Projecto Final

Sistema de Localização e Orientação Geográfica (S.L.O.G.)

Orientado por

Prof. Hugo Proença

do Departamento de Informática

da

Universidade da Beira Interior

AGRADECIMENTOS

Ao professor Hugo Proença, orientador deste projecto, pela ajuda, apoio, paciência e disponibilidade que sempre manifestou.
Ao meu colega de projecto Hélder Pires, pela sua amizade e espírito de entreajuda.
Aos colegas Bruno Silva e Rui Guarda pelo companheirismo e colaboração, assim como aos restantes colegas de curso.
Quero ainda agradecer de uma forma muito especial a minha namorada, aos meus pais e irmão, assim como aos restantes familiares e amigos pelo constante incentivo e forte apoio que nunca será esquecido.

Um obrigado a todos!

SUMÁRIO

Com a realização deste relatório pretende-se atingir determinados objectivos nomeadamente:

- Explicar as principais motivações que levaram à sua escolha;
- Expor o seu alcance;
- Mostrar o *background* necessário ao seu desenvolvimento;
- Descrever o processo de trabalho nas várias fases de desenvolvimento;
- Fazer uma reflexão crítica sobre o trabalho desenvolvido.

Índice

Índice de Figuras

Advertências

Por uma questão de clareza optei pela utilização de apenas um tipo de letra, a Arial. Todas as legendas das figuras respeitam esta opção.

Neste documento, todo o código ou pseudo código inserido é sempre referenciado utilizando o tipo de letra *Courier New*, tamanho 10 e quando que se justifique é emoldurado numa caixa de texto.

Os acrónimos utilizados estão disponíveis na secção de Acrónimos. Alguns termos em Inglês não estão traduzidos por desconhecimento de tradução adequada, embora sejam sempre referidos em itálico. É de referir ainda que certos termos e designações utilizados estão descritos no Glossário.

As imagens são referenciadas pelo número do capítulo a que pertencem e a ordem a partir desse capítulo.

Acrónimos

JSP	Java Server Pages
JSTL	JavaServer Pages Standard Tag Library
HTML	Hiper Text Markup Language
DHTML	Dynamic Hiper Text Markup Language
XML	Extensible Markup Language
JDBC	Java Data Base Connection
SQL	Structured Query Language
URL	Universal Resource Locater
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
J2SE	Java 2 Plataform, Standard Edition
J2EE	Java 2 Plataform, Enterprise Edition
SDK	Software Development Kit
EJB	Enterprise JavaBean
API	Application Program *Interface*
WSDP	*Web* Services Developer Pack
WWW	World Wide Web
PHP	Professional Home Page
ASP	Active Server Pages
SLOG	Sistema Localização Orientação Geográfica

Glossário

Driver	Controlador de conversão de formato de informação.
Script	Ficheiro que contém um ou mais módulos de programação.
Site	Sítio na Internet.
Download	Acção de descarregar ficheiros ou programas da Internet.
Tag	Indicações de elementos HTML e JSP.
Browser	Software Cliente de navegação na Internet.
Tomcat	Servidor *web* que suporta a tecnologia JSP. JSP *container* oficial da *Sun*.

Web

Consiste numa rede de redes interligadas, é a maior rede informática existente, interligando máquinas e redes à escala planetária. Permite o acesso a uma enorme variedade de serviços nas mais diversas áreas de actividade.

servlet

Programa Java que corre num servidor *Web*. Os *servlets* são carregados apenas uma vez e como são executados de forma *multi-thread* podem atender mais do que um pedido simultaneamente.

1. Introdução

1.1 Introdução

O presente relatório insere-se no âmbito do projecto denominado **E-Vinhais** desenvolvido na disciplina de Projecto da licenciatura de Matemática/Informática.
O Relatório pretende fornecer um enquadramento das várias fases de desenvolvimento do projecto.

1.2 Objectivo

O objectivo do projecto foi desenvolver uma aplicação, para posterior integração no site da Câmara Municipal de Vinhais, que permitisse calcular o trajecto óptimo entre dois pontos distintos, mediante uma função de custo.

Com este trabalho pretendi desenvolver uma interface agradável, que resulta-se num sistema de informação onde fosse possível analisar os resultados obtidos e ainda verificar o percurso através da visualização de imagens.

1.3 Motivações

Obviamente o interesse deste projecto não é meramente académico, antes pelo contrário as suas potencialidades de mercado são consideráveis. Na sociedade actual as pessoas necessitam cada vez mais de instrumentos que lhes permitam uma melhor localização e orientação geográfica.

O desenvolvimento de uma aplicação deste género, vai permitir a todos os utilizadores saber qual o caminho óptimo entre dois locais, quanto tempo necessita para realizar esse percurso, entre outras informações relevantes.

1.4 Restrições

Este projecto obedece a algumas restrições, nomeadamente a utilização da linguagem JSP/Java para a programação da Aplicação e a utilização do MySQL para a criação e manutenção da base de dados.

1.5 Descrição

Este projecto divide-se fundamentalmente em duas partes: a primeira diz respeito à procura do trajecto óptimo entre dois pontos distintos, a outra tem a ver com a *interface* gráfica onde serão Apresentados os resultados.

A primeira preocupação foi encontrar um algoritmo que respondesse a primeira parte, ou seja, dando dois locais distintos, o algoritmo devolve uma lista com as ruas e os locais a atravessar para de uma maneira óptima, chegarmos do local inicial ao local final.

A fase seguinte do projecto consistiu no desenvolvimento de uma *interface* gráfica agradável, simples e fácil de utilizar, através da qual fosse possível analisar os resultados da aplicação do algoritmo sobre os dados iniciais.

1.5.1 Funcionalidades

Ao iniciar a aplicação o utilizador depara-se com uma página inicial que para além de uma breve descrição tem ainda duas caixas de selecção e um botão.
Através das caixas de selecção o utilizador selecciona dois locais distintos (origem e destino).
O botão permite executar o algoritmo e assim determinar o trajecto óptimo entre ambos os locais.
A aplicação extrai toda a informação de que necessita para a execução do algoritmo de uma base de dados que define a rede viária da região.

Após a execução do algoritmo é apresentada numa página de resultados, uma imagem que mostra os locais, bem como as ruas que constituem o percurso. Nesta página podemos ainda observar outras informações, tais como a distância total do percurso, o tempo necessário a sua realização ou ainda o grau de interesse turístico, entre outras funcionalidades, como veremos no manual de utilização, que faz parte dos apêndices deste relatório.

1.5.2 Estrutura do Programa

O programa está dividido fundamentalmente em três blocos. Um que representa o conhecimento, a base de dados. Outro que trata da parte de *interface* com o utilizador baseado num ambiente em janelas. Por último temos o bloco de carácter essencialmente algorítmico que efectua as computações necessárias às respostas que o programa deve dar.

No que diz respeito ao primeiro grande bloco, ele é talvez o mais simples, uma vez que trata da representação do conhecimento através da descrição em predicados próprios da rede viária.

O segundo bloco é constituído pela *interface* gráfica responsável pelo controle das acções do utilizador, quando este interage com o ambiente gráfico (carregar em botões, para obter respostas). Esta *interface* é também responsável pela apresentação gráfica das respostas dadas pelo programa em forma de texto e imagem.

O último bloco possui as estruturas que manipulam a informação e efectuam os cálculos, de maneira a construir uma resposta a solicitação do utilizador. Como é evidente há em todo este processo de manipulação das estruturas um permanente contacto com o primeiro bloco.

1.5.3 Esquemas de Representação de Conhecimento

A estrutura utilizada para a representação do conhecimento, contém toda a informação necessária a concretização dos objectivos ao nível da aplicação que se pretende realizar.

Inicialmente a descrição do trajecto óptimo era apenas em termos de distâncias, mais tarde foram acrescentados outros critérios como o tempo ou o interesse turístico.
Também o valor das coordenadas, latitude e longitude foram introduzidos *à posteriori*.

Apesar de desde muito cedo se ter definido a estrutura base da representação do conhecimento ela foi sempre evoluindo com o evoluir das funcionalidades da aplicação.

Esta secção não trata, nem pretende tratar, minuciosamente o esquema de representação do conhecimento, esse tema será abordado no capítulo 3 deste relatório.

1.5.4 Implementação dos Esquemas de Representação de Conhecimento

Todo o conhecimento é representado por uma estrutura de dois predicados:

- vertices(idvertice,descricao)
- arestas(idaresta,descricao,idverticepartida,idverticeChegada,custo,tempo,turismo)

Os predicados traduzem a existência de vértices e arestas ambos com características importantes. No caso das arestas elas unem dois vértices, isto é, cada aresta é constituída por um vértice de partida e por um vértice de chegada.

A seguir será dada uma explicação mais detalhadamente de cada um deles.

- vertices(idvertice,descricao)

O predicado **vertices** guarda a informação relativa aos vértices existentes, onde cada vértice representa um local. Na estrutura *idvertice* é o identificador do vértice (único), por exemplo: 1, 18, etc.
A *descricao* não é mais que a designação do vértice, ou seja, o nome do local (por exemplo: Couto, Bastos, Pedroso, …).

- arestas(idaresta,descricao,idverticepartida,idverticeChegada,custo,tempo,turismo)

O predicado **arestas** guarda a informação relativa às ruas existentes em toda a rede viária.
O campo *idaresta* representa o código (único) atribuído a cada aresta. Na estrutura, *descricao* significa o nome da aresta (rua).
O *idverticepartida* representa o vértice de partida (local de origem), enquanto que o campo *idverticeChegada* representa o vértice de chegada (local de destino).
O campo *custo* tem o valor da distância em metros da aresta, que liga o vértice de partida ao vértice de chegada, assim como o *tempo* representa o tempo em minutos (valor aproximado) necessário para percorrer tal aresta.
Por último o campo *turismo* que denota o grau de interesse turístico que a aresta possui. O grau de interesse turístico é determinado com base nos monumentos, casas comerciais, entre outros factores relacionados com o turismo.

1.5.5 Análise da Complexidade do Algoritmo

A estrutura é apenas o suporte de todo um conjunto de dados disponíveis na base de dados sobre os quais podemos tirar conclusões para servir os utilizadores da aplicação.
Assim interessa em larga medida e com algum detalhe, descrever o algoritmo que manipula essa informação, e que dela extrai conclusões para chegar a um objectivo, que é responder da melhor maneira à solicitação do utilizador.

A estrutura descrita anteriormente pode ser vista como uma rede de vértices (locais) e arestas (ruas) fazendo lembrar um grafo. Foi exactamente tendo em conta esta semelhança, que se decidiu escolher como algoritmo de pesquisa do caminho óptimo o ***algoritmo de Dijkstra*** (também conhecido como *Shortest Path*).

Este algoritmo é aquilo a que se costuma chamar de algoritmo "ganancioso", pois procura a melhor solução a cada passo, tendo como objectivo final a obtenção da melhor solução global.

É um algoritmo que prima pela simplicidade, como já disse ele procura em cada passo (vértice) o caminho (aresta) com menor custo e percorre essa aresta, fazendo apenas uma verificação importante: a de ver se cada vértice não é percorrido mais do que uma vez (para impedir que entre em ciclo).

O algoritmo termina quando o novo vértice é aquele ao qual ele queria chegar (vértice de chegada).

No capítulo 4 será abordado com mais profundidade o ***algoritmo de Dijkstra***.

1.6 Cronograma

Este cronograma pretende ser um mapa do percurso desde o estado embrionário do projecto, até à sua conclusão.

Março:

- Análise de requisitos;
- Pesquisa de informação sobre JSP e MySQL;
- Pesquisa de informação sobre o ***algoritmo de Dijkstra***.

Abril – Maio:

- Instalação e configuração dos servidores (*Apache* e *Tomcat*);
- Primeiros contactos com a linguagem JSP;
- Primeira abordagem ao ***algoritmo de Dijkstra***;

Maio – Junho:

- Definição da base de dados;
- Implementação do ***algoritmo de Dijkstra***;
- Construção da *interface* Gráfica.

Junho – Julho:

- Conclusão da implementação da *interface* Gráfica;
- Elaboração do relatório de projecto.

1.7 Estrutura do Relatório

O Relatório está estruturado em seis Capítulos:

1ºCapítulo – Introdução

São apresentados o objectivo, motivações e restrições do projecto, bem como a sua descrição. É apresentado ainda um cronograma que descreve as várias etapas da realização do projecto.

2º Capítulo – Software

Descreve os vários recursos, linguagens de programação e ferramentas necessárias para a realização do projecto.

3º Capítulo – Base de Dados

Descrição de cada uma das tabelas que compõem a base de dados e algumas das suas características.

4º Capítulo – Algoritmo de Dijkstra

É feita uma descrição do algoritmo e do seu modo de funcionamento, bem como uma exemplificação da sua utilização.

5º Capítulo – Apresentação da Aplicação

Refere os critérios de funcionamento da arquitectura da aplicação, bem como o organograma da sua estrutura. Apresenta ainda imagem da aplicação.

6º Capítulo – Conclusões

Neste capítulo são feitas as considerações final acerca do projecto e ainda referenciados possíveis melhoramentos a aplicação.

2. Software

2.1 Introdução

Para a elaboração da aplicação, foi fundamental logo à partida decidir pelos recursos, ferramentas e linguagens de programação que fossem uma mais valia para o projecto e programador. No presente capítulo pretende-se descreve os elementos que suportam a aplicação bem como o porquê da sua escolha.

2.2 Ambiente de Desenvolvimento

Para o desenvolvimento da aplicação foi utilizado o *JBuilder* da *Borland Software Corporation*.

