INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
CAMPUS DE CAJAZEIRAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARCUS VINÍCIUS DE VASCONCELOS FILHO

PINCH NOTES: FERRAMENTA PARA CRIAÇÃO DE MAPAS MENTAIS ORIENTADA A ANOTAÇÕES

CAJAZEIRAS – PB

2019

MARCUS VINÍCIUS DE VASCONCELOS FILHO

PINCH NOTES: FERRAMENTA PARA CRIAÇÃO DE MAPAS MENTAIS ORIENTADA A ANOTAÇÕES

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, para obtenção de título em Tecnólogo de Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Esp. **Diogo Dantas Moreira**.

CAJAZEIRAS – PB

2019

IFPB / Campus Cajazeiras
Coordenação de Biblioteca Prof. Ribamar da Silva
Catalogação na fonte: Daniel Andrade CRB-15/593

V331p

Vasconcelos Filho, Marcus Vinícius de.

Pinch Notes: ferramenta para criação de mapas mentais orientada a anotações / Marcus Vinícius de Vasconcelos Filho; orientador Diogo Dantas Moreira.-

62 f.: il.

Orientador: Diogo Dantas Moreira. TCC (Tecnólogo em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Cajazeiras, 2019.

1. Aplicações Web 2. Mapas Mentais I. Título.

CDU 004.4(0.067)

MARCUS VINÍCIUS DE VASCONCELOS FILHO

PINCH NOTES: FERRAMENTA PARA CRIAÇÃO DE MAPAS MENTAIS ORIENTADA A ANOTAÇÕES

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, apreciado pela Banca Examinadora composta pelos seguintes membros:

Aprovado em: _____/_____/ **2019**

BANCA EXAMINADORA

_____

Prof. Esp. *Diogo Dantas Moreira* (Orientador)
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba

_____

Prof. Ms. *Francisco Paulo de Freitas Neto*
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba

_____

Prof. Ms. *George Candeia de Sousa Medeiros*
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba

AGRADECIMENTOS

A ideia do projeto *Pinch Notes* surgiu ainda como uma semente nos semestres iniciais do curso Análise e Desenvolvimento de Sistemas do IFPB.

Em decorrência do ambiente criativo e inovador que a instituição favorece, várias aplicações foram pensadas. Algumas se tornaram projetos pessoais, outras foram desenvolvidas como atividades disciplinares. O *Pinch Notes* surgiu, assim, a partir de uma dessas atividades.

Mesmo sem envolvimento direto com este trabalho, os professores Aristofanio Garcia e Ricardo Job tiveram importantes papéis para o desenvolvimento do projeto, em razão das suas aulas ricas em conhecimentos e práticas; considerando que as suas metodologias atentam mais para o aprendizado real do conhecimento do que para formalismos acadêmicos, e o verdadeiro desejo de conhecer e transmitir conhecimento inerente em suas personalidades.

Para ser utilizada como trabalho de conclusão de curso, a ideia da ferramenta de criação de mapas mentais orientada a anotações, foi avaliada e aceita pelos professores envolvidos no projeto, o orientador Diogo Dantas Moreira, e os participantes da Banca Examinadora, George Candeia de Sousa Medeiros e Francisco Paulo de Freitas Neto. Sem suas orientações e sugestões, o projeto *Pinch Notes* não teria sido concluído como um trabalho de curso. Por isso, agradeço-lhes as contribuições ofertadas nessa etapa da minha trajetória acadêmica.

Agradeço, igualmente, à minha família. Em especial, à minha mãe, Hercília Maria Fernandes, que motivou e ajudou imensamente na normalização do documento escrito, mesmo nos momentos mais difíceis. Com seu incentivo, consegui concluir a versão final do trabalho que, durante sua trajetória, se deparou com vários obstáculos oriundos da minha vida profissional e pessoal. E ao meu irmão Pedro Arthur, por me fazer parte das suas divagações tecnológicas, me mantendo alinhado à área de tecnologia da informação e sempre me ensinando cada vez mais.

Por fim, agradeço à comunidade que mantém o desenvolvimento da linguagem *javascript* ativo e efervescente, o que me possibilitou a entrada no mercado de trabalho através das empresas *Ruppells Griffon* e *Integritas*.

RESUMO

O Trabalho de Conclusão de Curso intitulado “Pinch Notes: ferramenta para criação de mapas mentais orientada a anotações” consiste em uma discussão teórica a respeito da utilidade dos mapas mentais e como desenvolver uma aplicação que facilite o estudo através de mapas mentais por estudantes que não conhecem a técnica. Nesse sentido, o objetivo geral do trabalho correspondeu a desenvolver uma aplicação *web* que auxilie o processo de aprendizagem através da criação de mapas mentais orientada a anotações lineares. A abordagem teórica, nesse sentido, guiou-se no entendimento da noção de “pensamento radial”, proposta por Buzan (1996). Assim sendo, a metodologia e os procedimentos metodológicos pertinentes envolveram a pesquisa bibliográfica; a pesquisa por aplicações existentes; a prototipagem; a modelagem de dados; a implementação do modo área livre; a implementação do modo mapa mental; o gerenciador; e, a escrita do documento de TCC. Cumpridas todas as etapas de investigação e construção do projeto, conclui-se que o desenvolvimento do *Pinch Notes* demonstrou que o objetivo da ferramenta, no sentido de diminuir a lacuna entre o estudante comum e a técnica de mapas mentais, foi atingido; pois, para criar mapas mentais na aplicação, o estudante não precisa mudar sua forma de organizar ideias, basta inserir textos e imagens que serão processados para a construção automática do mapa mental.

Palavras-chave: Aprendizagem. Mapa mental. Anotações lineares. Grafos.Javascript. Nodejs. Neo4j. Aplicação Web.

ABSTRACT

The Graduation Work entitled "Pinch Notes: linear notes oriented tool for creating mind maps" is composed by a theoretical discussion about the utility of mind maps and how to develop an application which helps the learning process through mind mapping by students who do not know this approach. Thus, the project's main goal was to develop a web application which helps the students even though they don't know or find difficult to apply mind mapping techniques based on the “Radial Thinking” theory proposed by Buzan (1996). To achieve this, the methodology encompassed the bibliographic research; available applications on the market; prototyping; data modeling; development of frontend and backend modules through reusable components; and the writing of this document. Through the journey across all the steps towards the delivery of this project, it was obtained results that provides a promising insight according to the effective way of diminishing the gap between the usual student and its understanding of mind mapping techniques, because the developed application allows users to create their mind maps only by the insertion of linear notes, full texts and images, as the application generates the full map automatically.

Keywords: Learning. Mind Map. Linear Notes. Graph. Javascript. Nodejs. Neo4j. Web Application.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ANNE - **Angular, Node, Neo4j, Express.**

API - **Application Program Interface.**

ACID - **Atomicity, Consistency, Isolation, Durability.**

CPU - **Central Processing Unit.**

CRUD - **Create, Read, Update, Delete.**

CSS - **Cascading Style Sheets.**

HTML - **Hyper Text Markup Language.**

HTTP - **Hyper Text Transfer Protocol.**

JSON - **Javascript Object Notation.**

JWT - **Json Web Signature.**

MVC - **Model View Controller.**

NOSQL - **Not Only SQL.**

NPM - **Node Package Manager.**

PHP - **Hypertext Preprocessor.**

PNG - **Portable Network Graphics.**

SQL - **Structured Query Language.**

SVG - **Scalable Vector Graphics.**

TCC - **Trabalho de Conclusão de Curso.**

URL - **Uniform Resource Locator.**

WYSIWYG - **What You See Is What You Get.**

SUMÁRIO

1 INTRODUÇÃO

O aprendizado é uma atividade primordial aos sujeitos pensantes. Apresenta várias implicações na formação do ser humano como agente integrante de uma sociedade constituída por operações das quais seus participantes executam, como as atividades escolares, profissionais, religiosas, civis, culturais e sociais.

A tarefa de aprender está na experiência de integrar novas informações na própria percepção do ser. Explorar novos caminhos através da reflexão dos dados desconhecidos ou não compreendidos; criar ramificações do conhecimento a partir de percepções compartilhadas por terceiros; e transmitir novos entendimentos ou descobertas para outras consciências que, por sua vez, realizam o ciclo evolutivo da construção do conhecimento.

O processo de aprendizagem, porém, não é uma prática trivial e simples. Alguns sujeitos a dominam com facilidade, pois, de algum modo, aprenderam a aprender. Outros indivíduos, por outro lado, precisam empreender maior esforço para assimilar bem um conteúdo e toda sua faixa de significados e conexões. Assim sendo, quanto às pessoas que possuem dificuldade em assimilar novos conhecimentos, provavelmente estão executando um processo de estudo ineficiente e inapropriado.

Nesse sentido, o estudo realizado e documentado pelos autores John Dunlosky, Katherine A. Rawson, Elizabeth J. Marsh, Mitchell J. Nathan & Daniel T. Willingham, no artigo "*Improving Students' Learning With Effective Learning Techniques: Promising Directions From Cognitive and Educational Psychology*" (2013), confirma essa direção de entendimento; ao analisar a aplicação isolada das estratégias de sumarização, notas lineares, imagens, releitura e exercícios. O artigo classifica, através de análise estatística, que a estratégia mais eficiente a curto e longo prazo é a aplicação de testes e exercícios; considerando que a atividade de reflexão e associação é mais estimulada por parte do estudante.

Dado um processo de aprendizagem composto, basicamente, por leitura excessiva, anotações lineares entediantes, memorização acompanhada por pouca

ou nenhuma reflexão e associação, é possível inferir que a razão da dificuldade dos estudantes que aplicam esses métodos se associa à falta da prática da reflexão.

Pensando neste problema enfrentado por muitos estudantes, Buzan (1996), o criador da metodologia de aprendizado através de criação e leitura de mapas mentais, dedicou sua vida ao estudo do comportamento dos estudantes e as técnicas utilizadas por eles. Em seu estudo, reconheceu que as técnicas comumente empregadas utilizam muito pouco do potencial do cérebro humano e, por isso, desenvolveu uma nova técnica de aprendizagem, a partir de observações e pesquisas sobre como a atividade cerebral influencia e é influenciada pelo aprendizado. Buzan (1996) nomeou essa técnica de *Radial Thinking* ou Pensamento Radial, e a ferramenta para sua aplicação como *Mind Map* ou Mapa Mental.

Segundo o livro *Mapeamento Mental*: guia passo a passo para iniciantes em criação de mapas mentais (THE BLOCKHEAD, 2016), é possível se conceber o mapa mental como uma estrutura semelhante à *web* composta por palavras, figuras ou imagens e linhas criadas com o propósito único de organização visual da informação. O conceito ou tópico principal é situado na parte central dessa estrutura, normalmente descrito em folha em branco e em formato paisagem. A partir do assunto principal, são delineados outros assuntos relevantes. Palavras e ideias sustentam as concepções, que são largamente conectadas aos principais, enquanto que tópicos de menor relevância vão se agrupando até alcançar a última ideia.

Buzan propôs uma metodologia de aprendizagem bastante relevante, ao identificar um processo de estudo melhor adaptado à natureza da atividade cerebral, porém, considerando os problemas do mundo atual altamente informatizado, onde o conhecimento evolui em uma velocidade nunca antes constatada na história oficial, entende-se que os *softwares* de mapas mentais avaliados não acompanharam as exigências do perfil moderno do estudante comum.

Essas exigências se dão pela imposição de uma sociedade informatizada e interconectada, sendo elas consequências de vários fatores, entre eles: o tempo cada vez mais limitado e mal aproveitado devido à quantidade excedente de informação distrativa; o fato de que a substituição do papel, caneta e livros por *softwares*, computadores e livros digitais, respectivamente, retirou certas vantagens

oferecidas pelos mapas mentais feitos de modo rudimentar, como, por exemplo, a facilidade e a criatividade; o aumento da complexidade dos assuntos estudados, sendo estes mal representados por *softwares* de mapas mentais que se limitam à disposição de “nós” formados por palavras-chaves, o que acarreta na dificuldade da leitura, compreensão e aprendizado de assuntos representados por mapas criados por terceiros, comumente contendo seu conteúdo organizado de maneira caótica; e, por último, o simples fato de que anotações lineares são muito bem integradas à tecnologia de informação, em razão da grande facilidade e poder de edição de textos oferecidos por aplicações de edição e visualização de textos – o que diminui o interesse de aprender e usar *softwares* de mapa mental.

Tendo em vista o exposto, se propõe realizar, neste trabalho, uma análise das exigências não satisfeitas por *softwares* de mapas mentais, de modo a identificar maneiras que possam agregar, aos mapas mentais, as metodologias comuns de estudo, oferecendo maior liberdade, criatividade, simplicidade e compatibilidade com anotações lineares, ao uso de mapas mentais através de *software*. Sendo assim, através dessa análise, deverá ser projetada e implementada uma aplicação *web* que permita a hospedagem e a criação de mapas mentais.

1.1 MOTIVAÇÃO

A maioria das pessoas não utiliza mapas mentais como auxílio para o aprendizado. O uso de mapas mentais, porém, para questões específicas é bem conhecido. Por exemplo, programadores e analistas de sistema usam diagramas de classes para representar os detalhes estruturais de uma solução em *software*. O que é um diagrama de classes senão um mapa mental com estrutura e símbolos específicos para atender as complexidades de um projeto de *software*? Se programadores usam diagramas de classe para entender sistemas complexos, por que estudantes não usam mapas mentais para entender assuntos que talvez sejam bem mais complexos?

O site MINDMEISTER – Uma solução online para criação de mapas mentais – publicou em seu blog uma lista de razões pelas quais a maioria das pessoas não

quer fazer uso de mapas mentais. Entre as razões citadas no artigo, as que chamam mais atenção são: i) Mapeamento mental requer mais tempo que anotações lineares; e, ii) As pessoas estão acostumadas a raciocinar linearmente.

Em relação ao mapeamento mental feito com canetas e papel, a primeira razão é equivocada, já que um mapa mental deve apresentar, pelo menos, os pontos importantes representados por palavras-chaves ligados ao tema central, a sua construção deve demandar muito menos tempo que uma anotação tradicional.

A respeito do mapeamento mental feito através de *softwares*, a primeira razão apresenta fundamento, haja vista os *softwares* de construção de mapas mentais existentes não oferecerem controles simples e intuitivos para criação e manipulação dos elementos gráficos do mapa. O modelo de usabilidade mais adotado por eles é através de menus extensos combinados aos movimentos e *clicks* no *mouse* ou atalhos de teclado, o que resulta no comportamento indesejado observado pelos autores Chick, Plimmer & Hosking:

> Embora essas ferramentas de mapas mentais possuam muitas funcionalidades, muito tempo é consumido com operações de movimentação e inserção de nós e conectores de barras de menu antes mesmo do usuário poder preencher os nós com informação e construir um diagrama compreensível (CHICK; PLIMMER; HOSKING, 2007, p. 196, tradução nossa).

Segundo Faste & Lin (2012, p. 1020), a usabilidade complexa é o preço a ser pago por ferramentas orientadas à construção gráfica de mapas mentais, porque à medida que uma vasta coleção de funcionalidades simples ou complexas é fornecida por esses *softwares,* suas interfaces com o usuário aumentam em termos de complexidade, exigindo que tempo de aprendizado seja gasto e dificultando a interação e o fluxo de trabalho do usuário.

Uma consequência da discrepância entre a complexidade de interface e o escopo de funcionalidades enfrentado por softwares de mapas mentais orientados à construção gráfica dos elementos, apontada por Faste & Lin (2012, p. 1020), é que nenhum dos *softwares* testados por eles apresentaram, simultaneamente, bons resultados nas operações de legibilidade, customização, e controles de contexto.

