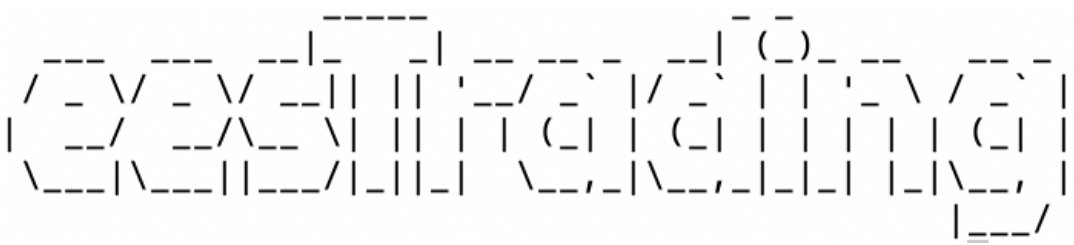
2019/2020



Trading Platform

ARQUITERURAS DE SOFTWARE

FÁBIO SENRA A82108

CÉSAR BORGES A81644

Índice

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.

Glossário

CFD: Contract For Differences.
Ativo financeiro: conjunto dos bens, valores ou direitos passíveis de serem convertidos em dinheiro e que são propriedade de uma pessoa singular ou coletiva.
Portfolio: conjunto de CFD's ainda em atividade.
Top Profit: valor máximo do CFD que se pretende que atinja.
Stop Loss: valor mínimo do CFD que o utilizador permite atingir.

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.

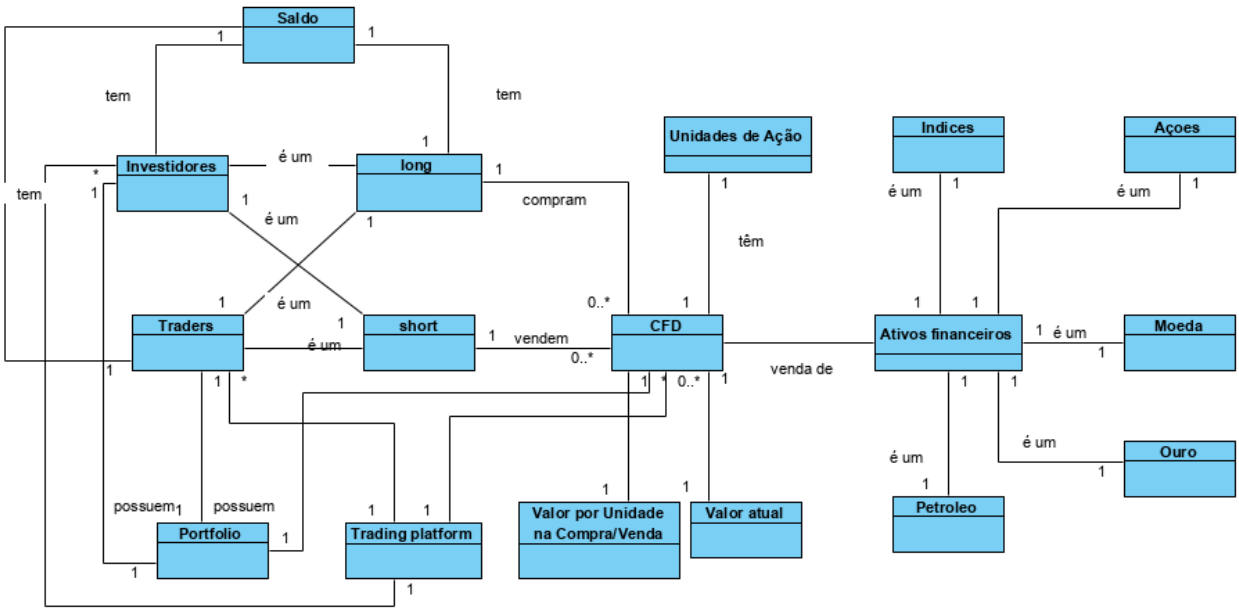


Figura 1 - Modelo de domínio

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 pelo mesmo.
- **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.

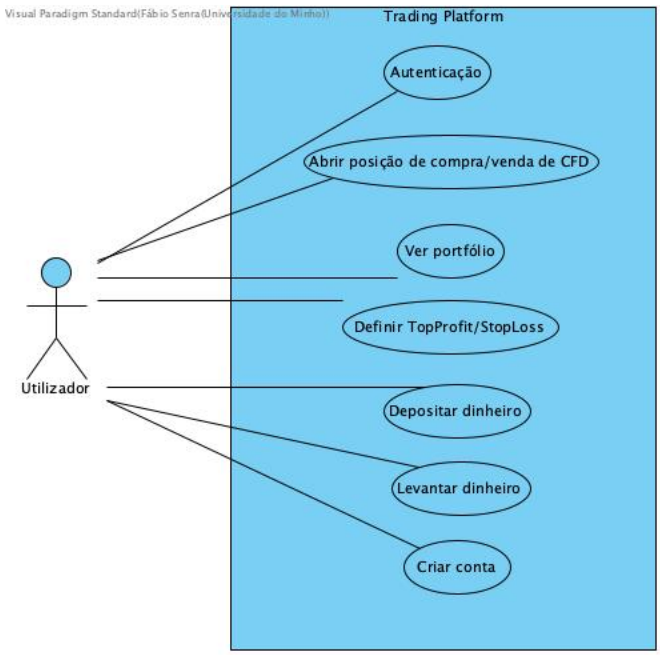


Figura 2 - Diagrama de Use Cases

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

Use Case – Compra de CFD

Use Case:	Abrir posição de compra de CFD	
Actor:	Utilizador	
Pré condição:	Login feito	
Pós condição:		
Cenário Normal	Actor input	System response
		1. Sistema mostra CFD's disponiveis
	2.Escolha qual CFD pretende comprar	
	3.Indica valor que pretende comprar	
		4.Sistema indica unidades que pretende comprar
	5.Utilizador confirma	
		6.Sistema verifica saldo
		7.Retira saldo da conta
		8.Confirma compra
		9.Adiciona CFD aos CFD's atuais do utilizador
Exceção 1 [Utilizador não escolhe CFD] (Passo2)		
Exceção 2 [utilizador não tem saldo suficiente] Passo6		6.1 Sistema informa que utilizador não tem saldo suficiente
		6.2 Sistema indica diferença do valor
Comportamento alternativo [utilizador não confirma valor] Passo5	5.1 Utilizador não confirma valor	
		5.2 Volta ao passo 3

