UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO POLITÉCNICO DA UFSM
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

Kelvin Teixeira Floriano

EXPLORANDO O USO DE BANCOS DE DADOS ORIENTADOS A GRAFOS EM UM CAMPUS UNIVERSITÁRIO

Santa Maria, RS
2024

Kelvin Teixeira Floriano

EXPLORANDO O USO DE BANCOS DE DADOS ORIENTADOS A GRAFOS EM UM CAMPUS UNIVERSITÁRIO

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Tecnologia em Sistemas para Internet da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Tecnólogo em Sistemas para Internet**.

Orientador: Prof. Dr. Daniel Lichtnow

Santa Maria, RS
2024

Floriano, Kelvin Teixeira

Explorando o uso de bancos de dados orientados a grafos em um campus universitário / por Kelvin Teixeira Floriano. – 2024.
50 f.: il.; 30 cm.

Orientador: Daniel Lichtnow
Trabalho de Conclusão de Curso - Universidade Federal de Santa Maria, Colégio Politécnico da UFSM, Curso Superior de Tecnologia em Sistemas para Internet, RS, 2024.

1. Banco de dados. 2. Neo4j. 3. Grafos. I. Lichtnow, Daniel. II.Explorando o uso de bancos de dados orientados a grafos em um campus universitário.

Kelvin Teixeira Floriano

EXPLORANDO O USO DE BANCOS DE DADOS ORIENTADOS A GRAFOS EM UM CAMPUS UNIVERSITÁRIO

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Tecnologia em Sistemas para Internet da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Tecnólogo em Sistemas para Internet**.

Aprovado em 15 de Agosto de 2024:

_____

Daniel Lichtnow, Dr. (UFSM)
Presidente/Orientador

_____

Marcos Alexandre Rose Silva, Dr. (UFSM)

_____

Rafael Gressler Milbradt, Dr. (UFSM)

Santa Maria, RS
2024

O que prevemos raramente ocorre; o que menos esperamos geralmente acontece.
(Benjamin Disraeli)

AGRADECIMENTOS

Agradeço aos meus pais e avós pelo apoio que me deram durante a realização desse trabalho.

Aos professores, por todo o conhecimento que foi me passado, em especial ao professor Daniel Lichtnow, por ter sido meu orientador e pelo suporte e ajuda oferecido nesse trabalho.

RESUMO

EXPLORANDO O USO DE BANCOS DE DADOS ORIENTADOS A GRAFOS EM UM CAMPUS UNIVERSITÁRIO

AUTOR: KELVIN TEIXEIRA FLORIANO
ORIENTADOR: DANIEL LICHTNOW

Na era atual, a eficaz gestão e análise de grandes quantidades de dados se tornaram essenciais para empresas e organizações em diversos setores, como comércio, telecomunicações e educação. O aumento do volume de dados disponíveis, juntamente com a complexidade das informações e suas inter-relações, desafia as abordagens convencionais de armazenamento e processamento de dados. Diante desse cenário, surge uma demanda crescente por estruturas de banco de dados mais flexíveis, capazes de lidar com a natureza interconectada dos dados atuais e fornecer análises valiosas a partir dessas relações. Com isso em mente este trabalho apresenta uma análise sobre o uso de bancos de dados orientados a grafos, com foco no Neo4j como exemplo relevante nesse campo. O estudo começa com uma visão geral sobre bancos de dados, explorando as aplicações desses bancos de dados em diferentes domínios e ilustrando seus conceitos e base teórica para obter um maior contexto do que está por seguir. Usando os recursos do Neo4j é criado um sistema que recomenda e auxilia os frequentadores de uma Campus universitário na localização de pontos de interesse.

Palavras-chave: Banco de dados. Neo4j. Grafos.

ABSTRACT

EXPLORING THE USE OF GRAPH-ORIENTED DATABASES ON A UNIVERSITY CAMPUS

AUTHOR: KELVIN TEIXEIRA FLORINAO
ADVISOR: DANIEL LITCHNOW

In today's age, the effective management and analysis of large amounts of data has become essential for companies and organizations in various sectors, such as commerce, telecommunications and education. The increase in the volume of data available, together with the complexity of the information and its interrelationships, challenges conventional approaches to data storage and processing. Faced with this scenario, there is a growing demand for more flexible database structures, capable of dealing with the interconnected nature of today's data and providing valuable analysis from these relationships. With this in mind, this paper presents an analysis of the use of graph-oriented databases, focusing on Neo4j as a relevant example in this field. The study begins with an overview of databases, exploring the applications of these databases in different domains and illustrating their concepts and theoretical basis to gain a greater context of what is to follow. Using Neo4j's resources, a system is created that recommends and assists those attending a university campus in locating points of interest.

Keywords: Database. Neo4j. Graphs.

LISTA DE FIGURAS

Figura 3.1 - Representação gráfica de um grafo 19
Figura 4.1 - Operação de adição 22
Figura 4.2 - Operação de remoção de aresta 23
Figura 4.3 - Operação de remoção de nó 23
Figura 4.4 - Operação de busca em profundidade 23
Figura 4.5 - Resultado da busca em profundidade 24
Figura 4.6 - Operação de busca em largura em primeiro nível 24
Figura 4.7 - Resultado da busca em largura em primeiro nível 25
Figura 4.8 - Operação de busca em largura em segundo nível 25
Figura 4.9 - Resultado da busca em largura em segundo nível 26
Figura 4.10 - Base de dados para algoritmo de menor caminho 26
Figura 4.11 - Representação no grafo 27
Figura 4.12 - Operação de algoritmo de menor caminho 27
Figura 4.13 - Resultado da operação de menor caminho 28
Figura 5.1 - Diagrama ER 30
Figura 5.2 - Estrutura no Neo4j 31
Figura 5.3 - Visualização no QGIS 32
Figura 5.4 - Reprojeto do shapefile 33
Figura 5.5 - Demonstração da arquitetura 35
Figura 5.6 - Dependências do backend 36
Figura 5.7 - Estrutura de diretórios do backend 37
Figura 5.8 - Código da classe Pessoa no model 38
Figura 5.9 - Código da classe LocalRepository no repository 39
Figura 5.10 - Estrutura de diretórios do frontend 39
Figura 5.11 - Dependências necessárias do frontend 40
Figura 5.12 - Página de autenticação 42
Figura 5.13 - Página de cadastro 43
Figura 5.14 - Usuário cadastrado no banco 44
Figura 5.15 - Página inicial com recomendações 44
Figura 5.16 - Página inicial com marcador específico 45
Figura 5.17 - Página inicial ao realizar filtro 46
Figura 5.18 - Informações adicionais de um local 47

Figura 5.19 - Página dos locais favoritos 48

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	*Application Programming Interface*
BFS	*Breadth First Search*
DFS	*Depht First Search*
HTML	*HyperText Markup Language*
IoT	*Internet of Things*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
MVC	*Model View Controller*
NoSQL	*Not Only SQL*
SQL	*Structured Query Language*
TCC	Trabalho de Conclusão de Curso
UFSM	Universidade Federal de Santa Maria
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS 14
1.1.1 Objetivo Geral 14
1.1.2 Objetivos Específicos 14
1.2 ORGANIZAÇÃO DO TRABALHO 14
2 CONCEITOS SOBRE BANCO DE DADOS 16
2.1 BANCO DE DADOS: CONCEITOS E MODELOS 16
2.2 BANCO DE DADOS RELACIONAIS 16
2.3 BANCO DE DADOS NÃO RELACIONAIS 17
3 GRAFOS 19
3.1 GRAFOS: CONCEITOS 19
4 BANCO DE DADOS ORIENTADO A GRAFOS 21
4.1 NEO4J 21
4.2 USANDO O NEO4J 21
4.2.1 Exemplos de operações 22
5 USO DO NEO4J EM UM SISTEMA 29
5.1 TECNOLOGIAS UTILIZADAS 29
5.2 ESTRUTURA DO BANCO 29
5.2.1 Estrutura no Neo4j 30
5.3 ARQUITETURA DA APLICAÇÃO 34
5.3.1 Backend 35
5.3.2 Frontend 39
5.4 EXEMPLO DE USO DA APLICAÇÃO 41

1 INTRODUÇÃO

Com o crescimento da massa de dados na Internet e consequentemente no mundo, principalmente nas grandes empresas, um grande problema emergiu no resgate das informações nestes ambientes, isto devido à complexidade na qual essa massa se encontra, assim como na falta de performance que os atuais paradigmas enfrentam para manusearem esta (SANTOS; SILVA, 2013).

“Durante os últimos anos, a quantidade de informações acumuladas em grandes repositórios de dados se tornou um impeditivo para o alto desempenho das Tecnologias de Informação e Comunicação” (ALVAREZ; CECI; GONÇALVES, 2016).

Dentro desse contexto, uma abordagem que tem ganhado destaque é a utilização de bancos de dados NoSQL. Com foco em lidar com requisitos específicos, como escalabilidade, flexibilidade de esquema e alta disponibilidade. O termo “NoSQL” não está definido de forma muito clara. Geralmente é aplicado a alguns bancos de dados não relacionais recentes, como o Cassandra, o Mongo, o Neo4j e o Riak. Eles utilizam dados sem esquema, são executados em clusters e trocam a consistência tradicional por outras propriedades úteis (SADALAGE; FOWLER, 2019).

Diferentemente dos tradicionais bancos de dados relacionais, que utilizam tabelas e relacionamentos fixos, os bancos de dados orientados a grafos permitem representar e explorar de forma eficiente as relações complexas existentes nos dados. Essa abordagem baseada em grafos, oferece um desempenho otimizado para consultas e análises de relacionamentos.

Diante desse cenário, este trabalho tem como objetivo apresentar uma análise sobre bancos de dados orientados a grafos, discutindo suas aplicações em diversas áreas e explorando especificamente o Neo4j como um exemplo relevante nesse campo. O Neo4j é um banco de dados orientado a grafos líder de mercado, amplamente utilizado em aplicações que requerem a análise e a manipulação eficiente de dados conectados.

O Neo4j, que é uma ferramenta de banco de dados orientado a grafos, armazena dados na forma de grafos, que podem representar objetos com os nós, arestas e propriedades. Consequentemente, é adequado armazenar relações complexas e dinâmicas entre objetos de dados (LU; HONG; SHI, 2017).

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo deste trabalho é fornecer uma visão geral sobre os bancos de dados orientados a grafos, com foco específico na plataforma Neo4j, explorando conceitos e aplicações, a fim de obter uma compreensão abrangente dessa tecnologia, bem como fazer uma aplicação que utilize o Neo4j.

1.1.2 Objetivos Específicos

Para atingir o objetivo principal deste trabalho, os seguintes objetivos específicos são relacionados:

- Explorar os fundamentos dos bancos de dados orientados a grafos;
- Investigar as características dos bancos de dados orientados a grafos em comparação com os bancos de dados relacionais;
- Explorar as funcionalidades e recursos dos bancos de dados orientados a grafos usando o Neo4j;
- Realizar um estudo de caso prático utilizando o Neo4j.

A justificativa para este trabalho baseia-se nos seguintes aspectos:

- Relevância das estruturas de grafos: Os bancos de dados orientados a grafos permitem uma representação de um sistema sem esquema das relações complexas entre os dados;
- Crescente adoção do Neo4j: O Neo4j é um exemplo proeminente de sistema de gerenciamento de banco de dados orientado a grafos;
- Contribuição para a área acadêmica e profissional: Este estudo contribuirá para a disseminação do conhecimento sobre bancos de dados orientados a grafos e, mais especificamente, sobre o Neo4j.