Sabendo que a maioria das pessoas prefere continuar a utilizar notas lineares em suas anotações, mesmo reconhecendo os benefícios da utilização de mapas mentais, é surpreendente o fato de que não exista uma única aplicação para criação de mapas mentais orientada a anotações eficientes e flexíveis para estudantes – Até o momento, a ferramenta TEXT2MINDMAP é a única orientada a anotações. Entretanto, pelo fato de exigir a indentação em cada elemento textual tratado como sub tópico de algum outro, não funciona bem para textos complexos.

Dessa maneira, é importante conceituar o que deve ser entendido por criação de mapa mental orientada a anotações. A ideia é permitir que o usuário escreva seus textos sem precisar se preocupar com formato, e, à medida em que ele sentir que conceitos, palavras ou ideias referentes ao assunto do texto necessitam de explicação, basta selecionar o trecho de texto desejado, realçá-lo e escrever os detalhes necessários em um contexto exclusivo ao respectivo elemento textual a ser detalhado. O tal contexto exclusivo poderia ser exibido gradativamente a partir da operação de *zoom* sobre o realce de texto na qual o *mouse* está posicionado – Essa seria uma alternativa à necessidade de indentar o texto que ocorre no TEXT2MINDMAP. Paralelamente ao processo de escrita do texto, um algoritmo é encarregado de identificar as relações entre elementos textuais e seus contextos exclusivos para computar os relacionamentos do mapa a ser gerado – resultando na criação e exibição de um grafo. Assim, a representação do mapa mental seria gerada de maneira transparente ao usuário.

De imediato, a criação de mapa mental orientada a anotações eliminaria o problema da distração, então apontado pelos autores Chick, Plimmer & Hosking. Mediante suas palavras: “Pesquisas na área de design sugerem que usuários se distraem ao manipular a interface gráfica da ferramenta em vez de se concentrarem no processo de solução do problema” (CHICK, PLIMMER; HOSKING, 2007 p. 196, tradução nossa).

Além de resolver os problemas de usabilidade comuns em ferramentas de criação de mapas mentais, a abordagem orientada a anotações servirá para que o compartilhamento de mapas mentais com outras pessoas, de fato, tenha fundamento, já que elas poderão compreender o que significa os “nós” e seus

relacionamentos da representação gráfica do mapa a partir das anotações que os deram origem. Poderá ser útil, também, para quem não tem a experiência de estudar através de mapas mentais, posto não ser necessário mudar seu estilo linear de pensar, sendo preservado da tarefa de criar a representação gráfica do mapa mental, e presenteado com o mapa construído, podendo, enfim, manter o foco somente no seu conteúdo.

1.2 OBJETIVOS

1.2.1 Geral

Desenvolver uma aplicação *web* que auxilie o processo de aprendizagem através da criação de mapas mentais orientada a anotações lineares, caracterizando um sistema conveniente à maioria das pessoas, formado pela simbiose entre notas lineares e notas não lineares para obtenção dos melhores benefícios de ambas as metodologias.

1.2.2 Específicos

- Possibilitar que a aplicação *web PinchNotes* disponibilize dois modos de edição: o modo “área livre” e o “mapa mental”. Assim, a construção do mapa mental é feita através da edição na área livre, na qual o usuário poderá inserir textos de qualquer natureza e relacioná-los através de realces coloridos com objetivo de relacionar palavras-chave ou trechos importantes a outros textos de outras partes do mapa. O relacionamento através de realces é utilizado pelo motor da aplicação para construir a visualização gráfica do mapa mental. Essa estratégia dual (área livre/mapa) permite que o usuário entenda o que cada “nó” representa, já que ele poderá ler os textos que deram origem a cada “nó” do mapa;

- Disponibilizar um sistema de busca no mapa que torne possível procurar por partes do mapa e ver quais informações estão relacionadas, exibindo também a distância de uma informação até a outra, aumentando a percepção do relacionamento entre as informações sem a necessidade de ler todo o mapa mental. A busca e visualização das relações e suas distâncias são implementadas através da utilização do banco de dados *Neo4j*, que é projetado para trabalhar com estruturas de grafos que

representam muito bem os “nós” e relacionamentos de um mapa mental, além de possuir um motor poderoso para busca avançada das informações dos grafos;

- Permitir que os mapas mentais possam ser publicados na rede da aplicação, de forma que qualquer usuário possa ter acesso aos mapas classificados como públicos. Os usuários poderão pesquisar e ler mapas mentais já criados por terceiros.

1.3 METODOLOGIA

O projeto *PinchNotes* foi desenvolvido através do desmembramento de um problema genérico; qual seja: “Como criar uma ferramenta que permita construir mapas mentais através de anotações de variadas formas e que sejam compreensíveis não somente para criadores, mas também para leitores comuns?” A fim de buscar soluções ao problema delimitado, considerou-se relevante identificar pequenos problemas isolados, com escopos bem definidos e traduzidos nas seguintes etapas:

1 - Pesquisa bibliográfica: o primeiro passo se deu pela realização de uma pesquisa bibliográfica em torno dos teóricos e autoridades no assunto de mapas mentais, a fim de identificar características e funcionalidades importantes a serem implementadas na aplicação.

2 - Pesquisa por aplicações existentes: os aplicativos *Xmind*, *Coggle*, *Mindjet* e o *Text2mindmap* foram analisados com o objetivo de obter requisitos mais detalhados e validar a utilidade do *Pinch Notes*.

3 - Prototipagem: foram desenvolvidos protótipos isolados e de escopos limitados com o objetivo de identificar a viabilidade e complexidade do desenvolvimento de uma aplicação, que permita manipulação de gráficos, textos e imagens em um ambiente completamente online.

4 - Modelagem de dados: após o desenvolvimento dos protótipos das ferramentas para criação do mapa mental através de anotações, se fez necessário

estudar qual estrutura de dados melhor representaria mapas mentais. Foram feitos dois protótipos de modelagem: um deles representando os “nós” e relacionamentos do mapa como registros em tabelas relacionadas através de chaves primárias. A outra abordagem foi a utilização da estrutura de grafo para representar os “nós” e seus relacionamentos. Em razão da complexidade encontrada para realizar operações como busca, inserção, remoção usando um modelo relacional de tabelas, se decidiu modelar os dados usando estrutura de grafo.

5 - Implementação do modo Área livre: nesta etapa, foi feito o refinamento dos protótipos da ferramenta de inserção, edição dos textos que o usuário pode inserir na área livre para a construção do mapa mental. Enfim, foi desenvolvido o módulo da área livre que é responsável por exibir um quadro branco de espaço ilimitado onde o usuário pode inserir, editar e remover textos e imagens.

6 - Implementação do modo mapa mental: através do refinamento dos protótipos de mapas mentais, foi desenvolvido o módulo responsável pela exibição do mapa mental através dos dados criados a partir da área livre.

7 - Gerenciador: foi desenvolvido o *site* para permitir a consulta, acesso e gerenciamento dos mapas, bem como a criação de contas de usuário.

8 - Escrita do documento de TCC: foram organizadas as anotações criadas durante todo o desenvolvimento do *Pinch Notes*, para serem utilizadas como matéria prima para a escrita do trabalho formal, que se deu no fim do desenvolvimento da aplicação.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por objetivo evidenciar, teórica e metodologicamente, a importância dos mapas mentais no aumento do poder de assimilação e reflexão nos estudos. Sendo um capítulo teórico-metodológico, desenvolve a discussão dos conceitos e tecnologias usados no desenvolvimento deste trabalho, que se sustenta, principalmente, na e pela utilização de bancos de dados não relacionais e orientados a grafos, e a linguagem de programação *javascript* para o desenvolvimento de aplicações *web* fortemente dinâmicas.

Nesse sentido, o trabalho se orienta nas discussões conceituais e metodológicas propostas pelos seguintes autores: Buzan (1996); Flanagan (2011); Freeman & Robson (2014); He, Wu, Miao & Yao (2014); McCreary & Kelly (2013); Nast (2006); Raj (2015); Vukotic (2014); entre outros pesquisadores citados direta ou indiretamente ao longo do texto.

2.1 O MAPA MENTAL

O sistema de ensino e aprendizagem é, majoritariamente, estabelecido através do que os pesquisadores da natureza do raciocínio, Tony Buzan e Jamie Nast, chamam de “Linear Thinking”, cuja tradução mais próxima do termo original é “Raciocínio Linear”.

Para demonstrar o significado real do raciocínio linear e suas implicações, Buzan (1996) realizou uma pesquisa, cujo objetivo correspondia a avaliar quais eram os padrões e estilos de anotação e de rascunho mais usados pelas pessoas. A cada pessoa avaliada foi fornecido um conjunto variado de materiais para a elaboração das anotações, porém a maioria escolheu apenas papel pautado e uma única caneta de cor preta, azul ou cinza.

A pesquisa de Buzan (1996) identificou que os estilos de anotação mais usados são os que expressam a própria natureza do raciocínio linear:

1 - O estilo narrativo consiste de simples escrita do que quer que seja comunicado na forma narrativa;

2 - O estilo lista envolve anotar ideias à medida que elas ocorrem;

3 – O estilo de anotação em ordem numérica ou alfabética consiste de categorias principais e subcategorias (BUZAN, 1996, p. 45).

Buzan (1996) identificou que os estilos de anotação mais usados, então identificados acima, são construídos através de símbolos; de padronização linear formada por sequência cronológica ou hierárquica dos elementos; e da análise cuja qualidade é negativamente afetada por causa da padronização linear.

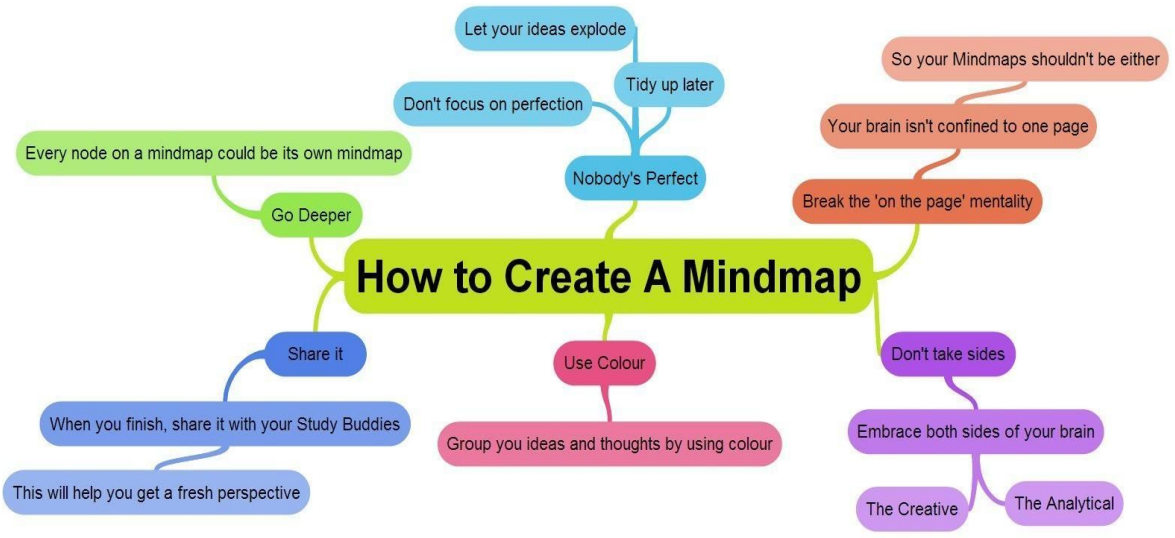
Ora, símbolos, padrão de linearidade e análise são apenas três das faculdades mentais indicadas por Buzan (1996, p. 46) como essenciais na maioria das funções cerebrais, especialmente as que envolvem recordação e aprendizado.

As outras faculdades mentais são: padrão visual, cor, imagem, dimensão, percepção espacial, plenitude e associação. E, em nenhuma dessas, é aplicada em notas lineares tradicionais. Portanto, torna-se evidente o desperdício do potencial cerebral efetuado pelo aprendizado estruturado somente no raciocínio linear.

Em razão desse desperdício e outros fatores negativos, Buzan (1996) procurou desenvolver uma metodologia de raciocínio e notação que melhor aproveitasse as faculdades cerebrais disponíveis a qualquer ser humano: o raciocínio radiante e, sua expressão natural, o mapa mental.

Nesse sentido, o mapa mental é uma ferramenta que serve para aplicar a técnica do “pensamento radial”, proposta por Buzan (1996). O pensamento radial refere-se à prática de refletir sobre um assunto, estimulando o ato de pensar em ideias relacionadas, de forma que seja possível agregar diferentes visões ou sentidos. O mapa mental é, nessa perspectiva, apenas a representação gráfica desse processo de pensamento através de desenhos, palavras chaves, imagens e linhas que interconectam ideias, sinteticamente representado na figura 1.

Figura 1 – Exemplo de mapa mental.



Fonte: Disponível em info.examtime.com (2016).

A utilização de mapa mental como ferramenta de apoio facilita o processo de aprendizado e memorização porque possui maior compatibilidade com a forma como o cérebro funciona. Diferentemente das notas lineares, o mapa mental é semelhante à estrutura neurológica do cérebro, já que a atividade cerebral é exercida pela interconexão de *links* entre milhares de neurônios. De fato, o cérebro humano pode ter um número dessas interconexões tendendo ao infinito.

Uma vantagem no uso de mapas mentais é que eles estimulam a atividade dos dois lados do cérebro, os hemisférios direito e esquerdo, de forma que eles se balanceiam. Na maioria das pessoas, o lado direito é relacionado a operações visuais, associativas e não verbais, já o lado esquerdo é relacionado a operações analíticas e lógicas. De acordo com estudos feitos por Buzan (1996), quando os dois hemisférios são usados de maneira balanceada, as capacidades de aprendizado e memorização aumentam. Isso é exatamente o que ocorre, porque a utilização de mapas mentais estimula tanto a criatividade por causa do uso de desenhos, imagens e cores, e também a lógica através da associação entre palavras-chaves.

Outra vantagem sobre o uso de mapas mentais está no fato de que a atividade cerebral é de natureza associativa e persegue um senso de completude

entre os dados. As associações entre ideias e a visão geral de um assunto são características da organização de conhecimento em mapas mentais. O que não é o caso de notas lineares, que atrapalham essa tendência natural do cérebro em associar e observar a visão geral.

Um artigo publicado pela Hewlett-Packard aponta para o potencial positivo dos mapas mentais: “[...] estudos mostram que pessoas se lembram de 10% do que escutam, 20% do que leem, mas 80% do que veem e fazem” (NAST, 2006, p. 1, tradução nossa). Dessa forma, como a aplicação de mapas mentais envolve informação visual e a criação de visões, ideias e relacionamentos, uma leitura a um mapa mental apresenta o potencial de recordação de 80% dos seus dados.

Por causa dos benefícios prometidos pela abordagem de aprendizado através de mapas mentais, estudos têm sido desenvolvidos para avaliar a pertinência dos resultados da adoção de mapas mentais pelos estudantes. Em um artigo de autoria de He, Wu, Miao, & Yao (2014), sobre a utilização de mapas mentais para o ensino da disciplina de estrutura de dados a uma turma de alunos, foi relatado uma experiência com duas turmas de alunos, em uma delas o conteúdo de estrutura de dados foi lecionado através do uso de mapa mental. Foi observado que os estudantes da porção que utilizou mapa mental obtiveram melhores resultados nos testes do que os estudantes da porção que não utilizou mapa mental.

Aparentemente, a utilização de mapa mental no processo de aprendizagem realmente aumenta a capacidade de entendimento e memorização do assunto a ser estudado. Entretanto, a metodologia pode apresentar alguns problemas, como os apontados por Taylor (2014). Esses problemas afetam principalmente iniciantes em mapas mentais, pois a aplicação de mapeamento mental requer a perda do hábito de pensar linearmente, e essa prática não é simples de ser alcançada por pessoas que aprenderam desde cedo a estudar e pensar dessa forma.

