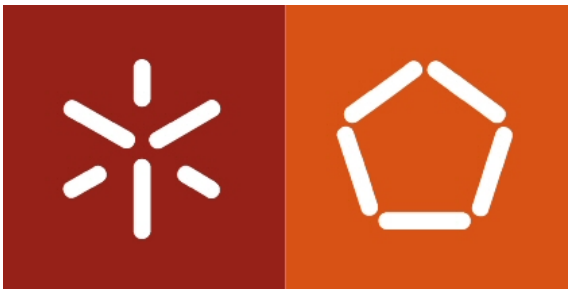

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Engenharia de Sistemas de Software

Arquiteturas de Software

ESS Trading Platform

Inês Alves (A81368)
Ricardo Caçador (A81064)

30 de Novembro de 2019

Resumo

No âmbito da Unidade Curricular de Arquiteturas de Software, enquadrada no perfil de especialização de Engenharia de Sistemas de Software, este projeto foi desenvolvido com o objetivo de abordar e compreender melhor os conceitos abordados nas aulas desta UC, sendo que se pretende implementar eventuais progressos no seu desenvolvimento à medida que vão sendo introduzidos novos conhecimentos.

Conteúdo

1 Introdução

O grande mote desta plataforma prende-se na sustentabilidade e suporte de negociações de ações financeiras referentes não só a *commodities* (ouro, prata, moeda...), mas também a acções de empresas e organizações (Google, Apple, IBM...).

Estas negociações terão por base o sistema de CFD - *Contract For Differences*, onde um comprador e um vendedor pactuam a diferença de valor de uma ação entre o tempo em que o contrato se encontra aberto. De notar que o comprador acaba por nunca tomar posse do bem em questão. O que acontece é que este recebe as receitas (ou assume as perdas, pagando ao vendedor) provenientes das flutuações de mercado desse bem.

Assim, a plataforma terá que ser capaz de manter os valores das ações a serem lidados via CFDs, permitir que duas forças motrizes deste tipo de negociações (compradores e vendedores) possam abrir contas com um saldo inicial de investimento e possam também estipular CFDs sobre as ações disponíveis. Por fim, repare-se que, tanto os compradores como os vendedores, poderão monitorizar o conjunto de CFDs em que atuam, assim como o valor de cada ação/posição.

2 Requisitos para o Sistema

Uma plataforma de negociação permite que os seus utilizadores (compradores e vendedores) realizem negociações de CFDs. Para a implementação de todas as funcionalidades da aplicação, foram levantados os seguintes requisitos:

- Um investidor deverá registar-se no sistema através de: e-mail, *username*, password, nome e idade. Neste registo terá ainda acesso ao seu *plafond* inicial;
- Uma vez registado, o investidor poderá iniciar sessão no sistema utilizando apenas o seu e-mail e a password definida aquando do seu registo;
- Ao aceder à aplicação, o investidor autenticado deverá ter disponível uma lista de eventos sobre os quais poderá agir;
- Um investidor deverá poder comprar e vender ativos;
- O sistema deverá conseguir, em tempo real, monitorizar a atividade de cada investidor;
- O sistema deverá ter uma lista com os valores mais atuais dos ativos adquiridos pelos investidores;
- Cada abertura de negociação deverá seguir o formato: valor de abertura, valor atual, quantidade negociada e, se aplicável, valores de "Take Profit"e "Stop Loss";
- Cada fecho de negociação deverá seguir o formato: valor de fecho e resultado da negociação;
- A plataforma deverá manter uma lista de ativos com que o investidor teve interação;
- Quando um negócio é fechado, todos os investidores devem ser notificado e os respetivos saldos atualizados consoante o resultado do negócio.
- Para determinar o valor ganho/perdido num investimento, deverá ser definido, para cada evento, as *odds* para os possíveis resultados e, sobre essas *odds*, será calculado o valor ganho/perdido.

A aplicacao esta idealizada para uso por parte de um utilizador apenas com acesso ao seu portfólio, nao havendo a possibilidade de consultar as posicoes de outro utilizador;

2.1 Requisitos de Qualidade

Cada requisito de qualidade é composto por 6 partes:

- **Stimulus:** condição que requer uma resposta quando chega ao sistema;
- **Stimulus source:** entidade que gera um estímulo;
- **Response:** atividade realizada como resultado da chegada do estímulo;
- **Response measure:** prova de que o requisito foi testado e que funciona;
- **Environment:** condições em que o estímulo é gerado;
- **Artifact:** o que é estimulado.

Qualidade	Motivação
Simplicidade	A plataforma deverá ser simples, permitindo que a sua utilização seja fácil e direta.
Eficiência	Os dados apresentados na plataforma devem sempre permanecer atualizados e sem falhas.
Correção	Os requisitos definidos sao fundamentais para o funcionamento base daplataforma e a sua validacão é fulcral.
Portabilidade	A plataforma tera de correrem ambientes diferentes, nao devendo ser confinada a uma maquina apenas.
Segurança	As informações de cada utilizador são únicas e nunca de acesso a outros.

Tabela 1: Requisitos de qualidade

Tendo estes tópicos como ponto de referência, bem como os requisitos levantados anteriormente, foram desenvolvidos os seguintes requisitos concretos de qualidade:

- **Performance**
  Sempre que o sistema apresentar problemas de performance, por exemplo, devido a um elevado n.º de acessos e não estiver a responder corretamente ou com a velocidade pretendida, a plataforma de negociação poderá ativar o modo lento de forma a poder controlar o acesso e assim libertar a fluidez da plataforma.

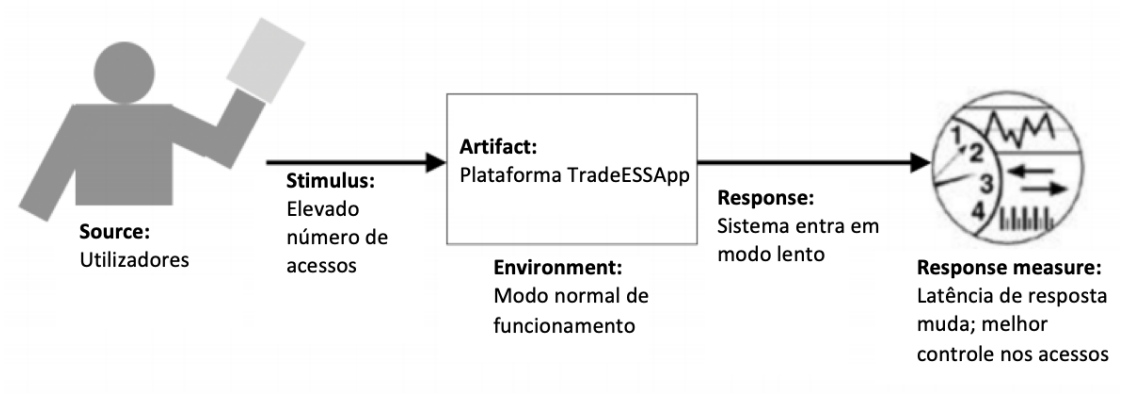

Figura 1: Requisito de Qualidade - Perfomance

- **Disponibilidade**
  No caso improvável de ocorrer uma falha na plataforma de negociação, o que acontece para garantir na mesma a perfomance do sistema é a reiniciação dos servidores responsáveis pela alocação da aplicação e, após 10 minutos, o sistema deve voltar a estar ativo, atualizado e 100% funcional.