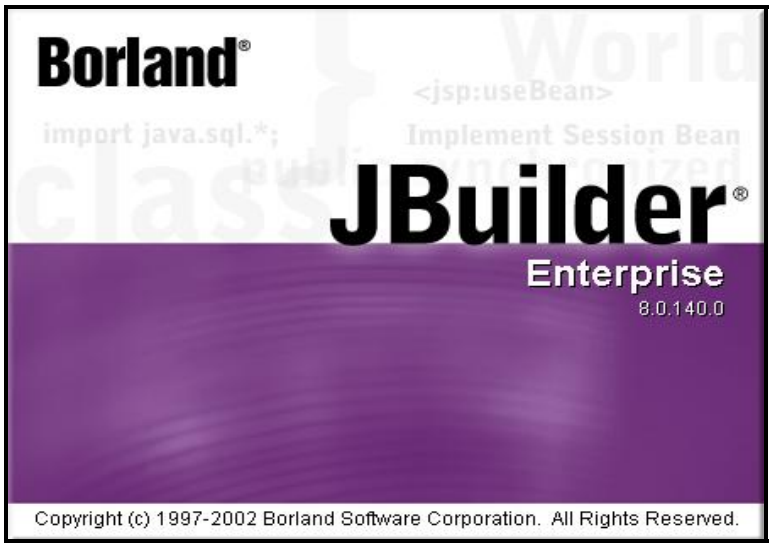


Figura 2.1: Versão do *JBuilder* utilizada no desenvolvimento da aplicação.

A aplicação foi desenvolvida com base nas seguintes condições físicas:

Servidor	
Computador	Portátil Airis Andromeda
CPU	Pentium(R) IV 2.50 GHz
RAM	512 MB
Disco rígido	30 GB
Ecrã	LCD 15 (res. 1024x768)
Placa de Rede	LAN 10/100 BaseT
Instalações	
Universidade	Sala de Projectos do DI
Outro Local	Casa

2.3 Linguagens de Programação

O primeiro desafio no início do projecto foi sem dúvida, optar pela melhor linguagem de programação.
Existem actualmente várias linguagens orientadas para o desenvolvimento de aplicações *Web* de conteúdo dinâmico, de seguida passo a enumerar alguns exemplos:

- PHP;
- Servlets;
- JSP;
- ASP.

As linguagens acima referidas têm em comum a possibilidade de suportar grande capacidade de armazenamento e processamento do lado do servidor (*Server Side*). Permitem ainda assegurar segurança e capacidade de manipulação de informações. No entanto, cada linguagem tem as suas performances o que como é evidente se compreende.
Para a escolha da linguagem a utilizar no desenvolvimento da aplicação, foram tomados em consideração vários critérios nomeadamente o facto de:

- Utilizar uma tecnologia relativamente recente;
- Permitir rápido desenvolvimento e fácil manipulação de informação no sistema;
- Permitir a Portabilidade de plataformas;
- Oferecer Robustez.

A escolha recaiu na linguagem JSP, não só porque preenche todos os requisitos atrás descritos, mas também porque comparativamente com qualquer uma das outras linguagens, embora como atrás foi dito todas têm as suas qualidades, foi aquela que a partida ofereceu mais garantias.

De seguida passo a enumerar algumas das razões que levaram a esta escolha, fundamentando assim a minha decisão.

- Uma das vantagens de JSP sobre ASP é o facto de se poder alterar a personalização de *tags*, através do uso de livrarias, enquanto que em ASP isso não é possível, outra vantagem é o facto de a parte dinâmica ser escrita em Java e não Visual Basic ou outra linguagem propriedade da *Microsoft* ($$);

- Embora a linguagem PHP seja mais fácil no desenvolvimento de pequenas aplicações para a Web, a medida que se passa para aplicações de maior dimensão, torna-se necessário o uso de linguagens com maior suporte à escalabilidade, como é o caso de Java (JSP);

- As páginas JSP ou Java Server Pages, foram criadas para contornar algumas limitações no desenvolvimento com *Servlets*. Por exemplo o facto de num *Servlet* a formatação da página HTML se misturar com a lógica da aplicação em si, dificulta a alteração dessa formatação. Em JSP a formatação esta separada da programação, podendo ser modificada sem afectar o resto da aplicação.

Mais a frente neste relatório, estes e outros pontos serão aclarados, nomeadamente a linguagem JSP e as suas características gerais.

Importante será também referir a importância da tecnologia Java no desenvolvimento de aplicações *Client Side*, *Server Side*, *Enterprise Solutions*, entre outras aplicações.

O JSP está inserido no pacote Java™ *Web* Services Developer Pack (Java WSDP) que abrange um conjunto de tecnologias *interface*s APIs que incorporam XML, JSP, JSTL, etc., com o intuito de simplificar o desenvolvimento de serviços orientados para a *Web* usando a plataforma *Java 2, Standard Edition v. 1.4*.

2.3.1 JSP

2.3.1.1 Definição

A tecnologia JSP ou Java Server Pages foi desenvolvida pela *Sun Microsystems* como resposta a aparição da linguagem ASP (Active server Pages) por parte da *Microsoft*. Porém tem a vantagem da portabilidade de plataforma da linguagem Java. Operando na camada de apresentação, as páginas JSP utilizam a tecnologia Java do lado do servidor para a criação do conteúdo dinâmico aliado com as *tags* HTML para manter o conteúdo estático.
Uma página JSP não é mais que uma página *Web* normal que contém porções (*tags*) especiais de código em Java juntamente com porções (*tags*) de código em HTML.
A extensão de um ficheiro em JSP é: `NomeDoFicheiro.jsp`.

2.3.1.2 Características Gerais

A tecnologia JSP oferece a vantagem de ser facilmente codificada, facilitando assim a elaboração e manutenção de uma aplicação *Web*. Além disso, permite a separação da programação lógica (parte dinâmica) da programação visual (parte estática). Outra característica importante é a possibilidade de produzir conteúdos dinâmicos que possam ser reutilizados.

Pode-se considerar as principais características do JSP como sendo:

- **Separação do conteúdo estático do dinâmico:**

  A tecnologia JSP permite separar a programação lógica (parte dinâmica) da programação visual (parte estática), facilitando o desenvolvimento de aplicações mais robustas, onde programador e *designer* podem trabalhar no mesmo projecto, mas de forma independente.

- **Independência da Plataforma:**

  Sendo a tecnologia JSP baseada na plataforma Java, as páginas JSP podem ser facilmente transportadas entre diferentes plataformas.

- **Diversidade de formatos:**

  Qualquer formato actual pode ser utilizado numa página JSP, além de HTML. Podem ser utilizadas linguagens XML, DHTML, entre outras.

- **Integração com outras API's Java:**

  Outra característica da tecnologia JSP é a possibilidade de integração com todas as API's Java já existentes e a utilização de componentes EJB (Enterprise JavaBean).

- **Utilização de código Java:**

  É possível utilizar os chamados *scriplets*, que não são mais que partes de código Java puro inseridos dentro da página JSP.

- **Open Source:**

  O JSP possui código aberto que permite contar com a participação de comunidades Java e JSP Internacionais na criação e aperfeiçoamento da tecnologia.

- **Robustez herdada do Java:**

  A tecnologia JSP usa a linguagem Java como base para sua linguagem de *scripts*, utilizando todo o potencial que esta lhe oferece, sendo exactamente por esse motivo que a tecnologia JSP se apresenta mais flexível e mais robusta que outras linguagens de *scripting*.

- **Possibilidade de criar *tags* personalizadas:**

  A criação de *tags* personalizadas também é possível, o que permite maior flexibilidade, podendo-se criar *tags* que sejam específicas para o sistema em causa.

- **Uso de *Servlets* provê uma melhor performance:**

  O aumento de performance deve-se ao facto de uma página JSP ser convertida num *Servlet* ao ser compilada e fica na memória do servidor. Ou seja, a aplicação é compilada do lado do servidor uma única vez. Quando uma página JSP é requisitada, o servidor procura por uma versão mais recente e faz a sua compilação, caso nenhuma versão mais recente seja encontrada, o servidor utiliza o *Servlet* já compilado. Isto produz uma significativa rapidez de capacidade de resposta aos pedidos efectuados ao sistema.

2.3.1.3 Arquitectura

A primeira vez que uma página JSP é carregada pelo *container* JSP, o código Java é compilado (não é uma linguagem interpretada mas sim compilada, o que torna a desempenho do sistema muito mais rápido) gerando um *Servlet,* que ao ser executado cria uma pagina HTML sendo esta enviada para o *browser*. As chamadas subsequentes são enviadas directamente ao *Servlet* que foi criado na primeira requisição, sem que seja necessário repetir as etapas de compilação e criação do *Servlet.*

A figura seguinte mostra um esquema das fases necessárias a execução de uma página JSP na primeira vez em que esta é requisitada.

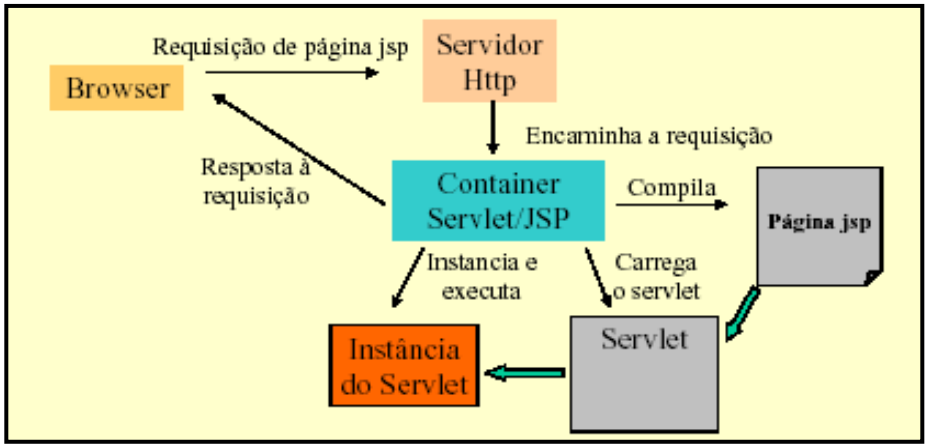


Figura 2.1: Fases da primeira execução de uma página JSP.

Numa primeira fase a requisição é enviada para o servidor *Web*, o qual por sua vez a reencaminha para o *container* Servlet/JSP. A fase seguinte consiste na verificação por parte do *container,* que não existe nenhuma instância de *Servlet* correspondente a página JSP. Neste caso, a página JSP é traduzida para código fonte, de uma classe *Servlet* que será usada na resposta a requisição.
Na próxima fase o código fonte do *Servlet* é compilado, criando uma instância da classe, por último é invocado o método `service()` da instância, que vai gerar a resposta à requisição.

Portanto um ficheiro JSP é compilado apenas na primeira vez que é chamado, ou seja, se o ficheiro JSP não for alterado, o servidor envia o *Servlet* resultante da última compilação.

2.3.1.4 Aprender JSP

Para se aprender JSP e assimilar os seus princípios básico é bastante importante ao utilizador, ter alguns conhecimentos da linguagem Java assim como de HTML, no meu caso especifico tive necessidade essencialmente de:

- Rever a linguagem Java;
- Aprimorar os conhecimentos em HTML (usado essencialmente no *design* da *interface*);

- Compreender o funcionamento de um servidor *Web;*
- Rever a sintaxe SQL;
- Compreender a estrutura e organização de uma aplicação *Web* em JSP.

Foi com base nos requisitos acima, que adquiri o *background* necessário, para avançar na programação em JSP e por conseguinte na realização da aplicação em causa.

2.3.2 JAVA

Para além do facto da linguagem Java ser uma constante na tecnologia JSP, na realização da aplicação foi necessário criar várias classes em Java, nomeadamente as classes:

- `Vertices.class`;
- `Arestas.class`;
- `Coordenas.class`;
- `Dados.class`;
- `Dijkstra.class`;
- `Grafo.class`;
- `Imagens.class`;
- `LatLong.class`;
- `Sessão.class`.

Que contêm métodos que são necessários para manipulação de informação na aplicação.

O JSP permite importar classes Java já compiladas e utilizar, livremente, os seus métodos.

Esta potencialidade permitiu que operações compostas por múltiplas instruções, frequentemente utilizadas na aplicação fossem traduzidas por métodos de classe.

A classe é, portanto, uma forma de poupar tempo e recursos.

2.4 Quadro Resumo do Software Utilizado

O quadro abaixo sintetiza as ferramentas necessárias para o desenvolvimento da aplicação.

Tipo	Nome	Descrição e Utilidade na Aplicação
Sistema Operativo	Microsoft Windows XP Professional	Sistema Operativo usado no desenvolvimento da aplicação.
SGBD	phpMyAdmin 2.5.3	Aplicativo gráfico que permite a gestão de bases de dados em MySQL. Utilizado para criação da base de dados da aplicação.
Ambiente integrado	*EasyPHP*	Pacote de soluções integradas para facilitar a instalação e configuração do servidor *web* Apache, base de dados MySQL e o PHPMyAdmin. O pacote instala todos os componentes e configura os vários componentes automaticamente.
Driver mysql para Java	mysql-connector-Java-3.1.0-alpha	*Driver* JDBC que permite estabelecer a ligação entre aplicação Java/JSP e a base de dados em MySQL.
Servidor *Web*	Apache	Servidor *Web* utilizado para estabelecer ligação à Internet.
Servidor *Web* Servlet/JSP *container*	jakarta-tomcat-5.0.25	Servidor *Web* que suporta a tecnologia JSP. Referência oficial da *Sun Microsystems* para implementação e desenvolvimento de aplicações em JSP. *Tomcat* é o resultado do projecto *Jakarta* do *Apache Software Foundation*.
Compilador Java	j2sdk1.4.2_02	*Kit* de Desenvolvimento de Software (SDK) *Java 2 Platform Standard Edition* (J2SE). Parte fundamental para o compilador Java funcionar correctamente no sistema.
Editor	NotePad	Editor de Texto usado para edição de todos os ficheiros JSP da Aplicação.
Editor	Borland JBuilder Enterprise 8.0.140.0	IDE usado para o desenvolvimento e compilação das classes Java da aplicação.
Browser	Internet Explorer 6	Browser que permite visualizar a *interface* da aplicação.

2.5 Conclusão

Com o presente capítulo pretendeu-se mostrar as condições em que o projecto foi desenvolvido.

Foram referidas as razões que levaram a escolha pelo JSP em detrimento das linguagens concorrentes.

Foi efectuado um breve enquadramento à linguagem, referindo as principais características, assim como a sua arquitectura.

Foi feito um levantamento do equipamento bem como as ferramentas de trabalho necessárias para o desenvolvimento da aplicação.

Portanto, o passo seguinte será abordar a base de dados que representa o conhecimento.

3. Base de Dados

3.1 Introdução

A aplicação recorre a uma base de dados em MySQL para suportar as transacções de informação a que a aplicação está sujeita.
Este capítulo apresenta o modelo da base de dados da aplicação: **basedados**.
É feita uma abordagem descritiva de cada relação e respectivas entidades que constituem o domínio da aplicação.