Figura 3 - Use Case - Compra de CFD

Use Case – Venda de CFD

Use Case:	Vender CFD	
Actor:	Utilizador	
Pré condição:	Login feito e possuir CFD's	
Pós condição:		
Cenário Normal	Actor input	System response
		1. Sistema mostra CFD's que possui atualmente
	2.Escolhe CFD	
		3. Sistema indica valor a receber
	4. Utilizador confirma venda	
		5.Atualiza saldo do utilizador
		6.Adiciona ao histórico de CFD's
Exceção 1 [Utilizador não escolhe CFD] (Passo2)		
Comportamento alternativo [utilizador não confirma valor] Passo4	4.1 Utilizador não confirma valor	
		4.2 Volta ao passo 2

Figura 4 - Use Case - Venda de CFD

Definição de Stop Loss/Top Profit

Use Case:	Definir TopProfit/StopLoss	
Actor:	Utilizador	
Pré condição:	Login feito e ter CFD's atuais	
Pós condição:		
Cenário Normal	Actor input	System response
		1. Sistema indica os CFD's atuais
	2. Utilizador escolhe CFD	
		3. Sistema pede valor de TopProfit/StopLoss
	4. Utilizador indica valor	
		5.Verifica valores
		6.Guarda informações
Exceção 1 [Utilizador não escolhe CFD] (Passo2)		
Exceção 2 [Utilizador indica valores incorretos de TopProfit/StopLoss] (Passo5)		5.1 Sistema indica valor de TopProfit abaixo do valor inicial ou StopLoss acima do valor inicial

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

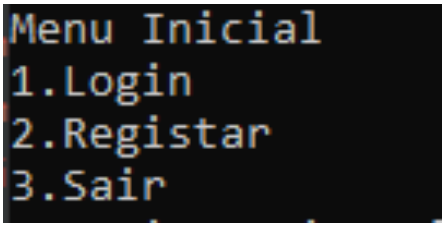


Figura 6 - Menu Inicial

Menu de registo

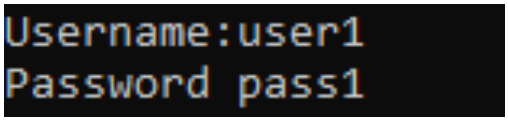


Figura 7 - Menu de registo

Menu de utilizador

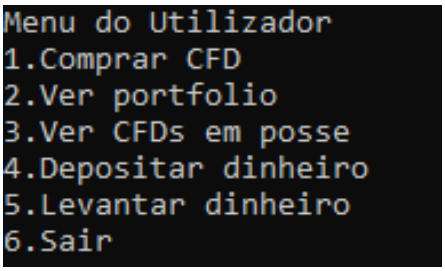


Figura 8 - Menu de utilizador

Menu “Comprar CFD”

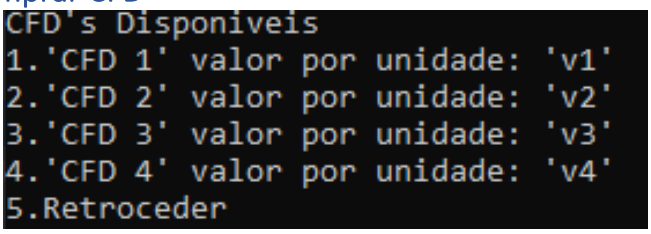


Figura 9 - Menu "Comprar CFD"

Menu “Ver portfolio”

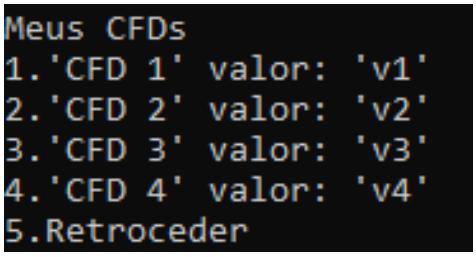


Figura 10 - Menu "Ver portfolio"

Menu “Ver CFDs em posse”

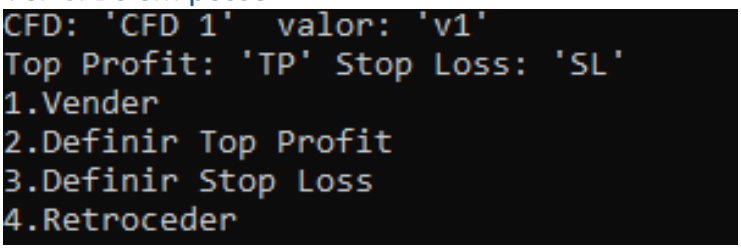


Figura 11 - Menu "Ver CFDs em posse"

Menu de definição de Top Profit/Stop Loss

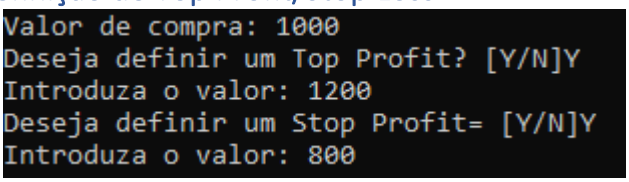


Figura 12 - Menu de definição de Top Profit/Stop Loss

Menu de CFD possuído

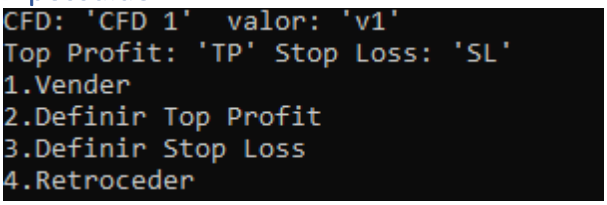


Figura 13 - Menu de CFD possuído

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.

