

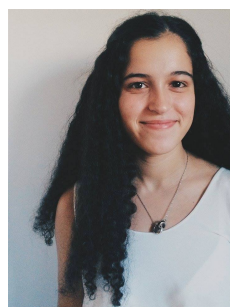
UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

ENGENHARIA DE SISTEMAS DE SOFTWARE

ARQUITETURAS DE SOFTWARE

ESS Trading Platform



Inês Alves (A81368)
Ricardo Caçador (A81064)

24 de Outubro 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	3
2	Requisitos para o Sistema	4
2.1	Requisitos de Qualidade	4
2.1.1	Público Alvo	7
2.2	Requisitos Não Funcionais	7
2.3	Contexto do Sistema	7
3	Desenho da Solução	8
4	Modelo de Domínio	8
5	Diagrama de Use Cases	9
6	Diagrama de Classes	10
6.1	Classes	10
6.1.1	<i>User</i>	10
6.1.2	<i>Position</i>	11
6.1.3	<i>Stock</i>	11
6.1.4	<i>API</i>	11
6.1.5	<i>TradeEssApp</i>	11
7	Diagrama de Packages	12
7.1	<i>Business Package</i>	12
7.2	<i>GUI Package</i>	12
7.3	<i>Data Package</i>	12
8	Diagramas de Sequência de Subsistema	13
8.1	Iniciar Sessão	13
8.2	Registar Utilizador	13
8.3	Abrir posição de compra	14
9	Diagrama de Instalação	14
10	Interface	15
10.1	Menu Principal	15
11	Conclusão e Análise Crítica	17

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 ações 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, password e nome;
- 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, data e hora de abertura e, se aplicável, valores de "Take Profit" e "Stop Loss";
- Cada fecho de negociação deverá seguir o formato: valor de fecho, resultado da negociação e data e hora do fecho da posição;
- 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.

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.

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.

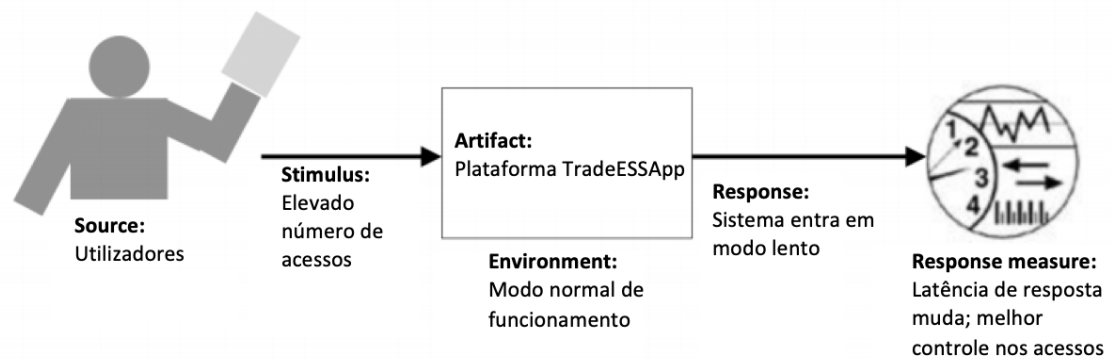


Figura 1: Requisito de Qualidade - Performance

- **Disponibilidade**

No caso improvável de ocorrer uma falha na plataforma de negociação, o que acontece para garantir na mesma a performance 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.