3.2 Modelo Entidade/Relacionamento

O modelo Entidade/Relacionamento tem como objectivo identificar as entidades e os relacionamentos entre estas numa dada organização, criando uma estrutura lógica de dados, independente da implementação física. As entidades podem ou não, estar relacionadas e têm os seus detalhes descritos como atributos.

A fugura seguinte motra o modelo Entidade/Relacionamento da aplicação.

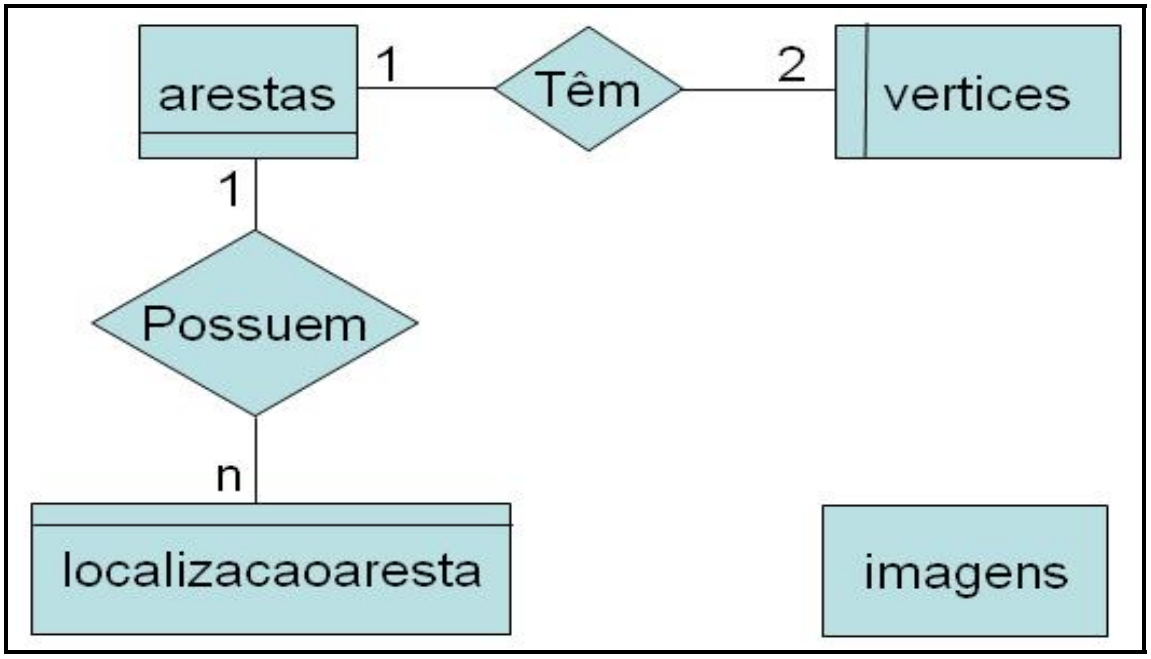


Figura 3.1: Modelo Entidade/Relacionamento da aplicação.

3.3 *Interface* Gráfico

Para uma melhor gestão e manipulação da base de dados, foi usado o SGBD *phpMyAdmin*. Que é um aplicação gráfica que permite a gestão de bases de dados em MySQL.
Na Figura 3.1 podemos ver essa aplicação.

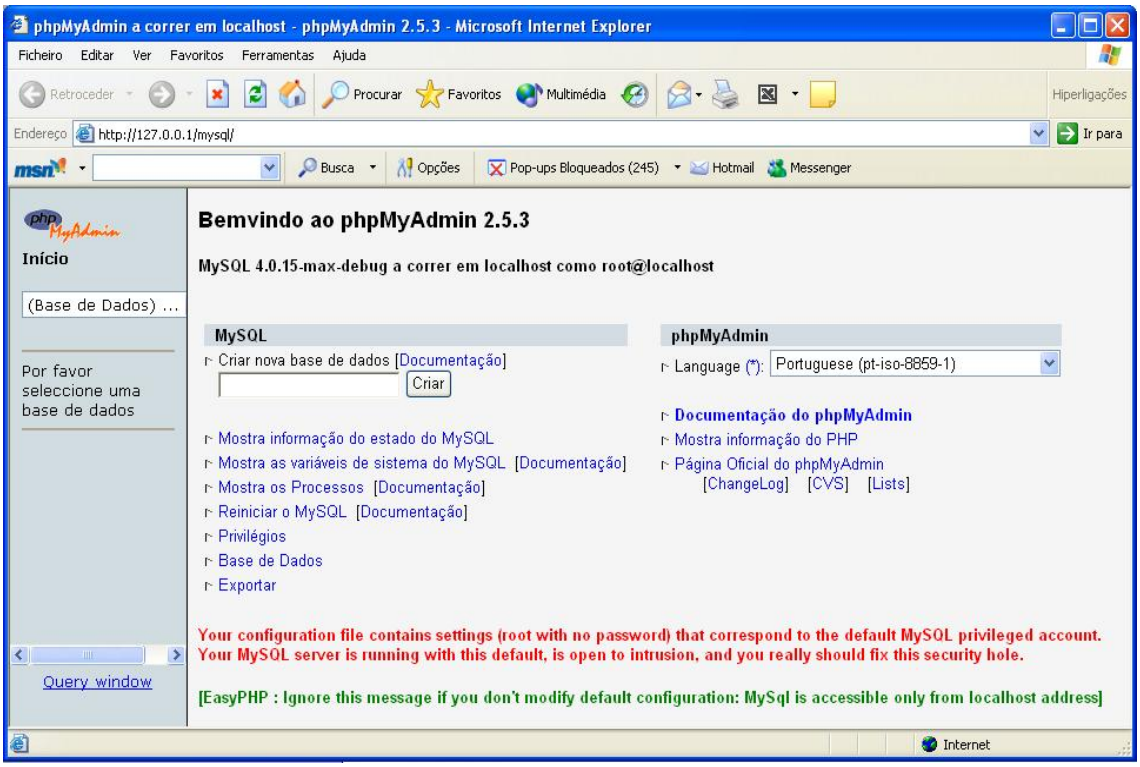


Figura 3.2: Versão do *phpMyAdmin* utilizada no desenvolvimento da aplicação.

Na figura abaixo podemos ver a base de dados da aplicação, que é denominada por **basedados**, a qual contêm 4 tabelas.

Figura 3.3: Lista das tabelas utilizadas na aplicação.

3.4 Modelo Físico

Para melhor perceber a utilidade de cada entidade da relação, é efectuada uma descrição de cada tabela da base de dados: **basedados**.

As figuras apresentadas em cada uma das relações foram retiradas do SGBD a partir do *phpMyAdmin* da tabela em causa.

Nesta descrição, é ainda mencionada a utilidade e o tipo de cada um das entidades da relação.

3.4.1 Relação *vertices*

Figura 3.4: Entidades da relação *vertices*.

A relação **vertices** permite armazenar os dados relativos aos diferentes vértices que constituem a rede viária.
De notar ainda que a intersecção de duas arestas é considerada um vértice, assim como os cruzamentos e os becos.

Entidade	Utilidade da Entidade	Tipo
idvertice	Identifica univocamente cada local. Chave primária.	Número inteiro
descricao	Nome do local.	Alfanumérico

3.4.2 Relação *arestas*

Figura 3.5: Entidades da relação *arestas*.

É na relação **arestas** que se encontra toda a informação respeitante as arestas que compõem a rede viária que serve de base a aplicação.

A ainda a realçar o facto de cada rua ter dois sentidos, portanto a aresta que vai do vértice A para o vértice B é necessariamente diferente da aresta que sai de B para A. Como é óbvio, as entidades ***descricao***, ***custo***, ***tempo*** e ***turismo*** são iguais para ambas as arestas.

Entidade	Utilidade da Entidade	Tipo
idaresta	Identifica univocamente cada rua. Chave primária.	Número inteiro
descricao	Nome da rua.	Alfanumérico
idverticepartida	Número que identifica o local de partida da aresta.	Número inteiro
idverticechegada	Número que identifica o local de chegada da aresta.	Número inteiro
custo	Distância da rua. Isto é, distância que separa o local de partida da aresta, do local de chegada da aresta.	Número real
tempo	Tempo necessário para percorrer a rua. Isto é, tempo que demora desde o local de partida da aresta até ao local de chegada da mesma.	Número real
turismo	Representa o grau de interesse que a rua tem ao nível do turismo. Para a atribuição desse grau são tomados em linha de custa determinados factores entre eles o número de casas comerciais, a existência ou não de monumentos, teatro, etc.	Número real

3.4.3 Relação *localizacaoaresta*

Figura 3.6: Entidades da relação *localizacaoaresta*.

A relação **localizacaoaresta** tem a informação respeitante as coordenadas (Latitude e Longitude) de cada um dos vertices que constituem a aresta, ou seja, é nesta relação que são guardadas as coordenadas do vértice de partida da aresta e do vértice de chegada da mesma.

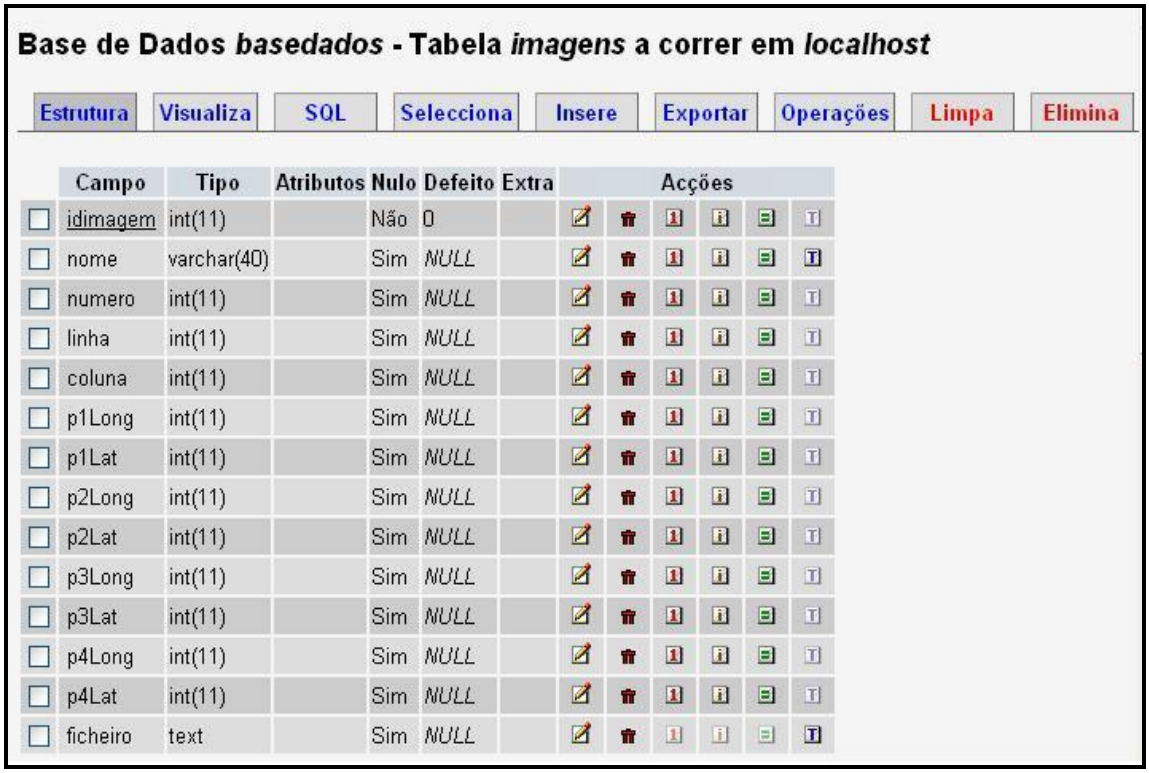
Por outro lado, como nem todas as arestas são apenas formadas por um segmento de recta (uma aresta pode ser constituída por vários segmentos de recta), ouve necessidade de distinguir as arestas que apenas são constituídas por uma recta, das outras arestas.

Para isso temos a entidade ***ordem*** que vai permitir saber em quantos segmentos de recta se divide a aresta, registando a informação respeitante a cada uma delas.

Por exemplo, se uma aresta for constituída por 3 segmentos então vão ser guardados na base de dados os três registos, obviamente com o mesmo valor para a entidade ***idaresta*** mas com valores distintos para as restantes entidades da relação.

Entidade	Utilidade da Entidade	Tipo
idaresta	Identifica univocamente cada rua. Chave primária.	Número inteiro
ordem	Identifica se uma aresta é constituída por um ou mais segmentos. Chave primária.	Número inteiro
longitude_1	Coordenada da longitude, do local de partida da aresta.	Número inteiro
latitude_1	Coordenada da latitude, do local de partida da aresta.	Número inteiro
longitude_2	Coordenada da longitude, do local de chegada da aresta.	Número inteiro
latitude_2	Coordenada da latitude, do local de chegada da aresta.	Número inteiro

3.4.4 Relação *imagens*



Base de Dados *basedados* - Tabela *imagens* a correr em *localhost*

Estrutura | Visualiza | SQL | Selecciona | Insere | Exportar | Operações | Limpa | Elimina

	Campo	Tipo	Atributos	Nulo	Defeito	Extra	Acções
☐	idimagem	int(11)		Não	0		
☐	nome	varchar(40)		Sim	*NULL*		
☐	numero	int(11)		Sim	*NULL*		
☐	linha	int(11)		Sim	*NULL*		
☐	coluna	int(11)		Sim	*NULL*		
☐	p1Long	int(11)		Sim	*NULL*		
☐	p1Lat	int(11)		Sim	*NULL*		
☐	p2Long	int(11)		Sim	*NULL*		
☐	p2Lat	int(11)		Sim	*NULL*		
☐	p3Long	int(11)		Sim	*NULL*		
☐	p3Lat	int(11)		Sim	*NULL*		
☐	p4Long	int(11)		Sim	*NULL*		
☐	p4Lat	int(11)		Sim	*NULL*		
☐	ficheiro	text		Sim	*NULL*		

Figura 3.7: Entidades da relação *imagens*.

