# RELATÓRIO- SPRINT 2

Estruturas de Informação 2022/2023

G064
Gonçalo Ribeiro 1210792
Luís Monteiro 1211250
Miguel Marques 1201078
Pedro Mesquita 1211171
Ruben Vilela 1211861

Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

# Índice

# Introdução

No âmbito de LAPR3, da Licenciatura em Engenharia Informática, foi-nos requisitado para fazer a análise, design e implementação de uma solução informática que apoie a gestão de uma empresa responsável pela gestão de uma instalação agrícola em modo de produção biológico (MPB). O desenvolvimento deste conjunto de funcionalidades que permitem gerir a informação relativamente a uma rede de distribuição de cabazes entre agricultores e clientes.

Neste sentido, implementamos uma série de funcionalidades, ao longo dos dois sprints, capazes de solucionar este ambiente.

Toda a sua resolução foi implementada na Linguagem *Java*.

# US307: Importar a lista de cabazes.

Nesta User Storie, o objetivo era ler os dados de um ficheiro csv com a informação relativa aos cabazes que iriam ser expedidos.

Como solução, acreditamos que a solução mais benéfica, jogando também com as próximas User Stories, seria de criar um Map<Integer, Map<Destinatario, float[]>> no método ReadCabaz.

```
public void ReadCabaz(File file, String separatorRegex, Map<Integer,Map<Destinatário,List<float []>>> cabazesMap) throws FileNotFoundException {
    Destinatário destinatário=null;
    Scanner reader = new Scanner(file);
    String header = reader.nextLine();
    String dest;
    float produtos[]=new float[20];
    int c=0,d,dia;
    Map<Destinatário, List<float[]>> mapa;
    for (d=0;d<20;d++){
        produtos[d]=0;
    }
    while (reader.hasNextLine()) {
        String[] split = reader.nextLine().split(separatorRegex);
        try {
            for (String s:split) {
                cleanString(s);
                //System.out.printf(s+" ");
            }
            for (d=2;d<split.length;d++){
                produtos[d-2]=Float.parseFloat(split[d]);
                //System.out.println(split[d]+"--"+d);
            }
            dest=null;
            if (split[0].charAt(0)==(char)34){
                dest=split[0].substring(1,split[0].length()-1);
            }
            else{
                dest=split[0];
            }
            dia=Integer.parseInt(split[1]);
            mapa=cabazesMap.get(dia);
            if (mapa==null){
                cabazesMap.put(dia,new HashMap<>());
                cabazesMap.get(dia).put(findDestinatário(dest),new ArrayList<>());
                cabazesMap.get(dia).get(findDestinatário(dest)).add(produtos.clone());
            }
            else{
                else{
                    mapa.put(findDestinatário(dest),new ArrayList<>());
                    mapa.get(findDestinatário(dest)).add(produtos.clone());
                }
            }
            catch (NumberFormatException e) {}
    }
    return;
}
```

Neste sentido, toda a informação foi armazenada neste Map, cujo cada Key Integer correspondia ao dia, e cada dia tem um outro Map em que o seu Key onde são os Destinatarios, isto é, os clientes e produtores. Depois cada destinatário terá a sua lista de produtos encomendados, para o caso dos clientes, ou produtos expedidos, no caso dos produtores.

# US308: Gerar uma lista de expedição para um determinado dia que forneça os cabazes sem qualquer restrição quanto aos produtores

Para implementarmos esta funcionalidade, criamos dois métodos: gerarListaClientesEProdutores e gerarLista.

No primeiro método, a partir do ficheiro lido da US anterior, organizamos os clientes e os produtores em duas diferentes listas, List<CabazExpedicao>, sendo CabazExpedicao uma classe criada para armazenar os dados de cada Cliente-Produtor.

