

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Eduardo Ferreira de Oliveira

**Um sistema para cálculo de rotas de caminho
mínimo utilizando pgRouting e dados do
OpenStreetMap**

Uberlândia, Brasil

2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Eduardo Ferreira de Oliveira

**Um sistema para cálculo de rotas de caminho mínimo
utilizando pgRouting e dados do OpenStreetMap**

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Humberto Luiz Razente

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2021

Eduardo Ferreira de Oliveira

Um sistema para cálculo de rotas de caminho mínimo utilizando pgRouting e dados do OpenStreetMap

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 18 de junho de 2021:

Prof. Humberto Luiz Razente
Orientador

Prof. Rodrigo Sanches Miani

**Prof. William Chaves de Souza
Carvalho**

Uberlândia, Brasil
2021

Dedico este trabalho a minha mãe Geralda, que foi pai e mãe dentro de casa, sempre trabalhou para dar educação e uma vida digna a mim e ao meu irmão. A minha avó Adélia, que é minha segunda mãe. Enquanto minha mãe trabalhava era quem cuidada de mim o dia inteiro. Ao meu irmão Bruno, que sempre me apoiou nos estudos e em tudo que precisei. Obrigado por todo o apoio, se sou o que sou hoje é graças ao amor de vocês.

Agradecimentos

Agradeço à minha mãe Geralda, à minha avó Adélia e ao meu irmão Bruno. O apoio de vocês ao longo da vida e do curso foi fundamental.

Ao meu orientador Humberto por ter me guiado nesta etapa tão importante que é o trabalho de conclusão de curso. Obrigado pela paciência, orientações, ideias e ensinamentos.

A todos os professores que me passaram conhecimentos, experiências e valores nesses anos de faculdade. Agradeço ao professor Mauricio Cunha Escarpinati por ter me ajudado a chegar até esta etapa final da graduação.

Aos funcionários que passaram pela coordenação e secretaria do curso de Sistemas de Informação. Obrigado por toda a ajuda de vocês ao longo do curso.

A todos os funcionários que cuidam desta universidade, pessoas que na maioria das vezes não aparecem para os alunos, mas são eles que a mantêm bem cuidada e organizada.

Por fim, agradeço aos meus colegas e amigos que passaram comigo por essa jornada. Cada ajuda, brincadeiras, conversas e conselhos foram muito importantes na minha trajetória dentro da faculdade.

Resumo

Este trabalho de conclusão de curso apresenta o processo de desenvolvimento de um sistema que tem o objetivo de calcular rotas de caminho mínimo utilizando dados gratuitos do OpenStreetMap. Adotou-se o uso do PostgreSQL, PostGIS e pgRouting para criar uma base de dados com suporte a objetos geográficos e roteamento geoespacial. O sistema implementado contém uma interface em que usuários enviam endereços de origem e de destino para encontrar a menor rota entre eles. É retornado o caminho a ser seguido e a distância total. Os resultados obtidos durante a pesquisa e construção do projeto demonstram uma das várias possibilidades de se trabalhar com dados geográficos de uma fonte aberta. Neste trabalho também será apresentado como esses dados podem ser definidos, obtidos e importados para uma base.

Palavras-chave: Bancos de dados geográficos. Dados espaciais. PostGIS. pgRouting. Problema do caminho mínimo.

Lista de ilustrações

Figura 1 – Componentes de um sistema de informações geográficas	15
Figura 2 – Representação de pontos, linhas e polígonos	16
Figura 3 – Exemplo de dados armazenados no formato raster	17
Figura 4 – Construção representada como uma relação multi-polígono	20
Figura 5 – Grafo dirigido e ponderado contendo seis vértices	22
Figura 6 – Passos para o cálculo do caminho mais curto entre os vértices A e F . .	23
Figura 7 – Menor caminho entre A e F obtido através do algoritmo de Dijkstra . .	25
Figura 8 – Formato da requisição para a Geocoding API	29
Figura 9 – Exemplo de uma chamada para a Geocoding API	29
Figura 10 – Parte da resposta de uma requisição para a Geocoding API	30
Figura 11 – Rota calculada sendo visualizada com o <i>geometry viewer</i> no pgAdmin 4	33
Figura 12 – Lista de tabelas do banco após a importação dos dados	35
Figura 13 – Visualização de alguns dados da tabela <i>configuration</i>	36
Figura 14 – Visualização do SRID 4326 na tabela <i>spatial_ref_sys</i>	38
Figura 15 – Representação dos nós em uma rede de ruas	39
Figura 16 – Exemplo de uma <i>linestring</i>	42
Figura 17 – Linha que liga um nó e origem e outro de destino	42
Figura 18 – Região da cidade de Uberlândia visualizada no OpenStreetMap	43
Figura 19 – Região da cidade de Uberlândia visualizada no QGIS	43
Figura 20 – Resultado da consulta utilizando a função <i>pgr_dijkstra</i>	46
Figura 21 – Exemplificação de uma busca pelo nó mais próximo dos pontos de ori- gem e de destino	48
Figura 22 – Fluxograma do sistema	49
Figura 23 – Tela inicial da aplicação	51
Figura 24 – Demonstração da validação de campos	52
Figura 25 – Exemplo com os dados preenchidos de forma correta	52
Figura 26 – Resposta da aplicação contendo as informações de menor caminho . . .	53
Figura 27 – Exemplo de cálculo de rota de custo mínimo para pedestres	54
Figura 28 – Exemplo de cálculo de rota de custo mínimo para veículos	54
Figura 29 – Distância total entre o Terminal Planalto e Uberlândia Shopping . . .	56
Figura 30 – Rota entre o Terminal Planalto e Uberlândia Shopping no Google Maps	56
Figura 31 – Rota entre o Terminal Planalto e Uberlândia Shopping no pgAdmin 4 .	56
Figura 32 – Rota entre o Parque Siquierolli e Museu Municipal	57
Figura 33 – Resposta da aplicação com a distância total do menor caminho entre o Parque Siquierolli e Museu Municipal	58
Figura 34 – Resultados do teste de desempenho	59

Figura 35 – Resultados da repetição do teste de desempenho	60
Figura 36 – Página inicial do Export Hot Tool	66
Figura 37 – Tela inicial da exportação	66
Figura 38 – Tela de seleção da cidade de Uberlândia para a exportação	67
Figura 39 – Área da cidade de Uberlândia selecionada automaticamente	67
Figura 40 – Tela de definição do nome do arquivo a ser exportado	68
Figura 41 – Tela de seleção do formato do arquivo a ser exportado	68
Figura 42 – Tela de escolha dos dados a serem exportados	68
Figura 43 – Tela de finalização da exportação	69
Figura 44 – Exportação finalizada e arquivo disponível para baixar	69
Figura 45 – Página de download do osmconvert	70
Figura 46 – Arquivos executável e pbf em um mesmo local	70
Figura 47 – Iniciação do osmconvert	71
Figura 48 – Etapa 1 da conversão do arquivo	71
Figura 49 – Etapa 2 da conversão do arquivo	71
Figura 50 – Etapa 3 da conversão do arquivo	71
Figura 51 – Representação do arquivo de formato osm criado	71
Figura 52 – Etapa de seleção dos componentes na instalação do PostgreSQL	72
Figura 53 – Etapa de seleção da instalação do PostgreSQL	73
Figura 54 – Tela de seleção do PostGIS 3.1 para ser instalado	73
Figura 55 – Tela de conclusão da instalação	73
Figura 56 – Tela de criação de uma nova <i>database</i>	74
Figura 57 – Etapa de escolha do nome da <i>database</i> a ser criada	74
Figura 58 – Etapa de criação de uma extensão em uma <i>database</i>	75
Figura 59 – Etapa de escolha do PostGIS para ser adicionado como extensão	75
Figura 60 – Lista de extensões criadas na <i>database</i>	75
Figura 61 – Navegação até a pasta <i>bin</i> do PostgreSQL	76
Figura 62 – Comando para a importação do arquivo osm	76
Figura 63 – Tela de quando uma importação é concluída com sucesso	77
Figura 64 – Consulta no pgAdmin para confirmar que os dados foram importados	77

Lista de abreviaturas e siglas

ANSI	American National Standards Institute
API	Application Programming Interface
CSS	Cascading Style Sheets
GIS	Geographic Information System
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JPA	Java Persistence API
JSON	JavaScript Object Notation
MG	Minas Gerais
ODbL	Open Database License
OGC	Open Geospatial Consortium
OSGeo	Open Source Geospatial Foundation
OSM	OpenStreetMap
PBF	Protocolbuffer Binary Format
POM	Project Object Model
SGBD	Sistema Gerenciador de Banco de Dados
SGBDG	Sistema de Gerenciamento de Banco de Dados Geográfico
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
SIG	Sistema de Informação Geográfica
SQL	Structured Query Language
WEB	World Wide Web
XML	Extensible Markup Language

Sumário

1	INTRODUÇÃO	11
1.1	Objetivos	11
1.2	Metodologia	12
1.3	Organização do trabalho	13
2	REVISÃO BIBLIOGRÁFICA	14
2.1	Sistemas de informações geográficas	14
2.2	Banco de dados Geográficos	15
2.2.1	Estrutura dos dados	16
2.3	OpenStreetMap	17
2.3.1	Formato dos dados	18
2.3.2	Elementos	19
2.4	Problema do caminho mínimo	21
2.4.1	Algoritmo de Dijkstra	22
3	DESENVOLVIMENTO	26
3.1	Tecnologias utilizadas	26
3.1.1	Front-end	26
3.1.1.1	HTML	26
3.1.1.2	<i>Thymeleaf</i>	26
3.1.1.3	<i>Bootstrap</i>	27
3.1.1.4	<i>Material Icons</i>	27
3.1.2	Back-end	27
3.1.2.1	Java	27
3.1.2.2	<i>Spring Framework</i> e <i>Spring Boot</i>	27
3.1.2.3	<i>Geocoding API</i>	28
3.1.2.4	Maven	30
3.1.3	Tecnologias de banco de dados empregadas	30
3.1.3.1	PostgreSQL	30
3.1.3.2	osmconvert	31
3.1.3.3	osm2pgrouting	31
3.1.3.4	PostGIS	31
3.1.3.5	pgRouting	32
3.1.4	pgAdmin 4	32
3.1.5	Stack Builder	33
3.2	Base de dados	34

3.2.1	Criação	34
3.2.1.1	Exportação do mapa de Uberlândia	34
3.2.1.2	Preparação da base de dados no PostgreSQL	34
3.2.1.3	Importação do mapa para o banco de dados	34
3.2.2	Estrutura da base de dados	35
3.2.2.1	Tabela <i>configuration</i>	35
3.2.2.2	Tabela <i>pointsofinterest</i>	37
3.2.2.3	Tabela <i>spatial_ref_sys</i>	37
3.2.2.4	Tabela <i>ways_vertices_pgr</i>	39
3.2.2.5	Tabela <i>ways</i>	41
3.2.3	Cálculo do menor caminho	45
3.3	Desenvolvimento do sistema	49
3.4	Interface e funcionamento	50
4	TESTES E RESULTADOS	55
4.1	Teste de menor caminho para pedestres e comparação com o Google Maps	55
4.2	Teste de menor caminho para veículos e comparação com o Google Maps	57
4.3	Teste de tempo médio para o cálculo de rotas	58
5	CONCLUSÃO	61
5.1	Trabalhos Futuros	61
5.2	Considerações Finais	62
	REFERÊNCIAS	63
	APÊNDICES	65
	APÊNDICE A – EXPORTAÇÃO E CONVERSÃO DE FORMATO DO MAPA DE UBERLÂNDIA	66
A.1	Exportação do mapa de Uberlândia	66
A.2	Convertendo o formato do arquivo	70
	APÊNDICE B – INSTALAÇÕES E CONFIGURAÇÕES DA BASE DE DADOS	72
B.1	Instalação do PostgreSQL e configuração do banco de dados	72
B.2	Importação do mapa	76

1 Introdução

Na sociedade tem-se uma grande necessidade no uso de aplicações que permitem traçar rotas ou buscar determinados pontos de interesse. Seja para uma viagem, trabalho, lazer ou qualquer outra finalidade, elas são frequentemente utilizadas. Com esse cenário surgem algumas soluções para suprir essas demandas. Dentre elas, destacam-se os sistemas de navegação como o GPS (*global positioning system*), e serviços de localização de endereço e de roteirização como o Google Maps¹.

Quando alguém procura por um endereço ou rota no Google Maps, ou em qualquer outra ferramenta que possua um mapa com essas funcionalidades, ele está lidando com um sistema de informações geográficas (SIG). Fitz (2008, p. 23) define um sistema de informação como um sistema capaz de “coletar, armazenar, recuperar, transformar e visualizar dados e informações a ele vinculadas”. De acordo com Casanova et al. (2005), os SIG são como sistemas de informações convencionais, diferem-se apenas por serem capazes de armazenar atributos descritivos e geometrias de diferentes tipos de dados geográficos.

A maioria dos usuários que usam esses sistemas no dia a dia não sabem como eles funcionam internamente. De que modo os objetos do mundo real podem ser representados, tratados, armazenados e analisados computacionalmente? Onde obter esses tipos de dados? Como armazená-los? Como computar uma rota entre dois endereços?

Este trabalho irá apresentar como manipular objetos geográficos extraídos de uma fonte de dados aberta. Será mostrado como esses objetos podem ser representados dentro de um Sistema de Gerenciamento de Banco de Dados Relacional (SGBDR), permitindo que sistemas de informações geográficas os utilize.

A fim de exemplificar uma das várias possibilidades de se trabalhar com dados geográficos armazenados em um banco, optou-se pelo desenvolvimento de um sistema WEB que computa rotas de caminho mínimo a partir de endereços fornecidos por usuários e apresenta em uma interface intuitiva.

1.1 Objetivos

O objetivo geral deste trabalho é desenvolver um sistema para o cálculo de rotas de caminho mínimo entre endereços utilizando os dados de mapas livres do OpenStreetMap. Entre os objetivos específicos destacam-se:

- A construção de uma base de dados geográfica com as informações da cidade de

¹ Google Maps: <https://www.google.com/maps>

Uberlândia (MG) obtidas do OpenStreetMap;

- A apresentação do padrão de dados do OpenStreetMap;
- O emprego das extensões PostGIS e pgRouting para lidar com dados espaciais, especialmente para o roteamento geoespacial na base de dados, permitindo o cálculo de rotas de menor caminho pelo SGBDR;
- A criação de um sistema em que os usuários consigam enviar um endereço de origem e outro de destino, para que seja retornado o menor caminho entre eles, considerando o sentido das vias (para percursos com automóveis) ou sem considerar o sentido das vias (para percursos à pé);
- A disponibilização de um tutorial para obtenção dos dados do OpenStreetMap e criação de uma base de dados geográficos com roteamento geoespacial.

1.2 Metodologia

Para alcançar o objetivo proposto, a primeira etapa do trabalho consistiu na pesquisa bibliográfica. Os sistemas de informações geográficas, os banco de dados geográficos, o padrão OpenStreetMap (OSM) e o problema do caminho mínimo em grafos foram estudados e são apresentados no Capítulo 2. O aprendizado sobre o OSM envolveu entender como se faz a exportação dos seus dados de mapa, como eles são organizados, quais formatos possuem e de que forma podem ser importados para um banco de dados.

Em seguida foram escolhidas as ferramentas para efetuar a importação de mapas para uma base e também definiu-se o sistema de gerenciamento de bancos de dados a ser usado. Optou-se pelo PostgreSQL, que é um gerenciador de banco de dados relacional de código aberto. O PostgreSQL possui uma extensão espacial chamada PostGIS, que permite o armazenamento de objetos GIS (*Geographic Information Systems*). O mapa de Uberlândia foi extraído do OpenStreetMap com o sistema HOT Export Tool, que permite exportar dados personalizados do OpenStreetMap. Para preparar e importar o mapa foi utilizado o `osm2pgrouting`, ferramenta de linha de comando que realiza a importação e deixa os dados aptos para serem trabalhados com o pgRouting. O pgRouting estende a base geoespacial PostGIS / PostgreSQL e está preparado para trabalhar com mapas do OSM, fornecendo roteamento geoespacial e outras funcionalidades de análise de rede. Esta extensão fornece vários algoritmos de roteamento, como funções para o cálculo do caminho mais curto entre dois pontos.

Após o mapa ter sido importado no PostgreSQL, foi definida uma função de computação de rota do pgRouting para o cálculo do menor caminho entre dois pontos. A função escolhida foi a `pgr_dijkstra`, que usa o algoritmo de Dijkstra e permite encontrar o menor caminho em um grafo.

Finalmente, foram selecionadas as tecnologias para o desenvolvimento da aplicação WEB. As últimas etapas foram a implementação, os testes e a análise de resultados do sistema.

1.3 Organização do trabalho

O trabalho é composto por cinco capítulos e dois apêndices. O Capítulo 1 trata da introdução, objetivos e da metodologia seguida para a construção do trabalho.

No Capítulo 2 é apresentada a revisão bibliográfica, que aborda o conjunto de conhecimentos necessários para a compreensão e desenvolvimento do trabalho. Nessa parte há o entendimento sobre os seguintes temas: sistemas de informações geográficas, banco de dados geográficos, *OpenStreetMap* e o problema do caminho mínimo.

O desenvolvimento do projeto está descrito no Capítulo 3, iniciado pelo apontamento das tecnologias empregadas. Em seguida são apresentados a criação e a estrutura da base de dados, as funções para o cálculo do menor caminho, a implementação do sistema, a interface e o seu funcionamento.

No Capítulo 4 são apresentados os testes realizados no sistema e os seus resultados. Nele são retratados os testes e comparações dos resultados com o Google Maps. Também foi realizado um teste de desempenho, com o objetivo de obter o tempo médio do cálculo de rotas da aplicação com o algoritmo de Dijkstra do *pgRouting*.

O Capítulo 5 apresenta a conclusão do projeto e as considerações finais. Trabalhos futuros também são sugeridos.

Por fim, nos apêndices A e B é apresentado como configurar a base de dados. Essa configuração passa pelas etapas de exportação do mapa, conversão de formato do arquivo, instalação do PostgreSQL e de suas extensões, e a importação do mapa no banco de dados.

2 Revisão Bibliográfica

Este capítulo apresenta um levantamento bibliográfico acerca dos assuntos necessários para o entendimento do trabalho. São apresentados os conceitos sobre sistemas de informações geográficas, banco de dados geográficos, OpenStreetMap e problema do menor caminho.

2.1 Sistemas de informações geográficas

Sistema de informação geográfica (SIG), ou do inglês, *Geographic Information System* (GIS), é definido por [Burrough e McDonnell \(1998, p. 11\)](#) como “um poderoso conjunto de ferramentas para coletar, armazenar, recuperar à vontade, transformar e exibir dados espaciais do mundo real para um conjunto particular de propósitos”. Segundo [Câmara et al. \(1996\)](#), dados espaciais são dados que descrevem fenômenos associados a alguma dimensão espacial. Dados geo-referenciados ou dados geográficos são um tipo de dados espaciais, eles são utilizados em SIGs e representam fatos, objetos e fenômenos da Terra associados à sua localização na superfície terrestre.

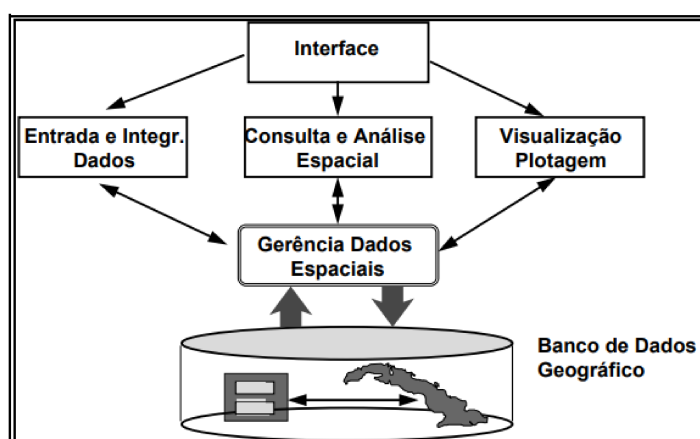
Os resultados gerados de um SIG estão vinculados ao espaço físico, mas também podem ser trabalhados com fenômenos climáticos, humanos, sociais, econômicos, etc. Um SIG pode ser aplicado, por exemplo, para o planejamento urbano de um município. Ele pode fornecer informações que auxiliam em tomadas de decisões, como o mapeamento do município, zoneamentos (ambientais, socioeconômicos, turísticos), monitoramento de áreas de risco e de proteção ambiental, estruturação de redes (energia, água e esgoto), estudos para expansão urbana, controle de ocupações e construções irregulares, entre outros ([FITZ, 2008](#)). Portanto, esses tipos de sistemas não servem apenas para descobrir qual é a sua localização, onde está algo, como chegar a determinado lugar. Decisões dos mais diversos tipos podem ser tomadas a partir da análise de dados. Por exemplo, uma prefeitura pode verificar qual o melhor local para adicionar novas delegacias de polícia com base no armazenamento e análise de endereços e atributos referentes aos locais que mais ocorrem crimes na cidade. Com o armazenamento de dados geográficos, é possível fazer consultas complexas incorporadas por uma infinidade de atributos que estejam disponíveis. Outro exemplo para o caso de uma prefeitura que gostaria de adicionar novas delegacias, elas poderiam realizar uma análise usando distâncias, buscando locais em que os crimes acontecem a uma distância de mais de vinte quilômetros de qualquer departamento de segurança já existente. Além de consultar pela distância, é pesquisado por delitos que sejam assaltos a mão armada e praticados contra comerciantes.

Os componentes de um SIG (Figura 1) são apresentados conforme as definições de

Casanova et al. (2005). Eles estão divididos em três níveis:

- Nível inicial: Interface homem-máquina que define como o sistema é operado e controlado.
- Nível intermediário: Refere-se aos mecanismos de processamento de dados espaciais. É dividido em Entrada e Integração de Dados, Consulta e Análise Espacial, Visualização e Plotagem. A Entrada e Integração inclui os mecanismos de conversão de dados. Consulta e Análise Espacial incluem operações topológicas, álgebra de mapas, estatística espacial, modelagem numérica de terreno e processamento de imagens. A visualização e plotagem devem oferecer suporte adequado para a absorção de conhecimento dos aspectos relevantes dos dados pesquisados.
- Nível interno: Possui um sistema de gerência de bancos de dados geográficos para armazenar e recuperar dados espaciais e seus atributos.