Diante desses aspectos, fica evidente a importância de realizar uma análise aprofundada sobre bancos de dados orientados a grafos, dando ênfase ao Neo4j.

1.2 ORGANIZAÇÃO DO TRABALHO

A pesquisa está dividida em 6 capítulos pertinentes aos objetivos.

O Capítulo 1 faz uma introdução ao tema abordado e uma contextualização dos objetivos desse trabalho

O Capítulo 2 contém os conceitos importantes de banco de dados.

No Capítulo 3 é feita uma breve descrição do que são os grafos para poder obter uma maior compreensão do trabalho.

Em seguida no Capítulo 4 é mais aprofundado sobre os bancos de dados orientados a grafos com o foco no Neo4j e suas operações dentro do banco.

No Capítulo 5 é realizada uma aplicação realizando a integração do Neo4j com outra tecnologia e aproveitando de suas funcionalidades.

E por último, o Capítulo 6 aborda sobre as considerações finais.

2 CONCEITOS SOBRE BANCO DE DADOS

Esse capítulo passa a visão geral sobre os bancos de dados, começando com alguns conceitos e modelos para fazer essa ilustração, logo após é feito uma breve introdução aos bancos de dados relacionais, e em seguida é repassado conceitos de bancos de dados não relacionais juntamente com suas características.

2.1 BANCO DE DADOS: CONCEITOS E MODELOS

Um banco de dados é uma coleção organizada de dados relacionados, projetada para atender às necessidades de armazenamento, recuperação, atualização e análise eficientes desses dados (SILBERSCHATZ, 2020). Os bancos de dados desempenham um papel fundamental em várias aplicações, desde sistemas de gerenciamento de informações empresariais até aplicações da web e dispositivos móveis.

Apoiando a estrutura de um banco de dados está o modelo de dados: uma coleção de ferramentas conceituais para descrever dados, relações de dados, semântica de dados e restrições de consistência (SILBERCHATZ, 2020). Esses modelos podem ser classificados em quatro categorias diferentes:

- Modelo relacional: Esse modelo usa uma coleção de tabelas para representar os dados e as relações entre eles. Cada tabela possui diversas colunas, cada qual com um único nome. Tabelas também são chamadas de relações;
- Modelo de entidade/relacionamento (E-R): O modelo E-R utiliza uma coleção de objetos básicos, chamados entidades, e os relacionamentos entre esses objetos. Uma entidade é uma “coisa” ou “objeto” no mundo real que é distinguível dos outros objetos;
- Modelo de dados semiestruturado: Esse modelo semiestruturado permite a especificação dos dados em que itens de dados individuais do mesmo tipo possam ter diferentes conjuntos de atributos.
- Modelo de dados baseado em objeto: Este modelo define o banco de dados como uma coleção de objetos, ou elementos de software reutilizáveis, com recursos e métodos associados.

2.2 BANCO DE DADOS RELACIONAIS

Um banco de dados relacional é um tipo de banco de dados que armazena e fornece acesso a pontos de dados relacionados entre si, são baseados no modelo relacional, uma maneira intuitiva e direta de representar dados em tabelas. Em um banco de dados relacional, cada linha

na tabela é um registro com um identificador exclusivo chamada chave. As colunas da tabela contêm atributos dos dados e cada registro geralmente tem um valor para cada atributo (ORACLE, 2023).

Os bancos de dados relacionais têm sido amplamente utilizados como a forma tradicional de armazenamento e gerenciamento de dados. Silberschatz (2020) afirma que “um banco de dados relacional é uma coleção de tabelas, onde cada tabela é composta por linhas e colunas, representando uma entidade e seus atributos”. Esses bancos de dados são baseados no modelo relacional, que organiza os dados em tabelas relacionadas por meio de chaves primárias e estrangeiras.

Esses bancos de dados oferecem uma estrutura rigorosa e uma linguagem padronizada (SQL) para consulta e manipulação dos dados. Eles são altamente eficientes para lidar com relacionamentos complexos e garantem a integridade dos dados por meio de restrições de integridade e transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) (SILBERCHATZ, 2020).

2.3 BANCO DE DADOS NÃO RELACIONAIS

Com o crescimento exponencial dos dados não estruturados e a necessidade de escalabilidade, surgiram os bancos de dados não relacionais ou NoSQL. Sadalage e Fowler (2019) definem os bancos de dados NoSQL como “uma categoria de bancos de dados que oferece uma abordagem alternativa de armazenamento e acesso aos dados, projetada para atender aos requisitos de escalabilidade, disponibilidade e desempenho”.

Os bancos de dados NoSQL atuam sem um esquema, permitindo que sejam adicionados, livremente, campos aos registros do banco de dados, sem ter de definir primeiro quaisquer mudanças na estrutura. Isso é especialmente útil ao lidar com dados não uniformes e campos personalizados (SADALAGE; FOWLER, 2019).

Os bancos de dados não relacionais abrangem uma variedade de tipos, cada um projetado para atender a requisitos específicos de armazenamento e acesso aos dados. Entre os principais tipos de bancos de dados NoSQL, destacam-se (SILVA; RIVA; ROSA, 2021):

- Bancos de dados de documentos: Esses bancos armazenam e recuperam dados no formato de documentos, geralmente usando um formato semelhante ao JSON ou XML. Os dados são estruturados de forma encadeada, podendo conter atributos das coleções, tags e metadados, e seguem uma hierarquia de informações. Eles são flexíveis em termos de esquema, permitindo que diferentes documentos dentro da

mesma coleção tenham estruturas diferentes. Exemplos populares incluem MongoDB[1] e CouchDB[2].

- Bancos de dados de colunas: Nesse tipo de banco é permitido que o usuário armazene valores em chaves mapeadas, os quais são agrupados em diversas famílias de colunas, como um mapa. Então é feito a armazenagem dos dados em colunas, em vez de linhas, permitindo uma recuperação eficiente de conjuntos de dados específicos. Eles são particularmente adequados para análises e consultas complexas em grandes volumes de dados. Exemplos notáveis são o Cassandra[3] e o HBase[4].
- Bancos de dados de chave-valor: Esses bancos de dados associam uma chave única a um valor, permitindo uma recuperação rápida de dados por meio de consultas de chave. São ideais para armazenar e recuperar dados simples, como cache e sessões de usuários. São exemplos populares o Redis[5] e o Riak[6].
- Bancos de dados orientado a grafos: Esses bancos de dados são projetados para armazenar e consultar dados relacionais complexos, com foco nas relações entre os elementos dos dados. Eles são adequados para aplicações que exigem a análise de redes complexas e a navegação por grafos. O Neo4j[7] é um exemplo proeminente de banco de dados de grafos.
- Outro banco de dados não relacional é o InfluxDB[8], ele faz parte de um tipo que é o banco de dados de séries temporais, projetado para armazenar e analisar grandes volumes de dados com base em métricas, eventos ou medições com registro de data e hora. Ele é especialmente adequado para aplicações que lidam com dados temporais, como monitoramento de infraestrutura, sensoriamento IoT e análise em tempo real (NAQVI; YFANTIDOU; ZIMÁNYI; 2017).

[1] https://www.mongodb.com/
[2] https://couchdb.apache.org/
[3] https://cassandra.apache.org/_/index.html
[4] https://hbase.apache.org/
[5] https://redis.io/
[6] https://riak.com/
[7] https://neo4j.com/
[8] https://www.influxdata.com/

3 GRAFOS

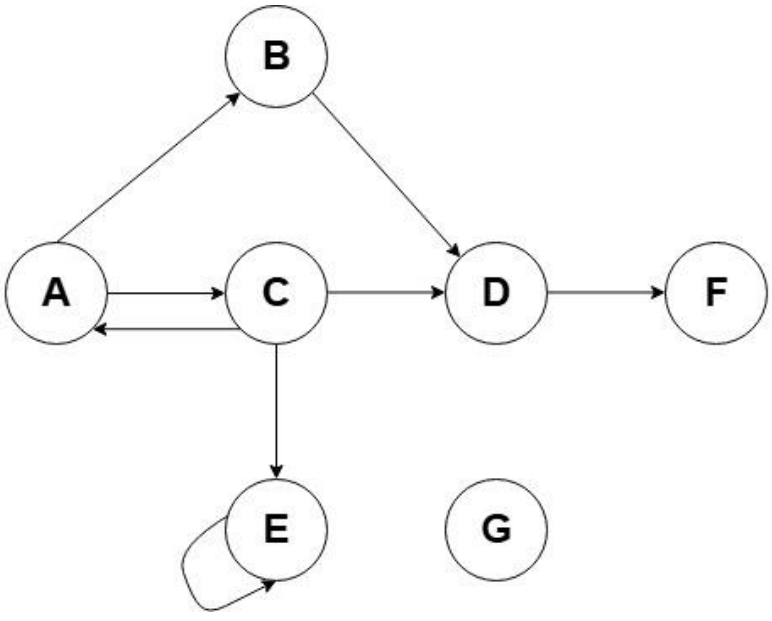
Esse capítulo começa descrevendo conceitos gerais sobre a teoria dos grafos, pois são a base de estudo para obter uma maior compreensão do que são os bancos de dados orientado a grafos.

3.1 GRAFOS: CONCEITOS

Grafos são estruturas de dados que consistem em um conjunto de nós (também conhecidos como vértices) com um conjunto de arcos (ou arestas). Eles são amplamente utilizados como um modelo para representar e analisar relações entre objetos ou entidades. Os grafos fornecem uma forma poderosa de visualizar e descrever interconexões complexas e relacionamentos entre elementos (TENENBAUM; LANGSAM; AUGENSTEIN, 1995).

Na Figura 3.1 é representado de forma gráfica uma estrutura de grafos que consiste na seguinte sequência de nós {A,B,C,D,E,F,G}, e o conjunto de arestas {(A,B), (A,C), (B,D), (C,D), (C,E), (D,F), (E,E)}, onde as arestas são as setas orientadas que unem os vértices que por sua vez são representados pelos pontos ou círculos.

Figura 3.1 - Representação gráfica de um grafo



Fonte: Autor

Nós ligados por arcos são ditos adjacentes, o nó C é adjacente ao nó A, D e E. Diz também que arestas são incidentes de ou a determinados nós, conforme partam ou cheguem a eles; (A,C) é incidente de A e (C,A) é incidente à A. Embora os nós A e F não estejam ligados

por uma aresta é possível a partir do nó A chegar ao nó F, percorrendo as arestas {A,C}, {C,D} e {D,F}. Tais arestas constituem um caminho de A à F. Um caminho é definido como sequência de uma ou mais arestas (VELOSO; et al, 1983).

As operações em grafos são as ações realizadas para manipular e explorar as estruturas de dados do grafo. Essas operações permitem extrair informações, identificar padrões, percorrer caminhos, calcular métricas e realizar análises diversas. A seguir, são abordadas algumas das operações mais comuns em grafos (VELOSO; et al, 1983):

- Adição e remoção de nós e arestas: Essas operações consistem em inserir ou remover novos nós e arestas do grafo. Podendo expandir a estrutura do grafo, adicionando elementos e estabelecendo novas conexões entre eles ou eliminar elementos desnecessários ou indesejados.
- Busca em profundidade (DFS): Essa operação é um algoritmo de travessia de grafos que se concentra em explorar o máximo possível em uma ramificação antes de retroceder. Ela começa em um nó de partida e, em seguida, explora cada um de seus vizinhos antes de continuar a se aprofundar na próxima ramificação. Isso é feito de forma recursiva, explorando a árvore do grafo até que todos os caminhos sejam percorridos ou até que um destino específico seja alcançado.
- Busca em largura (BFS): por outro lado, também é um algoritmo de travessia de grafos porém, que explora os nós em níveis sucessivos de distância a partir do nó de partida. Ela começa pelo nó inicial e, então, explora todos os seus vizinhos antes de passar para os vizinhos dos vizinhos. Isso é feito de forma iterativa, explorando a largura do grafo antes de ir mais fundo.
- Algoritmo de menor caminho: Esses algoritmos calculam a menor distância entre dois nós ou a menor soma dos pesos das arestas ao longo do caminho e são usados para encontrar o caminho mais curto entre dois nós em um grafo ponderado. Exemplos populares são o algoritmo de Dijkstra e o algoritmo de Floyd-Warshall. Eles são aplicados em problemas de otimização de rotas, planejamento de transporte e redes de comunicação (TENENBAUM; LANGSAM; AUGENSTEIN, 1995).