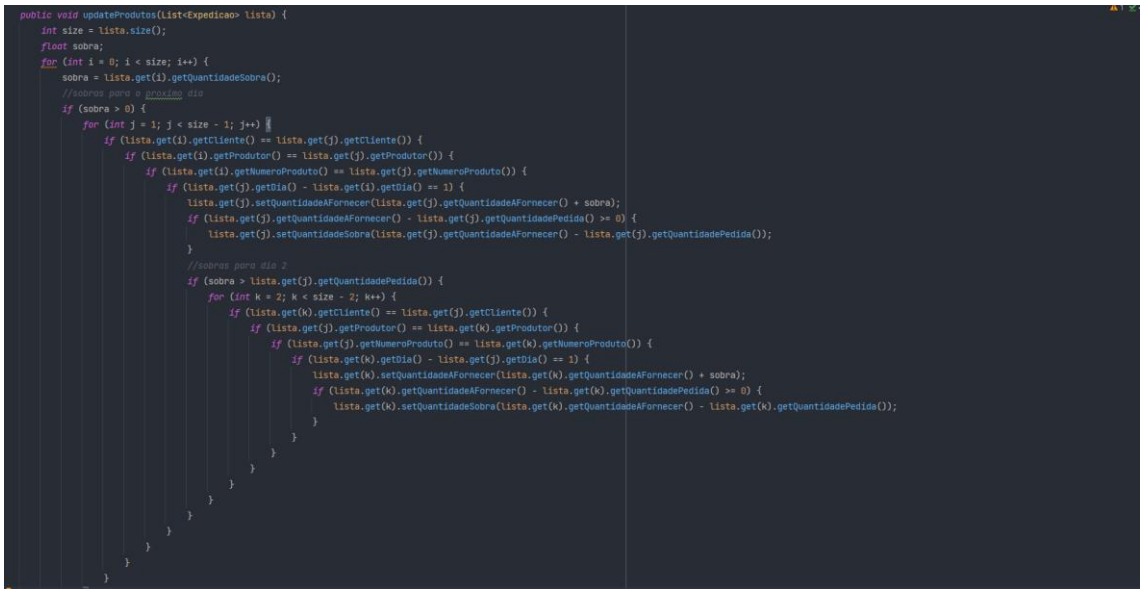
```
public void gerarListaClientesEProdutores(Map<Integer, Map<Destinatário, List<float[]>>> cabazesMap, List<CabazExpedicao> produtores, List<CabazExpedicao> clientes) throws Exception {

    if (cabazesMap.isEmpty()) {
        throw new Exception("MAP IS NULL");
    }

    for (Map.Entry<Integer, Map<Destinatário, List<float[]>>> entry : cabazesMap.entrySet()) {

        for (Map.Entry<Destinatário, List<float[]>> entry2 : entry.getValue().entrySet()) {
            char produtor = entry2.getKey().getName().charAt(0);
            CabazExpedicao expedicao = new CabazExpedicao(entry2.getKey(), entry2.getValue(), entry.getKey());

            if (produtor == 'P') {
                produtores.add(expedicao);
            } else {
                clientes.add(expedicao);
            }
        }

    }
}
```

Já no segundo método, fizemos uma comparação entre os clientes e produtores, na qual verificamos para cada produto, se havia um Produtor P capaz de entregar a quantidade de produtos que um determinado cliente pedia. Se este caso verificasse, eram guardadas as informações numa List<Expedicao>, onde Expedicao continha dados do Cliente, do Produtor, Local do hub, a quantidade pedida pelo cliente e a quantidade fornecida pelo produtor, assim como, a quantidade que sobrava, para um determinado dia.

```
public List<Expedicao> gerarLista(List<CabazExpedicao> clientes, List<CabazExpedicao> produtores, Graph<Local, Integer> map) {
    List<Expedicao> lista = new ArrayList<>();
    List<Expedicao> listaFinal = new ArrayList<>();

    for (CabazExpedicao c : clientes) {
        List<ParClienteHub> delivery_hub = US304.findNearestHubs(map, numberOfHubs: 1);
        for (CabazExpedicao p : produtores) {

            if (p.getDia() == c.getDia()) {

                List<float[]> produtosCliente = c.getProdutos();
                List<float[]> produtosProdutor = p.getProdutos();

                for (int i = 0; i < produtosCliente.size(); i++) {
                    float[] arrC = produtosCliente.get(i);
                    float[] arrP = produtosProdutor.get(i);

                    for (int j = 0; j < arrC.length; j++) {

                        if (arrC[j] <= arrP[j] && arrC[j] != 0) {
                            lista.add(new Expedicao(c.getDestinatario(), p.getDestinatario(), arrC[j], arrP[j], quantidadeSobra: arrP[j] - arrC[j], numeroProduto: j + 1, c.getDia(),delivery_hub.get(i).getEmpresa()));
                        }
                    }
                }
            }
        }
    }

    updateProdutos(lista);
```

```
    updateProdutos(lista);

    for (Expedicao expedicao : lista) {
        if (expedicao.getQuantidadeSobra() >= 0) {
            listaFinal.add(expedicao);
        }
    }
    return listaFinal;
}
```