Figura 1 – Componentes de um sistema de informações geográficas



Fonte: Casanova et al. (2005).

2.2 Banco de dados Geográficos

Um sistema de gerenciamento de banco de dados geográfico (SGBDG) é utilizado para armazenar e recuperar dados geográficos para serem usados pelos SIGs. De acordo com Obe e Hsu (2015), um banco de dados geográfico, ou banco de dados espacial, oferece ferramentas para armazenar, analisar e organizar dados espaciais. Eles possuem tipos de dados de coluna especiais para armazenar objetos do espaço em tabelas. Esse tipo de banco tem funções e índices espaciais para a consulta e manipulação de informações geográficas através de linguagens de consulta. Segundo os autores, a maioria dos bancos espaciais são de natureza relacional.

No trabalho foi utilizado um banco de dados objeto-relacional que é *open source*, o PostgreSQL. Junto ao PostgreSQL, foi adicionada a extensão PostGIS, que insere suporte espacial a ele. Detalhes sobre a base de dados do trabalho e as tecnologias usadas são discutidos no capítulo 3.

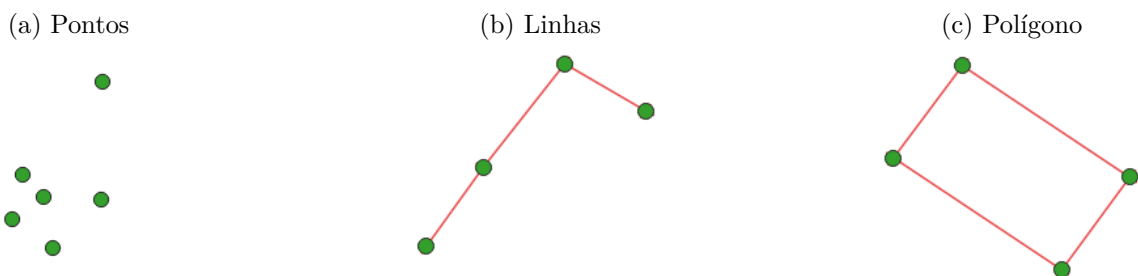
2.2.1 Estrutura dos dados

Segundo Fitz (2008), os bancos de dados geográficos usam tipos de dados espaciais e alfanuméricos. O autor apresenta a divisão dos dados espaciais em estruturas vetoriais e matriciais:

- Vetorial: é composta por pontos, linhas e polígonos, usando sistemas de coordenadas para serem representadas. Utiliza-se de um par de coordenadas para representar os pontos e conjuntos de pares de coordenadas para linhas e polígonos. As entidades espaciais podem ser caracterizadas nesse tipo de estrutura como cidades, delimitadores de área de loteamento, pontos de interesse, ruas, linhas de transmissão de energia, parques, etc.
- Matricial: Também chamada de *raster*, a estrutura matricial é representada por uma matriz de n linhas e m colunas, em que cada célula (*pixel*) possui um par de coordenadas e um valor associado z que pode indicar por exemplo, uma cor ou tom de cinza referente a imagens de satélites, fotografias aéreas digitais, mapas digitalizados, dentre outros. O vínculo com dados alfanuméricos é mais trabalhoso do que com a estrutura vetorial, pois nesse caso a relação deve ser *pixel a pixel*, enquanto que na vetorial o vínculo é com os pontos, linhas e polígonos.

A Figura 2 criada com o software QGIS¹ mostra uma representação dos dados vetoriais, que são pontos, linhas e polígonos. A Figura 3 apresenta um exemplo de dados do mundo real armazenados no formato matricial (*raster*).

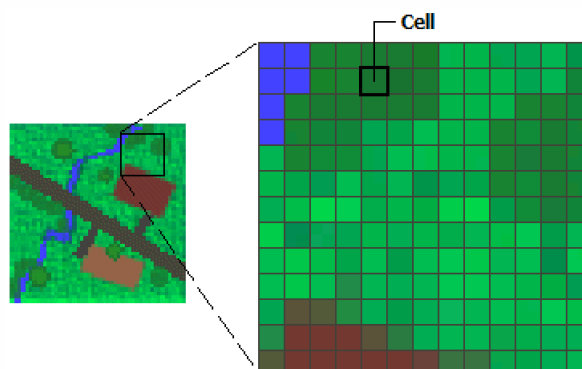
Figura 2 – Representação de pontos, linhas e polígonos



Fonte: O autor.

¹ <https://qgis.org/>

Figura 3 – Exemplo de dados armazenados no formato raster



Fonte: [ArcGIS \(2020\)](#).

Os dados em um SIG devem ter atributos vinculados aos dados espaciais, que são identificados por coordenadas. Os dados alfanuméricos são dados utilizados para descrever os dados espaciais, atribuindo informações a eles. O banco deve ser estruturado para que os dados tradicionais (alfanuméricos) sejam relacionados aos dados espaciais, isso acontece por meio de códigos identificadores ([FITZ, 2008](#)). Essas informações podem ser textos ou números, assim como em um banco de dados tradicional. Dados como o nome de uma rua, nome de uma cidade, quantidade de população em um bairro, dono de um loteamento, endereço e tipo de solo são exemplos de informações que ajudam a descrever as características dos dados espaciais armazenados em um SGBDG.

2.3 OpenStreetMap

Com o objetivo de obter dados geográficos para serem usados no projeto, utilizou-se o OpenStreetMap². De acordo com [Bennett \(2010\)](#), o OSM é um projeto de construção de um banco de dados geográfico gratuito e de mapeamento colaborativo. Toda a base de dados é criada e manipulada pelos colaboradores, em que a maioria são voluntários, mas também há empresas e órgãos do governo que contribuem. Eles criam e editam mapas com base em informações obtidas percorrendo áreas e gravando os seus movimentos com um GPS. Outros meios de conseguirem informações para a criação de mapas é por fontes gratuitas, como mapas não protegidos por direitos autorais e bancos de dados que podem ser de domínio público ou doados pelas empresas detentoras das informações. O OSM permite a adição de qualquer objeto geográfico como ruas, ciclovias, parques, banheiros públicos, edifícios, igrejas, etc.

O OpenStreetMap foi escolhido por disponibilizar os seus dados de forma gratuita. Conforme informado na página inicial da sua Wiki³, esse projeto teve início por conta das

² <https://www.openstreetmap.org/>

³ https://wiki.openstreetmap.org/wiki/Main_Page

restrições legais e técnicas que a maioria dos mapas disponibilizados na WEB possuem, como o Google Maps. Essas restrições limitam que as pessoas usem os dados geográficos de maneiras criativas, produtivas e até mesmo inesperadas. Segundo o site oficial do OSM ([OPENSTREETMAP, 2021a](#)), os dados estão sob a licença *Open Data Commons Open Database License* (ODbL)⁴, que permite que eles sejam manipulados e distribuídos seguindo os direitos autorais e responsabilidades da ODbL.

O OSM fornece uma ferramenta de exportação dos seus dados, para isso, basta pesquisar o local desejado referente a qualquer lugar do mundo e selecionar a área para a extração. O site oficial atualmente aplica um limite de 50 mil nós para serem extraídos de uma só vez, mas ele fornece e sugere o uso do Planet OSM⁵ e de outros provedores de dados⁶. O Planet OSM disponibiliza todos os dados do OSM em apenas um arquivo. Outros provedores fornecem extratos de arquivos do Planet OSM por região, país, continentes, cidades, e até extrações personalizadas de qualquer área sem a limitação de tamanho do site oficial do OpenStreetMap.

2.3.1 Formato dos dados

O OpenStreetMap disponibiliza os seus dados em vários formatos diferentes. Segundo a Wiki do OpenStreetMap ([OPENSTREETMAP, 2021d](#)), os principais formatos são:

- OSM XML: Arquivos do OpenStreetMap no formato legível para humanos XML (*Extensible Markup Language*). Pode ser facilmente editado, compactado e mesclado, mas possui um tamanho grande, ficando mais lento para ser escrito e processado.
- PBF: *Protocolbuffer Binary Format* é um formato alternativo ao OSM XML, sendo otimizado e bem mais compactado. Apesar dessas vantagens, ele não é legível por humanos, por isso sua edição é dificultada.
- OSM JSON: Fornece os arquivos OSM no formato JSON. É baseado na API Overpass.⁷
- o5m: Tenta combinar as vantagens dos formatos OSM XML e PBF, mas é pouco suportado pelos aplicativos. Tem um tamanho pequeno, o processamento é rápido, pode ser fundido facilmente com outros arquivos, mas não é legível por humanos e por isso não pode ser editado por editores de texto comuns.
- Overpass JSON: Utilizado pela API Overpass, sendo uma variante JSON do formato OSM XML.

⁴ <https://opendatacommons.org/licenses/odbl/>

⁵ <https://planet.osm.org/>

⁶ https://wiki.openstreetmap.org/wiki/Processed_data_providers

⁷ https://wiki.openstreetmap.org/wiki/Overpass_API

- Level0L: Formato compatível com o OSM XML. É mais legível por humanos e tem uma redundância menor.

2.3.2 Elementos

Os elementos básicos dos dados do OSM são descritos nesta subseção. De acordo com a sua documentação ([OPENSTREETMAP, 2021b](#)), eles são utilizados para representar o mundo físico dentro do modelo de dados do OpenStreetMap, possuindo três tipos:

- *Nodes* (nós): Um nó⁸ possui uma latitude, longitude e um identificador para representar um ponto específico na superfície da Terra. Assim como para as relações e caminhos, os nós também podem ter *tags*, que definem sua finalidade. Um nó pode ser um banco em um parque, poço de água ou qualquer outro ponto autônomo. Eles são mais usados para definir a forma de um caminho (*way*). Exemplo de um nó no formato OSM XML:

```
<node id="751950285" lat="-9.587184" lon="-35.7626398"
  version="6" timestamp="2019-12-05T16:46:46Z" changeset="
  78009162" uid="9625115" user="Barataria2">
  <tag k="highway" v="traffic_signals"/>
</node>
```

- *Ways* (vias ou caminhos): Caminhos⁹ são compostos de 2 a 2000 nós em uma lista ordenada. São usados para representar elementos lineares no solo e limites de áreas, como rios, estradas, caminhos, parede, edifícios, florestas, dentre outros. Eles podem ser caminhos abertos ou fechados, possuindo *tags* associadas ou sendo incluídos em uma relação. Exemplo de um caminho no formato OSM XML:

```
<way id="30621726" version="20" timestamp="2019-01-26T17
:21:56Z" changeset="66659833" uid="7792091" user="riuri">
  <nd ref="338661989"/>
  <nd ref="3371751126"/>
  <nd ref="1561879652"/>
  <nd ref="1561879697"/>
  <nd ref="1561879726"/>
  <nd ref="1561879759"/>
  <nd ref="1561879798"/>
  <nd ref="3050804950"/>
  <nd ref="1561879849"/>
```

⁸ <https://wiki.openstreetmap.org/wiki/Node>

⁹ <https://wiki.openstreetmap.org/wiki/Way>

```

<nd ref="1561879918"/>
<nd ref="1561879975"/>
<nd ref="3371751125"/>
<tag k="highway" v="residential"/>
<tag k="name" v="Avenida Portugal"/>
</way>

```

- *Relations* (relações): Uma relação¹⁰ é uma estrutura de dados multifuncional que possui uma ou mais *tags* que definem o significado da relação, e uma lista ordenada de um ou mais elementos, que são nós, caminhos e/ou outras relações. Tem a função de definir relações lógicas ou geográficas entre elementos e também pode descrever a parte que um determinado recurso desempenha em uma relação. Alguns exemplos de relação são: relação de rota listando os caminhos que formam uma rodovia, rota de ônibus e um multi-polígono que descreve o limite de uma área. A Figura 4 mostra uma construção representada como uma relação do tipo multi-polígono.

Figura 4 – Construção representada como uma relação multi-polígono



Fonte: [OpenStreetMap](https://www.openstreetmap.org/) (2021e).

Exemplo de uma relação representada no formato OSM XML:

```

<relation id="2918887" version="7" timestamp="2016-07-23T22
:31:11Z" changeset="40980665" uid="463507" user="naoliv">
  <member type="way" ref="428018883" role="outer"/>
  <member type="way" ref="428018885" role="outer"/>
  <member type="way" ref="432811949" role="outer"/>
  <member type="way" ref="432811969" role="outer"/>
  <member type="way" ref="432812057" role="outer"/>
  <member type="way" ref="432812027" role="outer"/>
  <member type="way" ref="30621194" role="outer"/>
  <member type="way" ref="428018887" role="outer"/>

```

¹⁰ <https://wiki.openstreetmap.org/wiki/Relation>

```
<member type="way" ref="220729442" role="inner"/>
<tag k="building" v="yes"/>
<tag k="type" v="multipolygon"/>
</relation>
```

- *Tags* (etiquetas): As *tags*¹¹ possuem uma chave e um valor que descrevem o significado de um elemento. Como mostrado nos exemplos referentes aos nós, caminhos e relações, todos eles têm pelo menos uma *tag*. Um exemplo é uma *tag* em que a sua chave tem o valor de *name* e valor é “Avenida Portugal”. Essa *tag* transmite o significado de que o elemento que ela está associada representa uma avenida com esse nome.

2.4 Problema do caminho mínimo

O trabalho tem o objetivo de buscar a construção de um sistema que computa o menor caminho entre dois endereços. Por isso é importante conhecer os conceitos do problema do caminho mínimo e um dos principais algoritmos que buscam sua resolução.

Ir diariamente para o trabalho gastando a menor quantidade de combustível possível, chegar rapidamente em algum lugar, viajar, seja qual for o propósito, descobrir a melhor maneira para ir de um local até outro poupando tempo e dinheiro faz parte do cotidiano das pessoas. Em uma viagem qualquer, em que o objetivo é ir de uma cidade até outra utilizando um mapa rodoviário para encontrar o caminho mais curto, [Cormen et al. \(2012\)](#) informa que esse mapa pode ser modelado como um grafo. As interseções entre as estradas são os vértices, os segmentos de estradas entre essas interseções são as arestas e os pesos associados a cada aresta representam as distâncias rodoviárias. Além de distâncias, os pesos também podem representar qualquer outro valor que se deseja minimizar, como tempo, quantidade de semáforos, possibilidade de multas, e custos.

Nesse tipo de problema, há um grafo dirigido e ponderado $G=(V,E)$, com uma função de peso $w : E \rightarrow R$ que vai mapear as arestas (E) em números reais (R) representando os seus pesos. Para um caminho $p = (v_0, v_1, \dots, v_k)$, o seu peso ($w(p)$) é o somatório dos pesos de suas arestas. Caso exista um caminho de u (origem) até v (destino), o peso do caminho mínimo ($\delta(u, v)$) será: $\min \{ w(p) : u \stackrel{p}{\rightsquigarrow} v \}$. Se nenhum caminho existir o peso será infinito. O caminho mínimo de u até v será qualquer caminho p com peso $w(p) = \delta(u, v)$ ([CORMEN et al., 2012](#)).

Trabalhar com o problema do caminho mínimo não serve apenas para ser aplicado em viagens encontrando uma distância mínima para ir de um ponto A até um ponto B. Para [Hillier e Lieberman \(2006\)](#), esse problema pode ser utilizado para diminuir o custo ou tempo total de uma sequência de atividades quaisquer.

¹¹ <https://wiki.openstreetmap.org/wiki/Tags>

Existem vários algoritmos que resolvem o problema do caminho mínimo, por exemplo, o A*, Bellman-Ford, Floyd-Warshall e Dijkstra. No projeto não buscou-se avaliar os diferentes algoritmos. Como o objetivo do trabalho é a construção de um sistema para cálculo de rotas em mapas a partir de um par de endereços, o algoritmo de Dijkstra foi o escolhido porque é um dos mais populares para se trabalhar com esse tipo de problema. Ele não trabalha com arestas de peso negativo, atendendo as necessidades do grafo de roteamento que foi criado na base de dados. Nesse grafo, quando os caminhos são de mão dupla, o custo e o custo reverso da aresta são positivos. Para vias de mão única, o custo é positivo e custo reverso é negativo, fazendo com que a direção contrária da via não seja considerada. Outro fator motivador importante para essa escolha foi a extensão pgRouting, que foi usada neste projeto para adicionar roteamento geoespacial ao banco de dados. Essa extensão fornece várias funções que usam o algoritmo Dijkstra.

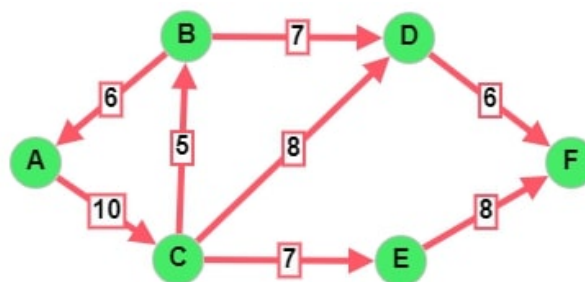
2.4.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra foi criado por Edsger Dijkstra e publicado em 1959 (DIJKSTRA, 1959). Ele é utilizado para resolver o problema do caminho mínimo em grafos ponderados. De acordo com Cormen et al. (2012), um grafo é considerado ponderado quando cada aresta pertencente a ele tem um peso associado. Esse algoritmo trabalha apenas com grafos onde o peso das arestas não são negativos.

O algoritmo rotula os nós como temporários e definitivos. Quando um nó j recebe um rótulo definitivo, o valor do seu nível é o comprimento do caminho mais curto entre a origem e ele. O algoritmo realiza iterações rotulando os nós até o nó de destino ser rotulado em definitivo. Para encontrar todos os nós do caminho mais curto é necessário fazer um *backtracking* a partir do nó de destino, buscando quem foi o responsável por rotular em definitivo cada um deles, chegando até ao nó inicial. (MARINS, 2011).

Um grafo dirigido e ponderado contendo seis vértices é exibido na Figura 5. A Figura 6 apresenta uma tabela com os passos para o cálculo do menor caminho entre os vértices **A** e **F** utilizando o algoritmo de Dijkstra.

Figura 5 – Grafo dirigido e ponderado contendo seis vértices



Fonte: O autor.

Figura 6 – Passos para o cálculo do caminho mais curto entre os vértices A e F

	A	B	C	D	E	F
1	(0,A)	∞	∞	∞	∞	∞
2	-	∞	(10,A)	∞	∞	∞
3	-	(15,C)	-	(18,C)	(17,C)	∞
4	-	-	-	(18,C)	(17,C)	∞
5	-	-	-	(18,C)	-	(25,E)
6	-	-	-	-	-	(24,D)

Fonte: O autor.

Detalhamento dos seis passos apresentados na tabela:

1. Passo inicial:

- A distância do vértice inicial até ele mesmo é definida como zero. Os outros vértices recebem o valor infinito, pois ainda não se sabe a distância mínima da origem até eles.
- Os vértices são enviados para uma fila prioritária. Estar nessa fila significa que os vértices estão rotulados como temporários.
- Vértices na fila: **A**=0, **B**= ∞ , **C**= ∞ , **D**= ∞ , **E**= ∞ , **F**= ∞ ;
- O vértice com a menor distância (**A**) é extraído da fila de prioridade, processado e rotulado definitivamente. Os vértices marcados em definitivo são representados na tabela com a cor vermelha.

2. Segundo passo:

- O vértice atual é o **A**, ele inspeciona os seus vizinhos levando em conta a direção das arestas. Para cada um deles a distância até o nó inicial é calculada. Nesse grafo o único vizinho é o vértice **C**.
- **C** é atualizado na fila de prioridade. O seu valor que antes era infinito é alterado para a distância calculada.
- Vértices na fila: **B**= ∞ , **C**=10, **D**= ∞ , **E**= ∞ , **F**= ∞ .
- O vértice com a menor distância (**C**) é extraído da fila de prioridade, processado e rotulado definitivamente.

3. Terceiro passo:

- O vértice atual é o **C**. A partir dele todos os vizinhos (**B**, **D** e **E**) são inspecionados e as distâncias deles até o vértice inicial são calculadas. O vértice **B**, por exemplo, recebe o valor (15,C) na tabela. Isso significa que a distância até ele é 15, e quem o rotulou foi o **C**. O valor 15 refere-se a distância da origem até **C** mais o custo da aresta de **C** até **B**.

- Cada vizinho é atualizado na fila de prioridade recebendo o novo valor de distância.
- Vértices na fila: $\mathbf{B}=15$, $\mathbf{D}=18$, $\mathbf{E}=17$, $\mathbf{F}=\infty$.
- O vértice com a menor distância (\mathbf{B}) é extraído da fila de prioridade, processado e rotulado definitivamente.

4. Quarto passo:

- O vértice atual é o \mathbf{B} . O seu único vizinho que não está rotulado em definitivo é o \mathbf{D} . Ele é inspecionado e a sua distância até o vértice inicial é calculada.
- A soma total da distância da origem até o vértice \mathbf{D} passando por \mathbf{B} é 22 ($10 + 5 + 7$), valor maior do que a distância total de \mathbf{D} passando pelo vértice \mathbf{C} , que é 18. Nesse caso nada é alterado. O valor de \mathbf{D} na fila de prioridade continua sendo 18.
- Vértices na fila: $\mathbf{D}=18$, $\mathbf{E}=17$, $\mathbf{F}=\infty$;
- O vértice com a menor distância (\mathbf{E}) é extraído da fila de prioridade, processado e rotulado definitivamente.

5. Quinto passo:

- O vértice atual é o \mathbf{E} . O seu único vizinho (\mathbf{F}) é inspecionado e a distância até o vértice inicial é calculada.
- \mathbf{F} é atualizado na fila de prioridade.
- Vértices na fila: $\mathbf{D}=18$, $\mathbf{F}=25$.
- O vértice com a menor distância (\mathbf{D}) é extraído da fila de prioridade, processado e rotulado definitivamente.