O facto de acrescentar os atributos de modificabilidade e de portabilidade adiciona qualidade ao código, ou seja, do ponto de vista de quem desenvolve construir módulos que são facilmente modificáveis ajuda muito qualquer alteração que possa surgir durante a evolução dos projetos, no que toca à portabilidade isto faz com que os módulos construídos seguindo esse atributo possam ser reutilizados noutros projetos o que permite uma poupança de tempo que pode ser útil noutras partes dos projetos.

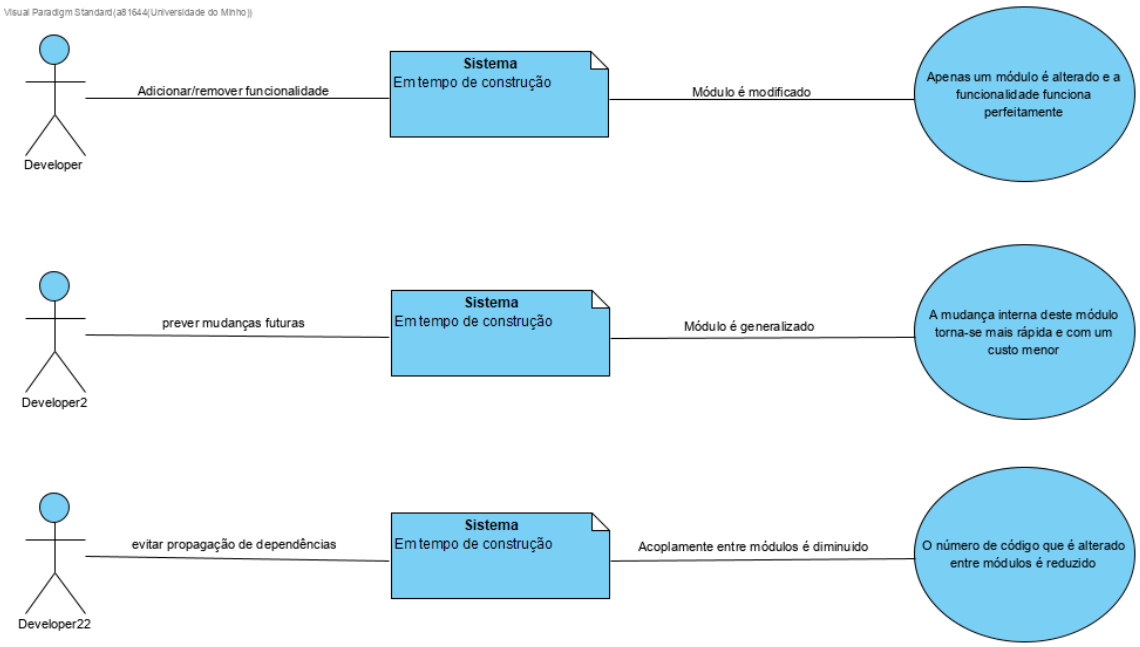


Figura 14 - Cenário Modificabilidade

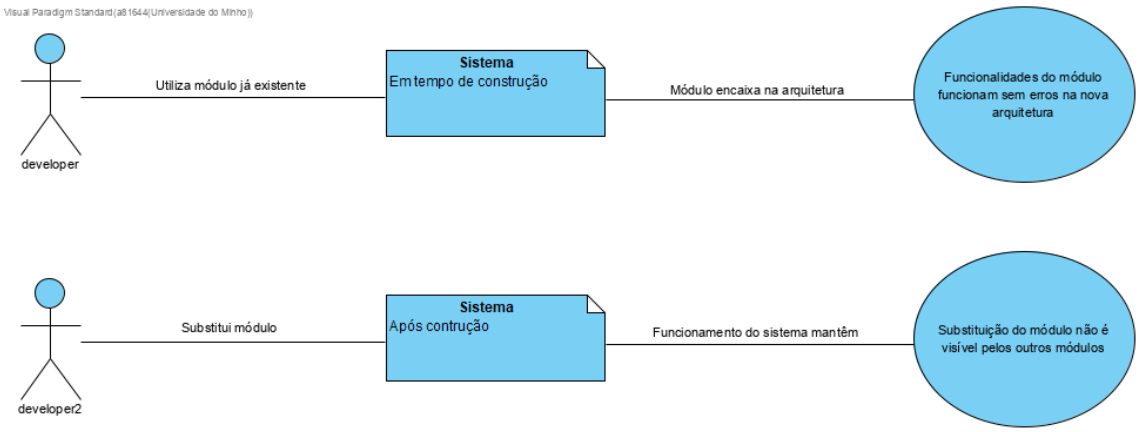


Figura 15 - Cenário Portabilidade

Estes atributos de qualidade foram escolhidos para permitir ao nosso sistema ser escalável, em termos de modificabilidade, ou seja, o tempo e custo de manutenção sejam sustentáveis e para que a adição ou alteração de módulos do sistema sejam incorporados facilmente.

Alterações efetuadas

As principais alterações feitas da primeira fase para esta 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 ocorrer.