A relação **imagens** tem a informação respeitante a cada uma das imagens, nomeadamente as coordenadas (Latitude e Longitude) de cada um dos cantos que constituem a imagem.

O ***nome*** representa a descrição da imagem, o ***numero*** como o próprio nome indica é o número atribuído a imagens (importante para definir as entidades ***linha*** e ***coluna***, como se verá abaixo) e ainda a entidade ***ficheiro*** que ao contrário do que se possa pensar, não guarda a imagem, mas sim o endereço do local onde ela está armazenada, no servidor.

As entidades ***linha*** e ***coluna*** servem para guardar a linha e coluna que a imagem representa numa matriz de imagens. Por exemplo para a imagem com o ***numero*** 321 o valor da ***linha*** será 2 e o valor da ***coluna*** será 1, estes dados são importantes para o desenvolvimento da aplicação como veremos no manual de utilização, que faz parte dos apêndices deste relatório.

Entidade	Utilidade da Entidade	Tipo
idimagem	Identifica univocamente cada imagem. Chave primária.	Número inteiro
nome	Nome atribuída a imagem.	Alfanumérico
numero	Representa o número atribuída a imagem.	Número inteiro
linha	Define a linha da imagem com base numa matriz.	Número inteiro
coluna	Define a coluna da imagem com base numa matriz.	Número inteiro
p1Long	Coordenada da longitude do canto superior esquerdo da imagem.	Número inteiro
p1Lat	Coordenada da latitude do canto superior esquerdo da imagem.	Número inteiro
p2Long	Coordenada da longitude do canto superior direito da imagem.	Número inteiro
p2Lat	Coordenada da latitude do canto superior direito da imagem.	Número inteiro
p3Long	Coordenada da longitude do canto inferior esquerdo da imagem.	Número inteiro
p3Lat	Coordenada da latitude do canto inferior esquerdo da imagem.	Número inteiro
p4Long	Coordenada da longitude do canto inferior direito da imagem.	Número inteiro
p4Lat	Coordenada da latitude do canto inferior direito da imagem.	Número inteiro
ficheiro	Guarda o endereço do local onde a imagem foi guardada.	Texto

3.5 Conclusão

Neste capítulo ouve uma preocupação especial em tentar descrever o melhor possível cada uma das relações, assim como abordar alguns aspectos considerados relevantes para sua melhor compreensão.

Considerar ao pormenor as tabelas da base de dados que compõem o domínio do conhecimento é sem dúvida, uma condição necessária para prosseguir para a programação propriamente dita.

Portanto, o passo seguinte será abordar o algoritmo que recebe este conhecimento e o transforma em resultados.

4. Algoritmo de Dijkstra

4.1 Introdução

O ***algoritmo de Dijkstra*** resolve o problema do caminho óptimo para o caso de custos ***não negativos***. Partindo do vértice inicial, o algoritmo vai escolher a aresta de menor custo para um vértice adjacente e continua assim até ter um caminho para todos os vértices com ligação.

É a partir da utilização do ***algoritmo de Dijkstra*** que se vai determinar o trajecto óptimo entre um local de origem e um local de destino, escolhidos pelo utilizador de entre os locais que compõem a rede viária.

4.2 Grafos

A teoria dos grafos tem contribuído para a análise de uma ampla variedade de problemas, nomeadamente, problemas de análise de caminho crítico, sistemas de comunicação, estudo de transmissão de informações, escolha de uma rota óptima, entre outros.

Um grafo pode ser visualizado através de uma representação geométrica, na qual os seus vértices correspondem a pontos distintos do plano em posições arbitrárias, enquanto que a cada aresta (*v*, *w*), é associada a uma linha arbitrária unindo os vertices correspondentes a *v* e *w*.

Um exemplo de grafo pode ser visto na figura 4.1.

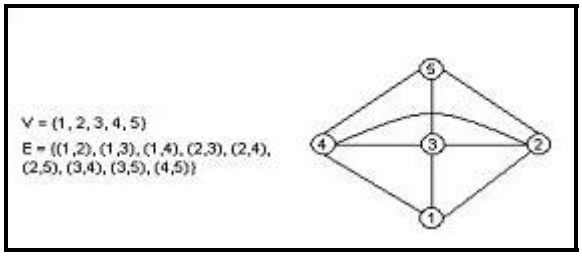


Figura 4.1: Grafo e sua representação geométrica.

Na figura podemos visualizar os conjuntos **V** e **E** que representam respectivamente o conjunto dos vertices e o conjunto das arestas, que formam o grafo.

4.3 Funcionamento do Algoritmo

Consideremos um conjunto **G**, que contém inicialmente apenas o vértice de partida (local de origem), que passamos a designar por ***s***. A qualquer momento **G** contém todos os vértices para os quais já foram determinados os menores caminhos, usando apenas vértices em **G** a partir de ***s***.

Então para cada vértice ***z*** não pertencente a **G**, mantemos a menor distância de ***s*** a ***z*** que representaremos por ***distancia[z]***, usando caminhos de modo que o único vértice que não está em **G** seja ***z***. É necessário também armazenar o vértice adjacente (predecessor) a ***z*** neste caminho, que representaremos por ***predecessor[z]***.

Questão:

Como fazer com que **G** *cresça, ou seja, qual vértice deve ser incluído em* **G** *a seguir?*

Fácil, tomamos entre todos os vértices que ainda não pertencem a **G**, aquele que tiver menor valor ***distancia***. Acrescentamos então esse vértice a **G** e chamemo-lo de ***vértice corrente***, de seguida recalculamos as distâncias (***distancia***) para todos os vértices adjacentes a ele que não estejam em **G**. Pois pode existir um caminho menor a partir de ***s***, passando pelo ***vértice corrente***, do que aquele que existia antes de ***vértice corrente*** ser agregado a **G**.

Caso exista um caminho mais curto precisamos actualizar o valor de ***predecessor[z]*** de forma a indicar que ***vértice corrente*** é o vértice adjacente a ***z*** pelo novo trajecto óptimo.

O algoritmo termina quando todos os vértices pertencem a **G**.

4.4 Exemplo de Utilização do Algoritmo

Vamos de seguida, mostrar um exemplo de utilização do ***algoritmo de Dijkstra*** usando para isso, um conjunto **V** dos vertices que constituem o grafo e um conjunto **A** das arestas, formados pelos seguintes elementos:

- **V** = {s,u,x,v,y};
- **A** = {(s,u,10),(s,x,5),(u,x,2),(u,v,1),(x,u,3),(x,v,9),(x,y,2),(v,y,4),(y,v,6),(y,s,7)}

Onde (s,u,10) representa uma aresta que liga o vértice s ao vértice u com custo 10.

Neste exemplo, serão utilizadas as designações usadas para a descrição do funcionamento do algoritmo (ponto 4.3 deste relatório).

Cada passo do algoritmo será apresentado por uma descrição, uma imagens ilustrativa e ainda um quadro.

➲ **Passo 1:**

Define-se inicialmente o vértice de origem (raiz), neste caso ***s***, e inclui-se este vértice em **G**. Atribui-se zero a sua distância (***distancia[s]***) porque o custo de ir de ***s*** a ***s*** é obviamente 0. Todos os outros vértices ***z*** têm as respectivas distâncias (***distancia[z]***) inicializadas com um valor bastante grande (∞).

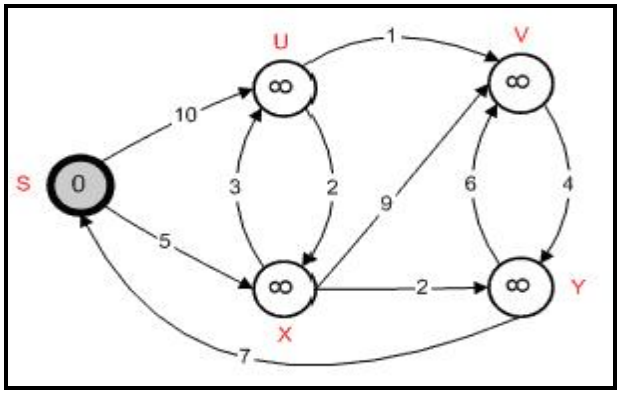


Figura 4.2: Exemplo de utilização do algoritmo-passo 1.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Não	∞	-
x	Não	∞	-
v	Não	∞	-
y	Não	∞	-

➲ **Passo 2:**

A partir de ***s*** consulta-se os vértices adjacentes a ele, que são ***u*** e ***x***. Para todos os vértices adjacentes, que chamaremos ***z***, calcula-se:

```
SE distancia[z] > distancia[s] + custo(s,z)
      distancia[z] = distancia[s] + custo(s,z)
      predecessor[z] = s
FIM SE
```

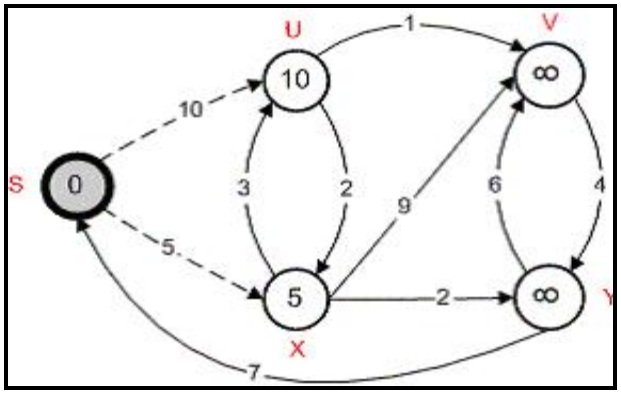


Figura 4.3: Exemplo de utilização do algoritmo-passo 2.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Não	10	s
x	Não	5	s
v	Não	∞	-
y	Não	∞	-

➲ **Passo 3:**

De entre todos os vértices não pertencentes a **G** escolhe-se aquele com a menor distância. Neste caso é o vértice ***x***, pois ***distancia[x]*** = 5.

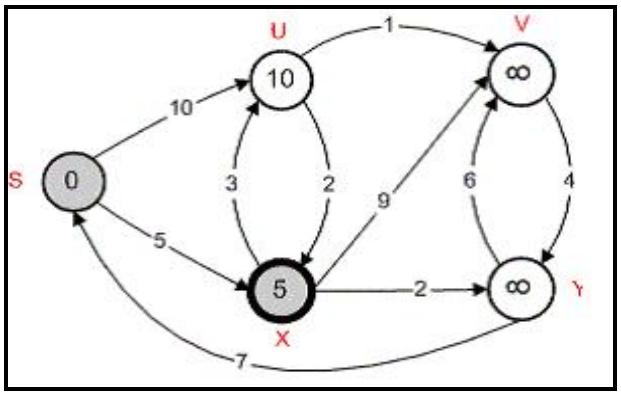


Figura 4.4: Exemplo de utilização do algoritmo-passo 3.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Não	10	s
x	Não	5	s
v	Não	∞	-
y	Não	∞	-

➲ **Passo 4:**

Inclui-se ***x*** em **G**. A partir de ***x*** consultam-se os vértices adjacentes a ele que não estão em **G**, que são ***u***, ***v*** e ***y***. Para todos os vértices adjacentes, que chamaremos *z*, calcula-se:

```
SE distancia[z] > distancia[x] + custo(x,z)
      distancia[z] = distancia[x] + custo(x,z)
      predecessor[z] = x
FIM SE
```

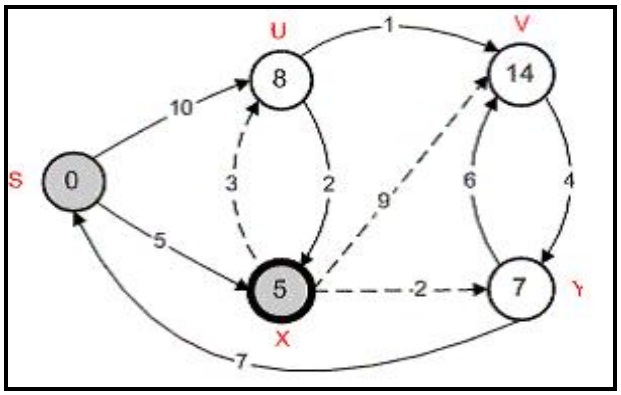


Figura 4.5: Exemplo de utilização do algoritmo-passo 4.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Não	8	x
x	Sim	5	s
v	Não	14	x
y	Não	7	x

➲ **Passo 5:**

De entre todos os vértices não pertencentes a **G** escolhe-se aquele com a menor distância. Neste caso é o vértice ***y***, pois ***distancia[y]*** = 7.

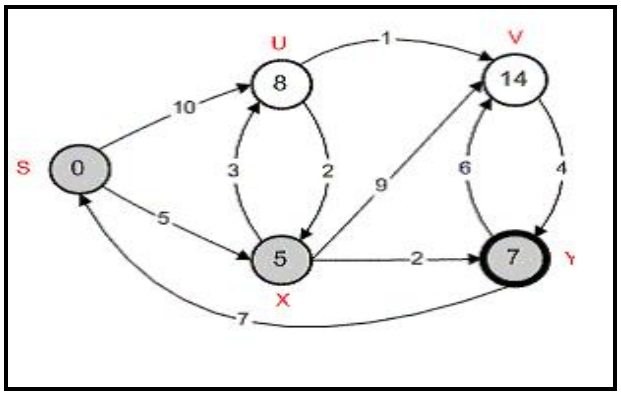


Figura 4.6: Exemplo de utilização do algoritmo-passo 5.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Não	8	x
x	Sim	5	s
v	Não	14	x
y	Não	7	x

➲ **Passo 6:**

Inclui-se ***y*** em **G**. A partir de ***y*** consultam-se os vértices adjacentes a ele que não estão em **G**, que é apenas o vértice ***v***. Para o vértice *v*, calcula-se:

```
SE distancia[v] > distancia[y] + custo(y,v)
      distancia[v] = distancia[y] + custo(y,v)
      predecessor[v] = y
FIM SE
```

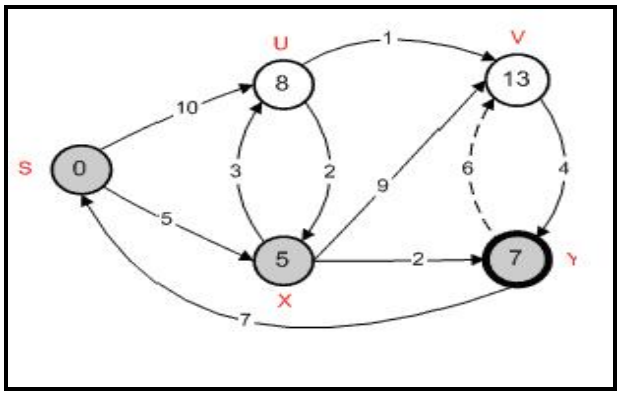


Figura 4.7: Exemplo de utilização do algoritmo-passo 6.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Não	8	x
x	Sim	5	s
v	Não	13	y
y	Sim	7	x

➲ **Passo 7:**

De entre todos os vértices não pertencentes a **G** escolhe-se aquele com a menor distância. Neste caso é o vértice ***u***, pois ***distancia[u]*** = 8.