4 BANCO DE DADOS ORIENTADO A GRAFOS

Os bancos de dados orientados a grafos são sistemas de gerenciamento de banco de dados projetados especificamente para armazenar e manipular dados com base no modelo de grafos. Esses bancos de dados são projetados para lidar com a representação e o processamento eficiente de relações complexas entre entidades.

No modelo de banco de dados orientado a grafos, as informações são organizadas em nós (vértices) e arestas, que representam as relações entre os nós. Cada nó pode conter propriedades que descrevem suas características, enquanto as arestas podem ter rótulos e propriedades adicionais para fornecer informações contextuais sobre a relação (SILVA; RIVA; ROSA, 2021).

Ao contrário dos bancos de dados relacionais, onde as informações são armazenadas em tabelas com relacionamentos pré-definidos, os bancos de dados orientados a grafos oferecem maior flexibilidade e escalabilidade para representar relações complexas. Esses bancos de dados são capazes de lidar com grandes quantidades de dados interconectados e são particularmente adequados para casos de uso que envolvem consultas de relacionamentos e navegação complexa por dados conectados. Um exemplo desse tipo de banco de dados é o Neo4J que é descrito a seguir.

4.1 NEO4J

Este banco de dados é uma solução de software de código aberto, desenvolvida em Java, que inclui uma linguagem de consulta conhecida como Cypher. A manipulação dos dados armazenados no banco de dados Neo4j pode ser feita usando uma API RESTful compatível com várias linguagens de programação ou por meio do console de comando incorporado fornecido pela plataforma (SILVA; RIVA; ROSA, 2021).

A seguir será abordado sobre sua manipulação utilizando o Cypher e explicando cada uma das instruções realizadas nesse banco de dados.

4.2 USANDO O NEO4J

Para realizar as consultas no Neo4j, é utilizado o Neo4j Sandbox[9], que é uma plataforma online que permite explorar e experimentar o Neo4j de forma rápida e fácil, sem a necessidade de configuração ou instalação local. Trata-se de um ambiente de teste virtual para começar a trabalhar com o Neo4j e experimentar seus recursos.

[9] https://neo4j.com/sandbox/

Dentro do Neo4j Sandbox, é utilizado a interface do Neo4j Browser, que é uma ferramenta de visualização e consulta. Com o Browser, é possível escrever consultas em Cypher, e executá-las diretamente no banco de dados. Assim permitindo explorar e analisar os dados de forma interativa.

4.2.1 Exemplos de operações

Para usar o Neo4j é utilizado a linguagem de consulta Cypher, ela seria como um equivalente ao SQL dos bancos de dados relacionais. Com suas semelhanças, Cypher permite os usuários focarem no que querem recuperar do grafo, em vez de como recuperá-lo (NEO4J).

As principais operações no Neo4j são mostradas a seguir em uma base de dados que consiste em uma espécie de rede social. O exemplo é de autoria do autor.

Operação de adição de nós e arestas: nessa operação é utilizada a cláusula “CREATE” para criar os nós e arestas no banco de dados de grafos. Então ela será utilizada para criar a base de dados conforme mostra na Figura 4.1.

Figura 4.1 - Operação de adição

```
# Nós que representam as pessoas
CREATE (Alice: Pessoa {nome: 'Alice'})
CREATE (Bob: Pessoa {nome: 'Bob'})
CREATE (Charlie: Pessoa {nome: 'Charlie'})
CREATE (David: Pessoa {nome: 'David'})
CREATE (Eve: Pessoa {nome: 'Eve'})
CREATE (Frank: Pessoa {nome: 'Frank'})
CREATE (Grace: Pessoa {nome: 'Grace'})
CREATE (Hannah: Pessoa {nome: 'Hannah'})
CREATE (Teste: Pessoa {nome: 'Teste'})

# Relacionamentos/arestas de amizade
CREATE (Alice)-[:AMIGO]->(Bob)
CREATE (Alice)-[:AMIGO]->(Charlie)
CREATE (Bob)-[:AMIGO]->(David)
CREATE (Bob)-[:AMIGO]->(Eve)
CREATE (Charlie)-[:AMIGO]->(Frank)
CREATE (Frank)-[:AMIGO]->(Grace)
CREATE (Grace)-[:AMIGO]->(Hannah)
CREATE (Teste)-[:AMIGO]->(Frank)
```

Fonte: Autor

Operação de remoção de nós e arestas: essa operação consiste na utilização da cláusula “DELETE” para remover um ou mais nós e/ou relacionamentos no banco de dados. A seguir, na Figura 4.2, é apresentado um exemplo de como fazer a remoção de um relacionamento, ao realizar essa operação será removido somente o relacionamento entre os nós indicados.

Figura 4.2 - Operação de remoção de aresta

```
MATCH (n:Pessoa {nome: 'Teste'})-[r:AMIGO]->(n2:Pessoa {nome: 'Frank'})
DELETE r
```

Fonte: Autor

E para realizar a remoção de um nó, não há muitas mudanças, aqui apenas é acrescentado o comando “DETACH” que pode ser opcional, porém ele serve para excluir o nó junto com os seus respectivos relacionamentos, como é possível visualizar na Figura 4.3.

Figura 4.3 - Operação de remoção de nó

```
MATCH (n: Pessoa {nome: 'Teste'})
DETACH DELETE n
```

Fonte: Autor

Busca em profundidade: para realizar uma busca em profundidade no Neo4j, é utilizada a cláusula “MATCH” para especificar o ponto de partida (nó inicial) e, também, usada a recursão para percorrer os nós vizinhos em profundidade. A seguir, na Figura 4.4, é apresentado um exemplo de como é feita essa operação.

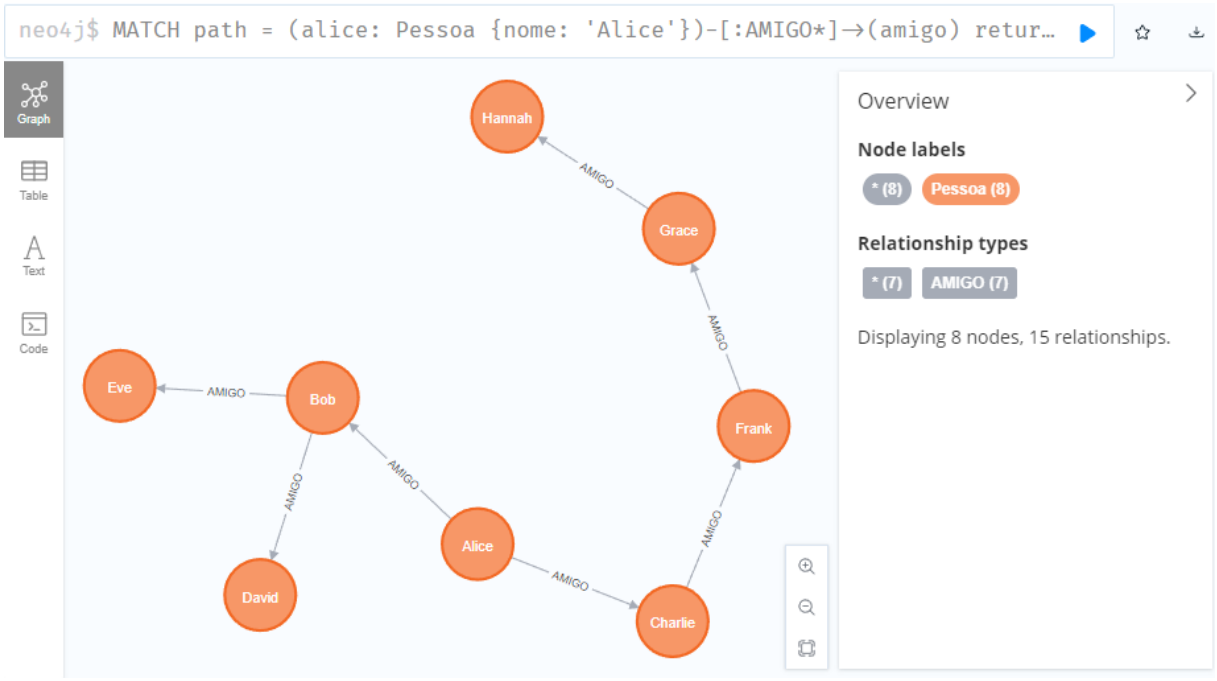
Figura 4.4 - Operação de busca em profundidade

```
MATCH path = (alice: Pessoa {nome: 'Alice'})-[:AMIGO*]->(amigo)
RETURN path
```

Fonte: Autor

Neste exemplo, é usado o relacionamento “[:AMIGO*]” com o operador “*” para indicar que desejamos obter todos os relacionamentos “AMIGO” em profundidade. O resultado é uma lista de amigos em profundidade a partir do nó "alice". Na Figura 4.5 é apresentado o resultado dessa consulta.

Figura 4.5 - Resultado da busca em profundidade



Fonte: Autor

Busca em largura: para executar a operação de busca em largura no Neo4j, novamente é utilizada a cláusula “MATCH”, mas diferente da busca em profundidade, essa busca explora todos os nós em um nível antes de passar para o próximo nível no grafo. Na Figura 4.6 é apresentado um exemplo da busca em largura no primeiro nível.

