

**ANDRESSA FARIA
EDILSON JUSTINIANO**

**BANCO DE DADOS ORIENTADO A GRAFOS APLICADO
À BUSCA POR MÃO DE OBRA**

**UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE – MG
2015**

**ANDRESSA FARIA
EDILSON JUSTINIANO**

**BANCO DE DADOS ORIENTADO A GRAFOS APLICADO
À BUSCA POR MÃO DE OBRA**

Trabalho de Conclusão de Curso apresentado
como requisito parcial à obtenção do título de
Bacharel em Sistemas de Informação na Uni-
versidade do Vale do Sapucaí – UNIVAS.

Orientador: Prof. MSc. Márcio Emílio Cruz
Vono de Azevedo

**UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE – MG
2015**

Faria, Andressa; Justiniano, Edilson

BANCO DE DADOS ORIENTADO A GRAFOS APLICADO À
BUSCA POR MÃO DE OBRA / Andressa Faria, Edilson Justiniano –
Pouso Alegre – MG: Univás, 2015.

66 f. : il.

Trabalho de Conclusão de Curso (graduação) – Universidade do Vale
do Sapucaí, Univás, Sistemas de Informação.

Orientador: Prof. MSc. Márcio Emílio Cruz Vono de Azevedo

1. mão de obra. 2. grafo. 3. banco de dados.

**ANDRESSA FARIA
EDILSON JUSTINIANO**

**BANCO DE DADOS ORIENTADO A GRAFOS APLICADO
À BUSCA POR MÃO DE OBRA**

Trabalho de conclusão de curso defendido e aprovado em 01/01/2016 pela banca examinadora constituída pelos professores:

Prof. MSc. Márcio Emílio Cruz Vono de Azevedo
Orientador

Prof^a. MSc. Nome da professora
Avaliadora

Prof. MSc. Nome do professor
Avaliador

De Andressa Faria.
Dedico este trabalho ...

De Edilson Justiniano.
Dedico este trabalho ...

AGRADECIMENTOS

De Andressa Faria

Agradeço ...

De Edilson Justiniano

Agradeço ...

*“Complicar é simples,
simplificar que é complicado.
(Paulo Sérgio dos Santos)*

FARIA, Andressa; JUSTINIANO, Edilson. **BANCO DE DADOS ORIENTADO A GRAFOS APLICADO À BUSCA POR MÃO DE OBRA**. 2015. Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, Pouso Alegre – MG, 2015.

RESUMO

Este trabalho apresenta . . .

Palavras-chave: mão de obra. grafo. banco de dados.

FARIA, Andressa; JUSTINIANO, Edilson. **BANCO DE DADOS ORIENTADO A GRAFOS APLICADO À BUSCA POR MÃO DE OBRA**. 2015. Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, Pouso Alegre – MG, 2015.

ABSTRACT

This work presents ...

Key words: manpower. graph. database.

LISTA DE FIGURAS

Figura 1 – Uma visão geral do ICONIX e seus componentes.	16
Figura 2 – O problema das 7 pontes de <i>Königsberg</i>	18
Figura 3 – Ilustração de uma representação gráfica de um simples grafo	19
Figura 4 – Ilustração de uma representação gráfica de um <i>multigraph</i>	20
Figura 5 – Imagem de uma representação gráfica de um grafo direcionado	20
Figura 6 – Imagem de um grafo isomórfico	21
Figura 7 – Grafo representado por meio de uma matriz adjacência	21
Figura 8 – Exemplo simples de um grafo no Neo4j	26
Figura 9 – Uma visão geral do processo de desenvolvimento de software.	29
Figura 10 – Exemplo de inclusão do estilo CSS inline	33
Figura 11 – Exemplo de inclusão do estilo CSS incorporado à página HTML	34
Figura 12 – Exemplo de inclusão do estilo CSS a partir de um arquivo externo	34
Figura 13 – Exemplo de inclusão do código em Javascript incorporado ao HTML	35
Figura 14 – Exemplo de inclusão do código Javascript de um arquivo externo	35
Figura 15 – Modelo de domínio inicial	42
Figura 16 – Diagrama de caso de uso	42
Figura 17 – Diagrama de robustez do caso de uso Localizar mão de obra	44
Figura 18 – Modelo de domínio atualizado	44
Figura 19 – Modelo de dados da aplicação	45
Figura 20 – Diagrama de caso de uso	46
Figura 21 – Diagrama de classes	47
Figura 22 – Tela de serviços do GitHub	48
Figura 23 – Ferramentas da IDE Eclipse	49
Figura 24 – Código de comunicação com o banco	50
Figura 25 – Exemplo de consulta usando a API <i>cypher</i>	52
Figura 26 – Tela de login	53
Figura 27 – Página inicial do usuário contratante	53
Figura 28 – Página de localização de parceiros	54

LISTA DE QUADROS

Quadro 1 – Fluxo de eventos para o caso de uso criar conta	43
--	----

LISTA DE SIGLAS E ABREVIATURAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	<i>Application Programming Interface</i>
HTTP	<i>Hypertext Transfer Protocol</i>
JCP	<i>Java Community Process</i>
JSF	<i>JavaServer Faces</i>
JSP	<i>JavaServer Pages</i>
JVM	<i>Java Virtual Machine</i>
MVC	<i>Model – View – Controller</i>
NoSQL	<i>Not Only SQL</i>
SQL	<i>Structured Query Language</i>
UML	<i>Unified Modeling Language</i>

SUMÁRIO

Introdução	13
2 QUADRO TEÓRICO	15
2.1 Iconix	15
2.1.1 Análise de requisitos	16
2.1.2 Análise e projeto preliminar	16
2.1.3 Projeto detalhado	17
2.1.4 Implementação	17
2.2 Teoria dos Grafos	18
2.3 Tecnologias	22
2.3.1 Banco de dados	22
2.3.2 Banco de dados relacionais	23
2.3.3 Banco de dados NoSQL	23
2.3.4 Neo4j	24
2.3.5 <i>Cypher Query Language</i>	26
2.3.6 Java	27
2.3.7 Tomcat 7	29
2.3.8 <i>Web Service REST</i>	30
2.3.9 HTML 5	32
2.3.10 CSS 3	32
2.3.11 Javascript	34
2.3.12 Angular JS	36
3 QUADRO METODOLÓGICO	37
3.1 Tipo de pesquisa	37
3.2 Contexto de pesquisa	38
3.3 Instrumentos	39
3.4 Procedimentos e resultados	39
3.4.1 ICONIX	40
3.4.2 Preparação do ambiente	47
3.4.3 Desenvolvimento	50
4 DISCUSSÃO DOS RESULTADOS.....	57
4.1 Neo4j	57
4.2 Aplicação web	58
5 CONCLUSÃO	59
REFERÊNCIAS.....	62
APÊNDICES	64
ANEXOS	65

INTRODUÇÃO

Com a constante evolução tecnológica é possível notar que, a cada dia, mais pessoas estão sendo inseridas em um mundo globalizado. Pertencer a este meio tem mudado completamente a maneira de se realizar tarefas, uma vez que, a revolução tecnológica busca facilitar o que até então era trabalhoso. Atualmente, nos deparamos com situações nas quais as pessoas desempenham mais de um papel, dividindo o seu tempo entre tarefas profissionais, pessoais e aquelas que chamamos de rotineiras as quais muitas vezes, não são realizadas por elas e sim por terceiros. Um profissional terceirizado é capaz de cuidar de todas as tarefas extras que não cabem na rotina, no entanto, encontrá-los tem se tornado cada vez mais difícil, uma vez que confiar a sua residência ou, ainda, a sua intimidade a uma pessoa não conhecida gera muita insegurança. Ainda, vale ressaltar que este tipo de trabalho está cada vez mais escasso e raro no cenário atual.

Com o advento da internet, e mais tarde sua popularização, uma série de aplicações para diferentes fins vem sendo desenvolvidas. Estes aplicativos tem como finalidade apresentar soluções para os mais diversos tipos de problemas. É possível notar que as redes sociais geram um grande impacto no modelo de vida que seguimos. Hoje, sabemos que tudo e todos estão conectados, direta ou indiretamente, e que as informações são trocadas a uma velocidade surpreendente. Precisamos obter informações de forma clara e rápida (BARBOSA, 2011).

Com intuito de obter acesso à informação de maneira instantânea, a migração dos antigos computadores pessoais (*desktops*) para dispositivos cada vez mais portáteis tornou-se comum. Podemos citar, como exemplo, a criação dos *tablets* que proporcionaram uma grande revolução tecnológica, pois, a partir desta invenção as pessoas passaram a ter liberdade de acesso, uma vez que as informações se encontram na palma de suas mãos. Seguindo esta tendência de mobilidade, os *smartphones* foram desenvolvidos.

O sucesso obtido por estes dispositivos móveis, somado a evolução das redes de dados 3G e 4G, permitiu a conexão entre as pessoas de qualquer lugar e a qualquer momento bastando alguns toques, tornando assim ainda mais fácil localizá-las e comunicar-se com elas.

A pesquisa realizada pelo Instituto Nielsen reafirma o contexto anteriormente mencionado. Ela comprova que o tempo gasto pelos usuários em seus *smartphones* é maior do que em seus computadores pessoais, sendo que boa parte deste tempo é utilizado para acesso às redes sociais. Isto ocorre pois os dispositivos móveis atuais são capazes de substituir os computadores pessoais em uma série de atividades, tornando-os um computador de mão disponível a qualquer

instante. A pesquisa, ainda demonstra o sucesso das redes sociais que, segundo a mesma, se deve a disseminação dos dispositivos móveis e ao modelo de vida que grande parte das pessoas segue atualmente.

Com a ampliação desta forma de comunicação, somada a evolução das tecnologias já percorridas, houve a necessidade de se realizar algumas melhorias voltadas aos bancos de dados, a fim de otimizar as tarefas realizadas por ele, levando assim, a criação de uma nova geração de bancos de dados, denominada NoSQL¹.

Dentre esta nova geração, o banco de dados orientado a grafos vem se destacando por apresentar grande potencial para manipulação de dados em diferentes áreas como: relacionamento interpessoal, biológicas, rotas geográficas, entre outras.

Penha e Carvalho (2013) utilizaram esta tecnologia para desenvolver uma aplicação, cujo principal objetivo é recomendar filmes aos seus usuários.

Silva e Pires (2013) realizaram uma comparação entre banco de dados orientado a grafos e os relacionais a fim de apresentar as vantagens que estes possuem sob os bancos de dados relacionais.

Junid et al. (2012) utilizaram a teoria dos grafos para tentar otimizar o processo de alinhamento da sequência de DNA a fim de determinar a região comum entre duas ou mais sequências deste.

Com o intuito de tentar solucionar o problema relacionado a busca por mãos de obra temporárias, este trabalho tem como proposta apresentar uma possível solução, através de uma aplicação *web*, seguindo o modelo de negócio das redes sociais, utilizando o conceito de orientação a grafos, a fim de gerar a familiarização desta ferramenta, uma vez que este tipo de serviço já está intrínseco na atualidade.

¹ NOSQL: *Not Only SQL* - Bancos de dados que utilizam não somente os recursos de Structure Query Language - SQL, a fim de obter melhor performance.

2 QUADRO TEÓRICO

Neste capítulo são discutidas as técnicas, metodologias e tecnologias que foram utilizadas no desenvolvimento deste trabalho.

2.1 Iconix

O ICONIX, segundo Rosenberg, Stephens e Collins-Cope (2005), foi criado em 1993 a partir de um resumo das melhores técnicas de desenvolvimento de *software* utilizando como ferramenta de apoio a *Unified Modeling Language* - UML³. Esta metodologia é mantida pela empresa *ICONIX Software Engineering* e seu principal idealizador é Doug Rosenberg.

Para Rosenberg e Scott (1999), o ICONIX possui como característica ser iterativo e incremental, somado ao fato de ser adequado ao padrão UML auxiliando, assim, o desenvolvimento e a documentação do sistema.

Atualmente, existem diversas metodologias de desenvolvimento de *software* disponíveis, contudo, o ICONIX, em especial, será utilizado para auxiliar no processo de desenvolvimento deste trabalho pois, segundo Silva e Videira (2001), essa metodologia nos permite gerar a documentação necessária para nortear o desenvolvimento de um projeto acadêmico.

De acordo com Rosenberg e Stephens (2007), os processos do ICONIX consistem em gerar alguns artefatos que correspondem aos modelos dinâmico e estático de um sistema e estes são elaborados e desenvolvidos de forma incremental e em paralelo, possibilitando ao analista dar maior ênfase no desenvolvimento do sistema do que na documentação do mesmo. A Figura 1, apresenta uma visão geral dos componentes do ICONIX.

³ UML: *Unified Modeling Language* - Linguagem de modelagem para objetos do mundo real que habilita os desenvolvedores especificar, visualizar, construí-los a nível de software.

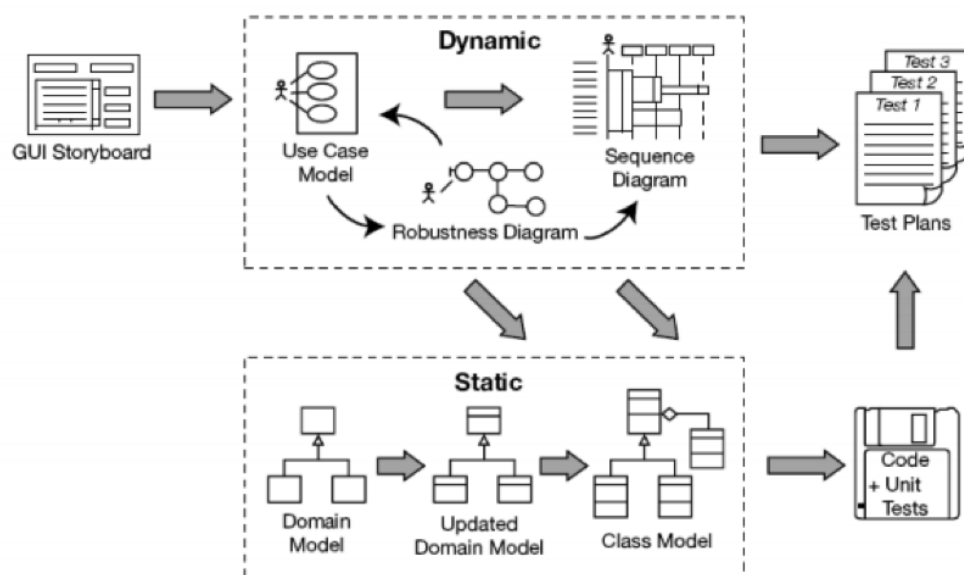


Figura 1 – Uma visão geral do ICONIX e seus componentes. **Fonte:** Rosenberg e Scott (1999)

Ao utilizar essa metodologia, o desenvolvimento do projeto passa a ser norteado por casos de uso (*use cases*) e suas principais fases são: análise de requisitos, análise e projeto preliminar, projeto e implementação. Abaixo será apresentada uma breve descrição de cada uma das fases do ICONIX, seguindo as ideias de Azevedo (2010).

2.1.1 Análise de requisitos

A função da fase de análise de requisitos é modelar os objetos do problema real a partir dos requisitos do software já levantados, e, a partir destes objetos gerar o diagrama de modelo de domínio. Ainda nesta fase, e, com base nos requisitos, deve-se definir os casos de uso do *software*. Estes casos de uso são o elo entre os requisitos e a implementação propriamente dita do *software*. Por isto, a definição destes diagramas se fazem tão importante para o ICONIX.

2.1.2 Análise e projeto preliminar

Nesta fase deve-se detalhar todos os casos de uso identificados na fase de análise de requisitos, por meio de diagramas de robustez, baseando-se no texto dos casos de uso (fluxo de eventos). O diagrama de robustez não faz parte dos diagramas padrões da UML. Porém, este é

utilizado para descobrir as classes de análise e detalhar superficialmente o funcionamento dos casos de uso.

Em paralelo, deve-se atualizar o modelo de domínio adicionando os atributos identificados às suas respectivas entidades, que foram descobertas na fase de análise de requisitos. A partir deste momento será possível gerar a base de dados do sistema.

