PROGRAMAÇÃO AVANÇADA

Relatório do Projeto

2022/2023

DOCENTE: LUÍS DAMAS

2º SW-01

CATARINA JESUS Nº 202000594

DANIEL ROQUE Nº 201901608

ANDRÉ MESEIRO Nº 202100225

PEDRO ANJOS Nº 202100230

Conteúdo

Introdução

O objetivo do nosso projeto é desenvolver uma aplicação em Java que utilize um grafo como modelo de dados, segundo os princípios da orientação a objetos e com utilização de padrões de software. O programa consiste numa aplicação gráfica na qual o utilizador visualiza, manipula e obtém informação diversa sobre uma rede de transporte em autocarro.

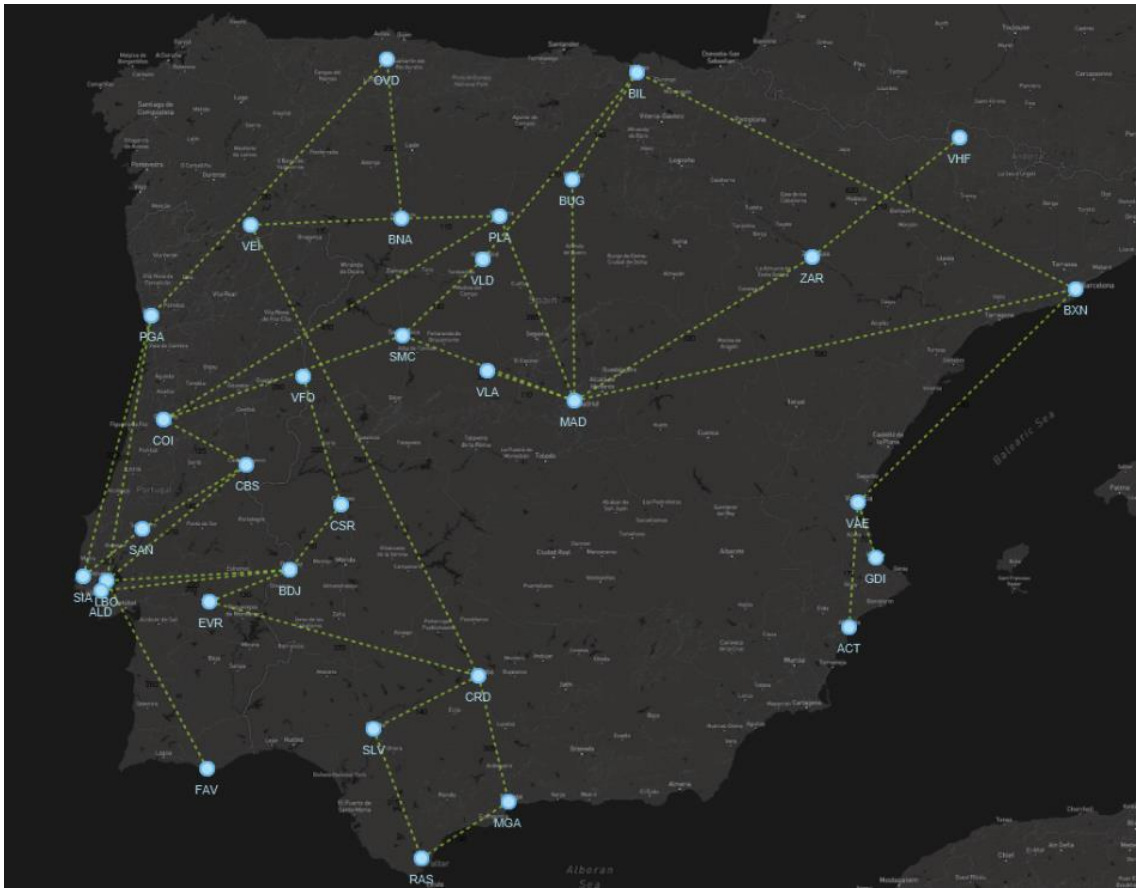


Figura 1 - Rede de transporte em autocarro (dataset: iberia)

Tipos Abstratos de Dados

Para a resolução do problema proposto, o tipo abstrato de dados utilizado foi o ADT Graph, que é constituído por duas componentes: Stop e Route (vértice e aresta, respetivamente).

Como referido anteriormente, foi feita a implementação do ADT Graph utilizando listas de adjacência como estrutura de dados. A mesma pode ser representada através de uma coleção de vértices, em que cada vértice guarda a sua lista de arestas incidentes:

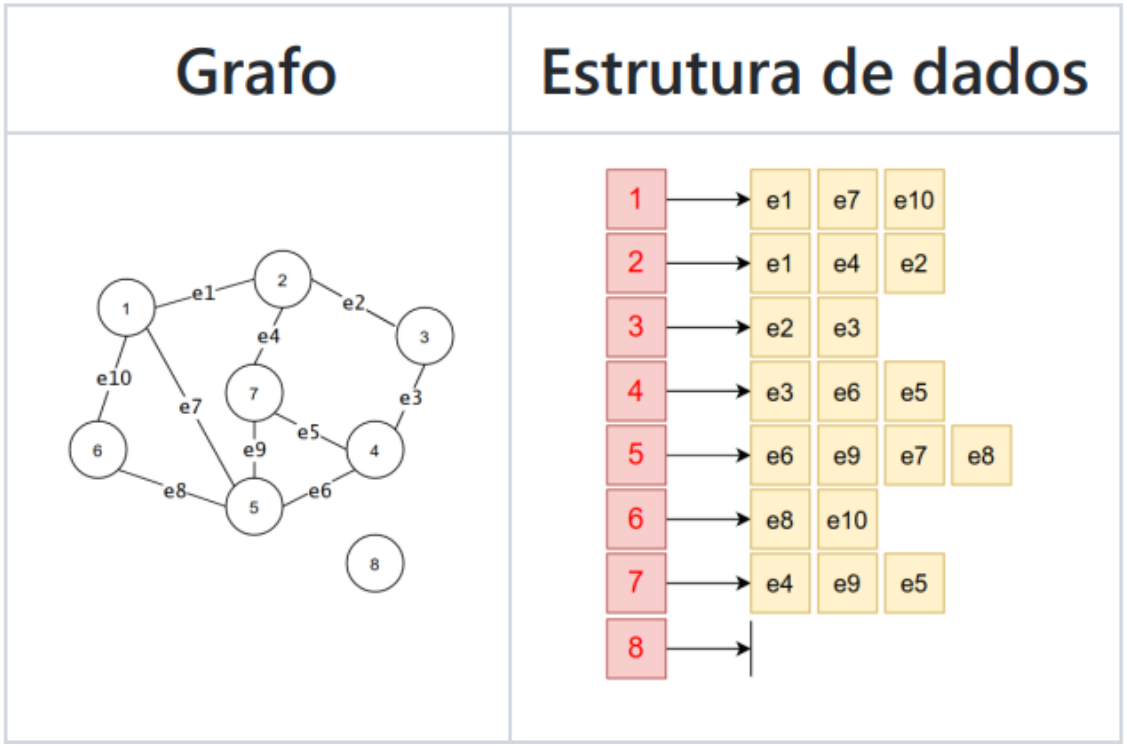


Figura 2 - Exemplo de lista de adjacências

Vantagens e desvantagens desta estrutura de dados:

- Vantagens:
  - Fácil adicionar vértices e arestas;
  - Fácil saber arestas incidentes a um vértice.
- Desvantagens:
  - Difícil saber se existe uma aresta entre v e w, ou seja, se existe uma aresta entre dois vértices.

Relativamente à interface gráfica desenvolvida, encontram-se em baixo screenshots das várias janelas, bem como uma breve descrição das mesmas.