Além disso, um problema que não somente afeta a iniciantes em mapas mentais é que quando se está começando a estudar um novo assunto, não é uma tarefa trivial possuir uma visão clara do assunto de forma que seja possível organizar os pensamentos de maneira fiel à estrutura lógica do conteúdo em questão; tendo

em vista que, em um primeiro momento, o assunto não pode ser devidamente apreendido, posto ser algo novo.

Nesse sentido, para se criar um bom mapa mental, é necessário escolher o assunto principal, as ideias relacionadas e as palavras-chave corretas. No momento em que o assunto ainda é algo desconhecido para o estudante, este não possui informações necessárias para escolher as palavras-chave e as ideias corretas para distribuí-las no mapa. Nessa situação, é melhor fazer anotações lineares sobre o assunto e, no momento em que o assunto se tornar claro o suficiente, convertê-las em um mapa mental.

2.2 BANCOS DE DADOS NÃO RELACIONAIS

Para oferecer suporte a aplicações mais dinâmicas, que fogem dos padrões estabelecidos pelos bancos de dados relacionais, foram criados bancos de dados não relacionais, que são conhecidos, também, como bancos de dados *NoSQL*. Raj (2015) os define como diferentes bancos de dados que surgiram como resposta ao crescimento exponencial do volume dos dados armazenados, e da frequência de acesso a esses dados que, por sua vez, aumenta a necessidade de uma maior performance de processamento.

O termo *NoSQl* significa "*Not Only SQL*", ou, não somente *sql*. Esse termo implica que não existe somente uma solução para persistência de dados para um *software* a ser desenvolvido. Um banco de dados *nosql* diferencia-se pelo fato de não usar um modelo relacional, por funcionar bem em *clusters*, por ser em sua maior parte *open-source*, por ter sido construído para atender as demandas atuais da internet, e por não exigir a definição de esquema da estrutura dos dados.

A característica que mais chama atenção para os novatos no mundo *nosql*, é que todos os bancos de dados *nosql* alegam não exigir a definição de esquemas. Segundo McCreary & Kelly (2013), o que isso significa de fato é que, em um banco de dados *nosql*, é permitido persistir qualquer dado, cada um podendo possuir atributos e estruturas individuais. Significa, também, que o desenvolvedor não

precisa definir um esquema para ser seguido por todos os dados de sua aplicação, e, por causa disso, qualquer código que manipule os dados terá de pressupor algumas considerações iniciais sobre a estrutura dos dados manipulados, como os atributos e os tipos de dados associados a esses atributos.

Nesse sentido, Raj (2015) descreve alguns modelos de dados *nosql*, cada um atendendo a um nicho de aplicações com características específicas:

1 - Bancos de dados chave-valor: operam de maneira semelhante a um dicionário, realizando o mapeando entre chaves e valores. Esses bancos não refletem estruturas ou relacionamentos. São recomendados para uma rápida persistência de dados em meio a operações de alto custo computacional.

2 - Bancos de dados família de colunas: permitem inserção de dados em registros formados por colunas, sendo que as colunas podem ser aninhadas em níveis profundos de dados. Por exemplo, é possível persistir um dado de determinada estrutura que possui outro dado, que possui outro dado, e assim por diante. São recomendados para a persistência de uma grande quantidade de dados não relacionados e de estruturas complexas e variáveis.

3 - Bancos de dados orientados a grafos: permitem a persistência de entidades e relacionamentos entre essas entidades, caracterizando o conjunto dos dados como um grafo. As entidades são também conhecidas como “nós” e possuem propriedades, bem como seus relacionamentos. São ideias para sistemas que empregam um modelo de dados semelhante a um grafo, como uma rede social, por exemplo.

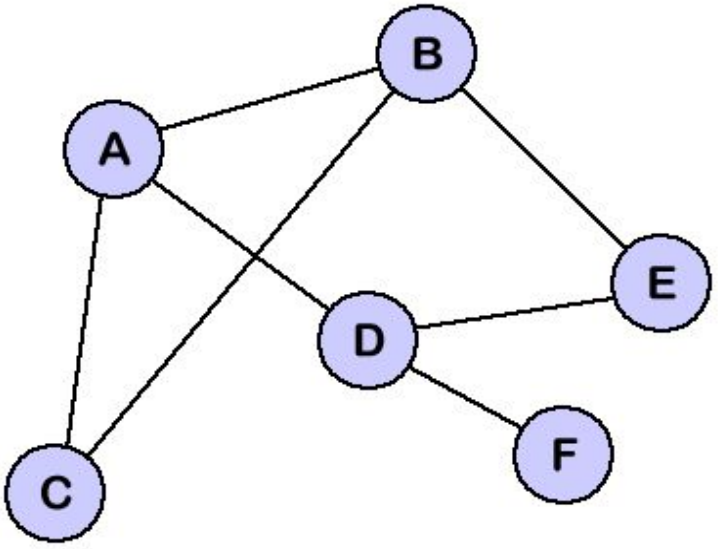
2.3 BANCOS DE DADOS ORIENTADOS A GRAFOS

Como citado anteriormente, um banco de dados orientado a grafos serve para persistir dados que representam uma estrutura de grafo, representada na figura 2, então disposta na próxima página. Na definição conceitual de Raj:

> Grafos são uma maneira de representar entidades e conexões entre elas. Matematicamente, grafos são coleções de nós e fronteiras que denotam entidades e

> relacionamentos. Os nós são entidades de dados cujos relacionamentos mútuos são denotados com a ajuda de fronteiras (RAJ, 2015, p. 8, tradução nossa).

Figura 2 - Representação de um grafo.



Fonte: Disponível em courses.cs.vt.edu (2016).

Uma característica interessante dos grafos é que os “nós” são organizados através de seus relacionamentos, o que permite a visualização de padrões ou caminhos presentes em um grafo. A organização de um grafo possibilita que seus dados sejam persistidos e interpretados de maneiras diferentes, dependendo dos relacionamentos ou entidades enfatizados. Esse atributo representa um desacoplamento entre os dados do grafo e a sua visualização, o que oferece um potencial maior para que diversas aplicações utilizem os mesmos dados e os represente, cada uma ao seu modo.

O banco de dados orientado a grafos é um dos tipos de banco *no-sql*, sendo definido por Raj como:

> Um banco de dados que apresenta suporte às operações de criação, leitura, alteração e remoção de nós e relacionamentos de grafos. Bancos de dados de grafos são sistemas de tempo real por natureza e são usados em sistemas transacionais. Um banco de dados de grafos representa os dados de uma maneira completamente diferente. Eles são representados na forma de estruturas de dados de tipo árvore ou grafos que possuem nós que são conectados uns aos outros através de relacionamentos (RAJ, 2015, p. 13, tradução nossa).

Um banco de dados orientado a grafos procura solucionar problemas enfrentados por aplicações que mantêm dados que se assemelham a grafos em bancos de dados relacionais, como, por exemplo, o problema da adição de um novo relacionamento entre entidades, operação que em um banco de dados relacional necessita da realização prévia de mudanças no esquema e da manipulação nos dados.

Em relação ao comportamento das buscas, percorrer os “nós” através dos relacionamentos entre eles é muito rápido em um banco de dados orientado a grafos, porque esses relacionamentos não são calculados em tempo de execução de uma busca no banco, eles são persistidos como “nós” de relacionamentos. Percorrer relacionamentos persistidos é mais rápido do que os calcular para cada busca.

Vukotic (2014) explica que cada “nó” de um banco de dados orientado a grafos contém fisicamente uma lista de registros de relacionamentos, sendo cada item organizado pelo seu tipo e pela sua direção, podendo manter propriedades adicionais. Sempre que uma operação equivalente a uma junção for executada, o banco de dados simplesmente utiliza essa lista de registros para ter acesso aos “nós” conectados, eliminando a necessidade de operações de busca com alto custo computacional.

Uma das funcionalidades mais importantes dos bancos de dados orientados a grafos, é que eles permitem que os “nós” do grafo possam ter diferentes tipos de relacionamentos, possibilitando a representação dos “nós” e relacionamentos do modelo domínio de uma aplicação incluindo também relacionamentos a objetos de fora do domínio, como, por exemplo, páginas visualizadas por usuários, categorias de vídeos que um usuário assistiu etc. Desde que não exista limite no número e no tipo de relacionamentos que um “nó” possa conter, todos eles podem ser representados em um único banco.

Essa funcionalidade é derivada da forma como os relacionamentos são percebidos por um banco de dados orientado a grafos. Nesse tipo de banco, segundo Raj (2015), os relacionamentos são os elementos centrais, e a maior parte do poder desse tipo de banco de dados é consequência disso. Os relacionamentos

podem possuir propriedades além do seu tipo e dos “nós” origem e destino. O uso de propriedades em relacionamentos pode aumentar a eficiência de como um problema de grafo é representado, por exemplo, em uma rede social, a distância entre dois amigos pode ser salva como uma das propriedades do relacionamento.

Os bancos de dados orientados a grafos estão se popularizando hoje em dia devido ao fato de oferecerem formas mais intuitivas de persistir, buscar e representar dados que podem ser entendidos como grafos. Exemplificando: em uma rede social, buscas pela menor distância entre dois amigos, ou buscas pelos amigos de amigos de um usuário, ou buscas por pessoas e amigos dessas pessoas que acessaram uma mesma página acessada por um usuário são muito mais simples e eficientes em bancos orientados a grafos do que em bancos relacionais ou outros bancos de dados *nosql*.

2.4 JAVASCRIPT

O *PinchNotes* é uma aplicação *web* estruturada em um modelo de arquitetura chamado ANNE, acrônimo que evidencia a utilização de quatro tecnologias: *AngularJS*, *NodeJS*, *Neo4J* e *ExpressJS*.

O denominador comum entre essas tecnologias é a linguagem *javascript*, antes usada somente para implementar componentes do lado do cliente, agora usada para criar aplicações *web* também no lado do servidor.

O fato de que todas as tecnologias do modelo ANNE são baseadas em *javascript*, permite que todos os módulos da aplicação *web* tenham acesso a funcionalidades excepcionais da linguagem *javascript*, como a orientação a objetos através de protótipos, a arquitetura orientada a eventos e os poderosos *closures*.

O poder dessas funcionalidades do *javascript* oferece fundamento à utilização da linguagem em todas as camadas da aplicação. Para entender a arquitetura do sistema, e a implementação de alguns dos requisitos, é necessário entender esses conceitos tão poderosos e dinâmicos do *javascript*. Assim, as próximas seções têm por objetivo explicar brevemente cada um.

2.4.1 Orientação a objetos através de protótipos

Embora o *javascript* seja orientado a objetos, ele não implementa as tradicionais classes usadas em linguagens de programação como *java* e *c#*. Mas, mesmo assim, por ser mais intuitivo pensar em classes, constantemente as pessoas utilizam o termo classe para designar a estrutura de um determinado objeto em *javascript*.

Flanagan (2011) enfatiza que quase tudo em *javascript* é um objeto, incluindo as funções, sendo que cada objeto possui uma propriedade interna chamada *prototype*, que referencia outro objeto. O objeto referenciado pelo *prototype* também tem uma propriedade *prototype* que referencia outro objeto e assim por diante. Essa cadeia de protótipos é o que chamam de *prototype chain*. A cadeia acaba quando o *prototype* aponta para o objeto mais genérico do *javascript*, o *Object*. A propriedade *prototype* do objeto (*Object*) é nula, finalizando a cadeia de protótipos.

A cadeia de protótipos em *javascript* é usada para simular heranças. Quando uma propriedade de um objeto é requisitada por um cliente, o *javascript* se encarrega de verificar se o objeto contém a propriedade desejada, caso não a encontre, ele verificará se a propriedade existe no objeto referenciado pelo protótipo do objeto requisitado. O *javascript* faz essa busca até encontrar a propriedade, passando de protótipo a protótipo, ou até alcançar o objeto cujo protótipo é nulo, o *Object*, resultando na não existência da propriedade requisitada.

Como evidenciado por Freeman & Robson (2014), as heranças do *javascript* ocorrem em tempo de execução do programa, permitindo uma maior flexibilidade em relação aos comportamentos de cada objeto, pois a qualquer momento é possível referenciar um outro *prototype* para herdar outras características e comportamentos, bem como adicionar propriedades e métodos ao objeto ou ao próprio *prototype* dinamicamente.

2.4.2 *Closure*

Segundo Freeman & Robson (2014), *closures* são objetos que contêm uma função e uma referência ao escopo no qual essa função foi declarada. No entanto, para entender os *closures*, é necessário entender dois elementos básicos do *javascript*: as funções de primeira classe e as funções aninhadas.

As funções em *javascript* são consideradas funções de primeira classe, porque elas podem ser manipuladas como qualquer outro objeto em *javascript*, pelo motivo de que uma função é nada mais que um objeto. Como um objeto possui referência, uma função pode ser retornada por outra função, podendo ser passada como argumento a outra função, e também comportar atributos e métodos atribuídos.

As funções aninhadas são funções declaradas dentro do escopo de outras funções. Toda vez que uma função possuir uma função aninhada chamada, uma instância da função aninhada é criada.

A instância de uma função aninhada possui acesso direto aos atributos e métodos contidos no escopo, onde a função aninhada foi declarada, mesmo que essa função aninhada seja chamada em outro contexto. E isso é implementado através dos *closures* do *javascript*.

Um *closure* é criado quando uma função aninhada é acessível fora do escopo de onde foi declarada, isto é, quando é possível referenciá-la fora do contexto de onde foi declarada. Isso pode ser feito quando uma função retorna uma função aninhada como valor de retorno, sendo, nesse caso, o valor de retorno apenas uma referência à função aninhada, como apontado por Freeman & Robson (2014).

Quando a referência retornada é usada para fazer uma chamada à função retornada, independentemente do escopo onde a chamada estiver sendo feita, a função retornada continuará mantendo uma referência ao ambiente onde foi declarada. Esse ambiente é, na verdade, um objeto que contém atributos e métodos do respectivo escopo.

Flanagan (2011) observa que os *closures* combinam muito bem com a arquitetura orientada a eventos do *javascript*, pois permitem que uma função *handler* de um observador de determinados eventos seja uma referência a uma função

pertencente a outro escopo, o que diminui o acoplamento entre o código responsável por detectar um evento e o código responsável por realizar as tarefas de resposta a esse evento.

2.4.3 Arquitetura orientada a eventos

Javascript é ótimo para realização de programação orientada a eventos. Ele permite essa funcionalidade através do padrão de projeto *observer*, onde os objetos interessados em determinados eventos são inscritos em um objeto publicador de eventos.

Segundo Freeman & Robson (2014), os eventos do *javascript* funcionam da seguinte maneira: o motor *javascript* de um navegador, como o do *Google Chrome*, possui, em sua implementação, um *pool* de *threads* usado para capturar eventos. Quando um evento é capturado por alguma dessas *threads*, ele é adicionado numa fila de eventos que é gerenciada por um mecanismo do *javascript* chamado *event loop*.

O *event loop* é executado em uma única *thread*, e tem a responsabilidade de, repetidamente, acionar o *handler*, ou tratador de evento, do primeiro evento da lista. Quando um *handler* acionado termina de executar, o evento respectivo é removido da lista de eventos, dando lugar ao próximo evento.

Assim, pelo fato do *event loop* ser executado em uma única *thread*, um *handler* que realize alto esforço computacional irá retardar o atendimento dos outros eventos da lista. Porém, existem técnicas para diminuir ou evitar esse problema. Como citado por Teixeira (2012), uma dessas técnicas trata-se de realizar o adiamento da execução do *handler* que exige esforço computacional, de modo que seja executado somente na próxima iteração do *event loop*.

A tarefa de associar um *handler* a um tipo de evento que pode ocorrer em um determinado objeto é feita, por exemplo, através do método *addEventListener* de um *dom object*, que tem por parâmetros o tipo do evento e um *callback* que representa o *handler*. No lugar desse *callback*, deverá ser passada uma referência para a função responsável por realizar o tratamento do evento. Assim, quando um

evento ocorre, ele é adicionado na fila de eventos, e o *callback* passado como argumento é executado pelo *event loop* em algum momento da sua iteração.

