UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CAMPUS PAU DOS FERROS
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA
CURSO DE GRADUAÇÃO EM BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

FRANCISCO LUCAS LIMA NUNES

DASHCARE: UM DASHBOARD PARA ANÁLISE DOS ACIDENTES NAS RODOVIAS FEDERAIS

PAU DOS FERROS – RN
2020

FRANCISCO LUCAS LIMA NUNES

DASHCARE: UM DASHBOARD PARA ANÁLISE DOS ACIDENTES NAS RODOVIAS FEDERAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Bacharelado em Tecnologia da Informação do Departamento de Engenharias e Tecnologia da Universidade Federal Rural Do Semi-Árido, como requisito parcial à obtenção do grau de bacharel em Bacharelado em Tecnologia da Informação.

Orientadora: Profa. Dra. Samara Martins Nascimento

Co-Orientador: Prof. Me. André Luiz Sena da Rocha

PAU DOS FERROS – RN

2020

N972d Nunes, Francisco Lucas Lima.
Dashcare: um dashboard para análise dos acidentes nas rodovias federais / Francisco Lucas Lima Nunes. - 2020.
55 f. : il.

Orientadora: Samara Martins Nascimento.
Coorientador: André Luiz Sena da Rocha.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Tecnologia da Informação, 2020.

1. Dados abertos. 2. Rodovia federais. 3. Visualização. 4. Dashboard. I. Nascimento, Samara Martins, orient. II. Rocha, André Luiz Sena da, co-orient. III. Título.

FRANCISCO LUCAS LIMA NUNES

DASHCARE: UM DASHBOARD PARA ANÁLISE DOS ACIDENTES NAS RODOVIAS FEDERAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Bacharelado em Tecnologia da Informação do Departamento de Engenharias e Tecnologia da Universidade Federal Rural Do Semi-Árido, como requisito parcial à obtenção do grau de bacharel em Bacharelado em Tecnologia da Informação.

Aprovada em: 06 / 02 / 2020

BANCA EXAMINADORA

Samara Martins Nascimento

Profa. Dra. Samara Martins Nascimento
(Orientadora)– UFERSA
Presidente

Ádller de Oliveira Guimarães

Prof. Dr. Ádller de Oliveira Guimarães – UFERSA
Membro Examinador

Prof. Dr. Claudio Andrés Callejas Olguín – UFERSA
Membro Examinador

Aos meus pais

AGRADECIMENTOS

Primeiro, agradeço a Deus, o autor da minha vida, que possibilitou o meu encontro com tantas pessoas que me agregaram ao longo do curso.

Agradeço a minha mãe Elizabete, que me deu força e coragem para continuar essa longa caminhada. Agradeço ao meu pai Arimatéia, que me incentivou e me manteve na linha. Agradeço ao meu irmão Victor, que me deu suporte sempre que precisei.

A Adby e Ernandes, os melhores copanheiros de projetos desde o segundo período, aprendi muito com vocês. A Samuel, por me ajudar a pensar fora da caixa e ter pronuncioado os melhores diálogos. A Malu, pelo forte companheirismo desde a primeira disciplina.

Agradeço a Vinícius e Antônio Neto, meus amigos. Também a todos os demais amigos e colegas da UFERSA, que estiveram presentes em todo momento, pelos incentivos, reconfortos e alegrias.

Agradeço a professora Samara e ao professor André, que contribuíram intelectualmente e me guiaram a conclusão deste trabalho. Assim como aos demais colaboladores da instituição.

Agradeço a todas as pessoas que fazem a SIASP, que nessa reta final, propiciaram a chance de crescimento profissional.

– Professor, isso é real, ou está acontecendo somente na minha mente?

– É claro que está acontecendo na sua mente, Harry, mas por que isso significa que não é real?

(J. K. Rowling)

RESUMO

O processo de mobilidade nas Rodovias Federais (RFs) do Brasil, enfrenta o problema de um grande número de acidentes anualmente, trazendo transtornos a sociedade e onerando sobre os gastos com saúde pública. Possíveis soluções para esse problema requerem medidas tomadas baseadas em informações precisas e claras, o que só é possível a partir da análise e tratamento de dados. Nesse contexto, é de responsabilidade da Polícia Rodoviária Federal (PRF) fazer o acompanhamento dos acidentes e obter dados acerca desses, como também disbonibilizá-los de forma pública. Os Dados Abertos (DAs) são dados de livre utilização, que colaboram com o fomento da inovação tecnológica. Desse modo, objetivo deste estudo é o desenvolvimento de uma ferramenta capaz de processar e fornecer um meio de visualização clara das informações contidas nos DAs da PRF. Para tanto, o estudo aborda uma metodologia de desenvolvimento Web orientada ao uso de dados de forma eficiente. Como resultado, foi obtido um *dashboard* extensível que possui três módulos, sendo o principal a de visualização que é capaz de mostrar informações com diferentes níveis de granularidade de acordo com ações do usuário.

Palavras-chave: Dados Abertos. Rodovia Federais. Visualização. Dashboard.

ABSTRACT

The mobility process on Federal Highways (FHs) in Brazil, faces the problem of a large number of accidents annually, which disturb society and burden public health spending. Possible solutions to this problem require measures taken based on accurate and clear information, what is only possible from the analysis and treatment of data. In this context, it is the responsibility of the Federal Highway Police (FHP) to monitor accidents and collect data about them, as well as making them publicly available. Open Data (OD) are free data that collaborate with the promotion of technological innovation. Thus, the objective of this study is the development of a tool capable of processing and providing a clear visualization of the information contained in the FHP's OD. To this end, the study addresses a method of Web development focused on the use of data efficiently. As a result, an extensible dashboard was obtained that has three modules, the main one being the visualization that is capable of showing information with different levels of granularity according to the user's actions.

Keywords: Open Data. Federal Highways. Visualization. Dashboard.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BD	Banco de Dados
BNS	Banco de dados NoSQL
CAP	Consistency, Availability and Partition Tolerance
CN	Conputação em Nuvem
CRUD	Create, Read, Update and Delete
CSS	Cascading Style Sheets
DA	Dados Abertos
ETC	Extração, Transformação e Carregamento
ETL	Extract, Transform and Load
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation
ORM	Object Relational Mapping
PRF	Policía Rodoviário Federal
SPA	Single Page Aplication
SQL	Structed Query Language
UI	User Interface

SUMÁRIO

1 INTRODUÇÃO

O meio de transporte rodoviário configura-se como o maior e mais importante meio de locomoção, que serve desde fins de transporte pessoal até fins econômicos (SCHMIDT *et al.*, 2011). Um levantamento do Departamento Nacional de Trânsito (DENATRAN) mostra que até agosto de 2019, são aproximadamente 55 milhões de automóveis registrados no Brasil (MINISTÉRIO DA INFRAESTRUTURA, 2019). Frente a isso, é válida a ótica de que, dado o inerente envolvimento de vida humana na locomoção rodoviária, deve-se reforçar a ideia de aprimoramento e manutenção da segurança no trânsito. Nesse cenário, as Rodovias Federais (RFs), atuam como uma das principais vias para deslocamento dos automóveis em todo o país. Em 2018, essas já abrangiam um total de, aproximadamente, 75 mil quilômetros de comprimento de acordo com o Ministério da Infraestrutura (2019) e é de responsabilidade da Polícia Rodoviária Federal (PRF) o levatamento das causas e dados. Além disso, a PRF também deve realizar o atendimento e salvamento das vítimas em acidentes que ocorram em pontos específicos das RFs (BRASIL, 2019).