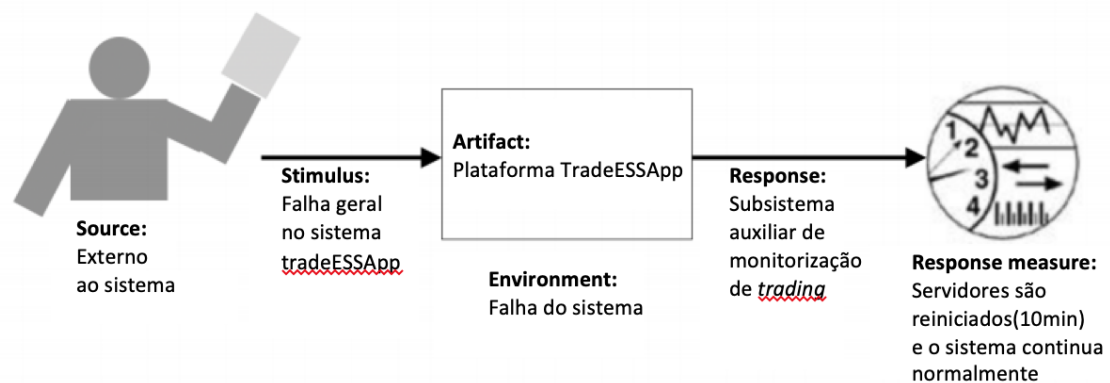


Figura 2: Requisito de Qualidade - Disponibilidade

- **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.

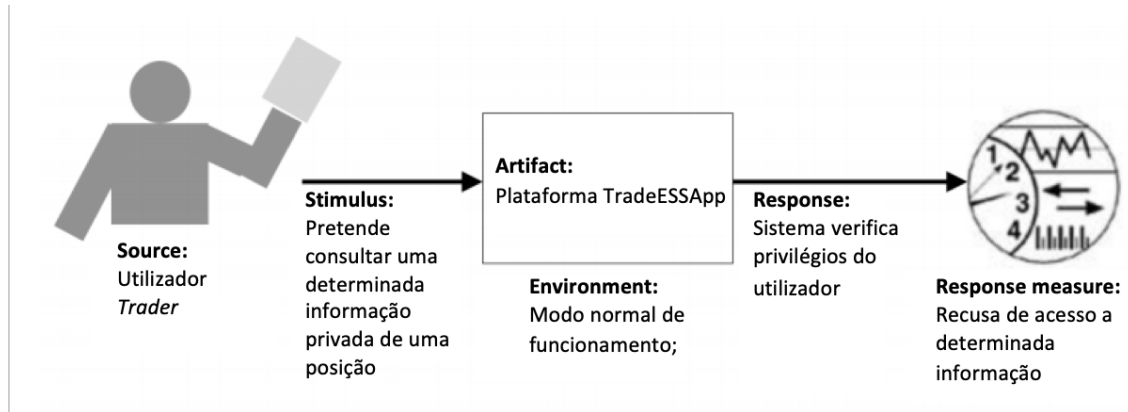


Figura 3: Requisito de Qualidade - Segurança

- **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.

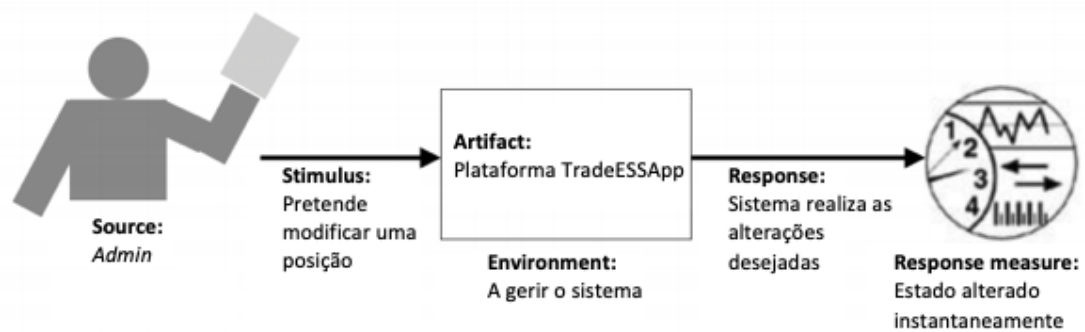


Figura 4: Requisito de Qualidade - Modificação

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

2.2 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.

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 ferramenta SQLiteStudio. Inicialmente, tínhamos 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 necessário 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.

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 **Position** 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, projetando o que seriam as possíveis classes da nossa plataforma.