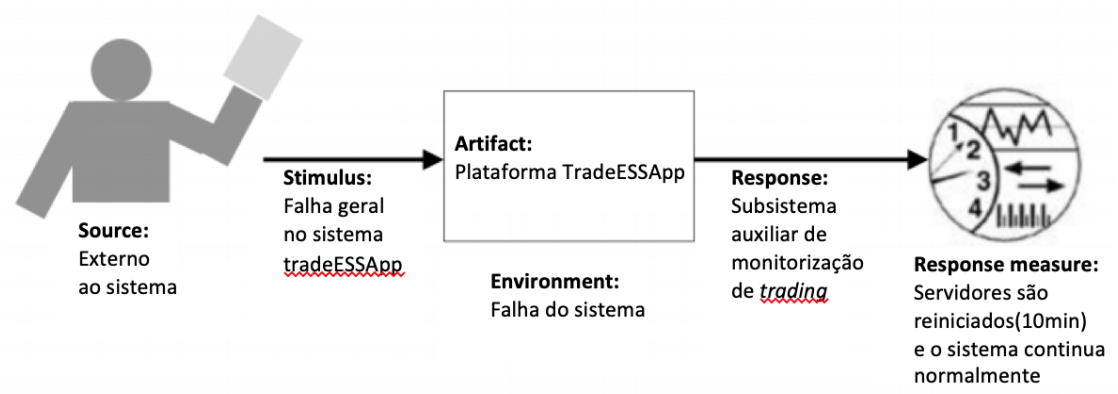

Figura 2: Requisito de Qualidade - Disponibilidade

- **Segurança**
  Quando um dado investidor indica que quer consultar um dado evento, é aberta uma nova janela com as informações públicas desse evento. No entanto, existem funcionalidades presentes nessa janela que apenas podem ser acedidas por um administrador do sistema (p.e: editar estado do evento). Então, antes de abrir essa janela, é verificado, previamente, se foi, ou não, um administrador a requer o acesso à página. Em caso afirmativo, ficam disponíveis as funcionalidades "extra"a que este tem direito.

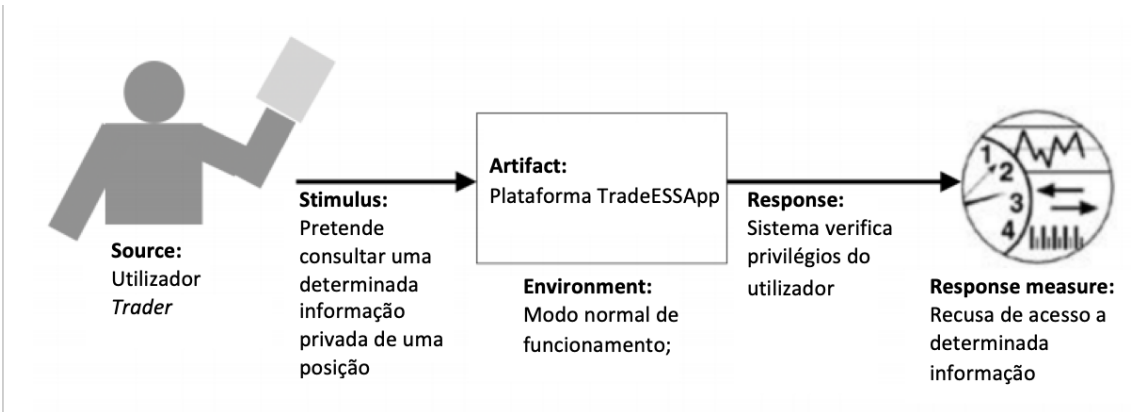

Figura 3: Requisito de Qualidade - Segurança

- **Modificação**
  No caso do administrador do sistema pretender alterar, por exemplo, algo numa posição, é necessário que esta alteração ocorra sem prejudicar o sistema. Desta forma, o sistema realiza as alterações pretendidas assegurando o seu bom funcionamento, sem atrasar as restantes operações e o estado do sistema é alterado instantaneamente.

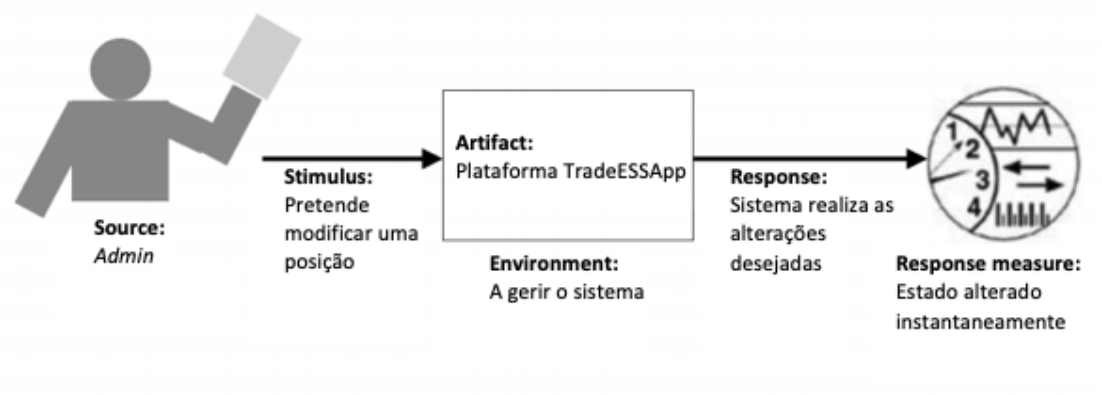

Figura 4: Requisito de Qualidade - Modificação

2.1.1 Público Alvo

Alvo	Objetivos
Docente	Para este trabalho o público alvo é o docente da UC, uma vez que esta plataforma procura ser algo simples apenas para fins de inicialização do projeto.

Tabela 2: Público alvo do sistema

2.1.2 Partes interessadas

Nome	Objetivos/Expectativas
Manager (Aluno)	Tomar decisoes acerta do sistema e do seu desenvolvimento e passá-las aos developers e arquitetos de software. Faz a negociacao dos requisitos da aplicacao.
Developer (Aluno)	Desenvolver a aplicacao com recurso a linguagem de programacao JAVA, sem aprofundar a parte grafica da mesma. O foco prende-se no cumprimento dos requisitos funcionais da aplicacao.
Trader (Utilizador)	Traders que procuram uma plataforma que lhes permita realizar as suas operacoes de investimento no mercado financeiro.
Arquiteto de Software (Aluno)	Procurar um exemplo de template que lhe permita organizar o seu trabalho.

Tabela 3: Stakeholders do sistema

2.2 Requisitos Não Funcionais

A fim de tornarmos a nossa implementação e compreensão do projeto mais lúcida, definimos alguns requisitos não funcionais que restringuem o trabalho de um arquiteto de *software*.

