



**Escola Superior
de Tecnologia
e Gestão**

Politécnico de Coimbra

Relatório de Projeto 3

Programação Aplicada | Programação IV

Avaliação [Periódica]

Autor(es):

Jorge Martins a2019100813
Rodrigo Carreira a2019153433

Data: Junho de 2021

Resumo

Este é um relatório de descrição do tipo de soluções efetuadas pelo estudante em relação à construção do seu projeto. Contrariamente a relatórios passados não serão demonstrados todos os métodos, mas sim serão demonstrados os pontos chave do programa indicando a generalidade dos métodos usados, dando como exemplo dos restantes, pois estes são similares.

Este relatório serve como base de compreensão do projeto idealizado e construído pelo estudante, assim guiando quem o consulte para uma perceção ágil das decisões tomadas e estruturas do trabalho.

Palavras-chave

Programação Aplicada, Projeto 3, Projeto Programação Aplicada, Bases de Dados Relacionais, Interface Gráfica.

Índice

Resumo.....	v
Lista de Figuras	xi
Lista de Tabelas.....	xiii
Lista de Acrónimos.....	xv
1. Introdução.....	1
2. Objetivos e Metodologias.....	3
2.1. Ferramentas e Tecnologias	3
2.2. Planeamento	3
3. Trabalho Desenvolvido	5
3.1. Requisitos Implementados.....	5
3.2. Classes e <i>Packages</i>	5
3.3. Algoritmos	11
3.4. Estruturas de Dados	14
3.5. Armazenamento de Dados.....	14
3.6. Procedimentos de Teste.....	24
4. Conclusões	25
4.1. Forças	25
4.2. Limitações	25
4.3. Trabalho Futuro.....	25
5. Referências.....	27
6. Anexos	29

Lista de Figuras

Figura 1 – GitHub	3
Figura 2 - Diagrama Classes da Package AcessoBD	6
Figura 3 - Exemplo UML, Aviso, Pedido, Notificacao, DadosStatic	6
Figura 4 - Package Produtos	8
Figura 5 - UML Utilizadores	9
Figura 6 - Parte do UML de InterFace.....	10
Figura 7 - Escuta de Evento	11
Figura 8 - Listagem Condicionada Utilizadores	12
Figura 9 – “Listador” Master de Utilizadores	12
Figura 10 - Conversor de ArrayList para String Bidimensional.....	13
Figura 11 - Exemplo de Listagem.....	13
Figura 12 - Criação de JFrame.....	14
Figura 13 - Dados Estáticos de Conexão.....	15
Figura 14 - Método Conecta	15
Figura 15 - Método desliga	15
Figura 16 - Método Insert Dados.....	16
Figura 17 - Método Update	17
Figura 18 - Lista Dados.....	18
Figura 19 - Pesquisa Especifica	18
Figura 20 - Verifica existência.....	19
Figura 21 - Notificação Geral Gestores.....	19
Figura 22 - Diagrama Conceptual	20
Figura 23 - Diagrama Físico.....	21
Figura 24 - Abertura de Ficheiro.....	22
Figura 25 - Retirar Dados do Ficheiro	22
Figura 26 - Escrita “Propertie”	23
Figura 27 - Menu “Properties”	23
Figura 28 - Listar "Properties"	23
Figura 29 - Testes Software	24

Lista de Tabelas

Tabela 1 - Requisitos Projeto 1	5
Tabela 2 - Requisitos Projeto 3	5

Lista de Acrónimos

ER	Modelo Entidade-Relacionamento
SoA	State Of the Art
PA	Programação Aplicada
BD	Base de Dados
ID	Identificador/Identidade

1. Introdução

Foi designado o desenvolvimento de um software que tenha a responsabilidade de adicionar utilizadores ao qual o acesso é através de um processo de validação de credenciais. Este software tem como principal objetivo o manuseamento de dados de utilizadores (gestores, clientes, funcionários) e o manuseamento de dados relativos a produtos. Por conseguinte as encomendas é o principal processo para o qual o programa está organizado, sendo o seu objetivo principal através dos seus dados base gerir encomendas e todos os seus estados envolvidos.

Este projeto diferencia-se pela nova temática de uso de Interfaces Gráficas em programação orientada a eventos.

No presente relatório a abordagem será demonstrar as classes dos objetos manipulados pelo programa, seguidamente serão demonstradas as opções gráficas de interação com o utilizador para a manipulação desses objetos e por fim demonstrar os métodos usados para a ligação com a base de dados.

Alargando a descrição do projeto não será apenas descrito o código e suas explicações, mas também esquemas de relação de classes, estruturas da base de dados, “script” de criação de base de dados.

2. Objetivos e Metodologias

2.1. Ferramentas e Tecnologias

Para este projeto foi utilizado o software PowerDesigner para o desenho da Base de Dados do software a desenvolver. Para a construção da base de dados relacional foi utilizado o programa HeidiSql, que foi “host” da base de dados mysql do software.

Relativamente ao desenvolvimento de código Java para o projeto foi utilizado o software Eclipse e IntelliJ, pois a primeira parte do desenvolvimento do mesmo foi realizada em Eclipse sendo que a parte gráfica foi realizada em IntelliJ.

Uma novidade usada como ferramenta para este projeto além do novo IDE foi também o uso do GitHub para sincronização do código dos estudantes.

2.2. Planeamento

Como já referido este projeto iniciou-se com um projeto anteriormente realizado e moldou-se o mesmo para que este tenha as componentes gráficas pois anteriormente este era um projeto com interação em linha de comandos. Para uma melhor organização e trabalho os alunos decidiram a utilização da plataforma GitHub que sincronizada com o IDE IntelliJ os alunos puderam programar e assim ter acesso ao mesmo repositório de código.

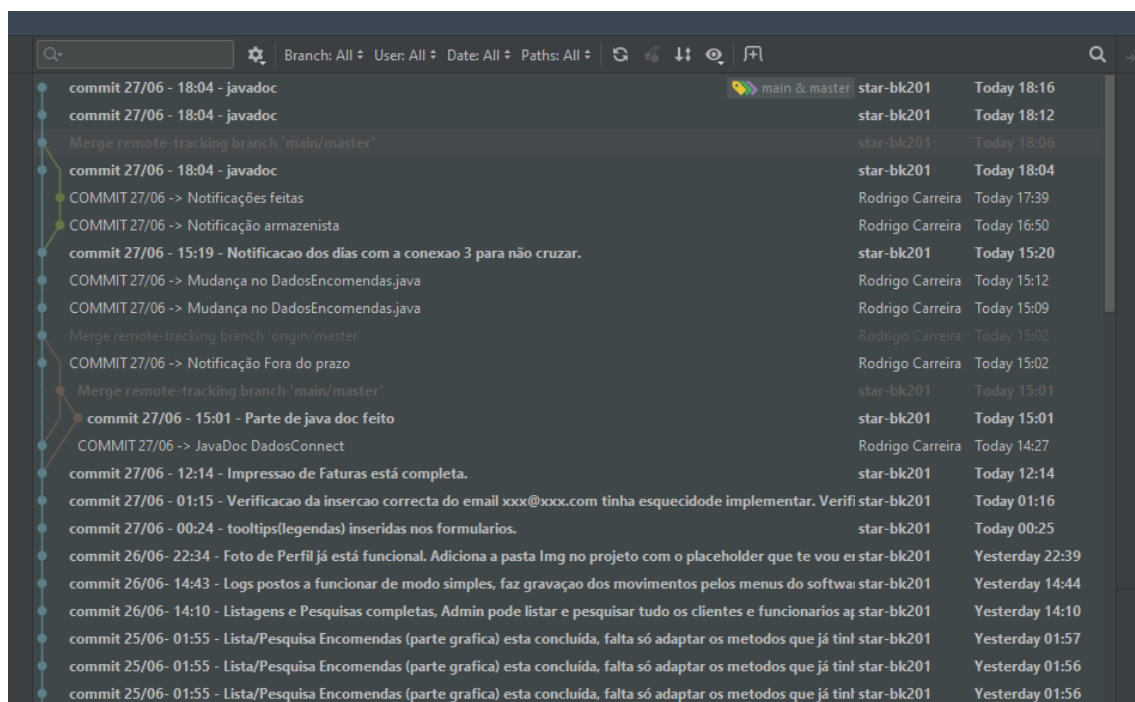


Figura 1 – GitHub

O método de trabalho utilizado foi através de tickets de trabalho pela qual cada aluno se responsabilizava por uma funcionalidade gráfica a implementar e após sua implementação realizava um “commit” no repositório para que o outro aluno tivesse acesso à mesma. A comunicação de trabalho entre alunos além de presencial nas aulas foi também através da plataforma “Discord” para que estes comunicassem os tickets que tinham se responsabilizado.