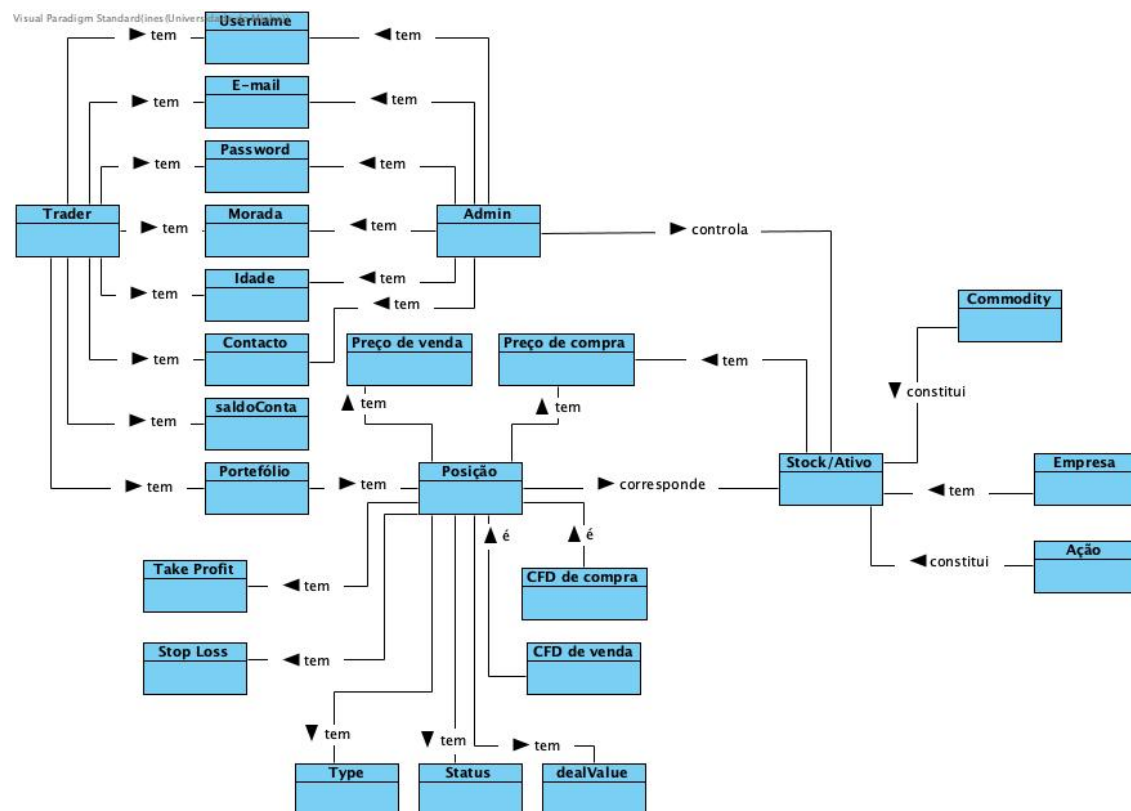


Figura 5: Modelo de Domínio

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 dois atores no sistema: *User* e *Admin*. Como podemos perceber, o *User* é, no nosso entender, o principal utilizador do sistema, uma vez que será ele que irá dar-lhe o maior uso. Já o *Admin* trata-se de um administrador responsável pela criação e manipulação de dados do sistema.