6. Último passo:

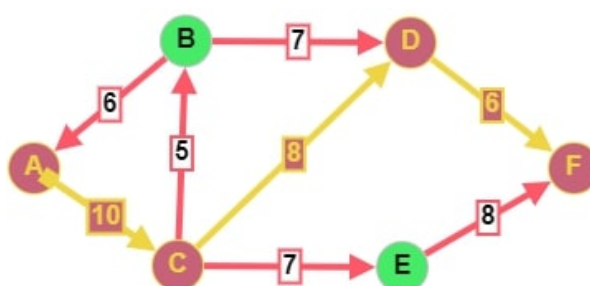
- O vértice atual é o \mathbf{D} . O seu único vizinho (\mathbf{F}) é inspecionado e a distância até o vértice inicial é calculada.
- A distância calculada é menor do que a encontrada anteriormente, onde o caminho vai de \mathbf{E} até \mathbf{F} . Portanto, o valor na fila de prioridade do vértice \mathbf{F} é alterado para essa menor distância.
- Vértices na fila de prioridade: $\mathbf{F}=24$.
- O vértice com a menor distância (\mathbf{F}) é extraído da fila de prioridade, processado e rotulado definitivamente.
- O algoritmo chega ao fim ao rotular definitivamente o vértice de destino.

Após rotular o vértice **F** que representa o destino, o menor caminho de **A** até ele é encontrado:

1. O vértice de destino é o **F**.
2. **F** foi rotulado definitivamente por **D**.
3. **D** foi rotulado definitivamente por **C**.
4. **C** foi rotulado definitivamente por **A**.
5. **A** é o vértice de origem.

Portanto, o caminho mínimo de **A** até **F** é: **A** -> **C** -> **D** -> **F**. A distância total a ser percorrida é 24. Pode-se visualizar o menor caminho na Figura 7, as suas arestas estão na cor amarela.

Figura 7 – Menor caminho entre A e F obtido através do algoritmo de Dijkstra



Fonte: O autor.

3 Desenvolvimento

Este capítulo descreve o desenvolvimento do projeto. São apresentadas as tecnologias utilizadas, a criação e a estrutura da base de dados, as funções para o cálculo do menor caminho, a implementação do sistema, a interface e o seu funcionamento.

3.1 Tecnologias utilizadas

Nesta seção são apresentadas as tecnologias que foram utilizadas no desenvolvimento do projeto. As tecnologias foram divididas em três seções: *front-end*, *back-end* e banco de dados. Para fins de organização, as ferramentas referentes ao banco de dados foram separadas das de *back-end*.

3.1.1 Front-end

No sistema criado, o *front-end* foi desenvolvido com HTML, *Thymeleaf*, *Bootstrap* e *Material Icons*. Essas tecnologias são descritas a seguir.

3.1.1.1 HTML

A linguagem de marcação HTML (*HyperText Markup Language*) baseia-se no uso de tags e elementos para estruturar o conteúdo e criar uma página WEB. Os navegadores de internet identificam e interpretam essas *tags* e elementos, determinando como o documento HTML será exibido para os usuários (W3S, 2021b).

Essa linguagem de marcação foi aplicada em conjunto com o *Thymeleaf*, *Bootstrap* e *Material Icons*, formando a interface de usuário. As próximas seções abordam esses outros três temas.

3.1.1.2 Thymeleaf

Thymeleaf é um *template engine* (mecanismo de modelo) Java para aplicações WEB. Ele atua do lado do servidor, e consegue processar seis tipos de modelos: HTML, XML, texto JavaScript, CSS e RAW. Com o uso do conceito de Modelos Naturais, é possível construir *templates* que não quebram a estrutura dos documentos, deixando o desenvolvimento deles mais simples e organizado. (THYMELEAF, 2018)

O *Thymeleaf* é bastante utilizado em aplicações Spring MVC, possuindo até mesmo um *starter* oficial para o *Spring Boot*, chamado de *spring-boot-starter-thymeleaf*. Ele facilita muito a criação de páginas WEB dinâmicas, por isso foi usado no *front-end* do sistema

para a troca de dados com o *back-end*. Além disso, foi aplicado para auxiliar na validação de erros no formulário da página inicial do projeto e, também, para o reaproveitamento de código da *tag* HEAD nas páginas HTML.

3.1.1.3 *Bootstrap*

O *Bootstrap*¹ é um *framework front-end* que em 2011 foi lançado como um produto *open source*. Ele permite facilmente a criação de designs responsivos e a estilização de página. Além disso, fornece componentes que auxiliam até mesmo um desenvolvedor com pouco conhecimento em *front-end* a criar designs mais complexos, como carrosséis de imagens, modais, *navbar*, entre outros. (W3S, 2021a).

Com o objetivo de estilizar as páginas do programa de forma simples e objetiva, foi feito o uso desse *framework*. Juntamente com o *Bootstrap*, o *Material Icons* forneceu ícones que melhoraram o *layout*.

3.1.1.4 *Material Icons*

Material Icons são ícones gratuitos disponibilizados pelo Google sob a Licença Apache Versão 2.0. Possui uma grande variedade de ícones que representam conceitos universais, e podem ser facilmente incorporados a um projeto WEB (GOOGLE, 2021b). No projeto, ele foi aplicado para representar pedestres, veículos e um pequeno mapa.

3.1.2 *Back-end*

O *back-end* foi implementado com a utilização das linguagens e arcabouços *Java*, *Spring Framework* e *Spring Boot*, *Geocoding API* e *Maven*. Optou-se por colocar as tecnologias de banco de dados e as de *back-end* em seções distintas. As técnicas de banco de dados empregadas são apresentadas na Seção 3.1.3.

3.1.2.1 *Java*

O sistema foi desenvolvido com a linguagem *Java*. Ela é uma linguagem de programação que se baseia no paradigma de orientação a objetos. A linguagem *Java* é uma das mais utilizadas no mundo, e segundo o próprio site², possui mais de 9 milhões de desenvolvedores.

3.1.2.2 *Spring Framework* e *Spring Boot*

O *framework Spring* surgiu em 2003 para ser uma alternativa contra as especificações do *Java Enterprise Edition*, que na época eram de difícil utilização. O *Spring*

¹ <https://getbootstrap.com>

² <https://www.java.com/pt-BR/about/>

visa facilitar a criação de aplicações corporativas e isso se deve aos vários módulos que possui. Os módulos oferecem injeção de dependências, mensageria, persistências de dados, suporte WEB, entre outros recursos (SPRINGIO, 2021). Com ele é fornecida uma infraestrutura que uma equipe de desenvolvimento gastaria muito tempo para implementar. O desenvolvedor pode se preocupar apenas com a lógica do problema a ser resolvido, e não com toda a infraestrutura a sua volta.

Além do *Spring Framework*, há vários outros projetos³ que fornecem infraestrutura e ajudam a construir aplicações. Um deles é o *Spring Boot*⁴. O *Spring Boot* é um projeto que visa facilitar a criação, configuração e publicação de aplicações, baseando-se no conceito de convenção sobre configuração. Em um arquivo de configuração do projeto, como o *pom.xml* do Maven, informa-se ao *Spring Boot* quais módulos serão utilizados, permitindo que ele o reconheça, instale todas as dependências e configure automaticamente. Isso acontece por meio dos *starters* do *Spring Boot*, que agrupam uma série de dependências em uma única dependência. Como exemplo, o *spring-boot-starter-web* traz consigo o *servlet container Tomcat* embutido e configurado, resolvendo outras dependências para a criação de aplicação WEB com *Spring MVC*. Isso não impede de individualizar suas configurações padrões, excluir certas dependências ou até mesmo trocá-las por outras. Caso necessite trocar, por exemplo, o *Tomcat* pelo *Jetty*, que é outro *servlet container*, basta adicionar uma exclusão e incluir o *starter* do *Jetty* (*spring-boot-starter-jetty*). Outro *starter* bastante importante aplicado no trabalho foi o *spring-boot-starter-data-jpa*, que faz o uso do *Hibernate* e JPA para trabalhar com a base de dados.

3.1.2.3 Geocoding API

A geocodificação (*Geocoding*) é o processo de converter um endereço legível por humanos em coordenadas geográficas. A *Geocoding* API do Google oferece um serviço que pode ser acessado via requisições HTTP, em que passamos um endereço e ele nos retorna com as coordenadas, como a latitude e a longitude desse local (GOOGLE, 2021c). A latitude e longitude serão usadas no projeto para encontrar os pontos mais próximos dos endereços de origem e de destino informados pelos usuários para o cálculo de rotas.

A *Geocoding* faz parte da *Google Cloud Platform*⁵, que possui serviços gratuitos e pagos. São oferecidos trezentos dólares em créditos de cortesia para novos clientes conhecerem e avaliarem os serviços de forma completa. Um ponto muito positivo é que, além disso, a Plataforma Google Maps disponibiliza um crédito mensal de duzentos dólares, sendo aplicados aos serviços qualificados, como o *Geocoding*. Uma tabela de preços é cedida pelo site⁶, em que é possível visualizar os valores de cada serviço e o uso gratuito

³ <https://spring.io/projects>

⁴ <https://spring.io/projects/spring-boot#learn>

⁵ <https://cloud.google.com/>

⁶ <https://cloud.google.com/maps-platform/pricing/sheet/>

equivalente. Para o *Geocoding*, atualmente é possível realizar até 40 mil requisições por mês com a cortesia. Uma solicitação para a API deve ter o formato apresentado na Figura 8.

Figura 8 – Formato da requisição para a Geocoding API

```
https://maps.googleapis.com/maps/api/geocode/outputFormat?parameters
```

Fonte: [Google \(2021c\)](#).

Em *outputFormat*, deve-se informar o formato que deseja receber os dados, podendo ser JSON ou XML. Em *parameters*, são informados uma série de parâmetros, que podem ser obrigatórios ou não. Para consultar a latitude e longitude apenas dois parâmetros foram utilizados, ambos são obrigatórios:

- *address*: Endereço que deseja geocodificar.
- *key*: Representa a chave da API. A chave (*key*) é um identificador único que autentica todas as requisições realizadas, para fins de monitoramento do uso e cobrança. Deve ser cadastrada no *Google Cloud Platform*.

Um exemplo de requisição das informações no formato JSON para o endereço “1600 Amphitheatre Parkway, Mountain View, CA” é apresentado na Figura 9.

Figura 9 – Exemplo de uma chamada para a Geocoding API

```
https://maps.googleapis.com/maps/api/geocode/json?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&key=YOUR_API_KEY
```

Fonte: [Google \(2021c\)](#).

Como resposta, é retornado o código de status da solicitação e o resultado, que contém informações de endereço geocodificadas e informações de geometria. Dentre esses dados estão a latitude (lat) e longitude (lng), conforme mostrado na Figura 10. Após obtê-los é possível definir o nó mais próximo do endereço em uma tabela de caminhos importada no PostgreSQL.

Figura 10 – Parte da resposta de uma requisição para a Geocoding API

```
"formatted_address" : "1600 Amphitheatre Parkway, Mountain View, CA 94043, USA",  
"geometry" : {  
  "location" : {  
    "lat" : 37.4224764,  
    "lng" : -122.0842499  
  },  
}
```

Fonte: [Google \(2021c\)](#).

3.1.2.4 Maven

Para o gerenciamento de dependências o Maven ⁷ foi escolhido. O Maven é uma ferramenta desenvolvida pela Apache ⁸, que faz a automatização de tarefas, construção final do artefato do projeto e todo o gerenciamento das dependências. Essas funções são descritas em um arquivo XML conhecido como POM (*Project Object Model*).

3.1.3 Tecnologias de banco de dados empregadas

Esta seção apresenta as tecnologias aplicadas para a criação e manipulação do banco de dados. Informações sobre a criação e estrutura da base serão apresentadas na Seção 3.2.

3.1.3.1 PostgreSQL

O PostgreSQL⁹ é um sistema gerenciador de banco de dados (SGBD) objeto-relacional. No banco de dados, optou-se por trabalhar com ele por ser de código aberto (*open source*), de livre utilização e principalmente por conta das excelentes ferramentas e extensões disponíveis para trabalhar com dados geoespaciais. Dentre elas está o PostGIS, que de acordo com o ranking¹⁰ do DB-Engines é a extensão espacial mais utilizada do mundo. A versão escolhida foi a 13.2, que no momento do desenvolvimento do projeto é a versão estável.

Para trabalhar com os dados no PostgreSQL, foram instaladas as ferramentas *osm-convert* e *osm2pgrouting*, e as extensões PostGIS e *pgRouting*. Todas essas ferramentas e extensões são apresentadas na Seção 3.2 referente a base de dados. Ao criar um novo banco de dados no PostgreSQL, uma outra extensão já é criada por padrão, a *plpgsql*. O PL/pgSQL¹¹ é uma linguagem procedural para o PostgreSQL. Com ela é possível agrupar

⁷ <https://maven.apache.org/>

⁸ <https://www.apache.org/>

⁹ <https://www.postgresql.org/>

¹⁰ <https://db-engines.com/en/ranking/spatial+dbms>

¹¹ <https://www.postgresql.org/docs/9.6/plpgsql.html>

blocos de computação e várias consultas dentro do banco de dados, diferente da linguagem SQL, em que cada instrução é executada individualmente.

3.1.3.2 osmconvert

O `osmconvert`¹² é usado para converter e processar arquivos do OpenStreetMap. Ele foi necessário para converter um arquivo *pbk* para o formato *osm*. Os dados de mapa baixados do site *Hot Export Tool*¹³ estão no formato *pbk*, que é mais compactado, e o programa escolhido (`osm2pgrouting`) que faz a importação dos dados para o PostgreSQL trabalha apenas com arquivos de extensão *osm*.

3.1.3.3 osm2pgrouting

Ferramenta de linha de comando que possui a finalidade de importar dados do OSM para um banco de dados PostGIS para o uso com o `pgRouting`. O `osm2pgrouting`¹⁴ foi escolhido por carregar dados OSM no PostgreSQL com os requisitos do `pgRouting`. O seu uso permite a construção automática da topologia de roteamento com os dados de rua. Para todas as bordas contidas nesses dados, as extremidades destas bordas serão conectadas a um nó exclusivo, assim como as outras bordas. Quando todas as arestas estão conectadas aos nós, é formado um grafo, que pode ser usado para roteamento com `pgRouting`.

3.1.3.4 PostGIS

PostGIS¹⁵ é uma extensão de código aberto e de livre utilização que insere suporte a objetos geográficos no PostgreSQL, permitindo, por exemplo, que consultas de localização sejam executadas em SQL. Com base na definição em seu manual¹⁶, o PostGIS possui suporte a índices espaciais, e funções para a análise e processamento de objetos SIG (sistema de informação geográfica). O PostGIS foi criado pela empresa *Refractions Research Inc*, que faz parte da fundação OSGeo¹⁷ (*Open Source Geospatial Foundation*), organização sem fins lucrativos que visa promover a adoção global da tecnologia geoespacial aberta para GIS. Essa extensão foi escolhida para trabalhar sobre o PostgreSQL, que de acordo com Obe e Hsu (2015) possui uma fácil extensibilidade e fornece alguns benefícios importantes, como o suporte transacional, suporte de índices essenciais para objetos espaciais e um gerador de planos de consulta pronto para ser usado. Ainda segundo o autor, com o fornecimento de aprimoramentos de indexação espacial, operadores, funções

¹² <https://wiki.openstreetmap.org/wiki/Osmconvert>

¹³ <https://export.hotosm.org/>

¹⁴ <https://pgrouting.org/docs/tools/osm2pgrouting.html>

¹⁵ <https://postgis.net/>

¹⁶ <https://postgis.net/docs/manual-3.1/>

¹⁷ <https://www.osgeo.org/>

e tipos de dados espaciais para o PostgreSQL, o PostGIS, em conjunto com esses recursos que o PostgreSQL e outros complementos oferecem, torna-se uma ferramenta bastante poderosa para aprender sobre GIS.

O PostgreSQL e PostGIS suportam alguns padrões, como o ANSI SQL, OGC (*Open Geospatial Consortium*) e o SQL Multimedia Spec (SQL/MM). Eles visam padronizar o acesso e distribuição de dados geográficos espaciais, além de definir como podem ser implementados e utilizados em um banco de dados com SQL (OBE; HSU, 2015).

Alternativamente ao PostGIS, há outras opções de bancos de dados espaciais, como o *Oracle Spatial* e *SpatialLite*. O *Oracle Spatial* não dispõe de licença para livre utilização, e o *SpatialLite* baseia-se no SQLite para sua execução, limitando seu uso (o SQLite¹⁸ é um sistema de banco de dados para ser embarcado em aplicações ao invés de um serviço cliente-servidor).

3.1.3.5 pgRouting

O pgRouting¹⁹ é uma extensão que se integra ao PostGIS / PostgreSQL e adiciona roteamento geoespacial e outras funcionalidades de análise de rede ao banco de dados. Ele oferece uma grande quantidade de algoritmos de roteamento, como funções para o cálculo do menor caminho. Citando algumas dessas funções:

- `pgr_floydWarshall`: Algoritmo de Floyd-Warshall
- `pgr_johnson`: Algoritmo de Johnson
- `pgr_aStar`: Algoritmo A*
- `pgr_dijkstra`: Algoritmo de Dijkstra

Uma curiosidade extraída de um workshop²⁰ do pgRouting, é que inicialmente ele se chamava pgDijkstra, porque o único algoritmo que implementava era a busca do caminho mais curto com o Dijkstra. Após a adição de outras funções, a biblioteca foi renomeada para pgRouting. No sistema construído, a função `pgr_dijkstra` foi aplicada, ela se baseia no algoritmo de Dijkstra, retornando o caminho mais curto entre dois pontos.

3.1.4 pgAdmin 4

O pgAdmin 4²¹ é uma ferramenta gráfica aberta que foi instalada com o PostgreSQL. Ele é usado para a administração do PostgreSQL, simplificando a criação, manutenção e o uso de objetos de banco de dados através de uma poderosa interface gráfica.

¹⁸ <https://www.sqlite.org/whentouse.html>

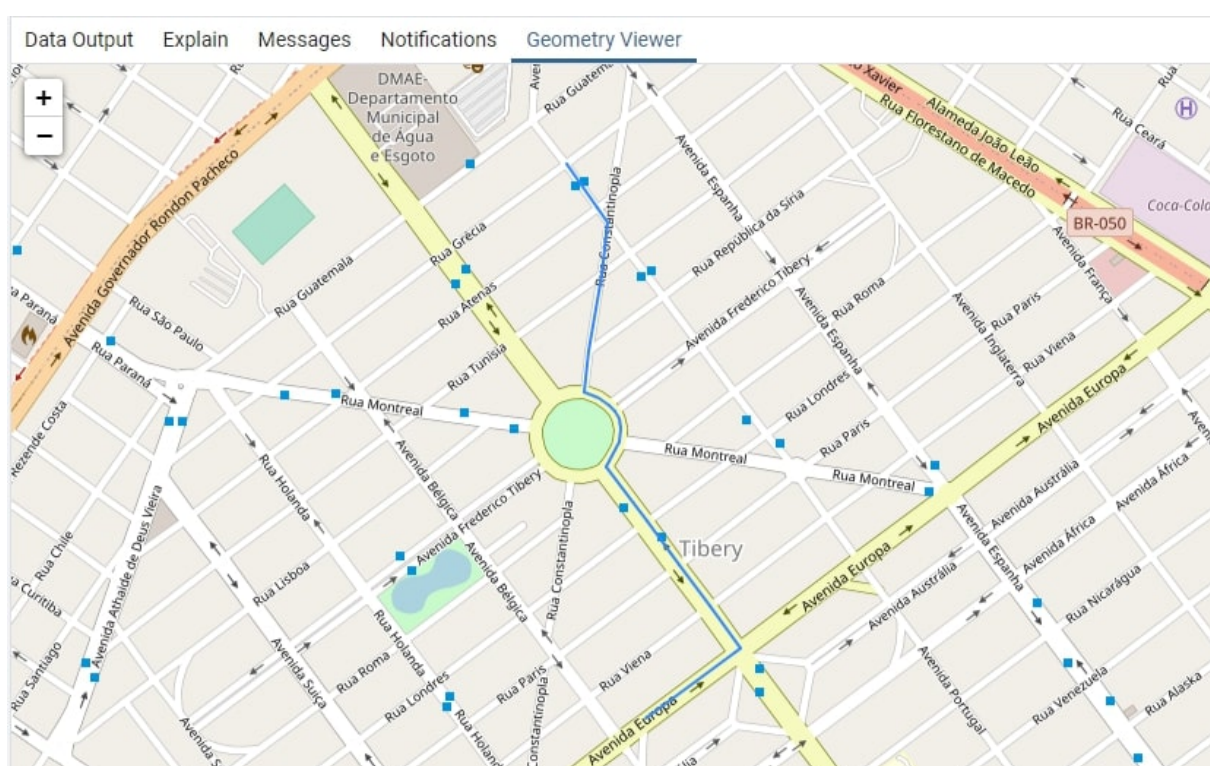
¹⁹ <https://pgrouting.org/>

²⁰ https://workshop.pgrouting.org/2.6/en/chapters/shortest_path.html

²¹ <https://www.pgadmin.org/>

O pgadmin 4 é uma reescrita completa do pgAdmin, foi construído em Python e Javascript / jQuery. Uma funcionalidade bastante interessante foi adicionada na versão 3.3²², inseriram na ferramenta um visualizador de geometria. Quando é executada uma consulta que inclui dados PostGIS, o pgAdmin sobrepõe seus dados no plano de fundo do OpenStreetMap dentro do geometry viewer, permitindo a visualização de dados PostGIS em uma guia separada no próprio pgAdmin. A Figura 11 mostra um exemplo de uso dessa funcionalidade no projeto, em que após o cálculo de uma rota de custo mínimo é possível visualizá-la no próprio banco de dados. Isso auxiliou os testes e comparação de resultados, pois não foi necessário uma ferramenta externa para construir essa visualização.