2.1.3 Projeto detalhado

Na fase denominada projeto detalhado deve-se elaborar o diagrama de sequência fundamentando-se nos diagramas de casos de uso identificados anteriormente, a fim de detalhar a implementação do caso de uso.

O diagrama de sequência deve conter as classes que serão implementadas e as mensagens enviadas entre os objetos corresponderão as operações que realmente serão implementadas futuramente. Estas operações devem ser incluídas no modelo de domínio em conjunto com as novas classes do projeto infentificadas, criando assim, o diagrama de classes final.

2.1.4 Implementação

Na fase de implementação deve-se desenvolver o código fonte do *software* e os testes necessários para obter um software com qualidade. O ICONIX não define os passos a serem seguidos nesta fase, ficando a cargo de cada um definir a forma como será implementado o projeto.

Ao término de cada fase um artefato é gerado, sendo respectivamente: revisão dos requisitos, revisão do projeto preliminar, revisão detalhada e entrega.

O ICONIX é considerado um processo prático de desenvolvimento de software, pois a partir das iterações que ocorrem na análise de requisitos e na construção dos modelos de domínios (parte dinâmica), os diagramas de classes (parte estática) são incrementados e, a partir destes, o sistema poderá ser codificado.

Por proporcionar essa praticidade, o ICONIX será empregado para o desenvolvimento deste projeto, pois por meio dele é possível obter produtividade no desenvolvimento do *software* ao mesmo tempo em que alguns artefatos são gerados, unindo o aspecto de abrangência e agilidade.

2.2 Teoria dos Grafos

A teoria dos grafos foi criada pelo matemático suíço Leonhard Euler no século XVIII com o propósito de solucionar um antigo problema, conhecido como as 7 pontes de *Königsberg* (HARJU, 2014).

Königsberg, atualmente conhecida como *Kaliningrad*, era uma antiga cidade medieval cortada pelo rio *Pregel* dividindo-a em 4 partes interligadas por 7 pontes. Ela era localizada na antiga Prússia, hoje, território Russo. O problema mencionado anteriormente consistia basicamente em atravessar toda a cidade, visitando todas as partes e utilizar todas as pontes desde que não repetisse uma das quatro partes ou uma das 7 pontes. A Figura 2 ilustra o problema mencionado.

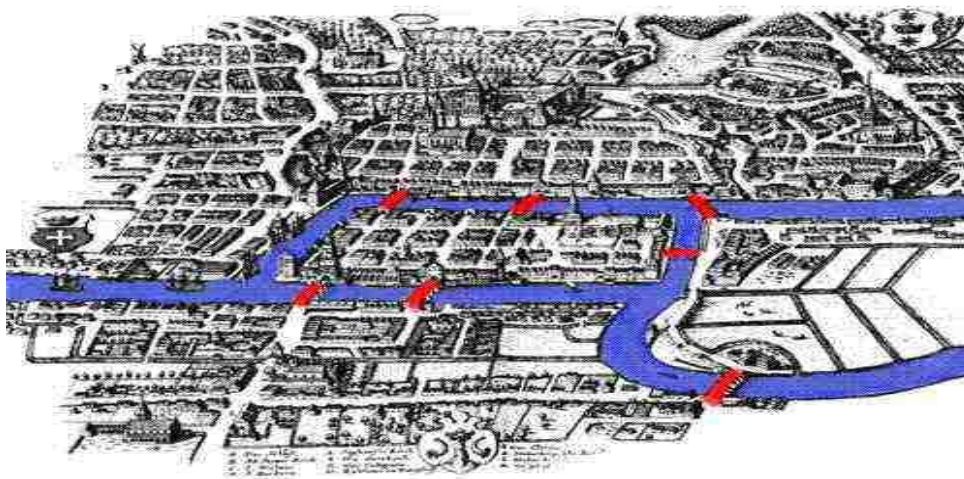


Figura 2 – O problema das 7 pontes de *Königsberg*. **Fonte:** Paoletti (2006)

De acordo com Bruggen (2014), para tentar solucionar o problema, Euler utilizou uma abordagem matemática ao contrário dos demais que tentaram utilizar a força bruta para solucionar tal problema, desenhando N números de diferentes possibilidades de rotas. Euler mudou o foco e passou a dar mais atenção ao número de pontes e não as partes da cidade. Por meio desta observação, foi possível perceber que realizar tal tarefa seria impossível, pois de acordo com sua teoria seria necessário possuir no mínimo mais uma ponte, uma vez que, o número de pontes era ímpar, não sendo possível realizar um caminho único e sem repetição. Desta forma, obteve-se a solução para este problema e criou-se o primeiro grafo no mundo.

Rocha (2013, p. 16) afirma que:

um *grafo* $G = (V, E)$ consiste em um conjunto finito V de vértices e um conjunto finito E de arestas onde cada elemento E possui um par de vértices que estão conectados entre si e pode ou não possuir um peso P .

Esta é a definição formal de um grafo. A partir desta definição, é possível identificar, no problema mencionado anteriormente, os vértices que neste caso são as pontes e as arestas que por sua vez são as partes da cidade.

Segundo Bondy e Murty (1976), muitas situações do mundo real podem ser descritas através de um conjunto de pontos conectados por linhas formando assim um grafo, como um centro de comunicações e seus *links*, ou as pessoas e seus amigos, ou uma troca de emails entre pessoas, entre outras. Isto é possível pois, de acordo com Rocha (2013), existem muitos problemas atualmente que podem ser mapeados para uma estrutura genérica possibilitando assim utilizar a teoria de grafos para tentar solucioná-los, tais como: rotas geográficas, redes sociais, entre outros.

A Figura 3 demonstra de maneira visual um grafo, conforme ideia de Bondy e Murty (1976), utilizando como exemplo o seguinte grafo $G = \{a, b, c, d, e, f, g, h\}$ e suas respectivas arestas $E_g = \{(a, b), (a, h), (a, e), (b, f), (c, e), (c, d), (c, g), (d, e), (d, h), (d, g), (f, h)\}$, sendo que os vértices serão representados por círculos e as arestas que os interligam por linhas.

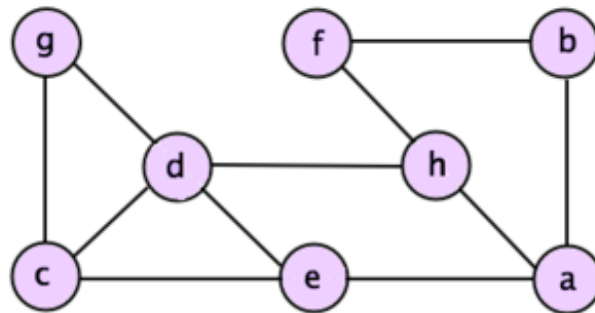


Figura 3 – Ilustração de uma representação gráfica de um simples grafo. **Fonte:** Rocha (2013)

Ruohonen (2013) afirma que os grafos podem ser gerados com a possibilidade de permitir *loops*⁴ e arestas paralelas ou multiplas entre os vértices, obtendo um *multigraph*. A Figura 4 ilustra um simples *multigraph*.

⁴ *loops* - Uma aresta que interliga o mesmo vértice.

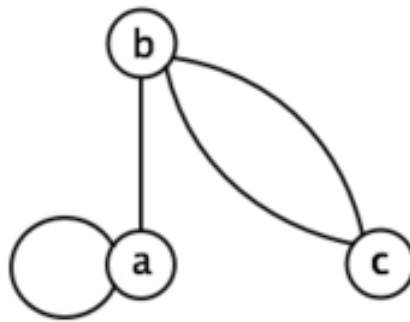


Figura 4 – Ilustração de uma representação gráfica de um *multigraph*. **Fonte:** Adaptado de Harju (2014)

Para Harju (2014), os grafos podem ser direcionados (*dígrafo*) ou não direcionados. Os direcionados são aqueles cujos vértices ligados a uma aresta são ordenados e permitem que uma aresta que conecta os vértices x e y seja representada apenas de uma forma, sendo ela $\{x, y\}$ ou $\{y, x\}$, ao contrário dos não direcionados que, para este mesmo caso, podem ser representado por ambas as formas (ROCHA, 2013). A Figura 5 demonstra um grafo direcionado.

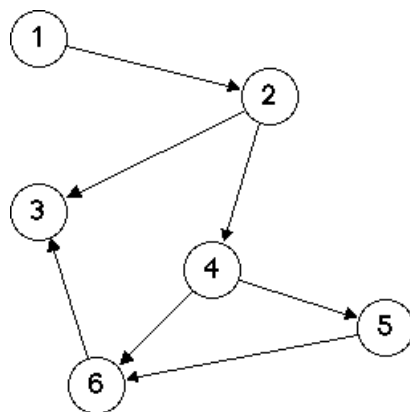


Figura 5 – Imagem de uma representação gráfica de um grafo direcionado. **Fonte:** Keller (2015)

Segundo Harju (2014), os tipos de grafos são:

- **grafo simples:** são aqueles grafos que não possuem *loops* ou arestas paralelas;
- **grafo completo:** são aqueles em que, qualquer par de vértices são adjacentes;
- **subgrafos:** são pequenos grafos que em conjunto constituem um grafo maior.
- **grafos isomórficos:** dois grafos são isomórficos se, ambos possuírem a mesma estrutura de nós, e relacionamentos, exceto pelos identificadores de cada nó que podem ser diferentes, veja na Figura 6 um exemplo de dois grafos isomórficos (HARJU, 2014).

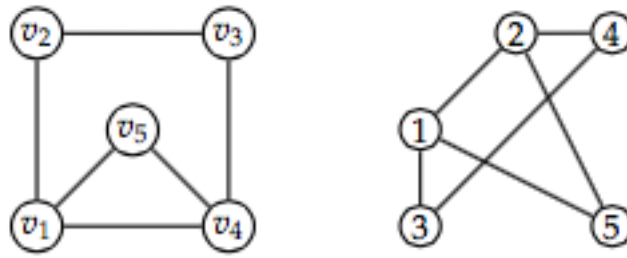


Figura 6 – Imagem de um grafo isomórfico. **Fonte:** Harju (2014)

- **caminho (travessia):** é uma sequência de vértices $\{v_1, v_2, \dots, v_n\}$ conectados por meio de arestas. Exemplo: $\{e_1 = \{v_1, v_2\}, \{v_1, v_3\}, \dots, \{v_n, v_m\}\}$;
- **grafo conexo:** são aqueles que, para qualquer par de vértices, há um caminho que os ligam;
- **grau do vértice:** é definido pela quantidade de arestas que se conectam ao vértice.

Para Rocha (2013) existem várias formas de se representar um grafo computacionalmente utilizando diferentes estruturas de dados. Entretanto, a mais utilizada e simples é a matriz adjacência.

Uma matriz adjacência consiste em uma matriz contendo o mesmo número de linhas e colunas ($n \times n$). Veja na Figura 7 um exemplo de representação de um grafo utilizando esta estrutura.

$$A_G = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Figura 7 – Grafo representado por meio de uma matriz adjacência. **Fonte:** Rocha (2013)

Na Figura 7, as posições da matriz cujo o valor é igual a 1, definem que há uma aresta conectando os vértices, tornando-os assim, adjacentes. Caso o valor seja 0, os vértices não estão conectados entre si no grafo e, portanto, não são vértices adjacentes.

Este conteúdo teórico foi escolhido para ser utilizado neste trabalho pois este visa equacionar o problema relacionado à busca por mão de obra, através do modelo utilizado pelas redes sociais. Isto é possível pois, como mencionado anteriormente por meio desta teoria, é possível

descrever várias situações do mundo real e como ela é muito bem aplicada à redes sociais, inclusive grandes empresas desta área já a utilizam. Devido a esses motivos, a teoria dos grafos foi utilizada para auxiliar no desenvolvimento deste trabalho.

2.3 Tecnologias

Nesta seção serão abordadas as linguagens de programação e as tecnologias que serão utilizadas para o desenvolvimento deste trabalho.

2.3.1 Banco de dados

A expressão "banco de dados" teve origem a partir do termo inglês *Databanks*, que foi substituído, mais tarde, pela palavra *Databases* (Base de dados) por possuir um significado mais apropriado (SETZER; SILVA, 2005).

De acordo com Date (2003), um banco de dados é uma coleção de dados persistentes, usada pelos sistemas de aplicação em uma determinada empresa. Sendo assim, um banco de dados é um local onde são armazenados os dados necessários para manter as atividades de determinadas organizações.

Um banco de dados possui, implicitamente, as seguintes propriedades: representa aspectos do mundo real; é uma coleção lógica de dados que possuem um sentido próprio e armazena dados para atender uma necessidade específica. O tamanho do banco de dados pode ser variável, desde que ele atenda às necessidades dos interessados em seu conteúdo (ELMASRI; NAVATHE, 2005).

A escolha do banco de dados que será utilizado em um projeto é uma decisão importante e que deve ser tomada na fase de planejamento, pois determina características da futura aplicação, como a integridade dos dados, o tratamento de acesso de usuários, a forma de realizar uma consulta e o desempenho. Portanto, essa decisão deve ser bem analisada, levando em consideração o tipo de software e no ambiente de produção que será utilizado.

Nas seções seguintes, são demonstrados os principais modelos de banco de dados, abordando suas características.

2.3.2 Banco de dados relacionais

O modelo de banco de dados relacional foi introduzido em 1970, por Edgar Frank Codd, em uma publicação com o título: “A relational model of data for large shared data banks”, na revista *Association for Computing Machinery* (ACM). Essa publicação demonstrou como tabelas podem ser usadas para representar objetos do mundo real e como os dados podem ser armazenados para os objetos. Neste conceito, a integridade dos dados foi levada mais a sério do que em qualquer modelo de banco de dados antes visto. A partir desta publicação, surgiram muitos bancos de dados que passaram a utilizar este conceito e se tornaram muito utilizados no desenvolvimento de aplicações (MATTHEW; STONES, 2005).

Segundo Matthew e Stones (2005), o conceito é baseado na teoria reacional da matemática e por isso há uma grande flexibilidade para o acesso e a manipulação de dados que são gravados no banco de dados. Utilizam-se técnicas simples, como normalização na modelagem do banco de dados, criando várias tabelas relacionadas, que servem como base para consultas usando uma linguagem de consulta quase padronizada, a *Structured Query Language* – SQL⁵.

A utilização de banco de dados relacionais geraram a necessidade de dividir os dados agregados utilizados na aplicação em várias relações conforme as regras da normalização. Para recuperar o mesmo dado agregado são necessárias consultas utilizando *joins*⁶, uma operação que, dependendo do tamanho das relações e da quantidade de dados, pode não ser tão eficiente. Nos casos em que se precisa obter uma resposta rápida de um sistema, isso se torna uma desvantagem (SADALAGE; FOWLER, 2013).

Este foi um dos fatores determinantes que motivaram a criação de novas tecnologias, a fim de sanar o problema mencionado acima. A partir desta motivação, foram desenvolvidos novos modelos de banco de dados, que serão apresentados a seguir.

2.3.3 Banco de dados NoSQL

A expressão NoSQL é um termo não definido claramente. Ela foi ouvida pela primeira vez em 1998 como um nome para o banco de dados relacional de Carlo Strozzi, que assim o nomeou por não fornecer uma SQL-API. O mesmo termo foi usado como nome do evento

⁵ SQL: *Structured Query Language* - Linguagem para consultas e alterações em bancos de dados.

⁶ *joins* - função utilizada para realizar a junção entre tabelas, facilitando a busca em bancos de dados relacionais.

NoSQL Meetup em 2009, que teve como objetivo a discussão sobre sistemas de bancos de dados distribuídos.

Devido à explosão de conteúdos na *web* no início do século XXI, houve a necessidade de substituir os bancos de dados relacionais por bancos que oferecem maior capacidade de otimização e performance, a fim de suportar o grande volume de informações eminentes a esta mudança (BRUGGEN, 2014).

Rocha (2013, p. 27) afirma que NoSQL é "um acrônimo para Not only SQL, indicando que esses bancos não usam somente o recurso de Structured Query Language (SQL), mas outros recursos que auxiliam no armazenamento e na busca de dados em um banco não relacional".

Segundo Bruggen (2014), os banco de dados NoSQL podem ser categorizados de 4 maneiras diferentes, são elas: *Key-Value stores*⁷, *Column-Family stores*⁸, *Document stores*⁹ e *Graph Databases*¹⁰.