Grande parte dos registros sobre os dados coletados a partir dos levantamentos realizados pela PRF são armazenados e disponibilizados em uma base de Dados Abertos (DAs) do Governo Federal (POLÍCIA RODOVIÁRIA FEDERAL, 2019). Os DAs são definidos como dados institucionais sem restrições ou licença, disponibilizados publicamente via internet e fomentam inovação aliada ao crescimento econômico, quando aplicadas as devidas Tecnologias da Informação (TI) (ZUIDERWIJK *et al.*, 2014).

Equipamentos, como dispositivos de GPS integrados aos veículos ou de telefonia móvel, podem ser usados para gerar informações contínuas sobre o deslocamento de objetos móveis (ZHENG *et al.*, 2013). Dentro desse contexto, é válido citar o *Big Data*, que corresponde a uma estratégia computacional que lida, em suma, com a variabilidade dos dados e seu crescimento escalável, aliado à velocidade na mudança das informações. Os algoritmos, com base nos conceitos de *Big Data*, permitem realizar análises sobre os dados de forma a obter resultados de medição para o processo de tomadas de decisões (CHEN *et al.*, 2013).

O presente trabalho tem como objetivo, propor o desenvolvimento do *DASHCARE: Um Dashboard para Análise dos Acidentes nas Rodovias Federais*, com tecnologias Web e um banco de dados NoSQL, para monitorar, analisar e exibir resultados de medições, que mudam continuamente, dado o recebimento de *streams*. Ao final deste trabalho, espera-se obter uma versão estável do *dashboard* proposto, que possa servir como auxílio à informação e visualização

semântica dos DAs da PRF, tanto para entidades que tomam decisões estratégicas como para a entidades que necessitam de uma análise sobre tais dados.

1.1 PROBLEMÁTICA

Dados divulgados pelo G1 (2018), revelam que, apesar do número total de acidentes ter diminuido ao longo do intervalo entre os anos de 2007 a 2017, o índice de ocorrências continua alto, e que ainda nesse intervalo, houve um pico de acidentes em 2011, com cerca de 192 mil casos. No ano de 2014, foi estimado que cada acidente em uma RF custou ao país a média de R$ 75 mil, valor que aumenta significativamente no que se refere a casos de acidente acompanhado de óbito (ANDRADE; ANTUNES, 2019). Mais adiante, no ano de 2017, foi contabilizado aproximadamente 89 mil ocorrências em RFs, dentre as quais, cerca de 6 mil resultaram em fatalidade (G1, 2018). Esses dados demonstram que, apesar da considerável queda do número de acidentes nas RFs, ainda existe um alto número de acidentes que transtorna o meio social, como também onera sobre os gastos com saúde pública no país.

Com a aplicação da Tecnologia da Informação (TI), há indícios da construção de sistemas, que garantem uma essencial aplicabilidade de soluções informatizadas de análise, organização e apresentação dos dados em tempo real (KOPETZ, 2011). Sendo assim, é possível, com a solução adequada, gerenciar e validar as informações dos dados levantados pela PRF das RFs, e ainda, podem servir como estratégias norteadoras para tomada de melhores decisões, que buscam aumento na segurança das estradas nacionais.

Entretanto, a quantidade dos dados acumulados ao longo do tempo por órgãos como a PRF é ligeiramente grande. Por exemplo, se fossem convertidos em uma planilha, essa chegaria a conter centenas de milhares de linhas. E segundo Marquesone (2016), a grande limitação para a construção de sistemas com processamento em tempo real é o desempenho sobre vários dados. Dessa forma, é importante que eles sejam escaláveis, seja considerando o volume de dados, os usuários e/ou recursos.

Para garantir a construção de sistemas que computam *streams*[1] de dados continuamente, isto é, realizam o processamento a cada minuto ou a cada segundo, é comum a otimização do processamento por meio de programação paralela, que divide o processamento em várias máquinas utilizando uma abordagem distribuída (CHEN; ZHANG, 2014). Essa estratégia garante

[1] Segundo (AGGARWAL, 2007), *streams* de dados é a emissão e coleta de dados de modo contínuo, podendo ser gerados a partir de várias fontes distintas.

o processamento com baixo tempo de resposta e alta disponibilidade (MARQUESONE, 2016).

1.2 PROBLEMA DE PESQUISA

Um dos grandes problemas relacionados ao processamento de um grande volume de dados, está no quão custosa computacionalmente essa tarefa pode ser, especialmente tratando-se da extração de informação de um montante de dados que, a priori, parecem não revelar nada. O cenário de estudo deste trabalho possui similaridade com esse problema. A quantidade de dados disponibilizados pela PRF tornam a atividade de manipulação e compreensão desses dados algo não trivial. Dessa forma, esta pesquisa focou na criação de um *dashboard* flexível para monitoramento. Com isso, espera-se que os dados sejam obtidos e tratados de forma eficiente, e que os recursos visuais possam facilitar a compreensão das informações acerca desses.

Dentre os desafios presentes nesta proposta, está a necessidade de fornecer informações consideradas relevantes de forma clara e concisa. Também está a formulação de uma arquitetura onde o processamento das informações possa ser divido entre lado servidor e cliente. Tomando-se o exposto, é levantada a seguinte questão de pesquisa: *"A partir do conjunto de DAs da PRF, como criar um dashboard flexível e eficiente, que provenha informação clara e objetiva acerca desses dados, e também seja acessível a qualquer entidade interessada?"*

1.3 OBJETIVOS

Esta Seção apresenta o objetivo maior deste trabalho (Subseção 1.3.1) e especificidades para o alcance desse (Subseção 1.3.2).

1.3.1 Objetivo Geral

Este trabalho visa a construção de um *dashboard* para a plataforma Web, que realize o tratamento, relacionamento e mostre graficamente as informações a respeito dos DAs da PRF, que possa servir como aparato de auxilio a medidas de combate aos problemas de mobilidade existentes nas RFs do Brasil.

1.3.2 Objetivos Específicos

Tem-se como objetivos específicos:

a) Realizar um lenvatamento teórico para adquirir direcionamento necessário para todo o desenvolvimento intelectual deste trabalho;
b) Definir o escopo dos DAs a serem trabalhados;
c) Desenvolver um módulo de processamento e consulta dos DAs da PRF em servidor;
d) Desenvolver um módulo de visualização gráfica das informações a cerca dos DAs da PRF;
e) Discutir os resultados obtidos.

1.4 JUSTIFICATIVA

O progressivo avanço na área da TI possibilita a disseminação de bases de dados de múltiplas naturezas, dentre elas as de uso governamental. Entretanto, a análise de grandes quantidades de dados pelo ser humano se torna uma tarefa impraticável, sem o auxílio das ferramentas computacionais apropriadas. Assim, é imprescindível o desenvolvimento de soluções tecnológicas, capazes de ajudar o homem na criação e seleção de estratégias que maximizem o entendimento sobre os dados (GOLDSCHMIDT; PASSOS, 2005).

