas pela classe *EESTrading*, podendo esta ser vista como uma *api*.

A *EESTrading* possui todos os utilizadores, contratos e ativos financeiros, cada ativo financeiro é referido nos contratos e os contratos são referidos pelos utilizadores, quer seja no seu portfolio ou como contratos ativos. O conjunto de contratos e/ou ativos serão futuramente colecionados através de um *Scrapper*, para que os dados sejam vistos em tempo real.



# Padrões de design

Os seguintes padrões de design foram utilizados para permitir inversões de dependências entre módulos e para diminuir a coesão nos módulos. O acoplamento entre o módulo do modelo de negócios para os restantes foi tornado o mínimo possível, enquanto que o acoplamento inverso não foi prioritário, mas mantido com algum cuidado.

## Observer

Este padrão foi incorporado para inverter a dependência, que inicialmente era do modelo de negócios para o modelo de apresentação.

Deste modo qualquer alteração que é feita aos dados do modelo de negócios é registada e notificada ao modelo de apresentação, sem que o modelo de negócios conheça o modelo de apresentação, por exemplo quando um ativo financeiro for atualizado, isto é, o seu valor for alterado, a visualização que o utilizador tem dos valores seja notificada e alterada posteriormente.

Uma imagem com captura de ecrã

Descrição gerada automaticamente

## DAO

Este padrão foi incorporado para que o modelo de negócio seja independente da forma como é feita a persistência de dados, a única dependência que o modelo de negócio tem ao modelo de dados são os objetos de acesso a dados relativos a cada entidade no modelo de negócios, por exemplo, o modelo de negócio depende de um CFDDAO que corresponde à entidade CFD. Cada objeto de DAO possui uma interface própria e a sua implementação possui uma conexão à base de dados.

O padrão DAO (Data Access Object) foi utilizado para que existisse persistência na aplicação permitindo que seja possível guardar e ir buscar dados.

Uma imagem com captura de ecrã

Descrição gerada automaticamente

## Factory Method

O *Factory method* foi utilizado para resolver os problemas de criação de objetos, principalmente para diminuir o acoplamento entre os módulos, visto que, esses módulos apenas dependem de uma *Factory* em vez de inúmeros objetos.

Por exemplo utilizou-se ainda o padrão *Factory Method* que disponibiliza métodos responsáveis pela criação de *scrappers* na instanciação do executável, tornando a main apenas dependente da *factory*. Foi implementado também na camada de Data para a criação das várias entidades que são persistidas e resolver o problema de criação destes objetos no modelo de negócios.

Uma imagem com captura de ecrã

Descrição gerada automaticamente

## Template Method

O template method foi utilizado para definir o algoritmo dos Scrappers, sendo este método o *start()*. Todas as sub classes de JSONAtivoFinanceiro terão que implementar o modo de como se extrai um conjunto de ativos financeiros de um objeto, mantendo o algoritmo base igual, permitindo o algoritmo manipular os ativos financeiros do modo que é suposto.

Este *design pattern* também foi utilizado no mediador das views com o mesmo intuito.

## Singleton

Implementou-se também o padrão *Singleton* para garantir uma única instância da principal classe da aplicação, ou seja, do ponto de acesso do modelo de negócios. Isto permite garantir que os dados que se vão buscar à API online e os dados que são mostrados na camada apresentação da aplicação estão em sintonia, ou seja, qualquer alteração que possa surgir altera a mesma instância da classe principal a ser usada.

## Mediator

Este *design pattern* foi incorporado na camada de apresentação para diminuir o acoplamento. Existem várias views, e sendo necessário cada *view* ter o conhecimento de cada *view* que a sucede, o mediador tem o papel de retirar esse mesmo conhecimento, controlando as interações que as *views* têm entre si, assim cada *view* apenas pode comunicar com o mediador em vez de o fazer com inúmeras outras *views*.

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Para abstrair ainda mais a comunicação entre módulos na aplicação foram ainda criadas várias Interfaces para que cada módulo conhecesse apenas a interface e os métodos que são utilizados na mesma.

# Diagramas de sequência

Depois de se ter recolhido as funcionalidades mais importantes, respetivas especificações dos use cases e de ter realizado o diagrama de classes mostra-se, neste capítulo, os diagramas de sequência que revelam o comportamento que cada uma das funcionalidades irá apresentar.