De acordo com Bruggen (2014), o banco de dados orientado a grafo (*graph database*) pertence a categoria NoSQL, contudo, ele possui particularidades que o torna muito diferente dos demais tipos de bancos de dados NoSQL. A seguir, será descrito com maiores detalhes o banco de dados orientado a grafos Neo4j.

2.3.4 Neo4j

O Neo4j foi criado no início do século XXI por desenvolvedores que queriam resolver um problema em uma empresa de mídias. Porém, eles não obtiveram êxito ao tentar resolver tal problema utilizando as tecnologias tradicionais, portanto, decidiram arriscar e criar algo novo. A princípio, o Neo4j não era um sistema de gerenciamento de banco de dados orientado a grafos como é conhecido nos dias atuais. Ele era mais parecido com uma *graph library* (biblioteca de grafo) em que as pessoas poderiam usar em seus projetos (BRUGGEN, 2014).

De acordo com Bruggen (2014), inicialmente ele foi desenvolvido para ser utilizado em conjunto com alguns bancos de dados relacionais como MySQL e outros, com a intenção de criar uma camada de abstração dos dados em grafos. Mas com o passar dos anos, os desenvolvedores decidiram tirar o Neo4j da estrutura dos bancos relacionais e criar sua própria estrutura de armazenamento em grafos.

⁷ *Key-Value stores* - armazenamento por um par de chave e valor.

⁸ *Column-Family stores* - armazenamento por colunas e linhas.

⁹ *Document stores* - armazenamento em arquivos.

¹⁰ *Graph Databases* - banco de dados orientado a grafo.

O Neo4j, como vários outros, também é um projeto de sistema de gerenciamento de banco de dados NoSQL de código fonte aberto.

Segundo Robinson, Webber e Eifrem (2013), os bancos de dados orientados a grafos possuem como diferencial a sua performance, agilidade e flexibilidade. Entretanto, a performance é o que mais se destaca entre eles, pois, a maneira como eles armazenam e realizam buscas no banco de dados é diferente dos bancos de dados convencionais. Primeiramente, esse tipo de banco de dados não utiliza tabela; ele armazena os dados em vértices e arestas. Isso permite realizar buscas extremamente velozes através de *traversals* (travessias), uma vez que estas implementam algoritmos para otimizar tais funcionalidades, evitando assim o uso de *joins* complexos, tornando-o tão veloz.

Neo4j (2013, p. 2) afirma que:

*A single server instance can handle a graph of billions of nodes and relationships. When data throughput is insufficient, the graph database can be distributed among multiple servers in a high availability configuration.*¹¹

Com estas informações, é possível mensurar o quanto o Neo4j pode ser rápido e robusto, sendo possível, até mesmo distribuí-lo a fim de obter uma melhor configuração, organização e facilidade de manutenção.

Segundo Neo4j (2013), o banco de dados Neo4j é composto por nós (vértices), relacionamentos (arestas) e propriedades. Os relacionamentos são responsáveis por organizar os nós e ambos podem possuir seus atributos. É possível realizar as buscas e/ou alterações no Neo4j de duas formas diferentes. Sendo a primeira através da API *Cypher Query Language*, que é uma *query language* para banco de dados orientado a grafos muito próxima da linguagem humana, cuja descrição completa será apresentada a seguir. A segunda é o *framework*¹² *Traversal* que utiliza a API *Cypher* internamente para navegar pelo grafo.

Rocha (2013) afirma que o Neo4j permite criar mais de um relacionamento entre o mesmo par de vértices, desde que estes sejam de tipos distintos. Isso possibilita navegar pelos vértices do grafo de forma mais rápida devido a esses diferentes tipos de arestas, o que torna possível implementar o algoritmo de busca desejado. A Figura 8 exemplifica um simples grafo utilizando um banco de dados Neo4j.

Há duas formas de executar o Neo4j, segundo Robinson, Webber e Eifrem (2013). A primeira é conhecida como *Server* e a segunda *Embedded*. O modo *Server* é utilizado princi-

¹¹ Um único servidor pode manipular um grafo de bilhões de nós e relacionamentos. Quando a taxa de transferência de dados é insuficiente, o banco de dados orientado a grafo pode ser distribuído entre vários servidores mantendo a mesma velocidade de processamento.

¹² *Framework* - Abstração que une códigos comuns entre vários projetos de software, a fim de obter uma funcionalidade genérica.

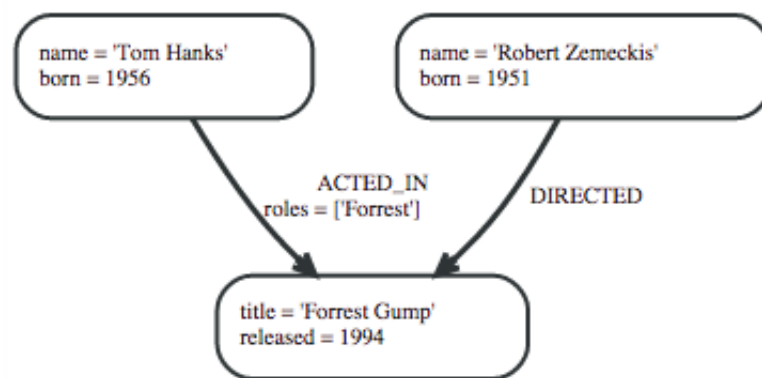


Figura 8 – Exemplo simples de um grafo no Neo4j. **Fonte:** Neo4j (2013)

palmente em *web-service* em conjunto com a API REST, este será aplicado neste trabalho. Já no modo *Embedded* o banco de dados é executado embarcado à aplicação Java.

Conforme Neo4j (2013), o Neo4j possui suporte as transações ACID (com atomicidade, consistência, isolamento e durabilidade).

O Neo4j é distribuído em duas versões sendo elas a *Enterprise* e a *Community*. A primeira possui um tempo de avaliação de 30 dias e após esse tempo, é necessário comprar uma licença para continuar a utilizá-lo. Como diferencial essa versão possui ferramentas para o gerenciamento do banco de dados, incluindo melhorias relacionadas à escalabilidade. A segunda versão é disponibilizada gratuitamente sem data limite de expiração, contudo, ela não possui os recursos mencionados anteriormente que estão presentes na versão *Enterprise*, mas é muito utilizada para fins didáticos e para pequenos projetos, aplicando-se perfeitamente a este projeto (NEO4J, 2013).

Por ser um banco de dados orientado a grafo bastante robusto, seguro e possuir uma documentação de fácil entendimento, além, é claro, de possuir um baixo custo de implantação devido a sua licença *open source*, esse banco de dados foi escolhido para ser utilizado neste trabalho.

2.3.5 Cypher Query Language

O *Cypher Query Language* é uma linguagem para consultas em banco de dados orientado a grafo específica para o banco Neo4j. Ela foi criada devido à necessidade de manipular os dados e realizar buscas em grafos de uma forma mais simples, uma vez que, não é necessário escrever *traversals* (travessias) para navegar pelo grafo (NEO4J, 2013).

Robinson, Webber e Eifrem (2013), afirmam que o *Cypher* foi desenvolvido para ser

uma *query language* que utiliza uma linguagem formal, permitindo a um ser humano entendê-la. Desta forma, qualquer pessoa envolvida no projeto é capaz de compreender as consultas realizadas no banco de dados.

Segundo Neo4j (2013), o *Cypher* foi inspirado em uma série de abordagens e construído sob algumas práticas já estabelecidas, inclusive a SQL. Por este motivo, é possível notar que ele utiliza algumas palavras reservadas que são comuns na SQL como *WHERE* e *ORDER BY*.

De acordo com Neo4j (2013), o *Cypher* é composto por algumas cláusulas, dentre elas, se destacam:

- *START*: define um ponto inicial para a busca, esse ponto pode ser um relacionamento ou um nó.
- *MATCH*: define o padrão de correspondência entre os nós. Para identificar um nó é necessário incluí-lo entre um par de parênteses, e os relacionamentos são identificados um hífen e um sinal de maior ou menor.
- *CREATE*: utilizado para criar nós e relacionamentos no grafo.
- *WHERE*: define um critério de busca.
- *RETURN*: define quais nós, relacionamentos e propriedades de ambos devem ser retornados da *query* realizada.
- *SET*: utilizado para editar as propriedades de um nó ou de um relacionamento.
- *UNION*: possibilita juntar o resultado de duas ou mais consultas.
- *FOREACH*: realiza uma ação de atualização para cada elemento na lista.

Outras *Query Languages* existem, inclusive com suporte ao Neo4j, porém devido às vantagens apresentadas acima, somada ao fato de que ele possui uma curva de aprendizagem menor e é excelente para lhe oferecer uma base a respeito de grafos, este *framework* será utilizado para realizar as tarefas de manipulação dos dados no banco de dados.

2.3.6 Java

Segundo Schildt (2007), a primeira versão da linguagem Java foi criada por James Gosling, Patrick Naughton, Chris Warth, Ed Frank e Mike Sheridan na *Sun Microsystems* em 1991 e

denominada "Oak" tendo como principal foco a interatividade com a TV. Mais tarde, em 1995, a *Sun Microsystems* renomeia esta linguagem e anuncia publicamente a tecnologia Java, focando nas aplicações *web*, que em pouco tempo e devido à grande ascensão da internet, cresceu e se mantém em constante evolução até os dias atuais.

De acordo com a Oracle (2010), a tecnologia Java não é apenas uma linguagem, mas também uma plataforma, que teve como modelo uma outra linguagem, o C++ que, por sua vez foi derivada da linguagem C. O C++ e o Java possuem em comum o conceito de orientação a objetos, o que permite a esta linguagem utilizar recursos como: generalização (herança), implementação, polimorfismo, entre outras. Tais funcionalidades permitem ao desenvolvedor escrever códigos reutilizáveis, a fim de facilitar o desenvolvimento do projeto.

Para Schildt (2007), o paradigma de orientação a objetos foi criado devido às limitações que o conceito estrutural apresentava quando era utilizado em projetos de grande porte, dificultando o desenvolvimento e manutenção dos mesmos. Este paradigma possibilita ao desenvolvedor aproximar o mundo real ao desenvolvimento de *software*, deixando os objetos do mundo real semelhantes a seus respectivos objetos da computação, possibilitando ao desenvolvedor modelar seus objetos de acordo com suas necessidades (FARIA, 2008).

Segundo Schildt (2007), o recurso denominado generalização (herança) permite ao desenvolvedor criar uma classificação hierárquica de classes. Além disso, é possível escrever uma classe genérica contendo comportamentos comuns, e as demais classes, cujos comportamentos também serão aplicados a ela, somente precisa generalizar esta classe.

O polimorfismo se refere ao princípio da biologia em que um organismo pode ter diferentes formas ou estados. Esse mesmo princípio também pode ser aplicado à programação orientada a objeto. Desta forma, é possível definir comportamentos que serão compartilhados entre as classes e suas respectivas subclasses, além de comportamentos próprios que apenas as sub classes possuem. Com isso, o comportamento pode ser diferente de acordo com a forma e/ou o estado do objeto (ORACLE, 2015a).

Retomando a ideia da Oracle (2010), uma das vantagens da tecnologia Java sob as demais é o fato de ela ser multiplataforma, possibilitando ao desenvolvedor escrever o código apenas uma vez e este, ser executado em qualquer plataforma inclusive em hardwares com menor desempenho. Isso é possível, pois, para executar um programa em Java é necessário possuir uma *Java Virtual Machine* - JVM¹³ - instalada no computador. A JVM compreende e executa apenas *bytecodes*¹⁴ e estes por sua vez são obtidos através do processo de compilação do código

escrito em Java.

Todo programa que utiliza Java necessita passar por algumas etapas essenciais. Conforme ilustrado na Figura 9, o código é escrito em arquivo de texto com extensão `.java`, após isso ele será compilado e convertido para um arquivo com extensão `.class`, cujo o texto é transformado em *bytecodes*. Este arquivo com extensão `.class` é interpretado pela JVM que é responsável por executar todo o código do programa.

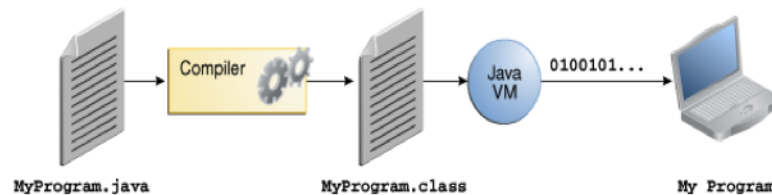


Figura 9 – Uma visão geral do processo de desenvolvimento de software. **Fonte:** Oracle (2010)

Por todas as vantagens descritas anteriormente, foi empregada esta tecnologia neste trabalho.

2.3.7 Tomcat 7

O Tomcat é uma aplicação *container*, capaz de hospedar aplicações *web* baseadas em Java. A princípio ele foi criado para executar *servlets*¹⁵ e *JavaServer Pages* - JSP¹⁶. Inicialmente ele era parte de um sub projeto chamado *Apache-Jakarta*, porém, devido ao seu sucesso ele passou a ser um projeto independente, e hoje, é responsabilidade de um grupo de voluntários da comunidade *open source* do Java (VUKOTIC; GOODWILL, 2011).

Segundo a Apache (2015), o Tomcat é um software que possui seu código fonte aberto e disponibilizado sob a *Apache License Version 2*. Isto o fez se tornar uma das aplicações *containers* mais utilizadas por desenvolvedores.

Containers são aplicações que são executadas em servidores e possuem a capacidade de hospedar aplicações desenvolvidas em Java *web*. O servidor ao receber uma requisição do cliente, entrega esta ao *container* no qual é distribuído. O *container*, por sua vez, entrega ao

¹³ JVM: *Java Virtual Machine* - Máquina virtual utilizada pela linguagem Java para execução e compilação de softwares desenvolvidos em Java.

¹⁴ *Bytecode* é o código interpretado pela JVM. Ele é obtido por meio do processo de compilação de um programa java como mencionado anteriormente.

¹⁵ *Servlet* - Programa Java executado no servidor, semelhante a um *applet*.

¹⁶ JSP: *JavaServer Pages* - Tecnologia utilizada para desenvolver páginas interativas utilizando Java *web*.

servlet as requisições e respostas HTTP¹⁷ e inicia os métodos necessários do *servlet* de acordo com o tipo de requisição realizada pelo cliente (BASHMAN; SIERRA; BATES, 2010).

Brittain e Darwin (2007) afirmam que o Tomcat foi desenvolvido utilizando a linguagem de programação Java, sendo necessário possuir uma versão do *Java Runtime Environment* - JRE¹⁸ - instalada e atualizada para executá-lo.

De acordo com Laurie e Laurie (2003), o Tomcat é responsável por realizar a comunicação entre a aplicação e o servidor Apache¹⁹ por meio do uso de *sockets*.

Assim como outros *containers*, Bashman, Sierra e Bates (2010) afirmam que o Tomcat oferece gerenciamento de conexões *sockets*, suporta *multithreads*, ou seja, ele cria uma nova *thread* para cada requisição realizada pelo cliente e gerencia o acesso aos recursos do servidor, além de outras tarefas.

O Tomcat, em especial, foi escolhido para ser utilizado neste trabalho, pois o objetivo é desenvolver uma aplicação *web* e para hospedá-la em um servidor, uma aplicação *container* se faz necessária. Por este motivo, e somado a sua facilidade de configuração, além das vantagens acima descritas tal decisão foi tomada.

2.3.8 Web Service REST

A definição computacional de serviço é um *software* que disponibiliza sua funcionalidade por meio de uma interface denominada contrato de serviço (ERL et al., 2012).

Web Service de acordo com Marzullo (2013), é uma materialização da ideia de um serviço que é disponibilizado na internet, e que, devido a isso, pode ser acessado em qualquer lugar do planeta e por diferentes tipos de dispositivos. Para ter acesso aos serviços que o *Web Service* disponibiliza, o solicitante envia requisições de um tipo anteriormente definido e recebe respostas síncronas ou assíncronas.

Marzullo (2013) afirma que, a implementação de um *Web Service* é relativamente simples, uma vez que, há inúmeras ferramentas que facilitam a implementação do mesmo. Outro fator que permite a um *Web Service* ser mais dinâmico é possuir uma estrutura interna fracamente acoplada, permitindo, assim, mudanças em suas estruturas sem afetar a utilização pelo cliente.