3. Trabalho Desenvolvido

3.1. Requisitos Implementados

Tabela 1 - Requisitos Projeto 1

Tabela Requisitos – Projeto 1							
Requisito	Estado	Requisito	Estado	Requisito	Estado	Requisito	Estado
R1	Completo	R21	Completo	R41	Completo	R61	Completo
R2	Completo	R22	Completo	R42	Completo	R62	Completo
R3	Completo	R23	Completo	R43	Completo	R63	Completo
R4	Completo	R24	Completo	R44	Completo	R64	Completo
R5	Completo	R25	Completo	R45	Completo	R65	Completo
R6	Completo	R26	Completo	R46	Completo	R66	Completo
R7	Completo	R27	Completo	R47	Completo	R67	Completo
R8	Completo	R28	Completo	R48	Completo	R68	Completo
R9	Completo	R29	Completo	R49	Completo	R69	Completo
R10	Completo	R30	Completo	R50	Completo	R70	Completo
R11	Completo	R31	Completo	R51	Completo	R71	Completo
R12	Completo	R32	Completo	R52	Completo	R72	Completo
R13	Completo	R33	Completo	R53	Completo	R73	Completo
R14	Completo	R34	Completo	R54	Completo	R74	Completo
R15	Completo	R35	Completo	R55	Completo	R75	Completo
R16	Completo	R36	Completo	R56	Completo	R76	Completo
R17	Completo	R37	Completo	R57	Completo	R77	Completo
R18	Completo	R38	Completo	R58	Completo	R78	Completo
R19	Completo	R39	Completo	R59	Completo	R79	Completo
R20	Completo	R40	Completo	R60	Completo	R80	Completo

Tabela 2 - Requisitos Projeto 3

Tabela Requisitos – Projeto 3							
Requisito	Estado	Requisito	Estado	Requisito	Estado	Requisito	Estado
R1	Completo	R21	Completo	R4	Completo	R7	Completo
R2	Completo	R22	Completo	R5	Completo	R8	Completo
R3	Completo	R23	Completo	R6	Completo	R9	Completo

3.2. Classes e Packages

Para uma melhor organização e legibilidade do código este foi dividido por packages agrupando as classes aos seus objetivos.

Neste ponto de um modo geral são demonstrados os packages e as suas classes ao qual serão usados diagramas de classe. Dividiu-se por package cada representação de diagrama para que seja mais compreensível a organização do código e assim de uma forma ágil se compreender a estrutura do projeto.



Figura 2 - Diagrama Classes da Package AcessoBD

Esta package contém os métodos responsáveis de acesso à base de dados relacional e o método de leitura e escrita do ficheiro “properties”.

Para uma organização cada classe está através do seu nome a direccionar o seu objetivo na ligação à base de dados relacional.

A classe “DadosConnect” é uma classe criada com variáveis estáticas e métodos estáticos de ligação à base de dados para que se reduza um pouco em código as ligações do software com a base de dados. Já explicando um pouco as variáveis como do tipo “preparedStatement” ou “resultSet” são assim reutilizadas várias vezes. Será descrito esta decisão mais à frente no corrente relatório.

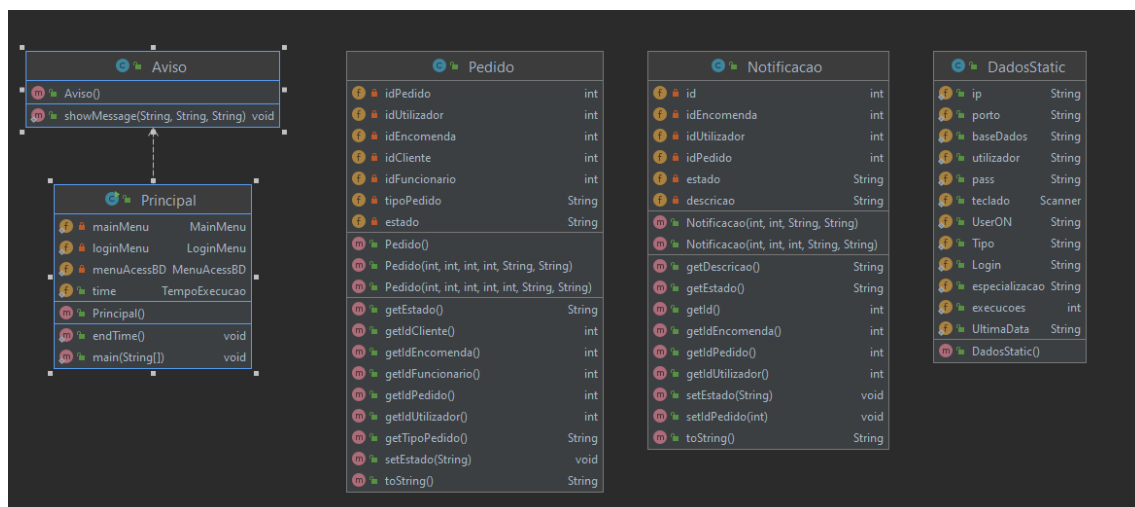


Figura 3 - Exemplo UML, Aviso, Pedido, Notificação, DadosStatic

Devido a estas packages serem mais pequenas em relação a classes, para a sua representação de diagrama de classes foi decidido que fossem em conjunto. Estas foram separadas por packages para que não só o programador criador do código consiga com facilidade ter uma legibilidade do seu próprio código, mas também um elemento exterior ao projeto não encontre desconforto na perceção do que foi programado. A classe “DadosStatic” é uma classe que contem algumas variáveis estáticas que serão usadas para informações importantes no decorrer do software, tais como as informações de ligação e pedidos à base de dados (exemplo lp, password, user, base de dados), mas também algumas variáveis que indicam informações acerca do utilizador que se encontra online a utilizar o software. A classe Principal como o nome indica é onde se encontra o método “main” do software.

Esta classe é uma tentativa por parte do estudante para começar a ter o seu código limpo e legível.

Na figura 3 pode-se observar a package de Pedidos e Notificações do software, esta como o nome indica contem as classes para criação de objetos relativos às notificações e pedidos.

A classe Pedido foi desenhada e criada para que pudesse ser usada de forma abrangente tendo em si a possibilidade de conectar o ID de um cliente, utilizador, funcionário. Desta forma esta classe futuramente pode ser usada para transformar o código para que o software se comporte de forma mais organizada, pois algumas soluções que foram criadas não acabaram por usar esta classe (ex. Pedido de remoção de conta apenas muda o estado), a classe mesmo não usando todas as suas possibilidades foi deixada a sua estrutura original com o pensamento numa otimização futura ou decisão de mudança de funcionamento. Inicialmente o pensamento também seria que numa fase em que fosse necessário filtrar informação acerca de pedidos realizados por cliente a gestor ou funcionário a gestor, através desta classe poderia ser fornecida essa informação. De se observar nesta classe que existem dois construtores diferentes, isto é devido ao momento em que a mesma é criada pelo software não terá uma iniciação com a variável ID de pedido, pois este número será dado pela base de dados, então o segundo construtor é para a criação de um objeto desta classe quando se pede a informação à base de dados, ficando assim o objeto com o ID que a Base de Dados tem acerca dele. O mesmo sucede com a classe notificação, esta classe foi desenhada para se poder usar numa forma geral para notificar tudo no software. Assim a notificação é

usada para o fornecimento de informação direcionada a um utilizador em específico ou a um grupo de utilizadores (ex. Gestores).