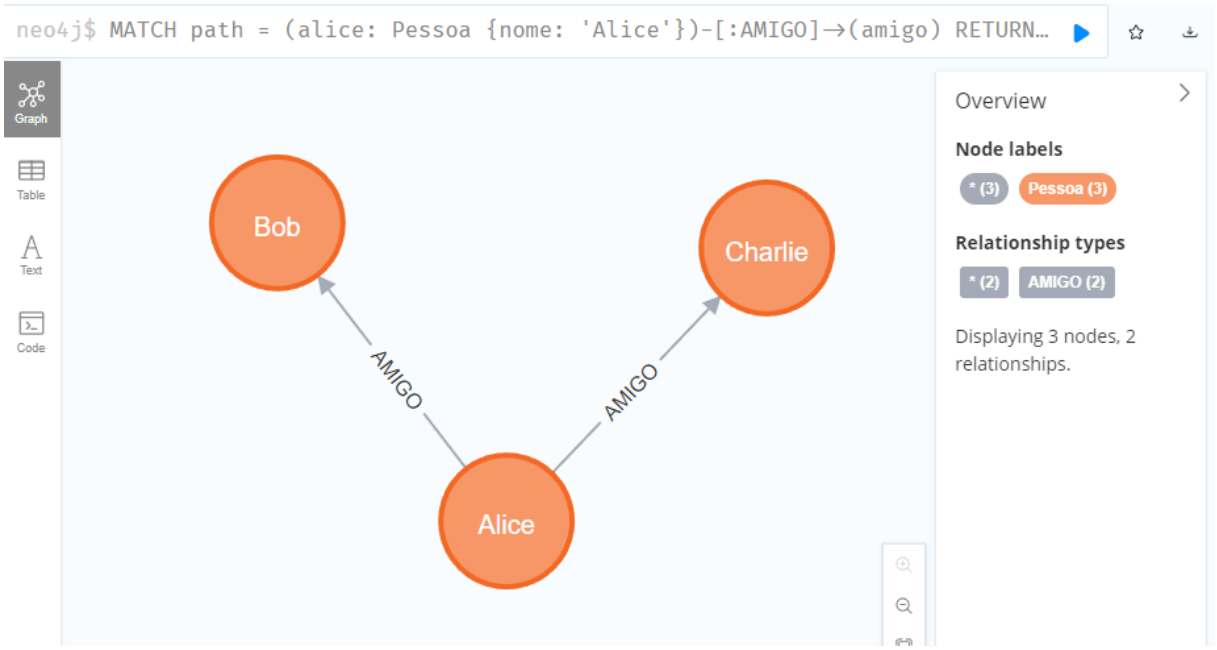
Figura 4.6 - Operação de busca em largura em primeiro nível

```
MATCH path = (alice: Pessoa {nome: 'Alice'})-[:AMIGO]->(amigo)
RETURN path
```

Fonte: Autor

No exemplo anterior, é realizada uma busca em largura dos amigos de “Alice” em um primeiro nível utilizando o relacionamento “AMIGO”, com o seguinte resultado, onde é apresentado os amigos de “Alice” no primeiro nível conforme mostra a Figura 4.7:

Figura 4.7 - Resultado da busca em largura em primeiro nível



Fonte: Autor

Então para o próximo nível da busca em largura, é demonstrado o seguinte exemplo a seguir na Figura 4.8, adicionando mais comandos da consulta realizada anteriormente:

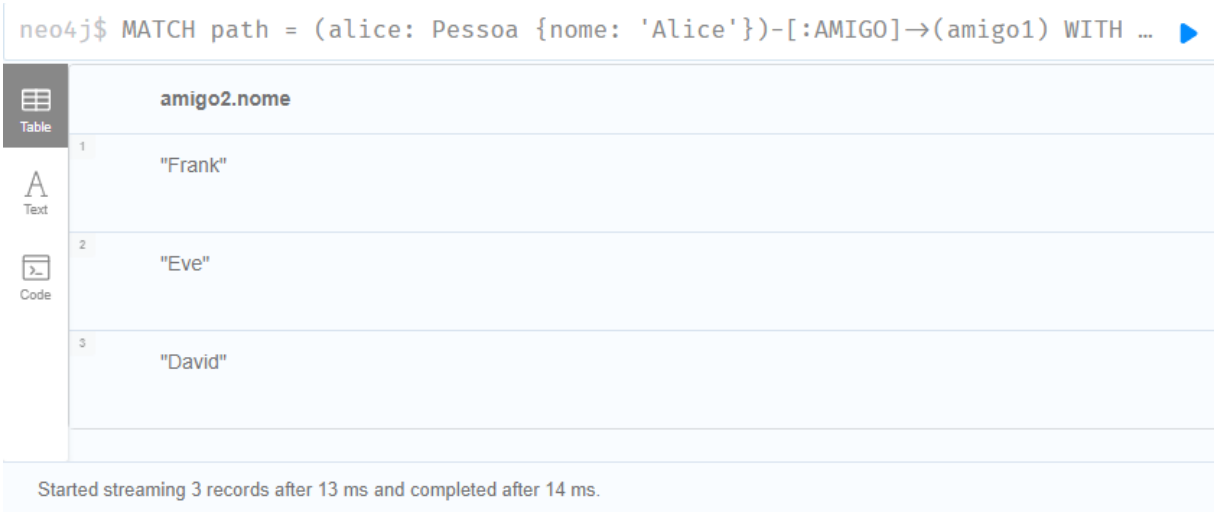
Figura 4.8 - Operação de busca em largura em segundo nível

```
MATCH path = (alice: Pessoa {nome: 'Alice'})-[:AMIGO]->(amigo1)
WITH amigo1
MATCH (amigo1)-[:AMIGO]->(amigo2)
RETURN amigo2.nome
```

Fonte: Autor

Nesta consulta, é realizada a mesma busca anterior, porém após recebermos o primeiro resultado dos seus amigos diretos (amigo1), estes são utilizados com a cláusula “WITH” e então é passado para a próxima parte da consulta, onde são encontrados os amigos diretos do “amigo1” (amigo2). Então no final, é retornado o nome dos amigos diretos do segundo nível, conforme ilustra a Figura 4.9.

Figura 4.9 - Resultado da busca em largura em segundo nível



Fonte: Autor

Algoritmo de menor caminho: para encontrar o caminho mais curto entre dois nós em um grafo, será utilizado o algoritmo de Dijkstra, com base no seguinte exemplo onde é apresentado os nós de “Cidade” e os relacionamentos de “CONEXAO”.

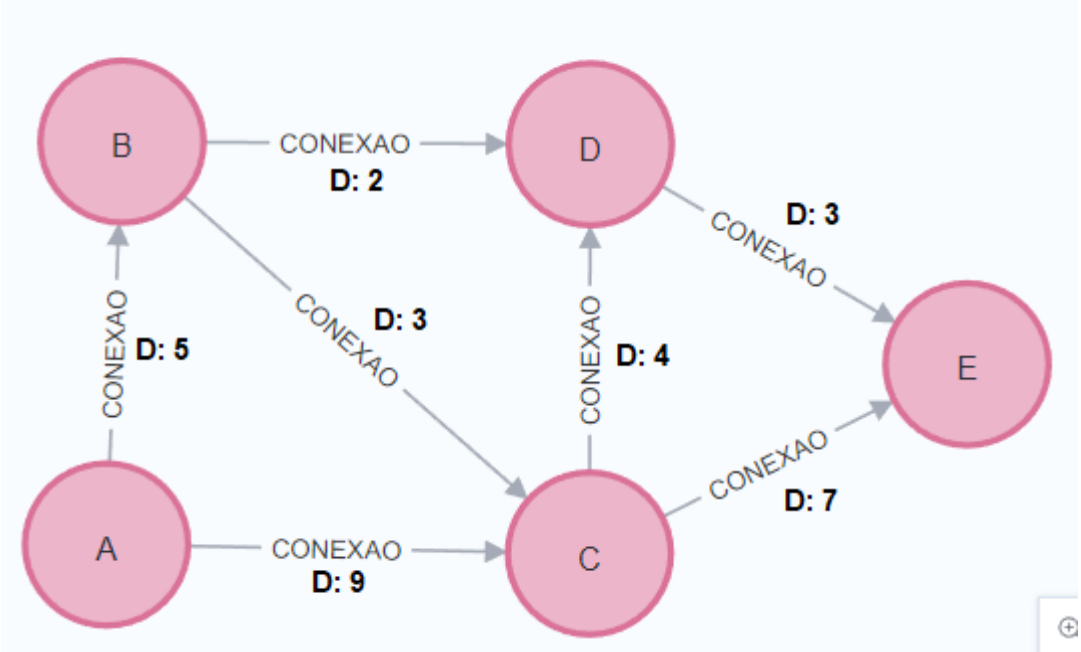
Figura 4.10 - Base de dados para algoritmo de menor caminho

```
# Nós
(A: Cidade {nome: 'A'})
(B: Cidade {nome: 'B'})
(C: Cidade {nome: 'C'})
(D: Cidade {nome: 'D'})
(E: Cidade {nome: 'E'})

# Relacionamentos
(A)-[:CONEXAO {distancia: 5}]->(B)
(A)-[:CONEXAO {distancia: 9}]->(C)
(B)-[:CONEXAO {distancia: 2}]->(D)
(B)-[:CONEXAO {distancia: 3}]->(C)
(C)-[:CONEXAO {distancia: 4}]->(D)
(C)-[:CONEXAO {distancia: 7}]->(E)
(D)-[:CONEXAO {distancia: 3}]->(E)
```

Fonte: Autor

Com essa base de dados, conforme é observado na Figura 4.10, temos a seguinte representação na forma de grafos para visualizar melhor, onde a letra “D” representa a distância da “CONEXAO” entre as cidades que será utilizada para realizar o cálculo e descobrir qual o menor caminho entre duas cidades.

Figura 4.11 - Representação no grafo



Fonte: Autor

Conforme a Figura 4.11 faz a ilustração, então será feito a consulta de menor caminho entre os nós da cidade “A” e cidade “E”, para isso é utilizado a biblioteca APOC (Incríveis procedimentos no Cypher), esta é uma biblioteca que provê uma série de procedimentos e funções úteis para usar no Cypher. Na Figura 4.12 é possível observar a seguinte consulta.

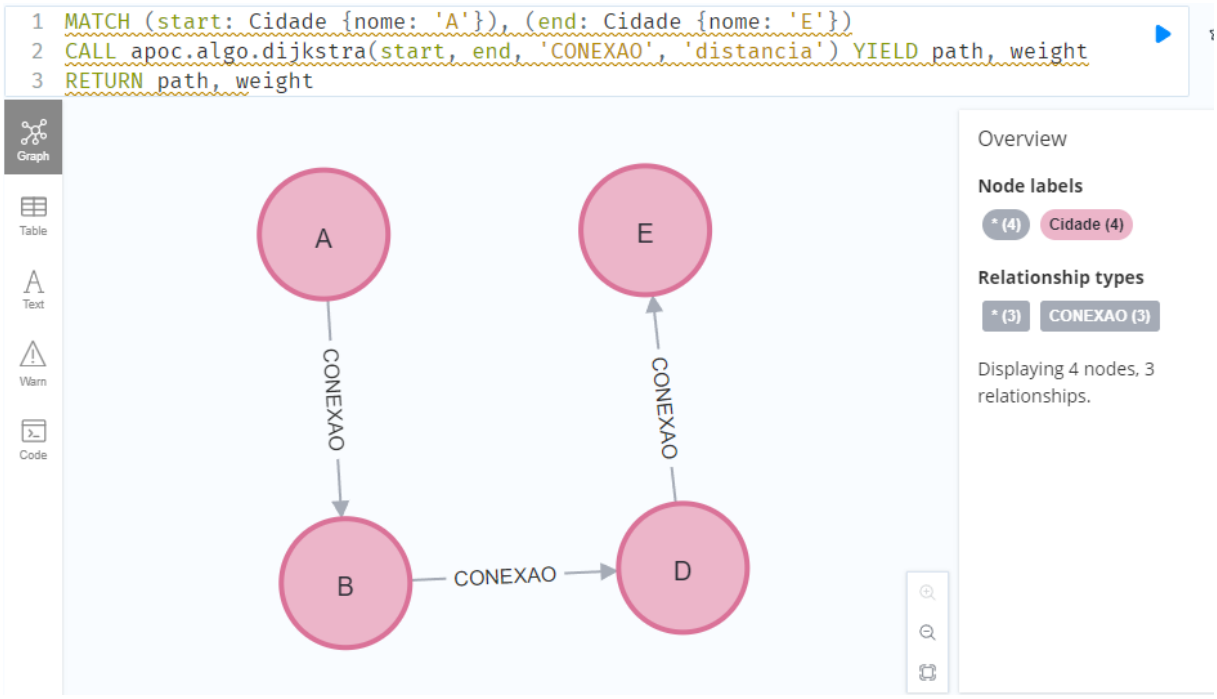
Figura 4.12 - Operação de algoritmo de menor caminho

```
MATCH (start: Cidade {nome: 'A'}), (end: Cidade {nome: 'E'})
CALL apoc.algo.dijkstra(start, end, 'CONEXAO', 'distancia') YIELD path, weight
RETURN path, weight
```

Fonte: Autor

Com esta biblioteca então é utilizado o algoritmo de Dijkstra, passando como parâmetros, os nós das cidades “A” e “E” ambos como o “start” e “end”, também é passado o nome do relacionamento entre ambos e então a métrica que será utilizada para fazer a medição, que nesse caso é a “distancia”, e, portanto, é retornado o caminho e o tamanho total do caminho que será percorrido para realizar este percurso, que é possível visualizar na Figura 4.13.