Figura 11 – Rota calculada sendo visualizada com o *geometry viewer* no pgAdmin 4



Fonte: O autor.

3.1.5 Stack Builder

Stack Builder²³ é um utilitário com interface gráfica que simplifica o download e instalação de módulos para o PostgreSQL. Ao instalar algum módulo por ele, todas as dependências de software são resolvidas automaticamente. O seu uso foi necessário para a instalação do PostGIS, pgRouting e osm2pgrouting.

²² https://www.pgadmin.org/docs/pgadmin4/development/release_notes_3_3.html

²³ https://www.enterprisedb.com/docs/supported-open-source/postgresql/installer/03_using_stackbuilder/

3.2 Base de dados

O modo de criação da base de dados, sua estrutura e as funções que foram usadas para o cálculo do menor caminho são apresentadas nesta seção.

3.2.1 Criação

3.2.1.1 Exportação do mapa de Uberlândia

Atualmente, o site do OpenStreetMap não permite a exportação dos seus dados de uma área muito grande, o limite informado é de 50000 nós. Como o mapa de Uberlândia ultrapassa esse limite, foi utilizado um dos provedores de dados processados²⁴ sugeridos por eles, o *HOT Export Tool*. Ele é um serviço aberto que cria extratos personalizados de dados OSM em vários formatos de arquivo. Os dados referentes ao mapa de Uberlândia foram extraídos com sucesso no formato PBF. O Apêndice A apresenta um passo a passo para a utilização dessa ferramenta.

3.2.1.2 Preparação da base de dados no PostgreSQL

Para trabalhar com roteamento geoespacial é necessário a preparação de uma base de dados no PostgreSQL. Essa preparação seguiu as seguintes etapas:

1. Instalação do PostgreSQL com os componentes pgAdmin 4 e *Stack Builder*.
2. Instalação da extensão PostGIS pelo utilitário *Stack Builder*. Ela que vai inserir suporte a objetos geográficos no PostgreSQL, permitindo que ele seja usado como um banco de dados espacial para sistemas de informações geográficas (GIS). Ao adicionar o PostGIS pelo *Stack Builder*, a extensão pgRouting e a ferramenta osm2pgrouting também serão instalados.
3. Criação de uma base de dados.
4. Inserção das extensões PostGIS e pgRouting na base de dados criada. As extensões permitirão o armazenamento e a manipulação dos dados geoespaciais.

Todas as etapas são detalhadas na Seção B.1 do Apêndice B. Nela mostra-se um passo a passo de como realizar a configuração.

3.2.1.3 Importação do mapa para o banco de dados

Após configurar o *database*, o próximo passo é o de importação. O arquivo baixado pelo *Hot Export Tool* necessitou ser convertido do formato *pbf* para o formato *osm*,

²⁴ https://wiki.openstreetmap.org/wiki/Processed_data_providers

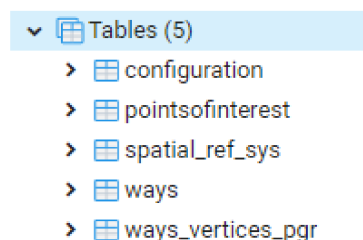
pois o programa escolhido (`osm2pgrouting`) para fazer a importação dos dados para o PostgreSQL trabalha apenas com arquivos de extensão `osm`. A ferramenta usada para a conversão foi o `osmconvert`, a Seção A.2 do Apêndice A detalha o seu uso.

Ao término da conversão de formato, os dados estão prontos para serem integrados à base de dados com a ferramenta de linha de comando `osm2pgrouting`. Conforme visto na seção de apresentação das tecnologias, ela foi escolhida por construir automaticamente uma topologia de roteamento, criando um grafo que pode ser usado para roteamento com o `pgRouting`. O `osm2pgrouting` converte os dados do OpenStreetMap para tabelas e campos no banco de dados. A Seção B.2 do Apêndice B demonstra o uso da ferramenta. Após o processo de importação os dados já estarão prontos para serem trabalhados com o `pgRouting`.

3.2.2 Estrutura da base de dados

A lista de tabelas do banco após os dados serem importados é apresentada na Figura 12. A tabela `spatial_ref_sys` foi criada automaticamente pelo PostGIS, as outras foram inseridas pelo `osm2pgrouting` no momento da importação.

Figura 12 – Lista de tabelas do banco após a importação dos dados



Fonte: O autor.

A seguir será comentado sobre a estrutura e alguns detalhes das cinco tabelas, além de como os dados estão armazenados. Os códigos SQL referentes a criação delas foram extraídos com o `pgAdmin 4`.

3.2.2.1 Tabela *configuration*

Tabela que possui informações sobre o tipo de estrada na tabela de caminhos (`ways`), ajudando na sua interpretação. Os seus dados são derivados do `mapconfig.xml`, arquivo de configuração padrão do `osm2pgrouting`. A Figura 13 exibe alguns dados dessa tabela. Colunas que merecem destaque:

- `tag_id`: Corresponde ao `tag_id` na tabela `ways`. Fornece informações sobre o tipo de caminho.

- `tag_key`: Permite saber a *tag* da chave do caminho, como *road* (estrada) e *cycleway* (ciclovía).
- `tag_value`: Possui a classe específica de estrada. Exemplos: *motorway* (autoestrada), *pedestrian* (pedestre), *residential* (residencial).

Figura 13 – Visualização de alguns dados da tabela `configuration`

	<code>id</code> [PK] integer	<code>tag_id</code> integer	<code>tag_key</code> text	<code>tag_value</code> text	<code>priority</code> double precision	<code>maxspeed</code> double precision	<code>maxspeed_forward</code> double precision	<code>maxspeed_backward</code> double precision	<code>force</code> character (1)
1	1	201	cycleway	lane	[null]		40	40	40 N
2	2	204	cycleway	opposite	[null]		40	40	40 N
3	3	203	cycleway	opposite_lane	[null]		40	40	40 N
4	4	202	cycleway	track	[null]		40	40	40 N
5	5	120	highway	bridleway	[null]		40	40	40 N
6	6	116	highway	bus_guideway	[null]		40	40	40 N
7	7	121	highway	byway	[null]		40	40	40 N

Fonte: O autor.

Da tabela `ways` consegue-se chegar até a `configuration` através da *foreign key* `ways_tag_id_fkey`. Ela é criada na tabela `ways`:

```
1 CONSTRAINT ways_tag_id_fkey FOREIGN KEY (tag_id)
2 REFERENCES public.configuration (tag_id) MATCH SIMPLE
```

Detalhes de criação:

```
1 CREATE TABLE public.configuration
2 (
3     id integer NOT NULL DEFAULT nextval('configuration_id_seq'::
4         regclass),
5     tag_id integer,
6     tag_key text COLLATE pg_catalog."default",
7     tag_value text COLLATE pg_catalog."default",
8     priority double precision,
9     maxspeed double precision,
10    maxspeed_forward double precision,
11    maxspeed_backward double precision,
12    force character(1) COLLATE pg_catalog."default",
13    CONSTRAINT configuration_pkey PRIMARY KEY (id),
14    CONSTRAINT configuration_tag_id_key UNIQUE (tag_id)
15 )
```

3.2.2.2 Tabela *pointsofinterest*

Essa é uma tabela vazia que não é utilizada no projeto. Criada para gerar pontos de interesse, pode ser usada com uma família de funções do pgRouting chamada de *withPoints*²⁵. Detalhes de criação:

```
1 CREATE TABLE public.pointsofinterest
2 (
3     pid bigint NOT NULL DEFAULT nextval('pointsofinterest_pid_seq
4         '::regclass),
5     osm_id bigint,
6     vertex_id bigint,
7     edge_id bigint,
8     side character(1) COLLATE pg_catalog."default",
9     fraction double precision,
10    length_m double precision,
11    tag_name text COLLATE pg_catalog."default",
12    tag_value text COLLATE pg_catalog."default",
13    name text COLLATE pg_catalog."default",
14    the_geom geometry(Point,4326),
15    new_geom geometry(Point,4326),
16    CONSTRAINT pointsofinterest_pkey PRIMARY KEY (pid),
17    CONSTRAINT pointsofinterest_osm_id_key UNIQUE (osm_id)
18 )
```

3.2.2.3 Tabela *spatial_ref_sys*

A *spatial_ref_sys*²⁶ foi criada automaticamente quando a extensão do PostGIS foi adicionada à base. Ela possui os IDs numéricos e as descrições textuais dos sistemas de coordenadas usados no banco de dados espacial. A tabela *spatial_ref_sys* usada pelo PostGIS lista milhares de sistemas de referência espacial (SRS) conhecidos e os detalhes necessários para transformar (reprojetar) entre eles. A Figura 14 apresenta alguns dados dessa tabela. Colunas:

- *srid*: Identificador do sistema de referência espacial dentro do banco de dados.
- *auth_name*: Nome da autoridade que especificou o SRS. Exemplo: EPGS (*European Petroleum Survey Group*).
- *auth_srid*: ID do SRS conforme definido pela autoridade do *auth_name*.
- *srttext*: Representação *Well-Known Text* (WKT) do sistema de referência espacial.

²⁵ <https://docs.pgrouting.org/3.1/en/withPoints-family.html>

²⁶ https://postgis.net/docs/manual-3.1/postgis-br.html#spatial_ref_sys

- `proj4text`: Coluna que contém uma *string* de definição de coordenada Proj4 para um SRID em específico. Proj4 é uma biblioteca utilizada pelo PostGIS que fornece a capacidade de transformação de coordenadas.

Existem vários tipos de sistemas de referência usados no mundo, tais sistemas são conhecidos também como datum. Os dados inseridos na base usam o datum WGS (*World Geodetic System*) 84, que é identificado pelo EPSG com o SRID (*Spatial Reference System Identifier*) 4326. O SRID define todos os parâmetros do sistema de coordenadas geográficas e projeção de um mapa em apenas um número. O WGS 84 é o sistema de coordenadas do OpenStreetMap²⁷ e também do Sistema de Posicionamento Global (GPS). Ele é geocêntrico e tem validade global. Ser geocêntrico significa que o seu ponto de origem coincide com o centro de massa da terra.

Figura 14 – Visualização do SRID 4326 na tabela `spatial_ref_sys`

	srid	auth_name	auth_srid	srtxt	proj4text
	[PK] integer	character varying (256)	integer	character varying (2048)	character varying (2048)
2157	4326	EPSG	4326	GEOGCS["WGS 84",DATUM["WGS_1984"...	+proj=longlat +datum=WGS84 +no_defs
2158	4327	EPSG	4327	GEOGCRS["WGS 84 (3D)",DATUM["Worl...	+proj=longlat +datum=WGS84 +no_defs
2159	4328	EPSG	4328	GEOCCS["WGS 84 (geocentric)",DATUM[...	+proj=geocent +datum=WGS84 +units=...
2160	4329	EPSG	4329	GEOGCRS["WGS 84",DATUM["World Geo...	+proj=longlat +datum=WGS84 +no_defs
2161	4330	EPSG	4330	GEOCCS["ITRF88 (geocentric)",DATUM["...	+proj=geocent +ellps=GRS80 +units=m ...
2162	4331	EPSG	4331	GEOCCS["ITRF89 (geocentric)",DATUM["...	+proj=geocent +ellps=GRS80 +units=m ...
2163	4332	EPSG	4332	GEOCCS["ITRF90 (geocentric)",DATUM["...	+proj=geocent +ellps=GRS80 +units=m ...

Fonte: O autor.

Detalhes de criação:

```

1 CREATE TABLE public.spatial_ref_sys
2 (
3     srid integer NOT NULL,
4     auth_name character varying(256) COLLATE pg_catalog."default"
5     ,
6     auth_srid integer,
7     srtxt character varying(2048) COLLATE pg_catalog."default",
8     proj4text character varying(2048) COLLATE pg_catalog."default
9     ",
10    CONSTRAINT spatial_ref_sys_pkey PRIMARY KEY (srid),
11    CONSTRAINT spatial_ref_sys_srid_check CHECK (srid > 0 AND
12        srid <= 998999)
13 )

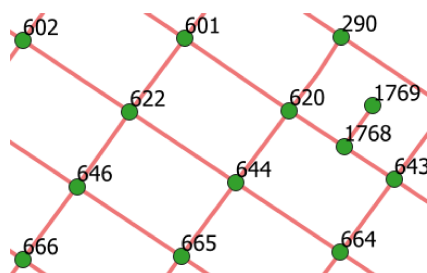
```

²⁷ https://wiki.openstreetmap.org/wiki/Converting_to_WGS84

3.2.2.4 Tabela *ways_vertices_pgr*

Com os dados importados, criou-se uma rede modelada como um grafo, possuindo vértices e arestas. A Figura 15 extraída da base de dados do projeto com o programa QGIS²⁸ mostra uma rede de ruas representada por um grafo.

Figura 15 – Representação dos nós em uma rede de ruas



Fonte: O autor.

Os pontos verdes nas extremidades de cada linha são os nós, ou vértices. Observe que cada nó possui um número de identificação (ID). Os segmentos de linha que estão vermelho e fazem a ligação entre dois nós são caminhos, em que grafos são chamados de arestas. Um caminho pode ser uma avenida, estrada, rua, dentre outros. A tabela *ways_vertices_pgr* contém todos os nós onde as arestas se juntam, e também nós que representam as extremidades dessas arestas. Esquinas onde ruas se cruzam, o início ou fim de uma avenida, são exemplos de vértices que correspondem a essas informações na tabela. Cada vértice na base de dados possui uma latitude e longitude, coordenadas que informam sua localização geograficamente. Os nós são armazenados nessa tabela como geometrias de ponto. Principais colunas:

- *id*: Esse número de identificação é usado como referência na tabela *ways* para os campos *source* e *target*. *Source* é o nó de início de um segmento de linha, e *target* representa o nó final.
- *lon*: Longitude do nó para descrever sua localização.
- *lat*: Latitude do nó para descrever sua localização.
- *the_geom*: Conforme informado no manual do PostGIS²⁹, ele suporta todos os objetos GIS e funções que foram especificados pelo OpenGIS Consortium. Essa especificação definiu duas formas de se expressar objetos espaciais em um banco de dados, a *Well-Known Text* (WKT) e a forma *Well-Known Binary* (WKB). O WKT fornece uma representação textual e o WKB uma representação binária. Os dois possuem informação sobre o tipo do objeto e as coordenadas que os formam. Exemplos das

²⁸ <https://qgis.org/en/site/>

²⁹ https://postgis.net/docs/postgis_usage.html

representações de textos (WKT) dos objetos espaciais são: POINT(0 0), LINESTRING(0 0,1 1,1 2), POINT Z (0 0 0), POINT ZM (0 0 0 0), LINESTRING(0 0,1 1,1 2), POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1)), MULTIPOINT((0 0),(1 2)), MULTIPOINT Z ((0 0 0),(1 2 3)), MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4)), MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1))), GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4)). Para armazenar objetos espaciais, a OpenGIS também pede a inclusão de um identificador de sistema de referência espacial (SRID).

A coluna *the_geom* é do tipo *geometry*, um tipo de dados espacial do PostGIS utilizado para representar um recurso em sistemas de coordenadas cartesianas. Segundo [Obe e Hsu \(2015\)](#), o *geometry* tem uma estrutura hierárquica, em que pode e deve ser declarado subtipos. Para o *geometry*, os subtipos são modificadores de tipo, alguns exemplos são: POINT, LINESTRING e POLYGON. Um exemplo de declaração do tipo *geometry* em uma base de dados com PostGIS é `geometry(POINT,4326)`. O *geometry* é o tipo de dados, *Point* é um subtipo modificador de tipo e 4326 é o modificador de tipo SRID. A *the_geom* da tabela `ways_vertices_pgr` usa essa mesma declaração do exemplo, `geometry (POINT,4326)`. *POINT* é um ponto no espaço especificado por suas coordenadas X e Y. Um único local na Terra pode ser representado por um ponto espacial. Exemplo de um *POINT* contendo duas coordenadas no formato WKT: POINT (-48.2486036 -18.8790378). As coordenadas X e Y se referem-se às colunas *lon* e *lat*, respectivamente. Além do *POINT*, tem-se o número 4326, que é o SRID do WGS 84, sistema de referência espacial utilizado no projeto.

Detalhes de criação:

```

1 CREATE TABLE public.ways_vertices_pgr
2 (
3     id bigint NOT NULL DEFAULT nextval('ways_vertices_pgr_id_seq'
4         ::regclass),
5     osm_id bigint,
6     eout integer,
7     lon numeric(11,8),
8     lat numeric(11,8),
9     cnt integer,
10    chk integer,
11    ein integer,
12    the_geom geometry(Point,4326),
13    CONSTRAINT ways_vertices_pgr_pkey PRIMARY KEY (id),
14    CONSTRAINT ways_vertices_pgr_osm_id_key UNIQUE (osm_id)

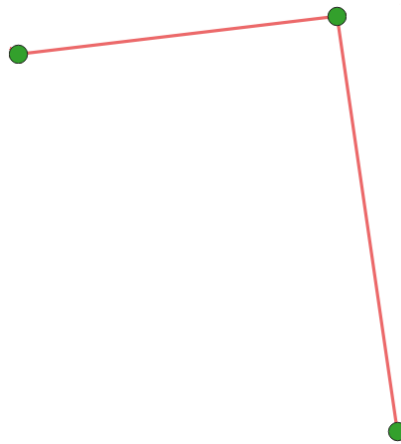
```

3.2.2.5 Tabela *ways*

Essa é a tabela de todos os segmentos de linhas individuais que fazem a ligação entre dois nós. Em um grafo, são representados por arestas. Ela possui um nó de origem (*source*) e de destino (*target*) para cada uma das arestas. Esses nós referenciam o campo *id* da tabela *ways_vertices_pgr*. Principais colunas:

- *tag_id*: Identificador do tipo de via para o segmento de linha. É chave estrangeira da coluna *tag_id* da tabela *configuration*.
- *length*: Comprimento do segmento de linha em graus.
- *length_m*: Comprimento do segmento de linha em metros.
- *name*: Nome da via.
- *source*: Nó inicial do segmento de linha.
- *target*: Nó final do segmento de linha.
- *cost*: Representa o comprimento em graus do segmento de linha, o peso da aresta (origem, destino). Quando está negativo a aresta não existe, não faz parte do grafo.
- *reverse_cost*: Representa o peso da aresta (destino, origem). É o custo para ir na direção oposta do segmento de linha.
- *oneway*: Indica se a via é unidirecional ou bidirecional. Se aplica apenas para roteamento de rotas com veículos.
- *x1*: Coordenada X do vértice de origem. Representa a longitude.
- *y1*: Coordenada Y do vértice de origem. Representa a latitude.
- *x2*: Coordenada X do vértice de destino. Representa a longitude.
- *y2*: Coordenada Y do vértice de destino. Representa a latitude.
- *the_geom*: Coluna do tipo *geometry* que armazena um subtipo *Linestring* e o SRID 4326. O subtipo *Linestring* é uma cadeia de linhas, um caminho entre locais. Segundo Obe e Hsu (2015), linhas retas que fazem a conexão entre dois ou mais pontos formam cadeias de linhas, e quando se tem uma linha individual entre pontos ela é chamada de segmento. Um segmento não é um subtipo do Postgis, ele continua sendo uma *linestring*, mas com apenas um segmento. A Figura 16 exemplifica uma *linestring*.

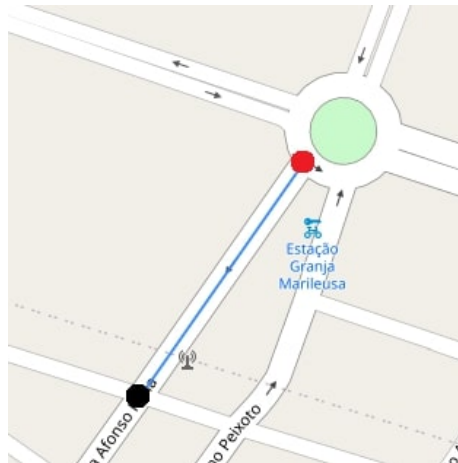
Figura 16 – Exemplo de uma linestring



Fonte: O autor.

Na tabela *ways*, uma *linestring* contém as coordenadas dos nós de origem e destino, como: `LINESTRING (-48.2486036 -18.8790378, -48.2497742 -18.8806218)`. Os números `-48.2486036` e `-18.8790378` são a latitude e longitude de origem, `-48.2497742` e `-18.8806218` referem-se à latitude e à longitude do ponto de destino. Na Figura 17 é possível visualizar na cor azul a linha projetada criada para ligar esses dois nós. O ponto na cor vermelha é a origem e o ponto na cor preta é o destino.

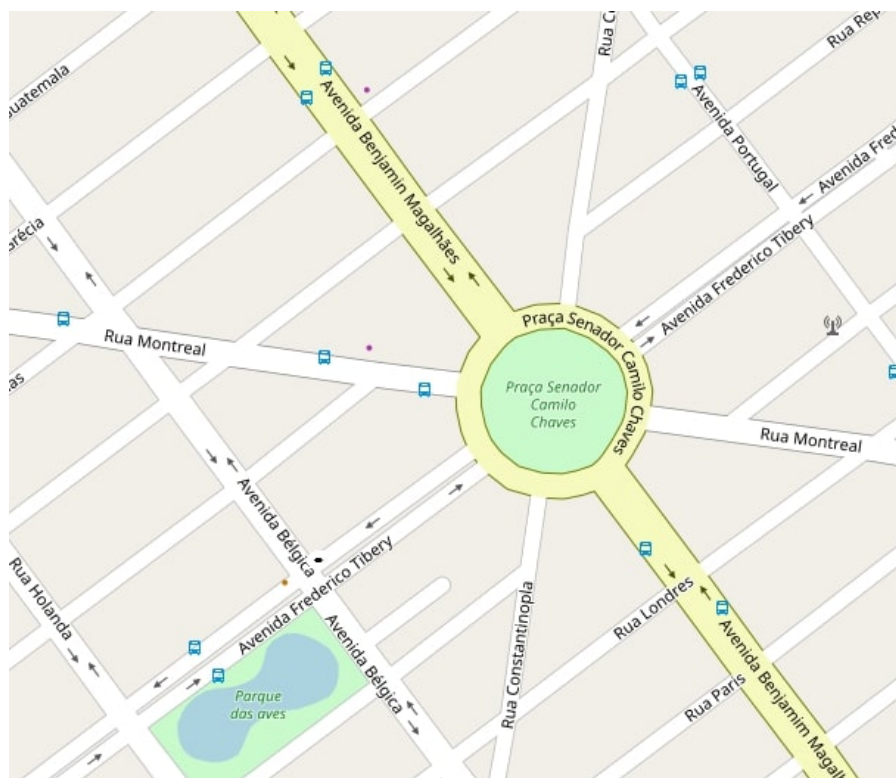
Figura 17 – Linha que liga um nó e origem e outro de destino



Fonte: O autor.