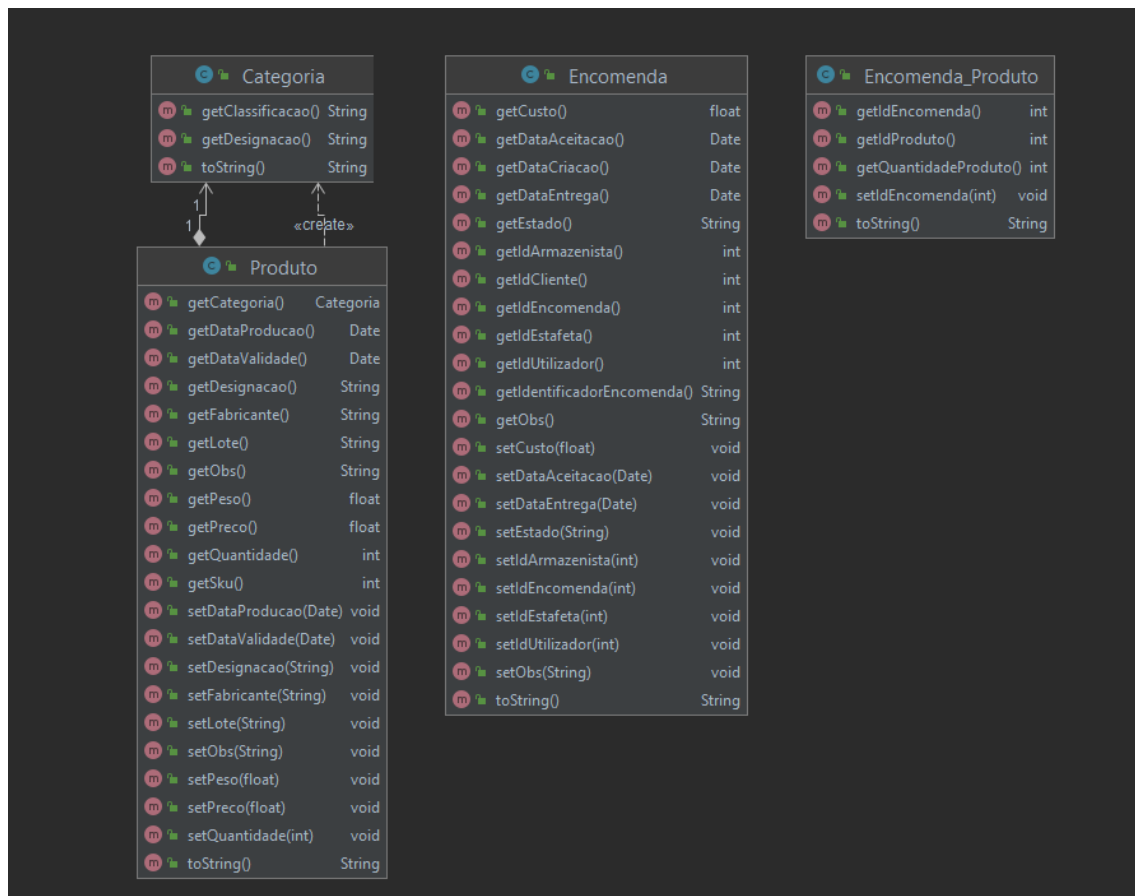


Figura 4 - Package Produtos

A package produtos contém as classes que se envolvem com produtos, por isso as encomendas terem sido também englobadas nesta package assim organizando classes com propósitos próximos sendo mais prático a um elemento externo na leitura do código se guiar onde poderá encontrar a informação de código que procura. A classe produto contém algumas variáveis que não são usadas neste momento pelo software, tais como peso e a data de Validade, estavam inseridos nos requisitos como possibilidades, mas não apareciam como chaves de pesquisa dos produtos, relevando não serem dados direcionados aos produtos que o software vai manusear. Entretanto decidiu-se deixar estes dados para um desenvolvimento posterior ao software (Update do mesmo) assim a classe já conter essas informações, o mesmo sucede na sua tabela na base de dados. Tal como a classe Produto como a classe Encomenda não seguem a mesma lógica de construtores diferentes, primeiramente para a classe produto através da sua variável única “SKU” é possível retirar a informação acerca do seu ID da base de dados quando o utilizador precisar, enquanto que na classe Encomenda esta sofre várias alterações no decorrer do seu processo de vida no software, processos que são isolados entre si, criação, aceitação (insere ID de gestor), delegar a funcionário armazenista (insere seu ID) e seguintes processos. Encontrada esta situação em relação a esta classe em vez de criar vários construtores diferentes, decidiu-se que deveria manter a sua iniciação original mas na iniciação do objeto através das informações da base de dados

se deveria usar os métodos “set” para que se desse a informação completa ao objeto da classe. A classe “encomenda_produto” não contém propriamente uma classe de nome método estático focado só em si devido ao facto desta classe ser o elo entre a quantidade de produtos existente em cada encomenda criada. Esta classe segue a mesma logica que a classe encomenda, pois primeiramente é criada com o ID de produto e a sua quantidade e só após a encomenda é terminada de se criar é que em seguida são geradas as tabelas relativas à informação da quantidade que cada produto constituinte da encomenda tem. Assim o objeto recebe através do método set o ID da encomenda e em seguida é dado à base de dados a informação completa, para iniciar o objeto através da informação da base de dados, primeiramente inicia-se o objeto com o ID de produto e com a quantidade do mesmo e em seguida usa-se o método “set” da classe para que seja dado o ID da encomenda.

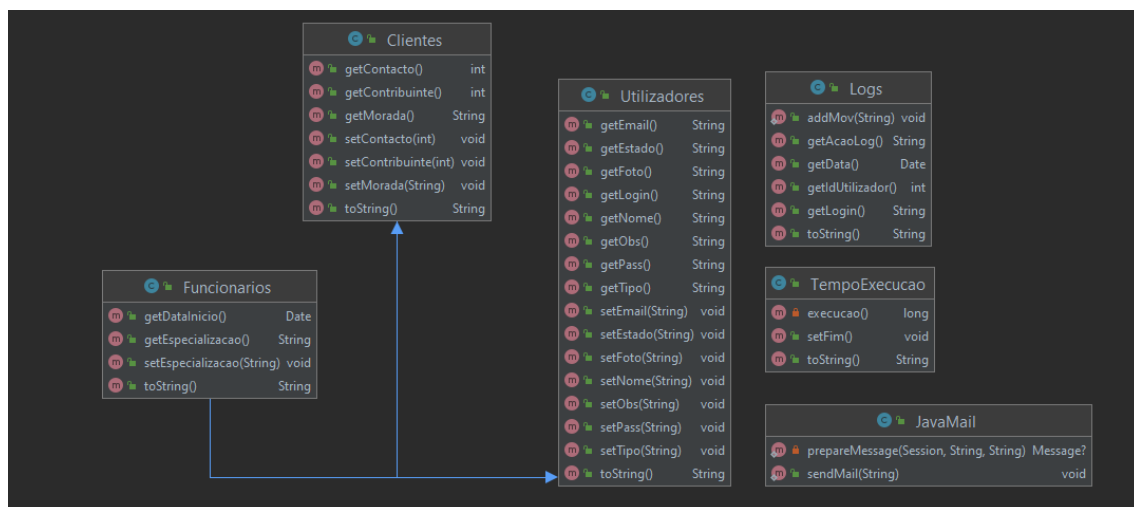


Figura 5 - UML Utilizadores

A package Utilizadores contém as classes responsáveis pelos dados relativos aos utilizadores do software. A classe utilizadora (Pai) é a classe que contém os dados mais comuns e existentes nos três tipos de utilizadores, sendo que a conta de gestor é a que usa o objeto da classe Utilizadores. As suas heranças foram realizadas devido ao facto das outras duas contas conterem algo específico sobre si, diferenciando-se em método de escada, assim Clientes herda de Utilizadores e Funcionários herda de Clientes. A classe “Logs” foi inserida neste package pois esta é responsável por guardar os movimentos do utilizador online na base de dados, identificando onde nos menus o utilizador escolheu as suas opções (ex. entrou no menu Utilizadores), assim visto ser uma classe com direção aos atos do utilizador decidiu-se que seria logico a alocar na package referente aos Utilizadores.

A classe “tempoExecução” é uma classe especialmente criada para o sistema identificar a hora de início e a hora do fim ao encerrar o processo este possa informar o utilizador do tempo decorrido de utilização. A classe Aviso foi criada com a intenção de conter os mais variados avisos que possam existir no software, sendo esta com o seu método dividida em “info” que surge uma janela de contexto de informação. A outra opção é ao

Breve explicação do funcionamento do software

Após a demonstração geral do diagrama de classes do software, será necessário demonstrar alguns exemplos acerca do código em si, elucidando o leitor a compreender melhor o código desenvolvido pelo programador. Assim é de interesse informar que não serão demonstrados todos os métodos realizados, pois existem muitos que são similares no seu funcionamento, são demonstrados os métodos gráficos de interação com o utilizador (criação, listagem, verificação) e os métodos utilizados para ligação na base de dados (insert, update, select).

O software funciona de maneira similar entre os seus métodos, o que difere são as direções para onde os métodos estão a ser usados, por exemplo o mesmo método de listagem é reutilizado para a pesquisa pois a forma de pedido à base de dados apenas difere na condição após o “from” na “query” de “sql” mas os objetos iniciados com as informações do “result set” são os mesmos.

3.3. Algoritmos

Para este software os algoritmos usados são muitos semelhantes pelo qual aqui apenas serão demonstrados exemplos dos mesmos.

```
public void actionPerformed(ActionEvent e) {
    String op = e.getActionCommand();
    switch (op){
        case "inserir": if(stockField.getText().equals("") || skuField.getText().equals("")){
            Aviso.showMessage( "Campo Vazio", "Aviso", "error");
        }else{
            if(DadosProdutos.pesquisaProduto(Integer.parseInt(skuField.getText())).getSku() != Integer.parseInt(skuField.getText())){
                Aviso.showMessage( "Sku nao existente", "Aviso", "error");
            }else {
                Produto product = DadosProdutos.pesquisaProduto(Integer.parseInt(skuField.getText()));
                int qtd = product.getQuantidade();
                product.setQuantidade(Integer.parseInt(stockField.getText()) + qtd);
                DadosProdutos.updateProduto(product);
                cleanerFields();
                Aviso.showMessage( "Stock actualizado com sucesso!", "Aviso", "error");
            }
        }
    }
}
```

Figura 7 - Escuta de Evento

Todas as classes gráficas implementam este método pois estas são criadas numa projecção orientada a eventos pelo qual foram adaptados os métodos estáticos do projeto 1 para este método. Este executa os eventos criados pelo utilizador.