Padrão arquitetural

Para o produto a desenvolver e para os atributos de qualidade considerados foi escolhido o modelo arquitetural em camadas, mais especificamente, uma arquitetura semelhante à *Clean arquitecture* de **Robert Cecil Martin**.

As duas partes de um modelo em duas 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.

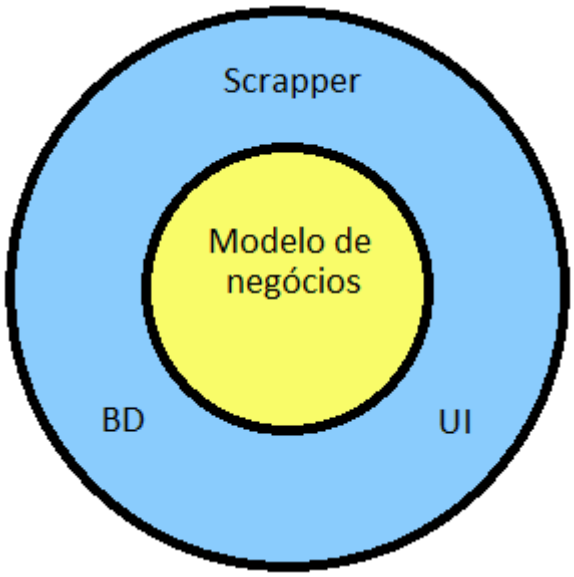


Figura 16 - Arquitetura

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.

Com esta arquitetura é possível desenvolver a camada de negócios sem ter conhecimento das camadas que a usem, sendo este o nosso principal argumento por ter escolhido esta arquitetura, que nos permitirá introduzir novas camadas ou módulos com responsabilidades distintas e com um número de mudanças reduzidas para a sua introdução no sistema.

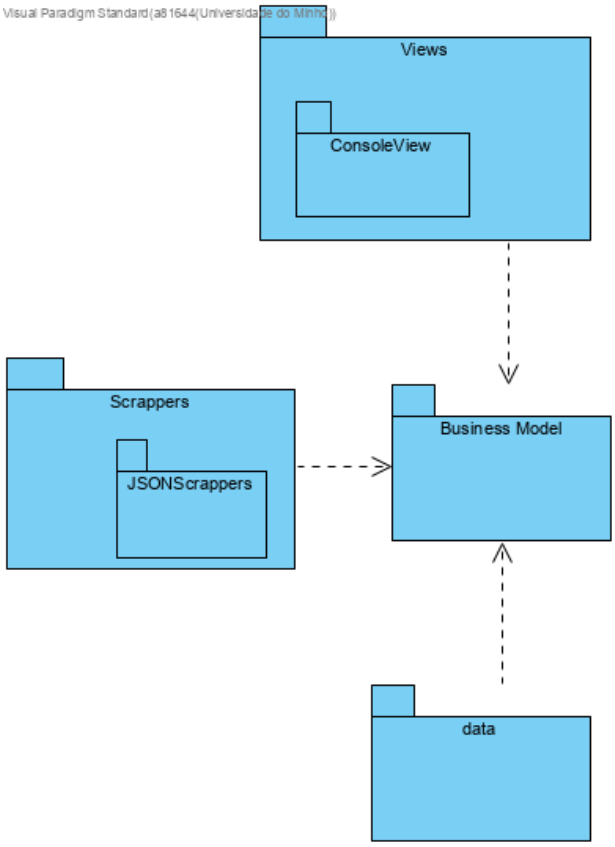


Figura 17 - Diagrama de packages

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 uma transação antiga já efetuada. O conjunto de contratos e/ou ativos são colecionados através de um *Scrapper*, para que os dados sejam vistos e atualizados em tempo real.