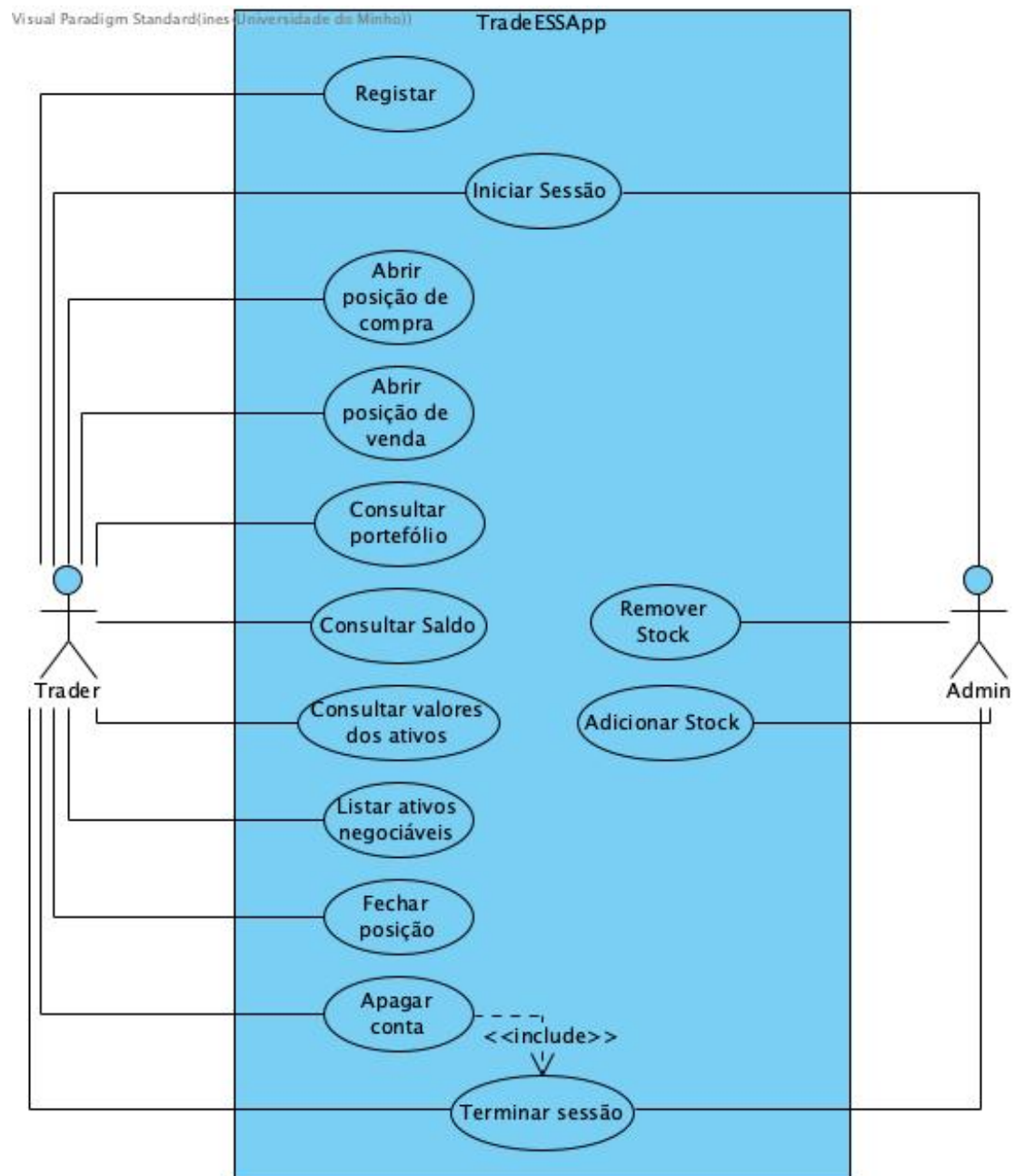


Figura 6: Diagrama de Use Cases

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 estar relacionadas umas com as outras através de associações.

Assim, construímos 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.