Restrição	Descrição
Implementação em Java	A plataforma de trading terá a sua implementação em Java, por isso é possível que surjam restrições inevitáveis para a implementação da solução. Estas restrições advêm da própria linguagem de programação.
Imparcialidade	A plataforma de trading terá de ser imparcial, isto é, independentemente do sistema operativo utilizado, deverá funcionar de igual forma e com bom desempenho.
Execução por terminal	A plataforma de trading não terá grande desenvolvimento gráfico, pelo que toda a sua execução/utilização deverá partir segundo uma linha de comandos em terminal.
Calendarização	O desenvolvimento da plataforma será fortemente influenciado pelo tempo disponível para a sua implementação.
Documentação da Arquitetura	Toda a implementação da plataforma de trading será documentada fazendo uso da ferramenta javadoc.

Tabela 4: Requisitos não funcionais

2.3 Contexto do Sistema

Ainda antes de partirmos para a implementação do sistema propriamente dita, é necessário fazermos referência aos elementos externos implícitos ao funcionamento da plataforma, sendo que estes interagem com ele. Consideremos, então, os seguintes fatores:

- A plataforma de *trading* terá que ser executada pelo terminal ou fazendo uso de um IDE, uma vez que não fizemos uso de nenhuma ferramenta de implementação de interfaces gráficas (ex.: Java SWING). Assim sendo, o nosso sistema possui uma interface gráfica bastante rudimentar, como iremos mostrar mais adiante;
- Quanto à base de dados que suporta o sistema, esta foi criada com a ajuda da ferramente SQLiteStudio. Inicialmente, tinhamos feito uso da ferramenta MySQL, no entanto, esta mostrou-se bastante mais complexa do que a nova implementação. Com o SQLiteStudio não temos a preocupação de verificar *passwords* para executar o programa. Uma vez que somos um grupo de 2 pessoas, era necessário estarmos a confirmar sempre se a *password* utilizada era a correta, relevando-se uma desvantagem em termos de comodidade. No entanto, a fim de executar o programa, é nessário ter a base de dados criada pelo grupo (database.db);
- Uma vez que a implementação do código foi feita na linguagem de programação Java, é necessário ter o JDK 11 instalado na máquina utilizada, a fim de compilar corretamente o código.

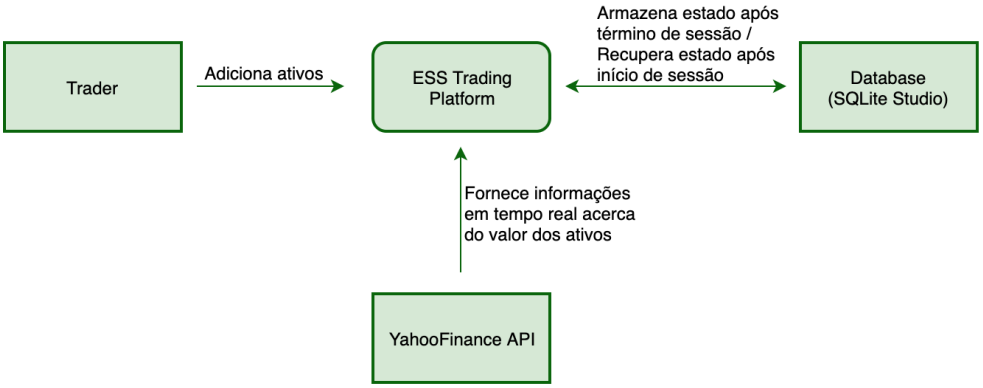

Figura 5: Contexto do sistema

- **Trader:** Um utilizador que pretenda investir no mercado financeiro deve registar-se na plataforma, podendo depois adicionar os ativos presentes na mesma ao seu portfólio, com as restrições pretendidas;
- **YahooFinance API:** API responsável por fornecer valores em tempo real dos ativos presentes na plataforma;
- **Database (SQLite Studio):** O estado da plataforma aquando do término da sessão é guardado e recuperado quando o utilizador volta a iniciar sessão, recalculando o valor dos seus ativos e tendo em conta as alteracoes no mercado financeiro enquanto esteve "offline".

3 Desenho da Solução

Estando feito o levantamento de requisitos para a plataforma de negociações TradeESSApp, foram desenvolvidos, através do uso da linguagem UML (Unified Modelling Language), as seguintes vistas de estrutura, comportamento e alocação:

- **Modelo de Domínio**
- **Diagrama de Use Cases**
- **Diagramas de Sequência de Subsistema**
- **Diagrama de Classes**
- **Diagrama de Packages**
- **Diagrama de Instalação**

4 Modelo de Domínio

Numa primeira abordagem e sendo este um dos requisitos enunciados no enunciado do projeto, construímos o modelo de domínio associado à nossa plataforma de *trading*.

Assim, pela análise do enunciado, foi-nos possível identificar algumas entidades principais: **User** e **Asset** e, à posteriori, todos os componentes que completavam o sistema e as entidades necessárias.

A construção do modelo de domínio facilitou-nos bastante a modelação do projeto, projetanto o que seriam as possíveis classes da nossa plataforma.