Figura 4.13 - Resultado da operação de menor caminho



Fonte: Autor

5 USO DO NEO4J EM UM SISTEMA

Para uma maior profundidade e exploração dos recursos existentes no Neo4j foi desenvolvido um aplicativo que permite a visualização interativa do mapa da UFSM, destacando seus diferentes prédios.

O aplicativo oferece uma funcionalidade de filtro que permite ao usuário pesquisar por uma área de conhecimento específica associada a um ou mais locais dentro do Campus e então é retornado os prédios que são associados àquela área. Além disso, ao aplicar esse filtro, o aplicativo utiliza a localização atual do usuário para identificar e destacar o prédio mais próximo que atenda aos critérios de busca bem como a parada de ônibus mais próxima (pensando em usuários que usam este meio de transporte).

5.1 TECNOLOGIAS UTILIZADAS

As tecnologias utilizadas no desenvolvimento desse aplicativo foram o Flutter para a visualização do aplicativo e o Java Spring Boot para realizar a manipulação de dados do Neo4j. Para trabalhar com os dados referentes ao mapa do Campus da UFSM foi utilizado o QGIS, que é um software de código aberto e gratuito utilizado para visualização, edição e análise de dados geoespaciais. Ele oferece uma ampla gama de funcionalidades, incluindo a criação de mapas, a manipulação de dados vetoriais, etc., além da alta variedade de formatos de dados geoespaciais suportados.

A base de dados utilizada do mapa do Campus da UFSM foi fornecida pelo curso de Geoprocessamento da UFSM, disponível no link Mapa do Campus Sede da UFSM[10], e foi elaborada por Leonardo Gabriel Rischter e Marcos Corrêa Silveira com a orientação da Profª. Drª. Franciele Rovani.

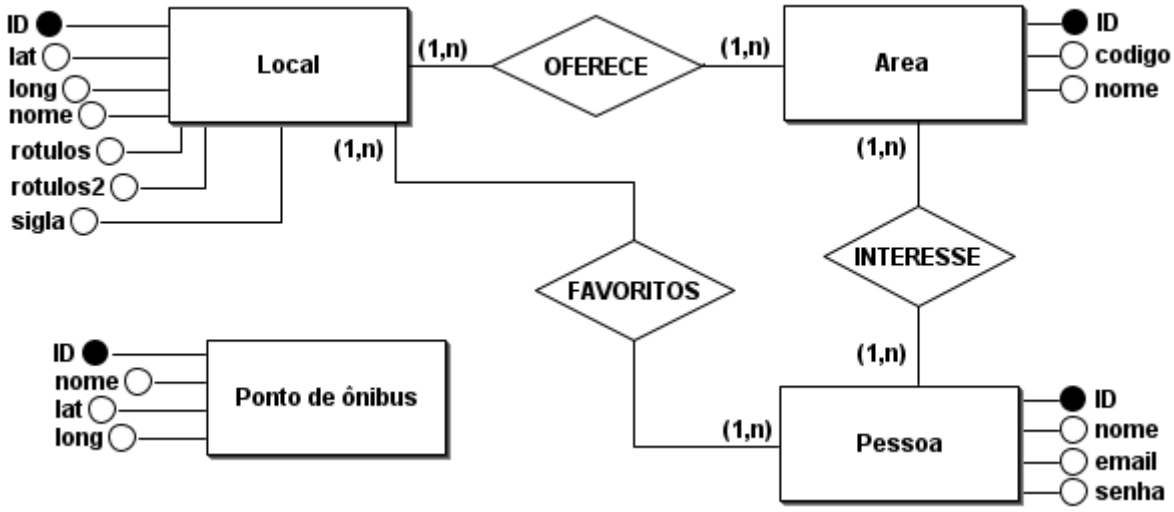
5.2 ESTRUTURA DO BANCO

A estrutura do banco é representada da seguinte forma com o diagrama entidade-relacionamento (ER) conforme é possível visualizar na Figura 5.1. No diagrama ER existe a entidade “Local” que representa os prédios da UFSM e que está relacionada à entidade “Area” através do relacionamento “OFERECE”, indicando quais áreas de conhecimento humano cada prédio oferece. A entidade “Pessoa” é utilizada para armazenar os dados dos usuários e possui

[10] geoprocessamento.maps.arcgis.com/apps/instant/portfolio/index.html?appid=6551639d5a8d40c4ae09792012154bba

dois relacionamentos, sendo um deles o de “INTERESSE” com a entidade “Area”, permitindo identificar as áreas de interesse de cada pessoa e o segundo sendo os “FAVORITOS” que se refere aos locais salvos como favoritos de uma pessoa. Por último, a entidade “Ponto” armazena dados sobre as paradas de ônibus disponíveis, facilitando a visualização de pontos de transporte dentro do campus. Esse diagrama ER oferece uma representação das conexões entre os vários elementos do sistema.

Figura 5.1 - Diagrama ER



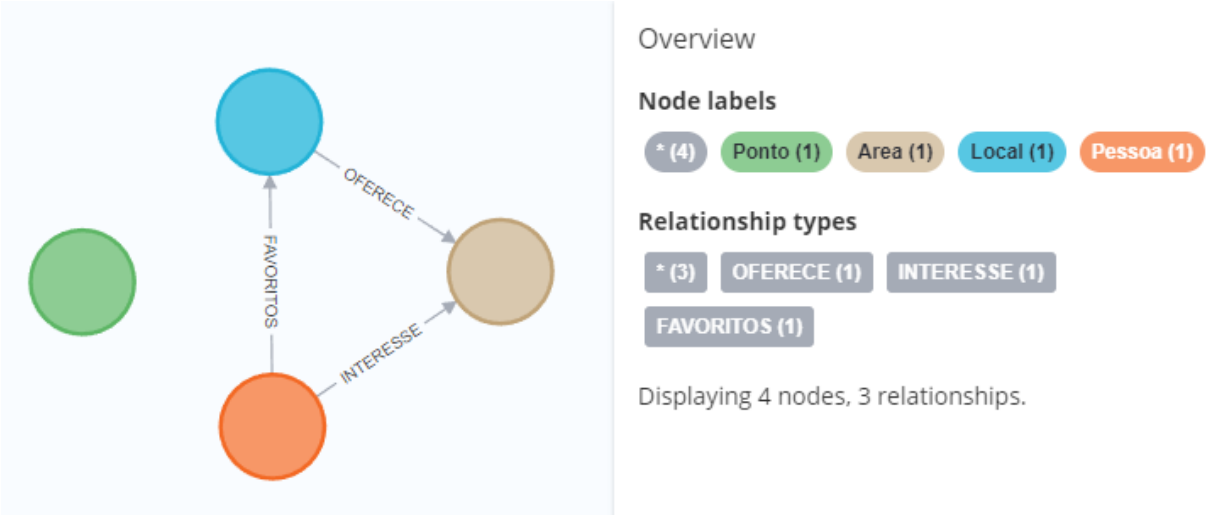
Fonte: Autor

5.2.1 Estrutura no Neo4j

Para explicar a estrutura definida no modelo ER no modelo do Neo4j, é importante entender que, ao contrário dos bancos de dados relacionais tradicionais que utilizam tabelas, o Neo4j é um banco de dados orientado a grafos que usa nós para representar entidades e arestas (ou relações) para representar os relacionamentos entre essas entidades.

Na Figura 5.2 é possível observar como o diagrama ER ficou armazenado no banco de dados Neo4j, onde estão as entidades Local, Área, Pessoa e Ponto que se refere ao Ponto de ônibus, com os relacionamento de “OFERECE” entre o Local e Área, para poder indicar quais áreas do conhecimento são ofertadas naquele local em específico, o relacionamento “INTERESSE” que tem como ligação a Pessoa e a Área que se refere às áreas de interesse que uma pessoa possa possuir, sendo uma ou mais áreas de interesse e vice-versa e por último o relacionamento de “FAVORITOS” entre a Pessoa e o Local, que indica quais são os locais favoritos de uma pessoa, sendo um ou mais.

Figura 5.2 - Estrutura no Neo4j



Fonte: Autor

As áreas representam diferentes campos de conhecimento humano presente nas universidades, e cada uma possui propriedades específicas como id, nome e código de classificação. Foi utilizada a classificação do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) como base para definir essas áreas, garantindo que a estrutura seja padronizada e por ser reconhecida nacionalmente. Essa padronização facilita a categorização e a busca pelos prédios que oferecem cursos ou atividades relacionadas a cada área. As áreas inseridas abrangem uma ampla gama de disciplinas, desde ciências exatas e engenharias até ciências humanas e sociais, permitindo uma organização detalhada e coerente dos recursos da universidade.

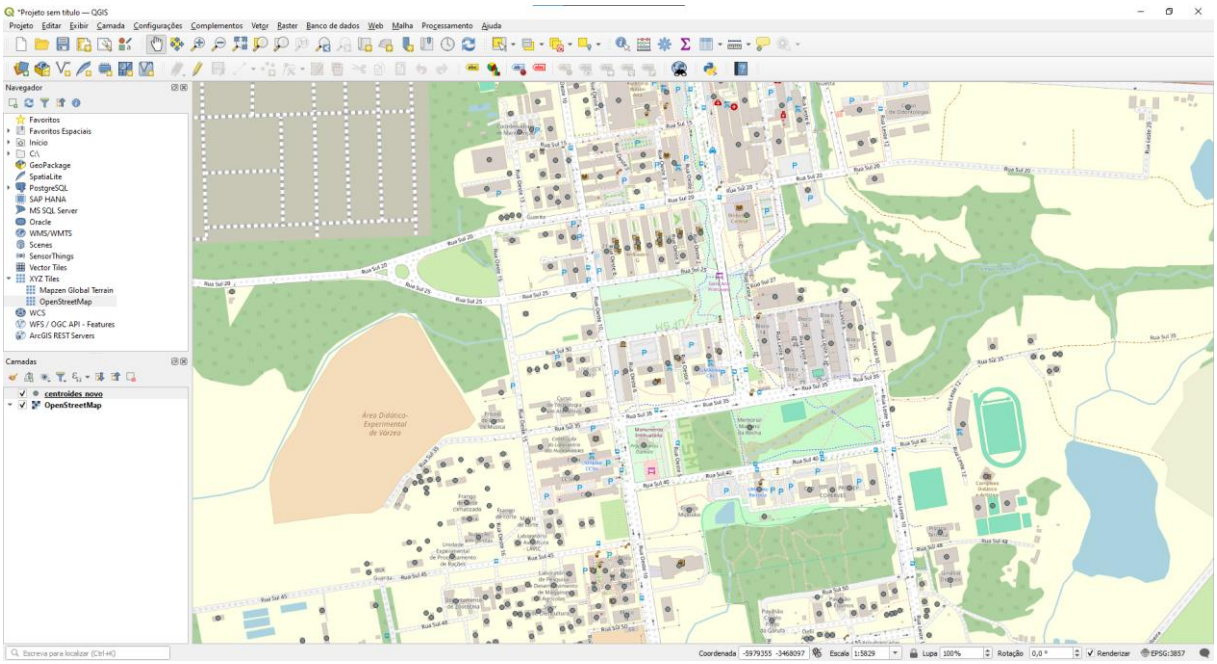
Para fazer a inserção das áreas no banco de dados Neo4j, foi desenvolvido um script em Python que gerou o código necessário para a inclusão dessas informações. O script utilizou os dados do CNPq para criar cada área com suas respectivas propriedades, automatizando o processo e garantindo a precisão dos dados inseridos. Após a geração do código, as áreas foram inseridas no Neo4j, permitindo que o sistema consiga realizar consultas eficientes sobre as áreas de conhecimento oferecidas pelos diferentes prédios da UFSM. Esse processo não só facilitou a inserção inicial dos dados, mas também proporcionou uma base sólida para esse estudo de caso.

