|  |  |
| --- | --- |
|  |  |
|  |  |

**Licenciatura em Engenharia Informática**

Escola Superior de Tecnologia e Gestão

*Instituto Politécnico de Viana do Castelo*

Unidade curricular de

Programação II

Relatório

2021/2022



26244, André Brandão

24888, Diogo Campos

Índice

[Introdução 3](#_Toc93011325)

[Desenvolvimento 4](#_Toc93011326)

[Compreensão do enunciado 4](#_Toc93011327)

[Planeamento 4](#_Toc93011328)

[Verificação e otimização 4](#_Toc93011329)

[Implementação do manual de utilizador 4](#_Toc93011330)

[Compreensão do enunciado 5](#_Toc93011331)

[Organização 5](#_Toc93011332)

[Diagrama de Classes 6](#_Toc93011333)

[Diagrama de Casos de Uso 9](#_Toc93011334)

[Questões de Implementação 11](#_Toc93011335)

[Conclusão 12](#_Toc93011336)

[References 13](#_Toc93011337)

# 

# Introdução

A presente aplicação pretende satisfazer as necessidades da empresa ***Zlovo***. A ***Zlovo*** é uma *start-up* portuguesa do sector tecnológico que oferece um serviço de gestão de pedidos de quaisquer produtos e entrega desses produtos no domicílio do respetivo cliente.

Esta aplicação foi desenvolvida dentro do âmbito da unidade curricular de Programação II. A linguagem utilizada para o desenvolvimento da aplicação foi a linguagem *Java 16.* Na parte da *GUI* (interface gráfica) utilizou-se a libraria do *Java*, *JavaFX*.

Um dos objetivos deste projeto é que permitiu aos alunos aplicar os conhecimentos adquiridos ao longo das aulas da unidade curricular. Desta forma, os alunos conseguiram ter uma perceção prática, isto é, uma aplicação do mundo real onde foi possível através dos conhecimentos adquiridos desenvolver uma aplicação com utilidade real.

O grupo tem ainda, como objetivo deste projeto o desenvolvimento de capacidades de trabalho em equipa. Os alunos tiveram de partilhar soluções e debater problemas, entre si, de forma a chegar à melhor solução e ao algoritmo mais eficiente.

Os requisitos funcionais e não funcionais iram ser descritos ao longo do documento, com a abordagem dos diversos diagramas feitos para a elaboração de uma aplicação útil e eficaz. O presente documento apresenta a abordagem implementada pelo grupo de trabalho e os resultados obtidos com essa implementação.

# Desenvolvimento

Para o desenvolvimento deste projeto, e para uma melhor organização do desenvolvimento do projeto, o grupo reuniu-se para definir uma estratégia que permitisse entregar o projeto cumprindo tudo o que era pedido dentro do prazo definido pela equipa docente. Essa estratégia foi dividida em etapas que são apresentadas posteriormente.

## Compreensão do enunciado

Nesta etapa, o grupo reuniu-se e leu o enunciado, em conjunto, para compreender tudo aquilo que era pedido, esclarecendo as dúvidas surgidas com a equipa docente.

## Planeamento

Nesta etapa, o grupo dividiu o projeto em tarefas e definiu em que altura essas tarefas deviam estar concluídas, nomeadamente:

* **Diagrama de classes e diagrama de casos de uso** 
  + prazo para terminar (04/12/2021)
* **Código *Java*** 
  + prazo para terminar (10/01/2022)
* **Relatório e manual de utilizador**
  + prazo para terminar (15/01/2022)

## Verificação e otimização

Nesta etapa, o grupo verificou se a aplicação cumpria todos os requisitos funcionais e não funcionais propostos no enunciado e se não existia qualquer tipo de *bug* no código. Após a revisão da sua criação, o grupo decidiu otimizar o seu algoritmo implementando diversos métodos como por exemplo, confirmar antes de sair da aplicação se deseja gravar, verificar se os *usernames* não se repetem, implementação de gráfico de barras, aperfeiçoar os diversos ficheiros *fxml* que o *JavaFX* usa no ambiente gráfico, entre outros.

## Implementação do manual de utilizador

Nesta etapa, o grupo decidiu implementar um manual de utilizador onde explica passo-a-passo as funcionalidades do programa.