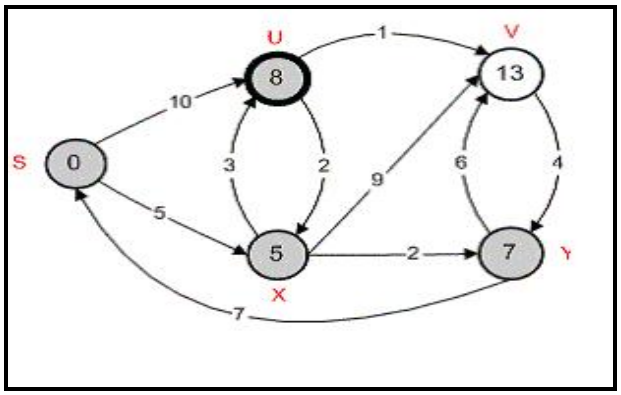


Figura 4.8: Exemplo de utilização do algoritmo-passo 7.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Não	8	x
x	Sim	5	s
v	Não	13	y
y	Sim	7	x

➲ **Passo 8:**

Inclui-se ***u*** em **G**. A partir de ***u*** consultam-se os vértices adjacentes a ele que não estão em **G**, que é apenas o vértice ***v***. Para o vértice *v*, calcula-se:

```
SE distancia[v] > distancia[u] + custo(u,v)
      distancia[v] = distancia[u] + custo(u,v)
      predecessor[v] = u
FIM SE
```

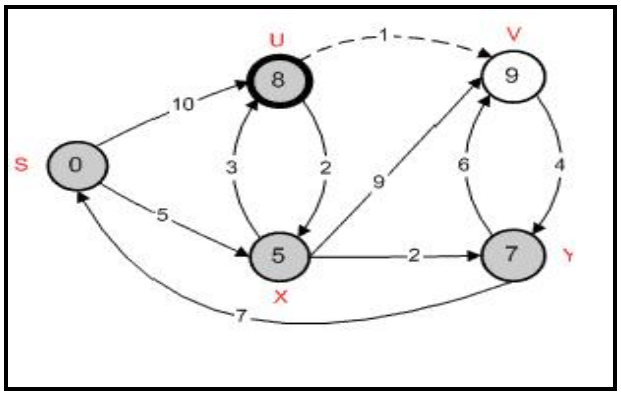


Figura 4.9: Exemplo de utilização do algoritmo-passo 8.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Sim	8	x
x	Sim	5	s
v	Não	9	u
y	Sim	7	x

➲ **Passo 9:**

De entre todos os vértices não pertencentes a **G** escolhe-se aquele com a menor distância. Neste caso ***v*** é o único vértice que resta e ***distancia[v]*** = 9.

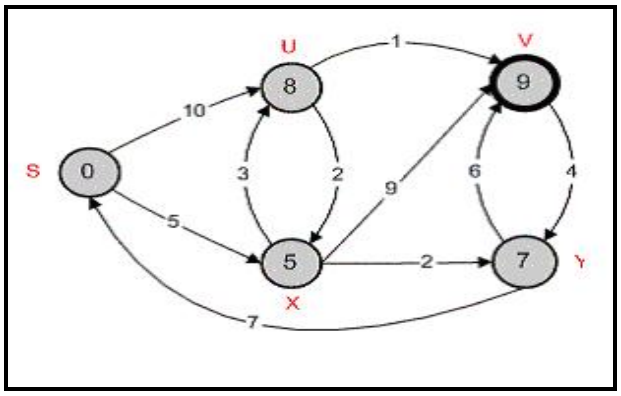


Figura 4.10: Exemplo de utilização do algoritmo-passo 9.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Sim	8	x
x	Sim	5	s
v	Não	9	u
y	Sim	7	x

➲ **Passo 10:**

Por fim faz-se ***v*** pertencer a **G**. Neste ponto, todos os vértices já estão em **G** acabando assim a execução do algoritmo.

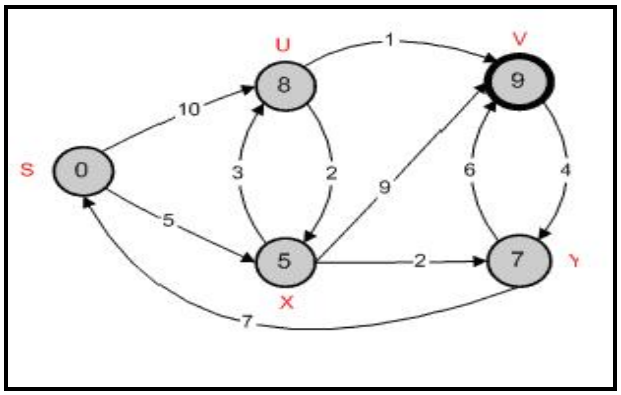


Figura 4.11: Exemplo de utilização do algoritmo-passo 10.

Vértice	Conjunto G	Distância	Predecessor
s	Sim	0	-
u	Sim	8	x
x	Sim	5	s
v	Sim	9	u
y	Sim	7	x

4.5 Conclusão

Neste capítulo ouve uma preocupação em tentar fazer a melhor descrição do funcionamento do ***algoritmo de Dijkstra***, devido a importância que este tem na implementação da aplicação. A aplicação funciona com base nos resultados fornecidos pela execução do algoritmo.

5. Sistema SLOG

5.1 Introdução

O “SLOG” é um Sistema de Localização e Orientação Geográfica, que permite aos utilizadores saber qual o trajecto óptimo entre dois quaisquer locais de uma determinada região.

Ao utilizador apenas lhe é pedido o local de onde quer partir assim como o local de destino (obviamente diferentes!!!), a partir dai, a aplicação mostra todo o percurso, nomeadamente os locais e as ruas por onde tem de passar desde o local de partida até ao destino e ainda o tempo necessário para percorrer esse caminho, bem como o grau de interesse turístico do percurso.

5.2 Arquitectura do Sistema

A arquitectura base de uma aplicação *Web* desenvolvida a partir da tecnologia JSP é diferente das restantes tecnologias concorrentes, como já foi visto anteriormente (capítulo 2).

Relativamente ao sistema SLOG, a arquitectura base é a seguinte:

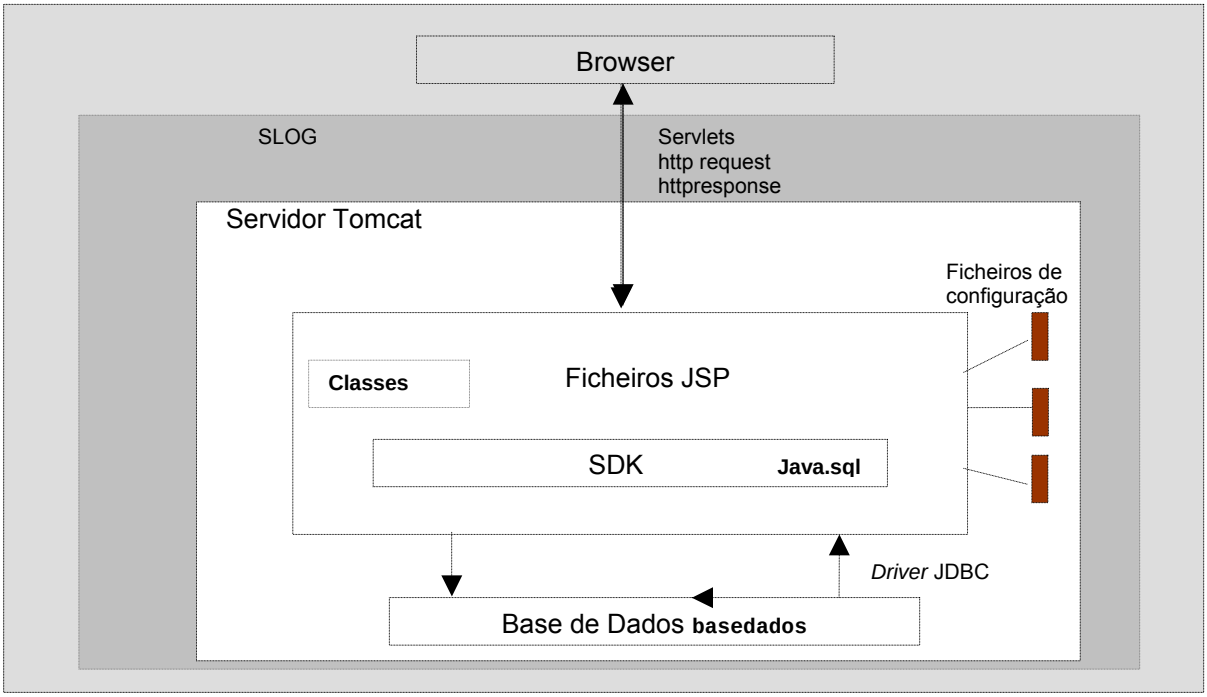


Figura 5.1: Arquitectura base do SLOG.

- Possui uma base de dados, denominada **base de dados** onde armazena toda a informação do sistema, e a partir da qual recebe toda ao informação que necessita para o normal funcionamento da aplicação;

- A manipulação e acesso à base de dados é da responsabilidade dos ficheiros JSP que:
  - ✓ Importam o *package* `Java.sql.*`;
  - ✓ Recolhem dos ficheiros de configuração a informação relativa ao *driver* JDBC e da base de dados;
  - ✓ Efectuam a ligação à base de dados;
  - ✓ Realizam as operações necessárias e por último, encerram a ligação.

- Os ficheiros que suportam as respostas dos pedidos são compilados no servidor *Web* Tomcat através do SDK e o resultado é enviado para o Browser.

5.3 Estrutura do Sistema

O sistema obedece à seguinte estrutura:

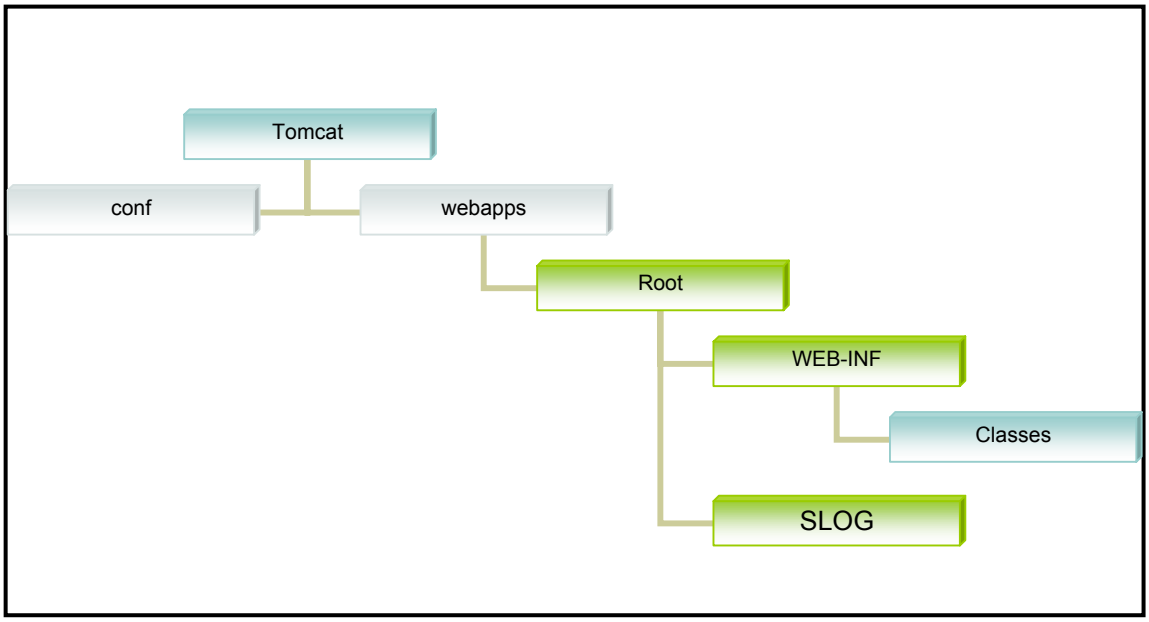


Figura 5.2: Estrutura do Sistema SLOG.

A directoria **SLOG** é a principal directoria da estrutura. É nesta directoria que esta grande parte da informação que diz respeito a aplicação, é onde estão todos os ficheiros JSP criados durante o seu desenvolvimento. A directoria **SLOG** contém ainda imagens e ícones que são usadas na aplicação.

A directoria **Classes** tem o pacote que contém todas as classes Java criadas durante o desenvolvimento da aplicação, e ainda outras funcionalidades importantes para o seu bom funcionamento.

Na directoria **conf** estão os ficheiros de configuração do sistema, responsáveis pelo bom funcionamento da estrutura.

5.4 *Interface* – SLOG

Nesta secção serão mostradas algumas imagens da aplicação.

5.4.1 Página Inicial do Sistema SLOG

Esta é a página de entrada da aplicação, onde se pode observar uma breve descrição da aplicação e das suas funcionalidades. É ainda nesta primeira página que o utilizador selecciona os locais de **Origem** e **Destino**.