Os *closures* do *javascript* permitem o registro de *callbacks* aos ouvintes de eventos, de modo que, quando executados, continuam com referências para os atributos e métodos pertencentes ao escopo onde foram inicialmente declarados. Esse atributo permite que o *event loop* não precise conhecer os objetos dos quais os métodos de tratamento de eventos fazem parte. Só quem precisará conhecê-los é o próprio *handler* ou *callback*, através do *closure*, o que aumenta a flexibilidade no tratamento de eventos.

3 PINCH NOTES: FERRAMENTA PARA CRIAÇÃO DE MAPAS MENTAIS

Este capítulo demonstra o que é a ferramenta *Pinch Notes* proposta por este TCC e quais problemas ela procura solucionar. Nessa parte, também é descrito o seu processo de implementação, abordando a etapa de análise, destacando os requisitos levantados. Em seguida, são expostos os aspectos do projeto, apresentando o projeto arquitetural e o esquema usado para a representação dos dados da aplicação no banco de dados. Finalizando o capítulo, é demonstrada a implementação das funcionalidades principais da aplicação.

3.1 A FERRAMENTA

Pinch Notes é uma aplicação web, atualmente com suporte somente para sistemas *desktop*, que oferece meios de criar mapas mentais através de imagens, notas lineares, textos e realces em textos. Através da aplicação, é possível pesquisar e ler os mapas mentais públicos criados na aplicação por outros usuários.

Visto que a utilização efetiva de mapas mentais depende da perda de hábitos já arraigados no cotidiano de um estudante comum, como o ato de organizar pensamentos linearmente, muitas vezes através de ferramentas de edição de texto em razão da popularização das tecnologias de informação. Essa aplicação visa oferecer ao estudante uma maneira de obter os benefícios da análise das informações através de mapas mentais, sem, contudo, forçá-lo a mudar sua metodologia de estudo e organização de pensamentos.

Em razão da ineficiência da assimilação de conhecimento a partir da leitura de mapas mentais de terceiros, devido à aparência caótica característica de um mapa mental, cuja representação é constituída de “nós” e relacionamentos formados por palavras-chave e imagens que se avistadas por um leitor não conhecedor do assunto abordado, apenas ocultam todas as nuances e entendimentos do conteúdo expresso no mapa. O *Pinch Notes* foi desenvolvido com objetivo de tornar os mapas

mentais materiais de apoio compartilháveis, auxiliando, assim, outras pessoas no processo de aprendizado de um assunto.

Os dois objetivos citados acima, criar mapas mentais sem precisar mudar de paradigma de raciocínio e ler mapas mentais de terceiros para entender um assunto foram alcançados através da implementação do conceito utilizado por essa aplicação: a criação de mapa mental orientada a anotações.

No *Pinch Notes*, as notas lineares assumem o papel principal. É a partir da inserção de textos e realces em trechos de textos feitos pelo usuário, que a ferramenta gera a visualização do mapa mental. Dessa forma, a aplicação conta com dois tipos de visualização: a área livre e o mapa mental.

A área livre é um espaço aberto onde o usuário pode inserir textos e imagens. Os textos da área livre não seguem formatação predefinida, de forma que o usuário tem a liberdade de inserir um texto em qualquer posição da área livre que já não esteja ocupada por outro texto.

A característica essencial da área livre é que nela os textos podem possuir marcações ou realces. Os realces em textos são uma solução utilizada não somente para enfatizar partes de um texto, mas, também, para criar novas camadas de profundidade acessíveis a partir da operação de *zoom* pelo *scroll* do *mouse* que podem ser usadas para agregar informações relacionadas às partes do texto realçadas.

As camadas de profundidade são novas áreas-livre que se referem às marcações no texto. Cada marcação no texto possui sua camada de profundidade, que, por sua vez, pode possuir novos textos, marcações e imagens. Através dessa maneira de organizar conteúdos, inserindo cada tipo de informação em um contexto que faça sentido, o *Pinch Notes* torna mais organizado, prático e intuitivo o processo de encontrar informações aninhadas em muitos níveis de detalhamento.

Além de permitir uma maior organização das informações, essa estratégia das camadas de profundidade torna possível a representação em mapa mental dos dados inseridos nas áreas-livre, com “nós” e relacionamentos, onde cada texto, marcação de texto e imagem é um “nó” do mapa mental, e a relação de uma

marcação de texto com os dados pertencentes a sua camada de profundidade é entendida como um relacionamento no mapa.

Através da visualização dos dados no modo mapa mental, é possível visualizar a imagem geral dos dados e seus relacionamentos inseridos na área livre pelo usuário. O modo de visualização em mapa mental também permite a manipulação gráfica dos nós do mapa, refletindo em novas posições para cada nó ao arrastar um nó qualquer sobre o espaço onde o mapa está desenhado.

Por utilizar dois modos de visualização – os modos mapa mental e área livre - para representar os dados inseridos pelos usuários, o *Pinch Notes* pode satisfazer as necessidades de vários tipos de clientes. De imediato, pode ser utilizável até por quem não tem o menor interesse em mapas mentais, mas que queira somente organizar melhor suas anotações e pensamentos.

O *Pinch Notes* pode servir para um domínio indefinido de necessidades, podendo ser uma boa ferramenta de apoio para quem:

1 - Esteja Interessado em usar mapas mentais, mas nunca os estudou;
2 - Já seja veterano na prática do mapeamento mental;
3 - Reconhece os benefícios dos mapas mentais, mas não gosta de utilizá-los ou não tem paciência para aprendê-los ou construí-los;
4 - Gostaria de estudar através de mapas mentais criados por professores ou outras pessoas que possuem melhor domínio no assunto abordado;
5 - Deseja realizar *brainstorm* para tomadas de decisões;
6 - É professor e quer avaliar a capacidade associativa dos seus alunos;
7 - Oferece serviços de ensino, tal como empresas de cursos para concursos públicos, e queira tornar seus materiais de apoio mais dinâmicos e menos entediantes;

Em suma, o maior objetivo do *software* desenvolvido neste trabalho, o *Pinch Notes*, é oferecer uma solução aos problemas já apontados na seção 1.1 Motivação deste trabalho, proporcionando ao usuário um ambiente no qual possa inserir suas anotações lineares, da forma como sempre estudou, aprendendo novos assuntos

através delas e aumentando seu entendimento e memorização através da visualização e manipulação do mapa mental gerado pela aplicação.

3.2 ANÁLISE

Requisito 1: Cadastro de usuários

O sistema deve prover um meio de cadastrar usuários de maneira rápida e simples.

Requisito 2: Perfil do usuário

O sistema deve possuir uma tela onde serão exibidas as informações relativas ao usuário com sessão ativa, contendo os dados do usuário informados na etapa do cadastro, uma seção de busca para consultar mapas, e uma seção para criação de um novo mapa.

Requisito 3: Criação, leitura e remoção de mapa mental

O sistema deve permitir que o usuário crie, leia e remova seus mapas mentais.

Requisito 4: Modo de visualização Área Livre

O sistema deve permitir a visualização de um mapa mental no modo área livre. A área livre é uma área de trabalho que deve permitir a inserção, edição, e a remoção de textos e imagens em um espaço ilimitado.

Requisito 5: Inserção, edição e remoção de texto em área livre

No modo área livre, deverá ser exibido um editor de texto para que se possa inserir, editar ou remover textos na área livre.

Requisito 6: Inserção e remoção de realces de texto

O editor de texto da área livre deve permitir a inserção e remoção de realce no trecho de texto selecionado. Os realces colocados no editor de texto devem ser refletidos no texto correspondente desenhado na área livre.

Requisito 7: Adição de detalhes a um realce de texto

O sistema deve prover um meio de adicionar textos e imagens relacionados a um realce de texto, de modo que esses novos elementos fiquem dispostos em uma nova área livre, ou seja, uma nova camada da área livre.

Requisito 8: Navegar entre camadas de área livre

O sistema deve permitir a navegação entre camadas da área livre através de operações de *zoom-in* e *zoom-out* sobre os elementos que representam realces de texto.

Requisito 9: Inserir ou remover imagem em área livre

Deve ser permitida a inserção ou remoção de imagens na área livre através do editor de texto.

Requisito 10: Visualização modo mapa mental

O sistema deve prover um modo de representar os dados inseridos na área livre através de uma estrutura de grafo, exibindo todos os realces de texto representados como “nós” do mapa, e os seus relacionamentos através de linhas.

Requisito 11: Pesquisa por mapas mentais públicos

O sistema deve permitir a pesquisa por mapas mentais públicos criados por terceiros.

Requisito 12: Definição de permissão de mapa mental

O sistema deve permitir que o usuário defina as permissões do seu mapa mental, podendo escolher entre público e privado.

Requisito 13: Pesquisa por “nós” do mapa que contenham determinado conteúdo

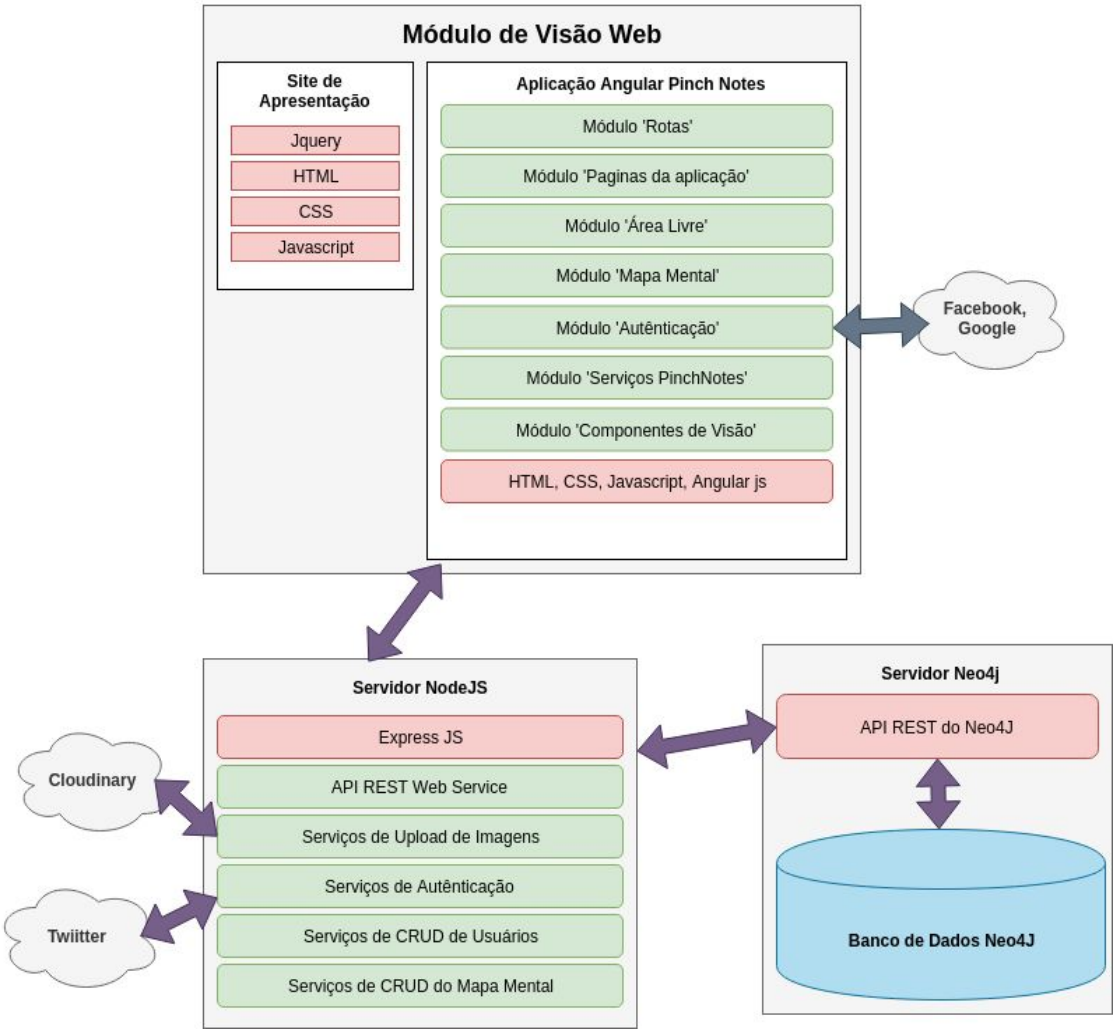
O sistema deve permitir que o usuário encontre quais “nós” do mapa possuem os termos de busca, podendo exibir os resultados no modo área livre ou mapa mental.

3.3 PROJETO ARQUITETURAL

Nesta seção será exposta uma visão geral do projeto arquitetural da aplicação, apresentando cada um dos módulos e componentes de *software* que, em conjunto, compõem a solução desenvolvida.

A aplicação *Pinch Notes* é composta por três módulos principais: o módulo que representa as aplicações *web* escritas com *javascript*; o módulo do servidor da aplicação implementado com *Nodej;* e, por fim, o módulo que representa o banco de dados orientado a grafos *Neo4j*. A figura exposta abaixo demonstra, visualmente, os módulos descritos acima, enfatizando as interações entre eles.

Figura 3 - Projeto Arquitetural.



Fonte: Do autor.

O módulo de visão *web* é composto por dois sub-módulos: o *site* de apresentação e a aplicação *Angular Pinch Notes*, que, de fato, permitem a criação e leitura dos mapas mentais. O *site* de apresentação tem como objetivo expor de maneira breve as funcionalidades do *Pinch Notes*, permitindo que o usuário acesse o aplicativo através de um *link.*

A aplicação *Angular* funciona através da coexistência e colaboração mútua de vários sub-módulos, cada um responsável por objetivos específicos isolados, de modo que possam ser reusados por outras aplicações:

1 - Módulo "Rotas": é responsável pelo mapeamento dos endereços das páginas da aplicação e seus respectivos conteúdos.

2 - Módulo "Páginas da aplicação": este módulo possui diretivas do *Angular* que representam as páginas da aplicação. Cada diretiva representa uma página de forma que essas páginas podem ser reusadas por outras diretivas sem esforço de adaptação, e, dessa forma, podendo também ser mapeadas para outros endereços caso necessário.

3 - Módulo "Área Livre": é constituído por diretivas que permitem a construção do mapa mental no modo área livre, entre elas a mais importante: a diretiva do editor de texto PELL, que permite a criação, edição de textos das mais variadas formas.

4 - Módulo "Mapa mental": módulo responsável pela renderização do mapa mental em modo grafo. É construído a partir da biblioteca de manipulação gráfica VISJS.

5 - Módulo "Autenticação": possui serviços para que a aplicação possa requisitar ações de autenticação do usuário e finalização de sessão ativa. A autenticação pode ser feita através das redes sociais *Facebook*, *Google* e *Twitter*. Nesse caso, o sistema verifica se o usuário está ativo através de um interceptador de requisições que intercepta toda requisição que precisa de uma sessão de usuário ativa, garantindo que ações específicas não sejam executadas sem que o usuário esteja autenticado.

6 - Módulo “Serviços Pinch Notes”: módulo responsável por se comunicar com a camada de serviços REST do servidor.

7 - Módulo “Componentes de visão”: é um conjunto de diretivas utilitárias reusáveis, que permitem a implementação rápida e organizada das funcionalidades. As diretivas deste módulo procuram abstrair controladores de busca, envio de imagens, botões do editor de texto, listas e cartões que representam os mapas mentais e usuários em listagem.

O segundo módulo principal da aplicação é o servidor *Node js*, responsável por expôr serviços REST de CRUD de usuários, mapas mentais, textos e imagens, junto com os serviços de autenticação. Entre os serviços de autenticação, é utilizada a biblioteca *node-twitter-signin* desenvolvida pelo autor deste TCC, para abstrair as complexidades de autenticação OAUTH do Twitter. Os serviços de envio de imagens são baseados na utilização do serviço *Cloudinary*, que permite armazenar imagens na nuvem.