É importante notar que após o desenvolvimento desta aplicação, o grupo optou por utilizar a *IDE* da *JetBrains*, *IntellijIDEA* juntamente com *Maven* de recurso para modelar todo o projeto.

Como o grupo perdeu bastante tempo na investigação de como funcionava realmente o *JavaFX*, não teve oportunidade de melhorar todos os cenários possíveis, logo a presença de *bugs*, especialmente através do *input* do utilizador é inevitável.

## Compreensão do enunciado

O grupo antes de começar a resolver o projeto em si, teve de ler várias vezes, atenciosamente, o enunciado apresentado, para evitar qualquer tipo de falha ou de lapso. Após várias leituras o grupo acabou por perceber o que era pretendido fazer e começou por organizar dados, objetivos e pela modelação dos diversos diagramas apresentados na seguinte secção.

## Organização

Como qualquer outro projeto, é essencial que o grupo seja organizado para um melhor desempenho e interpretação da aplicação, quer da parte do utilizador, quer da parte do programador. Por isso, o grupo optou, sob a indicação do docente Miguel Cruz, utilizar o sistema em diversas camadas.

* **UI** – *User Interface* – Responsável pela interação e apresentação dos dados ao usuário.
* **BLL** – *Business Logic Layer* - Responsável por tratar as regras de negócio da aplicação.
* **DAL** – *Data Access Layer* – Responsável por aceder e persistir os dados da aplicação.





# Diagrama de Classes

Figura 3 Diagrama de Classes

A aplicação é composta por vários utilizadores, sendo estes, o dono da empresa, os clientes, os motards e o administrador. Cada utilizador tem como atributos: o nome, número cartão de cidadão, número de identificação fiscal, número de telefone, morada, username, password e uma variável do tipo booleano loginStatus que serve como uma variável de controlo para a aplicação saber qual utilizador está ativo.

Um motard tem ainda como atributo estado, isto é, se se apresenta disponível para realizar uma entrega ou não, este ainda, consegue listar os seus trabalhos, e está relacionado com a ordem de trabalho, ou seja, pode ter ou não um serviço por realizar, porém uma ordem de trabalho terá de corresponder a um motard pelo menos.

A classe *OrdemTrabalho* está relacionada com uma ou mais encomendas, porém uma encomenda pode ou não ter uma *OrdemTrabalho,* esta classe apresenta como atributos uma descrição e um estado da entrega, ou seja, se está já foi realizada ou não.

O Cliente tem ainda um atributo saldo, o mesmo pode efetuar várias encomendas, porém cada encomenda corresponde a um e só um cliente. O cliente pode ainda consultar o seu historial de encomendas.

A encomenda apresenta como atributos, a quantidade, preço, data e o tipo de horário.

Uma encomenda pode ter apenas um tipo de horário, ou seja, só pode ser diurno ou noturno.

Uma encomenda esta relacionada com o pagamento, este tem como atributos o valor, o estado de pagamento, e o tipo de pagamento. Cada pagamento tem um tipo, porém um tipo de pagamento pode estar ligado a vários pagamentos. Os métodos de pagamento são: dinheiro, aplicação, cartão de crédito e multibanco.

Cada encomenda está relacionada com uma informação encomenda, ou seja, esta classe contêm os detalhes de cada encomenda, e apresenta como atributos: o id da encomenda, o username do cliente, o estado da encomenda, o estado da entrega, data do pedido, a data saída e por fim, a data de entrega. O estado da entrega está relacionado com muitas encomendas, contudo, uma encomenda está ligada a um estado de entrega, que pode ser do tipo: a preparar, a caminho ou entregue. O estado da encomenda está relacionado com muitas encomendas, contudo, uma encomenda está ligada a um estado de encomenda, que pode ser do tipo: a aguardar, anulada ou confirmada.

Uma encomenda está ainda relacionada com uma *LinhaEncomenda,* isto representa as várias linhas que constituem uma encomenda, cada encomenda pode ter uma ou várias *LinhaEncomenda*, mas uma *LinhaEncomenda* corresponde apenas a uma

encomenda. Esta classe tem como atributos, o produto, a quantidade, preço unitário e o preço final.