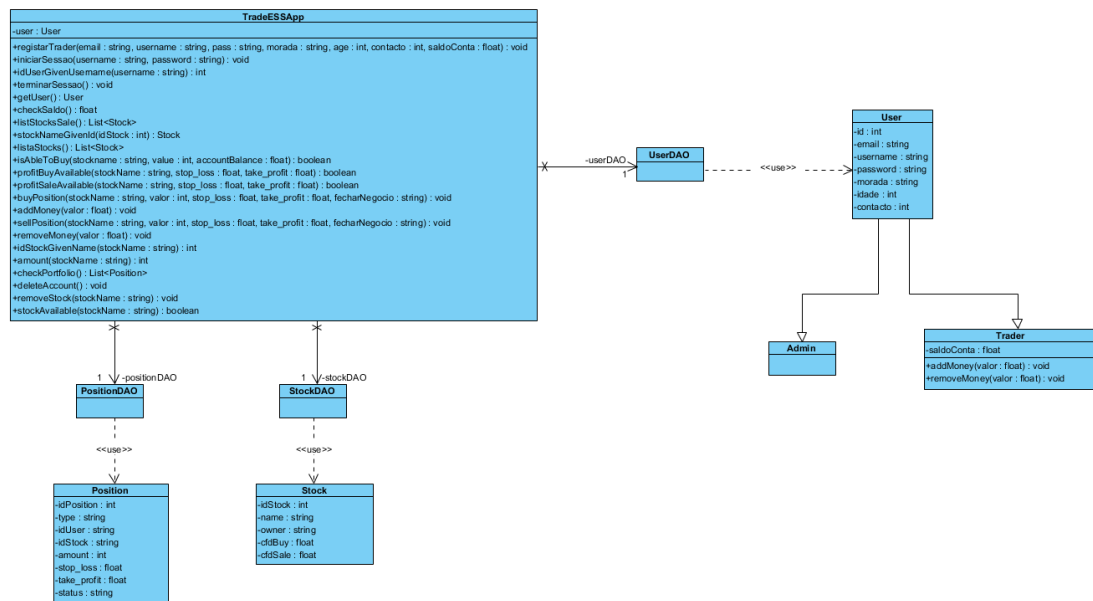


Figura 7: Diagrama de Classes

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, contacto, 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 conseguirmos aceder às características referidas da super classe, temos ainda acesso ao saldo da conta do *trader*.

6.1.1.2 Admin: Subclasse de *User*. Representa um utilizador com privilégios, isto é, com permissões para realizar modificações nos dados do sistema. É o administrador do sistema.

6.1.2 Position

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 = Para compra*, 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 *status* 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 "Negócio Fechado" se a compra for realizada; ou "Em espera", no caso do pedido de compra ficar em fila de espera, pois pode que não ser um negócio rentável naquele momento.

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 = "Para venda"*.

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 Stock

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

- um ID;
- um *name*, que corresponde ao símbolo da ação na API;
- um *owner*, que corresponde ao nome da companhia que possui a ação na API;
- um *cfdBuy*, correspondendo ao *bid* da ação na API;
- um *cfdSale*, correspondente ao *ask* da ação na API.

Todos estes parâmetros criados aquando do registo de uma ação são registados na base de dados e permanecem imutáveis. Idealmente, os valores de *cfdBuy* e *cfdSale* iam sendo atualizados pela API em certos períodos de tempo (através de threads de monitorização), mas a sua implementação não foi efetuada, deixando esta *feature* para uma fase seguinte;

6.1.4 API

Entidade representativa da API utilizada;

6.1.5 TradeEssApp

Classe principal desta plataforma. É aqui que se realizam, efetivamente, todas as operações de registo, início de sessão, término de sessão, etc...

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.

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*.