Para o tratamento da quantidade de produtos que sobravam e em união com o critério de que "Se a totalidade de um produto disponibilizado por um agricultor para venda, num determinado dia, não for totalmente expedido, fica disponível nos dois dias seguintes, sendo eliminado após esses dois dias" criamos o método updateProdutos.



```
public void updateProdutos(List<Expedicao> lista) {
    int size = lista.size();
    float sobra;
    for (int i = 0; i < size; i++) {
        sobra = lista.get(i).getQuantidadeSobra();
        //sobras para o proximo dia
        if (sobra > 0) {
            for (int j = i; j < size - 1; j++) {
                if (lista.get(i).getCliente() == lista.get(j).getCliente()) {
                    if (lista.get(i).getProdutor() == lista.get(j).getProdutor()) {
                        if (lista.get(i).getNumeroProduto() == lista.get(j).getNumeroProduto()) {
                            if (lista.get(j).getDia() - lista.get(i).getDia() == 1) {
                                lista.get(j).setQuantidadeAFornecer(lista.get(j).getQuantidadeAFornecer() + sobra);
                                if (lista.get(j).getQuantidadeAFornecer() - lista.get(j).getQuantidadePedida() >= 0) {
                                    lista.get(j).setQuantidadeSobra(lista.get(j).getQuantidadeAFornecer() - lista.get(j).getQuantidadePedida());
                                }
                                //sobras para dia 2
                                if (sobra > lista.get(j).getQuantidadePedida()) {
                                    for (int k = 2; k < size - 2; k++) {
                                        if (lista.get(k).getCliente() == lista.get(j).getCliente()) {
                                            if (lista.get(j).getProdutor() == lista.get(k).getProdutor()) {
                                                if (lista.get(j).getNumeroProduto() == lista.get(k).getNumeroProduto()) {
                                                    if (lista.get(k).getDia() - lista.get(j).getDia() == 1) {
                                                        lista.get(k).setQuantidadeAFornecer(lista.get(k).getQuantidadeAFornecer() + sobra);
                                                        if (lista.get(k).getQuantidadeAFornecer() - lista.get(k).getQuantidadePedida() >= 0) {
                                                            lista.get(k).setQuantidadeSobra(lista.get(k).getQuantidadeAFornecer() - lista.get(k).getQuantidadePedida());
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
```

Neste método, iniciamos por armazenar o valor de sobra de um determinado produto. Depois, numa cadeia fomos comparando se dois dados do tipo Expedicao continham as mesmas informações relevantes para o caso: Cliente, Produtor, numeroProduto e se estavam em dias seguidos. Em seguida, adicionamos o valor armazenado das sobras à quantidade para fornecer no dia seguinte. O mesmo fizemos para os dois dias a seguir à data original, no entanto caso ainda sobrasse produtos de dois anteriores, estes já eram desprezados.

## US309: Gerar uma lista de expedição para um determinado dia que forneça apenas com os N produtores agrícolas mais próximos do hub de entrega do cliente.

Tendo esta User Story, juntamente com a US308, seguir o seguinte critério:

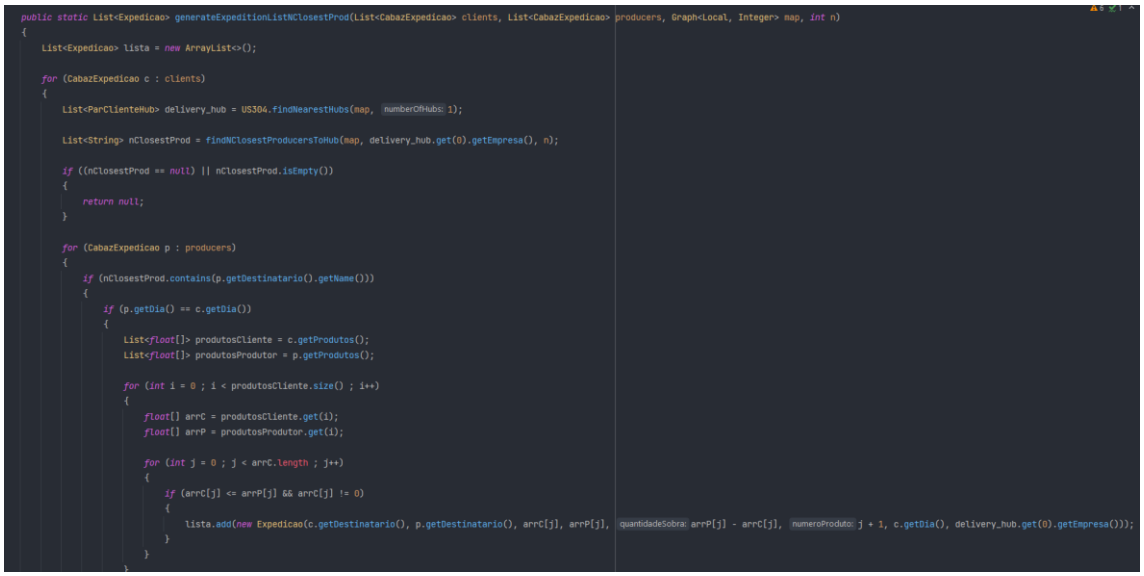
“Critério de Aceitação: A lista de expedição deve indicar para cada cliente/cabaz, os produtos, quantidade encomendada/expedida e o produtor que forneceu o produto.”