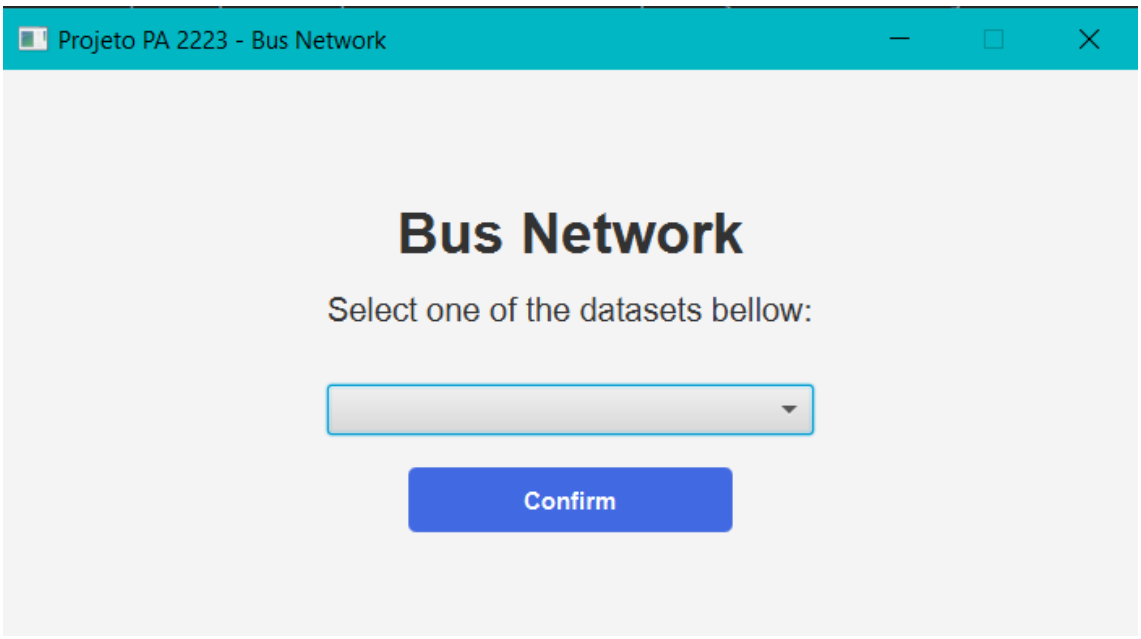


Figura 3 - Primeira janela em que se seleciona o mapa que se pretende visualizar

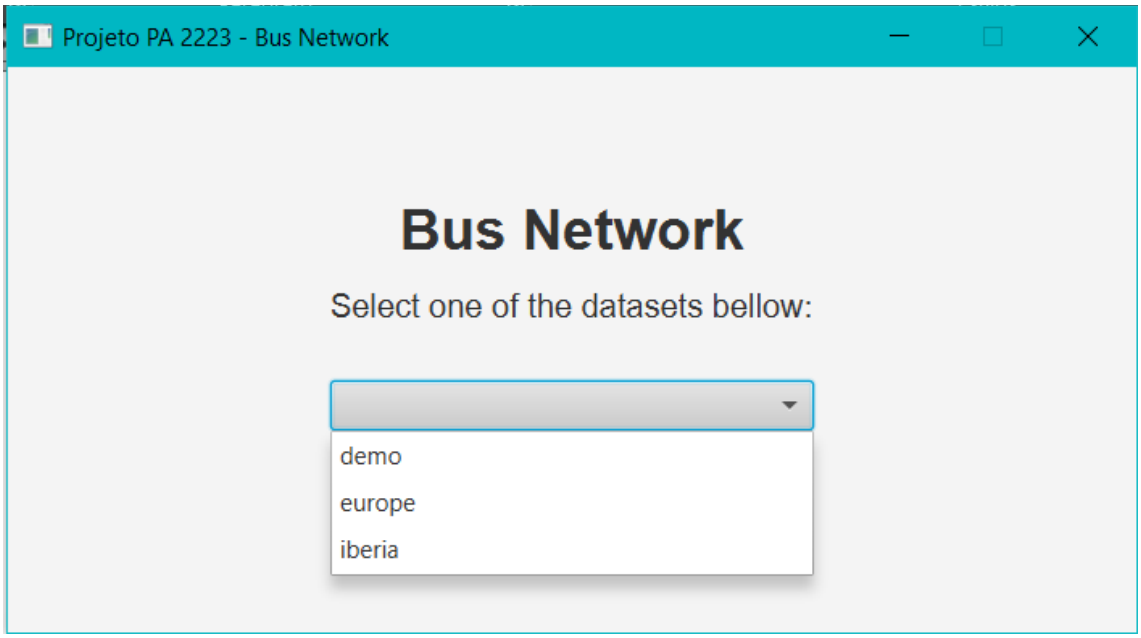


Figura 4 - Mapas disponíveis para escolha

Nas figuras 3 e 4, é possível visualizar as janelas para escolha do dataset que pretendemos visualizar. As opções de escolha são: demo, que é o mapa de Portugal; iberia, que é o mapa da Península Ibérica; europe, que é o mapa da Europa.

Após a escolha pelo utilizador, ao clicar no botão “Confirmar” o mesmo será levado às seguintes figuras:

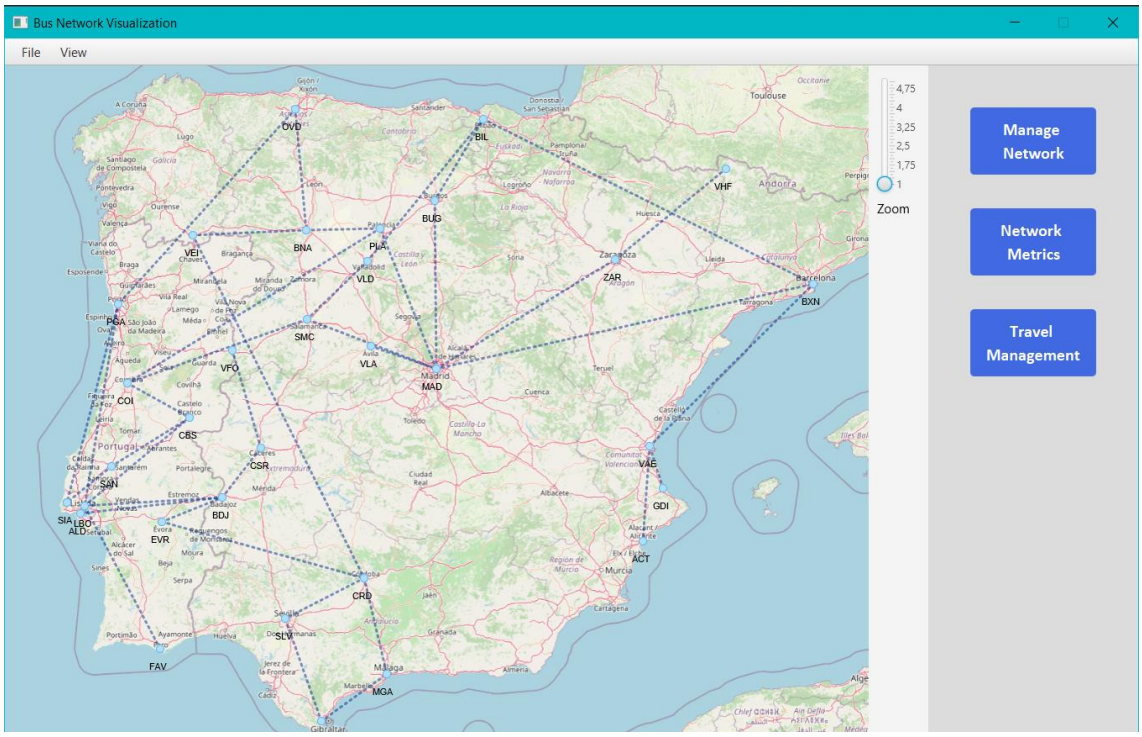


Figura 5 - Dataset Iberia e demo, correspondem ao mesmo

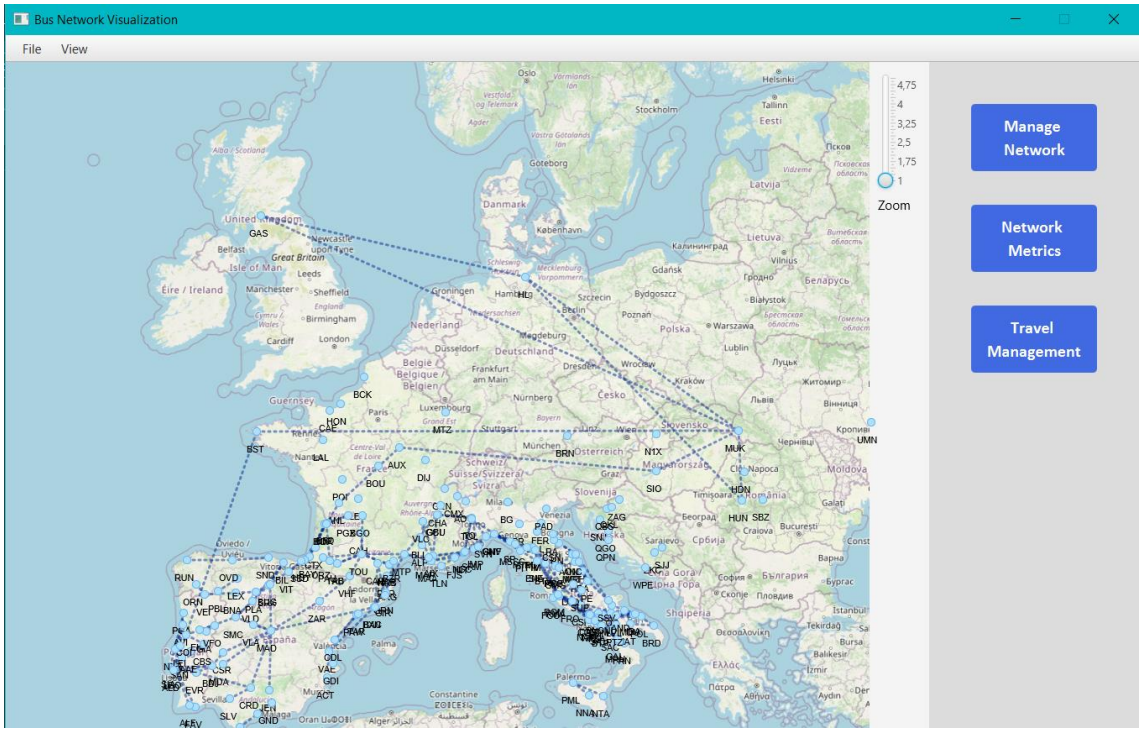


Figura 6 - Dataset Europe.

Para mudar de dataset, acede-se a “File” na janela das figuras 5 e 6, e seleciona-se a opção “Change Dataset”. Caso se queira exportar o dataset, seleciona-se a opção “Export Dataset”. Por fim, para sair da aplicação, seleciona-se a opção “Exit”.

Ao se aceder à opção “View”, é possível alterar o fundo do mapa visualizado, selecionando uma das opções de “Change Background”, como é apresentado na figura 8.

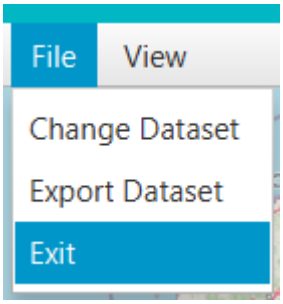


Figura 7 - Opções do File

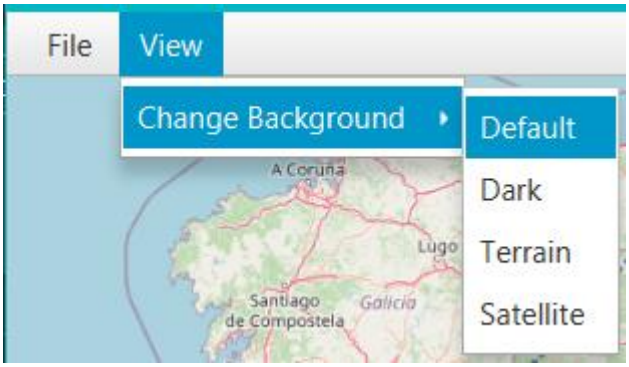


Figura 8 - Opções da View

Ao clicar no botão “Manage Network”, é apresentada a seguinte janela, que contém os Stops existentes no mapa, as Routes existentes no mapa e a opção de adicionar e remover Routes, em que o “Initial Stop Name” e o “End Stop Name” podem ser selecionados no mapa com dois cliques do rato, ou então definidos nos respetivos sítios destinados para tal.

Existe também uma funcionalidade de “Undo” implementada, que permite retirar uma Route depois de adicionada ou voltar a adicionar uma Route depois de removida.

Por fim, ao clicar na opção “Back to Main Menu”, volta-se à janela apresentada nas figuras 5 e 6.