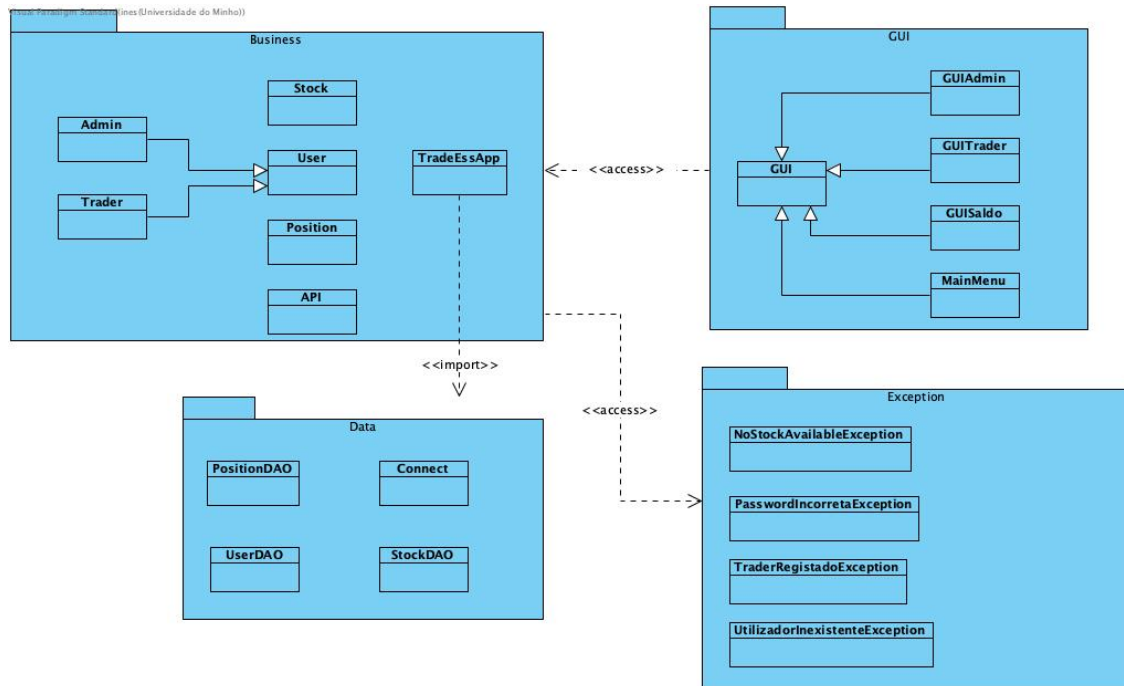


Figura 8: Diagrama de Packages

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, visíveis para outros fora dele. Os *packages* podem também apresentar dependências entre si, caso a alteração 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 Iniciar Sessão