A solução que encontramos foi por criar os métodos FindNClosestProducersToHub  e generateExpeditionListNClosestProd.

```
private static List<String> findNClosestProducersToHub(Graph<Local, Integer> map, Local hub, int n)
{
    if (n <= 0)
    {
        return null;
    }

    ArrayList<Local> prods = new ArrayList<>();

    for (Local l : map.vertices())
    {
        //get all companies
        if (l.getDestinatário().charAt(0) == 'P')
        {
            prods.add(l);
        }
    }

    if (prods.isEmpty())
    {
        return null;
    }

    Map<Local, Integer> hub_distance = new HashMap<>();

    for (Local p : prods)
    {
        LinkedList<Local> shortPath = new LinkedList<>();
        Integer dist = Algorithms.shortestPath(map, hub, p, Integer::compare, Integer::sum, zero: 0, shortPath);

        if (dist != null)
        {
            hub_distance.put(p, dist);
        }
    }

    List<Map.Entry<Local, Integer>> hubLst = new LinkedList<>(hub_distance.entrySet());
```

```
    List<Map.Entry<Local, Integer>> hubLst = new LinkedList<>(hub_distance.entrySet());

    hubLst.sort(Map.Entry.comparingByValue());

    List<String> closestHubs = new ArrayList<>();

    if (hubLst.size() < n)
    {
        n = hubLst.size();
    }

    for (Map.Entry<Local, Integer> entry : hubLst.subList(0, n))
    {
        closestHubs.add(entry.getKey().getDestinatário());
    }

    return closestHubs;
}
```

Neste primeiro método, o nosso propósito foi de primeiramente de verificar se o Destinatario tratava-se de um produtor. Se fosse o caso, era adicionado a uma lista, onde a partir dessa lista verificamos os hubs mais proximos , que seriam depois todos armazenado num Map, com Key do Local e a sua distancia que, por fim, seria traduzida para a List<String> dos hubs mais próximos.



```
public static List<Expedicao> generateExpeditionListNClosestProd(List<CabazExpedicao> clients, List<CabazExpedicao> producers, Graph<Local, Integer> map, int n)
{
    List<Expedicao> lista = new ArrayList<>();

    for (CabazExpedicao c : clients)
    {
        List<ParClienteHub> delivery_hub = US304.findNearestHubs(map,  numberOfHubs: 1);

        List<String> nClosestProd = findNClosestProducersToHub(map, delivery_hub.get(0).getEmpresa(), n);

        if ((nClosestProd == null) || nClosestProd.isEmpty())
        {
            return null;
        }

        for (CabazExpedicao p : producers)
        {
            if (nClosestProd.contains(p.getDestinatario().getName()))
            {
                if (p.getDia() == c.getDia())
                {
                    List<float[]> produtosCliente = c.getProdutos();
                    List<float[]> produtosProdutor = p.getProdutos();

                    for (int i = 0 ; i < produtosCliente.size() ; i++)
                    {
                        float[] arrC = produtosCliente.get(i);
                        float[] arrP = produtosProdutor.get(i);

                        for (int j = 0 ; j < arrC.length ; j++)
                        {
                            if (arrC[j] <= arrP[j] && arrC[j] != 0)
                            {
                                lista.add(new Expedicao(c.getDestinatario(), p.getDestinatario(), arrC[j], arrP[j],  quantidadeSobra: arrP[j] - arrC[j],  numeroProduto: j + 1, c.getDia(), delivery_hub.get(0).getEmpresa()));
                            }
                        }
                    }
```

```
                    for (int j = 0 ; j < arrC.length ; j++)
                    {
                        if (arrC[j] <= arrP[j] && arrC[j] != 0)
                        {
                            lista.add(new Expedicao(c.getDestinatario(), p.getDestinatario(), arrC[j], arrP[j],  quantidadeSobra: arrP[j] - arrC[j],  numeroProduto: j + 1, c.getDia(), delivery_hub.get(0).getEmpresa()));
                        }
                    }
                }
            }
        }
    }
    return lista;
}
```

No segundio método, e como está indicado nas imagens acima, verificamos ao longo da lista de clientes e de produtores, os seus hubs mais próximos entre eles, código esse que já tinha sido implementada na US304. Depois, com os hubs mais proximos, guardamos a informacao necessaria numa Lista<Expedicao>