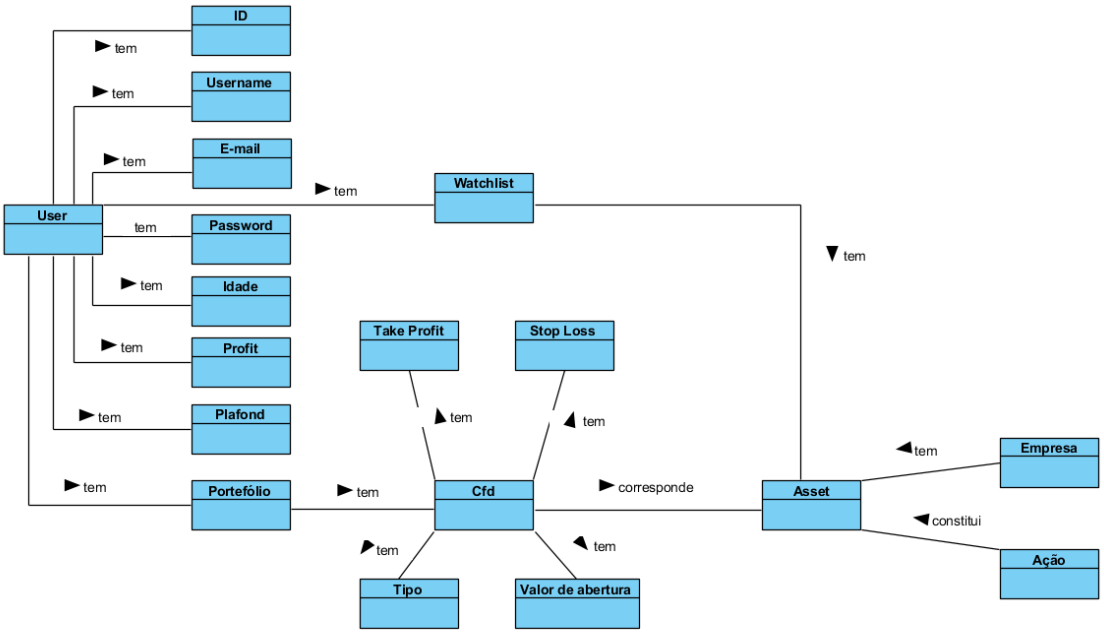

Figura 6: Modelo de Domínio

5 Diagrama de Use Cases

A partir do modelo de domínio apresentado na secção anterior, conseguimos, com maior facilidade, modelar o nosso modelo de Use Cases para o sistema em questão. Para este efeito, decidimos implementar apenas um ator no sistema: ***Trader***. Como podemos perceber, o *Trader*, isto é, o investidor, é, no nosso entender, o principal utilizador do sistema, uma vez que será ele que irá dar-lhe o maior uso. Ainda que na versão anterior deste trabalho tenha sido implementado um ator *Admin* que se tratava de um administrador responsável pela criação e manipulação de dados do sistema, a decisão de o absolver foi unânime.

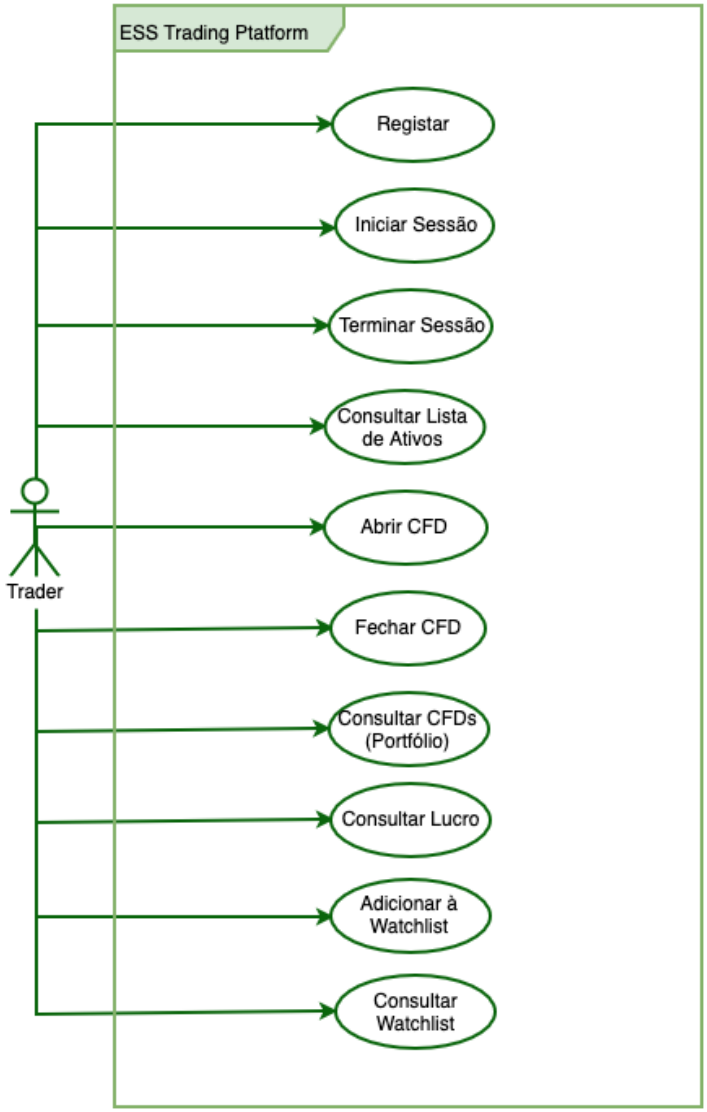

Figura 7: Diagrama de Use Cases

6 Diagrama de Classes

A partir do modelo de domínio, do diagrama de Use Cases apresentados anteriormente, foram identificadas as principais entidades do sistema e, consequentemente, potenciais candidatas a serem classes no desenvolvimento desta plataforma de *trading*. Como tal, passamos agora para a construção do Diagrama de Classes, dando mais um passo no desenvolvimento deste sistema de software.

Para realizar a correta construção deste diagrama foi necessário ter em atenção as classes que o nosso sistema iria possuir, bem como os métodos de que as mesmas iriam fazer uso.

Um diagrama de classes consiste na representação estrutural das classes que servem de modelo para os objetos do sistema, explicitando também as interações entre elas e o papel que cada uma tem na realização das operações solicitadas pelos utilizadores do mesmo.

Cada classe representa um grupo de objetos que partilham o mesmo tipo de estrutura (atributos e relacionamentos) e as mesmas operações, sendo que estas podem restar relacionadas umas com as outras através de associações.

Assim, construimos um modelo da aplicação onde é implementada a persistência através das diversas classes DAOs, ou seja, guardando os dados numa base de dados. De notar que todas as classes possuem os respetivos Getters e Setters das variáveis de instância, apesar de não estarem representadas no diagrama de classes abaixo apresentado, a fim de não prejudicar a perceção deste.

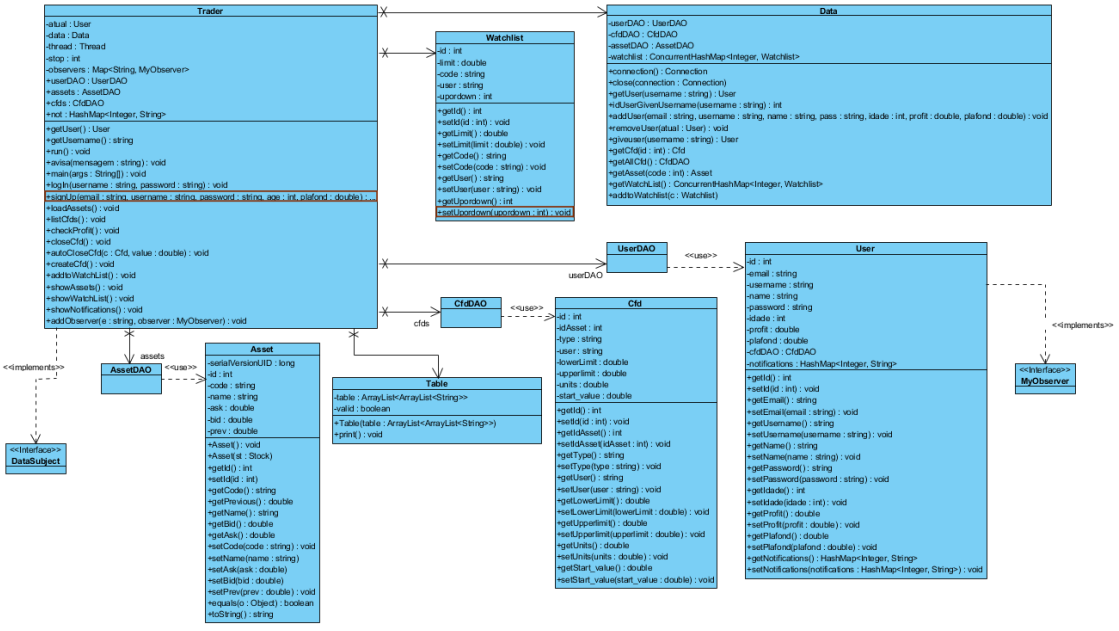

Figura 8: Diagrama de Classes

6.1 Classes

6.1.1 *User*

Classe que representa um utilizador geral da plataforma, contendo as suas características principais como o nome, e-mail, password, idade, *plafond*, entre outras...