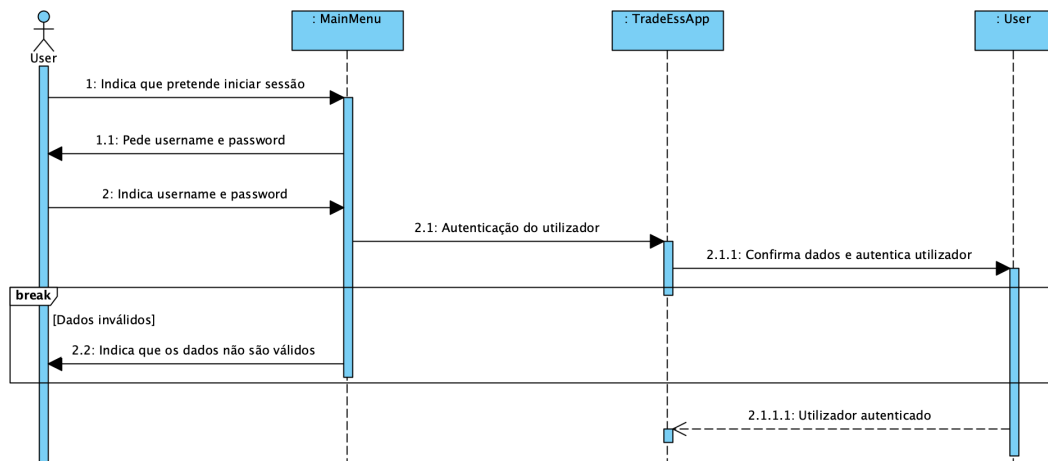


Figura 9: Diagrama de Sequência de Subsistema: Iniciar Sessão

8.2 Registar Utilizador

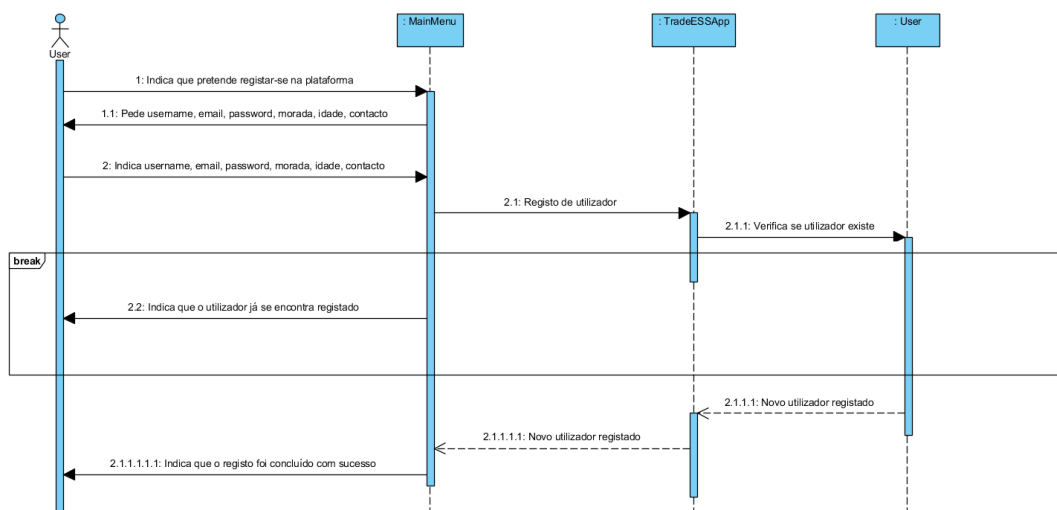


Figura 10: Diagrama de Sequência de Subsistema: Registar Utilizador

8.3 Abrir posição de compra

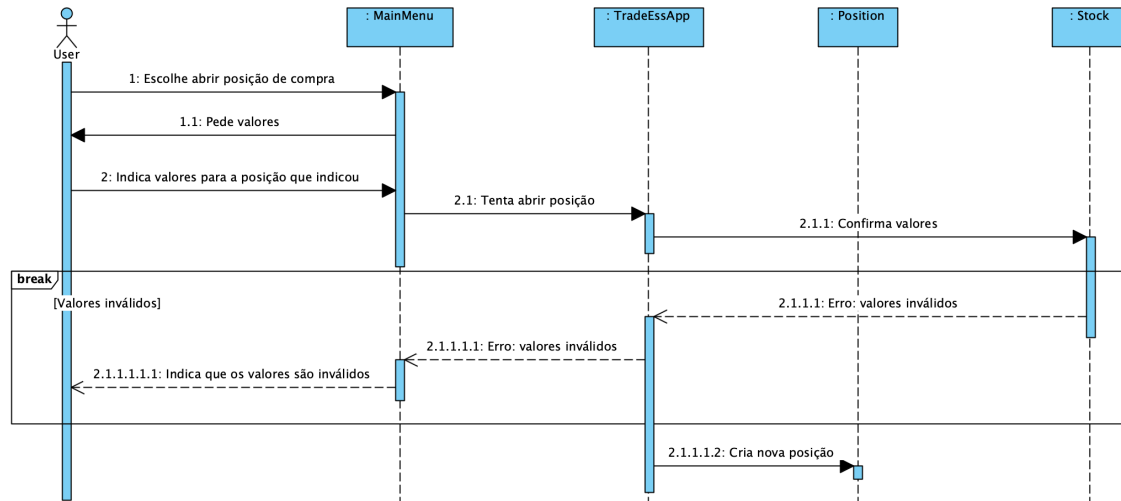


Figura 11: Diagrama de Sequência de Subsistema: Buy Position

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.