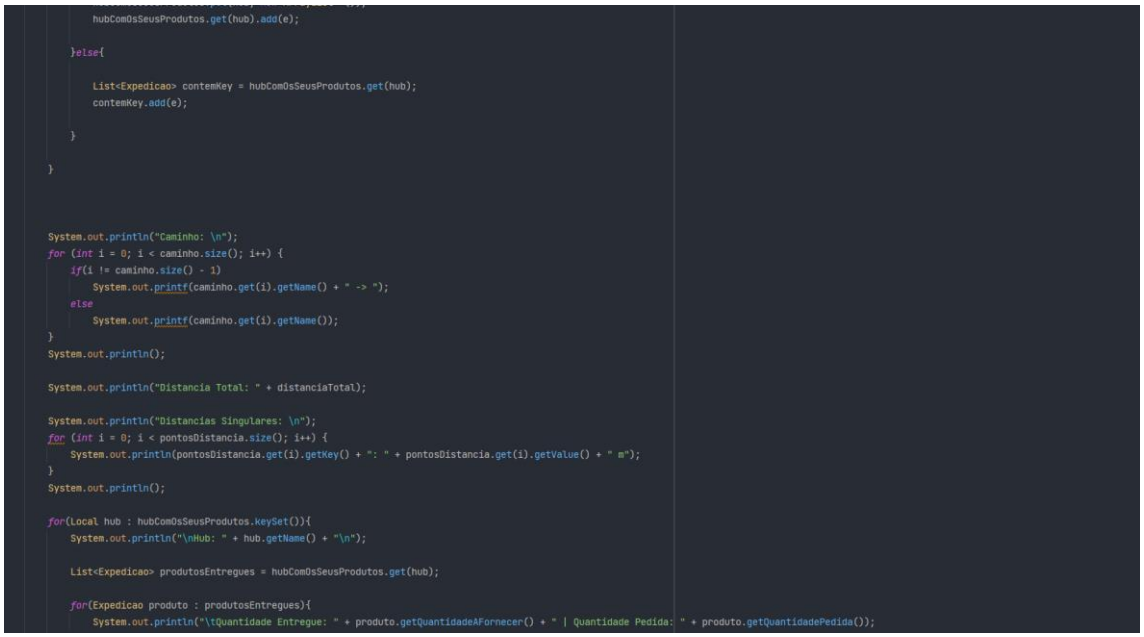
## US310: Para uma lista de expedição diária gerar o percurso de entrega que minimiza a distância total percorrida.

Tal como as User Stories anteriores, esta também continha o critério “Indicar os pontos de passagem do percurso (produtores e hubs), cabazes entregues em cada hub, distância entre todos os pontos do percurso e a distância total.”

Com base nessa informação, implementamos a ideologia em que primeiro obtemos os hubs e produtores para cada lista de expedição criada. Depois com esses valores, e com a ajuda do algoritmo ShortestPath calculamos o caminho mínimo entre todos os produtores que estão envolvidos, assim como os hubs. Esse caminho vai sendo guardado até ao final da leitura da lista de Expedição.

Por fim, com toda a informação já necessária, é impresso o caminho mais curto para a expedição bem como a distancia total e a distância entre cada ponto.

```
static void minPath(List<Local> hubs, List<Local> produtores, List<Expedicao> exp, Graph<Local, Integer> map){
    int nProdutores = produtores.size();
    int nHubs = hubs.size();
    List<Local> ProdutoresPassados = new LinkedList<>();
    List<Local> HubsPassados = new LinkedList<>();
    LinkedList<Local> caminho = new LinkedList<>();
    List<Map.Entry<String, Integer>> pontosDistancia = new ArrayList<>();
    Integer distanciaTotal = 0;

    ProdutoresPassados.add(produtores.get(0));
    caminho.add(produtores.get(0));

    while(nProdutores>0) {
        Integer minDist = getDistMinProdutor(map, caminho.getLast(), ProdutoresPassados, produtores, pontosDistancia, caminho);

        distanciaTotal += minDist;

        nProdutores--;
    }

    while(nHubs>0){
        Integer minDist =getDistMinHub(map, caminho.getLast(), HubsPassados, hubs, pontosDistancia, caminho);

        distanciaTotal += minDist;

        nHubs--;
    }

    Map<Local, List<Expedicao>> hubComOsSeusProdutos = new HashMap<>();

    for (Expedicao e : exp) {

        Local hub = e.getHub();

        if(!hubComOsSeusProdutos.containsKey(hub)){

            hubComOsSeusProdutos.put(hub, new ArrayList<>());
```



```
            hubComOsSeusProdutos.get(hub).add(e);

        }else{

            List<Expedicao> contemKey = hubComOsSeusProdutos.get(hub);
            contemKey.add(e);

        }

    }



    System.out.println("Caminho: \n");
    for (int i = 0; i < caminho.size(); i++) {
        if(i != caminho.size() - 1)
            System.out.printf(caminho.get(i).getName() + " -> ");
        else
            System.out.printf(caminho.get(i).getName());
    }
    System.out.println();

    System.out.println("Distancia Total: " + distanciaTotal);

    System.out.println("Distancias Singulares: \n");
    for (int i = 0; i < pontosDistancia.size(); i++) {
        System.out.println(pontosDistancia.get(i).getKey() + ": " + pontosDistancia.get(i).getValue() + " m");
    }
    System.out.println();

    for(Local hub : hubComOsSeusProdutos.keySet()){
        System.out.println("\nHub: " + hub.getName() + "\n");

        List<Expedicao> produtosEntregues = hubComOsSeusProdutos.get(hub);

        for(Expedicao produto : produtosEntregues){
            System.out.println("\tQuantidade Entregue: " + produto.getQuantidadeAFornecer() + " | Quantidade Pedida: " + produto.getQuantidadePedida());
```