¹⁷ HTTP: *Hypertext Transfer Protocol* - Protocolo de transferência de dados mais utilizado na rede mundial de computadores.

¹⁸ JRE: *Java Runtime Environment* - Conjunto de ferramentas necessárias para a execução de aplicações desenvolvidas na linguagem Java.

¹⁹ Apache - Servidor cujo aplicação Tomcat é executada.

Erl et al. (2012) afirmam que o REST²⁰, é uma das várias implementações utilizadas para criar serviços. Outra implementação também muito conhecida é: a SOAP²¹ em conjunto com o WSDL²², cujo responsabilidade é definir o contrato dos serviços.

O primeiro *Web Service* REST foi criado por *Roy Fielding* no ano 2000 na universidade da Califórnia. Ele foi criado para suceder a tecnologia SOAP. A fim de facilitar a utilização e a aceitação desta nova tecnologia, *Roy Fielding* lançou mão do protocolo HTTP e o definiu como o protocolo de comunicação para *Web Services* REST, sua decisão se baseou no fato de que, este protocolo já possui mecanismos de segurança implementados, além de ser o protocolo padrão utilizado na internet para transferência de dados e acesso a recursos como páginas *web*, o que tornaria-o mais bem aceito (RODRIGUEZ, 2008).

Segundo Oracle (2015b), na arquitetura do REST os dados e as funcionalidades são considerados recursos e ambos são acessados por meio de URIs²³. A arquitetura REST se assemelha a arquitetura *client/server* e foi desenvolvido para funcionar com protocolos de comunicação baseados em estado, como o HTTP. Como nos *Web Services* SOAP o acesso aos recursos do REST também é realizado por meio de contratos anteriormente definidos.

Para Rodriguez (2008), o *Web service* REST segue quatro princípios básicos. São eles:

- utiliza os métodos HTTP explicitamente;
- é orientado à conexão;
- expõe a estrutura de diretório por meio das URIs;
- trabalha com *-Extensible Markup Language - XML*²⁴, *Javascript Object Notation - JSON*²⁵ - ou ambos.

Essa tecnologia foi selecionada para ser utilizada neste trabalho, pois a forma de acesso ao banco de dados Neo4j utiliza uma API REST fornecida pelo próprio Neo4j,

²⁰ REST: *Representational State Transfer* - Tecnologia utilizada por *web services*

²¹ SOAP: *Simple Object Access Protocol* - Tecnologia utilizada por *web services* anteriores aos *Web services* REST.

²² WSDL: *Web Service Description Language* - Padrão de mercado utilizado para descrever *Web Services*.

²³ URI: *Uniform Resource Identifier* - Link completo para acessar um determinado recurso.

²⁴ XML: *Extensible Markup Language* - Tecnologia utilizada para transferência de dados, configuração, entre outras atividades.

²⁵ JSON: *Javascript Object Notation* - Notação de objetos via Javascript, porém seu uso não é restrito apenas ao Javascript.

2.3.9 HTML 5

Segundo W3C (2015a), *Hypertext Markup Language* - HTML²⁶ - é a linguagem usada para descrever o conteúdo das páginas *web*. Ela utiliza marcadores denominados *tags* para identificar aos navegadores de internet como eles devem interpretar tal documento.

Silva (2011a) afirma que o HTML foi criado única e exclusivamente para ser uma linguagem de marcação e estruturação de documentos (páginas) *web*. Portanto, não cabe a ele definir os aspectos dos componentes como cores, espaços, fontes, etc.

W3C (2015a) afirma que a primeira versão do HTML foi criada no ano de 1991 pelo inventor da *web*, Tim Berners-Lee. A partir desta versão, o HTML foi e continua em constante atualização e hoje se encontra na sua oitava versão. As oito versões são: HTML, HTML +, HTML 2.0, 3.0, 3.2, 4.0, 4.01 e a versão atual é a 5.

Ao longo dos anos e da evolução propriamente dita do HTML, novas *tags* foram criadas, padrões adotados, e claro, novas versões foram criadas, até que, em maio de 2007 o *World Wide Web Consortium* - W3C²⁷ - confirma a decisão de voltar a trabalhar na atual versão do HTML, também conhecida por HTML 5 (W3C, 2015a).

Silva (2011b) afirma que em novembro daquele mesmo ano o W3C publicou uma nota contendo uma série de diretrizes que descrevem os princípios a serem seguidos ao desenvolver utilizando o HTML 5 em algumas áreas. Tais princípios permitiram a esta versão maior segurança, maior compatibilidade entre navegadores e interoperabilidade entre diversos dispositivos.

Por estes motivos, somado ao fato de que ao utilizar essa tecnologia é possível obter uma maior flexibilidade no desenvolvimento das páginas *web*. O HTML em conjunto com outras tecnologias como: CSS 3, Javascript e Angular JS, cujas descrições são apresentadas nas próximas seções deste trabalho, foi selecionada para ser utilizada neste trabalho.

2.3.10 CSS 3

Silva (2011a) afirma que a principal função do *Cascading Style Sheet* - CSS²⁸ - é definir como os componentes anteriormente estruturados nos documentos *web*, por meio do HTML,

²⁶ HTML: *Hypertext Markup Language* - Linguagem usada para descrever o conteúdo das páginas *web*.

²⁷ W3C: *World Wide Web Consortium* - Consórcio internacional formado por empresas, instituições, pesquisadores, desenvolvedores e público em geral, com a finalidade de elevar a *web* ao seu potencial máximo.

devem ser apresentados ao usuário.

Para W3C (2015b), o CSS é "*a simple mechanism for adding style (e.g., fonts, colors, spacing) to Web documents*²⁹".

Tim Berners-Lee inicialmente escrevia as estilizações de seus documentos *web*, mesmo que de forma simples e limitada, nos próprios documentos HTML. Isto se deve ao fato de, ele acreditar que tal função deveria ser realizada pelos navegadores. Entretanto, em 1994 a primeira proposta de criação do CSS surgiu e, em 1996, a primeira versão do CSS denominada CSS 1 foi lançada como recomendação do W3C (SILVA, 2011a).

De acordo com Silva (2011a), atualmente o CSS possui quatro versões, são elas: A CSS 1, a 2, a 2.1 e atualmente a 3, que foi utilizada para o desenvolvimento deste trabalho.

Há três formas de incorporar o CSS em seu documento web segundo (SILVA, 2011a), são elas:

- **Inline:** É possível aplicar o estilo diretamente ao componente desejado, por meio do uso da propriedade `style` do componente HTML. Como é apresentado na Figura 10;

```
<!DOCTYPE html>
<html>
<head>
  <title>Título da página</title>
</head>
<body>
  <p style="color: red;">
    |Exemplo de estilo CSS aplicado diretamente no componente
  </p>
</body>
</html>
```

Figura 10 – Exemplo de inclusão do estilo CSS inline. **Fonte:** Elaborado pelos autores.

- **Incorporado:** Outra forma, é escrever todo CSS referente ao documento *web* dentro da *tag* `style` do documento HTML. Para tanto, esta *tag* deve ser inserida entre o início e o fim da *tag* `head` do documento. Como é apresentado na Figura 11;
- **Externo:** A última forma, é criar um arquivo externo com extensão `.css` e definir todas as regras de estilização do documento *web* neste arquivo. Desta forma, para vincular tal arquivo a um documento HTML específico será necessário utilizar a *tag* `link` entre o início e o fim da *tag* `head` do documento, como apresentado na Figura 12;

²⁸ CSS: *Cascading Style Sheet* - Documentos que definem estilos aos componentes da página *web*.

²⁹ O CSS é um simples mecanismo para adicionar estilo, como: fontes, cores, espaços para os documentos *Web*.

```

<!DOCTYPE html>
<html>
<head>
  <title>Título da página</title>
  <style>
    p {
      color: red;
    }
  </style>
</head>
<body>
  <p>
    Exemplo de estilo CSS aplicado incorporado à página HTML por meio da tag STYLE
  </p>
</body>
</html>

```

Figura 11 – Exemplo de inclusão do estilo CSS incorporado à página HTML. **Fonte:** Elaborado pelos autores.

```

<!DOCTYPE html>
<html>
<head>
  <title>Título da página</title>
  <link rel="stylesheet" href="estilo.css">
</head>
<body>
  <p>
    Exemplo de estilo CSS carregado a partir de um arquivo externo
  </p>
</body>
</html>

```

Figura 12 – Exemplo de inclusão do estilo CSS a partir de um arquivo externo. **Fonte:** Elaborado pelos autores.

O CSS 3 será utilizado neste trabalho, pois, ele permite definir estilos aos componentes das páginas *web* e possui recursos que não são existentes em suas versões anteriores, o que nos permite desenvolver páginas mais atrativas com menos recursos.

2.3.11 Javascript

Frank e Seibt (2002) afirmam que o Javascript foi criado e lançado pela Netscape em 1995, em conjunto com o navegador de internet Netscape Navigator 2.0. A partir deste lançamento, as páginas *web* passaram a ganhar vida com a possibilidade de implementar um mínimo de dinamicidade. Isto se deve ao modo como a linguagem acessa e manipula os componentes do navegador. Contudo, ela pode ser utilizada em diferentes dispositivos como *smartphones*, *smart tv*, entre outros, não limitando-se apenas a navegadores de internet.

O Javascript é uma linguagem de programação para *web*. A maioria dos sites usa essa linguagem, inclusive todos os navegadores mais modernos, vídeo games, *tablets*, *smart phones*,

smart tvs possuem interpretadores de *Javascript*, o que a tornou, a linguagem de programação mais ambígua da história (FLANAGAN, 2011).

Segundo Frank e Seibt (2002), as semelhanças entre o Javascript e o Java se limitam apenas ao nome. A primeira linguagem não deriva da segunda, apesar de ambas compartilharem alguns conceitos e detalhes. O Javascript, por ser uma linguagem interpretada, é mais flexível que o Java, que, por sua vez, é uma linguagem compilada.

De acordo com Flanagan (2011), o Javascript possui 6 versões, sendo elas: 1.0, 1.1, 1.2, 1.3, 1.4 e a atual versão 1.5.

Para Frank e Seibt (2002), há duas maneiras de incluir e executar o código escrito em Javascript nos documentos HTML. A primeira é incluir o código Javascript entre as *tags script* como mostra a Figura 13.

```
<!DOCTYPE html>
<html>
<head>
  <title>Título da página</title>
</head>
<body>
  <script type="text/javascript">
    document.writeln('Exemplo de um código Javascript incorporado ' +
                      'ao documento HTML por meio da tag SCRIPT!');
  </script>
</body>
</html>
```

Figura 13 – Exemplo de inclusão do código em Javascript incorporado ao HTML. **Fonte:** Elaborado pelos autores.

A segunda forma é incluir um arquivo externo com extensão *.js* através da mesma *tag script*. Veja um exemplo de inclusão de um arquivo contendo códigos em Javascript no documento HTML na Figura 14.

```
<!DOCTYPE html>
<html>
<head>
  <title>Título da página</title>
</head>
<body>
  <script type="text/javascript" src="Exemplo.js"></script>
</body>
</html>
```

Figura 14 – Exemplo de inclusão do código Javascript de um arquivo externo. **Fonte:** Elaborado pelos autores.

Para Flanagan (2011), as três tecnologias (HTML, CSS e Javascript) devem ser usadas em conjunto, uma vez que, cada uma delas possui seu papel específico, sendo eles: o HTML usado para especificar o conteúdo da página *web*, o CSS para especificar como os componentes serão apresentados e o Javascript para especificar o comportamento da página.

Pelos motivos acima mencionados, somado ao fato de que o Javascript permite ao desenvolvedor criar páginas *web* mais dinâmicas e flexíveis, atendendo perfeitamente os requisitos deste trabalho, essa tecnologia será utilizada para auxiliar a criação das páginas *web* deste trabalho.

2.3.12 Angular JS

Segundo Green e Seshadri (2013), o framework Angular JS foi criado para facilitar o desenvolvimento de aplicativos *web*, pois através dele, é possível criar um aplicativo *web* com poucas linhas.

Para o criador do Angular JS, Miško Hevery, o que o motivou a criar o Angular JS foi a necessidade de ter que reescrever determinados trechos de códigos em todos os outros projetos, o que para ele, tornou-se inviável. Portanto, Miško decidiu criar algo para facilitar o desenvolvimento de aplicativos *web* de uma forma que ninguém havia pensado antes.

O Angular JS é um framework *Model-View-Controller* - MVC³⁰ - escrito em Javascript. Ele é executado pelos navegadores de internet e ajuda os desenvolvedores a escreverem modernos aplicativos *web* (KOZLOWSKI; DARWIN, 2013).

Devido às facilidades que o Angular JS nos traz, é que ele foi escolhido para ser utilizado, a fim de auxiliar no desenvolvimento, não apenas das páginas web e seus respectivos conteúdos, como também na lógica e comunicação com o *Web Service* REST.

³⁰ MVC: *Model-View-Controller* - Design pattern.

3 QUADRO METODOLÓGICO

Neste quadro metodológico serão apresentados os passos que se fizeram necessários para a realização desta pesquisa. Nele estão descritos desde a escolha do perfil da pesquisa até os procedimentos utilizados para o seu desenvolvimento. Gil (1999) cita que a metodologia é um conjunto de procedimentos intelectuais e técnicos que trabalham para a realização do objetivo proposto.

Posterior ao estudo do levantamento teórico e técnico, foram definidos os procedimentos para a construção deste trabalho, iniciando pela escolha do tipo de pesquisa, demonstrada na seção a seguir.

3.1 Tipo de pesquisa

Para Pádua (2007, p. 31), pesquisa é:

Toda atividade voltada para a solução de problemas; como atividade de busca, indagação, investigação, inquirição da realidade, e a atividade que visa nos permitir, no âmbito da ciência, elaborar um conhecimento, ou um conjunto de conhecimentos, que nos auxilie na compreensão desta realidade e nos oriente em nossas ações.

De forma objetiva, a pesquisa é o meio utilizado para buscar respostas aos mais diversos tipos de indagações, tendo por base procedimentos racionais e sistemáticos. A pesquisa é realizada quando se tem um problema e não se têm informações suficientes para solucioná-lo. Por meio desta sessão, tem-se como objetivo explicar o tipo de pesquisa que norteou o desenvolvimento deste trabalho, justificando também como ele se enquadra no tipo escolhido.

Pesquisar é um trabalho que envolve planejamento, para que ela seja satisfatória, o pesquisador precisa estar envolvido e desenvolver habilidades técnicas que o levem a escolher o melhor caminho em busca da obtenção dos resultados.

Segundo Fonseca (2002), ter um método de pesquisa envolve o estudo dos fatores que compõem o contexto da pesquisa, tais como, a escolha do caminho e o planejamento do percurso. Essa escolha inicia-se com a definição do tipo de pesquisa utilizada. Para este trabalho, é utilizada a pesquisa aplicada, que é aquela cujo o pesquisador tem como objetivo aplicar os conhecimentos obtidos durante o período da pesquisa, em um projeto real, a fim de conhecer os

seus resultados. Gil (2007) afirma que este tipo de pesquisa dirige-se à solução de problemas específicos, de interesses locais.

Nesta pesquisa foram estudados os conceitos de banco de dados orientado à grafos e a sua aplicabilidade, desenvolvendo, por meio dos conhecimentos obtidos pela pesquisa, uma solução prática, disponibilizada por meio de um sistema *web*, que auxilia na busca por mão de obra temporária, que não caracterize vínculo empregatício.

Seguindo o enquadramento desta pesquisa, ela deve ser aplicada a um contexto específico, conforme será abordado a seguir.

3.2 Contexto de pesquisa

O trabalho informal é um elemento estrutural da economia no Brasil e nos países em desenvolvimento. Ele faz parte do cenário atual, crescente a cada dia, e contribui ativamente com a geração de renda. É considerado como um desdobramento do excesso de mão de obra, definido a partir de pessoas que criam sua própria forma de trabalho como estratégia de sobrevivência ou como forma alternativa de recolocação no mercado de trabalho. O fortalecimento deste tipo de trabalho ocorre a partir da construção de redes, formadas por parentes e amigos, criando laços de confiança que são fundamentais para o desempenho da atividade. No entanto, há uma grande dificuldade em se encontrar estes profissionais, uma vez que não há um lugar centralizado para divulgar o seu perfil profissional.