A entidade Local é utilizada para representar os prédios da UFSM e possui várias propriedades essenciais como id, lat (latitude), long (longitude), nome, rotulos, rotulos2 e sigla. As propriedades lat e long servem para armazenar a localização geoespacial detalhada dos prédios, a propriedade nome tem como objetivo ser a identificação do prédio e as propriedades

rotulos e rotulos2 refere-se descrições complementares (o Prédio F possui como rótulos 70F e Colégio Politécnico).

A base de dados fornecida veio no formato *Shapefile*, que é um formato de arquivos amplamente utilizado para armazenar dados geoespaciais em Sistemas de Informações Geográficas (SIG). Para manipular e processar esses dados, foi utilizado o QGIS, um software de código aberto que permite a visualização, edição e análise de dados geoespaciais de maneira eficiente conforme é demonstrado na Figura 5.3.

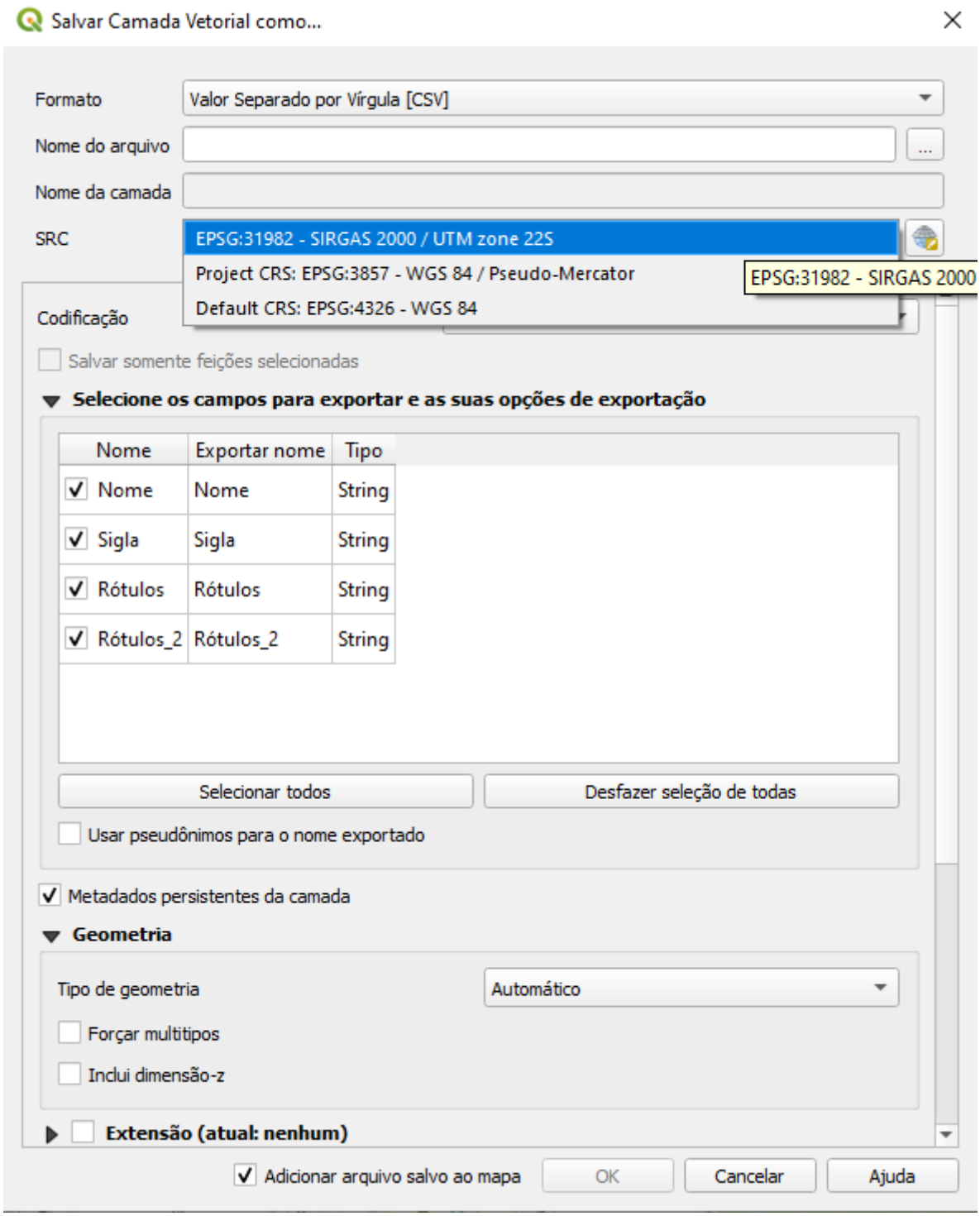
Figura 5.3 - Visualização no QGIS



Fonte: Autor

Inicialmente, o arquivo fornecido estava no formato EPSG:31982 - SIRGAS 2000 / UTM zone 22S, que é um sistema de referência de coordenadas específico para o Brasil. No entanto, para garantir a compatibilidade com o padrão global utilizado pelo Google Maps, foi necessário mudar o arquivo para o formato EPSG:4326 - WGS 84 como é possível ver na Figura 4.24. Este sistema de coordenadas é amplamente adotado para aplicações web e garante que os dados geoespaciais sejam interpretados corretamente em diferentes plataformas.

Figura 5.4 - Reprojeto do shapefile



Fonte: Autor

Após reprojetar os dados, foi exportado o shapefile para um arquivo de formato CSV. Esse formato é mais adequado para poder realizar a importação de dados no Neo4j, então foi utilizado a função LOAD CSV do Neo4j, que permite a importação de grandes volumes de dados de forma eficiente. Essa função leu o arquivo CSV e povoou o banco de dados com as

informações dos prédios da UFSM, garantindo que cada “Local” fosse corretamente identificado e posicionado no mapa interativo.

A entidade “Pessoa” é utilizada para guardar as informações sobre os usuários. Cada “Pessoa” possui as seguintes propriedades: id, nome, email e senha. O id é um identificador único para cada usuário, garantindo que cada indivíduo seja único dentro do banco de dados, o nome armazena o nome completo do usuário, enquanto o email é utilizado tanto para identificação quando o usuário realizar o login no aplicativo.

Além das informações pessoais, essa entidade pode estar relacionada a várias áreas de interesse, permitindo a personalização da experiência do usuário com base nas suas preferências e necessidades acadêmicas. Isso facilita a recomendação de prédios e recursos específicos dentro do campus que correspondam às áreas de conhecimento de interesse do usuário.

E por fim a entidade “Ponto de ônibus” que armazena informações sobre as paradas de ônibus disponíveis no campus. Cada "Ponto" possui as seguintes propriedades: nome, lat (latitude) e long (longitude). O nome identifica a parada de ônibus, facilitando a sua localização pelos usuários e as coordenadas de latitude e longitude posicionam precisamente cada parada no mapa, permitindo que os usuários vejam a localização exata e planejem suas rotas de deslocamento de maneira eficiente.

5.3 ARQUITETURA DA APLICAÇÃO

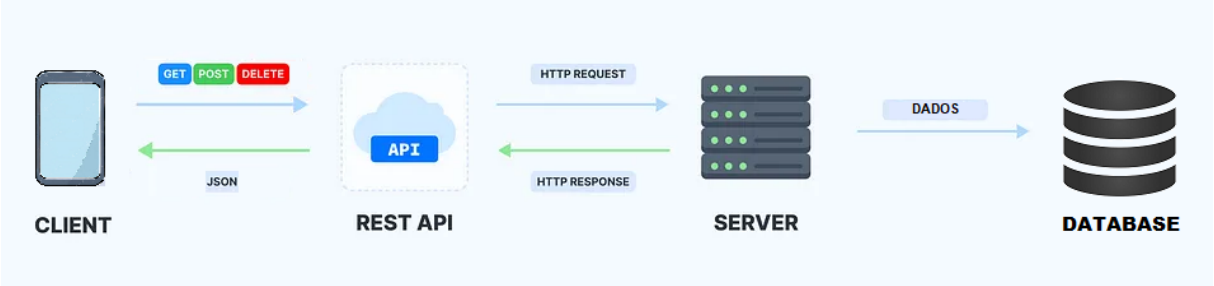
A arquitetura da aplicação é dividida em duas camadas principais: o backend, desenvolvido com Java Spring Boot, e o frontend, desenvolvido com Flutter. Essa separação clara entre as camadas permite uma modularidade e manutenção de maneira eficiente, além de facilitar a escalabilidade do sistema.

A comunicação entre ambas as camadas é realizada por meio de API RESTful conforme é possível visualizar na Figura 5.5. Esta abordagem permite a troca de dados e comandos entre o servidor e o cliente, utilizando protocolos HTTP para enviar requisições e receber respostas em formato JSON. Os principais aspectos desta comunicação incluem:

- Requisições HTTP: Permitem que o frontend interaja com o backend para realizar operações de leitura, criação e exclusão de dados.
- Autenticação: garante a segurança do sistema, permitindo que apenas usuários autenticados possam acessar e modificar qualquer tipo de dado.

– Respostas JSON: O backend retorna os resultados das operações em formato JSON, que são facilmente processados pelo frontend para atualizar a interface do usuário e vice-versa.

Figura 5.5 - Demonstração da arquitetura



Fonte: Autor

5.3.1 Backend

A camada backend da aplicação é a base central que suporta a lógica de negócios, gerencia os dados e realiza a comunicação com o frontend. A estrutura do backend é composta por vários componentes essenciais que trabalham em conjunto para garantir um funcionamento eficiente e seguro da aplicação.

Após criado o projeto Spring Boot, são configuradas algumas dependências para que o projeto possa funcionar, isso é realizado no arquivo “pom.xml”, para que a Maven, que é uma gerenciadora de pacotes disponíveis no Spring Boot, possa gerenciar e verificar se está tudo certo com as dependências do projeto. Na Figura 5.6 é demonstrado as dependências necessárias para a execução desse projeto.