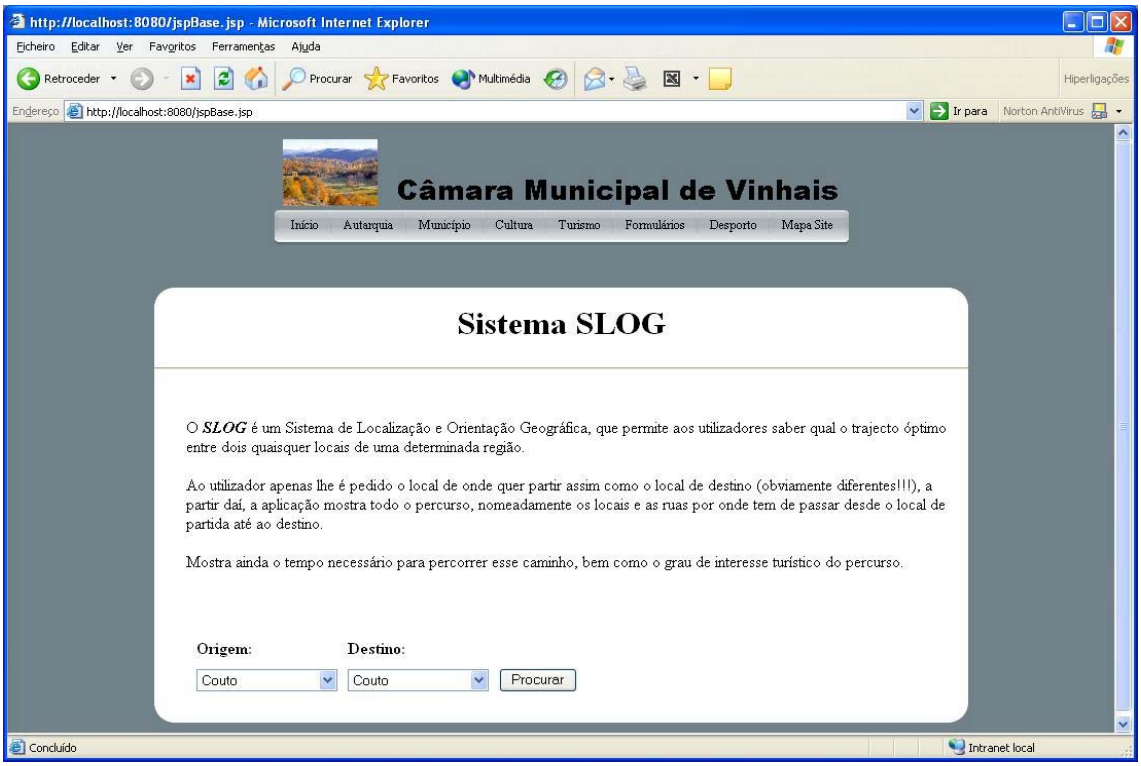


Figura 5.3: página inicial do SLOG.

5.4.2 Página de Resultados do Sistema SLOG

Esta é a página da aplicação, que mostra a imagem que contém os locais e as ruas por onde se tem de passar para, pelo trajecto óptimo chegar do local de **Origem** ao local de **Destino**.

Podemos ainda observar nesta página uma tabela com diversas informações relacionadas com o percurso. A descrição destas informações e das diversas funcionalidades desta página serão abordadas no manual de utilização, que faz parte dos apêndices deste relatório.

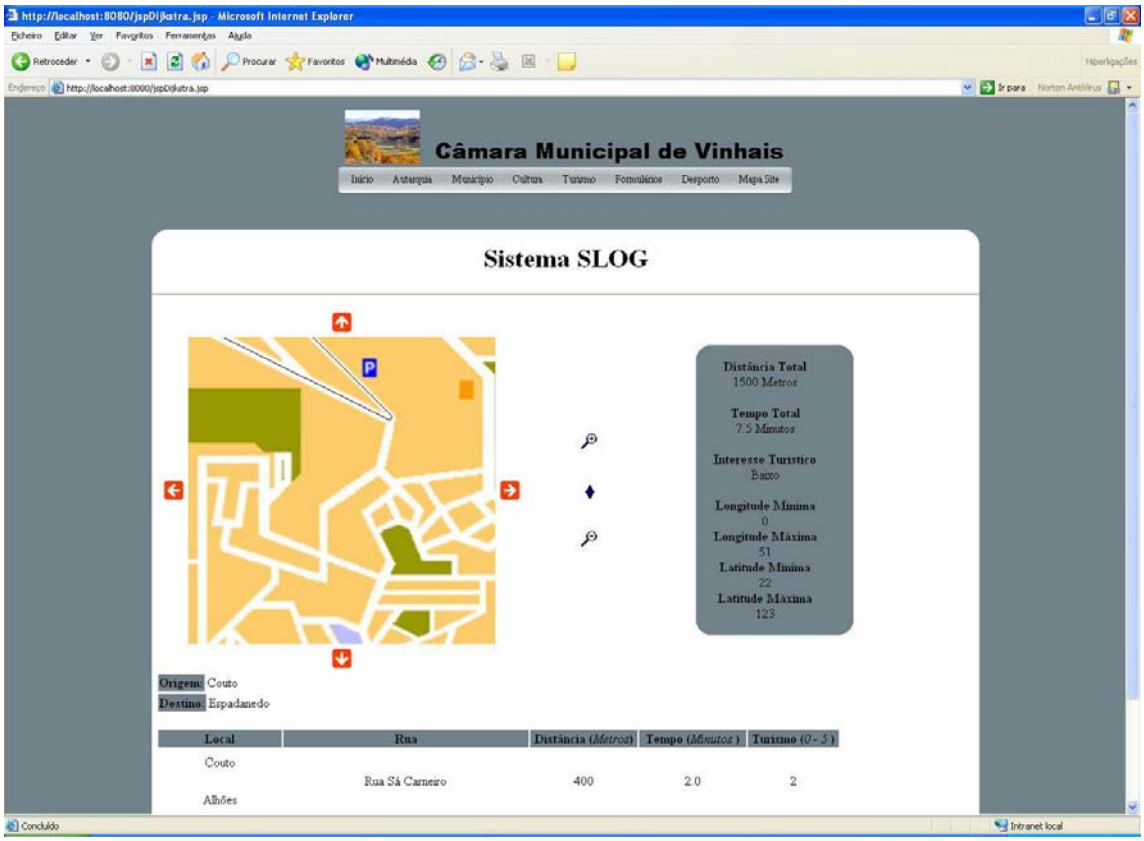


Figura 5.4: Página de resultados do SLOG.

5.5 Conclusão

O objectivo do presente capítulo é de certa forma, elucidar sobre o funcionamento e a arquitectura da aplicação, bem como as características inerentes ao seu desenvolvimento.

O manual de utilização, que faz parte dos apêndices deste relatório fornece informação mais detalhada sobre a utilização da aplicação.

6. Conclusões

6.1 Conclusão

Após o término deste grande desafio que delimita a conclusão da vida académica, posso afirmar que me sinto melhor preparado a todos os níveis para os desafios que se avizinham. O desenvolvimento do projecto nos últimos 5 meses foi sem dúvida um período de Aprendizagem a 100%.

Após a realização de um projecto como este, de viver todas as experiências que até à data me eram estranhas, tais como, o sentido de grande responsabilidade e de estruturação e organização de um plano de trabalho, posso dizer que esta experiência foi muito marcante e enriquecedora.

A experiência de trabalhar no desenvolvimento de páginas *Web* dinâmicas foi muito boa, gostei particularmente de aprender a trabalhar em JSP, que é uma linguagem muito poderosa e promissora, que se está a impor no mercado de desenvolvimento de aplicações orientadas para a *Web*.

O facto do projecto movimentar vários conceitos tais como: Bases de Dados, Algoritmos, Programação *Web*, entre outros, permitiu aplicar e relembrar conceitos apreendidos anteriormente em disciplinas desta licenciatura.

Com o passar do tempo a aplicação foi ganhando forma, onde cada obstáculo ultrapassado deixava antever um obstáculo ainda maior que o anterior. Foram meses de trabalho, onde a capacidade de organização, estruturação, análise e resolução de problemas foi grande.

Relativamente a aplicação, o nível de funcionalidade é satisfatório, preenchendo os requisitos que foram pedidos. A sua estética está directamente relacionada com a estética apresentada pelo site da câmara de Vinhais (projecto desenvolvido pelo meu colega Hélder Pires), uma vez que é parte integrante deste site.

Em suma, a aplicação é uma realidade e funciona, existindo no entanto, a possibilidade de melhorar as suas potencialidades à medida que surjam novas necessidades.

6.2 Melhoramentos

O primeiro melhoramento que sugeria que se fizesse na aplicação, era melhorar o sistema de marcação do percurso na imagem, isto é, marcar nas imagens os locais e as ruas que pertencessem ao percurso. Embora tenha iniciado a realização desta tarefa a falta de tempo limitou esse meu objectivo.

Outros melhoramentos poderão ser realizados o que tornaria a aplicação ainda mais interessante. Um deles é o *zoom*, que na aplicação é constituído por três níveis, aumentar o número de níveis seria uma mais valia. Outro seria dar a possibilidade ao utilizador de escolher qual a função de custo para determinar o caminho óptimo (distância, tempo ou interesse turístico).

Dotar a base de dados com mais informação tornaria a aplicação mais abrangente.

Bibliografia

[Temple, 04] André Temple – Jsp, Servlets e J2EE, Janeiro 2004 – Brasil

[Coelho, 99] Pedro Coelho – Programação em Java2, Março 1999 – Editora FCA

[Damas, 99] Luís Damas – SQL 4º Edição, Setembro 1999 – Editora FCA

[Coelho, 01] Pedro Coelho – HTML4 & XHTML, Março 2001 – Editora FCA

Referências da Internet

Nesta secção é apresentada uma compilação das referências retiradas da Internet.

➲ JSP

URL	Data Visita
http://java.sun.com/products/jsp/	Março 2004
http://www.visualbuilder.com/jsp/tutorial/	Março 2004
http://www.jspbrasil.com.br/jsp/tutoriais/tutorial.jsp	Março 2004
http://www.dca.fee.unicamp.br/~leandro/java/jspjava.html	Abril 2004
http://www.jdance.com/jspjavabeans.shtm	Abril 2004

➲ JDBC

URL	Data Visita
http://www.mysql.com/products/connector-j/index.html	Abril 2004

➲ JAVA

URL	Data Visita
http://java.sun.com/	Março 2004
http://java.sun.com/docs/books/tutorial/	Março 2004
http://www.terravista.pt/enseada/1624/tutojava.htm	Março 2004
http://www.portaljava.com/	Março 2004

➲ SQL

URL	Data Visita
http://www.mysql.com/	Abril 2004
http://www.mysql.com/documentation/	Maio 2004
http://www.mysql.com/doc/pt//index.html	Maio 2004

➲ Tomcat

URL	Data Visita
http://jakarta.apache.org/tomcat/	Abril 2004
http://jakarta.apache.org/tomcat/tomcat-5.0-doc/index.html	Abril 2004
http://www.mhavila.com.br/topicos/tomcat.html	Abril 2004

➲ Algoritmo Dijkstra

URL	Data Visita
http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dijkstra.html	Março 2004
http://renaud.waldura.com/doc/java/dijkstra/	Março 2004
http://rollerjm.free.fr/pro/graphs.html	Março 2004

➲ Outras Referências

URL	Data Visita
http://www.viamichelin.com/	Março 2004
http://www.geocities.com/marcoSchmidt.geo/java-highlight.html	Julho 2004

Apêndices

Instalação do Sistema

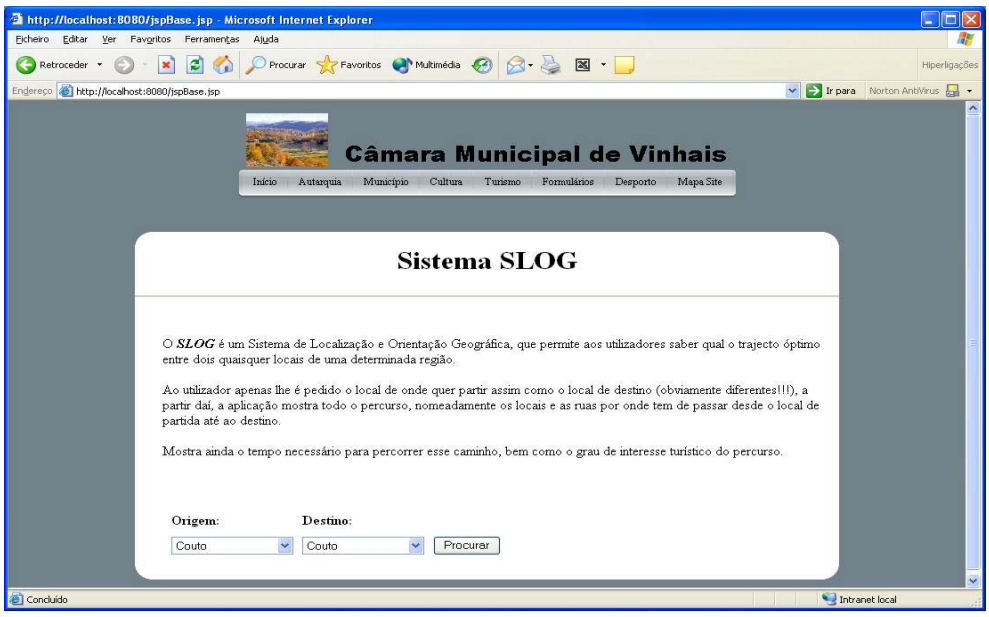
O processo de instalação refere-se apenas ao sistema operativo Windows XP. Contudo, como já foi referido no relatório de projecto, o sistema, tendo sido desenvolvido em JSP, é independente da plataforma.

A instalação do sistema consiste na passagem das pastas que contêm a aplicação para uma directoria própria, onde estas estão estruturadas de forma a permitir o bom funcionamento do sistema.

Na directoria **`C:\tomcat\webapps\ROOT\`** cria-se a pasta **SLOG** que vai conter todo o conteúdo da aplicação.

Para a instalação das classes da aplicação, na directoria **`C:\tomcat\webapps\ROOT\WEB-INF`** cria-se a pasta **classes**, depois copia-se o pacote **projecto** para dentro desta pasta.

Para testar o funcionamento da aplicação, basta digitar no browser localmente **`http://localhost:8080/SLOG/jspBase.jsp`.**
O sistema SLOG é agora visualizado.



Manual de Utilização

O objectivo deste manual é servir de orientação para os utilizadores do sistema "SLOG". Embora o próprio sistema seja bastante intuitivo, é necessário um documento que descreva cada funcionalidade da aplicação.