## Venda de CFD

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura - Diagrama de sequencia da venda de CFD

## Compra de CFD

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura - Diagrama de sequência da compra de CFD

## Definição do Top Profit

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura - Diagrama de sequência da Definição do Top Profit

## Definição do Stop Loss

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura - Diagrama de sequência da Definição do Stop Loss

# Diagrama de estados

Nesta secção pretende-se mostrar o fluxo que cada utilizador pode fazer desde o momento em que entra na plataforma até à saída do sistema, passando por todas as funcionalidades que já foram descritas em cima.

Este diagrama de estado reflete o comportamento que a interface do utilizador terá, cada estado será um menu, que dependendo da ação desejada do utilizador, passará a outro menu, as funcionalidades descritas em cada menu podem ser visualizadas nas *Mockups.*

Inicialmente, o utilizador pode-se registar ou entrar no sistema, após este entrar no sistema as funcionalidades pedidas podem ser encontradas através do menu principal, não havendo mais que 3 menus percorridos para que uma funcionalidade seja encontrada.

Uma imagem com texto, mapa

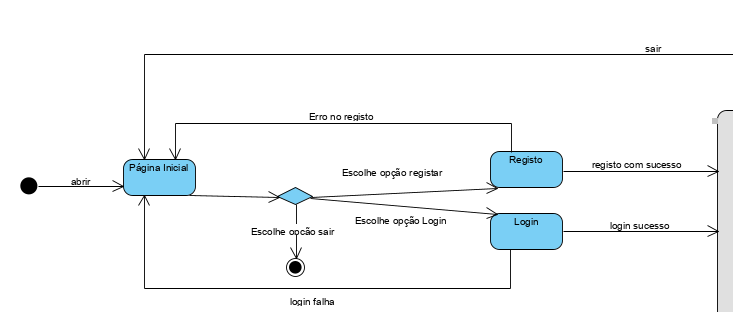
Descrição gerada automaticamente

Figura - Diagrama de estados

# Conclusão

Nesta primeira fase do trabalho foram exploradas diversas plataformas de trading para saber como estas funciona e perceber melhor o meio em que se inserem, com esta análise passou-se à criação de um modelo de domínio para conhecer melhor todo o contexto que existe em torno destas plataformas.

Posteriormente definiram-se as suas funcionalidades principais e os use cases envolvidos na utilização da plataforma, a especificação dos use cases focou-se essencialmente nas funcionalidades mais importantes da plataforma (vender CFD, comprar CFD e definir Stop Loss e Top Profit). Com as funcionalidades definidas, realizaram-se os mockups da aplicação para que o utilizador tenha uma primeira impressão de como irá interagir com a plataforma. Estabeleceram-se ainda os atributos de qualidade e os cenários respetivos.

Por fim construiu-se um diagrama de estados, para que fosse possível mostrar todo o fluxo da plataforma mostrando como o utilizador poderá navegar entre cada funcionalidade, e um diagrama de classes para mostrar a primeira arquitetura da aplicação.

# 2ª fase

## Funcionalidades suportadas

* Registo de um utilizador no sistema;
* Login de um utilizador já existente;
* Compra de um ativo financeiro;
* Na compra de um ativo financeiro o utilizador deve indicar a quantidade monetária que deseja desse mesmo ativo financeiro
* Definir um top profit e um stop loss para um contrato na sua compra e posteriormente;
* Vender um ativo financeiro conforme estipulado num contrato em qualquer momento que o utilizador deseja;
* Ver o portfolio do utilizador;
* No portfolio é possível visualizar a quantidade monetária investida em cada ativo financeiro de cada contrato e o valor atual com os juros aplicados;
* Ver os contratos já concluídos;
* Os contratos já concluídos mostram o lucro e o total vendido das transações feitas.
* Depositar uma quantia monetária;
* Levantar uma quantia monetária.

Na segunda fase do presente trabalho foi pedido que se implementasse a plataforma de trading usando padrões de design e arquiteturais, refinando assim a solução apresentada anteriormente.

## Alterações efetuadas

As alterações principais feitas foram a introdução de um padrão arquitetura, descrito no próximo capítulo, a introdução de novos módulos que têm como base o modelo de negócios e ligeiras alterações no modelo de negócios.