De uma forma a manter a ideologia da utilização de um método que liste na generalidade foi adaptado o método do projeto 1 de nome “listador” master.

```
public static ArrayList <Utilizadores> listarTodosUtilizadores() {...}

/** @param aCondicao ...*/
public static ArrayList <Utilizadores> listarUtilizadoresCondicao(String aCondicao){
    ArrayList <Utilizadores> lista = new ArrayList <> ();
    DadosConnect.conecta();
    try {

        StringBuffer sqlQuery = new StringBuffer();

        // prepared statement for select
        sqlQuery.append(" SELECT * FROM utilizadores ");
        sqlQuery.append(" WHERE "+aCondicao+" ; ");

        DadosConnect.ps = DadosConnect.conn.prepareStatement(sqlQuery.toString());
        DadosConnect.ps.clearParameters();

        DadosConnect.rs = DadosConnect.ps.executeQuery();

        if (DadosConnect.rs == null) {
            System.out.println("!! No Record on table !!");
        } else {
            lista = new ArrayList <Utilizadores> (); //garantir que a lista fica vazia..
            while (DadosConnect.rs.next()) {
                lista.add(new Utilizadores(DadosConnect.rs.getString( columnLabel: "NOME_UTILIZADOR"), DadosConnect
            }
        }
    }
}
```

Figura 8 - Listagem Condicionada Utilizadores

Este método de listagem é tanto utilizado para listagem de todos os utilizadores como para uma pesquisa avançada dos mesmos. A ideia é através do fornecimento dos dados da base de dados construir um “ArrayList” de objetos e em seguida os enviar. O próximo método irá manipular esse “ArrayList” de forma a que os dados sejam demonstrados ao utilizador.

```
public class MenuListMaster extends JPanel{

    private JPanel panelTop, userPanelTable, filterPanel;
    private JTable userTable;
    private JScrollPane scroll;

    /**
     * Construtor da classe Listar os utilizadores através de um arraylist, contem os elementos para
     * @param listUsers arraylist de utilizadores
     */
    public MenuListMaster(ArrayList <Utilizadores> listUsers) {
        String [][] userList = stringConverterUser(listUsers);
        setLayout(new BorderLayout());

        panelTop = new JPanel(new FlowLayout(FlowLayout.CENTER));
        panelTop.add(new JLabel( text: "Lista de Utilizadores"));

        userPanelTable = new JPanel(new FlowLayout(FlowLayout.CENTER));
    }
}
```

Figura 9 – “Listador” Master de Utilizadores

Este método já sendo o método gráfico, substituindo o anterior método estático de texto, recebe no seu construtor o “ArrayList” oriundo do método demonstrado anteriormente e em seguida irá o transformar em um “array” de “Strings”

bidimensionais para que seja possível usar no método “JTable” para demonstrar os dados em forma de tabela. De referenciar que esta classe estende “JPanel” em vez de “JFrame” para que a mesma possa ser chamada dentro de outro frame com a sua listagem e assim incorporar a tabela em uma outra situação.

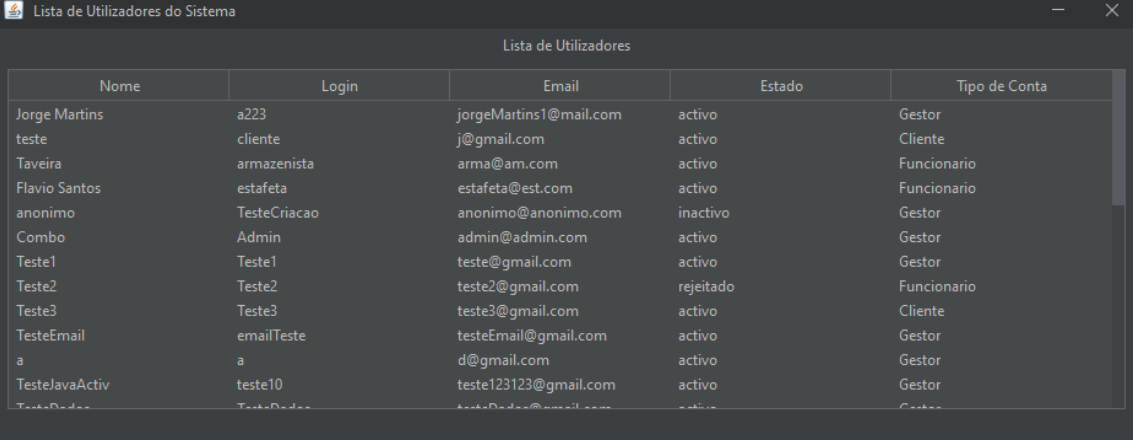
```
private static String[][] stringConverterUser(ArrayList <Utilizadores> listUsers){ //
    String [][] sendList = new String[listUsers.size()][5];
    for (int i = 0; i < listUsers.size(); ++i) {
        sendList[i][0] = " " + listUsers.get(i).getNome();
        sendList[i][1] = " " + listUsers.get(i).getLogin();
        sendList[i][2] = " " + listUsers.get(i).getEmail();
        sendList[i][3] = " " + listUsers.get(i).getEstado();
        sendList[i][4] = " " + listUsers.get(i).getTipo();
    }
    return sendList;
}
```

Figura 10 - Conversor de ArrayList para String Bidimensional

Este método apenas vai inserir em cada coluna o respetivo dado referente a uma linha para que estes fiquem em conformidade.

Após terminado o ciclo que percorre o “ArrayList” a tabela de dados bidimensional é enviada.

Assim estes dados são inseridos dentro de uma “JTable” que será inserida no “JPanel” principal ao qual será demonstrado ao utilizador algo como o exemplo abaixo.



Nome	Login	Email	Estado	Tipo de Conta
Jorge Martins	a223	jorgeMartins1@mail.com	activo	Gestor
teste	cliente	j@gmail.com	activo	Cliente
Taveira	armazenista	arma@am.com	activo	Funcionario
Flavio Santos	estafeta	estafeta@est.com	activo	Funcionario
anonimo	TesteCriacao	anonimo@anonimo.com	inactivo	Gestor
Combo	Admin	admin@admin.com	activo	Gestor
Teste1	Teste1	teste@gmail.com	activo	Gestor
Teste2	Teste2	teste2@gmail.com	rejeitado	Funcionario
Teste3	Teste3	teste3@gmail.com	activo	Cliente
TesteEmail	emailTeste	testeEmail@gmail.com	activo	Gestor
a	a	d@gmail.com	activo	Gestor
TesteJavaActiv	teste10	teste123123@gmail.com	activo	Gestor
TesteDado	TesteDado	testeDado@gmail.com	activo	Gestor

Figura 11 - Exemplo de Listagem

Se por ventura seja necessário chamar o método sozinho e que tenha o seu próprio “frame” não sendo necessário que o “JPanel” construído seja incorporado e outra seção foi feito o método estático da classe para criar um “frame” para a mesma.

```
public static JFrame createFrame(ArrayList<Utilizadores> users){  
    JFrame jframe = new JFrame();  
    MenuListMaster menuListMaster = new MenuListMaster(users);  
  
    jframe.setSize( width: 950, height: 400);  
    jframe.setLocationRelativeTo(null);  
    jframe.setTitle("Lista de Utilizadores do Sistema");  
    jframe.setResizable(false);  
    jframe.setVisible(true);  
  
    jframe.add(menuListMaster);  
    return jframe;  
}
```

Figura 12 - Criação de JFrame

Este método estático recebe um “arraylist” e chama a sua própria classe apenas a incorporando no “JFrame” ali criado.

3.4. Estruturas de Dados

Como já identificado no tema anterior a estrutura mais utilizada para a manipulação de dados é o “ArrayList”, utilizado para a listagem de dados oriundos da base dados, primeiramente são iniciados os objetos e em seguida inseridos no “ArrayList” a ser enviado para o seu método de listagem.

3.5. Armazenamento de Dados

3.5.1. Métodos Java de Acesso à Base de Dados

Os métodos responsáveis pelo armazenamento dos dados como já referenciados encontram-se no package de acesso á base de dados “AcessoBD”. Os métodos de acesso á base de dados são entre si similares, pelo qual para cada classe existe o seu método genérico mudando as condições de seleção de dados para o pedido.

O único caso do software pelo qual foi usada uma inserção de dados com a opção “autoCommit” da base dados em manual, foi nos dados relativos a clientes e funcionários, pois tratam-se de heranças e como tal na inserção dos dados na base de dados é necessária a existência de uma segurança se caso alguma exceção ocorra no software, fazendo assim um “rollback” dos dados. Referenciar que na criação dos dados só após obter todos os dados necessários por parte do utilizador e os validar é que os mesmos são pedidos para ser inseridos na base de dados, reduzindo as chances de erros assegurando um bom funcionamento do software.

```
17  */
18  public class DadosConnect {
19
20      static Connection conn = null;
21      static PreparedStatement ps = null;
22      static ResultSet rs = null;
23  }
```

Figura 13 - Dados Estáticos de Conexão

Para um código mais limpo e com métodos menos extensos, decidi usar variáveis de conexão à base de dados de forma estática, assim esta classe além de conter estas variáveis, também contém os métodos de as ligar e de as fechar.