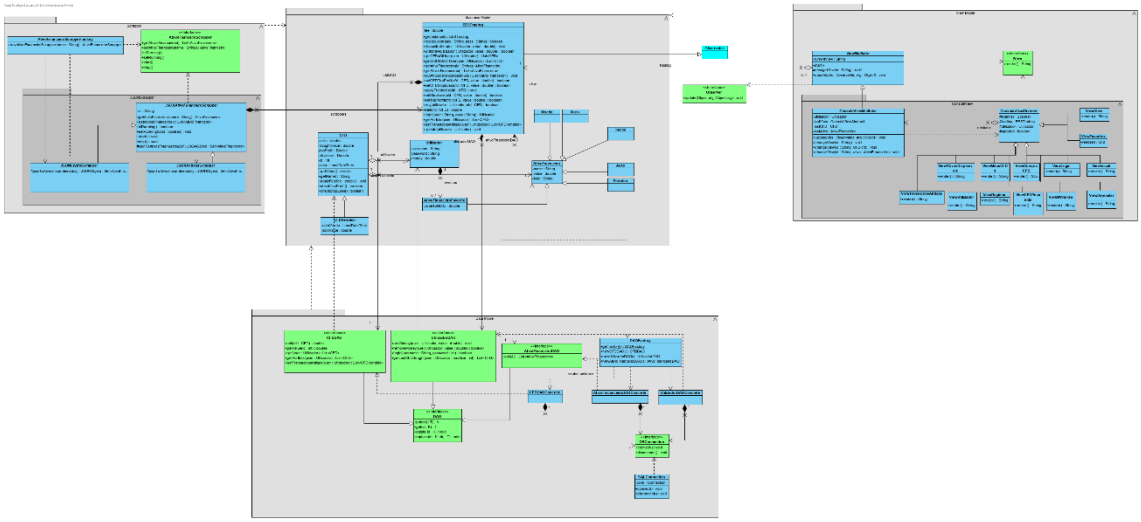


Figura 18 - Diagrama de classes

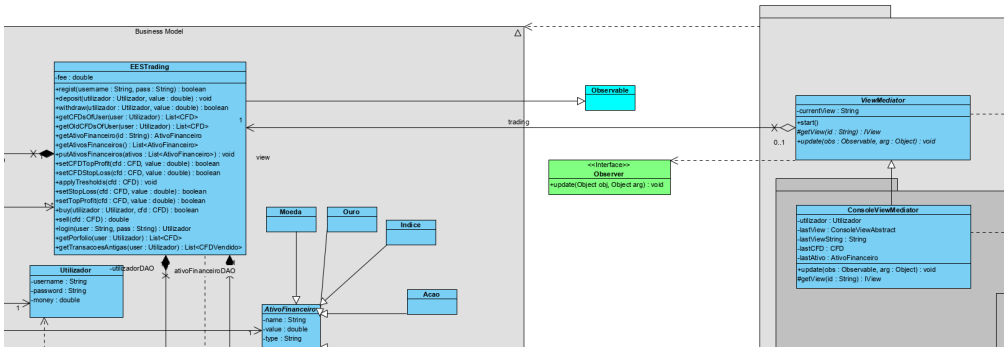
Padrões de design

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.

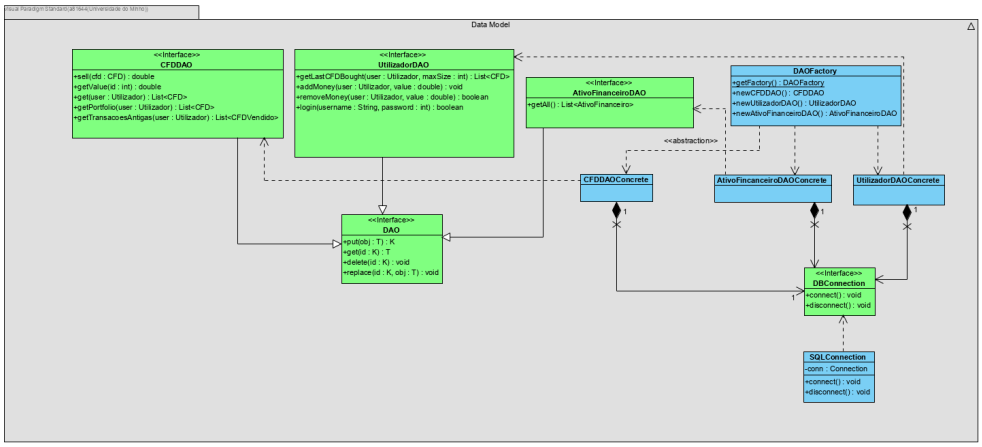


DAO

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.

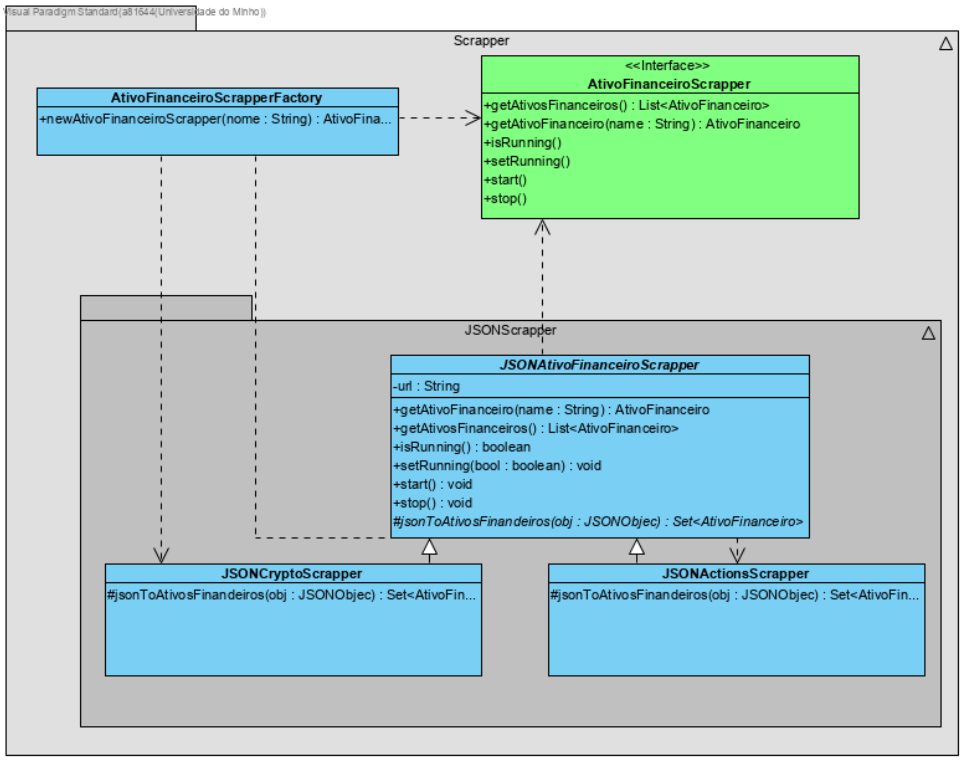
De realçar ainda que existe uma interface para a conexão à base de dados, isto é, caso se mude a base de dados a única coisa que se acrescentaria seria uma nova classe que implementava os métodos connect() e disconnect() não havendo necessidade de alterar outras conexões existentes.



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 subclasses 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.