Foram separadas as responsabilidades do modelo de negócio anterior, ou seja, foram retiradas as entidades responsáveis pela apresentação, pela persistência dos dados e pela obtenção de dados.

Após esta divisão fez-se uma inversão nas dependências de CFDs e de utilizadores, ou seja, anteriormente o utilizador disponha de dois conjuntos de CFDs e atualmente o CFD dispõe do utilizador que o possui. Esta alteração foi feita devido ao facto de prevermos que a entidade que seria mais volátil ser o CFD, logo qualquer alteração feita ao CFD não se irá refletir diretamente à entidade do utilizador, o que na versão anterior poderia occurer.

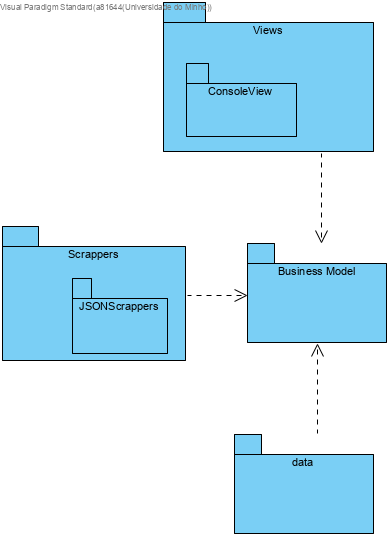
Foram ainda introduzidos design patterns para diminuir o acoplamento e a coesão, o uso destes será explicado nos capítulos seguintes.

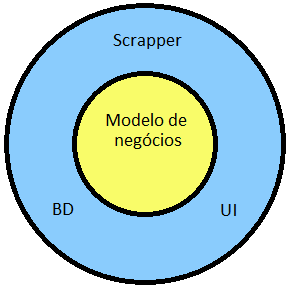
## Padrão arquitetural

Para o produto a desenvolver e para os atributos de qualidade considerados foi escolhido o modelo arquitetural em camadas.

As três partes de um modelo em três camadas são: camada de apresentação, camada de negócio e camada de dados. O software executado em cada camada pode ser substituído sem causar diferença para o sistema e, atualizações e correções de defeitos podem ser feitas sem prejudicar as restantes camadas. Por exemplo, alterações de interface podem ser realizadas sem o alterar a informação contida na base de dados.

No total o sistema tem 4 módulos, sendo estes a View, Business, Scrapper e Data. O módulo View é responsável pela parte que o utilizador vê, isto é, a parte gráfica da aplicação, em seguida o módulo de Business que é responsável por toda a lógica da plataforma e onde estão todas as principais entidades do sistema, o módulo Scrapper que tem como função ir buscar (através de uma API online) as informações relativas aos ativos financeiros (Nome da empresa, valor do ativo financeiro,...) e por fim o módulo Data que é onde está toda a persistência da plataforma.





O objetivo deste modelo arquitetural é de centralizar o modelo de negócio, ou seja, o módulo relativo ao modelo de negócio é independente dos restantes módulos e esses são dependentes do modelo de negócio.

Desta forma, é possível desenvolver o modelo de negócios sem haver conhecimento dos restantes módulos e desenvolvê-los após o modelo de negócios.