➲ Elementos Gráficos

A seguir, será feita a descrição de cada elemento gráfico da aplicação (botões, ícones, imagens, etc.) de forma a permitir a sua melhor compreensão.

Nesta imagem podemos ver as caixas de selecção que permitem seleccionar o local de **Origem** e o local de **Destino** que se pretende.

O botão **Procurar** quando premido, determina o percurso entre o locais de **Origem** e de **Destino** seleccionados.

Ícone responsável pelo deslocamento da imagem para cima.

Ícone responsável pelo deslocamento da imagem para baixo.

Ícone responsável pelo deslocamento da imagem para a esquerda.

Ícone responsável pelo deslocamento da imagem para a direita.

Imagem que mostra os locais e as ruas que constituem o percurso.

Permite aumentar o nível de *zoom*.

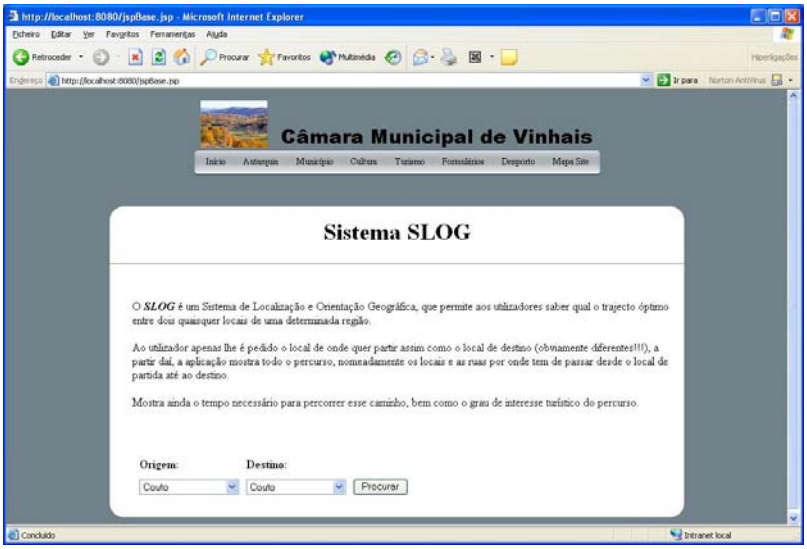
Permite diminuir o nível de *zoom*.

Indicador do nível de *zoom*.

➲ Funcionalidades da Aplicação

Esta secção apresenta uma descrição pormenorizada da aplicação, permitindo assim ao utilizador conhecer o melhor e mais rapidamente possível o seu modo de funcionamento.

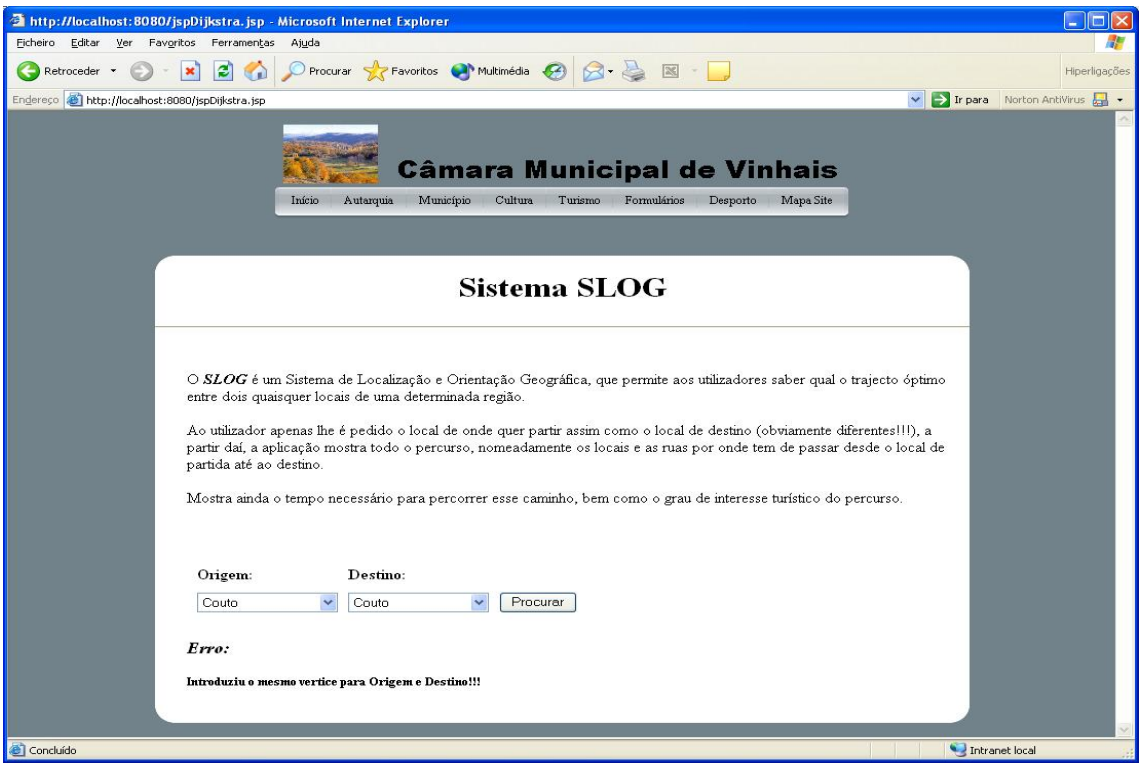
A página inicial da aplicação apresenta o seguinte *interface*:



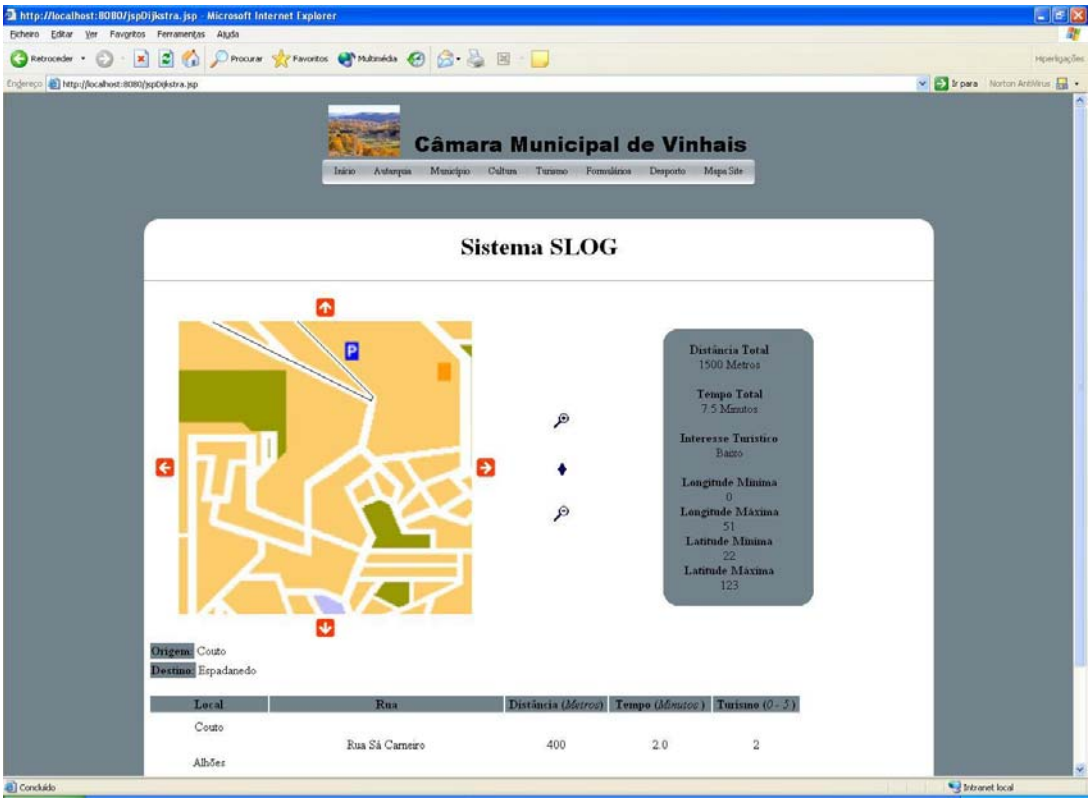
A página começa com uma descrição da aplicação. Abaixo dessa descrição temos duas caixas de selecção, uma para seleccionar o local de **Origem** e outra para o local de **Destino**. Após a selecção ter sido efectuada accionamos o botão **Procurar**, o qual vai permitir determinar o percurso óptimo entre a **Origem** e o **Destino** e apresentar os resultados noutra página.

Observação:

Caso o utilizador seleccione o mesmo local para a **Origem** e **Destino** é accionada uma mensagem de erro a dizer que, introduziu o mesmo local para **Origem** e **Destino** obrigando assim a nova selecção. Na imagem abaixo podemos ver esta situação.



Como atrás foi referido os resultados são apresentados numa outra página, a qual tem a seguinte aparência.



O início da página mostra uma imagem, rodeada de quatro ícones, que contém os locais e ruas que constituem o percurso óptimo entre os locais seleccionados anteriormente.

Olhando para a imagem em questão não dá para perceber, mas na realidade não é apenas uma imagem que ocupa aquele quadrado, mas sim uma matriz quadrada (2*2) de imagens, ou seja, a imagem que observamos não é mais do que a junção de quatro imagens, dando a sensação de ser apenas uma.

No capítulo 3 quando fazia a descrição da relação ***imagens*** disse: *"As entidades **linha** e **coluna** servem para guardar a linha e coluna que a imagem representa numa matriz de imagens."*, de seguida passo a explicar as razões que levam a tal utilização.

A rede viária utilizada na realização desta aplicação foi formada a partir de uma imagem da região de Vinhais, a partir dessa imagem foram criados 3 níveis de *zoom*, isto é, partiu-se a imagem inicial em quatro partes iguais, dando origem ás quatro imagens que formam o nível um do *zoom*. O segundo nível formou-se da mesma forma que o nível um, partindo cada uma das imagens do nível um em quatro, portanto o nível dois tem 16 imagens. O terceiro nível formou-se usando a mesma técnica dos níveis anteriores, dando origem a 64 imagens.

Portanto o nível um tem 4 imagens (pode-se representar como uma matriz quadrada 2*2), o dois tem 16 (matriz quadrada 4*4) e o três tem 64 (matriz quadrada 8*8), assim haveria que encontrar uma forma de apresentar a imagem. É aqui que entra a matriz quadrada (2*2), primeiro porque a imagem a apresentar será constituída no mínimo por 4 imagens (nível 1), e depois porque como uma matriz é formada por linhas e colunas vai permitir o que nos desloquemos pela matriz e consequentemente pelas imagens.

Após esta explicação para a utilização de uma matriz quadrada (2*2) na apresentação das imagens, voltemos a descrição da aplicação.

Falta no entanto especificar o método que permite definir as imagens que aparecem inicialmente, após cada utilização da aplicação. Este método funciona de seguinte maneira, assim que o **algoritmo de Dijkstra** determina o caminho óptimo entre os locais previamente seleccionados, já nos é possível saber em que nível de *zoom* podemos mostrar as imagens. Como também sabemos entre que coordenadas (Latitude e Longitude) se encontra o caminho óptimo, basta encontrar as imagens do nível determinado anteriormente, que contêm estas coordenadas e depois mostrar as imagens.

Uma vez explicada a forma como as imagens iniciais são escolhidas, prosseguindo com a descrição das funcionalidades, temos os ícones que rodeiam a imagem (quando falo em imagem refiro-me a matriz de imagens que constitui a imagem apresentada), um em cada um dos quatro lados da imagem. Estes ícones permitem que quando premidos a imagens se desloque na direcção escolhida.
Por exemplo, se accionarmos o ícone a direita da imagem, o valor da coluna é incrementado numa unidade o que implica que a imagem se desloque para a direita. Da mesma forma para os restantes ícones que rodeiam a imagem. Os ícones deixam de estar disponíveis (desaparecem) quando já não é possível o seu deslocamento numa determinada direcção.

Ao lado direito da imagem e respectivo ícone que tenho vindo a falar temos o **zoom**, que permite não só saber o nível de **zoom** da aplicação, mas também aumentar ou diminuir esse nível. Se aumentarmos o nível de **zoom** a aplicação mostra com maior detalhe a zona seleccionada e claro se o diminuirmos acontece precisamente o contrário.

Ainda na aplicação, a direita do **zoom** mas numa outra tabela temos a distância total do percurso, o tempo necessário a realização do mesmo e uma estimativa do seu grau de interesse turístico. As últimas informações fornecidas dizem respeito aos valores mínimos e máximos de longitude e latitude do percurso.

Por fim, vemos abaixo da imagem e respectivo ícone, uma tabela com diversas informações relacionadas com o percurso entre os locais de origem e destino.
A tabela começa por mostrar os locais escolhidos anteriormente como **Origem** e **Destino** do percurso. Depois mais abaixo mostra passo a passo os locais, ruas, tempo necessário para percorrer cada uma das ruas bem como o grau de interesse turístico de cada uma delas, tudo isto em relação ao percurso. Por exemplo, *vai do Couto para Espadanedo pela rua Sá Carneiro a qual tem uma distância de 400 metros e demora aproximadamente 2 minuto a percorrer. O interesse turístico desta rua é de grau 2.*

Espero com esta descrição da aplicação ter ajudado a uma melhor compreensão das suas funcionalidades.