6.1.1.1 *Trader:*

Subclasse de *User*. Esta classe representa um utilizador sem privilégios, isto é, sem permissões para realizar modificações nos dados do sistema. Nesta classe, para além de conseguimos aceder às características referidas da super classe, fornece ainda métodos para gestão dos ativos e cálculo dos valores aquando da abertura/fecho dos CFDs, bem como métodos que se realizam, efetivamente, todas as operações de registo, início de sessão, término de sessão, etc...

6.1.2 *Cfd*

Esta classe contém toda a informação que suporta a abertura de uma posição de compra ou venda por parte do utilizador da aplicação. Desta forma, se o utilizador decidir abrir uma posição de compra, será criada uma nova posição (*Position*) com *"type = buy"*, indicando, desta forma, que se trata de uma posição de compra, com o ID na base de dados da ação que foi solicitada para compra, da quantidade desejada para compra, do *stop loss* e *take profit* definidos pelo *trader* para a compra e do valor dessa mesma. Existe também um parâmetro *active* que serve para verificar se, no momento da abertura da posição de compra, foi possível, de imediato, fazer o negócio. Para isto, o parâmetro pode assumir os valores 1 ou 0, isto é, aberto ou fechado, respetivamente.

Posto isto, caso o utilizador faça um pedido de venda à plataforma, o raciocínio é o mesmo, sendo que a única diferença é que a posição criada terá *type="short"*.

Note-se que nas classes referidas anteriormente existem também a si associados identificadores únicos (IDs) utilizados, maioritariamente, para efetuar pedidos à base de dados, visto que as pesquisas são feitas fazendo uso dos mesmos.

6.1.3 *Asset*

Representa o mercado de ativos existente. Esta classe contém toda a informação que suporta a existência de uma ação na plataforma. Como tal, foi considerado que um *Asset* teria:

- um ID;
- um *code*, que corresponde ao símbolo da ação na API;
- um *name*, que corresponde ao nome da companhia que possui a ação na API;
- um *bid*, corrrespondendo ao preço de compra da ação na API;
- um *ask*, correspondente ao preço de venda da ação na API;
- um *prev*, que mostra qual era o preço de um cfd quando o mercado fechou anteriormente.

Todos estes parâmetros criados aquando do registo de uma ação são registados na base de dados e permanecem imutáveis.

6.1.4 Table

Classe utilizada para criar tabelas gráficamente apelativas na linha de comandos, tornando mais fácil a visualização de todos os dados.

Todas estas classes encontram-se na camada de negócio - *Business Package*. Foram ainda implementados outros *packages*, utilizados para guardar dados na base de dados e desenvolvimento de interfaces do sistema.

6.2 Persistência

Como já foi referido, a plataforma *ESS Trading Platform* faz uso da ferramenta **SQLite Studio** para guardar os dados e o estado da mesma.

6.3 User Interface

Também como foi já referido, a interface do utilizador é a linha de comandos da máquina utilizada ou a consola de um IDE à escolha do utilizador.

6.4 Tratamento de Erros/Exceções

Como é natural e altamente provável, erros nas introdução de dados podem acontecer, podendo afetar o funcionamento normal da plataforma. Para isso, estes potenciais erros são tratados através da implementação de exceções para os seguintes casos:

- Utilizador Inexistente;
- Utilizador já registado;
- Palavra-passe inválida

7 Diagrama de Packages

Uma vez que todas as classes já se encontram definidas, podemos, agora, visualizar a integração e relacionamento entre elas através de um Diagrama de *Packages*.

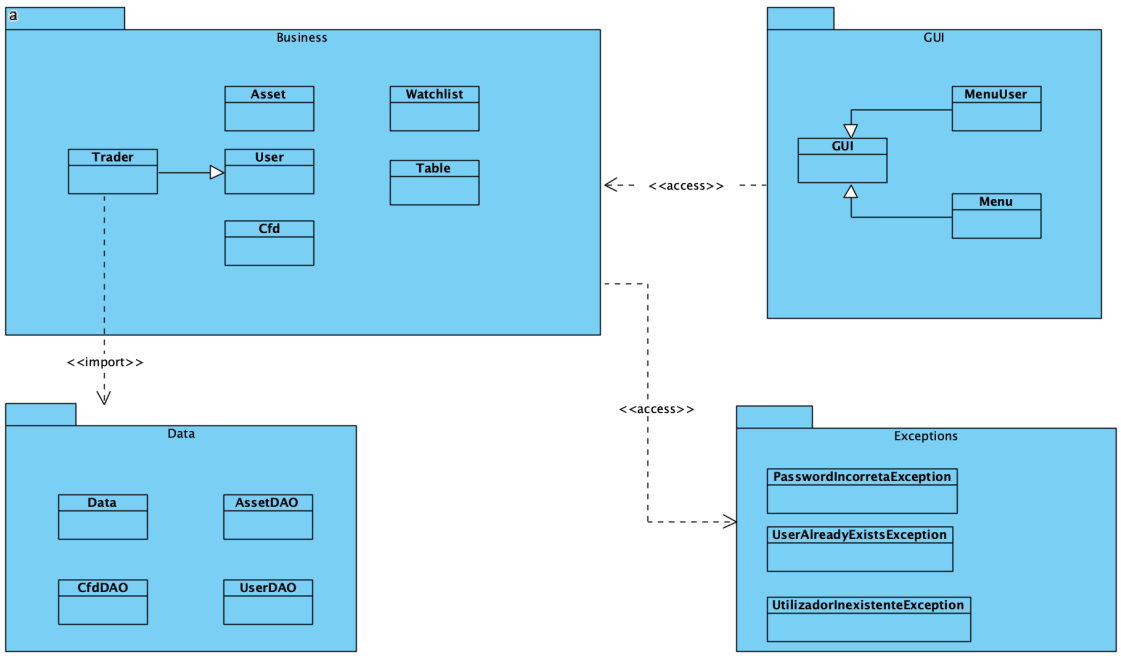

Figura 9: Diagrama de Packages

Este diagrama "resume"a modelação realizada até ao momento, traduzindo a arquitetura usada, bem como o padrão principal que trata da comunicação entre os diversos componentes

do sistema. Cada *package* corresponde a um mecanismo de agrupamento genérico com elementos relacionados, sendo que estes podem ser classes, interfaces, entre outros...

É importante notarmos ainda que, neste tipo de diagramas, os elementos de um determinado *package* podem ser, ou não, vísiveis para outros fora dele. Os *packages* podem também apresentar dependências entre si, caso a alterção de um afete outro.