# US311: Para uma lista de expedição calcular estatísticas

Sendo estas as estatísticas:

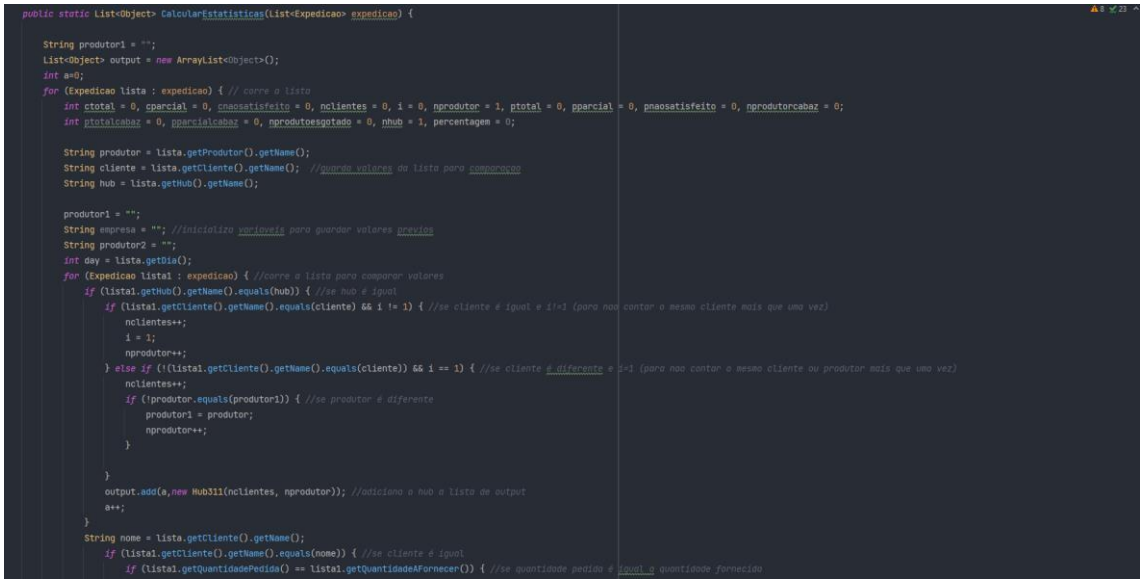
• por cabaz: nº de produtos totalmente satisfeitos, nº de produtos parcialmente satisfeitos, nº de produtos não satisfeitos, percentagem total do cabaz satisfeito, nº de produtores que forneceram o cabaz.

• por cliente: nº de cabazes totalmente satisfeitos, nº de cabazes parcialmente satisfeitos, nº de fornecedores distintos que forneceram todos os seus cabazes