Os tipos de decisões orientadas a dados são as melhores. O relacionamento e interpretação de forma inteligente capacita os tomadores de decisões a cometerem ações com base em evidências, e não apenas na intuição. Assim, é recomendável a construção de ferramentas que auxiliem a análise, interpretação e relacionamento dos dados, podendo ser crucial no processo de tomada de decisões (MCAFEE *et al.*, 2012).

Diante do exposto, pode-se considerar como principal contribuição deste trabalho, portanto, o desenvolvimento e disponibilização de uma ferramenta capaz de oferecer uma área de visualização de informações acerca dos DAs da PRF de forma rápida, escalável, clara e concisa, como também, o oferecimento de análise estatística sobre essas. Sendo elas agrupadas pelos diferentes campos presentes no escopo dos dados. Com isso, espera-se que o eventual uso do *dashboard* proposto por alguma entidade interventora, possa gerar medidas que impactem de forma positiva o número de acidentes da RFs, a exemplo da diminuição de ocorrências e gasto com a saúde pública.

1.5 METODOLOGIA DA PESQUISA

Levando em consideração os objetivos definidos na Seção 1.3, este trabalho é caracterizado de forma descritiva. Segundo Gil (2002), uma pesquisa descritiva refere-se à descrição das características de fenômenos e estabelecimento de relações entre variáveis, que é marcada pela coleta sistêmica e padronizada de dados.

Para o alcance dos objetivos pré-estabelecidos, foi realizada a delimitação do campo de atuação deste estudo. Para tanto, a pesquisa se restringe ao estudo do desenvolvimento de um ferramenta para DAs da PRF.

Aplicando o escopo deste trabalho a anterior definição de pesquisa descritiva, pode-se traduzir o fenômeno como o índice de acidentes nas RFs e a relação entre váriaveis é traduzida como as informações presentes nos DAs levantados pela PRF.

Quanto à abordagem, este trabalho desenvolve-se de forma quantitativa, pois o resultado obtido, quantifica informações obtidas a partir de dados previamente levantadados (GIL, 2002). Vale salientar, que neste trabalho é excluida uma abordagem qualitativa, visto que, não é de seu propósito, fazer alguma conceituação subjetiva e nem restringir a interpretação sobre qualquer informação disposta com o resultado.

As discussões deste trabalho são principalmente de caracter teórico, compreendendo uma revisão das discuções e conceitos acerca do tema. O levantamento do arcabouço teórico deste trabalho ocorreu por meio de fontes secundárias já analisadas, como livros e artigos científicos. A busca por trabalhos científicos sobreveio de repositórios *online*.

1.6 ORGANIZAÇÃO DO TRABALHO

O presente estudo encontra-se organizado da seguinte forma: No Capítulo 2 é desenvolvido os conceitos básicos que servem de embasamento teórico para o entendimento deste trabaho; o Capítulo 3 mostra alguns trabalhos relacionado com a temática deste; o Capítulo 4 descreve quais tecnologia foram utilizadas para o desenvolvimento da solução proposta; no Capítulo 5 é conceituado os três módulos já desenvolvidos da ferramenta *DASHCARE*; no Capítulo 6 é apresentado e discutido os resultados alcançados com os métodos apresentados; e por fim, no Capítulo 7 são apresentadas as considerações finais e os trabalhos futuros.

1.7 CONCLUSÃO

Neste Capítulo, foi descrito o trabalho proposto e o panorama de utilização das RFs brasileiras. Também foi explanada uma visão geral sobre o quadro do grande número de acidentes que incidem ao longo da vasta extensão destas. A partir disso, foi definido a competência da PRF sobre o atendimento em locais de acidentes, bem como a disponibilização sem restrições dos dados levantados acerca desses. Ainda neste Capítulo, foi discutido a problemática da manipulação de grandes volumes de dados e como a informação extraída, a qual pode fornecer base sólida para o auxílio a tomada de decisão sobre o real problema dos acidentes na RFs.

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo, são apresentados conceitos e definições importantes empregados ao longo de todo o documento. Todos eles são considerados importantes para o entendimento deste trabalho. Assim, são discorridos conceitos acerca de Banco de Dados (Seção 2.1), *Big Data* (Seção 2.2), Banco de Dados NoSQL (Seção 2.3), esses sendo conceitos necessários para o entendimento sobre o foco de aplicação da ferramenta. E, por fim, a Seção 2.4 discorre sobre Sistemas de Tempo Real, que fornce a base sobre o propósito deste trabalho.

2.1 BANCO DE DADOS

Um dado é um fato, o resultado de uma medição, que pode ser registrado e possui significado implícito. O conjunto ou coleção desses dados de forma relacionada é chamado de Banco de Dados (BD) (ELMASRI *et al.*, 2005).

Delimitando o conceito, o BD é a representação de algo do mundo real (universo de discurso) e as mudanças ocorridas no universo de discurso refletem no BD. Os relacionamentos presentes no BD, precisam ser coerentes e ter um significado intrínseco, e todo o projeto de BD deve ser concebido para um propósito específico (ELMASRI *et al.*, 2005). Uma estrutura organizacional comum de BD convencionais, trata-se do modelo relacional, onde os dados são armazenados de forma linear em relações chamadas de tabelas e cada linha dessa tabela é chamada de tupla e cada coluna é chamado de campo (GILLENSON, 2006). Ademais, o estado de um BD relacional está relacionado ao estado das relações de suas tabelas e tuplas, que geralmente estão sujeitas a restrições sobre os dados, derivadas a partir do universo de discurso (ELMASRI *et al.*, 2005).

Na literatura, existem diferentes tipos de bancos de dados, a exemplo dos Relacionais, Objeto-Relacionais e NoSQL (ELMASRI *et al.*, 2005; SADALAGE; FOWLER, 2019). Diferentes infraestruturas foram criadas para suprir faltas dos Sistemas de Bancos de Dados anteriores e o tipo do dado foi um dos fatores determinantes para o surgimento de uma nova proposta. Enquanto os Bancos de Dados Relacionais manipulam tipos de dados estritamente alfanuméricos, os Bancos de Dados Objeto-Relacionais manipulam dados complexos (como tipos de dados geográficos) (ELMASRI *et al.*, 2005). No entanto, nenhum deles consegue ser escalável e processa em tempo real *streams* (SADALAGE; FOWLER, 2019). Por esse motivo, surgiram conceitos como *Big Data* e Bancos de Dados *NoSQL*, que serão apresentados nas próximas

Seções.

2.2 *BIG DATA*

Na literatura, o termo *Big Data* refere-se conceitualmente a grandes volumes de dados. De acordo com (CHEN *et al.*, 2013), não existe um consenso sobre uma definição única do que é *Big Data*. Ainda conforme (CHEN *et al.*, 2013), *Big Data* trata-se de conjuntos de dados que superam o processamento e capacidade de softwares comuns de capturar, gerenciar e processar dados dentro de um tempo tolerável. Já (BEGOLI; HOREY, 2012), argumentam que *Big Data* refere-se a coleta de grande volume de dados aplicados sob algoritmos capazes de analisar tais dados.