O desenvolvimento deste trabalho se propôs a atuar sobre essa limitação. Uma pesquisa informal, realizada por meio de um questionário, na região do sul de Minas Gerais, com pessoas de diferentes perfis sociais, constatou que uma aplicação capaz de centralizar a busca por estes profissionais seria muito bem aceita. A partir deste resultado, validou-se a ideia de construir um ambiente *web* onde o trabalhador informal tenha espaço para centralizar suas habilidades e manter um perfil visível aos possíveis contratantes. Qualquer prestador de serviço informal pode ter acesso a este ambiente, desde que possua um dispositivo eletrônico capaz de se conectar à internet.

O ambiente desenvolvido também visa facilitar ao contratante a busca por estes profissionais, uma vez que não é fácil localizá-los por meio dos mecanismos de busca tradicionais. Desta forma, existe um benefício mútuo, em que contratados e contratantes dispõem da praticidade.

Enfim, o contexto ao qual esta pesquisa se destina busca ser bem abrangente, com o intuito de contribuir de forma relevante, proporcionando uma boa experiência aos envolvidos.

3.3 Instrumentos

Os instrumentos de pesquisa são as ferramentas usadas para a coleta de dados. Como afirma Marconi e Lakatos (2009, p. 117), os instrumentos de pesquisa abrangem “desde os tópicos de entrevista, passando pelo questionamento e formulário, até os testes ou escala de medida de opiniões e atitudes”. Eles são de suma importância para o desenvolvimento de um projeto, pois visam levantar o máximo de informações possíveis para nortear as tomadas de decisões. Para a realização desta pesquisa foram utilizados questionários e análise documental.

Segundo Gil (2007), o questionário é um dos procedimentos mais utilizados para obter informações, pois é uma técnica de custo razoável, apresenta as mesmas questões para todas as pessoas, garante o anonimato e pode conter as questões para atender a finalidades específicas de uma pesquisa. Se aplicada criteriosamente, essa técnica apresenta elevada confiabilidade. Podem ser desenvolvidos para medir opiniões, comportamento, entre outras questões, também pode ser aplicada individualmente ou em grupos.

Para este trabalho, foi desenvolvido um questionário informal, aplicado de forma individual, disponibilizado no ambiente virtual, o qual foi respondido com o intuito de analisar se a pesquisa aqui pretendida seria bem aceita por pessoas de diferentes perfis sociais.

A análise documental consiste em identificar, verificar e apreciar os documentos, atendendo a uma finalidade específica, que visa extrair uma informação objetiva da fonte. Para este trabalho foi utilizada a análise documental como material de apoio, que norteou tanto o desenvolvimento teórico quanto o prático.

Feita a escolha dos instrumentos, foram definidos os procedimentos necessários para que esta pesquisa fosse realizada. Estes procedimentos serão descritos na próxima seção.

3.4 Procedimentos e resultados

Para que esta pesquisa fosse levada a cabo, se fez necessária a implementação de algumas ações, as quais serão detalhadas.

O início da pesquisa deu-se através da escolha do tema, seguido pelo levantamento das tecnologias que seriam utilizadas. A princípio, foi definido um escopo contendo algumas tecnologias que haviam sido ministradas no ambiente acadêmico, diminuindo assim a curva de aprendizado. No entanto, foi preciso agregar alguns conhecimentos novos, despendendo um tempo maior para estudo. As tecnologias empregadas no desenvolvimento deste trabalho foram: a linguagem de programação Java, o banco de dados orientado a grafos Neo4j, juntamente com a API Cypher, Tomcat, Primefaces e JSF, sendo que no decorrer do desenvolvimento prático, viu-se a necessidade de substituir as duas últimas tecnologias citadas pelas linguagens HTML, CSS, Javascript e pelo *framework* Angular JS. As tecnologias que acompanharam o desenvolvimento deste trabalho até a sua conclusão estão descritas no quadro teórico desta pesquisa.

Para garantir que as tecnologias selecionadas fossem as melhores para o desenvolvimento, foram realizados alguns testes, por meio de aplicações simples. Os testes foram focados na avaliação do comportamento do banco de dados aplicado ao contexto desta pesquisa, cujo objetivo foi desenvolver uma aplicação de busca por mão de obra baseando-se em uma rede de relacionamentos. Estes testes também foram realizados como fins didáticos, visando gerar a familiarização com as tecnologias utilizadas.

Para nortear este trabalho, usou-se uma metodologia de desenvolvimento, apresentada a seguir.

3.4.1 ICONIX

O ICONIX foi escolhido como a metodologia de desenvolvimento de *software*, desempenhando um papel fundamental na organização. Sua abordagem proveu uma sequência de procedimentos, que foram seguidos conforme o necessário, levando à construção de uma aplicação estável. Como relatado no quadro teórico, foram seguidas as quatro fases definidas pelo ICONIX,

Na primeira fase, definida como análise de requisitos, foi realizado o levantamento das informações pertinentes ao desenvolvimento. Este levantamento foi realizado por meio da observação do comportamento das pessoas ao buscar por mão de obra temporária. A partir daí, foram levantadas as principais características, indispensáveis para a construção do *software* e desenvolvido o modelo de domínio inicial, como demonstra a Figura 15. Nesta fase, também foram definidas todas as ações que o usuário poderia realizar no sistema, por meio dos casos de

uso, conforme a Figura 16.

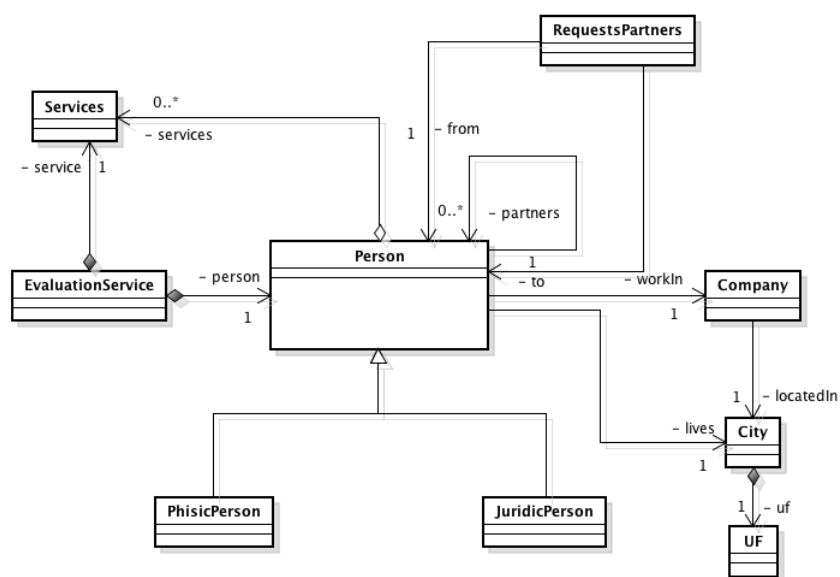


Figura 15 – Modelo de domínio inicial. **Fonte:** Elaborado pelos autores.

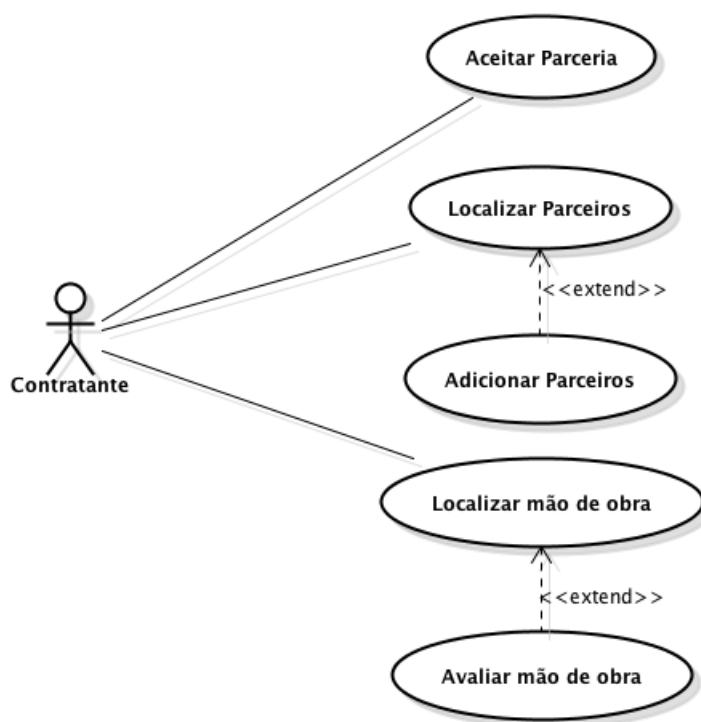


Figura 16 – Diagrama de caso de uso. **Fonte:** Elaborado pelos autores.

Agora será mostrado o exemplo do uso de fluxo de eventos apresentado no Quadro 1.

Criar Conta	
Ator principal	Usuário
Ator secundário	-
Pré-condições	
Pós-condições	Conta criada com sucesso
Fluxo Principal	
Ator	Sistema
1. O ator acessa a página inicial do sistema.	2. O sistema apresenta a página de boas vindas ao usuário.
3. O ator clica no menu “Criar conta” na barra de menu superior.	4. O sistema apresenta a tela para criar a nova conta.
5. O ator preenche alguns campos relacionados aos seus dados pessoais e clica no botão “Próximo”.	6. O sistema armazena a nova conta e redireciona o ator para a página contendo o formulário correspondente ao segundo passo para concluir a criação de conta.
7. O ator preenche alguns campos relacionados aos seus dados profissionais e clica no botão “Próximo”.	8. O sistema atualiza os dados da conta recém-criada e redireciona o ator para a página relacionada ao terceiro e último passo para criação da conta.
9. O ator insere a sua imagem de perfil e clica no botão “Salvar”.	10. O sistema atualiza a conta recém-criada e redireciona o ator para a sua página inicial.
Fluxo alternativo 1	
	1. No item 6 do fluxo principal, o sistema verifica que já existe um usuário com o mesmo e-mail.
	2. O sistema apresenta uma mensagem de erro informando a situação ao ator.
3. O ator lê a mensagem de erro.	4. O sistema mantém o estado atual da página, aguardando pela inserção de um e-mail válido.

Quadro 1 – Fluxo de eventos para o caso de uso criar conta. **Fonte:** Elaborado pelos autores

Na segunda fase, análise e projeto preliminar, houve um refinamento dos requisitos levantados na fase anterior, aperfeiçoando as ações do usuário, por meio dos diagramas de casos de uso ou fluxos de eventos. Posterior a esta definição, foram desenvolvidos os diagramas de robustez, como demonstra a Figura 17. Em paralelo, foi atualizado o modelo de domínio, acrescentando os novos atributos identificados, conforme a Figura 18. Com o modelo de domínio atualizado, foi feita a modelagem do banco de dados da aplicação, como apresenta a Figura 19.

Na terceira fase, definida como projeto detalhado, foram criados os diagramas de sequência, tendo como base os casos de uso modelados na fase anterior. Esta fase tem como objetivo detalhar todo o funcionamento do *software*, visando definir a melhor maneira de realizar sua implementação. A Figura 20 apresenta o diagrama de sequência do caso de uso localizar mão

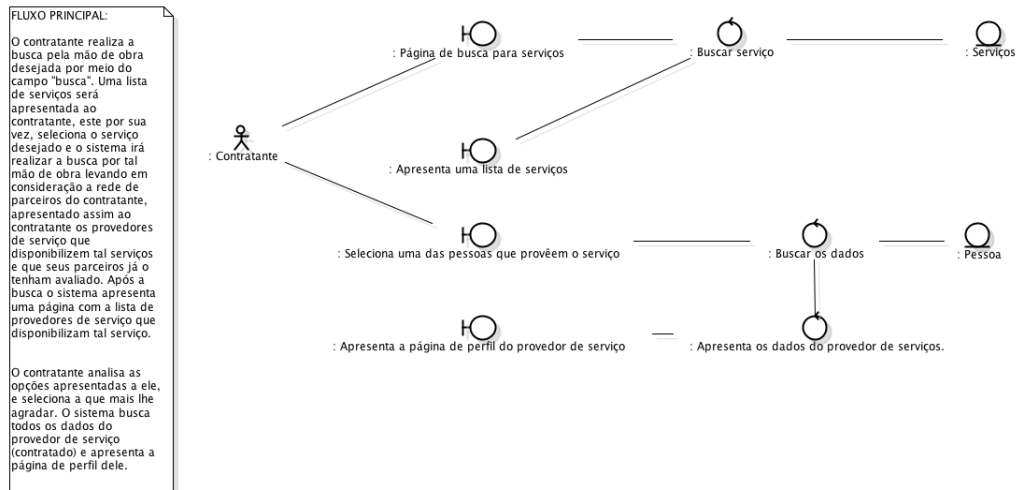


Figura 17 – Diagrama de robustez do caso de uso Localizar mão de obra. **Fonte:** Elaborado pelos autores.

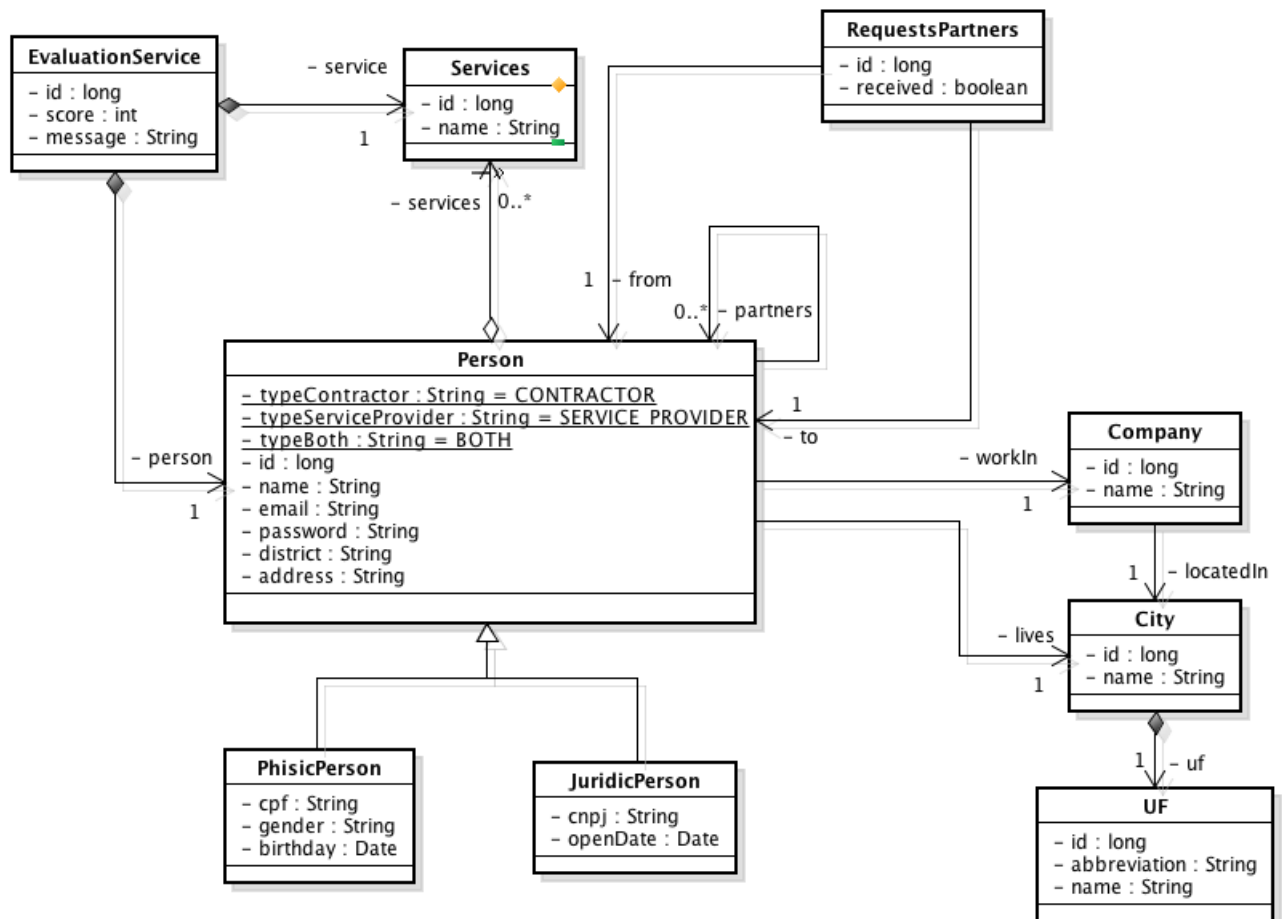


Figura 18 – Modelo de domínio atualizado. **Fonte:** Elaborado pelos autores.

de obra.