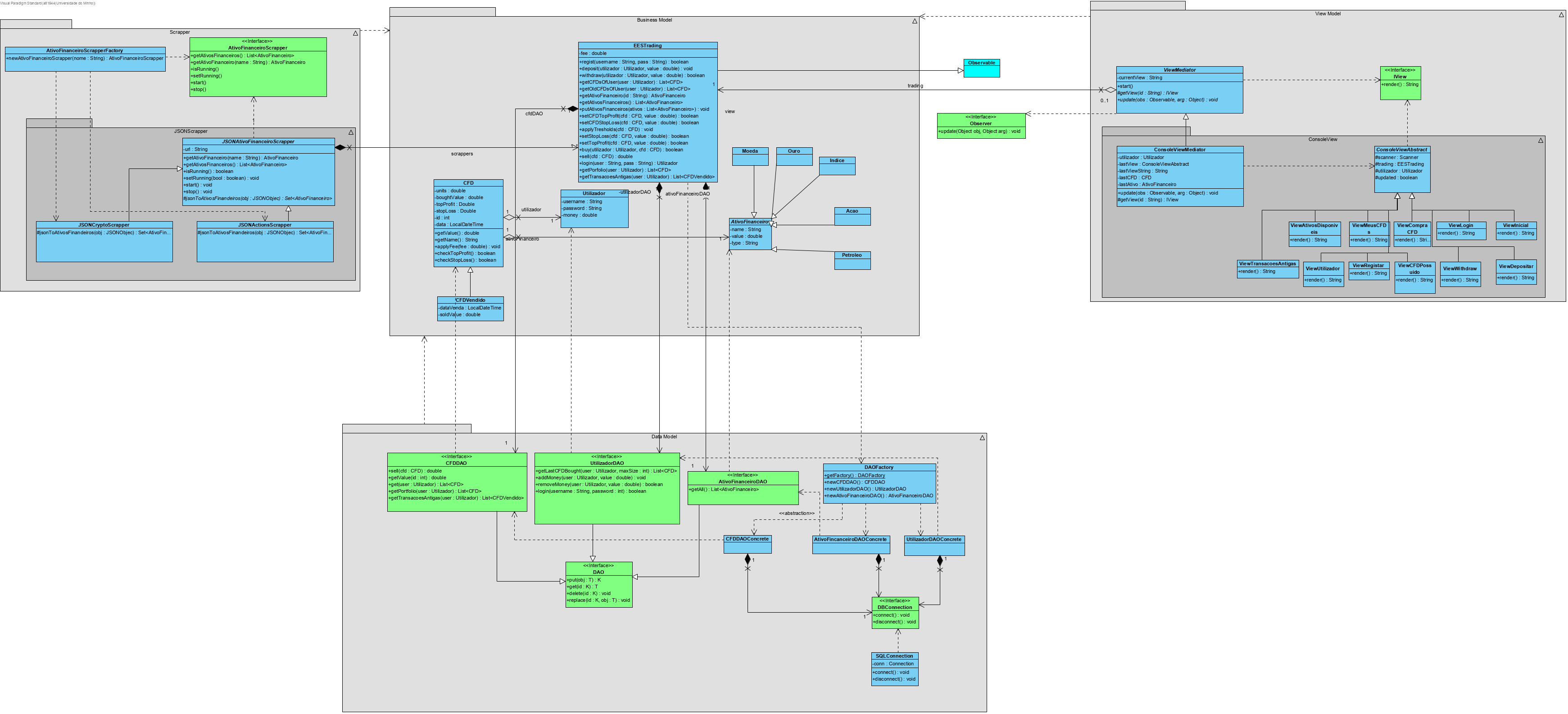
## Padrões de design

Os seguintes padrões de design foram utilizados para permitir inversões de dependências entre módulos e para diminuir a coesão nos módulos. O acoplamento entre o módulo do modelo de negócios para os restantes foi tornado o mínimo possível, enquanto que o acoplamento inverso não foi prioritário, mas mantido com algum cuidado.

## Observer

Este padrão foi incorporado para inverter a dependência, que inicialmente era do modelo de negócios para o modelo de apresentação.