Uma descrição mais detalhada sobre *Big Data* é vista como de três termos: (i) Volume; (ii) Variedade; e (iii) Velocidade. Esses são os chamados "3Vs" (WANG; WANG; ALEXANDER, 2015; GANDOMI; HAIDER, 2015). De acordo com (GANDOMI; HAIDER, 2015), a simplificação de suas definições é:

1. Volume: trata-se de uma grande quantidade de dados gerados e posto em conjunto. A definição para "grande" pode variar por alguns fatores, tais como tempo e tipo dos dados;
2. Variedade: refere-se à heterogeneidade no conjunto de dados, cujos dados podem ser estruturados, semi-estruturados ou não estruturados;
3. Velocidade: trata-se do quão rápido os dados são produzidos, analisados e postos em ação. Processamento tradicional não é capaz de lidar com tantos dados, por isso a abertura para as técnicas de *Big Data*.

Quanto ao processamento de grandes conjuntos de dados, a Computação em Nuvem (CN) fornece flexibilidade na alocação de recursos e gerência do armazenamento de dados da aplicação para processamento (MARQUESONE, 2016). A replicação em hardware e expansão de sistemas oferecidos pela CN, provê suporte a algoritmos de programação paralela como *Map Reduce*, que realiza o processamento dos dados de forma distribuída (KAISLER *et al.*, 2013).

O *Map Reduce* é um paradigma de programação baseado em outro paradigma, chamado "Dividir para Conquistar". Nesse caso, o conjunto de dados é particionado, de forma balanceada, em conjuntos de dados menores e as informações são processadas considerando pequenos volumes de dados. Para isso acontecer, os conjuntos são passados por uma função *map*, que manipula todos os dados de um conjunto, e depois, são passados por um função *reduce*, que reduz a manipulação anteior apenas aos dados necessários (DEAN; GHEMAWAT, 2010).

2.3 BANCO DE DADOS NOSQL

Banco de dados NoSQL, acrônimo para *Not Only SQL*, ou "Não Somente SQL", são banco de dados com suporte a dados de tipos diversos e dispensam o uso de *Structured Query Language* (SQL) para sua manipulação. Bancos *NoSQL* permitem processamento focado em performance, confiabilidade e agilidade (MONIRUZZAMAN; HOSSAIN, 2013).

A infraestrutura *NoSQL* de persistência dos dados pode se apresentar nas formas de orientação a documentos, chave-valor, família de colunas e orientado a grafos (MCCREARY; KELLY, 2014). Dos tipos citados, este trabalho utilizará o banco Orientado a Documentos, que também corresponde ao tipo mais geral, flexível e disseminado. Esse modelo armazena coleções de documentos, em que cada documento pode ser visto como um objeto de identificador único (chave) e um conjunto de campos. Essa chave pode não ter utilidade, porém, o usuário consegue recuperar quase qualquer item de uma coleção consultando qualquer valor dentro de um documento (MCCREARY; KELLY, 2014).

2.3.1 Teorema *CAP*

A medida que um sistema ganha grande escalabilidade, as propriedades de consistência dos BDs Relacionais se tornam um empecilho ao desenvolvimento das aplicações, o que torna preferível distribuição horizontal. Entretanto, essa distribuição impede algumas principais funcionalidades (restrição dos dados e transparência) e exige tratamento a nível de aplicação (DIANA; GEROSA, 2010). O teorema CAP trata especialmente dessa situação.

O acrônimo CAP significa *Consistency* (Consistência), *Availability* (Disponibilidade) e *Partition Tolerance* (Tolerância à Partição). A Consistência indica que, a leitura de um item, logo após uma escrita nele, deve retornar o valor mais atual. Já a Disponibilidade é o acontecimento de resposta a toda requisição feita ao sistema em funcionamento. E, por fim, a Tolerância à Partição é o atendimento às requisições, mesmo diante do particionamento de um sistema devido a algum problema.

O teorema CAP, explica que apenas duas dessas propriedades ao mesmo tempo podem estar presentes em um sistema distribuído, que são a Disponibilidade e a Tolerância à Partição. Isso ocorre porque o particionamento dos dados em infraestruturas de bancos *NoSQL* considera a propriedade de Consistência eventual, já que o particionamento dos dados pode ocorrer de forma redundante. Assim, caso um dado seja atualizado em uma base específica, todas as outras

bases precisam ser atualizadas também, o que nem sempre é garantido. No entanto, se um dos nós[1] deixar de computar informações, poderão existir outros nós com o mesmo dado, tornando a informação sempre disponível (DIANA; GEROSA, 2010).

2.3.2 MongoDB

O MongoDB é um tipo de banco de dados NoSQL Orientado a Documentos. Seu armazenamento é semelhante a arquivos *JavaScript Object Notation* (JSON) que são mapeados em objetos. Esse banco possui uma arquitetura distribuída e suas consultas, indexação e agregação facilita o trabalho com os dados (MONGODB, 2019).

O MongoDB foi desenvolvido para aplicações modernas, oferecendo o modelo de dados de documento, otimizando e facilitando o desenvolvimento das aplicações. Possui um design distribuído que permite a alocação de dados a critério do usuário, além da flexibilidade de execução em diversas plataformas (MONGODB, 2019). Os dados armazenados nesse banco são organizados em coleções e documentos, que equivalem a tabelas e tuplas de um BD Relacional, respectivamente.

2.4 SISTEMAS EM TEMPO REAL

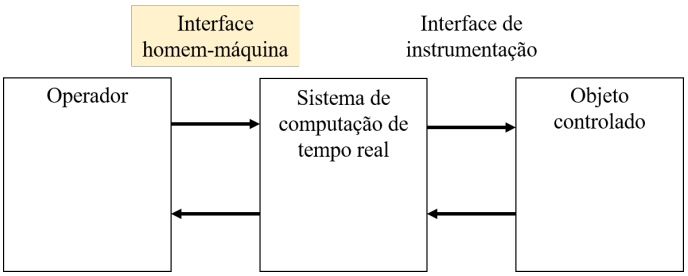
Um Sistema em Tempo Real (STR) é caracterizado quando a acurácia de suas informações não está associada somente aos resultados lógicos de seu processamento, mas também quanto ao tempo de processamento em que os resultados são alcançados (KOPETZ, 2011).

É importante que um STR possa reagir aos estímulos do meio exterior e retorne uma resposta em um determinado intervalo de tempo. Caso o STR não cumpra com o prazo correto, ocorre-se uma falha temporal. Um dado computado além das especificações do sistema, pode não ter utilidade nenhuma e até mesmo representar algum risco (FARINES; FRAGA; OLIVEIRA, 2000). A Figura 1 mostra um diagrama simplificado de um STR.

Na Figura 1, observando a partir da direita para esquerda, indica em seu primeiro módulo a representação do sistema que computa as informações em tempo real (Sistema de computação de tempo real). Existe um intermediador entre esses dois módulos (Interface de instrumentação). O último módulo representa a operação humana em um STR, ou seja, uma pessoa que interage com

[1] Um nó nesse contexto, é caracterizado como um componente de um sistema distribuído onde é armazenado um versionamento dos dados

Figura 1 – Modularização de um sistema em tempo real



Fonte: Adaptado de Kopetz (2011)

o sistema diretamente. O elemento que faz intermédio entre o operador e o sistema é “Interface humano-máquina”, o qual converte os comandos realizados pelo operador compreendidos pela aplicação STR.