7.1 *Business Package*

Package relativo à lógica do negócio, responsável pela comunicação entre a camada de persistência e a camada de apresentação.

7.2 *GUI Package*

Contém todas as classes responsáveis pela interface com o utilizador, fazendo a comunicação entre o utilizador e a plataforma.

7.3 *Data Package*

Responsável pela persistência de todos os dados e informação sobre a plataforma, estabelecendo comunicação com a camada de negócio através do *facade*. Então, aqui são implementados todos os DAOs.

8 Diagramas de Sequência de Subsistema

Definidos os subsistemas através do Diagrama de *Packages*, foram definidos também alguns diagramas de subsistema relativos a alguns *use cases*.

8.1 Registar utilizador

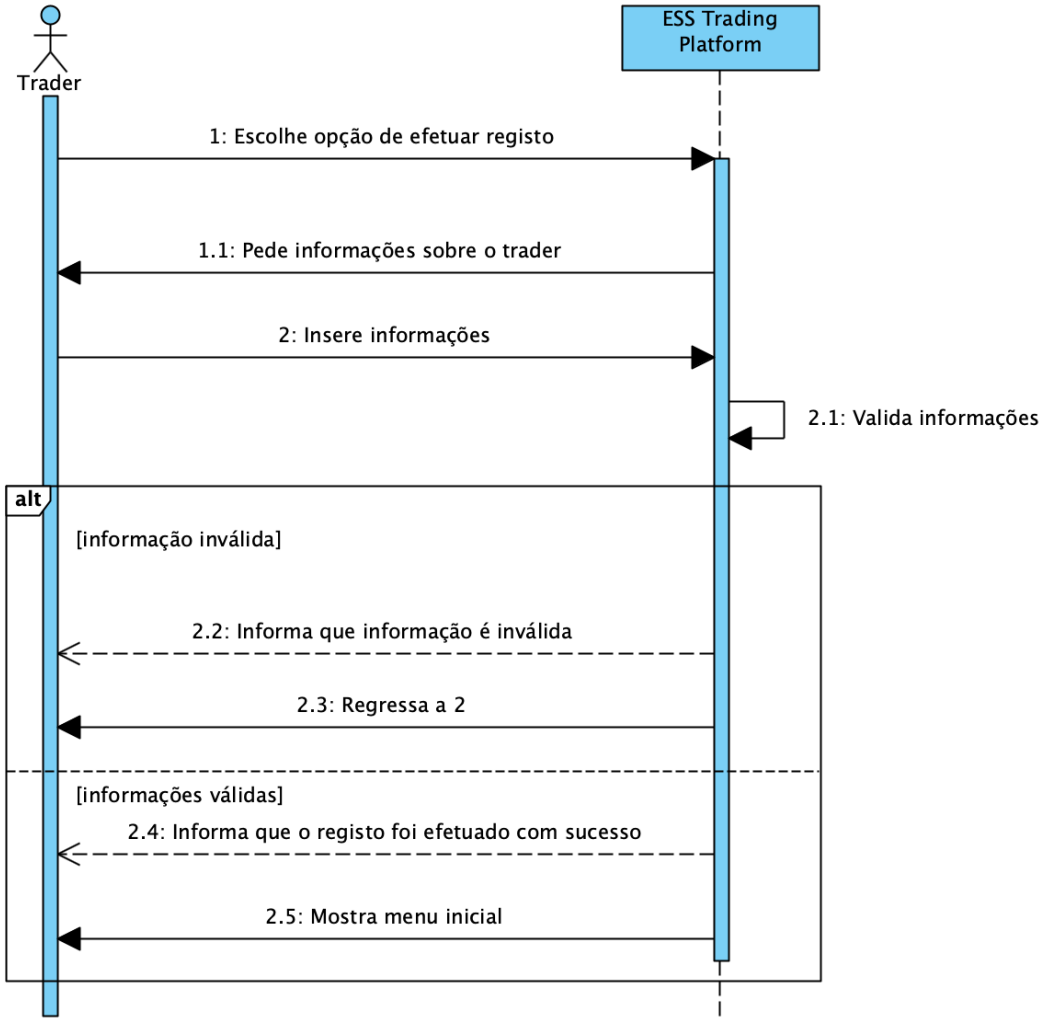

Figura 10: Diagrama de Sequência de Subsistema: Registar utilizador

8.2 Consultar ativos

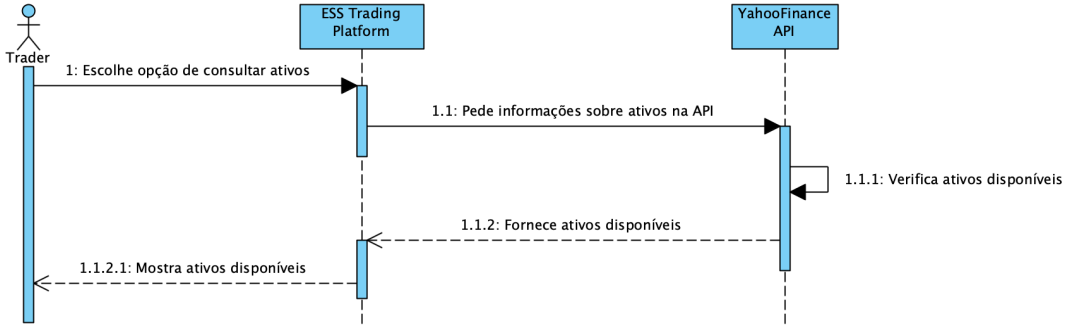

Figura 11: Diagrama de Sequência de Subsistema: Consultar ativos