De referenciar o facto de dentro desta classe existirem três grupos de variáveis estáticas e seus métodos, isto devido ao facto que em alguns pedidos à base de dados, alguns métodos em si mesmos realizam pedidos de validação e assim protege-se de nenhuma das variáveis ser fechada fora do tempo programado de uso.

```
public static void conecta() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        conn = DriverManager.getConnection("jdbc:mysql://" + DadosStatic.ip + DadosStatic.porto + "/" + DadosStatic.baseDados + "?useSSL=");
    } catch (SQLException e) {
        System.out.println("!! SQL Exception !!\n" + e);
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        System.out.println("!! Class Not Found. Unable to load Database Drive !!\n" + e);
    }
}
```

Figura 14 - Método Conecta

Todo o método de acesso à base de dados inicia com o chamar deste método (ou seus similares), este é responsável de iniciar a variável da classe “connection” à base de dados relacional que o software utiliza e aqui é verificado o caminho da mesma. Nesta situação são usadas as variáveis estáticas que retiram os dados oriundos do ficheiro “properties” para que seja realizada com eficácia a ligação à base de dados.

```
/**
 * Metodo 1 que desliga a ligação à base de dados de forma segura
 */
public static void desliga() {
    try {
        if(conn != null)
            conn.close();
        if(ps != null)
            ps.close();
        if(rs != null)
            rs.close();
    } catch (SQLException e) {
        System.out.println("!! SQL Exception !!\n" + e);
        e.printStackTrace();
    }
}
```

Figura 15 - Método desliga

Todo o método de acesso à base de dados também contém o chamamento ao método de desligar ou fechar as variáveis de acesso, mas que assim não existam erros posteriores no decorrer do funcionamento do programa e seus pedidos à base de dados.

```
public static boolean adicionarEncomenda(Encomenda encomenda) {
    DadosConnect.conecta();
    try {
        StringBuffer sqlQuery = new StringBuffer();

        // prepared statement for select
        sqlQuery.append(" INSERT INTO encomenda (CLI_ID_UTILIZADOR, IDENTIFICADOR_ENCOMENDA, CUSTO_ENCOMENDA, DATACRIACAO_ENCOMENDA) "
            + " VALUES(?, ?, ?, ?, ?); ");
        java.sql.Date sqlDate = new java.sql.Date(encomenda.getDataCriacao().getTime());

        DadosConnect.ps = DadosConnect.conn.prepareStatement(sqlQuery.toString());
        DadosConnect.ps.clearParameters();
        DadosConnect.ps.setInt(1, encomenda.getIdCliente());
        DadosConnect.ps.setString(2, encomenda.getIdentificadorEncomenda());
        DadosConnect.ps.setFloat(3, encomenda.getCusto());
        DadosConnect.ps.setDate(4, sqlDate);
        DadosConnect.ps.setString(5, encomenda.getEstado());

        DadosConnect.ps.executeUpdate();

    } catch (SQLException e) {
        System.out.println("!! SQL Exception !!\n"+e);
        e.printStackTrace();
        return false;
    }

    DadosConnect.desliga();
    return true;
}
```

Figura 16 - Método Insert Dados

Aqui está representada a estrutura de um método “insert” utilizado neste software, é usado o “prepared statement” para uma melhor segurança dos dados introduzidos pelo utilizador este método manipula a informação de um objeto que recebe de forma a introduzi-la nos campos corretos relativos à tabela de base de dados.

Cada método de introdução de dados na base de dados relacional segue este modelo aqui representado, diferenciando apenas a tabela que está a ser chamada na “query” tal como os campos que irão ser introduzidos, também difere o objeto que recebe, sendo o da classe representativa no código dos dados da tabela de Base de Dados, mas em suma toda a arquitetura é a mesma, o mesmo pensamento a mesma logica de resolução.


```
public static void updateEncomenda(Encomenda encomenda) {
    DadosConnect.conecta();
    try {
        java.sql.Date sqlDate;
        java.sql.Date sqlDate2;

        StringBuffer sqlQuery = new StringBuffer();
        sqlQuery.append(" UPDATE encomenda ");
        sqlQuery.append(" SET ESTADO_ENCOMENDA = ? ,");
        sqlQuery.append(" ID_UTILIZADOR = ? ,");
        sqlQuery.append(" FUN_ID_UTILIZADOR = ? ,");
        sqlQuery.append(" FUN_ID_UTILIZADOR2 = ? ,");
        sqlQuery.append(" DATAACEITACAO_ENCOMENDA = ? ,");
        sqlQuery.append(" DATAENTREGA_ENCOMENDA = ? ");
        sqlQuery.append(" WHERE IDENTIFICADOR_ENCOMENDA = ? ");

        DadosConnect.ps = DadosConnect.conn.prepareStatement(sqlQuery.toString());

        DadosConnect.ps.clearParameters();
        DadosConnect.ps.setString(1, encomenda.getEstado());
        DadosConnect.ps.setInt(2, encomenda.getIdUtilizador());
        if(encomenda.getIdArmazenista() != 0) {
            DadosConnect.ps.setInt(3, encomenda.getIdArmazenista());
        } else {
            DadosConnect.ps.setNull(3, Types.INTEGER);
        }
        if(encomenda.getIdEstafeta() != 0) {
            DadosConnect.ps.setInt(4, encomenda.getIdEstafeta());
        } else {
            DadosConnect.ps.setNull(4, Types.INTEGER);
        }
    }
}
```

Figura 17 - Método Update

Pegando no exemplo mais completo de “update” de uma tabela de base de dados, primeiramente comentar que para que seja utilizado nas mais diversas ocasiões para as classes de relativas a utilizadores (e suas heranças), produtos e encomenda o “update” é realizado sempre a todos os dados, mesmo estes se mantendo os mesmos, dando assim uma polivalência ao método de que liga á base de dados e a atualiza.

Existem nos casos das classes de Pedido e Notificação que após estudo da funcionalidade das mesmas, mesmo futuramente não faria sentido modificar dados além do seu estado, pois não foi um requisito pedido mas ser interessante implementar uma classe que apagasse os dados relativos a pedidos e notificações de 10 em 10 dias, mas também poderia ser viável para o utilizador ter esses dados futuramente para perceber comportamentos dos seus funcionários (visto os pedidos por exemplo apenas serem usados na fase de passagem de encomenda para um estafeta).

Chegando ao ponto inicial acerca de método, começou por se relatar que é um exemplo completo, pois aqui também se encontra uma situação pelo qual os dados desta tabela estão em constante transformação, pois inicialmente a mesma não contem dados acerca do gestor que a aceitou, o armazenista que a preparou, ou estafeta que a entregou, nem as respetivas datas de aceitação e entrega, então para estes casos ao qual é necessário enviar à base de dados um valor nulo, utilizou-se o “setNull” em seguida indica-se o tipo de valor nulo que se dá para qual tipo “Types.tipo”.

```
public static ArrayList <Encomenda> listarEncomendasCondicao(String aCondicao){
    ArrayList <Encomenda> lista = new ArrayList <Encomenda> ();
    DadosConnect.conecta();
    try {
        StringBuffer sqlQuery = new StringBuffer();

        // prepared statement for select
        sqlQuery.append(" SELECT * FROM encomenda ");
        sqlQuery.append(" "+aCondicao+" ");
        DadosConnect.ps = DadosConnect.conn.prepareStatement(sqlQuery.toString());
        DadosConnect.ps.clearParameters();

        DadosConnect.rs = DadosConnect.ps.executeQuery();

        if (DadosConnect.rs == null) {
            System.out.println("!! No Record on table !!");
        } else {
            lista = new ArrayList <Encomenda> (); //garantir que a lista fica vazia..
            while (DadosConnect.rs.next()) {
                lista.add(new Encomenda(DadosConnect.rs.getString("IDENTIFICADOR_ENCOMENDA"), DadosConnect.rs.getFloat("LO
            }
        }
    }
}
```

Figura 18 - Lista Dados

Este método é um exemplo dos seus similares na obtenção de vários dados da base de dados numa só opção, sabendo neste caso o método é adaptado para as encomendas então o “ArrayList” será para objetos de “Encomenda” e a primeira parte da “query” já automática será obter todos os dados das colunas da tabela encomenda da base de dados. Então é onde a diferenciação das condições para os diferentes tipos de listagem e pesquisa avançada acontecem é na segunda colagem da “String” que insere a condição vinda de um método estático responsável pela opção.

```
@param aLogin
public static Utilizadores pesquisaLogin(String aLogin) {
    DadosConnect.conecta();
    Utilizadores utilizador = new Utilizadores();
    try {
        StringBuffer sqlQuery = new StringBuffer();

        // prepared statement for select
        sqlQuery.append(" SELECT * FROM utilizadores ");
        sqlQuery.append(" WHERE LOGIN_UTILIZADOR = ? ;");

        DadosConnect.ps = DadosConnect.conn.prepareStatement(sqlQuery.toString());
        DadosConnect.ps.clearParameters();
        DadosConnect.ps.setString(1, aLogin);

        DadosConnect.rs = DadosConnect.ps.executeQuery();

        if (DadosConnect.rs == null) {
            System.out.println("!! No Record on table !!");
        } else {
            while (DadosConnect.rs.next()) {
                utilizador = new Utilizadores(DadosConnect.rs.getString("NOME_UTILIZADOR"), DadosConnect.rs.getString("LO
            }
        }
    }
}
```