Figura 5.6 - Dependências do backend

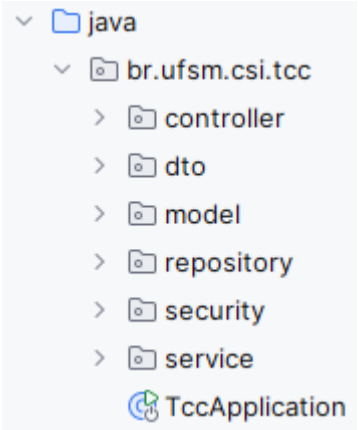
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-neo4j</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>com.auth0</groupId>
    <artifactId>java-jwt</artifactId>
    <version>4.4.0</version>
</dependency>
```

Fonte: Autor

- Primeira dependência: fornece um conjunto de bibliotecas e configurações padrão para integração com o Neo4j, assim facilitando operações de persistência de dados no Neo4j.
- Segunda dependência: fornece suporte para autenticação e autorização de segurança ao servidor ao realizar requisições.
- Terceira dependência: oferece funcionalidades relacionadas a desenvolvimento web, como o controle da MVC e tratamento de requisições HTTP.
- Quarta dependência: é uma biblioteca Java que ajuda a reduzir a verbosidade do código fonte, automatizando a geração de alguns métodos comuns e assim agilizando todo o processo de desenvolvimento.
- Última dependência: é uma biblioteca para criar e utilizada para validar JSON Web Tokens (JWTs).

Na Figura 5.7 é possível visualizar a estrutura de diretórios utilizada no desenvolvimento do sistema, a seguir é explicado brevemente cada uma delas.

Figura 5.7 - Estrutura de diretórios do backend



Fonte: Autor

- Controller: é responsável por receber requisições do cliente, interagir com a lógica de negócios e retornar respostas apropriadas.
- Dto: contém classes que são utilizadas para transferir dados entre diferentes camadas da aplicação.
- Model: representa a estrutura dos dados da aplicação, classes que representam as entidades de dados.
- Repository: este é responsável pela interação com o banco de dados. Ele fornece métodos para realizar operações de leitura e escrita no banco de dados, abstraindo e simplificando a complexidade das consultas.
- Security: detém as configurações relacionadas à segurança da aplicação. Isso inclui a configuração de autenticação, filtros de segurança e implementação da estratégia de segurança JWT.
- Services: é onde contêm a lógica de negócios da aplicação. Atuam como uma camada que intermédia os “Controllers” (que lidam com requisições) e os “Repositorys” (que lidam com o acesso ao banco de dados).

Dando prosseguimento, agora serão abordadas algumas classes chave do backend. Começando pelo diretório “model” existe a classe “Local”, que está diretamente ligada ao nó “Local” do banco de dados Neo4j, ou seja, serve para armazenar os dados vindos do Neo4j, como é demonstrado na Figura 5.8.

Figura 5.8 - Código da classe Pessoa no model

```java
@Node("Pessoa")
public class Pessoa {

    @Id
    @GeneratedValue
    Long id;
    String nome;
    String email;
    String senha;

    @Relationship(type = "INTERESSE", direction = Relationship.Direction.OUTGOING)
    private Set<Area> areas = new HashSet<>();

    @Relationship(type = "FAVORITOS", direction = Relationship.Direction.OUTGOING)
    private Set<Local> locaisFavoritos = new HashSet<>();
}
```

Fonte: Autor

Nesse código da Figura 5.8, a anotação “Node” representa um nó no banco de dados Neo4j com o rótulo “Pessoa”, em seguida é determinado os seus atributos e na anotação “Relationship” é utilizada para realizar a representação do relacionamento presente no banco de dados Neo4j diretamente no código, onde nesse caso o relacionamento parte do nó Pessoa para os nós Area ou Local.

A seguir, na Figura 5.9 é apresentado o código de um repositório que faz o acesso direto e consulta ao bando de dados. Este que estende “Neo4jRepository”, esta interface contém métodos personalizados para consultas específicas usando a linguagem Cypher, que é a linguagem de consulta do Neo4j. Além dos métodos já disponibilizados pela interface para realizar a criação, leitura e exclusão de dados, foi criado duas consultas adicionais para puxar os dados do banco.

A primeira consulta busca todos os nós “Local” que oferecem uma determinada área do conhecimento com base no nome da área fornecido pelo o usuário, já a segunda calcula a distância entre a localização geográfica do usuário e os locais que oferecem a área desejada, seleciona o local mais próximo, e depois calcula a distância entre esse local e todos pontos de ônibus, retornando a ponto de ônibus mais próximo junto com o prédio e a terceira consulta retorna recomendações de locais com base nas áreas de interesse da pessoa.

Figura 5.9 - Código da classe LocalRepository no repository

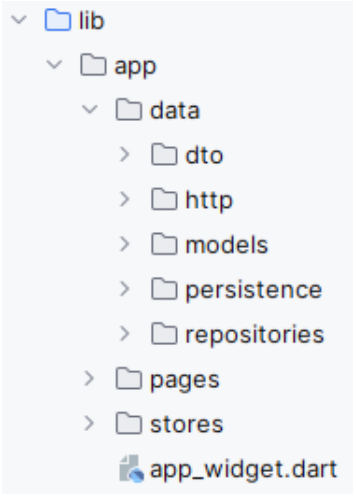
```
@Repository  5 usages  Kelvin Floriano *
public interface LocalRepository extends Neo4jRepository<Local, Long> {

    @Query("""  1 usage  Kelvin Floriano
            MATCH (l:Local)-[:OFERECE]->(a:Area {nome: $nome})
            RETURN l""")
    List<Local> findLocalsByArea(String nome);


    @Query("""  1 usage  Kelvin Floriano
            WITH point({latitude: $lat, longitude: $longi}) AS userLocation
            MATCH (l:Local)-[:OFERECE]->(a:Area {nome: $nomeArea})
            WITH l, userLocation, point.distance(point({latitude: l.lat, longitude: l.long}), userLocation) AS distance
            ORDER BY distance ASC
            LIMIT 1
            WITH l AS localMaisProximo
            MATCH (p:Ponto)
            WITH localMaisProximo, p, point.distance(point({latitude: p.lat, longitude: p.long}),
            point({latitude: localMaisProximo.lat, longitude: localMaisProximo.long})) AS distanciaParaLocal
            ORDER BY distanciaParaLocal ASC
            LIMIT 1
            RETURN localMaisProximo, p AS pontoMaisProximo
            """)
    Resultado findNearestLocalAndPonto(double lat, double longi, String nomeArea);

    @Query("MATCH (p:Pessoa)-[:INTERESSE]->(a:Area)<-[:OFERECE]-(l:Local) WHERE id(p) = $pessoaId RETURN l ORDER BY rand() LIMIT 5")
    List<Local> recomendacoesLocais(Long pessoaId);
}
```

Fonte: Autor

5.3.2 Frontend

O frontend foi desenvolvido utilizando o framework Flutter, uma tecnologia open-source criada pelo Google, que permite a criação de interfaces de usuário para múltiplas plataformas, incluindo Android, onde será feito os testes, a partir de uma única base de código. Na Figura 5.10 é possível visualizar a estrutura de diretórios utilizada para desenvolver a visualização do projeto e logo após uma explicação sobre cada um.

Figura 5.10 - Estrutura de diretórios do frontend



Fonte: Autor

- Data: esse diretório possui as camadas responsáveis pela gestão de dados e comunicação com o backend.
- Dto: contém classes que representam os objetos utilizados para encapsular e transportar dados entre o frontend e o backend.
- Http: abriga classes e serviços relacionados à comunicação com o backend via requisições HTTP. Isso inclui a configuração das chamadas de API e o tratamento das respostas.
- Models: são as classes que representam os modelos de dados da aplicação.
- Persistence: responsável por gerenciar o armazenamento local e a persistência de dados no banco de dados local.
- Repositories: encapsulam a lógica para acessar e manipular os dados, abstraindo a complexidade da interação com diferentes fontes de dados.
- Pages: são as classes que representam as diferentes telas ou páginas da aplicação. Cada página é responsável por renderizar uma parte específica da interface do usuário, como a tela de início por exemplo.
- Stores: esse diretório é utilizado para gerenciar o estado da aplicação. Em Flutter, os "stores" são responsáveis por armazenar e gerenciar o estado global ou local da aplicação, garantindo que a interface do usuário esteja sincronizada com os dados atuais.

A seguir, é abordado as dependências utilizadas no frontend, explicando como cada uma contribui para o funcionamento e desempenho da aplicação e a visualização da estrutura de diretórios na Figura 5.11.

Figura 5.11 - Dependências necessárias do frontend