2.5 CONCLUSÃO

Neste capítulo, foi explanado sobre os conceitos básicos relacionados a Banco de Dados, que se trata de um sistema capaz de armazenar dados de forma consistente. A explicação sobre Banco de Dados pecorre a definição de bancos relacionais até os Bancos *NoSQl*, sendo este último utilizado para criação da proposta deste trabalho. Além disso, foi explicado sobre a conceituação de *Big Data*, que está relacionado com novas tecnologias de processamento e armazenamento de grande volumes de dados. Ademais, foi definido o conceito a respeito de Sistemas de Tempo Real, que consiste em uma aplicação construída para reponder a estímulos do ambiente onde se encontra, fazendo com que essa informação se propague ao mesmo passo em que é descoberta. Todos os conceitos, apresentados neste Capítulo, fundamentam as bases teóricas deste estudo.

3 ESTADO DA ARTE

Neste Capítulo, serão expostos trabalhos que apresentam ideias similares a ideia proposta por esta pesquisa. Aborda-se outros *dashboards*, onde serão descritos os procedimentos de desenvolvimento como também seus resultados.

3.1 DESENVOLVIMENTO DE *DASHBOARD* DINÂMICO UTILIZANDO *BUSINESS INTELLIGENCE* SOBRE SOLICITAÇÃO DE SERVIÇO EM UM AMBIENTE GOVERNAMENTAL

O trabalho de Joazeiro (2015) visa o desenvolvimento de um processo de *Business Intelligence* (BI) utilizando-se dos dados de um sistema de solicitação de serviços de projetos e sistemas dentro de um contexto governamental. Tendo como produto final do trabalho, um *dashboard* sobre os dados levantados, que colabora com a tomada de decisão (JOAZEIRO, 2015).

A autora realizou um estudo de caso sobre a Secretaria de Estado do Mato Grosso, onde aplicou ténicas de BI para a sistematização de estratégias que compreendem os valores, impacto social e processos internos da secretaria. A metodogia de construção da solução seguiu três módulos: criação de uma *Data Warehouse* baseado em banco de dados relacional; construção de um módulo de extração, transformação e carregamento de dados; e implatação de módulo de visualização em *dashboard*.

No estudo realizado, o módulo de visualização (*dashboard*) é desenvolvido a partir da avaliação das necessidades de projeto da Secretaria, identificadas a partir das técnicas de BI. Tendo áreas específicas determinadas por parâmetros de busca relacionados ao projetos da Secretaria. O trabalho de Joazeiro (2015) é restrito ao escopo de um secretaria e não faz mensão quanto a ferramenta ter a possibilidade de ser escalável, dado o aumento do volume de dados.

3.2 CRIAÇÃO DE UM *DASHBOARD* PARA MONITORAMENTO DE PERFIS DE QUALIDADE DE SOFTWARE

O estudo de Santos (2017) visa o desenvolvimento de um painel de controle para o acompanhamento do controle de qualidade de *software* para órgãos públicos brasileiros. A solução proposta foca na simplicidade e objetividade das informações, tendo em vista que,

grande parte dos órgãos públicos terceirizam o serviço de *software* (SANTOS, 2017).

Para a construção do *dashboard*, o autor dividiu o processo em etapas, sendo a primeira a coleta de informações a respeito do tema, a segunda trata da simulação de uma abiente real sobre controle de qualidade de *software* para um órgão público, e por fim, a fase de implementação da proposta, que baseia-se na entrega incremental e funcional do sistema.

3.3 DESENVOLVIMENTO DE *DASHBOARD* PARA ANÁLISE DE DADOS DE AGRONEGÓCIO

O estudo de Silva (2018) visa a criação de um *dashboard* que emprega técnicas de BI buscando o levantamento de aspectos importantes que sirvam de indicadores para o segmento do agronegócio. A aplicação trata de um plataforma Web, que além das tecnologias base, suporta estratégias de manipulação e agrupamento de dados (SILVA, 2018).

Como resultado, o autor obteve uma ferramenta que trata de quatro requisitos para a criação de um indicador, todos os requisitos baseados em dados de produção e escoamento de uma safra. O *dashboard* alcançado é alimentado por um *Data Warehouse*, onde acumula dados extraídos para utilização direta da ferramenta.

3.4 RELAÇÃO ENTRE OS TRABALHOS

Os trabalhos desenvolvidos por Joazeiro (2015), Santos (2017) e Silva (2018) estão relacionados com este, pois todos visam a construção de uma ferramenta (*dashboard*), que oferece suporte a tomadas de decisões, as quais são baseadas em dados.

Entre os estudos apresentados, o promovido por Joazeiro (2015) ganha destaque, visto que esse aplica técnincas de BI em um contexto real de caso de uso, obtendo resultados concretos a partir de uma abordagem sistemática via módulos bem definidos. Já o trabalho de Santos (2017) aplica o estudo sobre uma simulação de caso de uso real. Porém, o objeto de pesquisa trata de algo subjetivo como a qualidade de *software*, abrindo margem para disparidade entre interpretações. Finalmente, o trabalho de Silva (2018), apresenta a construção de uma ferramenta com técnicas similares com o trabalho de Joazeiro (2015).

Os estudos mostrados até aqui não focam na criação de ferramentas para trabalho com grandes volumes de dados ou DAs. Dito isto, a proposta oferecida por este estudo é relevante por analisar dados sem restrições e de grandes volumes. Portanto, sendo uma aplicação escalável,

em comparação as descritas nesta Seção. Assim, visa-se a concretização de uma solução que explora uma abordagem para funcionamento com sistemas de infraestrutura de tempo real, a qual utiliza o paradigma *MapReduce*, que garante uma melhor estratégia de processamento para manipular grandes volumes de dados.

4 TECNOLOGIAS UTILIZADAS

Neste Capítulo, são apresentadas as tecnologias preliminares utilizadas neste trabalho de pesquisa. A Seção 4.1 explana sobre as linguagens de programação utilizadas e a Seção 4.2 discorre sobre as bibliotecas empregadas, que extendem as capacidades das linguagens da Seção 4.1.

4.1 LINGUAGENS

Nesta Seção, são apresentadas linguagens importantes utilizadas para construção da ferramenta proposta, elas serão descritas nas próximas seções. Assim, são discorridos conceitos acerca da linguagem JavaScript, HTML e CSS na Subseção 4.1.1.

4.1.1 JavaScript, HTML e CSS

Como principal linguagem de programação utilizada para o desenvolvimento da ferramenta *DASHCARE*, foi escolhida a linguagem JavaScript (JS). Essa linguagem, é baseada em scripts, interpretada, e multiparadgma. Sendo a principal linguagem em execução em ambientes Web (MDN, 2019c). Em sites e páginas Web, o JS pode ser utilizado para implementar a lógica de negócio e fornecer dinamicidade às aplicações (ELOQUENT JAVASCRIPT, 2019). A plataforma comunitária *Stack Overflow*, em 2018, realizou uma pesquisa entre seus usuários, na qual mostrou que, o JS é listado como a linguagem de programação mais popular entre o desenvolvedores profissionais. A Figura 2 ilustra o *ranking* da plataforma.