Figura 19 - Pesquisa Especifica

Para a obtenção de um dado específico como se pode observar as mudanças não são radicais apenas é feito algum ajuste, tal como a “query” já está preparada apenas necessita do valor (preparada também para valores inseridos pelo utilizador), assim este pedido devolve apenas um objeto da classe utilizador, mas existe o mesmo método similar em outras classes, para que a manipulação de dados específicos seja feita em objeto pelo software e a base de dados seja apenas responsável por atender a pedidos de atualização e inserção.

```
public static String verificaLogin(String login) {  
    DadosConnect.conecta2();  
    String envio = "";  
  
    try {  
  
        StringBuffer sqlQuery = new StringBuffer();  
  
        // prepared statement for select  
        sqlQuery.append(" SELECT ID_UTILIZADOR FROM utilizadores ");  
        sqlQuery.append(" WHERE LOGIN_UTILIZADOR = ? ");  
  
        DadosConnect.ps2 = DadosConnect.conn2.prepareStatement(sqlQuery.toString());  
        DadosConnect.ps2.clearParameters();  
        DadosConnect.ps2.setString(1, login);  
  
        DadosConnect.rs2 = DadosConnect.ps2.executeQuery();  
  
        if (DadosConnect.rs2 == null) {  
            System.out.println("!! No Record on table !!");  
        } else  
            while (DadosConnect.rs2.next()) {  
                envio = DadosConnect.rs2.getString("ID_UTILIZADOR");  
            }  
  
    } catch (SQLException e) {  
        System.out.println("!! SQLException !!");  
    }  
}
```

Figura 20 - Verifica existência

Sendo mais pratico verificar a existência através do que a base de dados devolve para um inteiro ou uma “String” em vez do objeto (alguns problemas com nullPointer), estes métodos são utilizados várias vezes para verificar existência de dados, tanto para validações como até contagens. Sendo que por exemplo no caso de inteiros a verificar se o contribuinte existe sabe-se que caso ele exista na base de dados ela devolve o contribuinte caso não exista ao devolver uma tabela vazia este é considerado 0, assim o número 0 representa a sua inexistência na base de dados.

```
public static int EncomendasForaDeTempo() {  
    int conta = 0;  
  
    StringBuffer sqlQuery = new StringBuffer();  
  
    DadosConnect.conecta();  
  
    sqlQuery.append(" SELECT COUNT(*) AS \"Contador\" FROM encomenda ");  
    sqlQuery.append(" WHERE ID_UTILIZADOR = '"+DadosUtilizadores.verificaLogin(DadosStatic.Login)+"' " +  
        + " AND ABS(DATEDIFF(DATAACEITACAO_ENCOMENDA, DATAENTREGA_ENCOMENDA)) > 10 ; ");  
  
    try {  
        DadosConnect.ps = DadosConnect.conn.prepareStatement(sqlQuery.toString());  
        DadosConnect.ps.clearParameters();  
  
        DadosConnect.ps = DadosConnect.conn.prepareStatement(sqlQuery.toString());  
        DadosConnect.ps.execute();  
        DadosConnect.rs = DadosConnect.ps.executeQuery();  
        while (DadosConnect.rs.next()) {  
            conta = DadosConnect.rs.getInt("Contador");  
        }  
    } catch (SQLException e) {  
        System.out.println("!! SQLException !!");  
    }  
}
```

Figura 21 - Notificação Geral Gestores

Ao contrario das notificações específicas que contam com uma “query” ao qual a condição identifica na tabela de notificações alguma ainda ativa que esteja com um ID direcionado para a conta que ficou online, aí é demonstrado ao utilizador a notificação e a mesma se torna inativa (mudança do seu estado).

Este exemplo de notificação geral ela tem tendência a persistir também devido à sua importância mas também porque é uma notificação de responsabilidade de uma classe (gestores), pelo qual é utilizado um contador na “query” e verifica se alguma encomenda se encontra nas condições impostas, caso não existam a tabela devolve vazio que como

inteiro é considerado 0, sendo que o seu método estático apenas verifica se o que é devolvido é maior que 0 para ser verdade e notificar a existência de alguma situação com aquelas condições.

Concluindo esta parte do tema, obvio que não existem apenas estes métodos, mas todos os métodos utilizados são bastante similares aos aqui exibidos apenas com alguns ajustes direcionados á sua classe e sua utilidade, penso que seja menos massivo para o leitor a compreensão através deste relatório sobre o funcionamento interno do software e seguidamente ao ler o código com facilidade entender as estruturas usadas.

3.5.2. Diagrama Conceptual

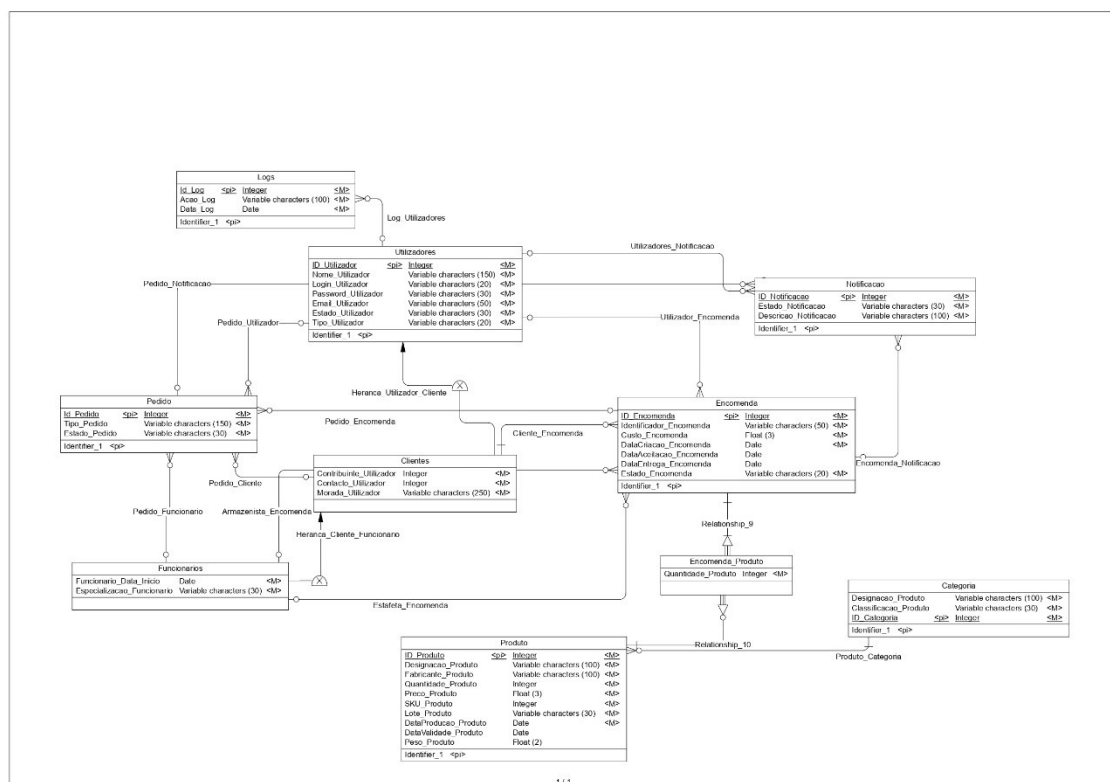


Figura 22 - Diagrama Conceptual

Em anexo é enviado com este relatório o diagrama conceptual do mesmo em formato pdf para uma melhor compreensão do mesmo.

Inicialmente ainda foi pensado em usar em todas as tabelas “n para n” mas devido ao facto depois em código ser mais complexo ter de estar a gerar e manipular várias tabelas, quando logicamente em certos requisitos apenas é necessário uma ligação, foi decidido criar uma boa base para suportar o software e não criar obstáculos ao programar.

A ligação para a tabela encomenda com funcionários poderia ser de “n para n” mas devido ao facto de apenas serem dois funcionários envolvidos criou-se duas ligações distintas, identificando os responsáveis (armazenista e estafeta) e assim apenas foi

necessário a manipulação dos seus ID como chave forasteira, sendo pratico e eficaz no problema encontrado.

Em alguns casos deixei também prioritária a existência de um dado de uma outra tabela (ex Produto com categoria) pois já no código será algo obrigatório então pensei que poderia deixar também na base de dados. Deste modo seguindo apenas a logica que seria aplicada no desenvolvimento do código do software.

Utilizei também que algumas variáveis fossem obrigatórias na inserção de uma linha numa tabela, mas optei por não usar na criação a unicidade das mesmas, deixando essa segurança para o código, pois caso tentasse através da base de dados apenas resultaria numa exceção “truncada” e o código iria parar.