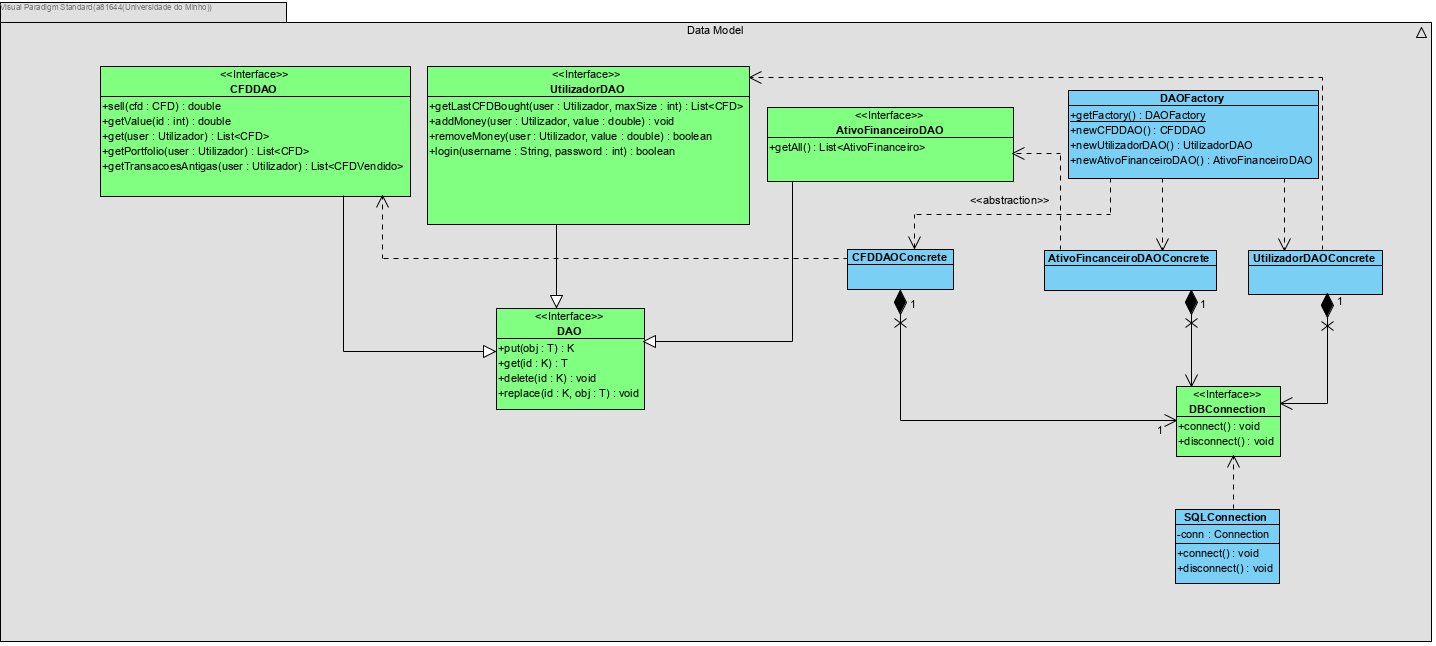
Deste modo qualquer alteração que é feita aos dados do modelo de negócios é registada e notificada ao modelo de apresentação, sem que o modelo de negócios conheça o modelo de apresentação, por exemplo quando um ativo financeiro for atualizado, isto é, o seu valor for alterado, a visualização que o utilizador tem dos valores seja notificada e alterada posteriormente.



## DAO

Este padrão foi incorporado para que o modelo de negócio seja independente da forma como é feita a persistência de dados, a única dependência que o modelo de negócio tem ao modelo de dados são os objetos de acesso a dados relativos a cada entidade no modelo de negócios, por exemplo, o modelo de negócio depende de um CFDDAO que corresponde à entidade CFD. Cada objeto de DAO possui uma interface própria e a sua implementação possui uma conexão à base de dados.

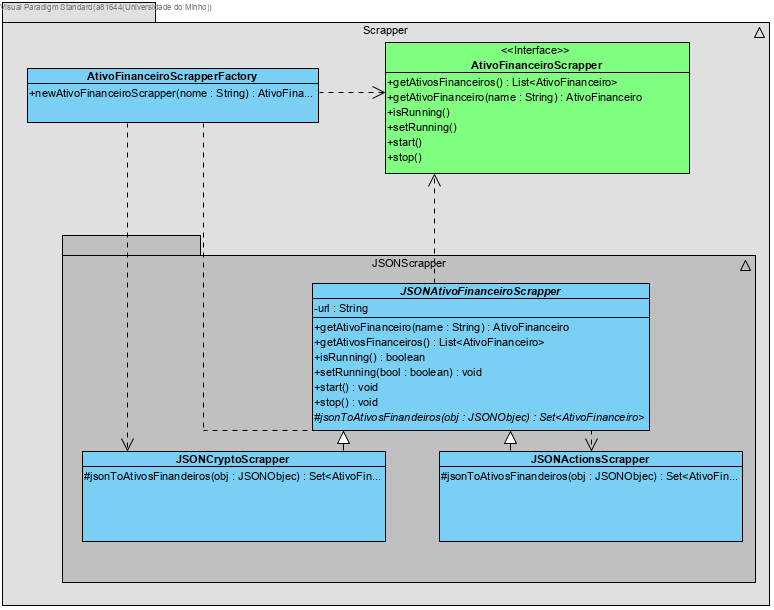
O padrão DAO (Data Access Object) foi utilizado para que existisse persistência na aplicação permitindo que seja possível guardar e ir buscar dados.