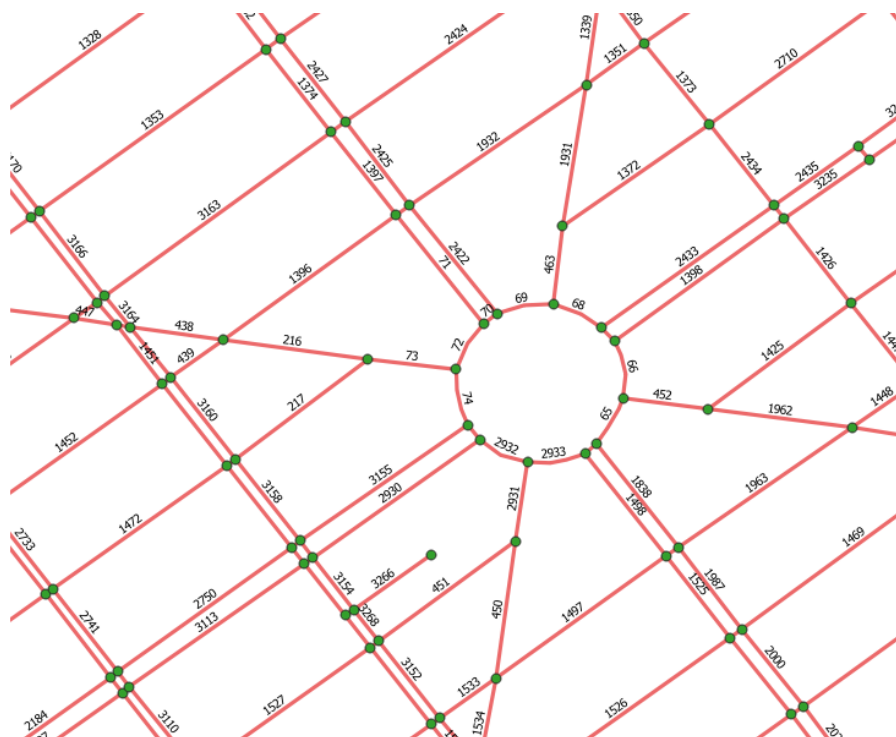
A Figura 18 mostra uma região de Uberlândia retirada do OpenStreetMap, e a Figura 19 exibe sua representação no QGIS como uma rede de caminhos em forma de grafo. Cada caminho do mapa é mostrado como uma linha na cor vermelha ligando dois nós que estão na cor verde. Os números em cada aresta é o seu identificador na tabela *ways*.

Figura 18 – Região da cidade de Uberlândia visualizada no OpenStreetMap



Fonte: [OpenStreetMap](https://www.openstreetmap.org) (2021c).

Figura 19 – Região da cidade de Uberlândia visualizada no QGIS



Fonte: O autor.

Detalhes de criação:

```
1 CREATE TABLE public.ways
2 (
3     gid bigint NOT NULL DEFAULT nextval('ways_gid_seq'::regclass)
4     ,
5     osm_id bigint,
6     tag_id integer,
7     length double precision,
8     length_m double precision,
9     name text COLLATE pg_catalog."default",
10    source bigint,
11    target bigint,
12    source_osm bigint,
13    target_osm bigint,
14    cost double precision,
15    reverse_cost double precision,
16    cost_s double precision,
17    reverse_cost_s double precision,
18    rule text COLLATE pg_catalog."default",
19    one_way integer,
20    oneway text COLLATE pg_catalog."default",
21    x1 double precision,
22    y1 double precision,
23    x2 double precision,
24    y2 double precision,
25    maxspeed_forward double precision,
26    maxspeed_backward double precision,
27    priority double precision DEFAULT 1,
28    the_geom geometry(LineString,4326),
29    CONSTRAINT ways_pkey PRIMARY KEY (gid),
30    CONSTRAINT ways_source_fkey FOREIGN KEY (source)
31        REFERENCES public.ways_vertices_pgr (id) MATCH SIMPLE
32        ON UPDATE NO ACTION
33        ON DELETE NO ACTION,
34    CONSTRAINT ways_source_osm_fkey FOREIGN KEY (source_osm)
35        REFERENCES public.ways_vertices_pgr (osm_id) MATCH SIMPLE
36        ON UPDATE NO ACTION
37        ON DELETE NO ACTION,
38    CONSTRAINT ways_tag_id_fkey FOREIGN KEY (tag_id)
39        REFERENCES public.configuration (tag_id) MATCH SIMPLE
40        ON UPDATE NO ACTION
41        ON DELETE NO ACTION,
```

```

41     CONSTRAINT ways_target_fkey FOREIGN KEY (target)
42         REFERENCES public.ways_vertices_pgr (id) MATCH SIMPLE
43         ON UPDATE NO ACTION
44         ON DELETE NO ACTION ,
45     CONSTRAINT ways_target_osm_fkey FOREIGN KEY (target_osm)
46         REFERENCES public.ways_vertices_pgr (osm_id) MATCH SIMPLE
47         ON UPDATE NO ACTION
48         ON DELETE NO ACTION
49 )

```

3.2.3 Cálculo do menor caminho

Após o entendimento de como os dados foram armazenados, foi feito um estudo de como trabalhar com as funções de roteamento do pgRouting, possibilitando encontrar o menor caminho entre dois nós que estão na base de dados. Dentre as várias funções fornecidas por ele, foi escolhida a *pgr_dijkstra*. Ela utiliza o algoritmo Dijkstra para calcular caminhos de custo mínimo de um vértice inicial(*source*) até um vértice final(*target*).

A *pgr_dijkstra* fornece várias possibilidades, calcular a rota de um vértice para outro vértice(*one to one*), um para muitos(*one to many*), muitos para um(*many to one*), muitos para muitos(*many to many*) e combinações. Pensando no projeto em que a funcionalidade pretendida é obter uma rota de menor caminho entre dois nós e que o custo é a distância, foi utilizada a *one to one*, que possui a seguinte assinatura:

```

pgr_dijkstra(Edges SQL, start_vid, end_vid [, directed])
RETURNS SET OF (seq, path_seq, node, edge, cost, agg_cost)
OR EMPTY SET

```

Exemplo de uso indo do nó 5530 ao nó 5625:

```

1 SELECT * FROM pgr_dijkstra(
2     'SELECT id, source, target, cost, reverse_cost FROM
3     edge_table',
4     5530, 5625
5 );

```

O resultado dessa consulta é ilustrado na Figura 20. Colunas retornadas:

- seq: Valor sequencial que se inicia em 1.
- path_seq: Identificador de sequência do caminho a percorrer para ir da origem até o destino. Inicia-se com o valor 1, indicando o início do caminho.
- node: Identificador do nó.

- `edge`: Identificador da aresta, faz referência ao id da tabela de caminhos.
- `cost`: Custo para percorrer do nó atual até o próximo nó da sequência de caminhos (`path_seq`).
- `agg_cost`: Custo agregado.

Figura 20 – Resultado da consulta utilizando a função `pgr_dijkstra`

	Data Output	Explain	Messages	Notifications	Geometry Viewer	
	seq integer	path_seq integer	node bigint	edge bigint	cost double precision	agg_cost double precision
1	1	1	5530	2119	.0008540956211138033	0
2	2	2	5545	2124	.0008430281608582024	.0008540956211138033
3	3	3	5560	31167	.0008664618283574494	.0016971237819720058
4	4	4	2288	22235	.0008266289191659238	0.002563585610329455
5	5	5	5585	188	.0008335002879412247	0.003390214529495379
6	6	6	460	23104	.0008662483997096936	0.004223714817436604
7	7	7	7861	22255	.0010614188287424694	0.005089963217146297
8	8	8	5625	-1	0	0.006151382045888766

Fonte: O autor.

A função recebe como entrada o ID da tabela de arestas, nó de origem (*source*), nó de destino (*target*), custo (*cost*) e custo reverso (*reverse_cost*). No exemplo não foi especificado o campo *directed* que informa se deseja considerar o grafo como direcionado ou não direcionado. Quando não é informado ele é considerado como *TRUE*. Ser direcionado quer dizer que o pgRouting irá considerar o sentido das vias, se elas são unidirecionais ou bidirecionais. Em um roteamento de rotas para veículos a direção tem que ser *TRUE*, pois ele não poderá trafegar na contramão de uma via. Já para pedestres, a direção da via não importa, pode ir e voltar em qualquer sentido.

Para atender às necessidades do sistema, foi utilizado a `pgr_dijkstra` em duas consultas distintas, uma para pedestres e outra para veículos. A consulta abaixo é usada quando o usuário deseja receber a rota para ser seguida por um pedestre:

```

1 select
2     *
3 from
4     pgr_dijkstra('
5         SELECT gid as id,
6             source,
7             target,
8             ST_Length(the_geom::GEOGRAPHY) AS cost,
9             ST_LENGTH(the_geom::GEOGRAPHY) AS reverse_cost

```

```

10         FROM ways', ID_VERTICE_ORIGEM, ID_VERTICE_DESTINO,
           directed := false ) as di
11 join ways on
12     di.edge = ways.gid
13 order by
14     seq;

```

Nessa consulta é feito um *join* com a tabela *ways*. Dessa forma, é possível pegar qualquer informação dos caminhos que a resposta da *pgr_dijkstra* não consegue retornar, como por exemplo o campo *name*. Com o *directed := false* é passado para o *pgRouting* que ele não deve considerar o sentido das vias, os dados serão tratados como um grafo não direcionado. Na base de dados criada, o identificador da tabela de arestas (*ways*) não tem o nome *id*, o seu identificador é a coluna *gid*. Nesse caso, foi necessário alterar de *id* para *gid AS id*. Outra mudança foi alterar os valores dos campos *cost* e *reverse_cost* de graus para metros utilizando a função *ST_Length* do PostGIS. *ID_VERTICE_ORIGEM* e *ID_VERTICE_DESTINO* devem ser substituídos pelos nós de origem e de destino.

A próxima consulta é referente ao cálculo do menor caminho para veículos. Nesse caso o sentido das vias deve ser considerado.

```

1 select
2     *
3 from
4     pgr_dijkstra('
5         SELECT gid AS id,
6             source,
7             target,
8             ST_Length(the_geom::GEOGRAPHY) AS cost,
9             CASE
10                WHEN oneway='YES' THEN -1
11                ELSE ST_Length(the_geom::GEOGRAPHY)
12            END AS reverse_cost
13         FROM   ways', ID_VERTICE_ORIGEM, ID_VERTICE_DESTINO,
           directed := true ) as di
14 join ways on
15     di.edge = ways.gid
16 order by
17     seq ;

```

Para informar ao *pgRouting* que o sentido das vias deve ser obedecido foi necessário passar o *directed* como verdadeiro (*directed := true*). Outra alteração em relação a consulta para pedestres foi a verificação do campo *oneway*. Caso ele tenha o valor *YES*, quer dizer que o caminho é de mão única, portanto o *reverse_cost* é alterado para o valor -1. Com

esse valor negativo, a aresta torna-se intransitável na direção oposta do tráfego.

No sistema desenvolvido, os usuários deverão passar dois endereços reais, o local e o destino. É necessário encontrar o nó mais próximo dos locais desejados, e em seguida passar para a função *pgr_dijkstra* que recebe os nós *source* e *target*. A consulta a seguir faz exatamente isso, a partir da latitude e longitude de um local no mapa ela retorna o ponto mais próximo.

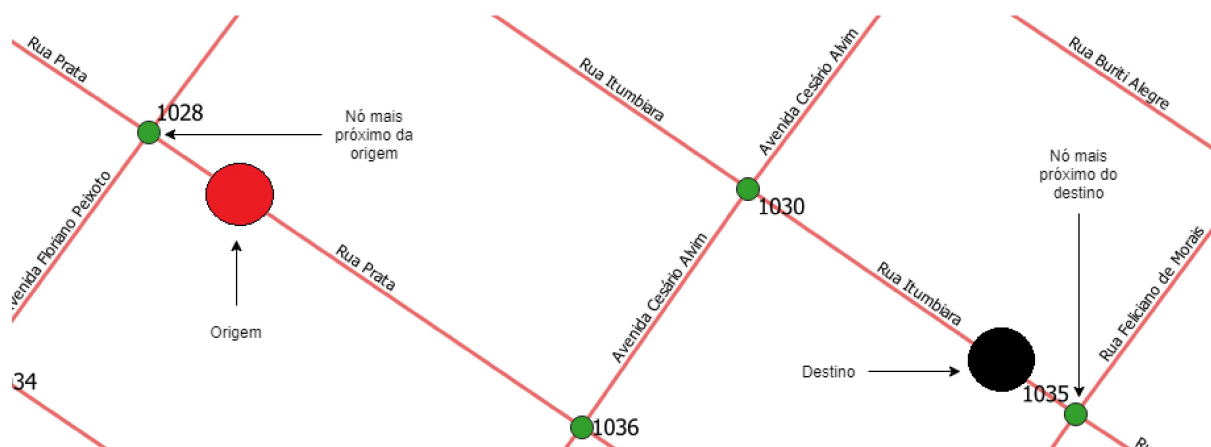
```

1 select
2     source
3 from
4     ways
5 order by
6     ST_Distance( ST_StartPoint(the_geom), ST_SetSRID(ST_MakePoint
7     (LONGITUDE, LATITUDE), 4326), true ) asc
8 limit 1;

```

LONGITUDE e *LATITUDE* devem ser substituídos pelas coordenadas do local, por exemplo: (-48.2683066, -18.8982462). A Figura 21 exemplifica o uso dessa consulta. Um usuário deseja ir de um endereço localizado na rua Prata até outro na rua Itumbiara. Os dois locais estão marcados na imagem, onde o ponto vermelho é a origem e o ponto preto é o destino. O primeiro passo é encontrar a latitude e longitude dos dois endereços, e depois passar para a consulta encontrar o nó mais próximo. Após executar a consulta, o nó mais próximo da origem será o de id 1028, e o do destino será o de id 1035.

Figura 21 – Exemplificação de uma busca pelo nó mais próximo dos pontos de origem e de destino



Fonte: O autor.

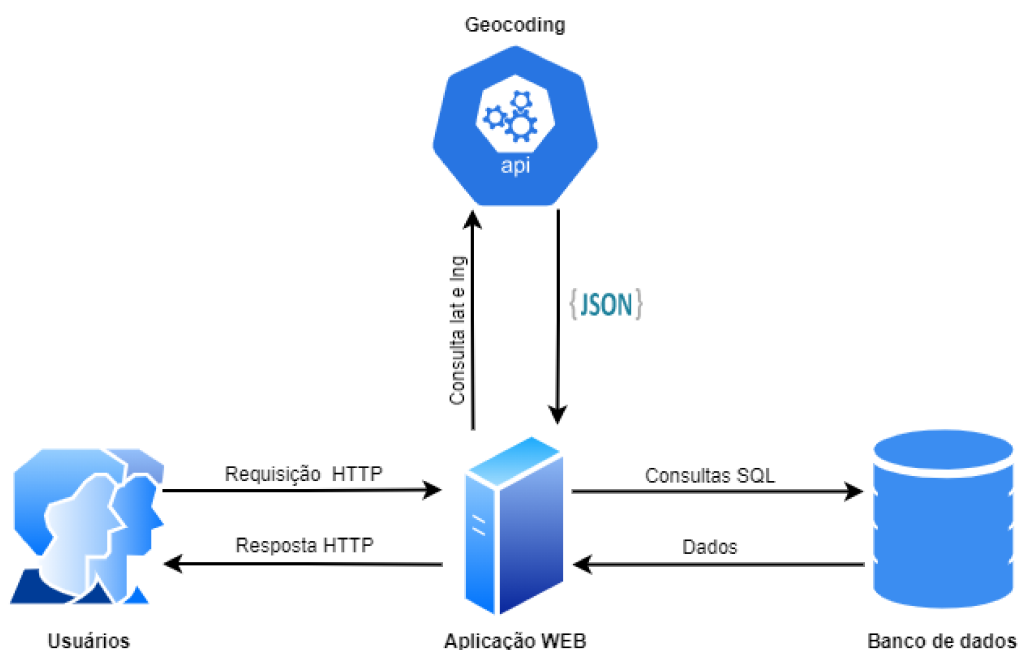
3.3 Desenvolvimento do sistema

Depois das etapas de pesquisa, escolha das tecnologias, estruturação da base de dados e testes com as consultas para obter o menor caminho, iniciou-se a implementação do projeto visando resolver o problema apresentado no Capítulo 1.1. Conforme detalhado na Seção 3.1, as tecnologias utilizadas no desenvolvimento foram:

- *Back-end: Java, Geocoding API, Maven, Spring Framework e Spring Boot.*
- *Front-end: HTML, Thymeleaf, Bootstrap e Material Icons.*

O fluxograma de comunicação do sistema foi pensado e definido conforme apresentado na Figura 22.

Figura 22 – Fluxograma do sistema



Fonte: O autor.

1. Por meio de um navegador WEB os usuários enviam requisições HTTP para a aplicação. As requisições devem conter os endereços da origem e do destino.
2. O sistema recebe esses dados e se comunica com a API *Geocoding*, enviando os endereços.
3. A API retorna um JSON contendo a latitude e a longitude exata dos locais.
4. Com a latitude e longitude, a aplicação consegue usar as consultas descritas na Seção 3.2.3 e encontrar uma rota de menor caminho entre dois endereços fornecidos pelo usuário.

5. A aplicação recebe os resultados da consulta de custo mínimo, processa os dados e retorna para os usuários via resposta HTTP. A resposta é visualizada no navegador WEB.

A *Geocoding* é de grande importância para o projeto. Essa API recebe os endereços de origem e de destino passados pelo usuário e retorna as suas coordenadas geográficas, a latitude e longitude. A primeira consulta que o sistema realiza no banco de dados é para encontrar os nós mais próximos da origem e do destino, por isso é necessário ter a latitude e longitude. Na próxima consulta que é para gerar a rota, os identificadores dos nós encontrados serão passados para a função *pgr_dijkstra* e o menor caminho será gerado. Exemplo de uma requisição para o *Geocoding* solicitando as coordenadas da Avenida João Naves de Ávila, número 1331, bairro Tibery:

```
<https://maps.googleapis.com/maps/api/geocode/json?address=1331+Avenida+Joao+Naves+de+Avila,+Tibery,+Uberlandia&key=API\_KEY>
```

No lugar de *API_KEY* deve ser passada a chave cadastrada no Google Cloud Platform, ela autentica as requisições. Como resposta são retornadas informações de endereço geocodificado e de geometria. O projeto faz o uso de apenas duas dessas informações, a latitude e longitude:

```
"location" : {  
  "lat" : -18.9093886,  
  "lng" : -48.2603495  
},
```

Como apresentado na Seção 3.2.3, é possível gerar rotas para pedestres ou veículos, basta modificar a consulta que utiliza a função *pgr_dijkstra*. O sistema foi construído para permitir que os usuários escolham entre esses dois tipos de locomoção. Dependendo da opção escolhida na tela inicial, a consulta referente a ela será realizada na base de dados.

3.4 Interface e funcionamento

Esta seção tem o objetivo de mostrar a interface e o funcionamento do sistema que foi desenvolvido. A Figura 23 representa a tela inicial da aplicação, sendo acessada pelo endereço <http://localhost:8080/menorCaminho>. A parte da esquerda da tela é referente aos dados de origem, a da direita refere-se aos dados de destino. Logo abaixo tem a caixa de seleção com os dois tipos de locomoção (pedestre e veículo), e, também, o botão para calcular rota.

Figura 23 – Tela inicial da aplicação

Calcular o Menor caminho

localhost:8080/menorCaminho

Calcular Menor caminho

Local de origem

Local de destino

Endereço

Endereço

Número

Número

Bairro

Bairro

Uberlândia

Uberlândia

Pedestre Veículo

Calcular

Fonte: O autor.

Para calcular o menor caminho entre dois locais o usuário tem que preencher os campos com endereço de origem e o endereço de destino, além de definir o tipo de locomoção. Caso o usuário queira percorrer a rota a pé, deve deixar marcada a opção “Pedestre”. Se desejar receber a rota calculada para veículos, como carros e motos, deverá marcar a opção “Veículos”. Por padrão, a opção de “Pedestre” já fica selecionada desde o carregamento da página. Campos do formulário que devem ser preenchidos:

- Endereço: Digite o endereço do lugar, como por exemplo “Avenida Afonso Pena” ou “Rua Viena”.
- Número: Número do endereço.
- Bairro: Nome do bairro.
- Uberlândia: Como o projeto trabalhou apenas com o mapa de Uberlândia em sua base de dados, esse campo já vem preenchido e não pode ser alterado.

Todos esses campos são obrigatórios. Como mostra a Figura 24, caso algum deles não seja informado, ocorrerá um erro na tela comunicando a obrigatoriedade do seu preenchimento. Outra validação para não deixar dados inválidos serem enviados para o sistema é referente ao número do endereço, só é permitido enviar números e eles devem ser positivos.

Na Figura 25 há um exemplo com os dados preenchidos de forma correta. Foram pegos dois endereços válidos quaisquer para exemplificar o uso da aplicação. Veículo foi definido como o meio de locomoção. Ao clicar em “Calcular”, os campos do formulário serão validados, se não estiverem com erro, uma rota de custo mínimo será calculada entre o local de origem e de destino.