3.5.3. Diagrama Físico

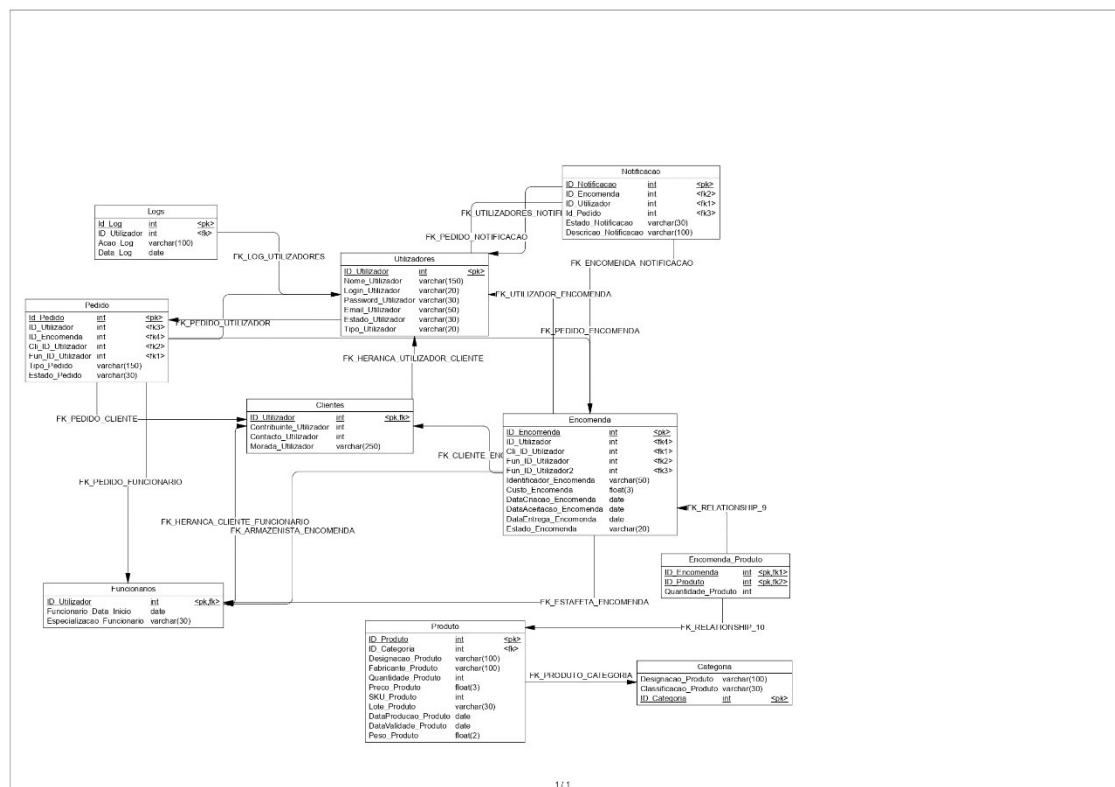


Figura 23 - Diagrama Físico

Observando o diagrama físico é possível perceber os resultados referidos em relação às chaves forasteiras existentes em cada tabela.

Este diagrama também se encontra em anexo para uma melhor legibilidade.

Para não ocupar espaço de forma desnecessária o script de criação da base de dados será enviado em anexo, este foi gerado automaticamente pelo software “power designer”, apenas adicionei o comando de “AUTO_INCREMENT” às chaves primarias para que estas sejam de incrementação automática.

3.5.4. Ficheiro “Properties”

A informação acerca dos dados necessários para que a ligação com a base de dados seja realizada de forma correta, encontra-se num ficheiro “Property”. Para este método utiliza-se um método de leitura e escrita de dados como de um documento de texto.

A leitura dos dados é realizada ao iniciar o software, mas na primeira camada do programa existe a possibilidade de o utilizador mudar a ligação à base de dados para uma outra que ele deseje.

```
public static boolean leituraFicheiroProp(String aCaminho) {  
  
    if (aCaminho != null && aCaminho.length() > 0) {  
        ficheiroLeitura = new File(aCaminho);  
        if (ficheiroLeitura.exists()) {  
  
            try {  
                fis = new FileInputStream(ficheiroLeitura);  
                return true;  
            } catch (IOException ioe) {  
                ioe.printStackTrace();  
            }  
        }  
    }  
    return false;  
}
```

Figura 24 - Abertura de Ficheiro

```
public static void leituraDadosProp() {  
    leituraFicheiroProp("Properties/dadosAcesso.properties");  
  
    Properties properties = new Properties();  
    try {  
        properties.load(fis);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
  
    DadosStatic.ip = properties.getProperty("ip");  
    DadosStatic.porto = properties.getProperty("porto");  
    DadosStatic.baseDados = properties.getProperty("baseDados");  
    DadosStatic.utilizador = properties.getProperty("utilizador");  
    DadosStatic.pass = properties.getProperty("pass");  
    try {  
        fis.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Figura 25 - Retirar Dados do Ficheiro

Através da figura 26 e 27 pode-se observar o processo de abertura do ficheiro e sua leitura, retirando os valores que são guardados como que num “hash-map” ao qual cada chave devolve um valor.

Como o ficheiro “properties” está agregado à pasta do projeto então apenas foi dado o caminho relativo da mesma ao qual o explorador inicia dentro da pasta projeto.

```
public static void escritaDadosProp(String aIp, String aPorto, String aBaseDados, String aUtilizador, String aPass) {  
  
    OutputStream os = new FileOutputStream("/Properties/dadosAcesso.properties");  
  
    Properties properties = new Properties();  
  
    properties.setProperty("ip", aIp);  
    properties.setProperty("porto", aPorto);  
    properties.setProperty("baseDados", aBaseDados);  
    properties.setProperty("utilizador", aUtilizador);  
    properties.setProperty("pass", aPass);  
  
    properties.store(os, null);  
  
    os.close();  
}
```

Figura 26 - Escrita “Propertie”

Para escrita dos dados é relativamente parecido o processo ao qual abrimos a ligação ao ficheiro (abrir o ficheiro) e em seguida são remetidos os valores para as chaves, neste processo que o ficheiro é reescrito é como se estivesse a inserir novas chaves e novos valores. A logica usada para o “update” em base de dados é usada aqui, qualquer gravação de dados, realiza uma atualização a todos os dados.

```
* @param teclado[]  
private static void modificaPropriedades(Scanner teclado){  
    FicheiroProperties.LeituraDadosProp();  
    int opcao = 0;  
    do{  
        ListaProp();  
        opcao = Principal.pedeDadosInteiro("Escolha opcao a modificar: \n1-Base Dados \n2-Ip \n3-Password \n4-Porto \n5-Utilizador \n6-Gravar \n7-Sair ");  
        switch(opcao){  
            case 1: DadosStatic.baseDados = Principal.pedeDadosString("\nInsira nova base de dados: ", teclado); Aviso.operacaoSucesso(); break;  
            case 2: DadosStatic.ip = Principal.pedeDadosString("\n Insira o novo Ip: ", teclado); Aviso.operacaoSucesso(); break;  
            case 3: DadosStatic.pass = Principal.pedeDadosString("\n Insira nova password da Base de Dados: ", teclado); Aviso.operacaoSucesso(); break;  
            case 4: DadosStatic.porto = Principal.pedeDadosString("\n Insira novo porto de ligação: ", teclado); Aviso.operacaoSucesso(); break;  
            case 5: DadosStatic.utilizador = Principal.pedeDadosString("\n Insira novo utilizador da base de dados: ", teclado); Aviso.operacaoSucesso(); break;  
            case 6: try {  
                FicheiroProperties.escritaDadosProp(DadosStatic.ip, DadosStatic.porto, DadosStatic.baseDados, DadosStatic.utilizador, DadosStatic.pass);  
            } catch (Exception e) {  
                e.printStackTrace();  
                Aviso.operacaoInsucesso();  
                break;  
            }  
            case 7: break;  
            default: Aviso.avisoErro();  
        }  
    } while(opcao != 7);  
}
```

Figura 27 - Menu “Properties”

Para que se um utilizador quiser apenas modificar um valor não estar a obrigar a modificar todos os valores, criou-se um menu que cada opção remete a uma ação de mudança a um dos valores de ligação à base de dados sendo que a sexta opção serve de gravação dos dados. Para tornar mais apelativo e informativo ao utilizador listei através do método “listaProp()” os dados atuais para que o utilizador tenha noção quais as palavras chave usadas na ligação à base de dados.