Figura 2 – Liguagens mais populares do Stack Overlflow em 2018.

Fonte: Adaptado de (STACK OVERFLOW, 2018)

Plataformas como a *Node.js* (Subseção 4.3.1) e *MongoDB* (Subseção 2.3.2), tornam possível a utilização e excução da linguagem JS em ambientes além dos navergadores de internet, tais como servidores e banco de dados (ELOQUENT JAVASCRIPT, 2019).

Para a estruturação semântica, divisionária e interoperabilidade do sistema, foi aplicado o HTML5 (*Hypertext Markup Language*), que se trata de uma linguagem para criação de páginas WEB. Com ela é permitido descrição precisa do conteúdo, é extensível para lidar com outras tecnologias, além de outras atribuições e funcionalidades (MDN, 2019b). A fim de personalização, foi escolhido o CSS3 (*Cascading Style Sheets*) que permite estilização direta de elementos HTML, além da aplicação de animações, transições e organização (MDN, 2019a).

A utilização dessa tecnologia, entretanto, ocorreu de forma indireta, por meio da aplicação da biblioteca React (Subseção 4.2.1), que utiliza JS para a montagem dos elementos HTML com seus respectivos estilos CSS.

4.2 LADO CLIENTE

Esta Seção explica sobre as bibliotecas empregadas neste trabalho. Elas foram importantes para a construção da ferramenta, pois ofereceram extensibilidade e alta abstração de funcionalidades. A Subseção 4.2.1 explica sobre *React*, principal biblioteca e base para a ferramenta; a Subseção 4.2.2 sobre *styled-components*, responsável pela estilização; a Subseção *Reacharts*, bilioteca para plotagem de gráficos; e, por último, a Subseção 4.2.4, que explica a *axios*, biblioteca para requisições ao lado servidor.

4.2.1 React

O React[1], é uma biblioteca JS para a criação de *User Interfaces* (UIs). Com ela é possivel a criação de UIs simples para cada estado de aplicação que a utiliza. O *React* é reativo, ou seja, irá atualizar e renderizar[2] de forma eficiente apenas os componentes necessários na medida em que os dados mudam. (REACT, 2019). No *React*, as UIs são construídas com a composição de unidades pouco acopladas, chamadas de componentes. Cada componente, tem seu próprio estado e propriedades (*props*) internas, podendo ser integrado a outros componentes a fim da construção de uma interface complexa (REACT, 2019).

[1] Disponível em: <https://pt-br.reactjs.org/>.

[2] O processo de renderização, trata de configuração do estado de um componente e sua montagem em tela

A ferramenta desenvolvida neste trabalho adota componentes na forma de funções. A Figura 3 mostra a implementação de um componente do *DASHCARE*, responsavél pela plotagem dos gráficos por estado, empregando *hooks*[3].

Figura 3 – Componente baseado em *hooks*.

```
1  import React, { useState, useEffect } from 'react' ···
13
14 export default function Estate (props) {
15
16     const [month, setMonth] = useState()
17
18     useEffect(() => {
19         loadInfo()
20     }, [month])
21
22     return (
23         <EstateStyled>
24             {/* COLUNA 1 */}
25             <Column d={4}>
26                 <Card style={{height: '600px', padding: 0}}>
27                     {o && <DashMap
28                         center={UF_COORDS[estate].coord}
29                         zoom={UF_COORDS[estate].zoom}
30                         coords={o}
31                     />}
```

Fonte: Próprio Autor.

Verificando a Figura 3, na linha 1, a instrução “import” importa a classe “React” e os *hooks*, “useState” (instancia um estado) e “useEffect” (executa uma função quando o estado muda) necessários; na linha 14, um componente é criado e exportado, no exemplo é chamado de “Estate”; na linha 16 é declarado o estado, “month”, e a função que altera-o, “setMonth”, por meio do retorno da chamada de “useState”; nas linhas 18 a 20, a chamada de “useEffect” executa a função passada como argumento a qualquer mudança no “month”, nesse exemplo, o que é feito é a execução da função “loadInfo”; por último, a partir da linha 22 é retornado os elementos JSX que será renderizado ao final, com uma interpolação dos valores armazenados nos estados ou variáveis globais.

O JSX, que não é uma liguagem de marcação como HTML, mas uma extensão do próprio JS similar ao HTML, torna possível a separação de conceitos. Nele, são escritos elementos e seus estilos específicos. O JSX também permite a interpolação de valores de estados e aplicação de lógica em JS puro.

[3] *Hooks* são métodos nativos do *React* que manipulam estados de um componente

4.2.2 Styled-components

Styled-components[4] é uma biblioteca para *React*, que possibilita a escrita de componentes *React* com seu estilo diretamente integrado ao elemento HTML final, sem necesidade de arquivos CSS externos. A estilização é feita com CSS, porém esse é escrito em JS (*CSS-in-JS*) por meio de *template literals*, característica do JS presente desde a versão ES6. A Figura 4 ilustra o código do "CardStyled" do *DASHCARE* gerado pela biblioteca *styled-components*.

Figura 4 – Componente baseado styled-components.

```
1   import styled from 'styled-components'
2
3   export const CardStyled = styled.div`
4
5       position: relative;
6       display: flex;
7       flex-direction: column;
8       width: 100%;
9       margin-bottom: .4rem;
10      padding: .4rem;
11      background-color: #fff;
12      box-shadow: 0 2px 1px 0 rgba(0, 0, 0, .07),
13                  0 3px 2px 1px rgba(0, 0, 0, .05);
14      &:hover {
15          background-color: #f8f8f8;
16      }
17
18  `
```

Fonte: Próprio Autor.

Verificando a Figura 4, na linha 1 do código, é importado a instância da biblioteca *styled-components*. Na linha 3, é declarado o componente a ser criado recebendo o retorno da execução de "styled" com a abertura da *template literals*. O método "styled" é chamado, correspondendo ao elemento HTML que aquele componente criado irá renderizar ao final. Por fim, as linhas de 5 a 16, são inseridos códigos CSS, que definirão o estilo do componente criado.

Como em qualquer componente *React*, um componente *Styled-components* aceita o recebimento de valores específicos e computáveis via *props*. Isso permite o cálculo dinâmico para algumas propriedades CSS por meio de interpolação dentro do código, ou seja, permite a inserção de código dinâmico dentro de um código estático.

[4] Disponível em: <https://www.styled-components.com/>.

4.2.3 Google GeoCharts e Recharts

O *Google GeoCharts*[5] é uma API disponibilizada pelo Google. Disponibiliza formas de visualização e grande número de tipos de mapas prontos para uso em site. A forma mais comun de utilizar o *Google GeoCharts* é através do JS. Os gráficos são visualizados utilizando componentes *React*, e são renderizados através de HTML e *Scalable Vector Graphics* (SVG), garantindo o funcionamento em diversos navegadores.

O *Recharts*[6] é uma biblioteca *React* para a criação de componentes de gráficos. Com o *Recharts* é possível a escrita de componentes desacoplados e reutilizáveis. Os gráficos gerados por essa biblioteca são renderizado em SVG, o que tornar seu resultado final rápido e confiável. A Figura 5 mostra composição de um componente personalizado da aplicação deste trabalho com um gráfico de barras nativo do *Recharts*.