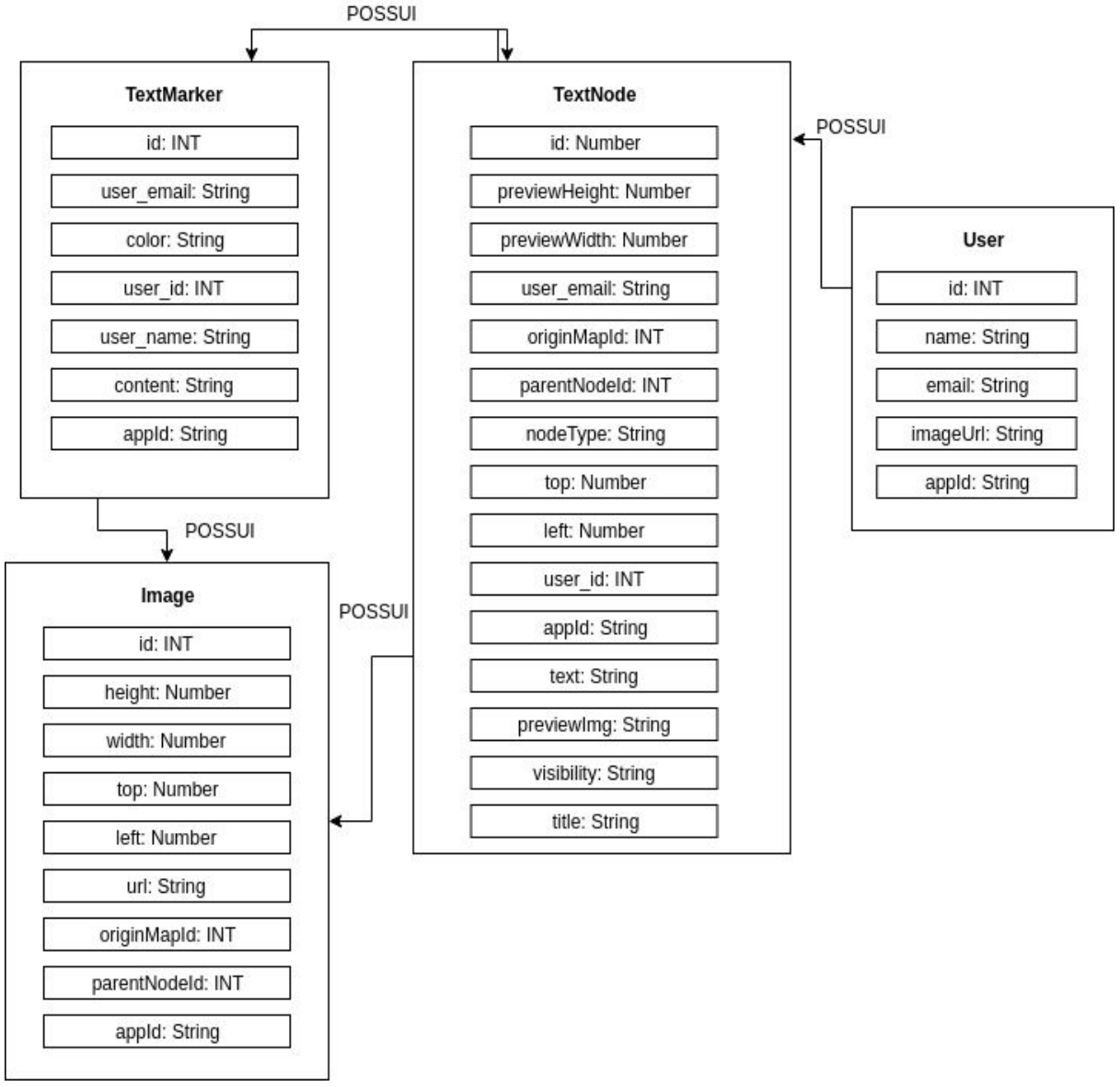
Por fim, o servidor concretiza as ações do usuário, realizando requisições para o terceiro módulo principal da aplicação, onde o banco de dados *Neo4j* é mantido e exposto através de uma *api* REST embutida no próprio sistema do banco de dados.

3.4 MODELAGEM DOS DADOS

Esta seção tem como objetivo apresentar o esquema lógico de dados usado para a representação dos dados de mapas mentais e dos usuários na aplicação.

Como os dados no banco de dados *Neo4j* são salvos através de “nós”, propriedades e relacionamentos, a aplicação *Pinch Notes* tem sua modelagem de dados realizada a partir da definição de quatro tipos de nós: *TextNode*, *TextMarker*, *User*, e *Image*. Cada um possuindo propriedades que representam suas características e informações necessárias para a sua renderização gráfica na aplicação web. A figura 4, então disponível na próxima lauda, exemplifica a estrutura inicial dos dados da aplicação.

Figura 4 – Modelagem dos dados.



Fonte: Do autor.

O *TextNode* é o tipo de “nó” que representa textos que podem ser inseridos na área livre do mapa mental. Também é usado para representar o mapa mental, em razão de que o mapa mental é, na prática, nada mais que o primeiro “nó” textual relacionado ao “nó” de tipo *User*. Portanto, é possível que “nós” textuais de um mapa

referenciem outros mapas da aplicação, através do mesmo tipo de relacionamento que será discutido mais adiante ao fim deste capítulo.

Sendo assim, os principais atributos do *TextNode* são: *id*, *previewHeight*, *previewWidth*, *originMapId*, *parentNodeId*, *top*, *left*, *text*, *previewImg*, *visibility* e *title*. Os atributos *top*, *left*, *previewHeight*, *previewWidth* são utilizados pelo módulo da área livre da aplicação *web* para renderizar corretamente o "nó" textual na posição e dimensão escolhidas pelo usuário. O atributo *visibility* indica se o mapa é público ou privado. Os atributos *originMapId* e *parentNodeId* permitem a construção de consultas mais performáticas ao banco de dados, diminuindo a quantidade possível de resultados obtidos por uma consulta por "nós" de um mapa.

O *TextMarker* é um tipo de "nó" que representa o realce de texto, possuindo os atributos *id*, *content* e *color* como seus atributos específicos. Esse tipo de "nó" é usado pela aplicação para a renderização dos realces de texto no modo mapa mental, no qual os "nós" são os realces de texto. A propriedade *content* é do tipo *string* e possui o texto realçado. Já a propriedade *color*, também *string*, contém o código hexadecimal da cor utilizada pelo realce.

O *User* é um tipo de "nó" usado para representar o usuário da aplicação, possui somente as propriedades *name*, *email*, *imageUrl*, devido ao fato de que a criação de conta de usuário e sua autenticação se dá somente através das redes sociais *Facebook*, *Twitter* e *Google*.

O tipo de "nó" *Image* é utilizado para representação de imagens adicionadas pelo usuário na área livre. Entre seus atributos, *height*, *width*, *top*, *left* são usados pela aplicação para renderização seguindo a posição e a dimensão escolhidas pelo usuário. Como as imagens são salvas pelo serviço de hospedagem na nuvem do *Cloudinary*, o atributo *url* guarda o endereço final para a imagem salva na nuvem. Os atributos *parentNodeId* e *originMapId* também são utilizados por esse tipo de "nó" para performance de consultas mais eficientes ao banco de dados.

Cada tipo de "nó" possui também informações replicadas do "nó" de tipo *User*, com o objetivo de reduzir a quantidade de consultas ao banco para serem obtidas informações do usuário-autor do "nó", seja o "nó" um mapa mental, um texto, um realce de texto ou uma imagem.

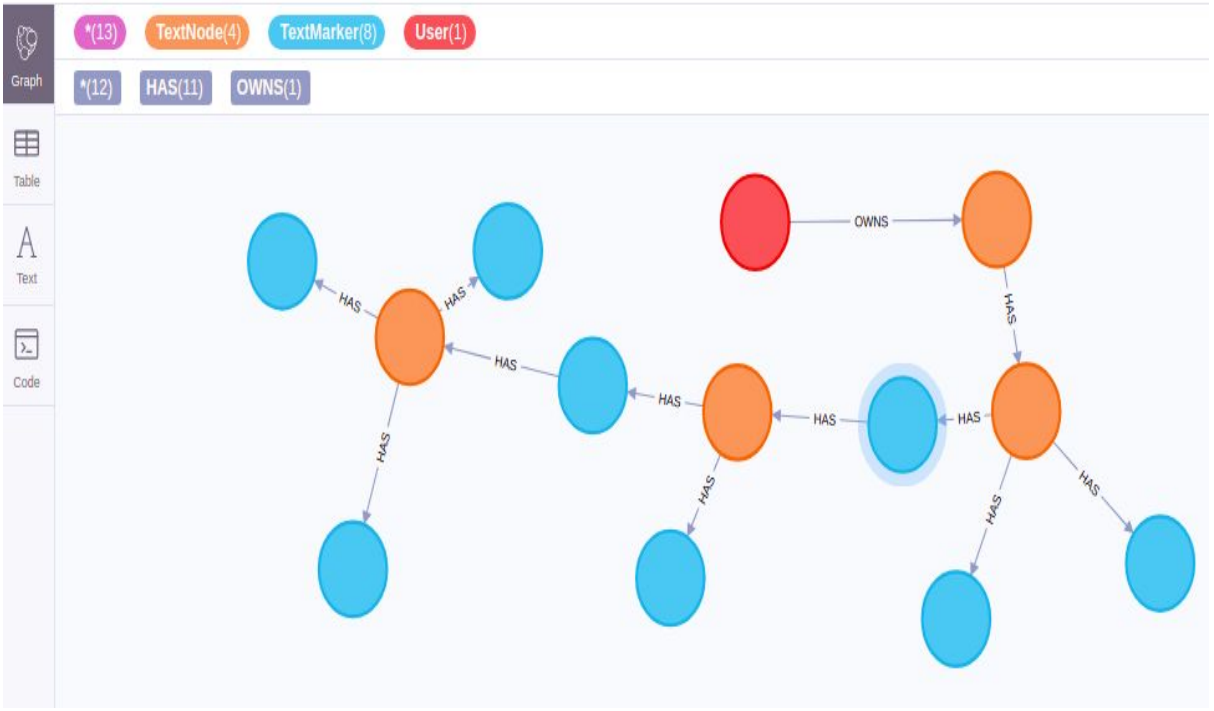
Sabendo que bancos de dados NOSQL não exigem regras para definição de esquemas lógicos a serem seguidos, e obtendo vantagem deste fato, alguns “nós” e relacionamentos podem ter um número diferente de propriedades do que a quantidade inicialmente definida na modelagem. Desse modo, é possível adicionar dinamicamente mais informações aos “nós” do banco. Assim, em alguma adaptação futura ao *Pinch Notes*, podem ser adicionadas aos “nós” informações que especificam ou representam os “nós” de maneira diferente sem ter que, para isso, criar novos esquemas de dados e possivelmente sofrer o risco de quebrar a compatibilidade com versões anteriores.

Para representar os relacionamentos entre os nós, o banco de dados *Neo4j* possui o conceito de relacionamentos. Os relacionamentos são criados entre dois “nós” do banco de maneira dinâmica em tempo de execução e sem restrições. Como os “nós” possuem tipos, os relacionamentos também. Esses tipos são utilizados pelo banco *Neo4j* para aprimorar a performance das consultas.

Na modelagem proposta pelo *Pinch Notes*, foi definido o tipo “possui”, indicando que um “nó” possui outro “nó” em seu conteúdo. A razão por essa nomenclatura se dá pelo fato de que, no *Pinch Notes*, cada realce de texto inserido em um texto cria uma camada da área livre, onde é possível inserir mais textos e imagens. Portanto, visualmente, o relacionamento entre os “nós” é percebida como o ato de possuir. Sendo assim, um mapa, representado por um “nó” de tipo *TextNode*, possui outros “nós” que podem ser de tipo *TextNode*, *TextMarker* ou *Image*. Como os “nós” são relacionados através do relacionamento de tipo “possui”, e o mapa é representado através do tipo *TextNode*, um mapa pode conter “nós” que referenciam outros mapas mentais, através da criação do relacionamento entre os “nós” *TextMarker* e *TextNode*, independente do mapa referenciado ser de autoria de outro usuário.

Para melhor entendimento de como se efetuam os relacionamentos entre os “nós” da aplicação, segue a figura 5, então extraída da ferramenta de visualização WEB, do servidor do banco de dados *Neo4j*. Na ilustração, os “nós” em cor azul são do tipo *TextMarker*, os em laranja representa o *TextNode*, e os em vermelho simboliza o tipo *User*.

Figura 5 – Grafo do Neo4j.



Fonte: Imagem extraída do servidor do banco de dados *Neo4j*.

3.5 IMPLEMENTAÇÃO

Nesta parte, descreve-se o processo de construção da ferramenta proposta neste TCC, descrevendo a implementação das funcionalidades principais do *Pinch Notes*.

3.5.1 Autenticação

Para facilitar a manutenibilidade do aplicativo *Pinch Notes*, foi decidido utilizar bibliotecas que abstraem a autenticação de usuários através das redes sociais *Facebook*, *Google* e *Twitter*. Para o *Facebook*, foi utilizada a biblioteca *angular-facebook*. Para o *Google*, utilizou-se a *angular-google-plus*. Já para o

Twitter, a biblioteca *node-twitter-signin* foi desenvolvida pelo autor, visando o aprendizado concreto da implementação do processo de autenticação *OAuth*.

O fluxo de criação de conta de usuário e autenticação é iniciado a partir do *click* em um dos botões de *login*, na primeira página da aplicação. O *click* dispara a ação de chamada à biblioteca responsável pela rede social escolhida, abrindo uma nova aba com a página da rede social para que o usuário possa inserir seus dados.

A partir do momento em que a biblioteca de autenticação recebe os dados do usuário, como *e-mail*, nome e foto, a aplicação *angularjs* realiza uma chamada à *api* REST do servidor responsável por salvar e autenticar o usuário. O servidor cria um *token* JWT - um padrão (RFC 7165) que define como transmitir de forma segura objetos JSON compactos entre aplicações - a partir dos dados do usuário agregados no corpo da requisição.

O servidor salva o usuário no banco de dados se não existir, e, caso já exista, será atualizado. Em seguida, uma resposta contendo o *token* gerado é enviada para a aplicação que será responsável por adicionar o *token* numa área persistente do navegador chamada de *localstorage*.

As próximas requisições que a aplicação *angularjs* realizar serão autenticadas através de um interceptador que decora o serviço *$http* do *angularjs*. O código da figura 6, então exibida na próxima lauda, exemplifica o processo:

Figura 6 – Interceptor HTTP.

```
angular.module('Authentication')
.factory('AuthInterceptor', AuthInterceptor)
.config(($httpProvider) => {
    $httpProvider.interceptors.push('AuthInterceptor');
});

function AuthInterceptor ($location, AuthService, $q) {
  'ngInject';

  return {
    request: request,
    responseError: responseError
  };

  function request(config) {
    config.headers = config.headers || {};

    if (AuthService.getToken()) {
      config.headers['Authorization'] = 'Bearer ' + AuthService.getToken();
    }

    return config;
  }

  function responseError(response) {
    if (response.status === 401 || response.status === 403) {
      $location.path('/signin');
    }
    return $q.reject(response);
  }
}
```

Fonte: Do autor.

O interceptador customizado é responsável por decorar os métodos *request* e *responseError*, que são implementados originalmente no serviço *$http* do *angularjs*. O método *request* agora também obtém o *token* armazenado no *localstorage* a partir do serviço *AuthService*, e, em seguida, adiciona o *token* nos *headers* da requisição HTTP.