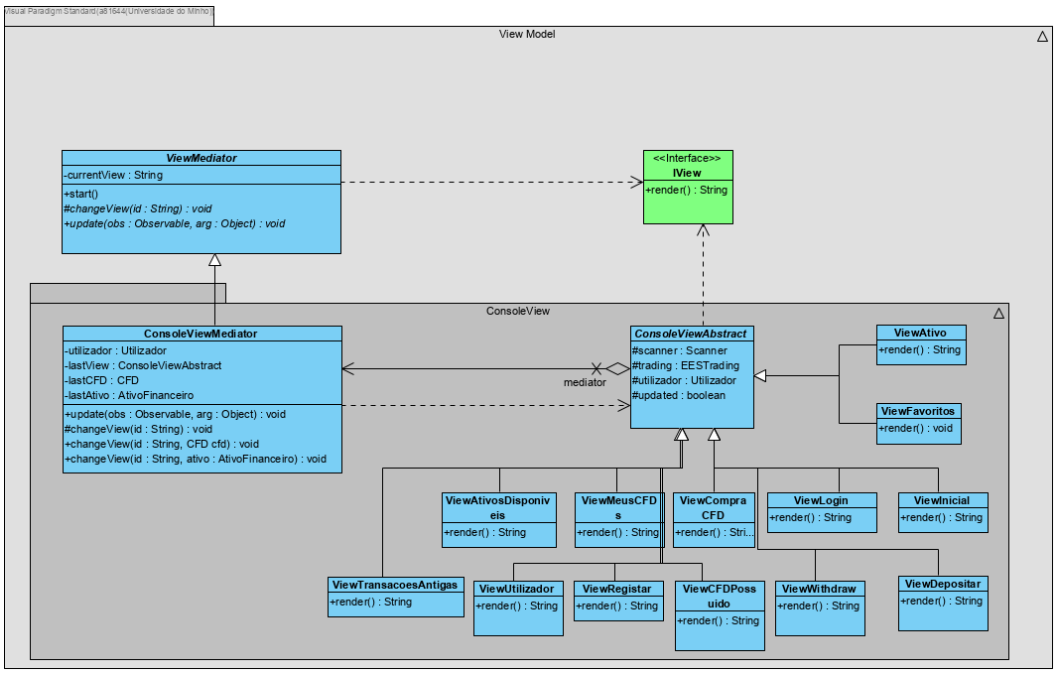
Este padrão poderia ser também utilizado nas classes dos DAOS visto que existe uma parte do código que é várias feita e poderia estar num método superior onde obrigasse apenas a implementar as partes que mudavam.

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.

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.