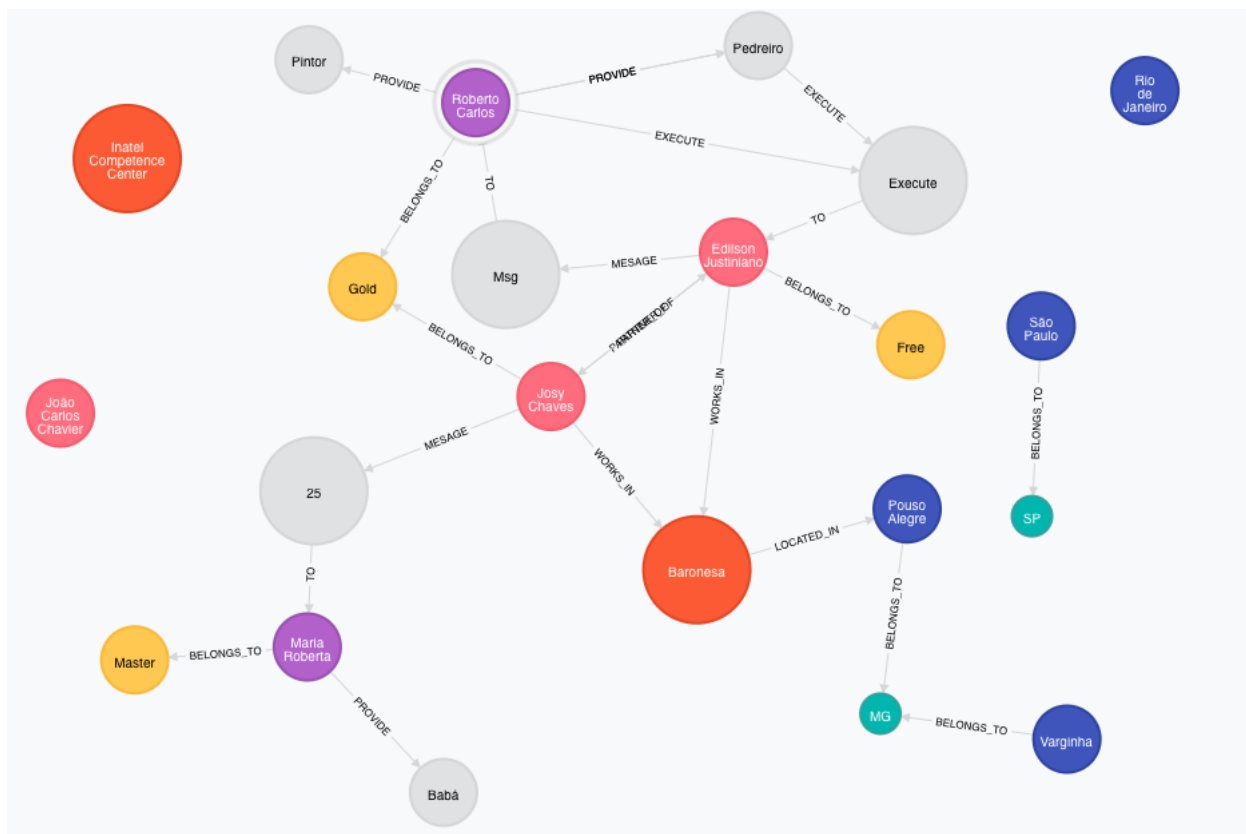


Figura 19 – Modelo de dados da aplicação. **Fonte:** Elaborado pelos autores.

Ainda na fase de projeto detalhado, após a modelagem dos diagramas de sequência, as operações encontradas nestes diagramas foram adicionadas ao modelo de domínio, em conjunto com as novas classes identificadas, gerando assim, o digrama de classes como mostra a Figura 21.

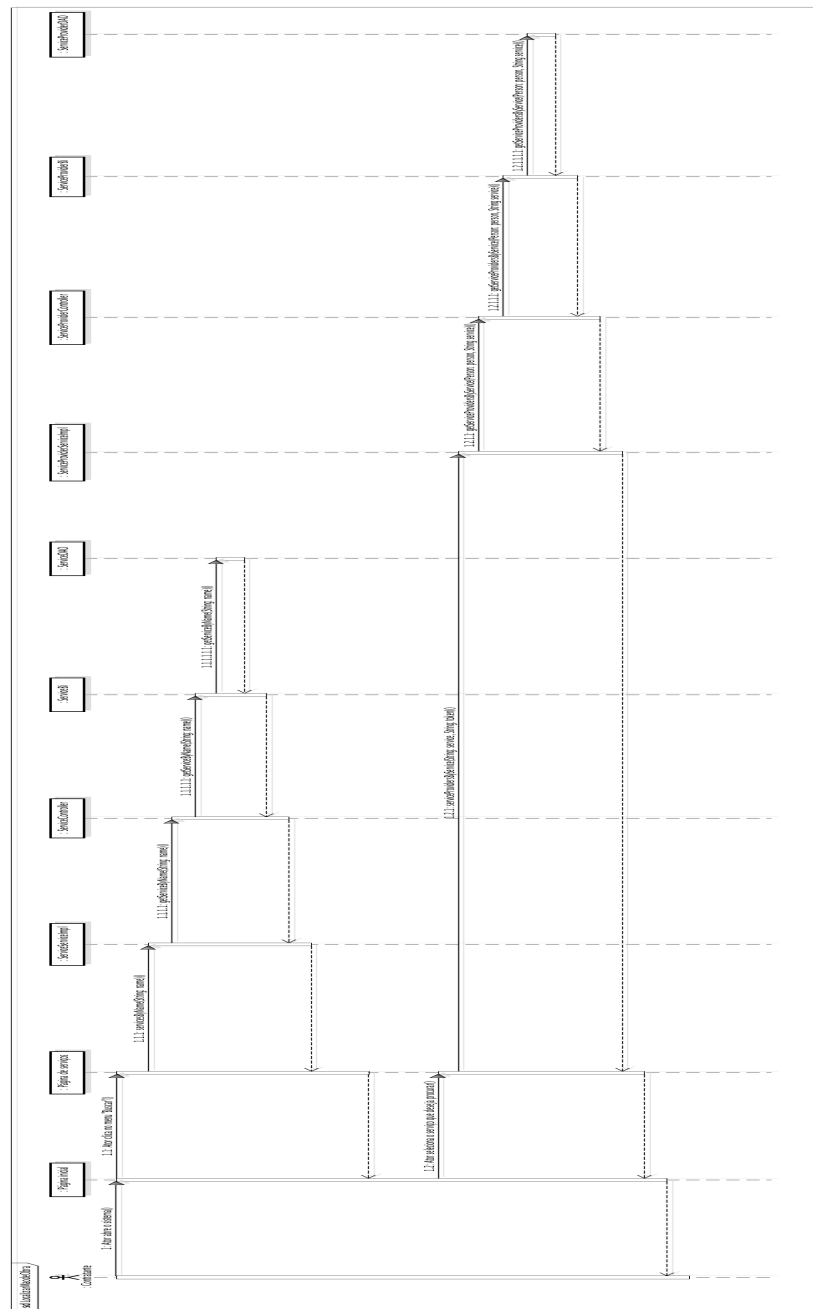


Figura 20 – Diagrama de caso de uso **Fonte:** Elaborado pelos autores.

Na quarta e última fase do ICONIX, denominada implementação, iniciou-se a preparação do ambiente, incluindo a instalação dos *softwares* necessários para o desenvolvimento prático da aplicação. Essa preparação é abordada a seguir.

estudo de uma nova ferramenta de apoio. O GitHub é uma ferramenta bem difundida e permite que os seus usuários colaborem com os projetos que estão armazenados em seus repositórios³¹. A Figura 22 demonstra a tela de serviços provida pelo GitHub.

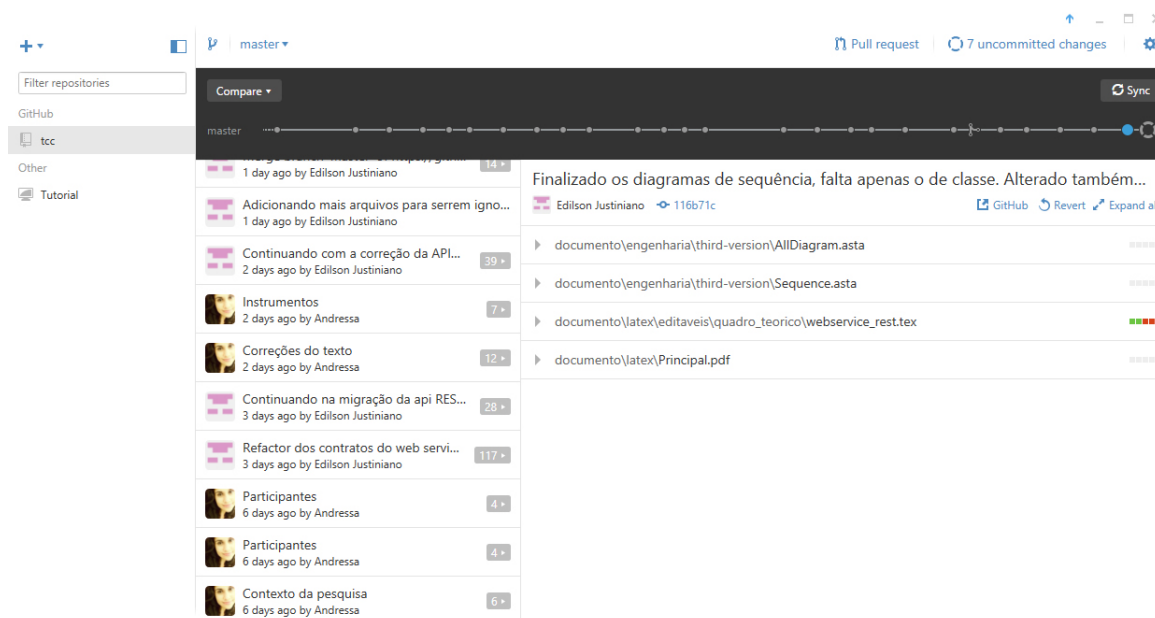


Figura 22 – Tela de serviços do GitHub **Fonte:** Elaborado pelos autores.

Os passos de instalação detalhados do GitHub são descritos no Apêndice I deste trabalho.

Como mencionado no quadro teórico, neste trabalho foi utilizada a linguagem Java, sendo assim necessária a utilização de uma *Integrated Development Environment* - IDE³² - de apoio. A IDE escolhida foi o Eclipse, pois se trata de uma ferramenta *open source*, muito utilizada no mercado e que permite a escrita de um código mais legível, facilitando tarefas como *debug* e configurações do trabalho.

O Eclipse possui várias ferramentas, dentre elas, pode-se citar o editor de texto, usado não somente para a escrita de códigos em Java, e também a perspectiva de configuração para servidores *web*, utilizada neste trabalho, conforme apresenta a Figura 23. Por meio desta perspectiva, foi configurada a aplicação *container* Tomcat na versão 7.

³¹ Repositório: local cujo desenvolvedor utiliza para armazenar os documentos relacionados ao *software*.

³² IDE: *Integrated Development Environment* - Aplicação contendo uma série de ferramentas para auxiliar no desenvolvimento de *software*.

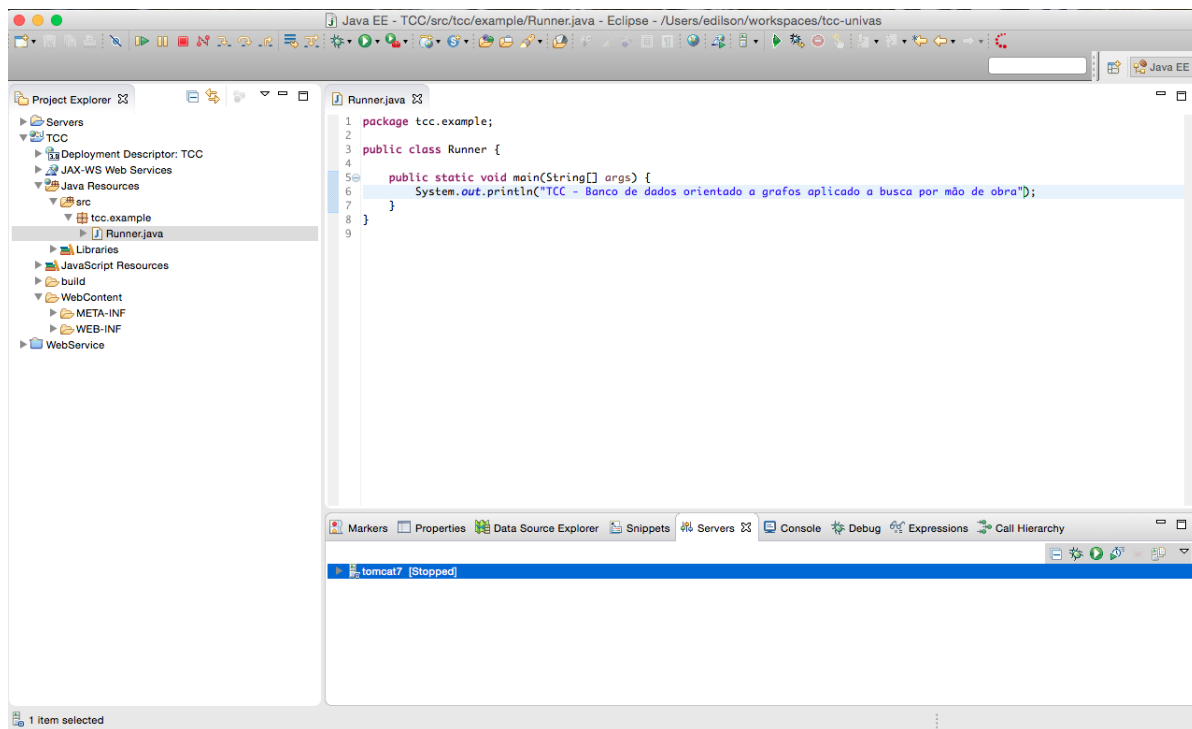


Figura 23 – Ferramentas da IDE Eclipse **Fonte:** Elaborado pelos autores.

O Tomcat desempenhou um papel fundamental na execução desta aplicação, pois serviu como hospedeiro para a aplicação Java desenvolvida neste trabalho.

Os passos de instalação e configuração do Eclipse e do Tomcat são descritos no Apêndice II deste trabalho.

Para a escrita do código relacionado ao HTML, CSS e Javascript, foi utilizado o mesmo editor de texto citado anteriormente.

O trabalho fez uso de um banco de dados orientado a grafos, o Neo4j. A escolha desse banco se deu pela sua simplicidade de instalação, configuração, facilidade de integração com a API *Cypher* e por disponibilizar uma API REST para acesso aos seus dados, conforme descrito no quadro teórico deste trabalho. O Neo4j faz parte do enquadramento de softwares livres, seguindo o conceito *open source*, o que permite ao desenvolvedor utilizá-lo da forma que melhor lhe convier.

Os passos para a instalação do banco de dados Neo4j são detalhados no apêndice tal desta pesquisa.

Posterior à configuração do ambiente, iniciou-se o desenvolvimento propriamente dito, apresentado a seguir.

3.4.3 Desenvolvimento

A princípio, utilizou-se as tecnologias Neo4j, sendo executado de forma *embedded*, Primefaces e JSF. Porém não estava fluindo como o esperado. Outro problema encontrado ao utilizar tais tecnologias foi que tanto a parte cliente (*front end*) quanto a parte servidor (*back end*) se encontravam totalmente acoplados em uma aplicação Java *web*. Por estes motivos decidiu-se mudar algumas das tecnologias utilizadas.