Um Dono de uma empresa pode ter várias empresas, e cada empresa tem um único dono. A classe *Empresa* tem como atributos, o nome, a morada, número telefone e morada.

Tanto a classe *Empresa* como a classe *Utilizador* estão relacionadas com uma classe *Morada,* que representa separação da morada em rua, número de porta e localidade.

Ambas as relações são únicas, ou seja, um utilizador tem uma morada e uma morada tem um utilizador, o mesmo acontece para a relação da empresa com a morada.

A classe *Morada* está ainda ligada a uma classe *Localidade,* que representa a decomposição da localidade em número de código postal e nome da localidade. Uma morada tem uma localidade, mas uma localidade pode ter diversas moradas ou nenhuma.

Cada empresa está ligada a uma ou várias localidades, contudo uma categoria pode não ter uma empresa. Esta classe *Categoria,* tem como atributo o nome.

Tal como, a classe *Empresa*, a classe *Categoria* e aclasse *LinhaEcomenda*, estão relacionadas com um produto. Este tem como atributos o id da empresa, o preço, o nome, a dimensão, o peso, o id próprio e o tipo de unidade. A relação *Categoria-Produto*, caracteriza-se por ser uma relação de 1 para muitos, ou seja, uma categoria pode ter vários produtos ou nenhum, porém um produto tem uma categoria. Tal como a relação *Categoria-Produto,* a relação *Empresa-Produto,* também é uma relação de muitos para muitos, isto é, uma empresa tem vários produtos ou nenhum, contudo um produto tem uma empresa. Já a relação *LinhaEncomenda-Produto*, caracteriza-se por um produto ter várias *LinhaEncomenda* ou nenhuma, contudo uma *LinhaEncomenda* tem um produto.

A classe *Produto,* está ainda relacionada com a classe *TipoUnidade* e tem como subclasse *PackProduto,* a classe *TipoUnidade* apresenta como atributos as unidades de medida dos produtos tais como: Litros, Metros, Kilos e número de Unidades. Assim, um produto tem um *TipoUnidade*, e um *TipoUnidade* tem vários produtos ou não. Por fim, um produto pode pertencer a um conjunto de produtos, caso os produtos comprados pertençam a um conjunto de produtos, o cliente terá um desconto. Um *PackProduto* pode ter muitos *Produto* e muitos *Produto* podem pertencer a vários conjuntos.

# Diagrama de Casos de Uso

Figura 4 - Diagrama Casos de Uso

O diagrama de Casos de Usos é composto por 4 atores, sendo estes, o administrador, o Motard, o Cliente e o Dono da Empresa.

O cliente quando entra pela primeira vez na aplicação tem de criar uma conta, e para tal deve fornecer os seus dados. Quando o mesmo está na sua conta, pode listar as suas encomendas, podendo consultar a encomenda e verificar o estado da mesma. O cliente pode ainda listar as empresas, consultá-las, listar produtos, consultar os mesmos, selecioná-los, deve selecionar a quantidade, deve ainda selecionar a data e o horário, e caso pretende efetuar o pedido deve confirmar, caso contrário, deve cancelar o pedido.

O Dono da Empresa quando entra pela primeira vez na aplicação tem de criar uma conta, e para tal deve fornecer os seus dados. Quando o mesmo está na sua conta, pode listar as encomendas, podendo consultá-las, podendo ainda confirmar ou anular a mesma.  
O Dono da Empresa pode ainda listar as empresas, consultá-las, caso pretenda pode alterar os dados das empresas, pode ainda listar produtos, caso pretenda pode adicionar novos produtos, consegue consultar os mesmos, e ainda tem permissões para alterar dados relativos aos produtos ou eliminá-los.

O Administrador quando entra pela primeira vez na aplicação tem de criar uma conta. Quando o mesmo está na sua conta, pode criar lista de categorias de produtos, consultá-las, editá-las ou removê-las. O Administrador pode ainda, consultar o sistema, ou seja, pode desativar empresas e os respetivos donos. Este pode ainda, cria motards ou administradores. Por fim, pode ainda verificar o estado das encomendas e caso seja necessário pode enviar motards para efetuarem os serviços de entregas.