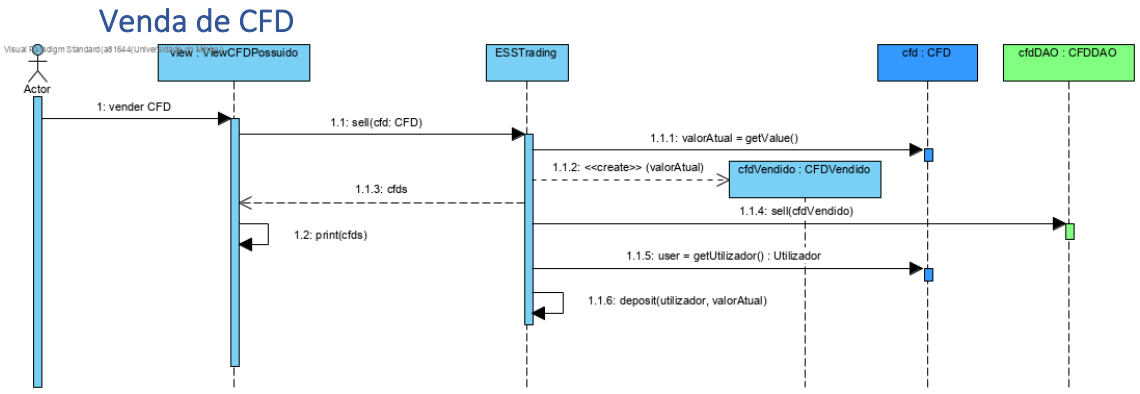


Figura 19 - Diagrama de sequência da venda de CFD

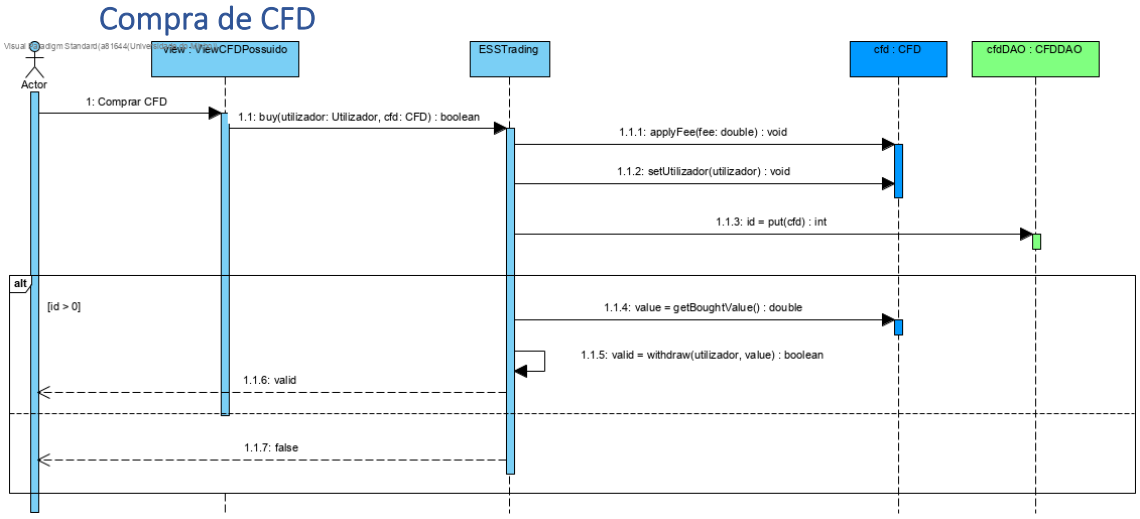


Figura 20 - Diagrama de sequência da compra de CFD

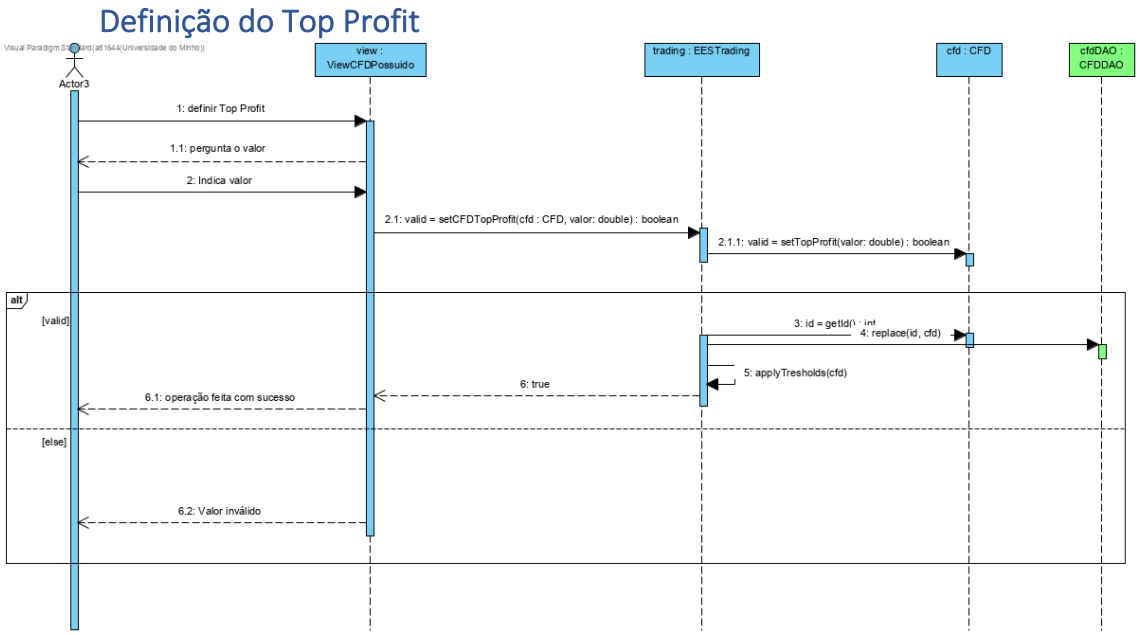


Figura 21 - Diagrama de sequência da Definição do Top Profit

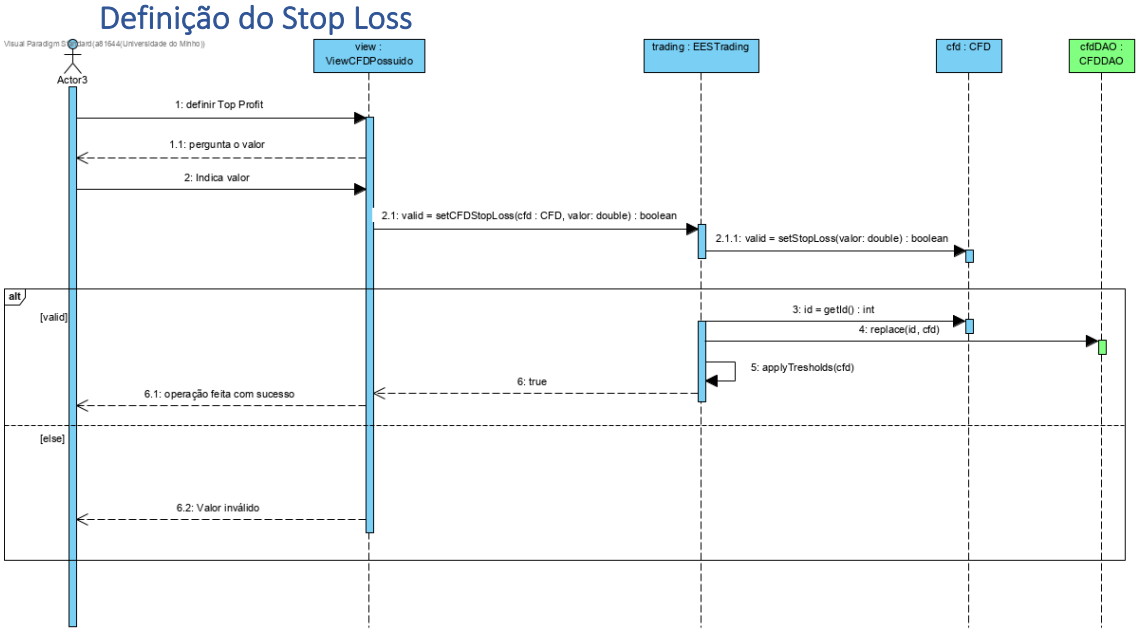


Figura 22 - Diagrama de sequência da Definição do Stop Loss

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.