Posterior a esse incidente, passou-se a utilizar então as linguagens HTML 5, CSS 3, Javascript e o *framework* Angular JS para auxiliar no desenvolvimento do *front end*, ao invés de Primefaces e JSF. Para acesso ao banco de dados, lançou-se mão da forma *embedded* e passou-se a utilizar a API REST disponibilizada pelo próprio banco. Tais decisões nos permitiram desacoplar o sistema e manter o *front end* e o *back end* independentes, evitando, assim, que o mesmo problema voltasse a ocorrer.

Como a forma de conexão ao banco de dados foi alterada, houve a necessidade de reescrever a classe responsável por realizar esta conexão, conforme apresenta a Figura 32.

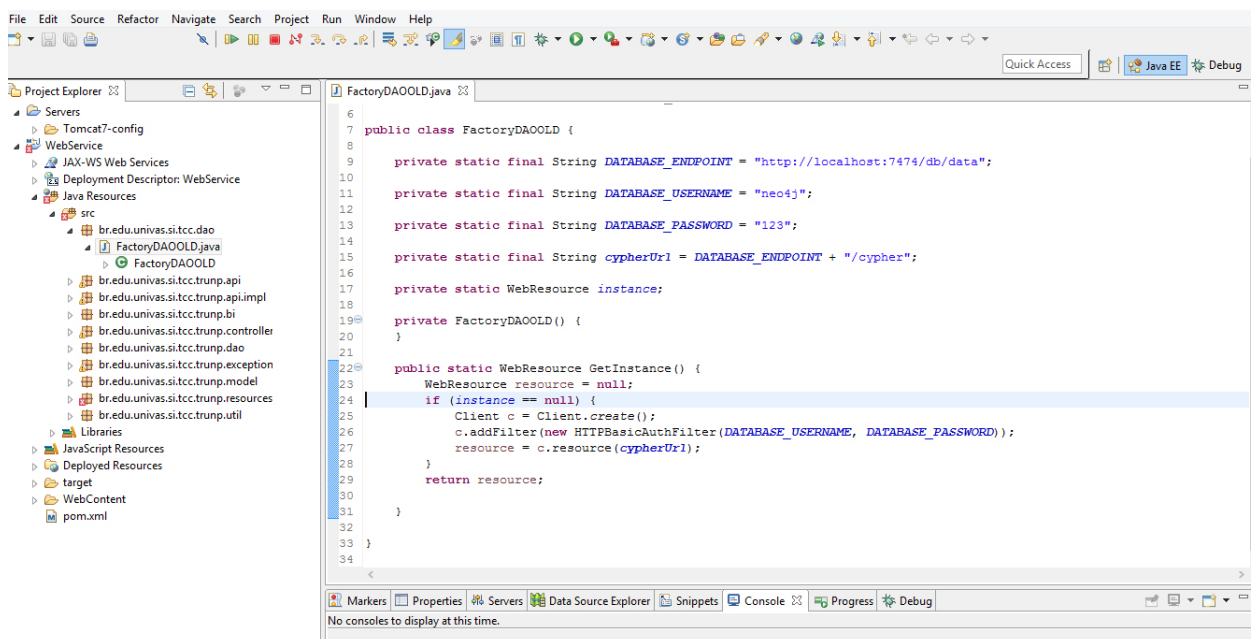


Figura 24 – Código de comunicação com o banco **Fonte:** Elaborado pelos autores.

Após realizar a mudança de tecnologias, foram executados alguns procedimentos para compreender o funcionamento do *web service* REST e em paralelo, foi feito o levantamento dos materiais de referência do *framework* Angular JS. Foi preciso realizar testes para validar a conexão com o banco de dados Neo4j via API REST, fornecida por ele, além de realizados testes funcionais para envio de requisições e recebimento de respostas do *web service* REST,

utilizando o Angular JS. Para validar a conexão ao banco de dados via API REST foi necessário desenvolver algumas consultas em *cypher* como apresenta a Figura 33.

```

159- /**
160-  * Used to get all data of person to show the profile data
161-  *
162-  * @param partnerEmail
163-  * @return
164-  * @throws JSONException
165-  */
166- public JSONArray getPersonData(String partnerEmail) throws JSONException {
167-
168-     WebResource resource = FactoryDAO.GetInstance();
169-
170-     String query = null;
171-     query = "{\nquery\": \" MATCH (partner:Person {email: ' + partnerEmail + '}), \"
172-         + \"(city:City), \"
173-         + \"(company:Company), \"
174-         + \"(partner)-[:LIVES_IN]->(city), \"
175-         + \"(partner)-[:WORKS_IN]->(company) \"
176-         + \"RETURN DISTINCT(name: partner.name, email: partner.email, photo: partner.photo, city: city.name, \" +
177-         + \"company: company.name) as partner; \n}";
178-     ClientResponse responseCreate = resource
179-         .accept(MediaType.APPLICATION_JSON)
180-         .type(MediaType.APPLICATION_JSON).entity(query)
181-         .post(ClientResponse.class);
182-     String resp = responseCreate.getEntity(String.class);
183-
184-     JSONObject json = null;
185-     JSONArray objData = null;
186-     json = new JSONObject(resp);
187-     objData = json.getJSONArray("data");
188-     List<JSONObject> parser = JSONUtil.parseJSONArrayToListJSON(objData);
189-     JSONArray arr = new JSONArray(parser);
190-
191-     return arr;
192- }

```

Figura 25 – Exemplo de consulta usando a API *cypher*. **Fonte:** Elaborado pelos autores.

Ao realizar estes testes, foi constatado que seria necessário desenvolver uma maneira de converter os resultados das buscas realizadas no banco de dados Neo4j que, por padrão, não retorna os resultados no formato JSON comum, portanto, houve a preocupação em tratar estas respostas, a fim de retornar um JSON válido ao usuário que futuramente viria a utilizar a API REST fornecida por este *software*. É possível visualizar este tratamento na Figura 33.

A partir deste ponto, a aplicação estava totalmente desacoplada, sendo necessário realizar uma configuração, a fim de permitir que as requisições enviadas pelo *front end* fossem aceitas pelo *back end*, localizado em outro domínio.

Devido à mudança de tecnologias já comentadas, houve a necessidade de atualizar os diagramas de sequência e de classe, inserindo os contratos de serviços do *web service* REST. Com a definição deste contrato, deu-se início ao desenvolvimento dos casos de uso, identificados na primeira fase do ICONIX.

Posterior à realização dos testes e da escolha definitiva da arquitetura que seria utilizada, iniciou-se a implementação dos casos de uso. O primeiro a ser implementado foi o caso de uso de criação de conta. Para este caso de uso, teve-se o cuidado de criar um mecanismo de criptografia de dados sigilosos, como usuário e senha, visando garantir a segurança da aplicação. Estas informações criptografadas são enviadas a cada requisição e validadas pelo *web service*, sendo atualizadas caso sejam válidas, tornado mais complexo a quebra desta criptografia. Este mecanismo foi desenvolvido com base no sistema de *login* via *token*. Segundo o embasamento usado na criação de contas, deu-se início ao desenvolvimento do sistema de *login* e *logout*, que também utilizam o conceito de criptografia via *token*. A Figura 34 apresenta a página de *login*.

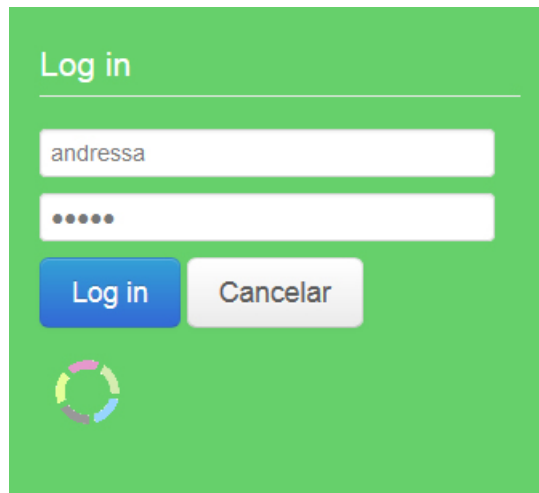


Figura 26 – Tela de login **Fonte:** Elaborado pelos autores.

Com o funcionamento do sistema de *login*, passou-se a desenvolver a página inicial da aplicação, contendo as informações que são restritas ao usuário cadastrado. O sistema apresenta uma página inicial diferente para cada tipo de conta, sendo elas: contratantes, provedores de serviço ou ambos, contendo apenas as informações que são liberadas de acordo com o acesso do usuário, sendo essas informações relatórios, últimas atualizações na rede de parceiros, avaliações de serviços e prováveis parceiros. A página inicial do tipo contratante é apresentada na Figura 35.

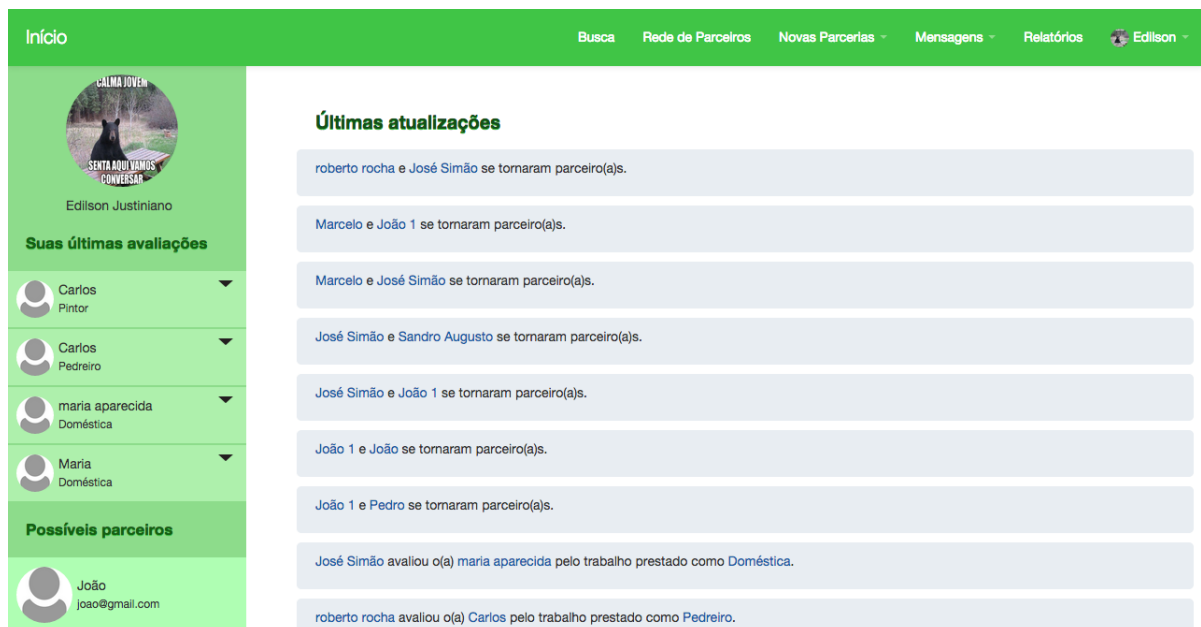


Figura 27 – Página inicial do usuário contratante **Fonte:** Elaborado pelos autores.

O caso de uso localizar parceiros foi desenvolvido após a conclusão do caso de uso criar conta. A lógica deste caso de uso consiste em buscar por possíveis parceiros, com base na rede de parceria do contrante. A Figura 36 apresenta esta busca.

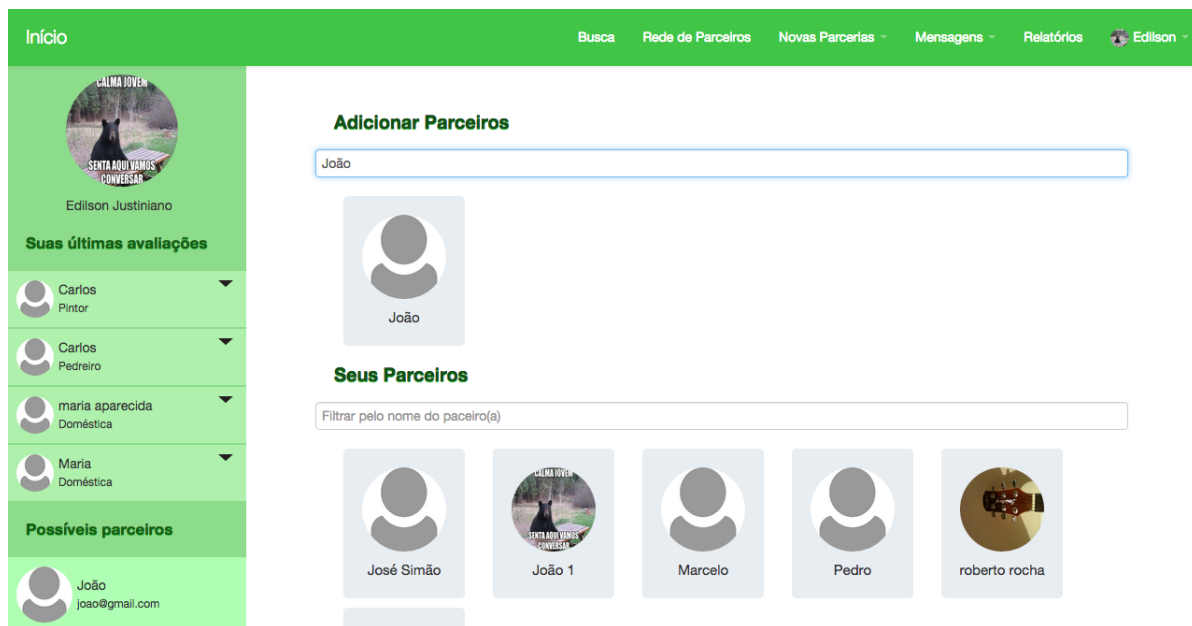


Figura 28 – Página de localização de parceiros **Fonte:** Elaborado pelos autores.

Ainda relacionado ao tipo de conta contratante ou ambos, foi implementado o caso de uso adicionar parceiro, que permite ao usuário convidar um possível parceiro para fazer parte da sua rede. Após a implementação da lógica para adicionar um novo parceiro, houve a necessidade de implementar o serviço de requisições de parcerias, uma vez que não bastava apenas um contratante convidar outro para se tornarem parceiros, mas sim que o contratante convidado aceitasse sua solicitação de parceria, para assim se tornarem parceiros. Visando disponibilizar estas solicitações de forma agradável ao usuário, foi desenvolvida uma funcionalidade para que o usuário pudesse aceitar ou rejeitar a solicitação enviada à ele.

Após realizada a implementação do caso de uso adicionar parceiro, houve a necessidade de desenvolver a busca por todos os usuários que possuíam o tipo de conta contratante ou ambos e que possuíam um relacionamento de parceria com o usuário autenticado no sistema, além da funcionalidade de localizar novos parceiros, baseando-se na localização da empresa na qual o usuário trabalha e na cidade onde ele vive, sempre ordenando os resultados por meio da quantidade de parceiros em comum.

O caso de uso gerenciar serviços foi implementado em sequência, abrangendo as principais funcionalidades de gerenciamento: cadastrar e adicionar um novo serviço ao usuário, cujo tipo de conta é provedor de serviços, listar os serviços atribuídos a ele, e remover serviços quando necessário. Visando melhorar a usabilidade, foi implementado um mecanismo de busca, que permitiu filtrar os resultados por meio de um campo que possui a função auto completar, evitando assim, possíveis erros e diminuindo o tempo gasto pelo usuário para adicionar o serviço. A função realiza a busca em uma lista de serviços anteriormente cadastrados, no

entanto, caso não haja o serviço solicitado, o usuário tem a liberdade de cadastrá-lo e atribuí-lo a si mesmo.