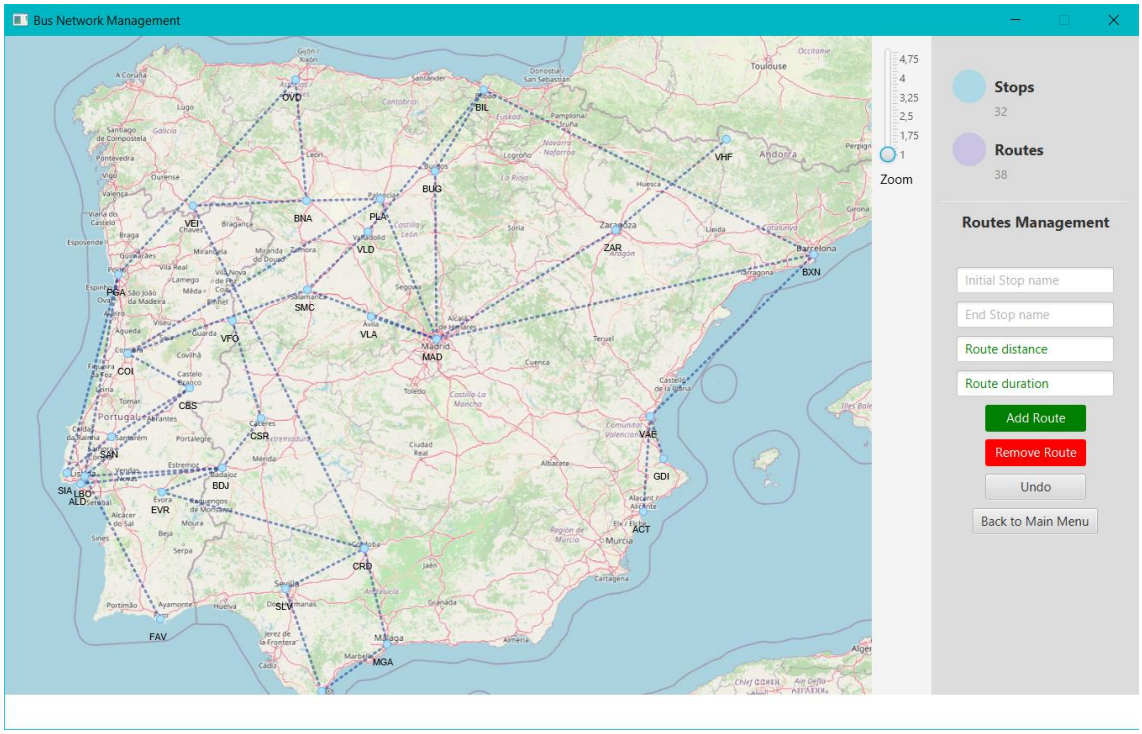


Figura 9 - Manage Network

Ao clicar no botão “Network Metrics”, é possível obter informação acerca das métricas do mapa selecionado: o número de Stops, o número de Routes, o número de Componentes, a centralidade dos Stops com o nome do mesmo e o número de Stops adjacentes, bem como um gráfico dos Stops Centrais com o número de Stops adjacentes por cidade.

Selecionando a opção “Return to Menu”, é apresentado o menu apresentado nas figuras 5 e 6.

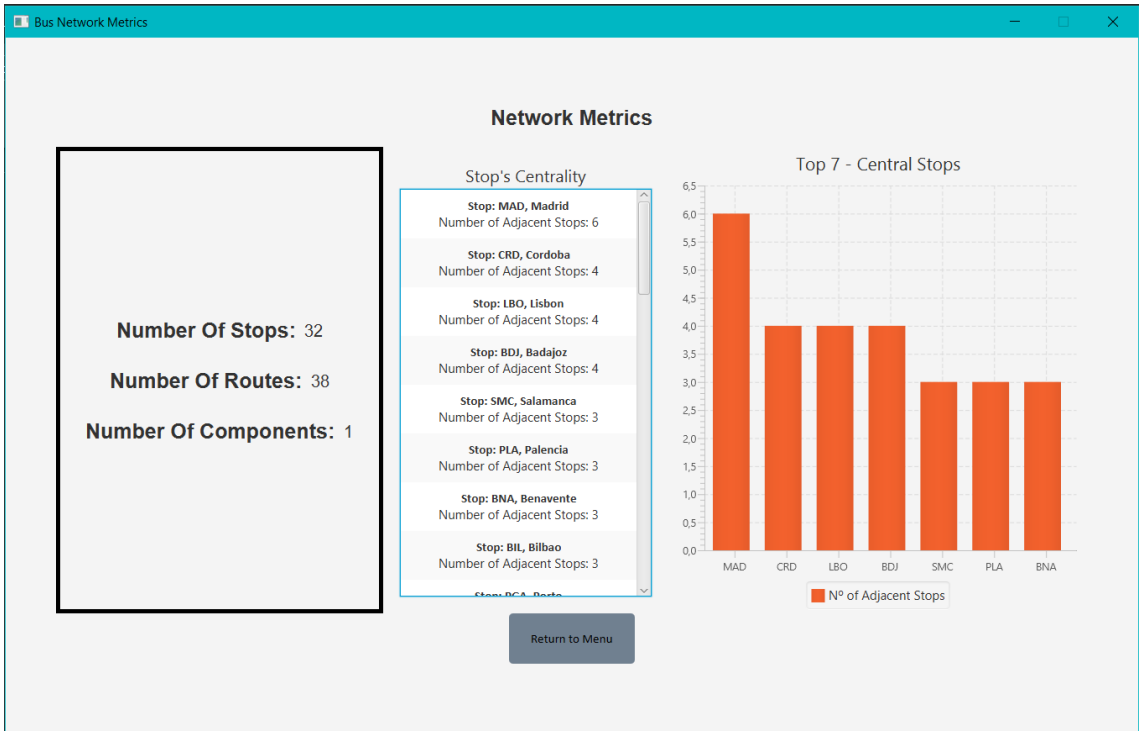


Figura 10 - Network Metrics

Na opção “Travel Management”, é possível descobrir o caminho mais curto entre dois Stops, aqui também é possível selecionar os Stops com dois cliques do rato, sendo possível calcular conforme a distância ou duração e no “Cost” irá aparecer o valor correspondente à distância ou duração, consoante a opção selecionada.

Também é possível descobrir os Stops que distam N rotas do Stop selecionado, escolhendo o Stop inicial e o número máximo de Stops, sendo o resultado apresentado posteriormente no “Number of Stops”.