Figura 5 – Componente personalizado composto por gráfico Recharts.

```
64  export function DashBarChart (props) {
65
66      return (
67          <ResponsiveContainer width="100%" height="100%" >
68              <BarChart data={props.data}>
69                  <XAxis dataKey={props.xKey} height={props.xSize} tick={<Axis r={-60}/>} interval={0}/>
70                  <YAxis dataKey={props.yKey} height={props.ySize} tick={<Axis r={0}/>}/>
71                  <Tooltip />
72                  <Legend verticalAlign="top"/>
73                  <Bar dataKey={props.bKey} barSize={props.bSize} fill={props.color} />
74              </BarChart>
75          </ResponsiveContainer>
76      )
77
78  }
```

Fonte: Próprio Autor.

A Figura 5 ilustra como um componente reponsivo é criado (linha 67). Na linha 68, é criado um componente de gráfico de barras que recebe via *props* os dados que deverão ser mostrados. Da linha 69 a 78, são passados componentes que configuram os eixos do gráfico (como o efeito de *mouse* e aparências das barras). Como todos eles são componentes *React*, qualquer mudança nos dados passados é refletida automaticamente na renderização do gráfico.

[5] Disponível em: <https://developers.google.com/chart/interactive/docs/gallery/geochart>.
[6] Disponível em: <http://recharts.org/en-US>.

4.2.4 Axios

A *Axios*[7] é uma biblioteca cliente *Hypertext Transfer Protocol* (HTTP) que é executável em ambiente de navegador ou *Node.js* (Subseção 4.3.1). Dessa forma, o mesmo código pode ser usado para ambos os ambientes. Em linha gerais, *Axios* se trata de uma API que fornce um meio de comunicação para o envio de requisições e aquisição de respostas pelo protocolo HTTP.

As requisições da *Axios* são baseadas em *Asynchronous JavaScript and XML* (AJAX). Essa característica permite o tráfego da informação em formatos como JSON, HTML e XML, por exemplo. Também, com a *Axios*, as requisições podem ser realizadas de forma assíncrona, o que significa que cada requisição não necessita da atualização da página por inteiro. Em contextos de reatividade, como o oferecido pelo *React*, cada requisição pode ser atualizada para modificar parte do estado dos componentes da aplicação, permitindo renderizar apenas o necessário de acordo com as ações do usuário.

4.3 LADO SERVIDOR

Nesta Seção, é descrito quais as bibliotecas e *frameworks* foram utilizados para a montagem de ambientes em lado servidor do *DASHCARE*. A Subseção 4.3.1 discorre sobre o interpretador que torna possível a execução de JS fora de um navegador. A Subseção 4.3.2 descreve sobre o *framework* que trata das requisições HTTP ao servidor. A Subseção 4.3.3 explica a biblioteca utilizada para o mapeamento dos DAs ao banco de dados utilizado.

4.3.1 Node.js

O *Node.js* é um interpretador JS assíncrono orientado a eventos, de código aberto, projetado para aplicações escaláveis. A principal característica do *Node.js* é a execução do JS fora do lado cliente (*browser*) e no lado do servidor. A implementação do *Node.js* é baseada sobre o motor V8, uma *engine* de interpretação JS que compõe o navegador *Google Chrome*.

Uma das principais vantagens desse interpretador, é o seu tratamento especial de requisições HTTP. O protocolo HTTP no *Node.js* é tratato com prioridade, sendo projeto para que as requisições possam ser feitas em um grande fluxo e apresentem baixa latência. Além disso, com o *Node.js*, não há bloqueio de recursos devido a qualquer "competição" de requisições

[7] Disponível em: <https://github.com/axios/axios>

de entrada/saída. Isso torna esse interpretador uma boa escolha para utilização aliada a uma biblioteca ou *framework* (NODE.JS, 2019).

O *Node.js* é a base do lado servidor do *DASHCARE*. Apesar de oferecer uma API própria, permite a execução de APIs terceiras. Dessa forma, elimina-se a necessidade de criação de código para toda funcionalidade desejada, deixando o foco do desenvolvimento apenas para tratamentos específicos do *dashboard*.

4.3.2 Express.js

O *Express.js* é um "micro" *framework* para *Node.js*, suas principais características é ser minimalista, rápido e flexível. Sua orientação é para aplicações Web e móvel, oferecendo um conjunto robusto de recursos. O *Express.js* também fornece uma série de métodos utilitátios e *middleware* para a manipulação de requisições HTTP, facilitando a criação de uma API para consumo de dados. Apesar disso, por ser minimalista, o *Express.js* oferece uma síntese de recursos, sem mascarar os recursos nativos do *Node.js*.

Esse micro *framework* foi utilizado para a instanciação do servidor em si e o tratamento das requisições a API do *DASHCARE*, abstraindo diversas configurações do *Node.js* para esses propósitos. Ainda, o *Express.js* permitiu o tratamento de dados via *Mongoose* (ver Subseção 4.3.3) da forma desejada, antes que cada resposta a uma requisição seja enviada.

4.3.3 Mongoose

O *Mongoose* é uma biblioteca para *Node.js* que proporciona uma solução baseada em esquemas para modelar os dados da aplicação. Essa biblioteca é orientada ao banco de dados MongoDB, e possui um sistema de conversão de tipos, validação, criação de consultas e lógica de negócios. O *Mongoose* foi projetado para trabalho em ambiente assíncrono. De forma básica, trata-se de *Object Relational Mapping* (ORM), o que significa que o *Mongoose* transforma e traduz os documentos do MongoDB em objetos JS e vice-versa. Dessa forma, é criado um fluxo de trabalho suaviazado e otimizado com o JS.

Essa biblioteca foi utilizada para a conexão do lado servidor com o banco de dados que persiste os DAs do *dashboard*, além de fazer uma tradução dos dados entre servidor e banco de dados. Também foi utilizado sua API para manipulação dos dados persistidos, tais como operações de agrupamento do MongoDB.

4.4 PROCESSOS

O processamento da ferramenta se divide em duas etapas, mostradas nesta Seção. A primeira, discorrida na Subseção 4.4.1 explica como os dados são obtidos e computados; Já a segunda, mostrada na Subseção 4.4.2 descreve como os dados são consumidos e mostrados na ferramenta.

4.4.1 Processamento dos dados

O Algoritmo 1, mostra a configuração básica para lado do servidor do *DASHCARE*, como a criação de uma instância de servidor e conexão com o banco de dados. É válido ressaltar que o algoritmo se adequa à ferramenta proposta e possui considerável grau de abstração.