Figura 24 – Demonstração da validação de campos

The screenshot shows a web browser window with the URL `localhost:8080/menorCaminho/calculador`. The page title is "Calcular Menor caminho". The form is divided into two columns: "Local de origem" and "Local de destino".

Local de origem:

- Endereço: (Error: O campo rua é obrigatório)
- CEP:
- Tibery:
- Uberlândia:

Local de destino:

- Rua Londres:
- CEP:
- Bairro: (Error: O campo bairro é obrigatório)
- Uberlândia:

Below the form, there are radio buttons for "Pedestre" (selected) and "Veículo". A blue "Calcular" button is at the bottom.

Fonte: O autor.

Figura 25 – Exemplo com os dados preenchidos de forma correta

The screenshot shows the same web browser window as Figure 24, but with the form fields filled correctly. The "Local de origem" fields contain "Avenida Benjamim Magalhães", "36", "Tibery", and "Uberlândia". The "Local de destino" fields contain "Rua Londres", "285", "Tibery", and "Uberlândia". The "Pedestre" radio button is selected, and the "Calcular" button is visible at the bottom.

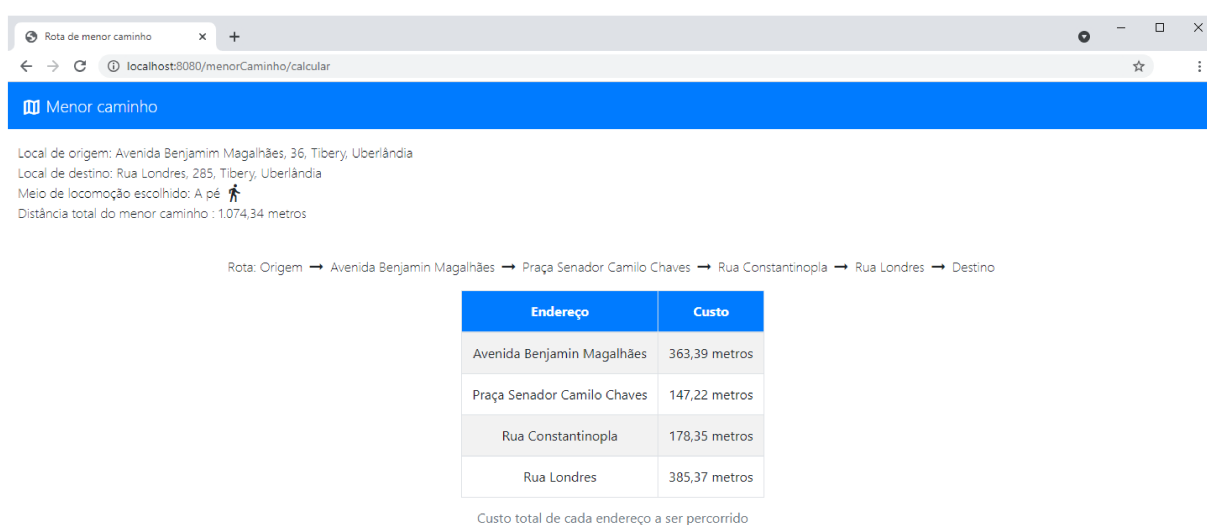
Fonte: O autor.

A Figura 26 demonstra o resultado final do cálculo do menor caminho entre os dois endereços usando o algoritmo de Dijkstra do pgRouting. Na tela são apresentadas para o usuário as seguintes informações:

- Local de origem: Endereço de origem informado na tela inicial.
- Local de destino: Endereço de destino informado na tela inicial.
- Meio de locomoção escolhido: Meio de locomoção definido na tela inicial. Nesse caso a escolha foi “Pedestre”, o que significa que o usuário deseja percorrer a rota a pé.
- Distância total do menor caminho: Custo mínimo total em metros do local de origem até o local de destino. Esse custo é referente a rota de menor caminho que foi calculada.

- Rota: Representação da rota que o usuário deverá seguir para ir da origem até o destino com um custo mínimo. Nesse exemplo em que ele é um pedestre, precisa começar a sua rota indo da origem até a Avenida Benjamin Magalhães, deverá seguir nela até a Praça Senador Camilo Chaves, depois ir pela rua Constantinopla até chegar na rua Londres, que é o destino final.
- Tabela com endereço e custo: Tabela informativa detalhando o custo total de cada endereço. A distância total é a soma de cada um desses valores.

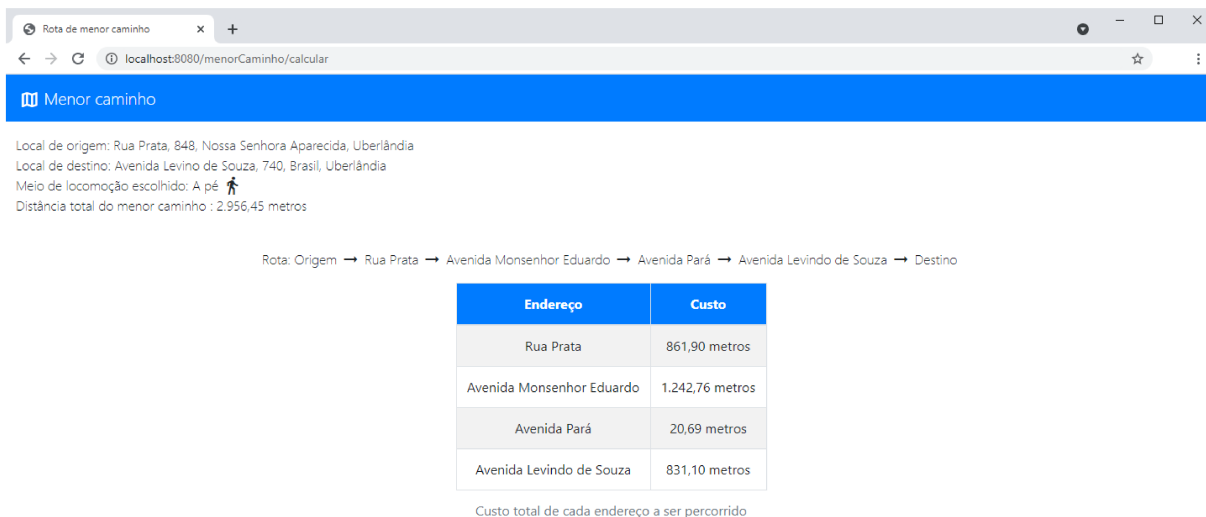
Figura 26 – Resposta da aplicação contendo as informações de menor caminho



Fonte: O autor.

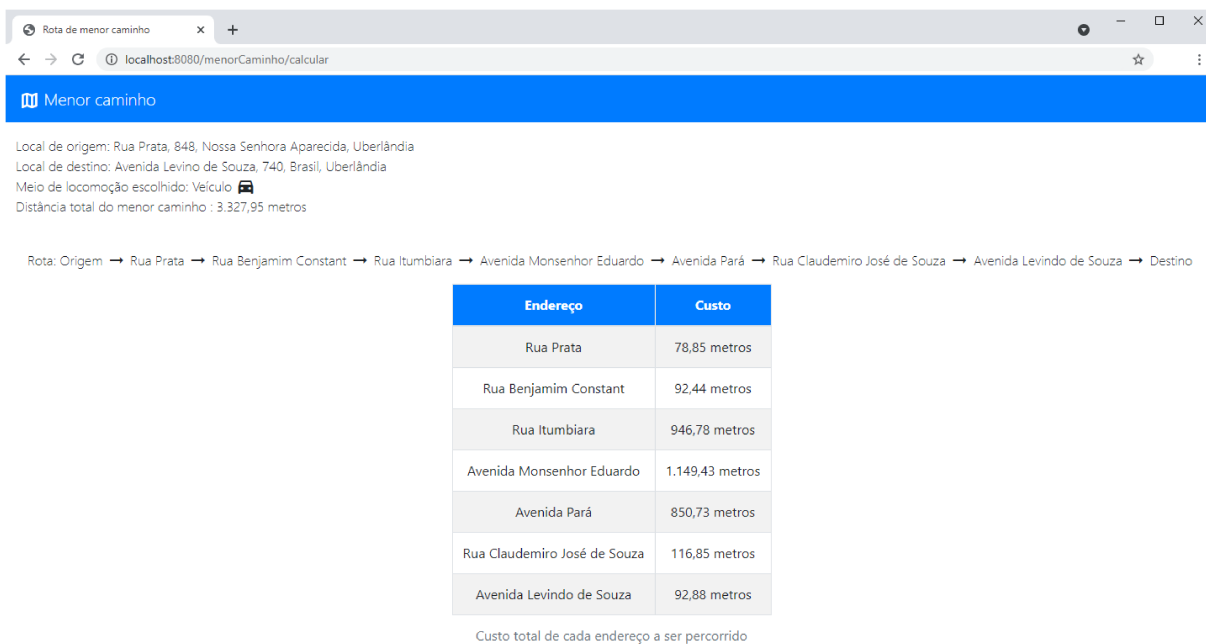
A escolha correta do tipo de locomoção é de suma importância. Para um pedestre não importa o sentido das vias, já para um veículo o sentido delas é essencial, pois não podem trafegar pela contramão. O sistema faz essa tratativa, realiza um cálculo diferente para pedestres e veículos. Isso é demonstrado com as Figuras 27 e 28, em que uma mesma rota é calculada para os dois diferentes tipos de locomoção. Pode-se observar que apesar de terem a mesma origem e destino, os dois caminhos e distâncias se diferem. A rota calculada para pedestres possui um custo mínimo de 2.956,45 metros e passa apenas por quatro vias distintas. Já a rota para veículos tem um custo mínimo de 3.327,95 metros e um total de 7 vias.

Figura 27 – Exemplo de cálculo de rota de custo mínimo para pedestres



Fonte: O autor.

Figura 28 – Exemplo de cálculo de rota de custo mínimo para veículos



Fonte: O autor.

4 Testes e resultados

O capítulo tem como objetivo descrever um estudo de caso e os seus resultados. Primeiramente serão apresentados dois testes de cálculo do menor caminho, um para pedestres e outro para veículos. Os resultados do custo e a visualização da rota gerada no pgAdmin 4 foram comparados com o Google Maps. Para observar o traçado de rota no pgAdmin, utilizou-se o sistema para lançar no console da IDE (*Integrated Development Environment*) os nós mais próximos da origem e do destino assim que fossem encontrados. Após obter os nós, a consulta com a função *pgr_dijkstra* foi realizada no pgAdmin e os resultados foram observados no *geometry viewer*. O terceiro teste é de desempenho, para medir o tempo médio da aplicação usando algoritmo de Dijkstra do pgRouting. São calculadas cinco mil rotas para cada um dos tipos de locomoção, pegando os nós de origem e de destino aleatoriamente, com o objetivo apenas de apresentar uma noção do tempo gasto com o cálculo das rotas em um computador pessoal com configuração comum. Esse último teste foi repetido uma vez.

4.1 Teste de menor caminho para pedestres e comparação com o Google Maps

Os testes para pedestres e veículos desse capítulo foram feitos com locais conhecidos na cidade de Uberlândia e que são distantes uns dos outros. No teste para pedestres a origem foi o Terminal Planalto e o destino foi o Uberlândia Shopping. Endereços:

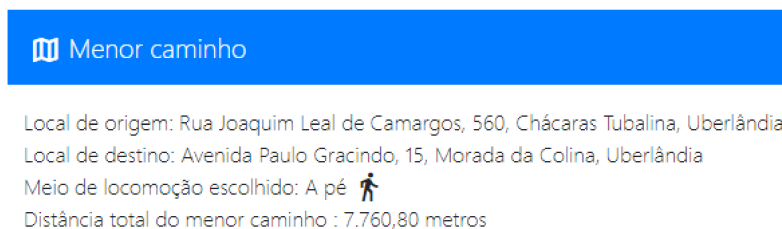
- Terminal Planalto: Rua Joaquim Leal de Camargos, número 560, bairro Chácaras Tubalina e Quartel.
- Uberlândia Shopping: Avenida Paulo Gracindo, número 15, bairro Morada da Colina.

Ao calcular a rota de custo mínimo pela aplicação, o retorno da distância total foi de 7.760,80 metros. O resultado pode ser visualizado na Figura 29.

Uma rota que vai do Terminal Planalto até o Uberlândia Shopping foi criada no Google Maps, escolhendo a opção de pedestres. Conforme pode ser observado na Figura 30, são sugeridos três caminhos, em que o mais rápido (em azul) possui 7,7 quilômetros (7.700 metros) de distância. Distância muito parecida com os 7.760,80 metros retornados pelo sistema com o algoritmo de Dijkstra do pgRouting. Outro resultado observado foi o comparativo entre a rota tracejada no Google Maps e a da Figura 31, que foi gerada

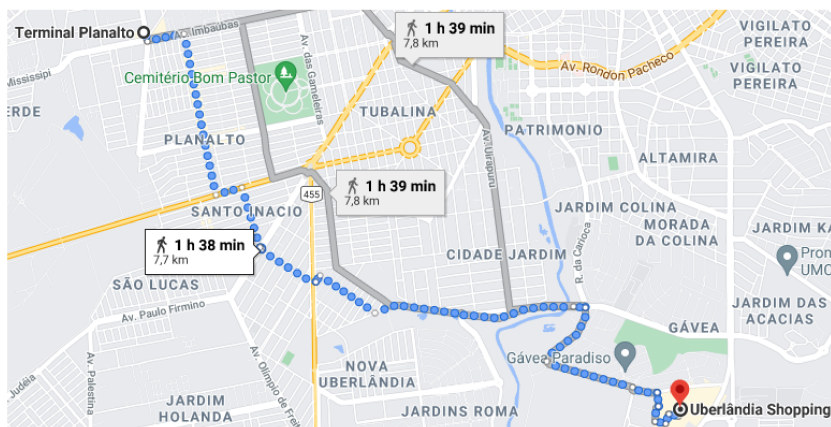
no banco de dados e visualizada com o *geometry viewer* do pgAdmin. Os dois caminhos assemelham-se bastante, tendo poucas diferenças entre algumas das vias escolhidas para ir da origem até o destino.

Figura 29 – Distância total entre o Terminal Planalto e Uberlândia Shopping



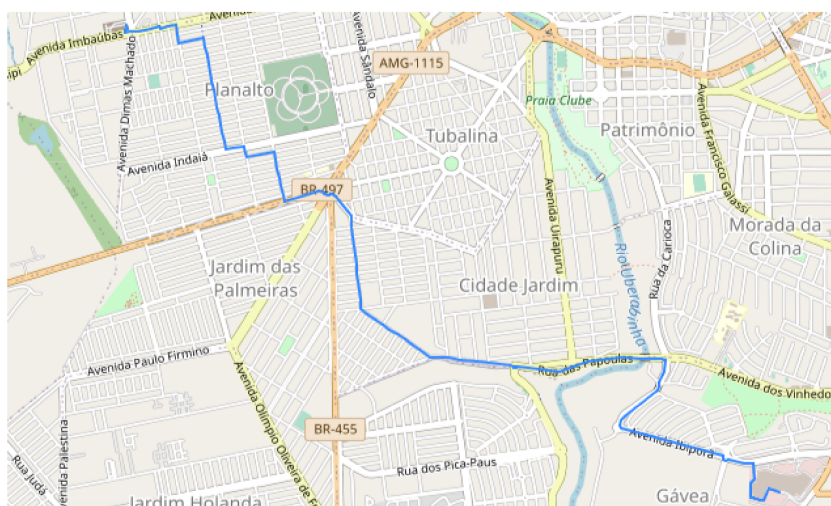
Fonte: O autor.

Figura 30 – Rota entre o Terminal Planalto e Uberlândia Shopping no Google Maps



Fonte: Google (2021a)

Figura 31 – Rota entre o Terminal Planalto e Uberlândia Shopping no pgAdmin 4



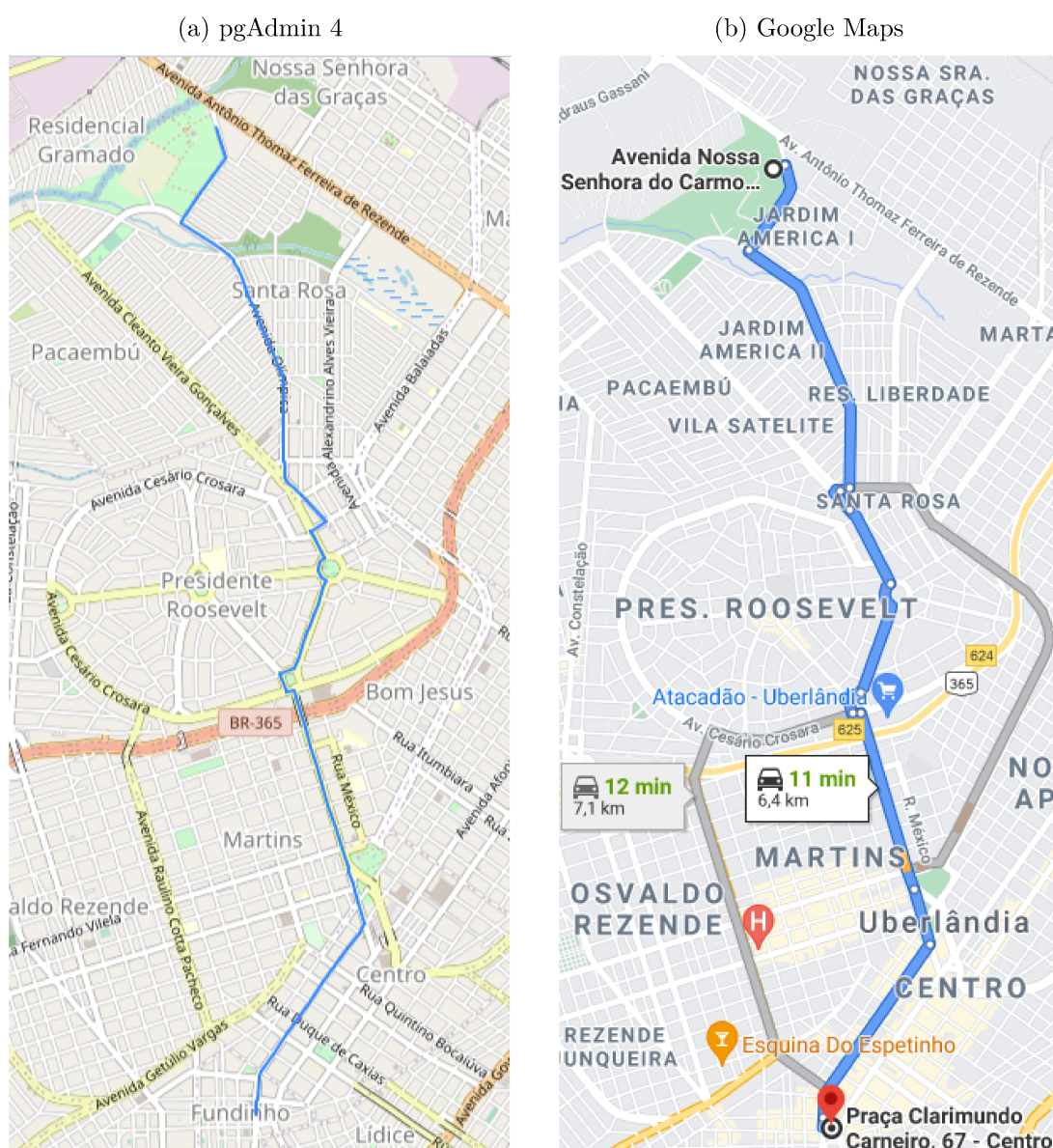
Fonte: O autor.

4.2 Teste de menor caminho para veículos e comparação com o Google Maps

A análise do teste de menor caminho para veículos foi feita com a origem sendo o Parque Municipal Victório Siquierolli, e o destino foi o Museu Municipal. Endereços:

- Parque Municipal Victório Siquierolli: Avenida Nossa Senhora do Carmo, número 707, bairro Jardim América I.
- Museu Municipal: Praça Clarimundo Carneiro, número 67, bairro Centro.

Figura 32 – Rota entre o Parque Siquierolli e Museu Municipal

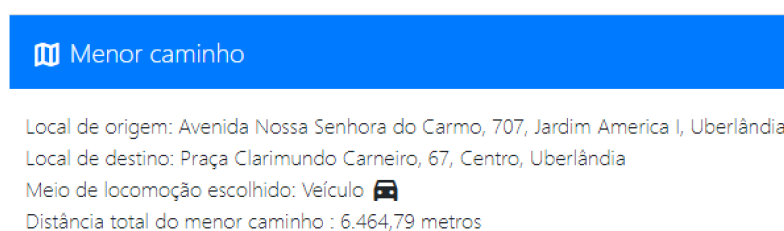


Fonte: O autor.

A Figura 32 apresenta uma comparação entre a rota de custo mínimo encontrada utilizando a função *pgr_dijkstra* (32a) e a rota de menor tempo do Google Maps(32b). Elas são quase idênticas, sendo diferentes em apenas um local.

Além do traçado do caminho, a distância também foi comparada. A Figura 33 mostra o resultado da aplicação para o cálculo da rota entre os dois endereços, que foi de 6.464,79 metros. A distância total no Google Maps foi de 6,4 km (6400 metros).

Figura 33 – Resposta da aplicação com a distância total do menor caminho entre o Parque Siquierolli e Museu Municipal



Fonte: O autor.

Esses dois primeiros testes do capítulo foram importantes para obter um comparativo do traçado e da distância das rotas obtidos na aplicação. Em ambos foram observados resultados bastante semelhantes com o Google Maps.

4.3 Teste de tempo médio para o cálculo de rotas

O teste de desempenho foi realizado gerando cinco mil rotas para cada um dos tipos de locomoção que o sistema suporta, que são pedestres e veículos. O objetivo foi obter um tempo médio que a aplicação leva para calcular uma rota usando o algoritmo de Dijkstra do pgRouting. Esse teste foi feito com a ferramenta Postman¹ em um notebook que possui as seguintes configurações:

- Sistema operacional: Windows 10 Home Single Language - 64 bits
- Memória RAM: 8,00 GB
- Processador: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz, 2 núcleos, 4 processadores lógicos
- SSD 480GB, SATA, capacidade de leitura de até 500 MB/s, capacidade de gravação de até 450 MB/s.