```
dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.6
  http: ^1.2.1
  google_maps_flutter: ^2.6.0
  geolocator: ^11.0.0
  flutter_secure_storage: ^9.2.2
  sqflite: ^2.0.0+3
  path: ^1.8.0
```

Fonte: Autor

- Cupertino_icons: fornece ícones estilizados no estilo iOS para aplicações Flutter.
- Http: biblioteca para realizar requisições HTTP. É utilizada para integrar a aplicação frontend com APIs RESTful, manipulando respostas e enviando dados para o backend.
- Google_maps_flutter: permite a integração com o Google Maps em aplicações Flutter, com esta biblioteca, é possível exibir mapas interativos, adicionar marcadores, entre outras funcionalidades do Google Maps.
- Geolocator: oferece funcionalidades para acessar a localização geográfica do dispositivo. Inclui recursos para obter a localização atual do usuário, calcular distâncias entre pontos e monitorar mudanças na localização, sendo crucial para esse projeto.
- Flutter_secure_storage: biblioteca para armazenamento seguro de dados sensíveis, utiliza criptografia para armazenar informações de maneira segura no dispositivo.
- Sqflite: biblioteca necessária para gerenciar bancos de dados SQLite locais em aplicações Flutter.
- Path: facilita o trabalho com caminhos de arquivos no sistema de arquivos. Oferece funções para manipulação e construção de caminhos de arquivos de forma segura e portátil.

5.4 EXEMPLO DE USO DA APLICAÇÃO

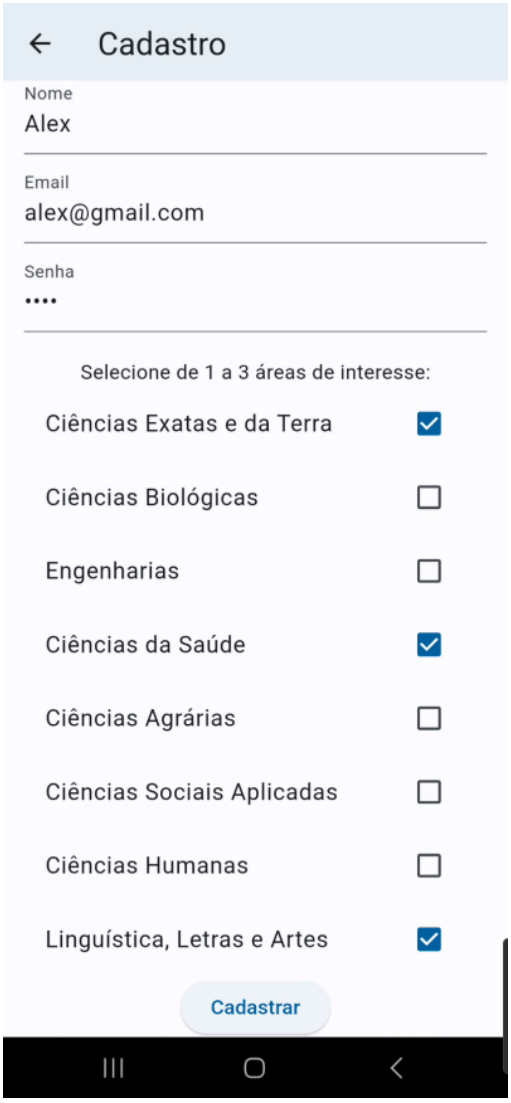
Para proporcionar uma visão abrangente da interface do usuário e da experiência de navegação, será explorado cada tela do aplicativo em detalhe. A seguir, é apresentado desde o momento em que o usuário abre o aplicativo pela primeira vez indo da página de cadastro, página de autenticação até a página inicial onde são demonstradas as funcionalidades principais.

Figura 5.12 - Página de autenticação

Fonte: Autor

Quando o usuário abre o aplicativo pela primeira vez, é apresentado a ele então a tela de autenticação, conforme é possível visualizar na Figura 5.12, onde é necessário fornecer um email e senha para poder ir para a página inicial, mas antes disso então é necessário realizar o cadastro desse usuário, clicando no botão abaixo de “Login” é feito o redirecionamento para a página de cadastro demonstrado na Figura 5.13.

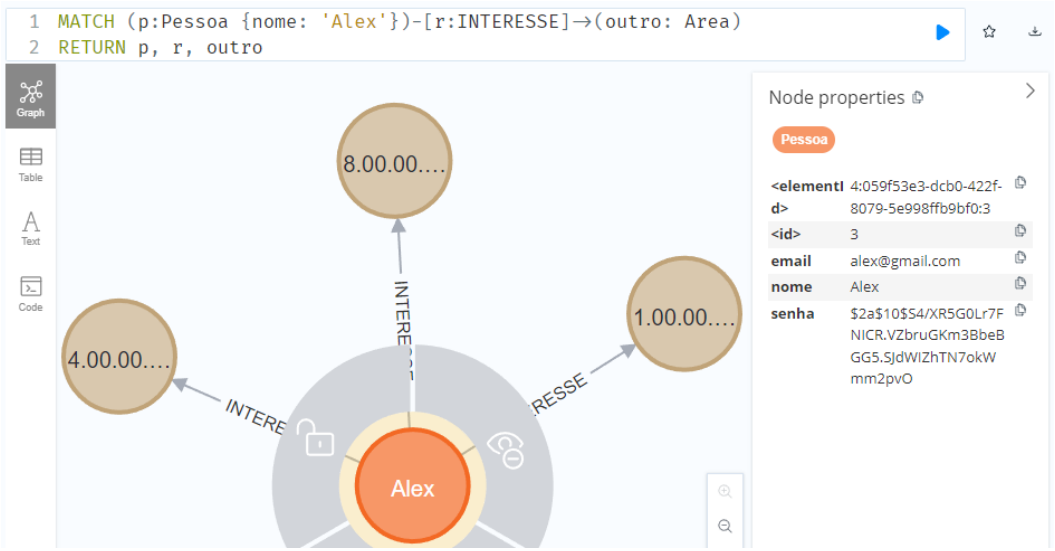
Figura 5.13 - Página de cadastro



Fonte: Autor

Para realizar o cadastro do usuário é necessário fornecer um nome, email, senha e escolher de uma a três áreas do conhecimento que ele possua interesse, após tudo preenchido e selecionado, o usuário clica no botão Cadastrar e é redirecionado para a página de autenticação conforme mostra a Figura 5.12, e com o email e senha cadastrados é realizada a autenticação. Na Figura 5.14 é possível visualizar a criação desse usuário dentro do banco de dados Neo4j, com todos seus campos cadastrados, além dos relacionamentos também criados com áreas de interesses indicadas.

Figura 5.14 - Usuário cadastrado no banco



Fonte: Autor

Quando o usuário realiza a autenticação é apresentada a página inicial conforme mostra a Figura 5.15, sempre que o usuário se autenticar, serão mostradas a ele cinco recomendações de locais dentro do campus baseado nas suas áreas de interesse (usa consulta apresentada na Figura 5.9, mostrando os locais em ordem de proximidade).

Figura 5.15 - Página inicial com recomendações

Fonte: Autor

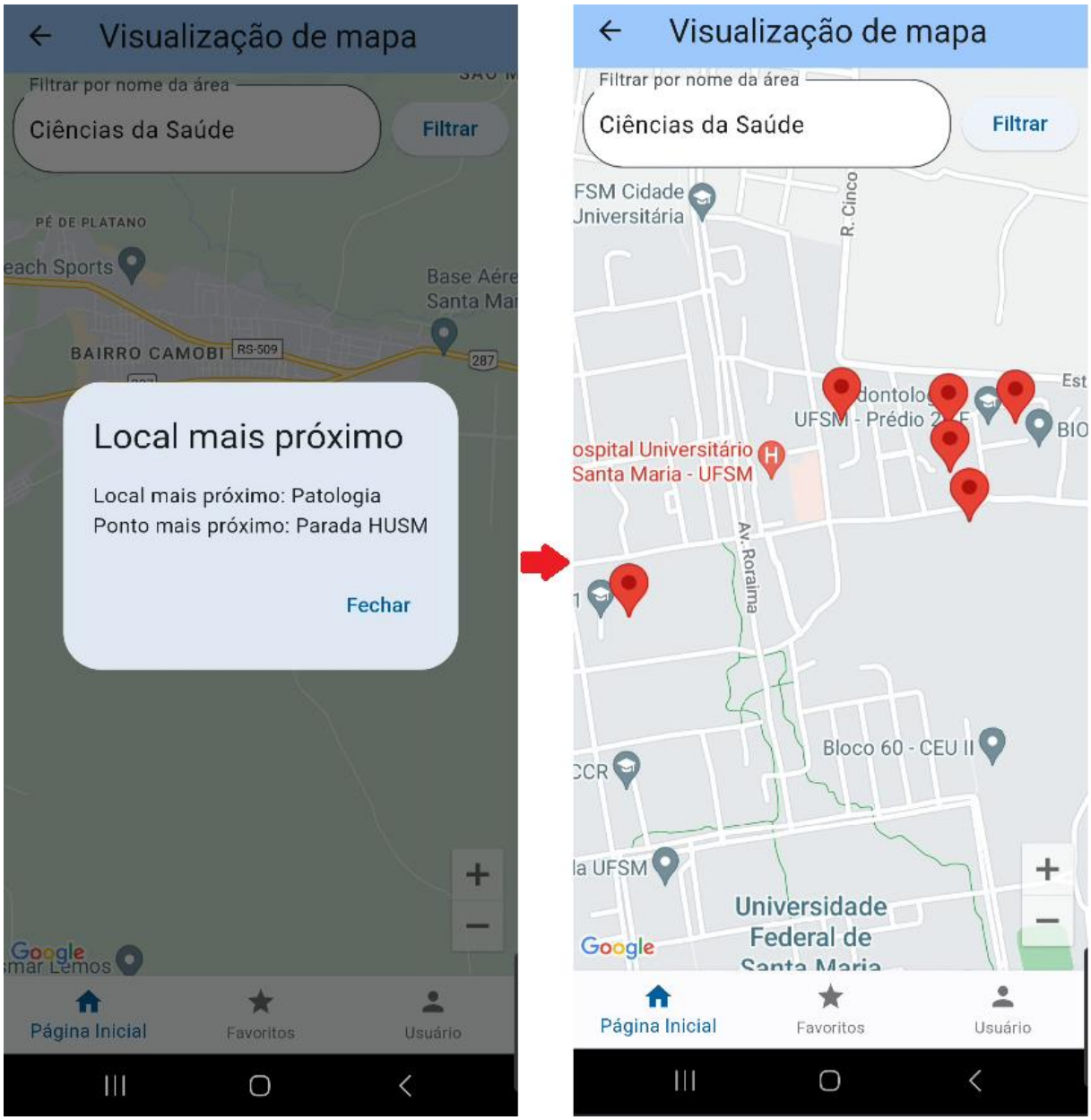
Ao clicar em um desses locais, o diálogo é fechado e então é apresentado no mapa o local que o usuário clicou com algumas informações em cima conforme mostra a Figura 5.16.

Figura 5.16 - Página inicial com marcador específico

Fonte: Autor

Ao clicar em fechar o diálogo ou clicar em algum local recomendado específico, é possível visualizar a página inicial, onde há vários marcadores indicando que são locais da UFSM, em cima há uma barra para poder filtrar por algum nome de área específico e então é retornado o local mais próximo da localização atual do usuário junto com o ponto de ônibus mais próximo para esse local e logo após mostrado no mapa apenas os locais que oferecem a área filtrada especificamente, conforme mostra na Figura 5.17 onde é filtrado pela a área “Ciências da Saúde”. Para realizar esse filtro é então utilizada a segunda consulta do Neo4j conforme mostra na Figura 5.9, onde é feito o cálculo da distância do usuário com os locais.

Figura 5.17 - Página inicial ao realizar filtro



Fonte: Autor

Ainda na página inicial, há os botões na parte inferior para realizar a navegação no aplicativo caso o usuário queira ver os seus locais favoritos, voltar para a página inicial ou para sair da conta conectada no momento. E ao clicar em algum marcador desses locais é apresentada uma caixinha com algumas informações sobre o local e se o usuário deseja adicionar aos locais favoritos, onde é possível visualizar na Figura 5.18, nesse caso o local será adicionado aos favoritos como teste para apresentar a página de favoritos.

Figura 5.18 - Informações adicionais de um local

Fonte: Autor

E por fim a página de favoritos, conforme mostra a Figura 5.19, onde é mostrado os locais favoritos que foram selecionados pelo o usuário com o decorrer do uso do aplicativo.

Figura 5.19 - Página dos locais favoritos

Fonte: Autor

6 CONSIDERAÇÕES FINAIS

Diante disso, o objetivo central deste trabalho foi explorar e demonstrar as capacidades dos bancos de dados orientados a grafos, com ênfase na plataforma Neo4j, e aplicar esses conceitos em um sistema prático. O Neo4j se destaca entre os bancos de dados orientados a grafos, sendo o mais utilizado[11]. Com base nos resultados encontrados no desenvolvimento desse trabalho, pode-se indicar que o objetivo proposto foi alcançado.

No trabalho, além de ter sido detalhado as operações nos grafos e suas aplicações, foi aprofundado a compreensão sobre bancos de dados orientados a grafos com um enfoque no Neo4j. Além disso, foi desenvolvido uma aplicação que realiza a integração do Neo4j com outras tecnologias, permitindo ao usuário visualizar e filtrar prédios em um campus universitário e identificar o prédio mais próximo com base na localização do usuário, demonstrando a aplicabilidade do Neo4j em um cenário real.

[11] https://db-engines.com/en/ranking/graph+dbms

REFERÊNCIAS

ALVAREZ, Guilherme M.; CECI, Flávio; GONÇALVES, Alexandre L. **Análise Comparativa dos Bancos Orientados a Grafos de Primeira e Segunda Geração Uma Aplicação na Análise Social**. CEP, v. 88040, p. 900, 2015.

IBM. **O que é Java Spring Boot?** Disponível em: https://www.ibm.com/br-pt/topics/java-spring-boot. Acesso em: 13 nov. 2023.

LU, Huiling; HONG, Zhiguo; SHI, Minyong. **Analysis of film data based on Neo4j**. In: 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS). IEEE, 2017. p. 675-677.

NAQVI, Syeda N Z.; YFANTIDOU, Sofia; ZIMÁNYI, Esteban. **Time series databases and influxdb**. Studienarbeit, Université Libre de Bruxelles, v. 12, 2017.

NEO4J. **Cypher Query Language**. Disponível em: https://neo4j.com/developer/cypher/. Acesso em: 11 jul. 2023.

ORACLE. **O que é um banco de dados relacional (RDBMS)**. Disponível em: https://www.oracle.com/br/database/what-is-a-relational-database/. Acesso em: 01 jun. 2023.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL essencial: um guia conciso para o mundo emergente da persistência poliglota**. São Paulo: Novatec Editora, 2019.

SANTOS, Erik U.; SILVA, Marcos A L. **Abordagem ao Banco de Dados Orientado a Grafos Neo4j em um Nível Empresarial**. 2013.

SILBERSCHATZ, Abraham. **Sistema de Banco de Dados**. 7. ed. Rio de Janeiro: Grupo GEN; 2020.

SILVA, Luiz F C.; RIVA, Aline D.; ROSA, Gabriel A.; et al. **Banco de Dados Não Relacional**. Porto Alegre: Grupo A, 2021.

TENENBAUM, Aaron M.; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J. **Estruturas de dados usando C**. São Paulo: Makron Books, 1995.

VELOSO, Paulo; DOS SANTOS, Clesio; AZEVEDO, Paulo; FURTADO, Antonio. **Estrutura de dados**. Rio de Janeiro: Campus, 1983.