O Motard quando entra na sua conta, pode listar os trabalhos que ainda tem de realizar, consultá-los, atualizar o seu estado para entregues, pode ainda registar um problema e deve descrever o mesmo.

## Questões de Implementação

Ao longo do desenvolvimento do projeto foram surgindo várias questões quer de lógica quer de implementação. O grupo, numa fase inicial, foi-se questionando de como iria conseguir implementar uma aplicação em ambiente gráfico, visto que seria a primeira vez a o fazer. Contudo, o resultado foi positivo. Ao longo do desenvolvimento da parte do código, surgiram algumas dúvidas na parte da algoritmia em si do *Java*, como por exemplo, percorrer mapas, manipulação de dados através do *input* do utilizador, leitura de ficheiros, entre outros.

Após a planificação do enunciado o grupo obteve por implementar umas opções adicionais, aos requisitos funcionais propostos pela equipa da *Zlovo*. Uma destas sendo a recuperação da password.

Tal como referido anteriormente, o grupo optou por adotar as diferentes camadas, para assim, conseguir tornar a aplicação tanto mais acessível de visualizar como de desenvolver, o que ajudou a minimizar o tempo à procura dos diversos ficheiros do projeto.

É importante referir que o grupo não foi capaz de implementar o requisito não funcional *multi-threading* com o *JavaFX*.

# Conclusão

Durante o desenvolvimento do projeto, o grupo foi-se deparando com algumas dificuldades, sendo estas de interpretação ou de implementação, que conseguiu superar recorrendo a vários recursos.

Por um lado, uma grande fonte de ajuda foram as aulas teóricas da unidade curricular de Engenharia de Software I lecionadas pelo docente Miguel Cruz, onde o mesmo ensinou os diversos modelos UML, que foram bastante úteis para a modelação e planificação do projeto. Após a modelação e planificação do projeto, o grupo passou para o desenvolvimento do código na linguagem Java. Na parte de *back-end,* tanto as aulas TP como PL da unidade curricular de Programação II lecionadas, respetivamente, pelos docentes Miguel Cruz e Ricardo Freitas, foram um ótimo auxílio motivador para conseguir ultrapassar qualquer dificuldade sobre o projeto. Ambos os docentes se mostraram disponíveis para ajudar e explicar como ultrapassar tal impasse, não deixando o grupo desanimar ou ficar desmotivado, muito pelo contrário.

Por outro lado, em relação ao *JavaFX* o grupo apoiou-se na documentação disponibilizada no [Youtube pelo Bro Code](#video), deste modo, o grupo foi capaz de desenvolver uma interface gráfica agradável para o utilizador, como forma de melhorar as suas capacidades de autoaprendizagem. De um modo geral, as opiniões dos elementos do grupo podem traduzir-se numa opinião unânime.

Assim sendo, a realização deste projeto permitiu desenvolver competências de trabalho em equipa que são muito enriquecedoras, pois ao longo da carreira profissional certamente será necessário realizar trabalhos em equipa. De outra forma, poder aplicar os conceitos teóricos de programação numa aplicação de contexto real, permitiu melhorar a perceção da utilidade da programação nos dias de hoje. Por fim, as dificuldades encontradas ao longo do trabalho obrigaram os alunos a procurar soluções ambiciosas e a desenvolver as capacidades de algoritmia, tornado o desenvolvimento deste projeto um desafio que permitiu adquirir competências e assimilar melhor os conteúdos de programação em Java.

Em suma, a apreciação final deste projeto é positiva. O grupo, foi capaz de evoluir não só as suas capacidades de algoritmia, como de visualização e planificação do problema. Com isto, o grupo tenciona melhorar as suas técnicas de trabalho para se tornar mais eficiente e eficaz e acabou por ficar mais motivado para projetos futuros a um alto nível, talvez envolvendo base de dados ao projeto desenvolvido em Java.

# Bibliography

Code, B. (Director). (2021). *JavaFX GUI Course* [Motion Picture].

Cruz, A. M. (2019, Setembro 27). Programação Orientada por Objectos co UML e Java. *Programação Orientada por Objectos co UML e Java.* Instituto Politécnico de Viana do Castelo, Viana do Castelo.