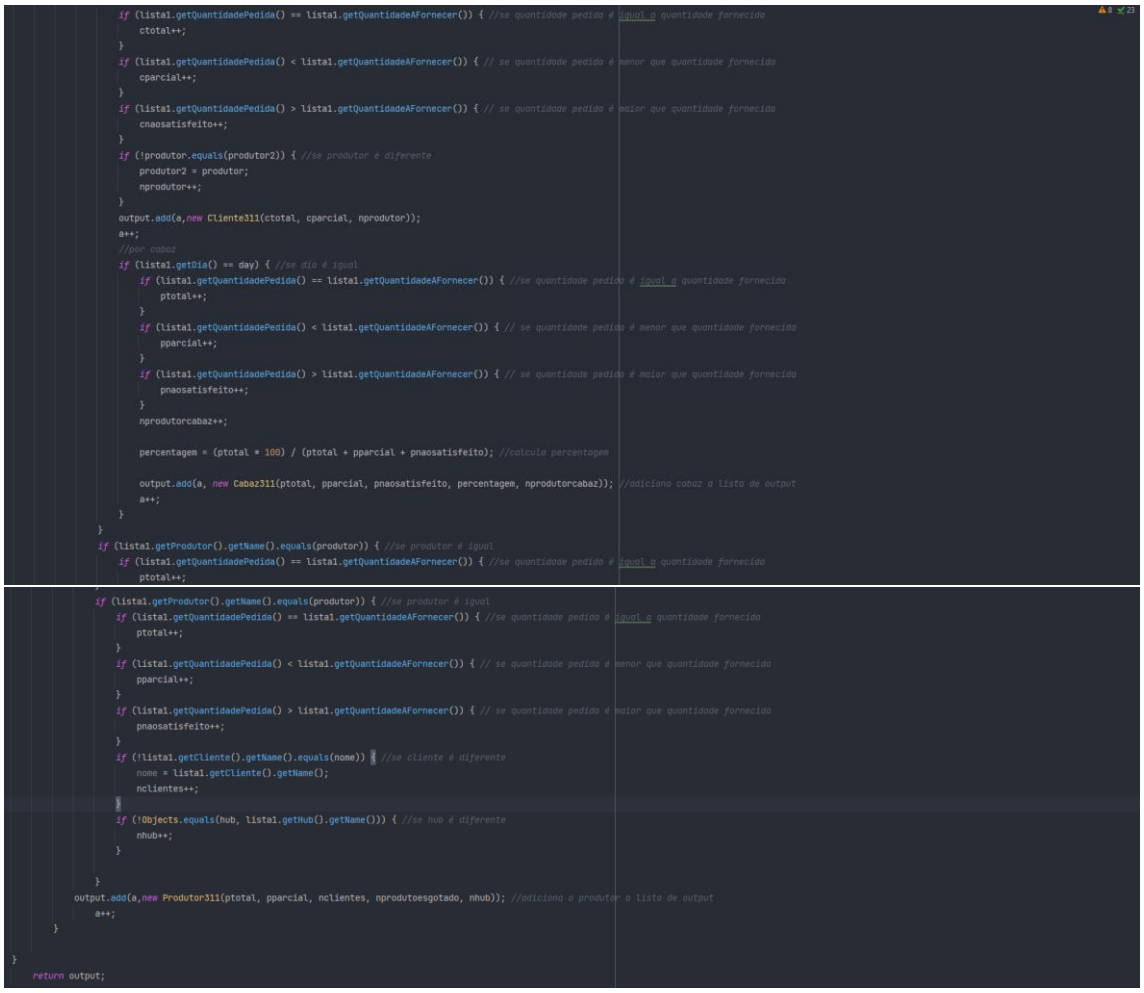
• por produtor: nº de cabazes fornecidos totalmente, nº de cabazes fornecidos parcialmente, nº de clientes distintos fornecidos, nº de produtos totalmente esgotados, nº de hubs fornecidos.

• por hub: nº de clientes distintos que recolhem cabazes em cada hub, nº de produtores distintos que fornecem cabazes para o hub.