A tabela *ways_vertices_pgr* possui 26.991 nós distintos. Cada uma das rotas calculadas teve os nós de origem e de destino gerados aleatoriamente, pegando um número de

¹ <https://www.postman.com/>

1 a 26991. Dessa forma, garantiu-se uma grande variedade de caminhos testados, tendo diversos tamanhos. A Figura 34 apresenta o resultado do teste. O tempo médio para se calcular uma rota para o tipo de locomoção veículo (34a) foi de 551 milissegundos, enquanto que para pedestres (34b) a média foi de 628 milissegundos. O campo tempoTotalMilissegundos é a somatória dos tempos de cada uma das cinco mil rotas.

Figura 34 – Resultados do teste de desempenho

(a) Veículo

```
GET localhost:8080/menorCaminho/calcularTempoMedio?quantidadeRotas=5000&tipoLocomocao=veiculo

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

1 {
2   "quantidadeRotas": 5000,
3   "tempoTotalMilissegundos": 2759711,
4   "tempoMedioMilissegundos": 551
5 }
```

(b) Pedestre

```
GET localhost:8080/menorCaminho/calcularTempoMedio?quantidadeRotas=5000&tipoLocomocao=pedestre

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

1 {
2   "quantidadeRotas": 5000,
3   "tempoTotalMilissegundos": 3143059,
4   "tempoMedioMilissegundos": 628
5 }
```

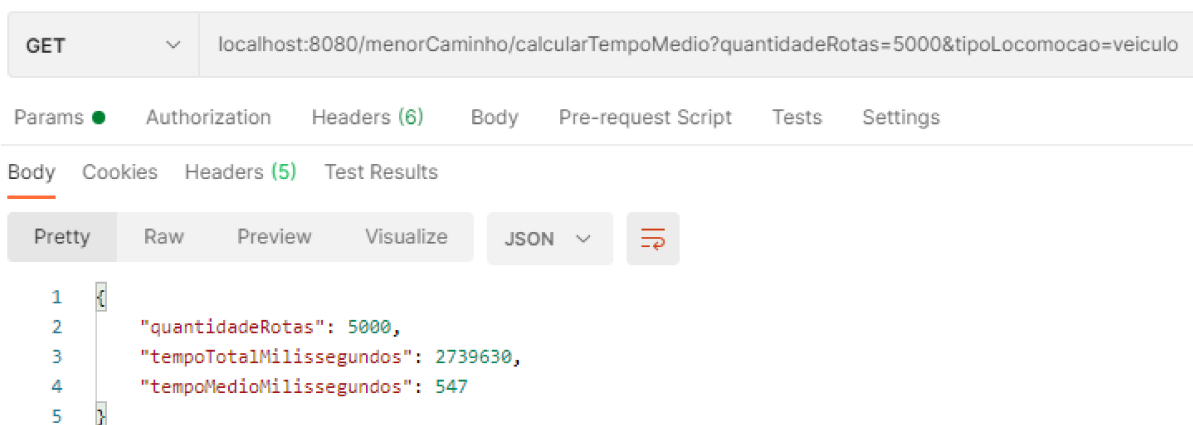
Fonte: O autor.

Repetindo o mesmo teste, os tempos obtidos foram semelhantes ao primeiro. Os resultados podem ser visualizados na Figura 35, a média foi de 547 milissegundos para veículos (35a) e 598 milissegundos para pedestres (35b).

Com os dois testes concluídos, o tempo médio geral para se obter uma rota pela aplicação utilizando o algoritmo de Dijkstra do pgRouting foi de 581 milissegundos. Esse valor foi encontrado fazendo a somatória dos resultados de média das vinte mil rotas geradas: $(551 + 628 + 547 + 598) / 4 = 581$.

Figura 35 – Resultados da repetição do teste de desempenho

(a) Veículo



```
GET localhost:8080/menorCaminho/calcularTempoMedio?quantidadeRotas=5000&tipoLocomocao=veiculo


Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

1 {
2   "quantidadeRotas": 5000,
3   "tempoTotalMilissegundos": 2739630,
4   "tempoMedioMilissegundos": 547
5 }
```

(b) Pedestre



```
GET localhost:8080/menorCaminho/calcularTempoMedio?quantidadeRotas=5000&tipoLocomocao=pedestre

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

1 {
2   "quantidadeRotas": 5000,
3   "tempoTotalMilissegundos": 2993533,
4   "tempoMedioMilissegundos": 598
5 }
```

Fonte: O autor.

As rotas para veículos utilizam o conceito de grafos direcionados e as de pedestre não. Apesar dessa diferença, observou-se tempos médios muito semelhantes entre eles, com os cálculos de menor caminho para pedestres demorando poucos milissegundos a mais nos dois testes realizados.

5 Conclusão

Neste trabalho foram realizados estudos e o desenvolvimento de um sistema com o objetivo de conhecer e de trabalhar com dados de mapas abertos do OpenStreetMap, armazenando-os em uma base de dados geográfica para serem utilizados no cálculo de rotas de menor caminho entre dois endereços.

O objetivo do trabalho foi atingido por meio do uso de dados geográficos do OpenStreetMap e de ferramentas como o PostGIS e pgRouting, além da criação do sistema com interface WEB. Os dados da cidade de Uberlândia foram incluídos em um banco PostgreSQL, onde foi instalada a extensão PostGIS que inseriu suporte a objetos geográficos. Rotas de caminho mínimo foram geradas com o pgRouting, que adicionou roteamento geoespacial ao PostgreSQL e forneceu uma função que utiliza o algoritmo de Dijkstra.

O resultado obtido demonstra uma das inúmeras possibilidades de se trabalhar com dados geográficos livres. Foi apresentado como esses dados do mundo inteiro podem ser definidos, obtidos e importados para uma base, possibilitando utilizá-los das mais diversas formas, não se limitando ao roteamento. O sistema pode ainda ser aprimorado para se tornar uma alternativa a ferramentas comerciais pagas ou gratuitas com limitações de uso.

Durante o estudo do trabalho, a maior dificuldade foi descobrir uma forma de calcular rotas de menor caminho utilizando endereços reais e não nós aleatórios da base de dados. Essa dificuldade foi superada com o uso de uma consulta para encontrar o nó mais próximo a partir da latitude e longitude de um local. Para descobrir as coordenadas de um endereço, utilizou-se a API *Geocoding* do Google. Essa API atende muito bem às expectativas de um trabalho acadêmico, oferecendo um alto número de requisições gratuitas por mês. Porém, ela traz a limitação de não ser *open source*. Por isso, é sugerida como uma melhoria para trabalhos futuros a troca dessa API, tornando todas as ferramentas gratuitas e sem limite de utilização.

5.1 Trabalhos Futuros

Estudos e melhorias podem ser realizados a fim de evoluir o sistema que foi desenvolvido. Para trabalhos futuros, sugere-se as seguintes ideias:

- Substituir a API *Geocoding* por alguma outra que seja *open source*.
- Utilizar um conjunto de dados maior, como o de um país ou continente.

- Oferecer mais de uma opção de rota para o usuário ir da origem até o destino desejado.
- Permitir o cálculo de rotas com pontos de interesse entre a origem e o destino.
- Testar e comparar resultados de outras funções do pgRouting que resolvem o problema de caminho mínimo. Por exemplo, a função *pgr_KSP* pode ser usada para sugerir rotas alternativas.
- Melhorar o *front-end* apresentando as informações de uma melhor maneira para os usuários. Como exemplo, pode adicionar na tela um mapa contendo o caminho da rota que foi gerado, assim como o que pode ser visualizado no pgAdmin 4.

5.2 Considerações Finais

O trabalho me ajudou a evoluir nos conhecimentos técnicos e de pesquisa. Foi muito desafiador e empolgante aprender como objetos do mundo real são representados dentro de um banco de dados geográfico e como essas informações podem ser aplicadas em problemas reais.

Referências

- ARCGIS. *What is raster data?* 2020. Disponível em: <<https://desktop.arcgis.com/en/arcmap/latest/manage-data/raster-and-images/what-is-raster-data.htm>>. Acesso em: 14 maio 2021. Citado na página 17.
- BENNETT, J. *OpenStreetMap: Be your own cartographer*. Birmingham: Packt Publishing, 2010. Citado na página 17.
- BURROUGH, P. A.; MCDONNELL, R. A. *Principles of Geographical Information Systems*. 2. ed. [S.l.]: Oxford University Press, 1998. Citado na página 14.
- CÂMARA, G. et al. *Anatomia de Sistemas de Informação Geográfica*. Campinas: Instituto de Computação, UNICAMP, 1996. Disponível em: <<http://www.dpi.inpe.br/gilberto/livro/anatomia.pdf>>. Citado na página 14.
- CASANOVA, M. A. et al. *Bancos de Dados Geográficos*. Curitiba: Mundogeo, 2005. Disponível em: <<http://www-di.inf.puc-rio.br/~casanova//Publications/Books/2005-BDG.pdf>>. Citado 2 vezes nas páginas 11 e 15.
- CORMEN, T. H. et al. *Algoritmos: Teoria e Prática*. Rio de Janeiro: Elsevier, 2012. Citado 2 vezes nas páginas 21 e 22.
- DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik*, v. 1, p. 269–271, 1959. Citado na página 22.
- FITZ, P. R. *Geoprocessamento sem complicação*. São Paulo: Oficina de Textos, 2008. Citado 4 vezes nas páginas 11, 14, 16 e 17.
- GOOGLE. *Google Maps*. 2021. Disponível em: <<https://www.google.com.br/maps>>. Acesso em: 3 maio 2021. Citado na página 56.
- GOOGLE. *Material Icons Guide*. 2021. Disponível em: <https://developers.google.com/fonts/docs/material_icons>. Acesso em: 07 abr. 2021. Citado na página 27.
- GOOGLE. *Overview*. 2021. Disponível em: <<https://developers.google.com/maps/documentation/geocoding/overview>>. Acesso em: 09 abr. 2021. Citado 3 vezes nas páginas 28, 29 e 30.
- HILLIER, F. S.; LIEBERMAN, G. J. *Introdução à pesquisa operacional*. 8. ed. São Paulo: McGraw-Hill, 2006. Citado na página 21.
- MARINS, F. A. S. *Introdução à pesquisa operacional*. São Paulo: Cultura Acadêmica, 2011. Citado na página 22.
- OBE, R. O.; HSU, L. S. *PostGIS in Action*. 2. ed. [S.l.]: Manning Publications, 2015. Citado 5 vezes nas páginas 15, 31, 32, 40 e 41.
- OPENSTREETMAP. *Direitos autorais e licença*. 2021. Disponível em: <<https://www.openstreetmap.org/copyright>>. Acesso em: 15 maio 2021. Citado na página 18.

OPENSTREETMAP. *Elements*. 2021. Disponível em: <<https://wiki.openstreetmap.org/wiki/Elements>>. Acesso em: 15 maio 2021. Citado na página 19.

OPENSTREETMAP. *OpenStreetMap Search*. 2021. Disponível em: <<https://www.openstreetmap.org>>. Acesso em: 25 abr. 2021. Citado na página 43.

OPENSTREETMAP. *OSM file formats*. 2021. Disponível em: <https://wiki.openstreetmap.org/wiki/OSM_file_formats>. Acesso em: 15 maio 2021. Citado na página 18.

OPENSTREETMAP. *Relation:multipolygon*. 2021. Disponível em: <<https://wiki.openstreetmap.org/wiki/Relation:multipolygon>>. Acesso em: 18 maio 2021. Citado na página 20.

SPRINGIO. *Spring Framework Overview*. 2021. Disponível em: <<https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html#overview>>. Acesso em: 08 abr. 2021. Citado na página 28.

THYMELEAF. *1 Introducing Thymeleaf*. 2018. Disponível em: <<https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html>>. Acesso em: 07 abr. 2021. Citado na página 26.

W3S. *Bootstrap Get Started*. 2021. Disponível em: <https://www.w3schools.com/bootstrap/bootstrap_get_started.asp>. Acesso em: 07 abr. 2021. Citado na página 27.

W3S. *HTML Introduction*. 2021. Disponível em: <https://www.w3schools.com/html/html_intro.asp>. Acesso em: 04 abr. 2021. Citado na página 26.

Apêndices

APÊNDICE A – Exportação e conversão de formato do mapa de Uberlândia

O Apêndice A descreve um passo a passo para obter-se o mapa de Uberlândia e de como converter o arquivo *pbk* para o formato *osm*.

A.1 Exportação do mapa de Uberlândia

Os dados do mapa podem ser obtidos a partir do site [Export Hot OSM](https://export.hotosm.org) por meio dos seguintes passos:

- Login com uma conta do OpenStreetMap ou de terceiros, como Google e Facebook;
- Acesso ao menu “**Criar**” localizado no canto superior direito da tela (Figura 36);

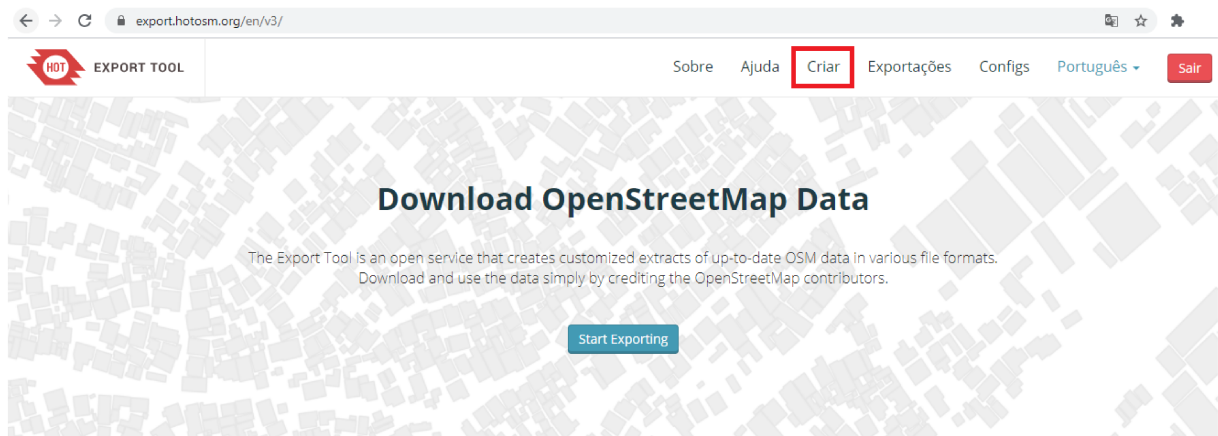


Figura 36 – Página inicial do Export Hot Tool

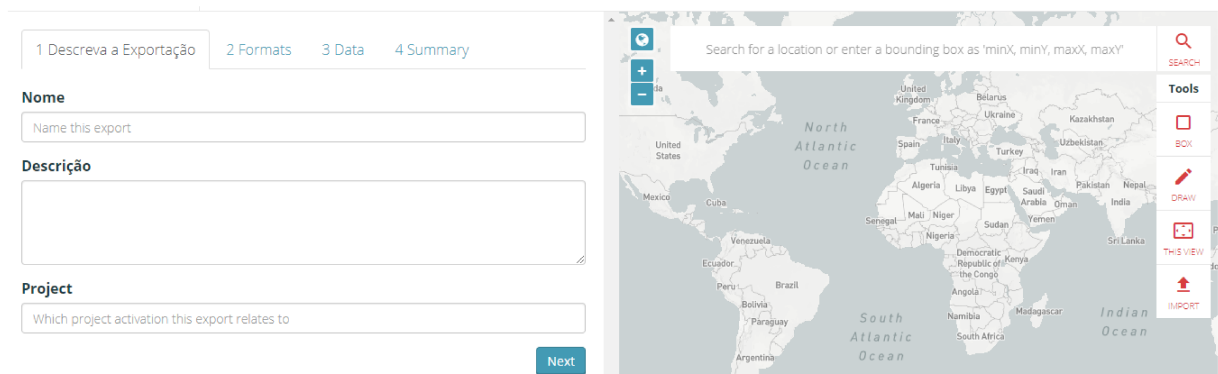


Figura 37 – Tela inicial da exportação

- Busca pela cidade de Uberlândia no campo de pesquisa localizado no mapa do lado direito da página (Figura 38);
- Seleção da área a ser exportada. A área da cidade escolhida será definida automaticamente, mas é possível utilizar as ferramentas *Box*, *Draw* e *This View* para selecionar a área desejada (Figura 39);

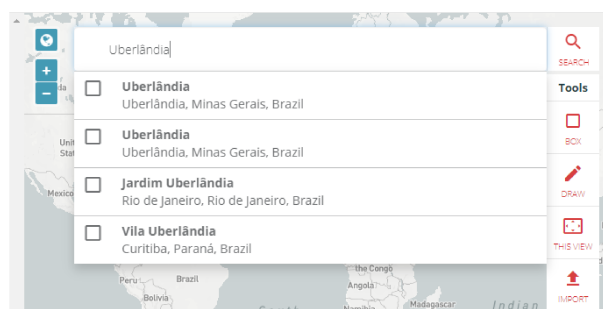


Figura 38 – Tela de seleção da cidade de Uberlândia para a exportação

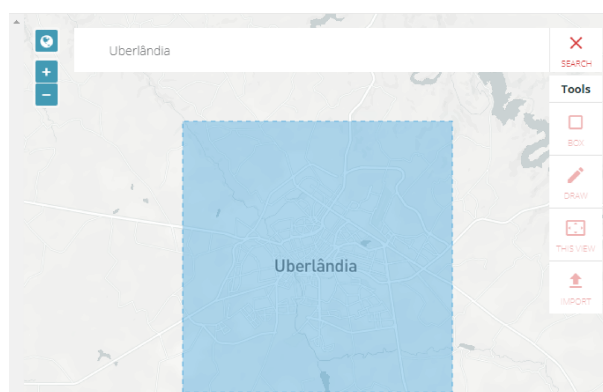


Figura 39 – Área da cidade de Uberlândia selecionada automaticamente

- Definição de algumas informações no lado esquerdo da página:
 1. Descreva a exportação: Nome para o arquivo de exportação. Não é necessário informar a *Descrição* e o *Project* (Figura 40);
 2. Formats: Escolha do formato **OSM (.pbf)** (Figura 41);
 3. Data: Marcação de todas as caixas de seleção (Figura 42);
 4. Summary: Finalização da exportação (Figura 43).
- Download do arquivo clicando sobre o seu nome (Figura 44).

Figura 40 – Tela de definição do nome do arquivo a ser exportado

Figura 41 – Tela de seleção do formato do arquivo a ser exportado

Figura 42 – Tela de escolha dos dados a serem exportados

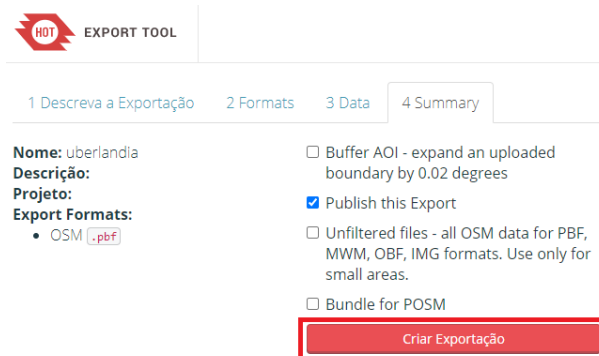


Figura 43 – Tela de finalização da exportação

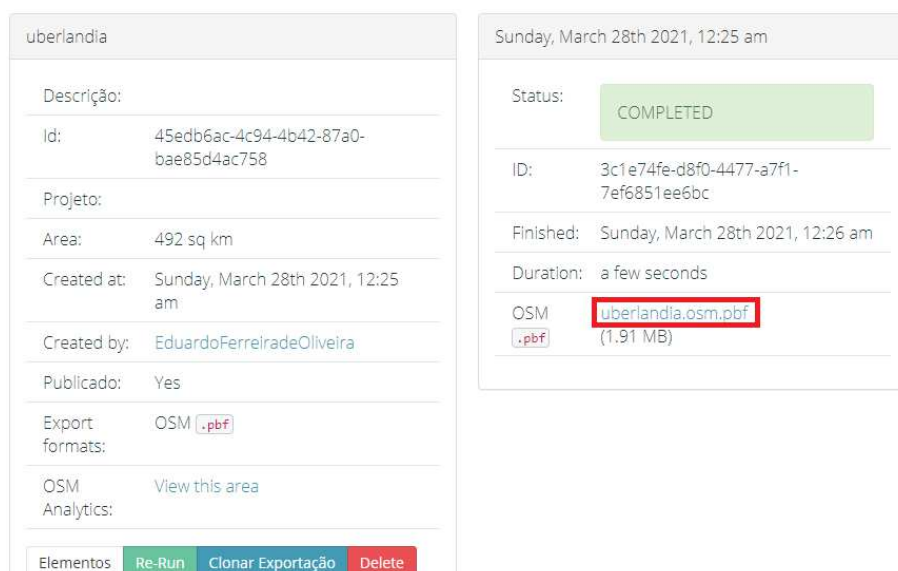


Figura 44 – Exportação finalizada e arquivo disponível para baixar

A.2 Convertendo o formato do arquivo

A ferramenta `osmconvert` é utilizada para transformar o arquivo de `pbf` para o formato `osm`. Ela pode ser obtida a partir do sítio¹ do OpenStreetMap. No projeto foi utilizada a versão *binary for Windows 64 bit (with large file support)*.