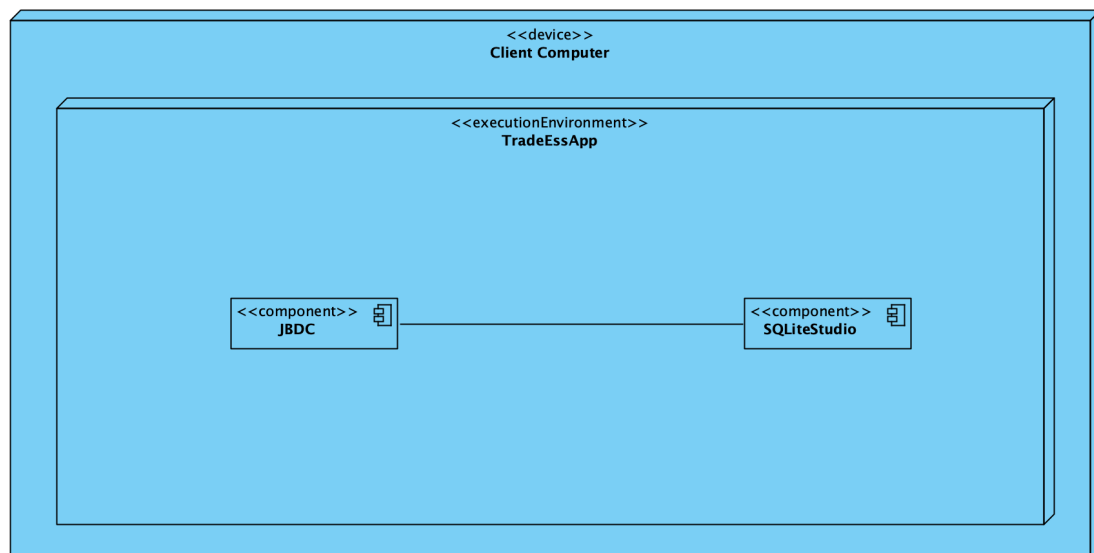


Figura 12: Diagrama de Instalação

10 Interface

10.1 Menu Principal

Quando iniciamos a nossa plataforma podemos encontrar o seguinte menu:

```
----- Trade ESS App -----
1 - Efetuar Registo
2 - Iniciar Sessão
3 - Consultar Mercado
0 - Sair
-----
```

Figura 13: Menu Principal da plataforma

De seguida, podemos efetuar o registo de um utilizador:

```
---- Efetuar Registo ----
Insira o seu email:
a81368@alunos.uminho.pt
Insira o seu username:
inês
Insira a sua password:
password
Insira a sua morada:
Rua da Universidade
Insira a sua idade:
20
Idade válida!
Insira o seu contacto telefónico:
910123456
```

Figura 14: Menu de Registo de um utilizador

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

```
Deseja iniciar sessão? (Sim | Não)
Sim
Sessão iniciada com sucesso.
---- Bem-vindo, inês! ----
Nome: inês
Saldo:0.0
-----
1 - Consultar e gerir saldo
2 - Abrir Buy Position
3 - Abrir Sale Position
4 - Consultar Portefólio
5 - Apagar Conta
6 - Terminar sessão
-----
```

Figura 15: Menu de funcionalidades de um utilizador

A partir daqui, podemos usufruir de todas as funcionalidades deste sistema, como é o caso da consulta e gestão do saldo:

```
--- Gerir Saldo ---
1 - Ver saldo
2 - Depositar Dinheiro
0 - Sair
-----
2
Qual a quantia que quer depositar?
10
10.0
```

Figura 16: Menu referente à consulta e gestão de saldo

De notar ainda que, caso iniciemos sessão com as credenciais de administrador, obtemos o seguinte resultado:

```
---- Iniciar Sessão ----
Username:
admin
Password:
admin
---- Bem-vindo, Administrador ----
1 - Adicionar Stocks
2 - Remover Stocks
0 - Terminar sessão
-----
```

Figura 17: Menu de início de sessão de um Administrador

11 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. Apesar de não ser algo necessário nesta fase do trabalho (que só pedia um esqueleto do código) a nossa plataforma já possui algumas funcionalidades que se aproximam mais de uma plataforma real para *trading*. Estabelece uma conexão à base de dados e processa a informação relativa aos vários utilizadores, negócios/contratos e posições.

A fim de termos uma plataforma de *trading* fiável, tentámos fazer uso de uma API (Yahoo Finance), conseguindo, desta forma, dados reais e de confiança. O objetivo seria serem introduzidas posições na base de dados, posições 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...

No entanto, infelizmente, esta implementação não conseguiu ser finalizada, pelo que não foi implementada a parte relativa à utilização da API. Apesar disso, a estratégia de solução para este problema já foi discutida, passando pela implementação de *threads*, atualizando os valores que pretendemos. Estudaremos ainda a viabilidade de realizar um programa Cliente-Servidor, possibilitando o acesso de vários clientes ao servidor e, mais uma vez, recorrendo a *threads*.

Ainda assim, apesar de algumas limitações desta implementação, fazemos um balanço positivo deste trabalho. Contamos ainda, como já referido, numa implementação futura, conseguir atribuir a real dinâmica ao sistema, conseguindo, por isso, uma monitorização real dos valores das posições e o aperfeiçoamento desta solução.