A partir deste ponto, foi possível iniciar o desenvolvimento do caso de uso "localizar mão de obra", uma vez que, este caso de uso dependia diretamente das implementações das funcionalidades adicionar parceiros para os usuários contratantes e adicionar serviços aos provedores de serviço. Para facilitar a localização e deixar o *software* mais usual, esta busca se baseia inicialmente no serviço buscado pelo usuário, sendo posteriormente modificada para também levar em consideração a funcionalidade avaliar serviço que foi implementada paralelamente. A avaliação de serviço permite ao contratante dar uma nota ao serviço que foi prestado a ele. Com estas informações foi possível desenvolver uma busca que levaria em consideração, além destas informações, a rede de parceiros do usuário contratante, a fim de lhe apresentar as melhores opções possíveis.

A fim de abranger a busca e possibilitar que novos prestadores de serviços sejam avaliados pelos contratantes, a consulta que antes apresentava apenas provedores de serviços que possuíam avaliações, sendo elas, positivas ou negativas, foi ampliada, possibilitando que profissionais não avaliados também entrassem na lista de prováveis provedores de serviços.

Para auxiliar na tomada de decisão do usuário contratante, foi implementada uma funcionalidade que realiza o cálculo da média de avaliações de um serviço prestado por um profissional temporário, tendo como base as avaliações já realizadas pela rede de parceiros do usuário autenticado, da empresa onde ele trabalha e da cidade onde vive, oferecendo assim uma forma simples de obter acesso a qualidade do serviço prestado.

Após realizada todas as implementações já descritas, houve a preocupação de desenvolver uma interface, que além de amigável fosse prática ao usuário, desta forma, foi disponibilizada algumas informações relevantes, que auxiliam o usuário a compreender o que está ocorrendo em sua rede de parceria. Como exemplo é possível citar a lista de parceiros em comum entre o usuário autenticado no sistema e um determinado contratante por meio da página de perfil dele.

A fim de agregar mais funcionalidades para o usuário provedor de serviços, foi criado na página inicial do *software* uma funcionalidade que visa apresentar algumas dicas interessantes que contribui com a sua imagem perante ao *software*, levando-o assim a obter uma quantidade maior de oportunidades de trabalho.

Para finalizar o desenvolvimento será necessário desenvolver alguns gráficos que apresentem ao usuário informações a respeito da qualidade do serviço prestado pelo provedor de serviços, comparando-os com os demais prestadores, porém estes gráficos ainda não foram

implementados até o momento.

4 DISCUSSÃO DOS RESULTADOS

Neste capítulo serão apresentados e discutidos os resultados obtidos por esta pesquisa e desenvolvimento do *software*, apresentando os seus pontos positivos e negativos.

Inicialmente foram utilizadas as tecnologias Java, Primefaces e JSF para desenvolver este trabalho, no entanto, viu-se a necessidade de alterar algumas destas tecnologias, visando agilizar o processo de desenvolvimento. Desta forma, passou-se a utilizar HTML, CSS, Javascript em conjunto com o *framework* Angular JS.

A mudança de tecnologias trouxe como benefício o desacoplamento das partes cliente (*front-end*) e servidor (*back end*), não sendo mais necessário recompilar, construir e publicar a aplicação no servidor *web*, como era feito até então a cada alteração.

Após esta mudança, notou-se que a forma como o banco de dados era acessado poderia ser ajustada, a fim de seguir a mesma ideia proposta pela troca de tecnologias já mencionadas, uma vez que o banco de dados Neo4j permite duas maneiras de conexão, sendo elas: *embedded* e por meio da API REST.(XXXXX).Desta forma, passou-se a utilizar a API REST ao invés da forma *embedded*, utilizada até então.

Posterior a estas mudanças, deu-se continuidade no desenvolvimento da aplicação, onde os seus principais resultados da são apresentados a seguir.

4.1 Neo4j

Para desenvolver este trabalho foi necessário utilizar um banco de dados orientado a grafo, a fim de aplicar o mesmo conceito utilizado pelas redes sociais, para que o objetivo proposto por este trabalho fosse alcançado. Foi realizada um levantamento para validar as opções de banco de dados orientado a grafos disponíveis, entretanto, o Neo4j demonstrou-se superior aos demais pela sua simplicidade de instalação, utilização, configuração e por possuir uma linguagem para realizar consultas que se assemelha muito à linguagem humana, a API *Cypher*.

No início do desenvolvimento deste projeto esperava-se que o Neo4j fosse utilizado por meio da forma *embedded*, porém, devido a dificuldades que esta forma apresentou, uma vez que, não era possível conectar-se à base de dados pela aplicação em desenvolvimento e pelo sistema de gerenciamento do Neo4j simultaneamente, impedindo que as alterações no banco de

dados fossem validadas pelo sistema de gerenciamento no momento em que as alterações eram realizadas no *software* desenvolvido neste trabalho, somado ao fato que foi necessário alterar as tecnologias que inicialmente esperavam ser utilizadas no *front-end*, houve-se a necessidade de lançar mão da forma *server* e utilizar a API REST que o próprio Neo4j disponibiliza e abrir mão antiga maneira proposta inicialmente.

Após estas definições foram criados testes para validar a conexão ao banco de dados, a fim de constatar que a conexão com o Neo4j via API REST em conjunto com a também API *Cypher* havia sido realizada de forma correta.

Na próxima seção serão apresentados os resultados obtidos no desenvolvimento da aplicação *web*.

4.2 Aplicação web

A linguagem Java foi proposta desde o princípio para ser utilizada como linguagem padrão para o *back-end*, porém devido aos problemas já citados relacionados a troca de tecnologias, foi proposto realizar a troca da linguagem Java por PHP, pois, apesar dos autores deste trabalho obterem conhecimento nesta linguagem, o PHP não foi abordado no curso o que seria muito bem aceito, entretanto, foi realizado alguns testes e constatou-se que, esta troca não seria conveniente devido a escassez de documentação que esta nova linguagem apresenta se tratando da comunicação com o Neo4j via API REST. Sendo assim, optou-se por continuar a utilizar o Java, o que evitou o despendimento de tempo para resolver problemas com falta de documentação.

Ainda após as trocas de tecnologias e da forma de conectar-se ao banco de dados, foi necessário alterar as classes responsáveis pela conexão com o Neo4j e também as classes que realizam as operações no banco.

5 CONCLUSÃO

A conclusão deste trabalho é ...

Assim conclui-se que ...

REFERÊNCIAS

- APACHE. *Apache Tomcat*. 2015. Disponível em: <<http://tomcat.apache.org/index.html>>. Acesso em 15 de janeiro de 2015.
- AZEVEDO, M. E. C. V. de. *Quantas fases tem o ICONIX?* 2010. <<http://oengenheirodesoftware.blogspot.com.br/2010/11/quantas-fases-tem-o-iconix.html>>. Acesso em: 22/07/2015.
- BARBOSA, K. *Por que as pessoas usam as redes sociais?* 2011. Disponível em: <<http://super.abril.com.br/blogs/tendencias/por-que-as-pessoas-usam-as-redes-sociais/>>. Acesso em 28 de dezembro de 2014.
- BASHMAN, B.; SIERRA, K.; BATES, B. *Use a Cabeça! Servlets e JSP*. [S.l.]: Alta Books, 2010.
- BONDY, J. A.; MURTY, U. S. R. *Graph Theory with applications*. 1976.
- BRITTAIN, J.; DARWIN, I. F. *Tomcat The Definitive Guide*. 2. ed. [S.l.: s.n.], 2007.
- BRUGGEN, R. V. *Learning Neo4j*. [S.l.]: Packt, 2014.
- DATE, C. J. *Introdução a sistemas de bancos de dados*. 8. ed. [S.l.]: Elsevier, 2003.
- ELMASRI, R.; NAVATHE, S. B. *Sistemas de Banco de Dados*. 1. ed. [S.l.]: Pearson, 2005.
- ERL, T. et al. *SOA with REST: principles, patterns & constraints for building enterprise solutions with REST*. [S.l.]: Novatec, 2012.
- FARIA, M. A. de. *ASPECTJ: Programação orientada a aspecto em Java*. [S.l.: s.n.], 2008.
- FLANAGAN, D. *Javascript: The Definitive Guide*. 6. ed. [S.l.]: O' Reilly, 2011.
- FONSECA, J. J. S. *Metodologia da pesquisa Científica*. [S.l.]: UEC, 2002.
- FRANK, D. R.; SEIBT, L. *Javascript*. 2002.
- GIL, A. C. *Métodos e técnicas de pesquisa social*. 5. ed. [S.l.]: Atlas, 1999.
- GIL, A. C. *Como elaborar projetos de pesquisa*. 4. ed. [S.l.]: Atlas, 2007.
- GREEN, B.; SESHADRI, S. *Javascript: The Definitive Guide*. [S.l.]: O' Reilly, 2013.
- HARJU, T. *Graph theory*. 2014.
- JUNID, S. A. M. A. et al. *Potential of graph theory algorithm approach for dna sequence alignment and comparison*. 2012.
- KELLER, R. M. *Acyclic Graph*. 2015. <<http://www.cs.hmc.edu/~keller/courses/cs60/s98/examples/acyclic/>>. Acesso em: 20 de Abril de 2015.
- KOZLOWSKI, P.; DARWIN, P. B. *Mastering Web Application Development with AngularJS*. [S.l.]: Packt Publishing, 2013.

- LAURIE, B.; LAURIE, P. *Apache: The Definitive Guide*. [S.l.]: RepKover, 2003.
- MARCONI, M. A.; LAKATOS, E. M. *Metodologia do trabalho científico*. 7. ed. [S.l.]: Atlas, 2009.
- MARZULLO, F. P. *SOA na prática: inovando seu negócio por meio de soluções orientadas a serviços*. [S.l.]: ServiceTech Press, 2013.
- MATTHEW, N.; STONES, R. *Beginnig databases with postgresql - from novice to professional*. 2005.
- NEO4J. *The Neo4j Manual*. [S.l.: s.n.], 2013.
- ORACLE. *About the Java Technology*. 2010. Disponível em: <<http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>>. Acesso em 28 dez. 2014.
- ORACLE. *Polymorphism*. 2015. <<https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html>>. Acesso em: 22/07/2015.
- ORACLE. *What are RESTful Web Services?* 2015. <<https://docs.oracle.com/jvae/6/tutorial/doc/gijqy.html>>. Acesso em 23/07/2015.
- PAOLETTI, T. *Leonard Euler's Solution to the Konigsberg Bridge Problem*. 2006. <<http://www.maa.org/publications/periodicals/convergence/leonard-eulers-solution-to-the-konigsberg-bridge-problem>>. Acesso em 26 de fevereiro de 2015.
- PÁDUA, E. M. M. de. *Metodologia da Pesquisa: Abordagem Teorico-Pratica*. 16. ed. [S.l.]: Papirus, 2007.
- PENHA, A. L. da S.; CARVALHO, W. *Sistema de recomendação de filmes utilizando graph database*. [S.l.: s.n.], 2013.
- ROBINSON, I.; WEBBER, J.; EIFREM, E. *Graph Databases*. [S.l.]: O'Reilly, 2013.
- ROCHA, R. R. *Algoritmos de particionamento e banco de dados orientado a grafos*. 2013.
- RODRIGUEZ, A. *RESTful Web services: The basics*. 2008. <<http://www.ibm.com/developerworks/library/ws-restful/>>. Acesso em 23/07/2015.
- ROSENBERG, D.; SCOTT, K. *Use Case Driven Object Modeling With UML A Pratical Approach*. 1. ed. [S.l.: s.n.], 1999.
- ROSENBERG, D.; STEPHENS, M. *Use Case Driven Object Modeling With UML Theory and Practice*. 1. ed. [S.l.: s.n.], 2007.
- ROSENBERG, D.; STEPHENS, M.; COLLINS-COPE, M. *Agile Development with ICONIX Process: People, Process, and Pragmatism*. 2005.
- RUOHONEN, K. *Graph Theory*. 2013.
- SADALAGE, P. J.; FOWLER, M. *NoSQL Distilled - A Brief Guide to the Emerging World of Polyglot Persistence*. [S.l.: s.n.], 2013.
- SCHILDT, H. *Java: The Complete Reference*. 7. ed. [S.l.: s.n.], 2007.

SETZER, V. W.; SILVA, F. S. C. da. ***Banco de dados: Aprenda o que são, melhore seu conhecimento, construa os seus.*** [S.l.]: Edgard Blucher, 2005.

SILVA, A.; VIDEIRA, C. ***UML, Metodologias e Ferramentas Case.*** 1. ed. [S.l.: s.n.], 2001.

SILVA, A. C. da; PIRES, J. P. ***Banco de dados relacional versus banco de dados orientado a grafos aplicados a redes sociais.*** [S.l.: s.n.], 2013.

SILVA, M. S. ***CSS 3: desenvolva aplicações web profissionais com o uso dos poderosos recursos de estilização das CSS 3.*** [S.l.]: Novatec, 2011.

SILVA, M. S. ***HTML 5.*** [S.l.: s.n.], 2011.

VUKOTIC, A.; GOODWILL, J. ***Apache Tomcat 7.*** [S.l.]: Apress, 2011.

W3C. ***The Basics of HTML.*** 2015. <https://docs.webplatform.org/wiki/guides/the_basics_of_html>. Acesso em 24/07/2015.

W3C. ***What is CSS?*** 2015. <<http://www.w3.org/Style/CSS/>>. Acesso em 24/07/2015.

Apêndices

TÍTULO DO APÊNDICE I

Aqui deve conter o texto do Apêndice 5. Na Figura 29 é ilustrada a primeira tela deste processo.

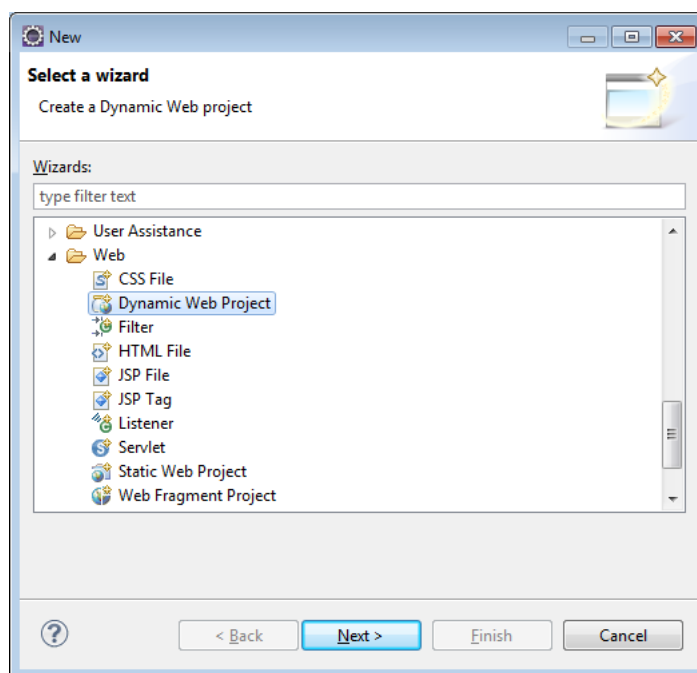


Figura 29 – Outra imagem. **Fonte:** Elaborado pelos autores

Após a seleção do tipo de projeto ...

TÍTULO DO APÊNDICE 2

Primeira seção do apêndice 2

Neste apêndice é mostrado ... de acordo com a Figura 30 é ilustrada a primeira tela deste processo.

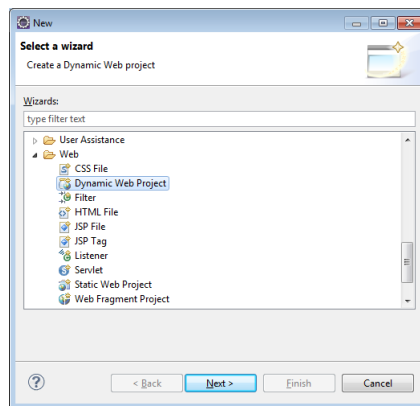


Figura 30 – Outra imagem ainda. **Fonte:** Elaborado pelos autores

Segunda seção do apêndice 2

Continuando ... na figura Figura 31 é mostrado um exemplo de XML.

```
<project>
...
<dependencies>
...
<dependency>
  <groupId>org.neo4j</groupId>
  <artifactId>neo4j</artifactId>
  <version>1.9.4</version>
</dependency>
...
</dependencies>
...
</project>
```

Figura 31 – Exemplo de código XML. **Fonte:** Elaborado pelos autores

ANEXO I