Através deste trecho de código, é possível observar a utilização do conceito *closure*, então explicado na seção 2.4.2. A partir dessa poderosa funcionalidade do *javascript*, quando as funções *request* e *responseError* são executadas pelo motor interno do *angularjs* já em um outro contexto, as referências para os serviços *$location*, *AuthService* e *$q* não são perdidas, pois estão atreladas às funções que,

em *javascript*, são também objetos, como propriedades de seu escopo, ou, melhor, *closure*.

O servidor também possui um interceptador de requisições para validar a autenticação do usuário. Em termos de comportamento, se parece com o interceptador demonstrado acima. Segue a exposição do código na próxima figura:

Figura 7 – Autenticação do Servidor.

```
5   module.exports = function authMiddleware(req, res, next) {
6       if (req.method === 'OPTIONS' || req.method === 'options') {
7           HttpUtils.sendSuccessResponse(res);
8           return;
9       }
10
11      const token = req.headers['x-access-token'];
12
13      if (token) {
14          jsonwebtoken.verify(token, secretKey, (err, decoded) => {
15              if (err) {
16                  HttpUtils.authError(res, { msg: "Failed to authenticate user" });
17              } else {
18                  req.user = decoded.user;
19                  next();
20              }
21          });
22      } else {
23          HttpUtils.authError(res, { msg: "No token provided" });
24      }
25
26  };
```

Fonte: Do autor.

A função *authMiddleware* é executada sempre antes de um *endpoint*, que precisa de um usuário autenticado para ser requisitado. A figura abaixo demonstra como a função *authMiddlware* é utilizada:

Figura 8 - Uso do *Middleware*.

```
mapApi.use(require('./../../authentication/middleware.js'));
```

Fonte: Do autor.

O objeto *mapApi* é uma instância da função *Router* fornecida pelo *framework Express*. Através do *mapApi*, é possível adicionar e mapear todos os métodos HTTP à *urls* específicas, relacionando-as com as funções a serem executadas pelo servidor. O método *use* é responsável por enfileirar funções antes da execução das funções de tratamento das requisições. Neste caso, o método *use* está sendo usado para adicionar a função *authMiddleware* na primeira posição na fila de funções a serem executadas para tratar as requisições feitas ao *endpoint 'api/map'*. Dessa forma, nenhuma função posterior será executada caso a autenticação resulte em erro, pois a função *authMiddleware* já envia a resposta para o cliente informando um erro de autenticação como visto na figura 7.

3.5.2 Autenticação com *Twitter*

Como informado no início da seção 3.5.1, a autenticação através do *Twitter* foi implementada através da utilização de uma biblioteca desenvolvida pelo autor e disponibilizada para o público através do *github* e *npm*: *Node-Twitter-Signin*.

A biblioteca desenvolvida tem o único objetivo de abstrair as complexidades para a implementação de uma autenticação *OAuth,* em um servidor *nodejs* implementado com o *framework Express*. Ela atinge esse objetivo expondo uma interface simples, onde o desenvolvedor apenas precisa fornecer as informações de segurança do seu aplicativo da plataforma do *Twitter*, e as funções que serão executadas quando os dados do usuários forem autenticados. A demonstração de sua utilização encontra-se exposta na figura abaixo:

Figura 9 - Biblioteca *Node-Twitter-Signin*.

```
const TwitterApi = require('node-twitter-signin');

//really simple, right?
app.use('/', TwitterApi(key, secret, callbackUrl, callback, callback_render ));
```

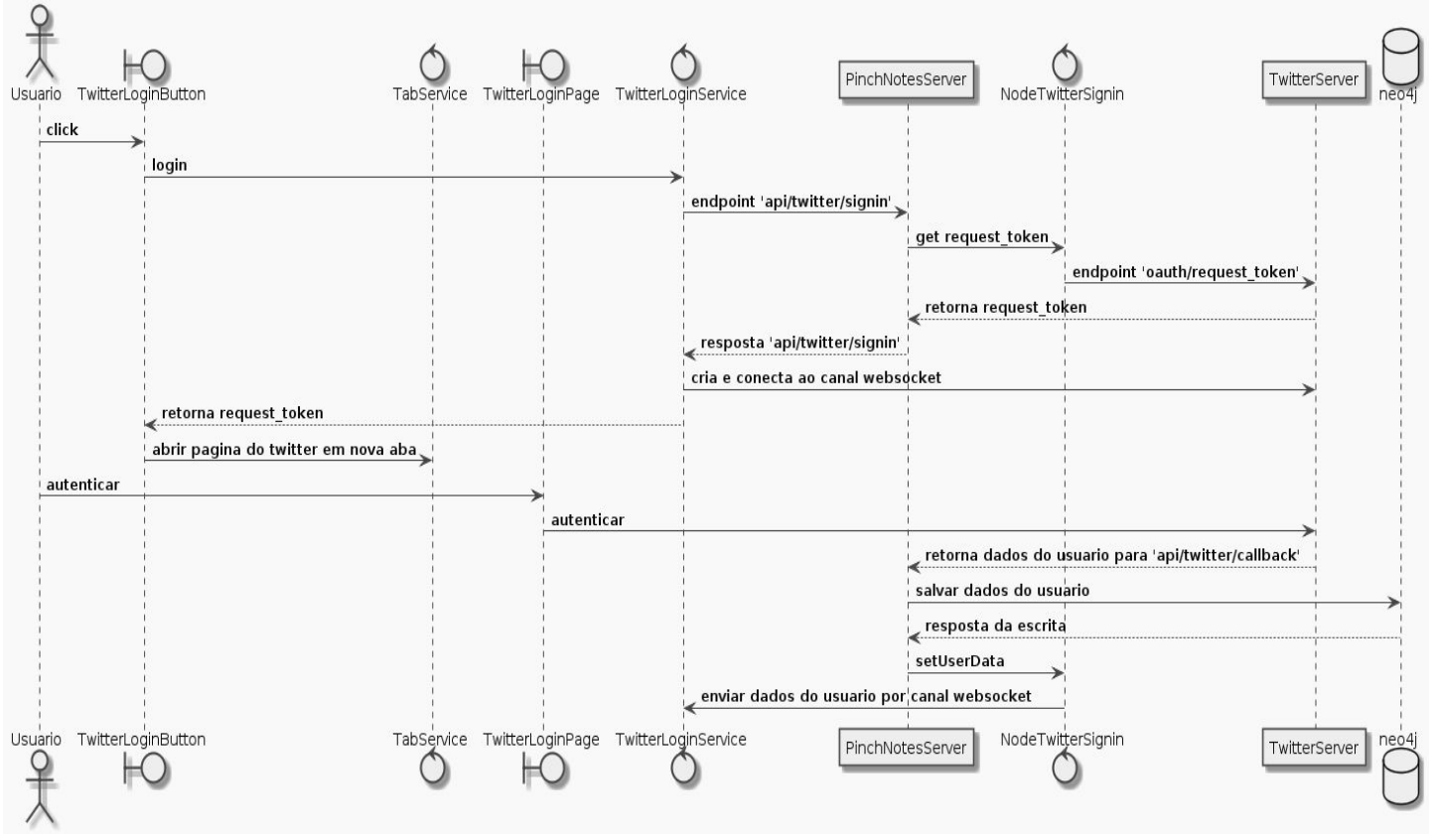
Fonte: Do autor.

No código acima, a biblioteca é utilizada através da adição de sua função na fila de funções que tratam as requisições através do método *use* fornecido pelo *framework Express*. A biblioteca precisa de algumas informações para funcionar; quais sejam:

- ***key e secret***: informações secretas da aplicação do *Twitter*;
- ***callbackUrl***: *url* do servidor que o servidor do *Twitter* irá requisitar quando realizar com sucesso a autenticação de um usuário;
- ***callback***: função que será chamada recebendo as informações do usuário autenticado, cabendo ao desenvolvedor fazer o que quiser com as informações, como por exemplo, salvar em um banco de dados.
- ***callback_render***: função que será chamada para receber a função que realiza a resposta da requisição, permitindo que o desenvolvedor realize a resposta da maneira que desejar.

Finalizando a demonstração da implementação da funcionalidade de autenticação, resta demonstrar de forma gráfica o fluxo realizado no processo de autenticação. A demonstração encontra-se exposta na seguinte figura:

Figura 10: Diagrama de Sequência da Autenticação no *Twitter*.

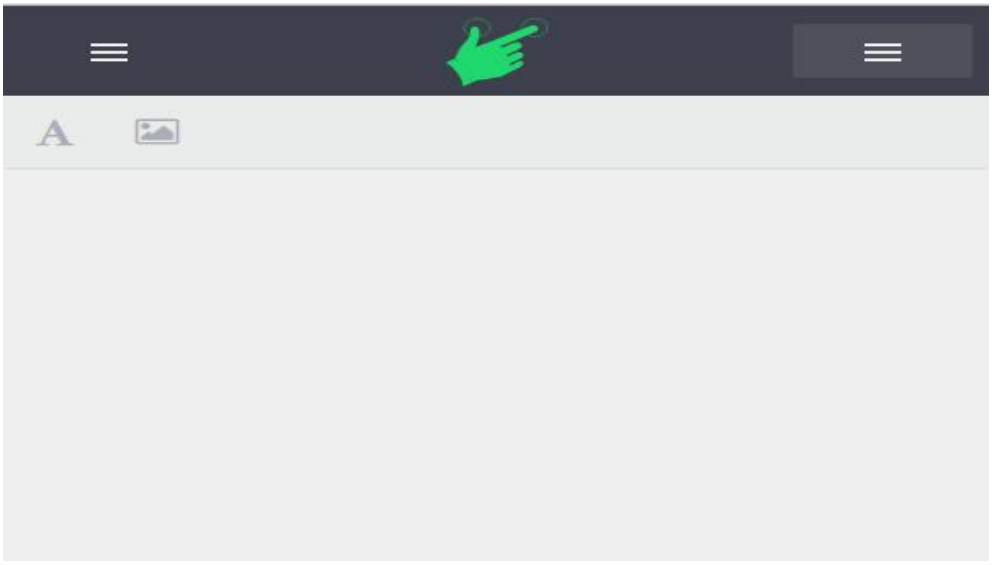


Fonte: Do autor.

3.5.3 Inserção de Texto

A inserção de texto na aplicação *Pinch Notes* é feita através da área livre, composta em um quadro branco sem limites onde o usuário pode inserir textos e imagens sem delimitações de posição. A figura exposta a seguir exemplifica o conceito de área livre:

Figura 11 – Área livre.



Fonte: Do autor.

Acima da área livre, existe uma barra com o botão de inserção de texto e o botão de inserção de imagem. Para inserir um texto na área livre, o usuário deve arrastar o botão até a posição da área livre em que ele deseja inserir o texto. O mesmo procedimento ocorre para adição de imagens.

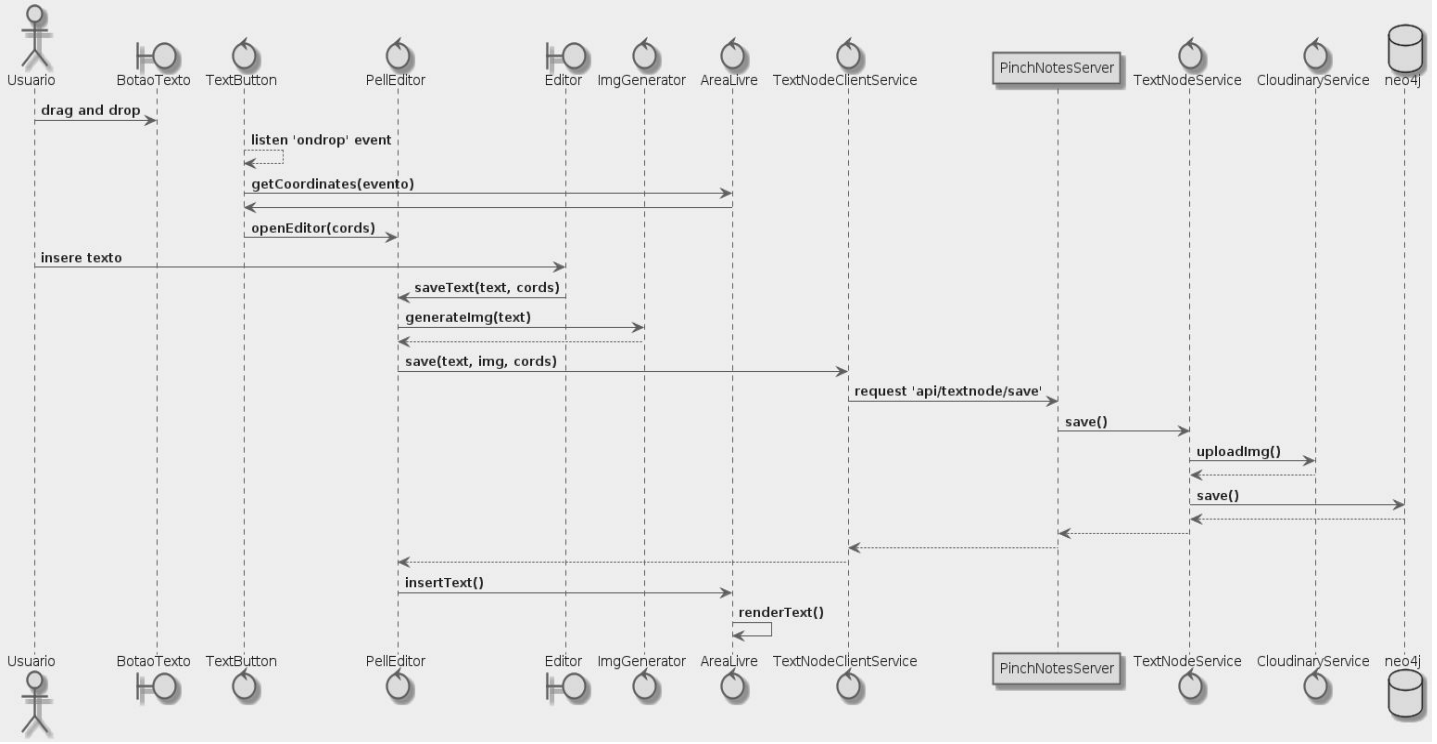
Ao mover o botão de texto para a posição desejada da área livre, a aplicação exibe o editor de texto, através do qual o usuário pode digitar texto diretamente, ou apenas copiar o texto de algum outro documento ou de alguma pagina da internet.

No caso de cópia de texto de alguma outra fonte, a aplicação se certifica de manter a formatação original, até mesmo quando o texto é renderizado na área livre. Essa funcionalidade é atingida através da biblioteca *Pell*, que fornece um editor de

texto WYSIWYG desenvolvido com *html, css e javascript* puro sem dependências. Além de fornecer de maneira fácil as funcionalidades que todo editor deve possuir, o *Pell* possui essa poderosa funcionalidade, em que é possível manter a formatação do texto original copiado. Portanto, esse fator foi decisivo na escolha desta biblioteca.

Porém, para ser usado pela aplicação *Pinch Notes*, foi necessário adaptar sua lógica para inserção de realces no texto, e, para isso, foram desenvolvidas diretivas do *angularjs* que decoram o *Pell*, integrando suas funcionalidades com o ciclo de vida dos componentes *angularjs*. A próxima figura ilustra o fluxo de operações executadas para a inserção de texto, através do diagrama de sequência:

Figura 12 – Inserção de texto.



Fonte: Do autor.

Como exemplificado no diagrama acima, a diretiva que implementa o botão de inserção de texto, escuta eventos de *drag and drop*, de modo que quando o *drop* for acionado; ou seja, o botão for inserido na área livre, a diretiva do botão se

comunica, assim, com a área livre através de um evento, pedindo para fornecer as coordenadas da posição na área livre.

É fato que o evento de *drag and drop* fornece a coordenada onde o botão foi deixado. Porém, é uma coordenada absoluta em relação ao *viewport* do navegador. Portanto, é necessário converter essa coordenada para a coordenada específica da área livre, que pode ter sofrido alterações, já que a área livre é infinita e responde a operações de *zoom*.