Existe também a funcionalidade de guardar a Route em formato PDF sob a forma de bilhete em 3 formas diferentes: simples, intermédio e completo, de acordo com um nome escolhido pelo utilizador, e neste caso terão que estar selecionados dois Stops, o inicial e o final, para que se possa conseguir guardar o Ticket, como é mostrado na figura 12.

Ao selecionar a opção “Show Furthest Stops”, é possível visualizar o caminho, mostrado a vermelho, e os Stops com um círculo a preto, dos Stops mais distantes, sempre calculados pela distância, como é possível verificar na figura 13.

Por fim selecionando a opção “Back to Main Menu”, é apresentado o menu presente nas figuras 5 e 6.

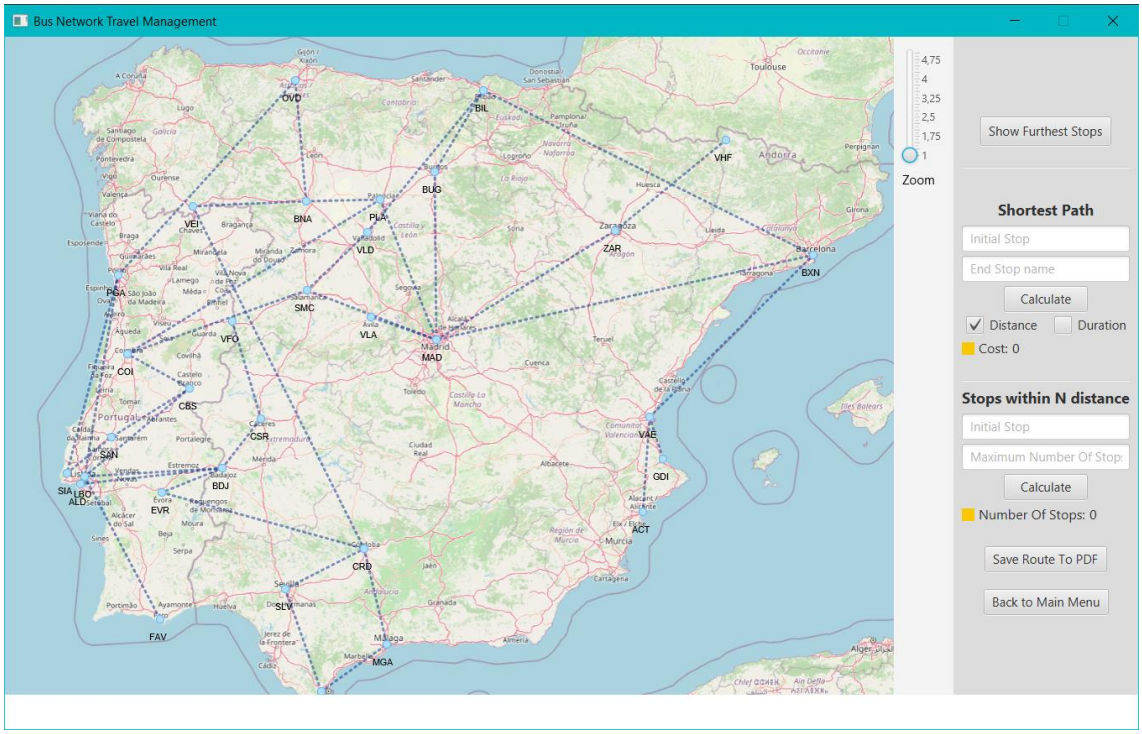


Figura 11 - Travel Management

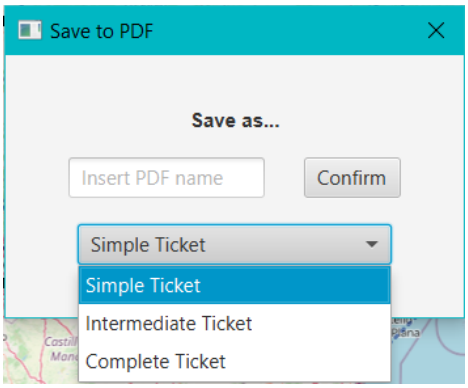


Figura 12 - Save to PDF

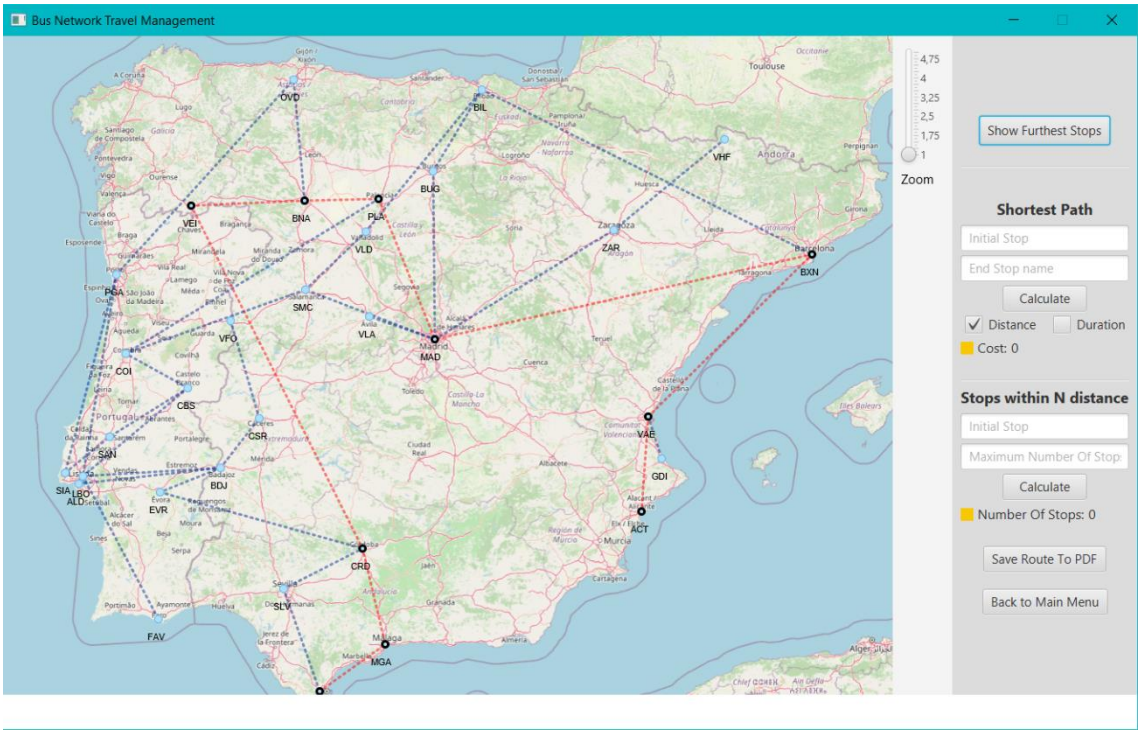


Figura 13 - Show Furthest Stops

Diagrama de Classes

Figura 14 - Diagrama de Classes completo

Padrões de software

Padrão MVC (Model View Controller):

Explicação: Devido à popularidade e à importância deste padrão de software em várias aplicações, o mesmo foi implementado com o objetivo de facilitar o trabalho ao dividir todas as lógicas pelas suas pastas, a pasta do Model, a pasta das View e a pasta Controller que controla todo o modelo, separando assim as funcionalidades principais do negócio.

- **Model:** Este participante preocupa-se com os dados que serão apresentados e contém as operações a serem aplicadas para transformar os dados.
- **View:** Este conhece o participante **Model**, mas não o manipula, apenas consulta a sua informação. Também define como os dados serão visualizados pelo utilizador e delega ao **Controller** as ações/intenções do utilizador.
- **Controller:** Este participante sincroniza as ações do utilizador com a manipulação do modelo, implementa a lógica de controle e validação e pode manipular o participante **View** diretamente.