Exemplo de Execução do Sistema

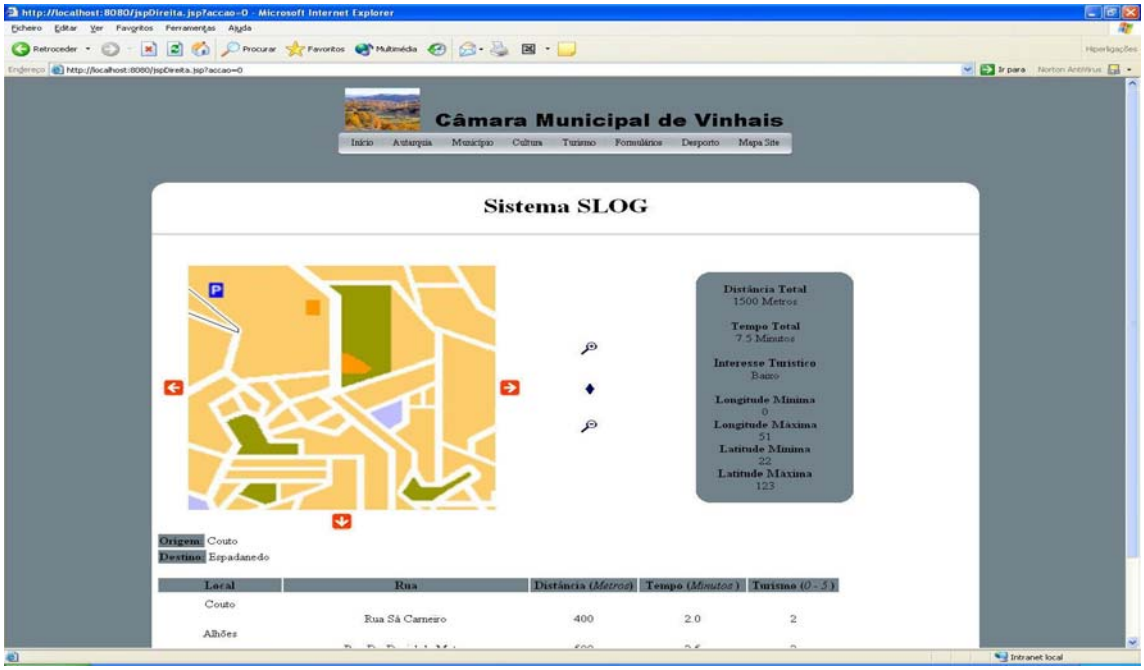
O primeiro passo ao utilizar esta aplicação é a escolha de um local de partida e de um local de chegada, que têm que ser diferentes. Na imagem seguinte podemos visualizar um exemplo deste primeiro passo.

A imagem seguinte mostra a janela da aplicação com os resultados obtidos com base nos locais de origem e destino seleccionados.

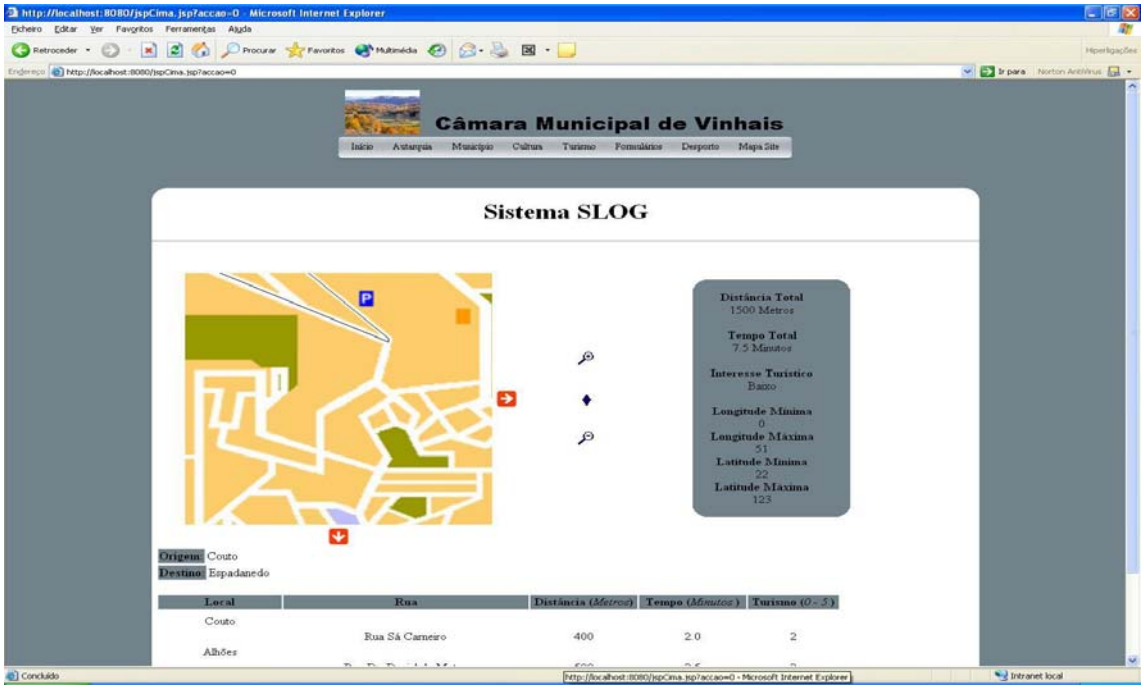
A próxima imagem mostra a aplicação após ter diminuído o *zoom*.

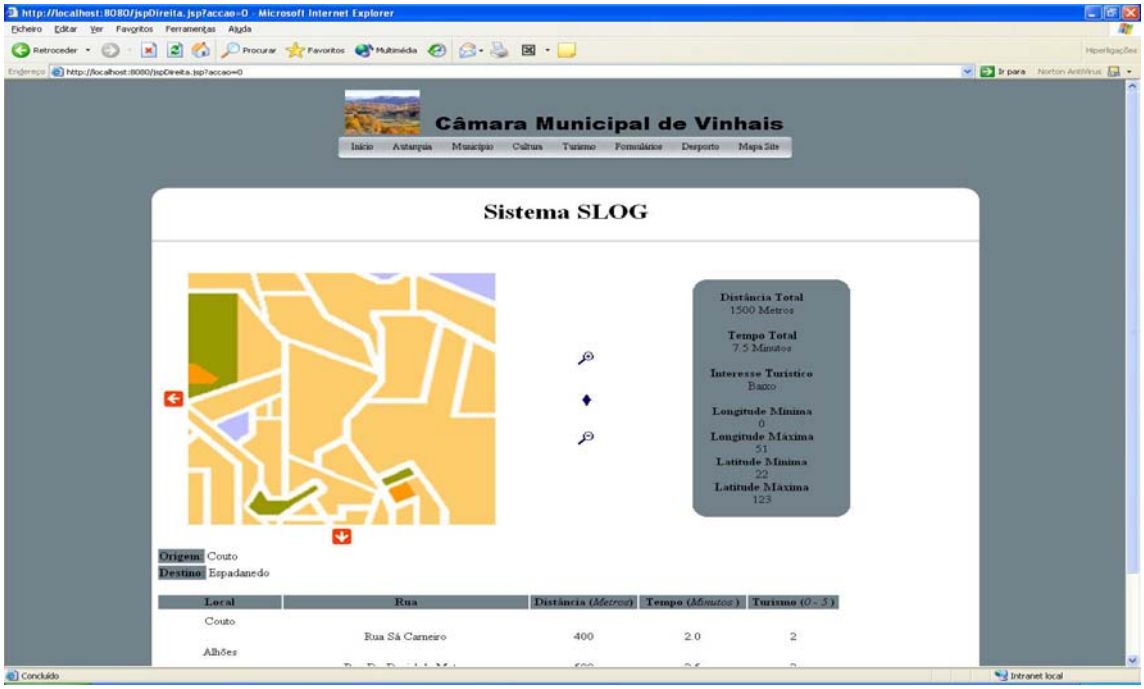
A imagem seguinte mostra a aplicação após ter aumentado o *zoom*.

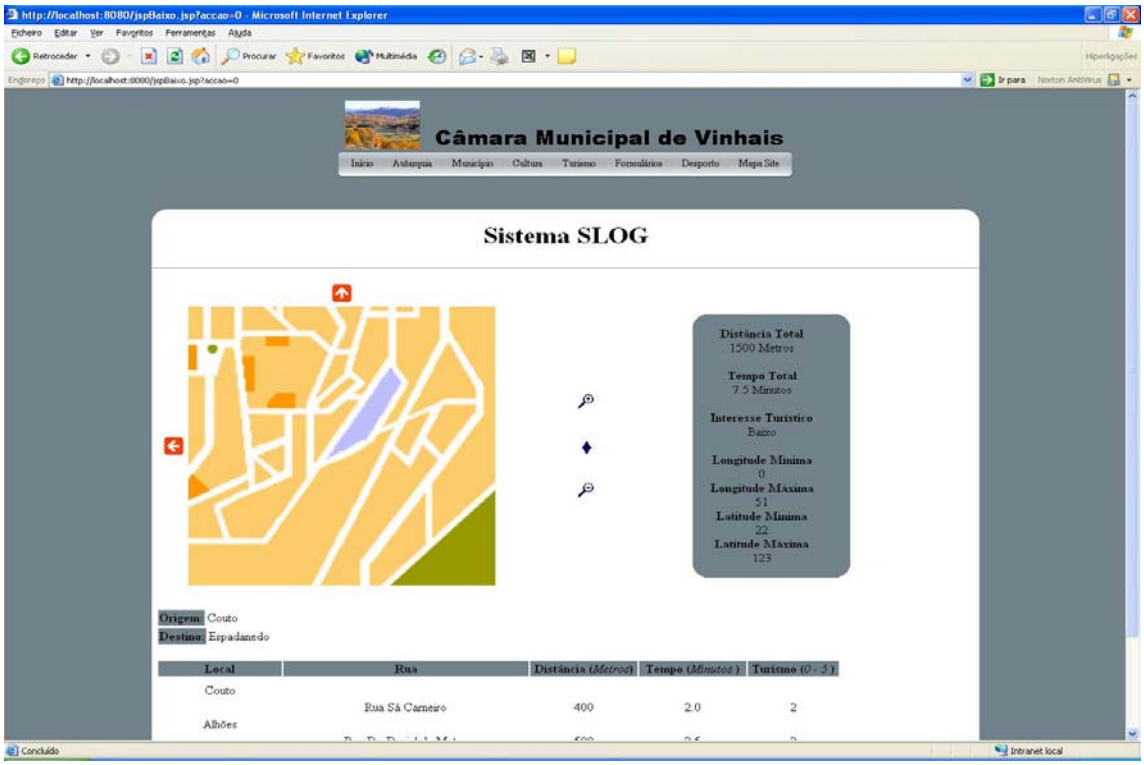
A imagem seguinte mostra a aplicação após ter accionado o ícone para a direita.

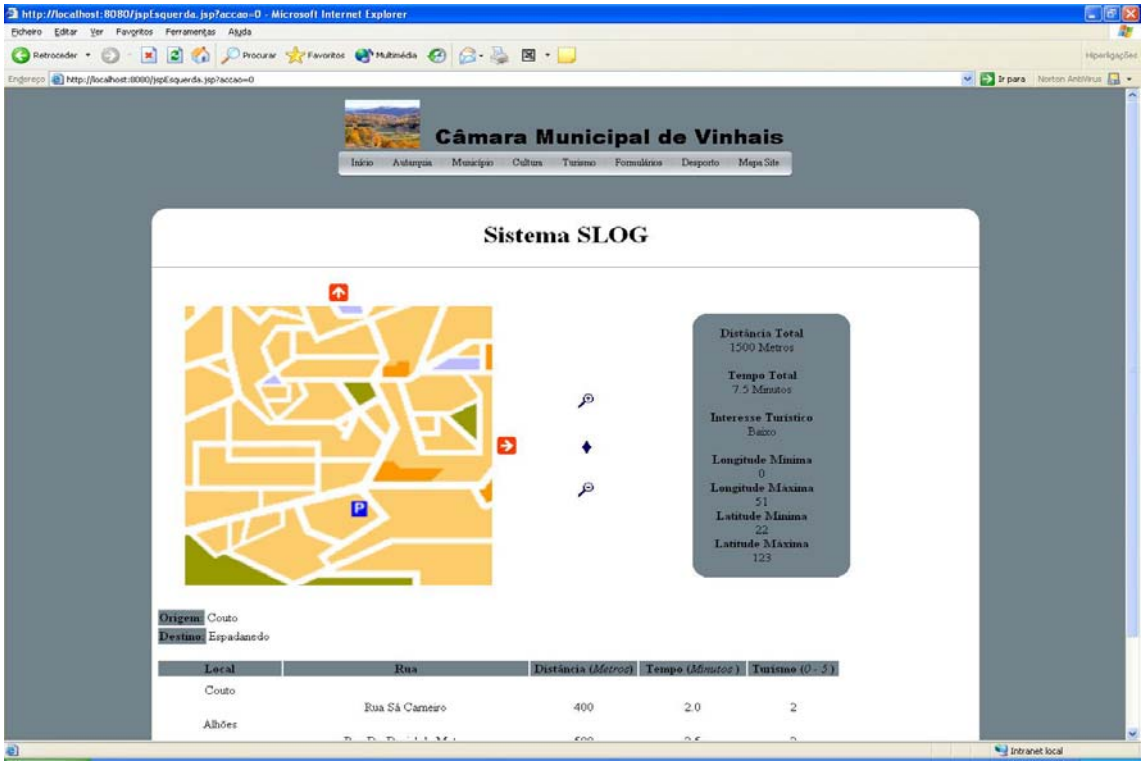


Por último, nas próximas imagens podemos ver os casos em que os ícones que rodeiam a imagem desaparecem.






http://localhost:8080/jspBaixo.jsp?accao=0 - Microsoft Internet Explorer
Câmara Municipal de Vinhais
Início Autarquia Município Cultura Turismo Formulários Desporto Mapa Site
Sistema SLOG
Distância Total
1500 Metros
Tempo Total
7.5 Minutos
Interesse Turístico
Baixo
Longitude Mínima
0
Longitude Máxima
51
Latitude Mínima
22
Latitude Máxima
123
Origem: Couto
Destino: Espadanedo
Local
Rua
Distância (Metros)
Tempo (Minutos)
Turismo (0 - 5)
Couto
Rua Sá Carneiro
400
2.0
2
Albões


http://localhost:8080/jspEsquerda.jsp?accao=0 - Microsoft Internet Explorer
Câmara Municipal de Vinhais
Início Autarquia Município Cultura Turismo Formulários Desporto Mapa Site
Sistema SLOG
Distância Total
1500 Metros
Tempo Total
7.5 Minutos
Interesse Turístico
Baixo
Longitude Mínima
0
Longitude Máxima
51
Latitude Mínima
22
Latitude Máxima
123
Origem: Couto
Destino: Espadanedo
Local
Rua
Distância (Metros)
Tempo (Minutos)
Turismo (0 - 5)
Couto
Rua Sá Carneiro
400
2.0
2
Albões