Quando a área livre responde enviando a coordenada real da posição, a diretiva *TextButton* envia um evento para que a diretiva *PellEditor* seja aberta e o usuário possa inserir seu texto.

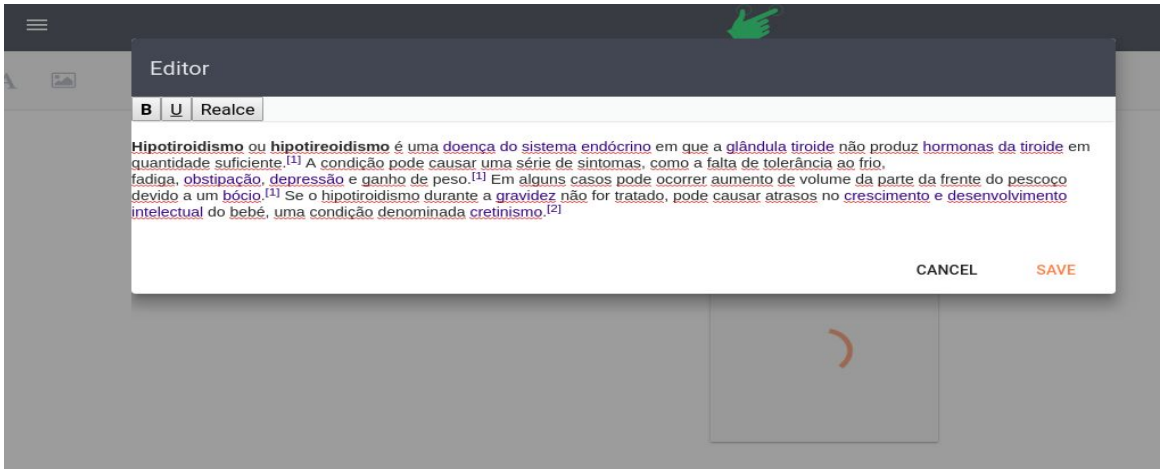
Ao clicar no botão salvar que a diretiva *PellEditor* possui, a diretiva se comunica com um serviço chamado *ImgGenerator*, para gerar uma imagem *png* do texto inserido. Essa imagem será usada para mostrar uma visualização prévia do texto no modo mapa mental da aplicação, de forma que o usuário não precisa entrar no modo área livre para visualizar aspectos do texto relacionados aos “nós” exibidos no mapa.

Após gerada a imagem de visualização prévia do texto, a diretiva *PellEditor* se comunica com o serviço *TextNodeClientService*, para que este realize a requisição de criação do “nó” no serviço exposto pelo servidor a partir do *endpoint* “*api/textnode/save*”.

Em seguida, o servidor realiza uma chamada ao serviço *TextNodeService*, para que ele se encarregue de salvar o texto e realize o envio da imagem de visualização através do serviço *CloudinaryService* para o serviço *Cloudinary*. Caso a operação se realize sem erros, o servidor responde a requisição com os dados do texto e seu *id* no banco de dados.

A diretiva *PellEditor* recebe a resposta e então se comunica com a diretiva *AreaLivre*, que se encarrega de renderizar o texto na posição escolhida. A figura disponível na próxima lauda exemplifica como é inserido um texto na área livre através do editor *Pell*:

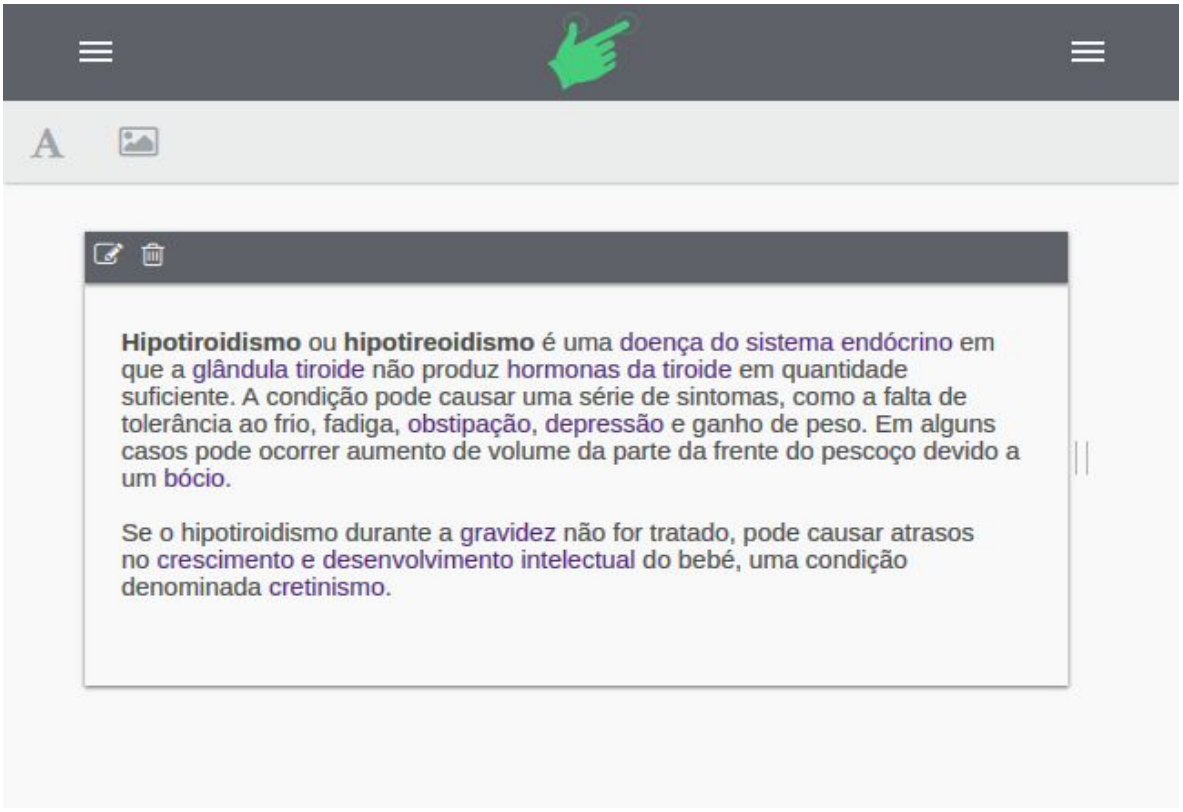
Figura 13 – Editor de texto *Pell.*



Fonte: Do autor.

Já a próxima figura demonstra como o texto é renderizado na área livre:

Figura 14 – Texto na área livre.



Fonte: Do autor.

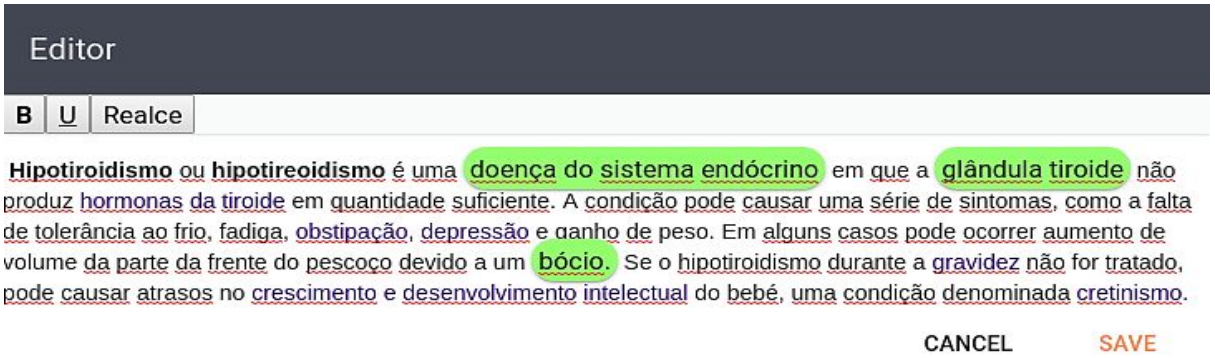
3.5.4 Inserção de Realce em Texto

Os realces de trechos do texto são realizados também a partir do editor de texto *Pell*. Se o texto já estiver salvo na área livre, é possível adicionar realces no texto através do botão de editar visível acima do texto renderizado na área livre, o botão apenas envia um evento para que a diretiva *PellEditor* mande exibir o editor com o conteúdo do texto renderizado na área livre.

O editor de texto possui o botão de realce que, quando clicado, executa a operação responsável por realçar o texto selecionado. Para transformar o realce em um elemento inteligente entre o texto capaz de responder a eventos, foi necessário construir uma diretiva que engloba o trecho do texto selecionado. Essa diretiva responde a eventos de *zoom* e *click*, para que seja exibida a camada da área livre relacionada ao realce de texto, e, por isso, foi necessário decorar o editor *Pell*. Felizmente, a biblioteca é construída de forma orientada a eventos, e, quando sua função de realce é executada, ela dispara um evento de realce que pode ser capturado para adicionar comportamento específico.

O comportamento específico adicionado foi o responsável por editar o trecho de texto selecionado e colocar em volta dele a diretiva *TextMarker*. Quando o texto é finalmente renderizado na área livre, ele passa por um processo de compilação manual para identificar possíveis elementos do *angularjs* embutidos no código *html* do texto. Portanto, a diretiva *TextMarker* é compilada e renderizada junto com o texto. A figura abaixo exibe o realce de texto:

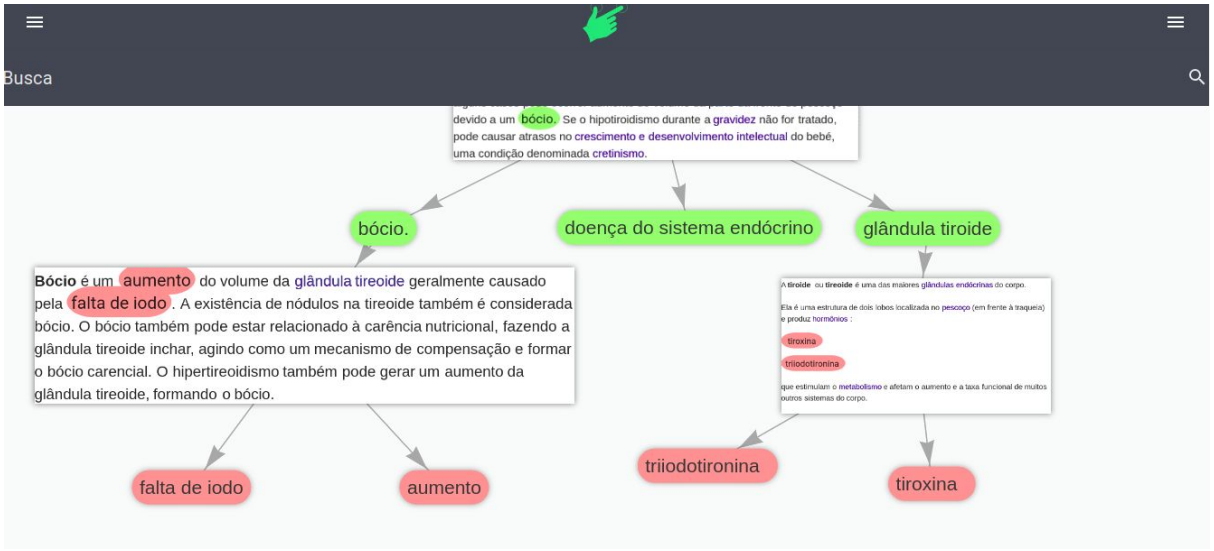
Figura 15 – Realces de texto.



Fonte: Do autor.

A próxima figura, por sua vez, exibe como os realces em textos são renderizados no modo mapa mental:

Figura 16 – Modo Mapa Mental.

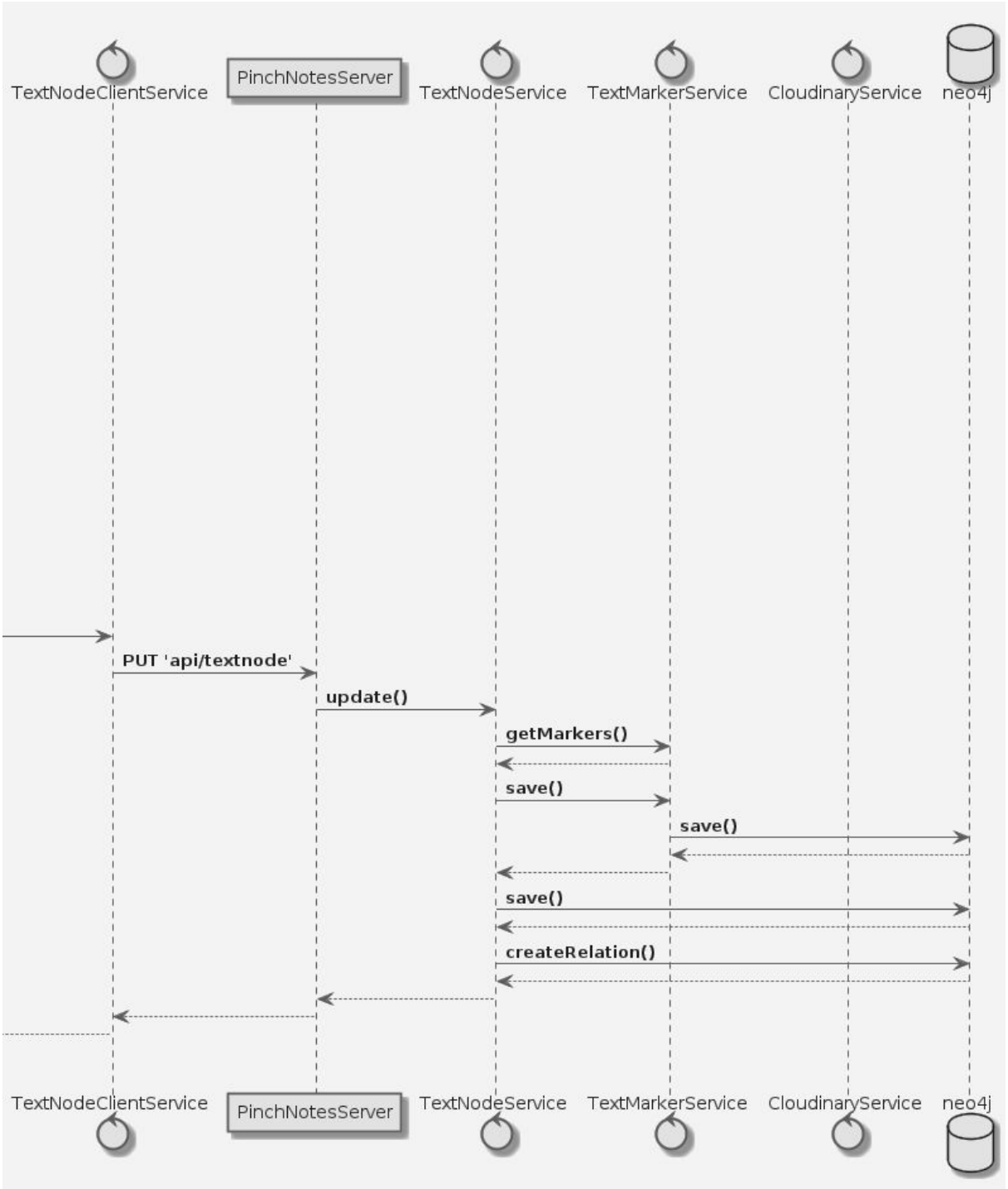


Fonte: Do autor.

Como a figura acima demonstra, os realces de textos, no modo mapa mental, foram transformados em “nós” do mapa mental, enquanto que os textos são exibidos como as imagens de visualização prévia.