8.3 Consultar CFDs

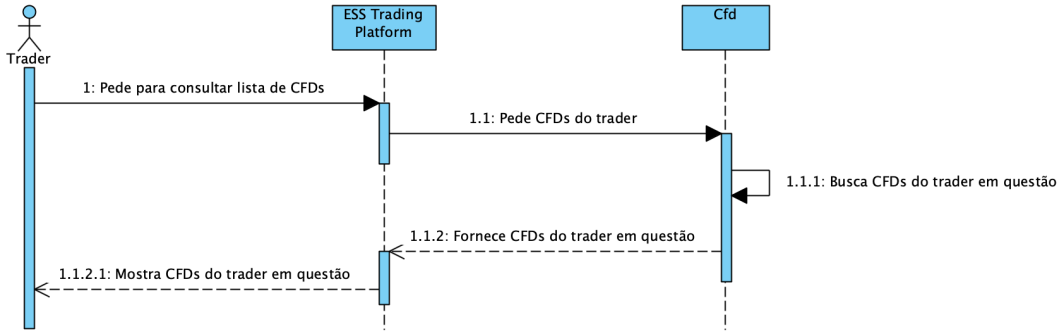

Figura 12: Diagrama de Sequência de Subsistema: Consultar CFDs

8.4 Fechar CFD

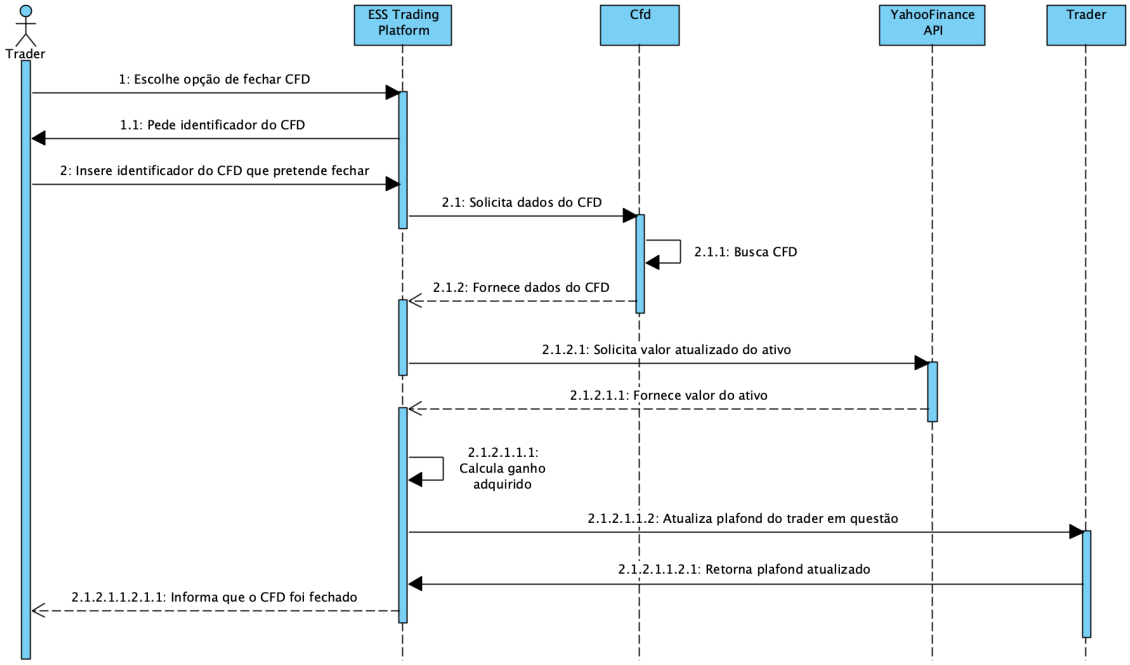

Figura 13: Diagrama de Sequência de Subsistema: Fechar CFD

9 Diagrama de Instalação

Nesta fase, procedemos à esquematização do sistema de *software* ao nível dos componentes físicos que ele abrange, modelando, por isso, a sua topologia de *hardware*. Apesar do ponto fulcral do UML ser a estruturação e o comportamento do *software* de um sistema, este tipo de diagramas aborda também a porta da aplicação a ser desenvolvida, demonstrando os componentes *hardware* nos quais o *software* desenvolvido se encontra implementado.

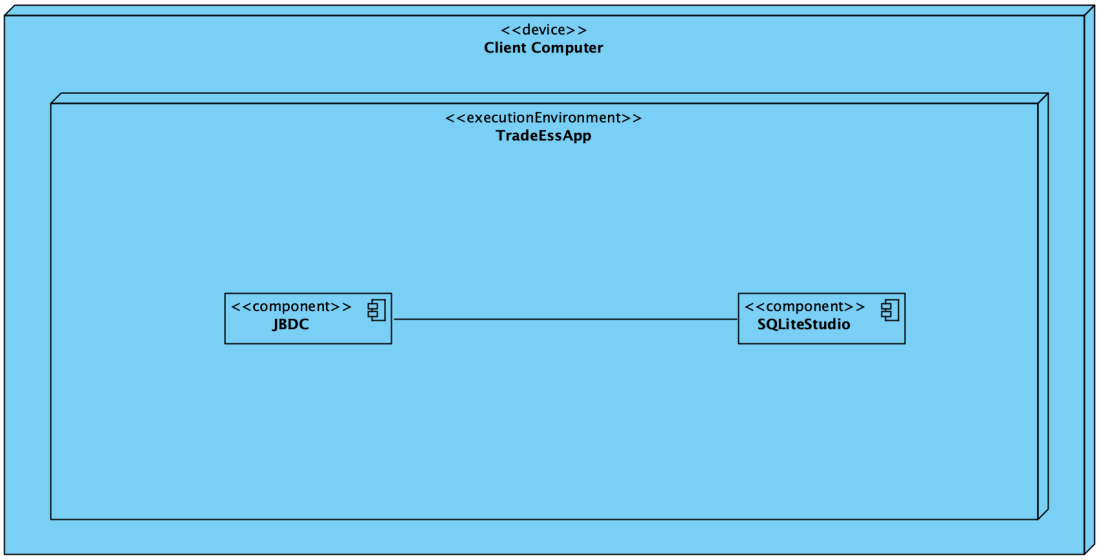

Figura 14: Diagrama de Instalação

10 Diagrama de Componentes

Por fim, optamos por criar um diagrama de componentes, facilitando a compreenssão de toda a plataforma.