```
private static void listaProp() {  
    System.out.println("Base de dados : "+DadosStatic.baseDados+"\n" + "Ip: " +DadosStatic.ip +"\n"+"Password: " + DadosStatic.pass);  
}
```

Figura 28 - Listar "Properties"

Ainda foi pensado em inserir palavra passe de acesso a esta opção, mas devido à natureza do projeto não solicitar o mesmo e este se debruçar de um produto de exercício académico, esta opção acabou por ser abandonada.

3.6. Procedimentos de Teste

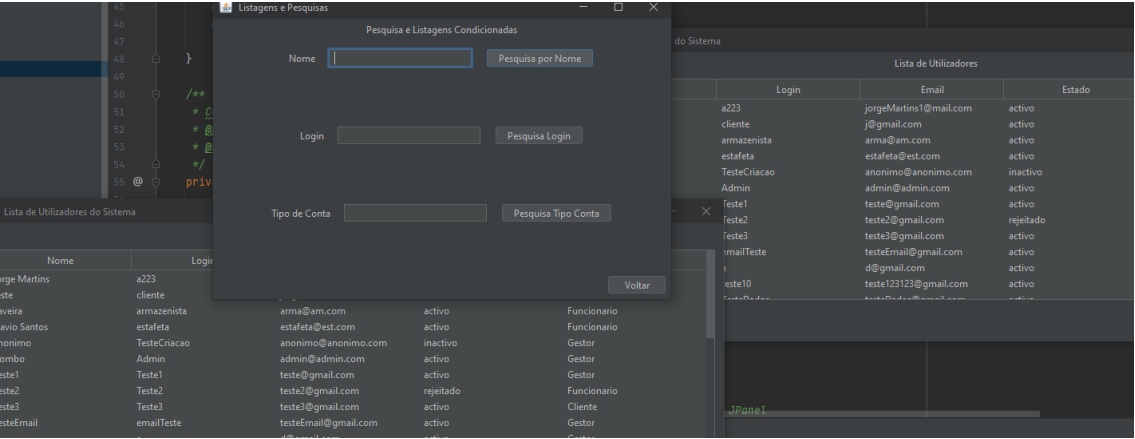


Figura 29 - Testes Software

Para testar o nosso software realizamos uma bateria de testes de utilização do programa de forma a tentar perceber se existem alguns erros do mesmo, por exemplo falhas de verificações dos dados de utilizador, acesso indevido a opções do programa.

4. Conclusões

Este foi um projeto que nos fez desenvolver bastante enquanto programadores, pois esta foi a primeira vez que realizamos um projeto de uma aplicação nativa de uma forma gráfica através da linguagem Java. Tivemos oportunidade de consolidar os nossos conhecimentos algo que se nota no desenvolvimento do código que vai ficando cada vez mais eficiente.

Contrariamente a relatórios anteriores decidimos não criar um relatório com demasiada informação, pois documentação densa tende a algumas vezes confundir o leitor na perceção do funcionamento e construção do programa, assim foi entendido que os pontos principais acerca do funcionamento do mesmo foram abordados, demonstrando as logicas adotadas.

4.1. Forças

Este é um programa funcional e eficaz para os requisitos desejados, através de uma ligação de base de dados relacional manipula dados relativos a utilizadores, produto e encomendas, gerando notificações aos utilizadores de ações a prevenir ou de atuação imediata.

Tanto para o utilizador com uma interface gráfica simples e tentando manter alguma logica de aproximação das ações a realizar por cada menu, a construção do software ao se manter na mesma logica de funcionamento nos seus variados métodos torna o software simples de se compreender.

4.2. Limitações

Este trabalho ou projeto tem algumas limitações e uma encontra-se na sua eficiência, sei que poderia ter alguns dos métodos mais simplificados de forma a reutilizar em mais situações.

Uma limitação que encontramos foi a falta de tempo, mesmo tendo conseguido realizar o projeto antes do tempo exigido, devido há existência de outros trabalhos para outras disciplinas não foi possível dedicar mais tempo na observação de bugs / erros no software, tal como na simplificação do código de forma a este ser o mais eficiente possível.

4.3. Trabalho Futuro

Para uma melhoria no projeto seria dedicação de mais tempo na resolução de erros que possam surgir de situações atípicas do software que algumas vezes apenas sucedem após várias variáveis estarem reunidas.

Algo que penso que seria de valorizar seria uma adição de alguns métodos, como a existência de um “super” gestor que tivesse apenas acesso aos ficheiros “properties”.

5. Referências

- Anio. (15 de 06 de 2014). *HeidiSql*. Obtido de HeidiSql: <https://www.heidisql.com/forum.php?t=15612>
- Anonimo. (2020 de 03 de 02). *Convertworld*. Obtido de Convertworld: <https://www.convertworld.com/pt/tempo/horas.html>
- Anonimo. (11 de 11 de 2011). *Javapoint*. Obtido de Javapoint: <https://www.javatpoint.com/java-string-to-date>
- Anonimo. (14 de 08 de 2019). *tutorialspoint*. Obtido de tutorialspoint: <https://www.tutorialspoint.com/jdbc/commit-rollback.htm>
- Bernal, G. (28 de 12 de 2014). *StackOverflow*. Obtido de StackOverflow: <https://pt.stackoverflow.com/questions/1386/express%C3%A3o-regular-para-valida%C3%A7%C3%A3o-de-e-mail>
- Bernal, G. (12 de 04 de 2017). *StackOverflow*. Obtido de StackOverflow: <https://pt.stackoverflow.com/questions/1386/express%C3%A3o-regular-para-valida%C3%A7%C3%A3o-de-e-mail>
- Chintala, U. (27 de 06 de 2011). *DZone*. Obtido de DZone: <https://dzone.com/articles/techtip-use-setlenient-method>
- Developer. (15 de 09 de 2017). *Youtube*. Obtido de Youtube: <https://www.youtube.com/watch?v=TAJoVkaVv-8>
- ForLearners. (14 de 09 de 2018). *Youtube*. Obtido de Youtube: <https://www.youtube.com/watch?v=EYIJXP-ZmD0>
- Java, C. H. (04 de 02 de 2014). *Youtube*. Obtido de Youtube: <https://www.youtube.com/watch?v=FEI2W1Mhkf0>
- Matheus. (10 de 6 de 2010). *GUJ*. Obtido de GUJ: <https://www.guj.com.br/t/como-utilizar-a-funcao-parse-date/72210>
- Mentor, S. T. (13 de 09 de 2020). *Youtube*. Obtido de Youtube: <https://www.youtube.com/watch?app=desktop&v=Vp4bl36GFCo>
- Mick. (23 de 09 de 2015). *coredump*. Obtido de coredump: <https://pt.coredump.biz/questions/32739719/what-happens-on-connectionsetautocommit-false>
- Microsoft. (9 de 12 de 2017). *Docs.Microsoft*. Obtido de Microsoft: <https://docs.microsoft.com/en-us/sql/t-sql/language-elements/rollback-transaction-transact-sql?view=sql-server-ver15>

- Oracle. (01 de 01 de 2020). *Apache*. Obtido de Apache: <https://commons.apache.org/proper/commons-validator/apidocs/org/apache/commons/validator/routines/DateValidator.html>
- Pirilon. (10 de 10 de 2008). *guj*. Obtido de guj: <https://www.guj.com.br/t/transformar-a-data-em-um-inteiro/48781>
- Plankton, S. (01 de 06 de 2004). *coderanch*. Obtido de coderanch: <https://coderanch.com/t/395538/java/import-SimpleDateFormat>
- Rosa, G. (26 de 09 de 2019). *Alura*. Obtido de Alura: <https://cursos.alura.com.br/forum/topico-statement-fica-em-aberto-92878>
- Sakinala, K. (31 de 10 de 2019). *Youtube*. Obtido de Youtube: <https://www.youtube.com/watch?app=desktop&v=OBnjTszNe3c>
- Singh, A. (31 de 12 de 2018). *GeeksforGeeks*. Obtido de GeeksforGeeks: <https://www.geeksforgeeks.org/localdate-parse-method-in-java-with-examples/>
- Singh, A. (31 de 12 de 2018). *GeeksforGeeks*. Obtido de GeeksforGeeks: <https://www.geeksforgeeks.org/localdate-parse-method-in-java-with-examples/>
- Telusko. (15 de 09 de 2016). *Youtube*. Obtido de Youtube: <https://www.youtube.com/watch?app=desktop&v=w7D5YB2U2jU>
- Wilk, A. (28 de 11 de 2012). *StackOverflow*. Obtido de StackOverflow: <https://stackoverflow.com/questions/13605248/java-converting-image-to-bufferedimage/13605411>

6. Anexos

1. Ficheiros Java e Extras
 - Ficheiros da pasta src com suas packages e ficheiros .java
 - Pasta lib
 - Pasta com ficheiro Properties com credenciais da base de dados
 - Diagramas de Classes Gerados pela Eclipse e IntelliJ Organizados por Package
 - Pasta Img com foto de Placeholder
2. Javadoc
3. Projecto3_BaseDados_Scripts_Diagramas
 - BD_Proj3.sql (Script de criação de Base de Dados)
 - Diagrama Conceptual em Pdf
 - Diagrama Físico em Pdf
 - Diagramas Físico e Conceptual do software PowerDesing
4. Projecto3.jar (Executavel .jar)
5. Diagramas de Classes em Imagens
6. Manual de Utilizador