Desenvolvemos o método CalcularEstatisticas.



```
public static List<Object> CalcularEstatisticas(List<Expedicao> expedicao) {

    String produtor1 = "";
    List<Object> output = new ArrayList<Object>();
    int a=0;
    for (Expedicao lista : expedicao) { // corre a lista
        int ctotal = 0, cparcial = 0, cnaosatisfeito = 0, nclientes = 0, i = 0, nprodutor = 1, ptotal = 0, pparcial = 0, pnaosatisfeito = 0, nprodutorcabaz = 0;
        int ptotalcabaz = 0, pparcialcabaz = 0, nprodutoesgotado = 0, nhub = 1, percentagem = 0;

        String produtor = lista.getProdutor().getName();
        String cliente = lista.getCliente().getName();  //guarda valores da lista para comparacao
        String hub = lista.getHub().getName();

        produtor1 = "";
        String empresa = ""; //inicializa variaveis para guardar valores previos
        String produtor2 = "";
        int day = lista.getDia();
        for (Expedicao lista1 : expedicao) { //corre a lista para comparar valores
            if (lista1.getHub().getName().equals(hub)) { //se hub é igual
                if (lista1.getCliente().getName().equals(cliente) && i != 1) { //se cliente é igual e i!=1 (para nao contar o mesmo cliente mais que uma vez)
                    nclientes++;
                    i = 1;
                    nprodutor++;
                } else if (!(lista1.getCliente().getName().equals(cliente)) && i == 1) { //se cliente é diferente e i=1 (para nao contar o mesmo cliente ou produtor mais que uma vez)
                    nclientes++;
                    if (!produtor.equals(produtor1)) { //se produtor é diferente
                        produtor1 = produtor;
                        nprodutor++;
                    }

                }
                output.add(a,new Hub311(nclientes, nprodutor)); //adiciona o hub a lista de output
                a++;
            }
            String nome = lista.getCliente().getName();
                if (lista1.getCliente().getName().equals(nome)) { //se cliente é igual
                    if (lista1.getQuantidadePedida() == lista1.getQuantidadeAFornecer()) { //se quantidade pedida é igual a quantidade fornecida
```



```
                if (lista1.getQuantidadePedida() == lista1.getQuantidadeAFornecer()) { //se quantidade pedida é igual a quantidade fornecida
                    ctotal++;
                }
                if (lista1.getQuantidadePedida() < lista1.getQuantidadeAFornecer()) { // se quantidade pedida é menor que quantidade fornecida
                    cparcial++;
                }
                if (lista1.getQuantidadePedida() > lista1.getQuantidadeAFornecer()) { // se quantidade pedida é maior que quantidade fornecida
                    cnaosatisfeito++;
                }
                if (!produtor.equals(produtor2)) { //se produtor é diferente
                    produtor2 = produtor;
                    nprodutor++;
                }
                output.add(a,new Cliente311(ctotal, cparcial, nprodutor));
                a++;
                //por cabaz
                if (lista1.getDia() == day) { //se dia é igual
                    if (lista1.getQuantidadePedida() == lista1.getQuantidadeAFornecer()) { //se quantidade pedida é igual a quantidade fornecida
                        ptotal++;
                    }
                    if (lista1.getQuantidadePedida() < lista1.getQuantidadeAFornecer()) { // se quantidade pedida é menor que quantidade fornecida
                        pparcial++;
                    }
                    if (lista1.getQuantidadePedida() > lista1.getQuantidadeAFornecer()) { // se quantidade pedida é maior que quantidade fornecida
                        pnaosatisfeito++;
                    }
                    nprodutorcabaz++;

                    percentagem = (ptotal * 100) / (ptotal + pparcial + pnaosatisfeito); //calcula percentagem

                    output.add(a, new Cabaz311(ptotal, pparcial, pnaosatisfeito, percentagem, nprodutorcabaz)); //adiciona cabaz a lista de output
                    a++;
                }
            }
            if (lista1.getProdutor().getName().equals(produtor)) { //se produtor é igual
                if (lista1.getQuantidadePedida() == lista1.getQuantidadeAFornecer()) { //se quantidade pedida é igual a quantidade fornecida
                    ptotal++;
                }
            if (lista1.getProdutor().getName().equals(produtor)) { //se produtor é igual
                if (lista1.getQuantidadePedida() == lista1.getQuantidadeAFornecer()) { //se quantidade pedida é igual a quantidade fornecida
                    ptotal++;
                }
                if (lista1.getQuantidadePedida() < lista1.getQuantidadeAFornecer()) { // se quantidade pedida é menor que quantidade fornecida
                    pparcial++;
                }
                if (lista1.getQuantidadePedida() > lista1.getQuantidadeAFornecer()) { // se quantidade pedida é maior que quantidade fornecida
                    pnaosatisfeito++;
                }
                if (!lista1.getCliente().getName().equals(nome)) { //se cliente é diferente
                    nome = lista1.getCliente().getName();
                    nclientes++;
                }
                if (!Objects.equals(hub, lista1.getHub().getName())) { //se hub é diferente
                    nhub++;
                }

            }
        output.add(a,new Produtor311(ptotal, pparcial, nclientes, nprodutoesgotado, nhub)); //adiciona o produtor a lista de output
            a++;
        }

}
    return output;
```

Com este método, o que fizemos foi percorrer a lista de Expedicoes e comecamos por armazenar valores para a comparacao, como os nomes do Cliente, do Produtor e nome Hub.

Verificamos primeiramente se os hubs eram iguais, tal como os clientes/produtor, onde aí houve uma verificacao caso já tivessem sido contados para a estatística. Com isso adicionamos esses dados da estatistca a uma List<Object>. O raciciocinio é análogo para as seguintes estatisticas. Pelo Cabaz verificamos as comparacoes entre a quantidade pedida pelo o Cliente e a quantidade Fornecida, satisfazendo tambem para os dados necessarios para solucionar todas as estatisticas dos clientes e produtores, armazenando, consequentemente, os dados relevantes para a User Story na lista output.