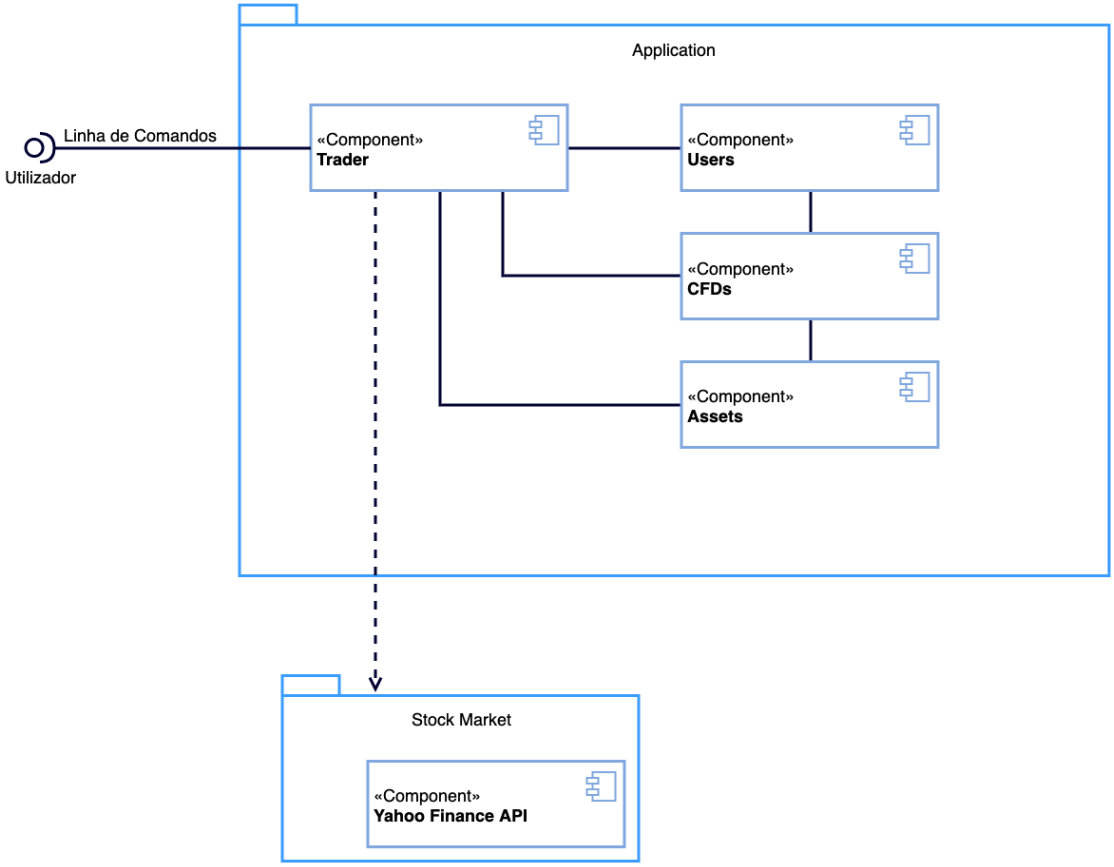

Figura 15: Diagrama de Componentes

11 Alterações relativamente ao trabalho anterior

1. **Inclusão de *threads* para verificação de valores**

   Um requisito importante que não estava concluído na versão anterior deste trabalho é a verificação contínua dos valores de *"take profit"* e de *"stop loss"* deifinidos pelo *trader* em questão. Tal como já havia sido referido, a resolução deste problema consistia, principalmente, na inclusão de uma *thread* que começa a sua inicialização logo que a aplicacação começa a correr. É feita uma verificação destes de 1 em 1 segundo e, no caso de algum deles ser alcançado, é realizado um fecho automático do CFD. Esta *thread* é também utilizada para verificar o valor dos ativos que o *trader* colocou na sua *watchlist*, sendo notificado no devido tempo;

2. **Inclusão do novo requisito: Watchlist**

   Para realizar a correta implementação do novo requisito, foram feitas algumas modificações na classe *user* e na classe *trader*, bem como na classe *data*. Para além disso, como seria de esperar, foi criada uma nova classe *Watchlist* que contém as informações sobre o *Asset* para o qual o utilizador deseja ser notificado. Por omissão, e definido por nós, o utilizador receberá uma notificação sempre que houver uma variação de 5% quer para uma subida ou para uma descida do valor. Também para este requisito foi necessária a implementação de um *ConcurrentHashMap* para guardar as alterações de cada utilizador;

3. **Inclusão do *Observer Pattern***

   Por fim, para aplicar o *observer pattern* optamos por criar duas interfaces: *MyObserver* e *DataSubject*. A primeira é uma interface do *User* e a segunda do *Trader* que acaba por ser a classe fundamental de todo este projeto. Os observadores são notificados quando existe um valor de *"take profit"* ou de *"stop loss"* que é atingido, fechando, automaticamente, o CFD.

12 Interface

Quando iniciamos a nossa plataforma podemos encontrar o seguinte menu:

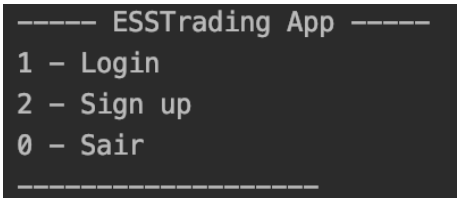

Figura 16: Menu Principal da plataforma

De seguida, podemos efetuar o registo de um utilizador:

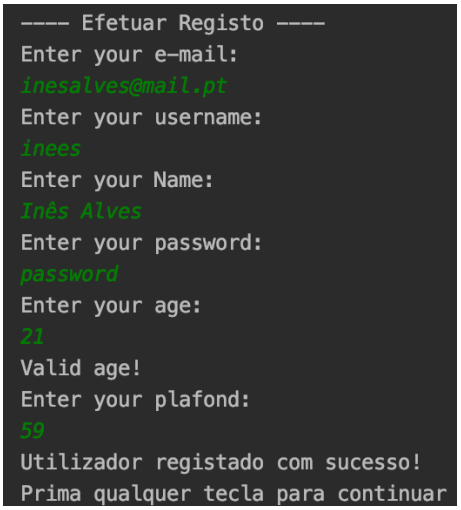

Figura 17: Menu de Registo de um utilizador

E, então, se informarmos o sistema que queremos iniciar sessão, é-nos apresentado o seguinte menu:

Figura 18: Menu de funcionalidades de um utilizador

A partir daqui, podemos usufrir de todas as funcionalidades deste sistema, como é o caso da consulta de CFDs:

NR	CODE	NAME	ASK [$]	BID [$]	CLOSED PRICE [$]
1	FB	Facebook, Inc.	201.53	201.5	202.0
2	AAPL	Apple Inc.	266.09	266.06	267.84
3	TSLA	Tesla, Inc.	329.26	328.79	331.29
4	DE	Deere & Company	168.74	168.51	169.06
5	GOOG	Alphabet Inc.	1309.98	1308.43	1312.99
6	ORCL	Oracle Corporation	56.28	56.27	56.61
7	PFE	Pfizer Inc.	38.65	38.64	38.63
8	IBM	International Business Machines Corporation	133.35	133.33	133.77
9	PBR	Petroleo Brasileiro S.A. - Petrobras	14.8	14.79	14.8
10	OI	Owens-Illinois, Inc.	9.72	9.71	9.66
11	GDDY	GoDaddy Inc.	66.77	66.7	66.58
12	VZ	Verizon Communications Inc.	60.33	60.32	60.1
13	ATC.AS	Altice Europe N.V.	CLOSED	CLOSED	5.294
14	AIR.PA	Airbus SE	CLOSED	CLOSED	133.62

Figura 19: Menu referente à consulta de CFDs

13 Conclusão e Análise Crítica

Realizado todo o planeamento e modelação, podemos afirmar que o trabalho aqui apresentado respeita os objetivos mencionados no enunciado do projeto.

A fim de termos uma plaforma de *trading* fiável, optamos, tal como tínhamos tentado fazer na fase anterior, por fazer uso de uma API (Yahoo Finance), conseguindo, desta forma, dados reais e de confiança. O objetivo passou por tornar possível introduzir CFDs na base de dados, de compra e de venda, introdução correta de certos valores das ações provenientes da API, consultar posições que o utilizador tenha na sua posse, entre outras...

A implementação do *Observer Pattern* revelou-se um pouco mais confusa do que aquilo que estariamos à espera, tornando todo o seu desenvolvimento dependente da interpretação que o grupo deu ao problema.

Para além disso, relativamente ao novo requisito, optamos por fazer uso de um *ConcurrentHashMap* em vez de um *HashMap* rudimentar, já que, durante o desenvolvimento do projeto, nos deparamos com diversos problemas de memória. Isto acontecia, no nosso entender, porque modificar um *HashMap* enquanto o mesmo está a ser iterado, não é uma boa prática, gerando inconscistência nos dados.

Ainda assim, apesar de algumas limitações desta implementação, fazemos um balanço positivo deste trabalho.