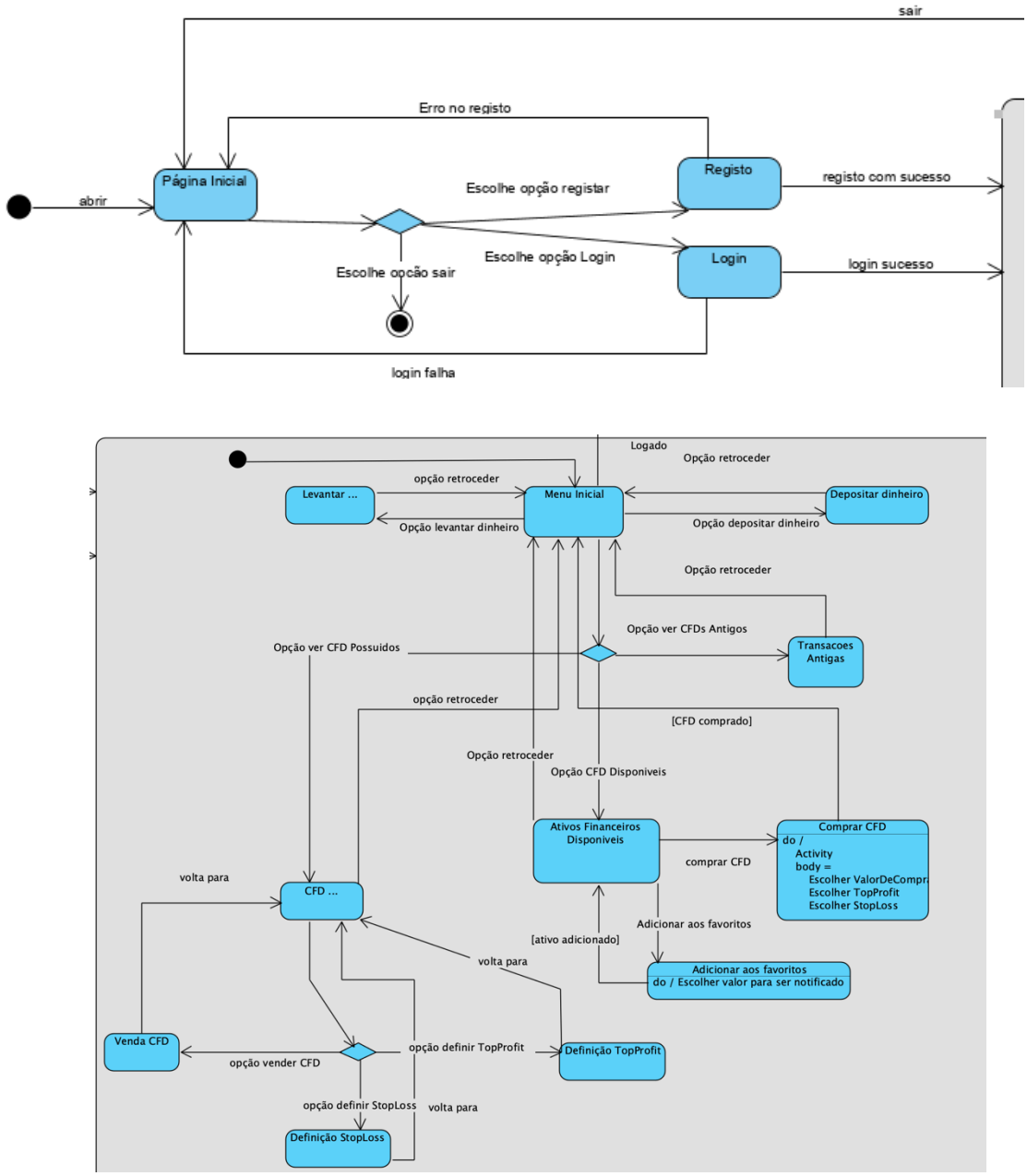


Figura 23 - Diagrama de estados

Requisito de última hora

Foi-nos proposto o seguinte requisito para implementar antes da entrega do produto: "permita aos "traders" seguir a cotação de um conjunto ativos, sendo notificados quando um desses ativos sofre uma variação significativa no seu valor".
Para concretizar este requisito supomos que o utilizador define o valor que quer ser notificado quando o ativo atinge esse mesmo valor.

Módulo do negócio

Para incorporar este requisito no modelo de negócios já existente foi adicionada uma lista, ao utilizador, de uma subclasse AtivoFinanceiroFavorito, que estende a classe AtivoFinanceiro. Nesta classe é guardada toda a informação do ativo financeiro e o valor que o é preciso para haver a notificação por parte do utilizador.
Ainda foram introduzidos métodos para manipular esta lista na classe cliente.

Módulo de persistência de dados

Para suportar esta nova funcionalidade teve-se que alterar o módulo de persistência e o módulo de apresentação dos dados.
No módulo de persistência de dados alterou-se apenas o método que atualizava a informação do utilizador e como essa informação era retirada, respetivamente o put(), o get() e a adição de métodos privados addFavorito(), removeFavorito() e putFavoritos() da implementação concreta do DAO do utilizador, de forma a que a nova lista fosse persistida.

Módulo de apresentação

No módulo de apresentação foram criados 2 novas *views*, uma para apresentar ao utilizador o que fazer com o ativo financeiro, comprar e adicionar ou remover o ativo financeiro da lista de favoritos e a outra para gerir e visualizar a lista de ativos financeiros favoritos.

Em suma, o módulo dos *scrappers* foi inalterado, foi introduzida uma nova classe e uma nova lista desta mesma classe no utilizador no modelo de negócios, 2 novas classes no módulo das views para representar e comunicaras ações do utilizador ao modelo de negócios e os métodos get() e put() do utilizador DAO foram atualizados.

Conclusão

Neste 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.

Depois deste processo elaborou-se arquitetura estabelecendo os principais módulos a implementar para a aplicação. Em seguida apresentaram-se todos os padrões de desenho utilizado na implementação das várias funcionalidades e justificaram-se o seu uso.

Passou-se ainda pela especificação em diagrama de sequência para explicar a forma como as funcionalidades mais importantes funcionam

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.

No final de todo este trabalho é possível retirar que a elaboração de uma boa arquitetura não é uma tarefa fácil pois acarreta várias implicações para a forma como se implementa a aplicação, é algo que tem de ser feito com bastante cuidado para que a sua implementação não se torne demasiado complicada. Torna-se claro que não existem arquiteturas perfeitas, mas sim arquiteturas com prós e contras que necessitam ser bem medidos para se alcançar uma boa arquitetura adequada às necessidades do sistema.