A sequência de operações para efetivar a criação do realce de texto é semelhante à sequência para criação de um texto demonstrada na figura 12. A diferença ocorre, principalmente, no servidor que, antes de salvar o texto no banco de dados, realiza uma busca pela diretiva *TextMarker* no conteúdo do texto, para então salvar os realces de textos isoladamente. Após salvar os realces de textos, o texto é salvo, e, em seguida, são criados os relacionamentos entre os textos e seus realces. Essa operação pode ser verificada na figura 17, disponível na próxima lauda, que ilustra a parte do diagrama de sequência onde as operações do servidor ocorrem:

Figura 17 – Diagrama de Sequência de Inserção de Realce de Texto.



Fonte: Do autor.

3.5.5 Modo Mapa Mental

Como representado na figura 16, é possível visualizar, no modo mapa mental, todas as informações inseridas na área livre e suas camadas. Os textos,

realces de texto e imagens são exibidos como “nós” do mapa, e seus relacionamentos são exibidos como setas. Um realce de texto tem relacionamentos para outros “nós” de texto, que possuem relacionamento para outros realces, e assim por diante.

A renderização dos dados inseridos na área livre em estrutura de grafos é possível porque os dados já são salvos em estrutura de grafos como explicado na seção 3.3. O banco *neo4j* possui papel importante nesse processo porque através dele é possível realizar consultas que retornam os resultados em um *json* com estrutura compatível à estrutura necessária para a renderização do mapa através da biblioteca *Vis.js*, que é correspondente a uma biblioteca de visualização dinâmica, baseada em navegador. Nesse sentido, conforme expresso em o site *Vis.js*, a biblioteca é modelada com objetivo de ser fácil de usar, com capacidade para tratar grandes quantidades de dados dinâmicos e permitir a manipulação e interação desses dados.

Quando o banco *neo4j* recebe uma *query* especificando o retorno como sendo do tipo *path*, ele monta os resultados em um *json* com a estrutura exemplificada na figura exposta abaixo:

Figura 18 – Estrutura da Resposta do Banco.


```
{
    "results": [{
        "columns": ["n"],
        "data": [{
            "row": [{
                "uid": "1",
                "date": "29-03-15",
                "value": "10",
                "stat_id": "3"
            }]
        }, {
            "row": [{
                "uid": "1",
                "date": "24-04-15",
                "value": "1",
                "stat_id": "4"
            }]
        }]
    }],
    "errors": []
}
```

Fonte: Extraída do livro *Beginning Neo4j.*

O JSON retornado consiste do *array results* e do *array errors*. Nesse caso, existe apenas um item no *array results,* porque apenas uma instrução foi executada na transação. Contudo, múltiplas instruções resultam em múltiplos itens no *array results*. Cada item do *array data* é um nó do grafo, e cada item dentro do *array row* possui as propriedades que o "nó" possui. Se a instrução executada informasse para retornar relacionamentos em vez de "nós", então cada item dentro de *results* seria um relacionamento.

Para obter os resultados em formato JSON, é necessário executar consultas que o banco *neo4j* entenda. As consultas mais importantes utilizadas pela aplicação no modo mapa mental são duas: uma para buscar todos "nós" e relacionamentos de um mapa; e outra para pesquisar por todos os "nós" e relacionamentos de um mapa que possuem conteúdo relacionado aos termos de busca do usuário. A primeira consulta é demonstrada na figura abaixo:

Figura 19 – Consulta Mapa Mental.

```
function getGraph(originId, fromTextNode) {
    const query = `MATCH path = (tn)-[:HAS*]->(node) WHERE id(tn)=${originId} RETURN path;`;
    return startTransaction()
    .then((transactionId) => addStatement(transactionId, query, null, true))
    .then((response) => commit(transactionId));
};
```

Fonte: Do autor.

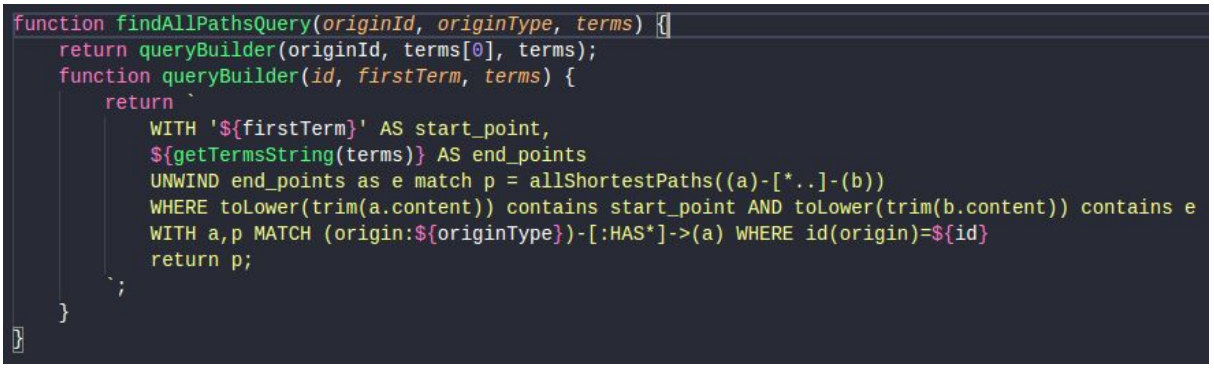
A função *getGraph* tem como objetivo buscar todos os "nós" que estão relacionados ao "nó" inicial do mapa que possui *id* igual ao argumento *originId*. Primeiro, é preciso montar a *query*, que traduzida em bom português significa: seguindo o padrão onde um nó *"tn"* tem relacionamento de qualquer nível com outro "nó" qualquer "*node*", procure os "nós" seguindo esse padrão onde *"tn"* possui *id* igual ao *id* do argumento *originId*.

Em seguida, a chamada da função *startTransaction* retorna uma *Promise*, que, ao ser encadeada, chama a função *addStatement*, que retorna outra *Promise*, que, por fim, chama a função *commit* que retorna uma *Promise*, que será resolvida quando o banco de dados concluir a operação. Por fim, a função *getGraph* retorna

uma *Promise* que será resolvida quando os dados forem salvos no banco de dados. A resolução da *Promise* fornecerá o resultado da instrução, que, neste caso, agrega todos os “nós” e relacionamentos encontrados seguindo a estrutura demonstrada anteriormente na figura 18.

A segunda consulta, demonstrada na figura logo a seguir, é responsável por exibir todos os “nós” que possuem o conteúdo dos termos de busca fornecidos pelo usuário, bem como os “nós” relacionados a estes que estejam dentro do conjunto de “nós” que possuem os termos de busca.

Figura 20 – Consulta Mapa Mental 2.



Fonte: Do autor.

Desta vez, a *query* retornada pela função *queryBuilder* utiliza várias palavras chaves da linguagem de *query* do *neo4j* chamada *cypher*. Começa definindo o argumento *firstTerm* como uma variável que será utilizada por outras partes da query. Em seguida, são criados nomes para as variáveis resultantes dos termos de busca, como por exemplo *start_point* e *end_points*.

Após isso, o termo UNWIND é utilizado para tratar os termos de busca encontrados no array *end_points* como um conjunto de linhas reconhecido pelo banco de dados. O padrão a ser utilizado pela *query* é definido pela palavra-chave MATCH. Este padrão é combinado com a utilização de uma função interna do banco *neo4j* chamada *allShortestPaths*, que é responsável por encontrar todos os menores caminhos de relacionamentos entre os “nós” encontrados pela consulta.

O termo WHERE serve para especificar as condições da consulta, delimitando os resultados somente aos “nós” que possuem em seu conteúdo os termos de busca definidos pelo usuário. Após a instrução WHERE, o termo MATCH é utilizado mais uma vez para definir padrão que especifica o ponto de origem da busca.

Por fim, a *query* sinaliza o retorno do conjunto de “nós” e relacionamentos que satisfazem as condições da consulta.

4 CONSIDERAÇÕES FINAIS

Através da apreciação de pesquisas envolvendo os métodos de estudo aplicados por estudantes e professores, e também comparando com estudos e testes envolvendo a aplicação de mapas mentais, foi entendido que o uso de mapas mentais, para conteúdos complexos e com muitos níveis de detalhamento, pode melhorar e acelerar o entendimento por parte de estudantes que já conhecem a técnica de mapeamento mental.

Visto que a técnica não é muito eficiente para estudantes que não dominam o mapeamento mental, este trabalho propôs uma ferramenta para diminuir a lacuna entre as práticas já cotidianas de estudo e a abordagem de mapeamento mental, através da inserção comum de textos e realces de textos que, em seguida, são transformados em mapas mentais.

Para o desenvolvimento da ferramenta *Pinch Notes*, que visa preencher esta lacuna, foi necessário entender qual modelagem e banco de dados seria melhor apropriada para representar os dados. Inicialmente, foi proposto o uso de um banco de dados relacional. Porém, foi perceptível a dificuldade encontrada para a realização de inserção de dados, e de consultas avançadas, como, por exemplo, a consulta que retorna todos os “nós” de um mapa que possuem determinado conteúdo e, além disso, mostrar o menor caminho entre estes relacionamentos.

Em razão das dificuldades mencionadas, foi decidido, após a implementação de protótipos utilizando o banco *neo4j*, a utilização de um modelo de dados orientado a grafos que demonstrou melhor adequação de encaixamento para a aplicação.

Para o desenvolvimento da aplicação *web*, foi decidido optar por *javascript*, já que sua infraestrutura é completamente orientada a eventos e permite o uso de funcionalidades poderosas demonstradas na seção 2.4. Além disso, a comunidade que mantém o desenvolvimento da linguagem *javascript* é muito forte e presente, incluindo a disponibilidade de diversas bibliotecas que possibilitaram o desenvolvimento deste trabalho.

Como foi identificado que a aplicação *Pinch Notes* seria extensivamente utilizada para a escrita e leitura de dados, sem alto teor de processamento do lado do servidor, já que a aplicação *web* do lado do cliente executa a maior parte do esforço computacional, tornou-se viável a utilização de *nodejs*, para a implementação do servidor. Já que o servidor não precisa manter explicitamente *threads*, sessões, e conexões abertas com o cliente, o servidor *nodejs* executa muito bem sua função de apenas terceirizar ações para o banco de dados e outros serviços externos, mantendo sua resposta rápida, já que todo seu mecanismo se dá por meio de ações assíncronas.

Através do desenvolvimento deste trabalho, também foram aproveitadas as oportunidades para o desenvolvimento de bibliotecas próprias e sua disponibilização para o público, como foi o caso do *Node-Twitter-Signin* e outras pequenas bibliotecas para manipulação de componentes *web*.

Devido a implementação do trabalho não ocorrer através da colaboração mútua entre vários desenvolvedores, não tornou-se necessário a utilização de processo de desenvolvimento formal e bem delimitado. Porém, à medida em que projeto tomou escopo maior, dividido através de bibliotecas próprias e de terceiros, foi necessário desenvolver a aplicação seguindo as melhores práticas do controle de versão *git*, criando *branchs* para cada versão de cada módulo e biblioteca, e apenas mesclando estas *branchs* à versão principal do projeto, quando já estivessem testadas e funcionando sob variadas circunstâncias.

O desenvolvimento do projeto também seguiu a abordagem de pequenas entregas, porém especificamente para a entrega de pequenos módulos funcionais, como, por exemplo, o módulo do editor de texto que foi o primeiro a ser entregue. Em seguida, o módulo de área livre, que mantém comunicação com o módulo do editor de texto através de uma interface simples que oculta seus detalhes.

Portanto, através do desenvolvimento deste trabalho, foi obtido aprendizado concreto para o desenvolvimento de aplicações complexas divididas em módulos e componentes com interfaces bem definidas.

Por fim, o desenvolvimento do *Pinch Notes* demonstrou que o objetivo da ferramenta, no sentido de diminuir a lacuna entre o estudante comum e a técnica de

mapas mentais, foi atingido, pois a própria aplicação constrói o mapa mental, a partir dos textos e realces inseridos pelo próprio usuário. Porém, percebe-se que ainda há muitos caminhos a serem trilhados para tornar a aplicação disponível ao público, visto que foi testada por usuários reais poucas vezes.

Entretanto, o *Pinch Notes* mostra ter potencial para conquistar os usuários que não possuem afinidade com mapeamento mental, haja vista ainda ser uma abordagem não utilizada por ferramentas de mapeamento mental disponíveis no mercado, e, portanto, é possível o aperfeiçoamento das ideias e funcionalidades desenvolvidas neste trabalho para um nível maior de usabilidade, disponibilizando a compatibilidade para dispositivos móveis; aumentando, assim, a interatividade em tempo real.

REFERÊNCIAS

BUZAN, Tony. **The Mind Map Book** – How to Use Radiant Thinking to Maximize Your Brain's Untapped Potential. Plume Books: New York (USA), 1996.

BRANDNER, Raphaela. The top 7 reasons not to mind map. Disponível em <http://www.mindmeister.com/blog/2013/10/03/the-top-7-reasons-not-to-mind-map/>. Acesso em 02 fev. 2016.

CHIK, Vincent; PLIMMER, Beryl; HOSKING, John. **Intelligent Mind-mapping.** In: **ACM Digital Library**, 2007. Disponível em: http://dl.acm.org/. Acesso em: 04 fev. 2016.

DUNLOSKY, John; RAWSON, Katherine A.; MARSH, Elizabeth J.; NATHAN, Mitchell J.; WILLINGHAM, Daniel T.. Improving Students' Learning With Effective Learning Techniques: Promising Directions From Cognitive and Educational Psychology. **Psychological Science in the Public Interest**, v.14, n.1, p. 4–58, 2013.

FASTE, Haakon; LIN, Honray. The Untapped Promise of Digital Mind Maps. In: **ACM Digital Library.** 2012. Disponível em: http://dl.acm.org/. Acesso em: 04 fev. 2016.

FLANAGAN, David. **JavaScript: The Definitive Guide** - Activate Your Web Pages. O'Reilly Media: Sebastopol (USA), 2011.

FREEMAN, Eric T; ROBSON, Elisabeth. **Head First JavaScript Programming.** O'Reilly Media: Sebastopol (USA), 2014.

HE, Feijuan; WU, Bei; MIAO, Xianglin; YAO, Siyu. **Using Mind Map as Learning Tool in "Data Structure" Teaching** In: **ACM Digital Library**, 2007. Disponível em: http://dl.acm.org/. Acesso em: 04 fev. 2016.

MCCREARY, Dan; KELLY, Ann. **Making Sense of NoSQL**. A guide for managers and the rest of us. Manning Publications: Connecticut (USA), 2013.

NAST, Jamie. **Idea Mapping** – How to Access Your Hidden Brain Power, Learn Faster, Remember More, and Achieve Success in Business. John Wiley & Sons: New Jersey, 2006.

PROCESS. Addison-Wesley: London (England), 2012.

RAJ, Sonal. **Neo4j High Performance**. Packt Publishing: Birmingham (England), 2015.

RHODES, John S. **Mind Maps:** How to Improve Memory, Write Smarter, Plan Better, Think Faster, and Make More Money. JJ Fast Publishing, LLC, 2013.

SITE MINDMEISTER. Disponível em: www.mindmeister.com. Acesso em 02 fev. 2016.

SITE BIBLIOTECA vis. js. Disponível em: http://visjs.org/. Acesso em 02 fev. 2016.

SITE GITHUB. Disponível em: https://github.com/vmvini/node-twitter-signin. (Página pessoal).

TAYLOR, Michael. **Mind Maps:** Quicker Notes, Better Memory, and Improved Learning 3.0. CreateSpace Independent Publishing Platform, 2014.

TEIXEIRA, Pedro. **Professional Node.js**: Building Javascript Based Scalable Software.

TEXT2MINDMAP. Disponível em <https://www.text2mindmap.com/> Acesso em 02 fev. 2016.

THE BLOCKHEAD. **Mapeamento Mental**: guia passo a passo para iniciantes em criação de mapas mentais. Editora Babelcube, 2016. (The Blockhead Series).

WROX: Birmingham (England), 2012.

VUKOTIC, Aleksa. **Neo4j in Action**. Manning Publications: Connecticut (USA), 2014.