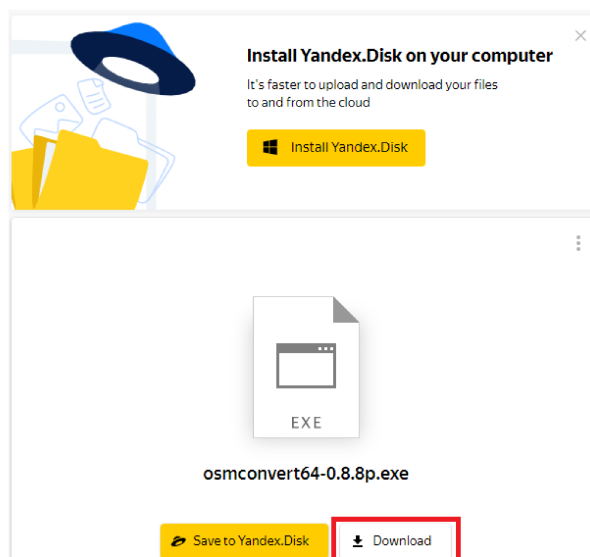


Figura 45 – Página de download do `osmconvert`

Após a instalação do `osmconvert` o arquivo pode ser convertido por meio dos passos:

1. Inserção do `osmconvert` e do arquivo `pbf` em um mesmo diretório (Figura 46);
2. Abertura do Prompt de Comando do Windows, navegação até a pasta onde os dois arquivos estão e iniciação do conversor por meio do comando: **`osmconvert64-0.8.8p.exe`** (Figura 47);
3. Escolha da opção “a” (Figura 48);
4. Inserção do nome do arquivo `pbf` (Figura 49);
5. Escolha da opção “1” nas duas próximas etapas (Figuras 49 e 50). O arquivo com extensão `.osm` será criado (Figura 51).

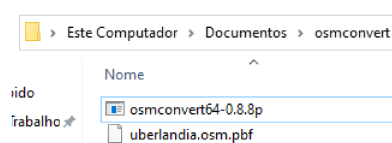


Figura 46 – Arquivos executável e `pbf` em um mesmo local

¹ <https://wiki.openstreetmap.org/wiki/Osmconvert#Windows>

```

C:\Users\eduardo.oliveira>cd C:\Users\eduardo.oliveira\Documents\osmconvert
C:\Users\eduardo.oliveira\Documents\osmconvert>osmconvert64-0.8.8p.exe
osmconvert 0.8.8

Converts .osm, .osm, .pbf, .osc, .osh files, applies changes
of .osc, .osc, .osh files and sets limiting borders.
Use command line option -h to get a parameter overview,
or --help to get detailed help.

If you are familiar with the command line, press <Return>.

If you do not know how to operate the command line, please
enter "a" (press key E and hit <Return>).

```

Figura 47 – Iniciação do osmconvert

```

If you do not know how to operate the command line, please
enter "a" (press key E and hit <Return>).
a
-----
Hi, I am osmconBert - just call me Bert.
I will guide you through the basic functions of osmconvert.

At first, please ensure to have the "osmconvert" file
(resp. "osmconvert.exe" file if windows) located in the
same directory in which all your OSM data is stored.

You may exit this program whenever you like. Just hold
the <Ctrl> key and press the key C.

Please please tell me the name of the file you want to process:

```

Figura 48 – Etapa 1 da conversão do arquivo

```

Please please tell me the name of the file you want to process:
uberlandia.osm.pbf
Thanks!
-----
What may I do with this file?

1  convert it to a different file format
2  use an OSM Changefile to update this file
3  use a border box to limit the geographical region
4  use a border polygon file to limit the geographical region
5  minimize file size by deleting author information
6  display statistics of the file
To options 3 or 4 you may also choose:
  a  keep ways complete, even if they cross the border
  b  keep ways and areas complete, even if they cross the border

Please enter the number of one or more functions you choose:
1
All right.

```

Figura 49 – Etapa 2 da conversão do arquivo

```

Please choose the output file format:

1 .osm (standard XML format - results in very large files)
2 .osm (binary format - allows fast)
3 .pbf (standard binary format - results in small files)

Enter 1, 2 or 3:
1
Thanks!
-----
Now, please hang on - I am working for you.
If the input file is very large, this will take several minutes.

If you want to get acquainted with the much more powerful
command line, this would have been your command:

osmconvert uberlandia.osm.pbf --out-osm -o=uberlandia.osm_01.osm
-----
Finished! Calculation time: 0s.
I just completed your new file with this name:
uberlandia.osm_01.osm
Thanks for visiting me. Bye!
Yours, Bert
(To close this window, please press <Return>.)

```

Figura 50 – Etapa 3 da conversão do arquivo

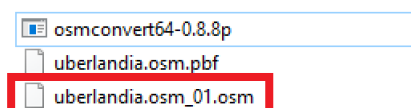


Figura 51 – Representação do arquivo de formato osm criado

APÊNDICE B – Instalações e configurações da base de dados

Neste apêndice é apresentado a instalação e configuração da base de dados. Será mostrado como instalar o banco de dados PostgreSQL e como configurá-lo para adição das extensões, bem como realizar a importação do arquivo *.osm*.

B.1 Instalação do PostgreSQL e configuração do banco de dados

No projeto foi utilizado a versão 13.2 do PostgreSQL, que pode ser obtido a partir do sítio¹ do PostgreSQL. Durante a instalação, foram selecionados os componentes conforme apresentado na Figura 52.

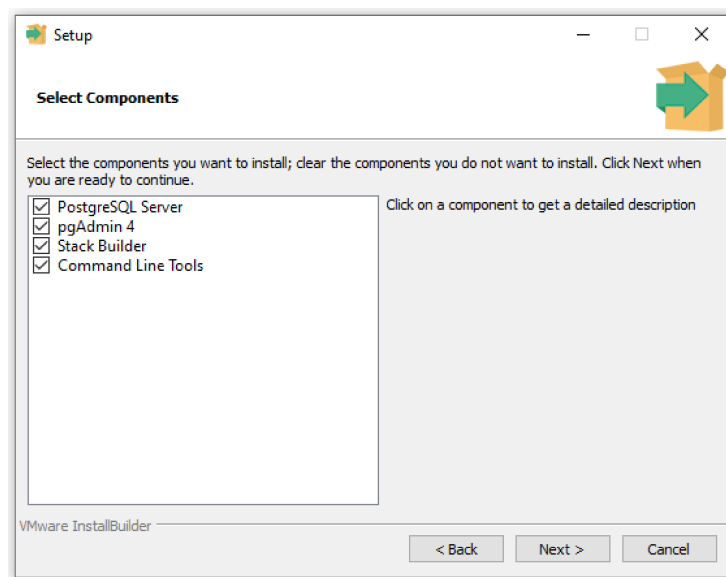


Figura 52 – Etapa de seleção dos componentes na instalação do PostgreSQL

Após a finalização da instalação execute o Stack Builder. Com ele é possível instalar a extensão espacial PostGIS, por meio dos passos:

1. Escolha do banco de dados instalado (Figura 53);
2. Seleção do **PostGIS 3.1** em **Spatial Extensions** o (Figura 54);
3. Permissão para que o Stack Builder configure as variáveis de ambiente;
4. Finalização da instalação do PostGIS (Figura 55).

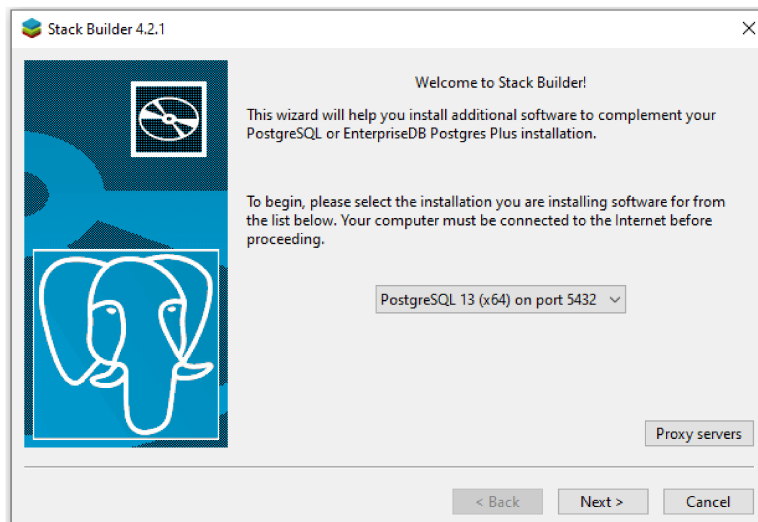


Figura 53 – Etapa de seleção da instalação do PostgreSQL

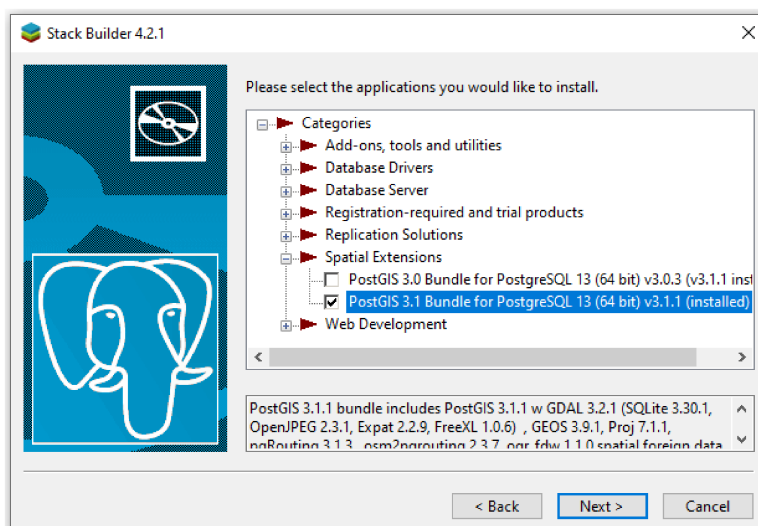


Figura 54 – Tela de seleção do PostGIS 3.1 para ser instalado

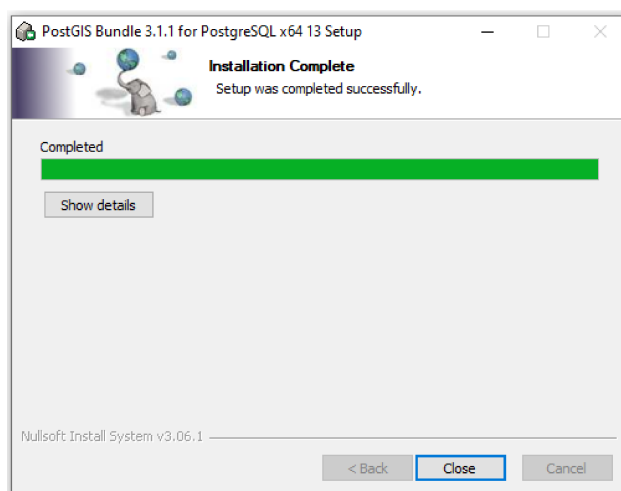


Figura 55 – Tela de conclusão da instalação

Ao término da instalação do PostGIS deve-se executar o pgAdmin 4. Ele é usado para administrar o banco de dados. Para a criação e a configuração de uma *database* são seguidos os passos:

- Criação e definição do nome de um novo *database* (Figuras 56 e 57);
- Criação de uma nova extensão (Figura 58);
- Seleção da extensão **postgis** (Figura 59);
- Instalação da extensão **pgrouting** seguindo os dois passos anteriores, alterando apenas o nome no momento da escolha;
- Listagem para confirmar a criação das extensões. Uma terceira extensão chamada **plpgsql** é adicionada por padrão (Figura 60).

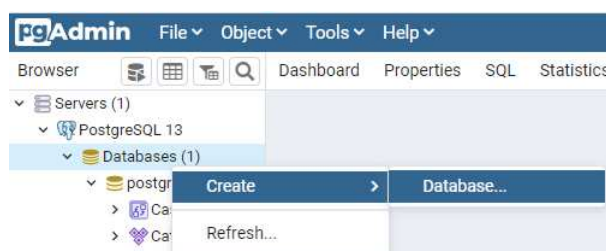


Figura 56 – Tela de criação de uma nova *database*

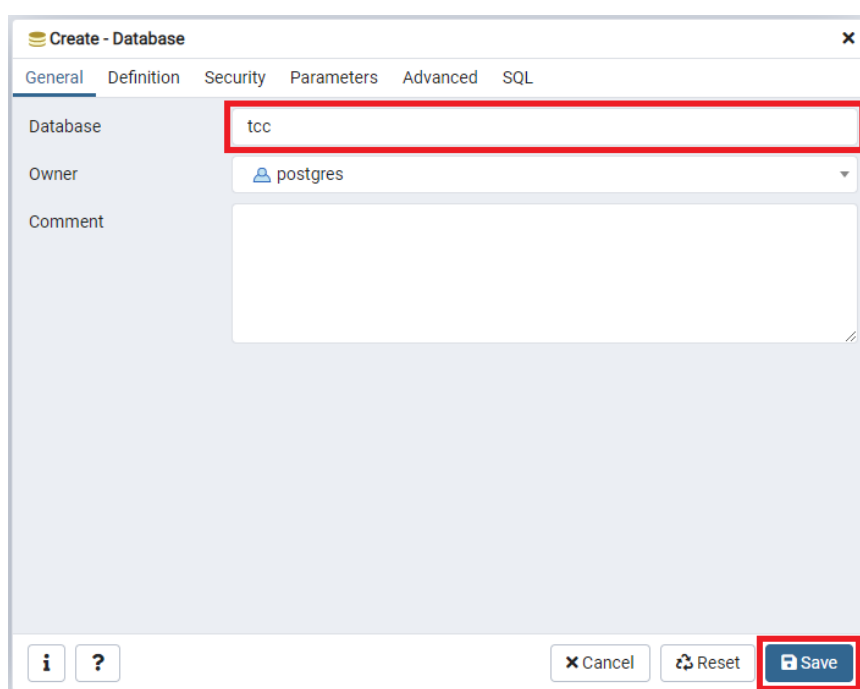


Figura 57 – Etapa de escolha do nome da *database* a ser criada

¹ <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

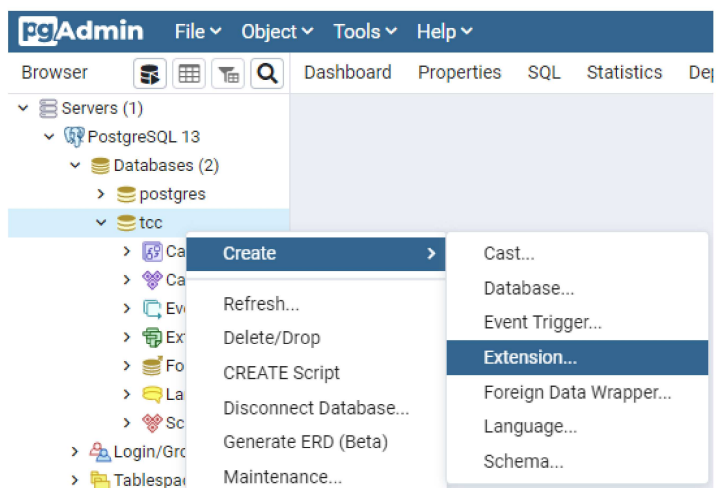


Figura 58 – Etapa de criação de uma extensão em uma database

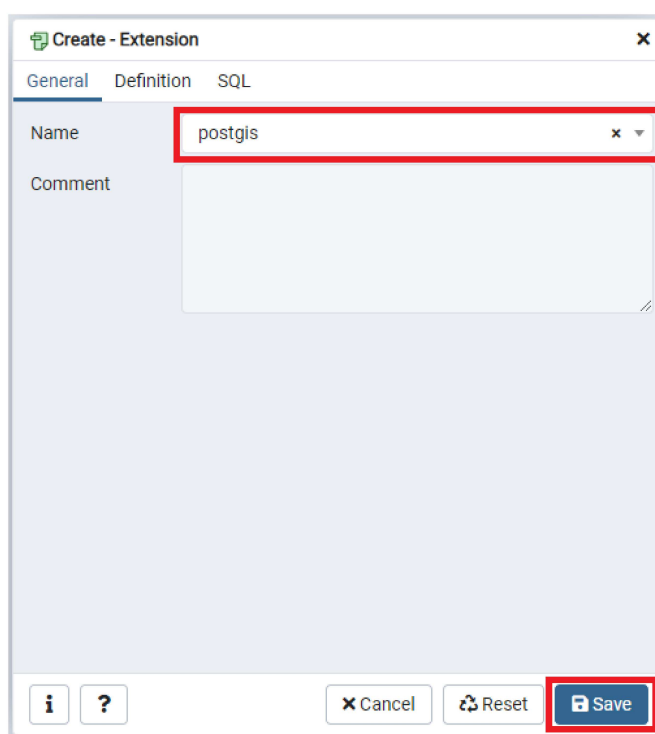


Figura 59 – Etapa de escolha do PostGIS para ser adicionado como extensão

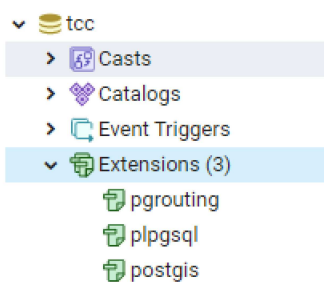
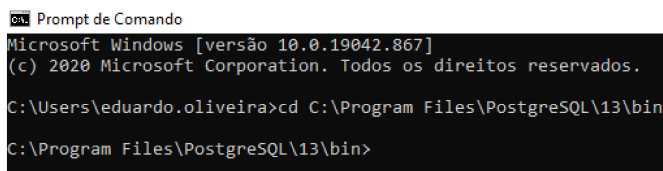


Figura 60 – Lista de extensões criadas na database

B.2 Importação do mapa

Para importação dos dados do mapa de Uberlândia utilizou-se o `osm2pgrouting`. Abra o *prompt* de comando e navegue até a pasta `bin` da instalação do PostgreSQL (Figura 61).



```
Prompt de Comando
Microsoft Windows [versão 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos os direitos reservados.
C:\Users\eduardo.oliveira>cd C:\Program Files\PostgreSQL\13\bin
C:\Program Files\PostgreSQL\13\bin>
```

Figura 61 – Navegação até a pasta `bin` do PostgreSQL

Comando usado para a importação do arquivo `osm`:

```
osm2pgrouting -f CAMINHO_E_NOME_DO_ARQUIVO.OSM -d NOME_DA_DATABASE -p
PORTA_DO_POSTGRESQL -U USUARIO -W SENHA -c mapconfig.xml --clean
```

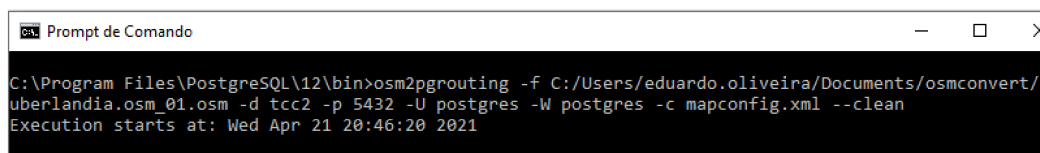
Os valores em caixa alta são substituídos de acordo com os utilizados no projeto:

- **CAMINHO_E_NOME_DO_ARQUIVO.OSM**: Caminho completo do arquivo `.osm` criado na etapa de conversão de formato;
- **NOME_DA_DATABASE**: Nome da *database* criada no PostgreSQL;
- **PORTA_DO_POSTGRESQL**: Porta do banco de dados;
- **USUARIO**: Usuário do banco;
- **SENHA**: Senha do banco.

Informações sobre os argumentos `-c mapconfig.xml` e `--clean`:

- `-c mapconfig.xml`: Nome do arquivo de configuração padrão do `osm2pgrouting`.
- `--clean`: Deleta tabelas que podem ter sido criadas anteriormente.

Para iniciar a importação o comando é inserido no terminal (Figura 62).



```
Prompt de Comando
C:\Program Files\PostgreSQL\12\bin>osm2pgrouting -f C:/Users/eduardo.oliveira/Documents/osmconvert/uberlandia.osm_01.osm -d tcc2 -p 5432 -U postgres -W postgres -c mapconfig.xml --clean
Execution starts at: Wed Apr 21 20:46:20 2021
```

Figura 62 – Comando para a importação do arquivo `osm`

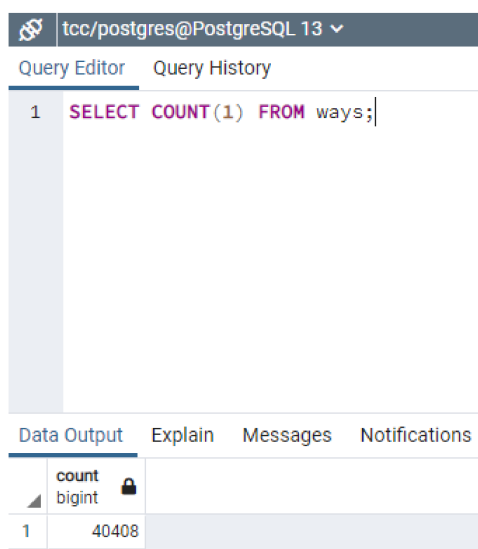
Caso não ocorram erros, os dados serão importados. O `osm2pgrouting` informa o tamanho das ruas e o tempo de execução (Figura 63).

```
#####  
size of streets: 18932  
Execution started at: Wed Apr 21 20:46:20 2021  
Execution ended at: Wed Apr 21 20:46:39 2021  
Elapsed time: 19.372 Seconds.  
User CPU time: -> 19.371 seconds  
#####  
C:\Program Files\PostgreSQL\12\bin>
```

Figura 63 – Tela de quando uma importação é concluída com sucesso

Consulta no pgAdmin para confirmar que o banco de dados foi preenchido com as informações do arquivo de mapa:

```
1 SELECT COUNT(1) FROM ways;
```



The screenshot shows the pgAdmin interface. At the top, it says 'tcc/postgres@PostgreSQL 13'. Below that are tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active and contains the SQL query: '1 SELECT COUNT(1) FROM ways;'. Below the query editor are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Data Output' tab is active and shows a table with one row and one column. The column is labeled 'count' and has a data type of 'bigint'. The value in the row is '40408'.

	count
1	40408

Figura 64 – Consulta no pgAdmin para confirmar que os dados foram importados

Nesta consulta foi feito um *count* e o retorno foi de 40408 linhas (Figura 64).