Neste caso, o **Model** é apresentado nas classes AutoCompleteText, Route e Stop, onde estão os métodos para auto-completar o texto, ou seja, os campos, os seletores da Route e do Stop.

O **Controller** é representado pelas classes CalculateTravelController e RouteManagerController e a **View** é representada pelos packages do MainMenu, Metrics, RouteManage e TravelManage, onde cada um contém as classes necessárias para o menu principal, métricas, gestão de rotas e gestão de viagem, respetivamente.

Padrão Observer:

Explicação: Padrão que define uma dependência um-para-muitos entre objetos de modo que quando um objeto muda o estado, todos os seus dependentes são notificados e atualizados automaticamente, uma vez que não existe herança múltipla em Java a utilização deste padrão tornou-se pertinente. Foi utilizado como auxílio à importação para o Grafo e ao MVC, para que a View seja notificada quando existem alterações no Model e assim fazer a atualização na parte gráfica da aplicação.

Padrão Strategy:

Explicação: Este padrão permite que um objeto troque de estratégia ou de comportamento em tempo de execução. Neste caso, foi implementado para mostrar a parte da duração e distância, evitando assim a existência de código duplicado.

Padrão Command:

Explicação: Este padrão serve para gerir a execução de comandos. O mesmo foi utilizado na implementação das funcionalidades de Undo, Add e Remove, também está presente nos comandos utilizados e numa interface.

Padrão Factory Method:

Explicação: O Factory Method é um padrão de projeto criacional que fornece uma interface para criar objetos em uma superclasse, mas permite que as subclasses alterem o tipo de objetos que serão criados. O mesmo foi utilizado na parte correspondente aos bilhetes, uma vez que são pedidos três formatos dos mesmos, foram utilizadas três classes, uma para cada tipo, havendo assim a delegação de trabalho.