## Factory Method

O *Factory method* foi utilizado para resolver os problemas de criação de objetos, principalmente para diminuir o acoplamento entre os módulos, visto que, esses módulos apenas dependem de uma *Factory* em vez de inúmeros objetos.

Por exemplo utilizou-se ainda o padrão *Factory Method* que disponibiliza métodos responsáveis pela criação de *scrappers* na instanciação do executável, tornando a main apenas dependente da *factory*. Foi implementado também na camada de Data para a criação das várias entidades que são persistidas e resolver o problema de criação destes objetos no modelo de negócios.



## Template Method

O template method foi utilizado para definir o algoritmo dos Scrappers, sendo este método o *start()*. Todas as sub classes de JSONAtivoFinanceiro terão que implementar o modo de como se extrai um conjunto de ativos financeiros de um objeto, mantendo o algoritmo base igual, permitindo o algoritmo manipular os ativos financeiros do modo que é suposto.

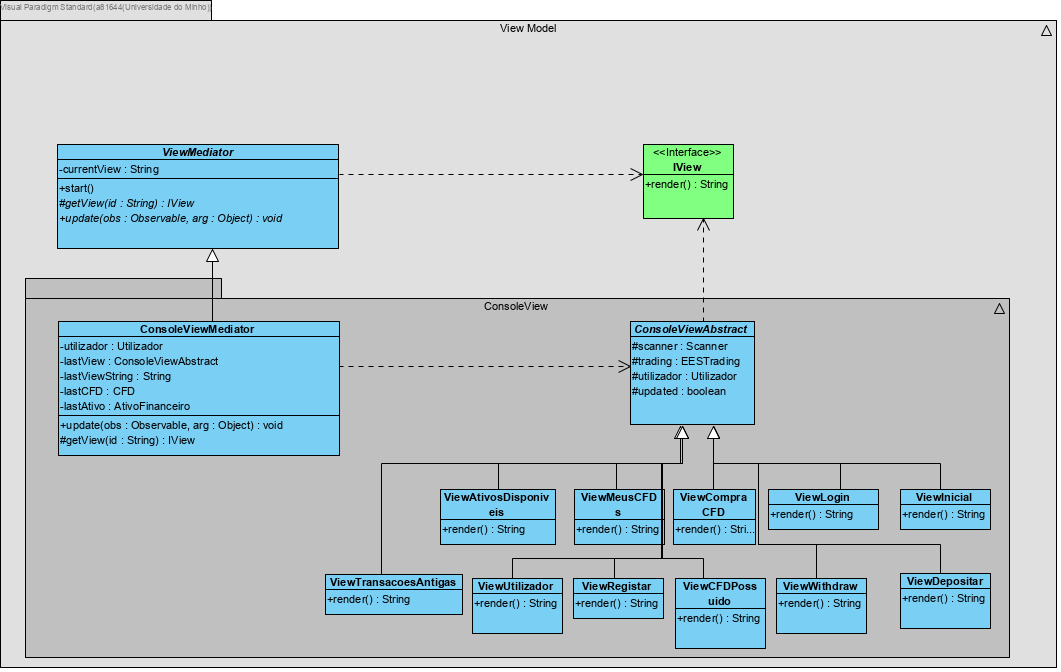
Este *design pattern* também foi utilizado no mediador das views com o mesmo intuito.

## Singleton

Implementou-se também o padrão *Singleton* para garantir uma única instância da principal classe da aplicação, ou seja, do ponto de acesso do modelo de negócios. Isto permite garantir que os dados que se vão buscar à API online e os dados que são mostrados na camada apresentação da aplicação estão em sintonia, ou seja, qualquer alteração que possa surgir altera a mesma instância da classe principal a ser usada.

## Mediator

Este *design pattern* foi incorporado na camada de apresentação para diminuir o acoplamento. Existem várias views, e sendo necessário cada *view* ter o conhecimento de cada *view* que a sucede, o mediador tem o papel de retirar esse mesmo conhecimento, controlando as interações que as *views* têm entre si, assim cada *view* apenas pode comunicar com o mediador em vez de o fazer com inúmeras outras *views*.



Para abstrair ainda mais a comunicação entre módulos na aplicação foram ainda criadas várias Interfaces para que cada módulo conhecesse apenas a interface e os métodos que são utilizados na mesma.