Refactoring

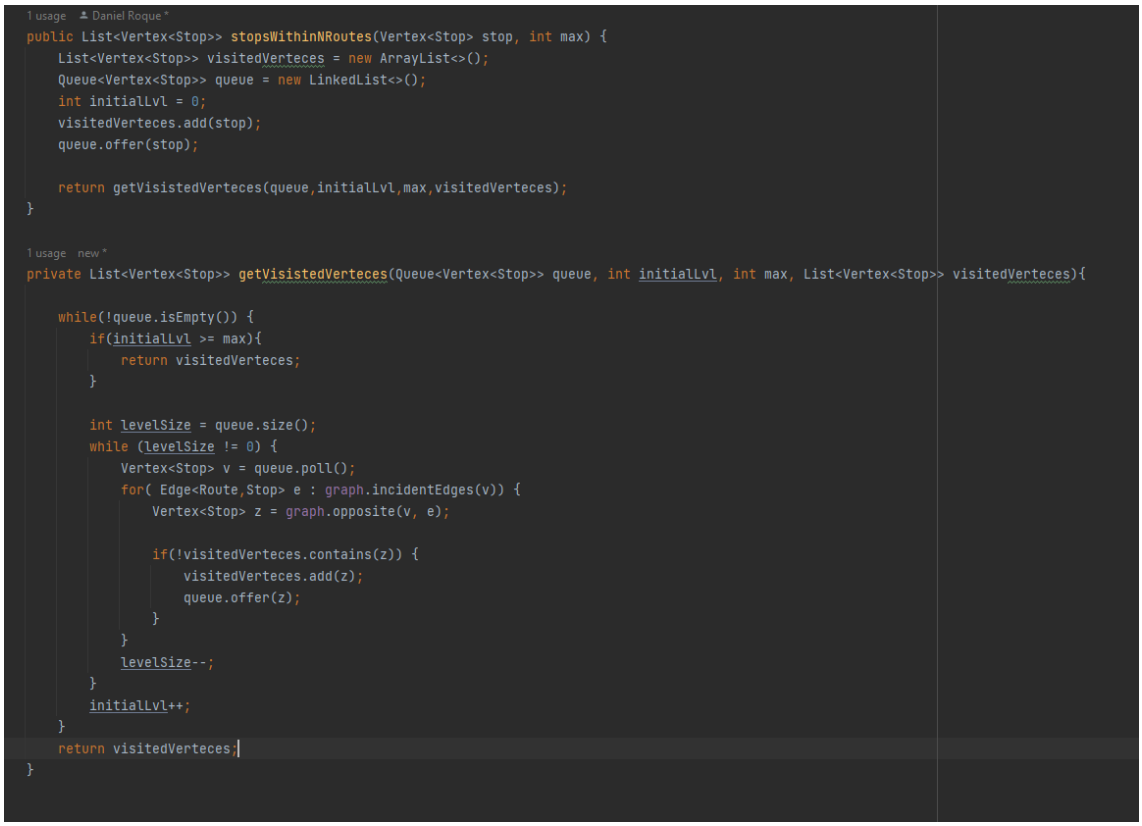
Tipo de Code Smells	Quantidade de situações	Técnica de Refactoring
Long Method	4	Extract Method

O code smell Long Method indica a presença de demasiado código num só método. Neste caso, encontra-se este tipo de code smell na classe ImportToGraph, no método Paragens (stops) que distam N rotas da paragem P.

```java
/**
 * Finds all the vertices with max N routes.
 *
 * @param stop Vertex the stop
 * @param max maximum number of routes to search for
 * @return List with the stops
 */
1 usage  Daniel Roque
public List<Vertex<Stop>> stopsWithinNRoutes(Vertex<Stop> stop, int max) {
    List<Vertex<Stop>> visitedVerteces = new ArrayList<>();
    Queue<Vertex<Stop>> queue = new LinkedList<>();
    int initialLvl = 0;
    visitedVerteces.add(stop);
    queue.offer(stop);

    while(!queue.isEmpty()) {
        if(initialLvl >= max){
            return visitedVerteces;
        }

        int levelSize = queue.size();
        while (levelSize != 0) {
            Vertex<Stop> v = queue.poll();
            for( Edge<Route,Stop> e : graph.incidentEdges(v)) {
                Vertex<Stop> z = graph.opposite(v, e);

                if(!visitedVerteces.contains(z)) {
                    visitedVerteces.add(z);
                    queue.offer(z);
                }
            }
            levelSize--;
        }
        initialLvl++;
    }
    return visitedVerteces;
}
```

Figura 15 - Code smell Long Method

A solução para o code smell apresentado é aplicar a técnica de refactoring “Extract Method”, dividindo o long method em dois métodos, ficando com o seguinte resultado, ao acrescentar um método auxiliar privado.



```
1 usage   Daniel Roque *
public List<Vertex<Stop>> stopsWithinNRoutes(Vertex<Stop> stop, int max) {
    List<Vertex<Stop>> visitedVerteces = new ArrayList<>();
    Queue<Vertex<Stop>> queue = new LinkedList<>();
    int initialLvl = 0;
    visitedVerteces.add(stop);
    queue.offer(stop);

    return getVisistedVerteces(queue,initialLvl,max,visitedVerteces);
}

1 usage   new *
private List<Vertex<Stop>> getVisistedVerteces(Queue<Vertex<Stop>> queue, int initialLvl, int max, List<Vertex<Stop>> visitedVerteces){

    while(!queue.isEmpty()) {
        if(initialLvl >= max){
            return visitedVerteces;
        }

        int levelSize = queue.size();
        while (levelSize != 0) {
            Vertex<Stop> v = queue.poll();
            for( Edge<Route,Stop> e : graph.incidentEdges(v)) {
                Vertex<Stop> z = graph.opposite(v, e);

                if(!visitedVerteces.contains(z)) {
                    visitedVerteces.add(z);
                    queue.offer(z);
                }
            }
            levelSize--;
        }
        initialLvl++;
    }
    return visitedVerteces;
}
```

Figura 16 - Code smell resolvido

Conclusão

Com a realização deste projeto, foram aprofundadas as capacidades no que toca à programação orientada a objetos e trabalho em equipa. No decorrer do mesmo, foram encontradas algumas dificuldades, tal como a implementação de algumas funcionalidades, por não existirem certezas absolutas da forma como deveria ser feita a mesma, mas foram ultrapassadas com o decorrer do projeto.

Por fim, foi um projeto relativamente bem-sucedido, pois apesar das dificuldades, as mesmas foram ultrapassadas, e a maioria das funcionalidades pedidas foi implementada com sucesso, para além das obrigatórias.