Algoritmo 1: Conexão e computação dos dados

```
   Data: servidor, dependencias, objeto_banco, controlador, rotas
 1 objeto_banco ← criarObjetoBanco();
 2 rotas ← criarRotas();
 3 dependencias ← requererDependencias();
 4 if dependencias then
 5 |   servidor ← instanciarServidor();
 6 |   servidor.liberarAcesso(dependencias);
 7 |   dependencias.conectarBanco();
 8 |   if requer(objeto_banco) ∧ requer(rotas) then
 9 |   |   servidor.definir(objeto_banco, rotas);
10 |   |   servidor.executar();
11 |   end
12 end
```

O Algoritmo 1 mostra o fluxo de construção do servidor, indicando como ocorre o fornecimento e computação dos dados. Na linha 1, é criado o objeto que controla o banco de dados com a chamada de "criarObjetoBanco" e o retorno (objeto que controla o banco de dados) é posto em "objeto_banco"; na linha 2, a chamada de "criarRotas" retorna um objeto contendo uma função de manipulação de dado ou requisição para cada rota[8] da aplicação, que é armazenado em "rotas". Uma função desse objeto é executada quando sua respectiva rota for acessada; na linha 3, são requeridas dependências[9] da aplicação, isso é feito com "requererDependencias", o resultado

[8] Uma rota é um caminho especifico que leva a uma ação especifico ou local da aplicação.

[9] As depedências requeridas são: "cors" (permisões de acesso ao servidor), "express" (instância da API do *framework Express.js*) e "mongoose" (instância da API da biblioteca *Mongoose*).

dessa operação é atribuído a “dependencias”; a linha 4 indica a verificação da veracidade das dependencias obtidas. Se for valor for *verdade*, ou seja, tiver sido instanciada corretamente, a condição é atendida e segue o procedimento esperado. Caso contrário, o algoritmo encerra e o servidor não é criado. Na linha 5, é feita a instanciação do servidor. É com essa instância que é possível ter acesso aos métodos que aplicam o uso das depêndencias ao servidor; na linha 6, é liberado o acesso as requisições a API do servidor, pois quando executado de forma local, não há impedimentos ao acesso da API.Porém, em situações em que as requisições são feitas por algum sistema a parte, existem restrições quanto à permissão. A linha 7 indica a tentativa de conexão com banco de dados pela instância das depedências. Em caso de falha na conexão, o servidor ainda é posto em operação, contudo qualquer requisição ao banco de dados retorna um *log* de erro e não os dados esperados. Na linha 8, é realizada uma avaliação sobre a veracidade dos requerimentos dos objetos (“objeto_banco” *e* “rotas”), por meio das chamadas de “requer”. Dessa forma, se ambos os objetos existirem e forem valores de *verdade*, a condição é avaliada como verdadeira e é dado prosseguimento à consolidação do servidor, do contrário, a execução do algortimo encerra-se e o servidor não é posto em execução. Na linha 9, é configurado o controlador para o banco de dados (“objeto_banco”) e também as funções para cada rota (“rotas”) pelo método “definir” do objeto “servidor”. Dessa forma, nesse ponto, as requisições já são tratadas de forma correta; na linha 10, o servidor é finalmente posto em execução pelo método “executar” e, por fim, na linha 12, termina o fluxo de execução da condição iniciada na linha 4.

4.4.2 Visualização dos dados

O Algoritmo 2 descreve como ocorre a visualização dos dados pela ferramenta *DASH-CARE*. Este Algoritmo é relativamente mais simples se comparado ao Algoritmo 1, mas como esse, ele também se adequa apenas a ferramenta proposta e está mostrado de forma abstrada. É importante ressaltar, que o Algoritmo aqui descrito, explica como ocorre a visualização sobre um componente apenas, haja vista que cada um matém seu estado distinto e realiza suas próprias requisições quando válido. Além disso, todo o módulo de visualização não passa de uma agregação entre vários componentes.

O fluxo de execução de um compoenente (parte do módulo de visualizaão) é mostrado pelo Algoritmo 2, que descreve como ocorre os processos dentro de um componente, que posteriormente se torna algo interativo ao usuário. Na linha 1, são requeridas dependências via “requerDependencias()”, que são necessárias para a montagem do componente na tela do cliente.

Algoritmo 2: Obtenção e visualização do dados

```
   Data: dependencias, componentes, estados, funcoes
1  dependencias ← requererDependencias();
2  componentes ← importarComponentes();
3  funcoes ← importarFuncoes();
4  if dependencias ∧ componentes then
5  |  estados ← dependencias.usarEstados();
6  |  dependencias.usarEfeito(dependencias.api);
7  |  dependencias.userEfeito(estados, funcoes);
8  |  return dependencias.renderizar(componentes, estados);
9  end
```

O retorno é armazenado em “dependencias”. Na linha 2, ocorre um processo análogo, mas desta vez, tratando-se dos outros componentes que formam o componente em questão, esses componentes são importados via “importarComponentes()” e seu retorno é colocado em “componentes”. A linha 3 mostra a importação das ações que são executadas dentro do componente, elas podem ser específicas ou genéricas, e são armazenadas em “funcoes”; na linha 4, é avaliada a veracidade dos objetos “dependencias” e “componentes”, caso eles existam. O fluxo da execução segue como o esperado, caso contrário, o Algoritmo computa a linha 9, onde sua execução encerra sem mostrar os gráficos na tela do navegador. Na linha 5, são definidos os estados internos de um componente em “estados”, dado o retorno de “dependencias.usarEstados()”. Os estados são, de fato, o que guarda as informações mostradas aos usuários em tela. Mas, além disso, guarda valores que são de interesse apenas dos componentes internos da aplicação. A linha 6 indica a execução de “dependencias.usarEfeito(dependencias.api)”, que computa uma requisição da API descrita no Algoritmo 1. Essa requisição é feita internamente pelo objeto “api”, que é passado como argumento e seu propósito é trazer os dados do servidor; na linha 7, é feita uma nova chamada de “dependencias.usarEfeito”, mas desta vez passando os estados internos, que são manipulados pelas “funcoes” especificas do componentes. Ambos são passados como argumento ao método. Por fim, a linha 8 discorre sobre o retorno da chamada do método “dependencias.renderizar (componentes, estados)”, que é responsável por renderizar na tela do navagador os elementos gráficos (componentes) com seus repectivos dados (estados), que são passados como argumentos ao método invocado.

4.5 CONCLUSÃO

Este capítulo discorreu sobre as tecnologias básicas que foram utilizadas para a construção do *DASHCARE*. O destaque do mesmo está relacionado com as linguagens de programação, as quais tem relacionamento com domínios Web, sendo elas JS, HTML e CSS. Como uma extensibilidade da linguagem JS, foi apresentado a biblioteca *React*, que possui uma arquitetura baseada em componentes e opera do lado cliente da aplicação e serve como suporte para outras bibliotecas empregadas. Neste Capítulo, também foram apresentados os processos envolvidos para o alcance dos objetivos deste trabalho, definido a partir de dois Algoritmos. O primeiro descrevendo a elaboração do lado de manipulação e aquisição dos dados, e o segundo descrevendo como as informações são mostradas para o usuário. É importante ressaltar que todas as tecnologias utilizadas são de código aberto e livres para uso.

5 DASHCARE: UM DASHBOARD PARA ANÁLISE DOS ACIDENTES NAS RODOVIAS FEDERAIS

O propósito deste Capítulo é discorrer sobre a construção do *DASHCARE*, proposto neste trabalho de pesquisa. A solução construída pode ser usada para analisar os acidentes que ocorrem nas rodovias federais do Brasil, usando os dados abertos do Governo Federal. A Seção 5.1 discorre sobre a arquitetura geral da aplicação e a Seção 5.2 realiza uma explicação mais detalhada sobre o desenvolvimento dos módulo do *DASHCARE*.

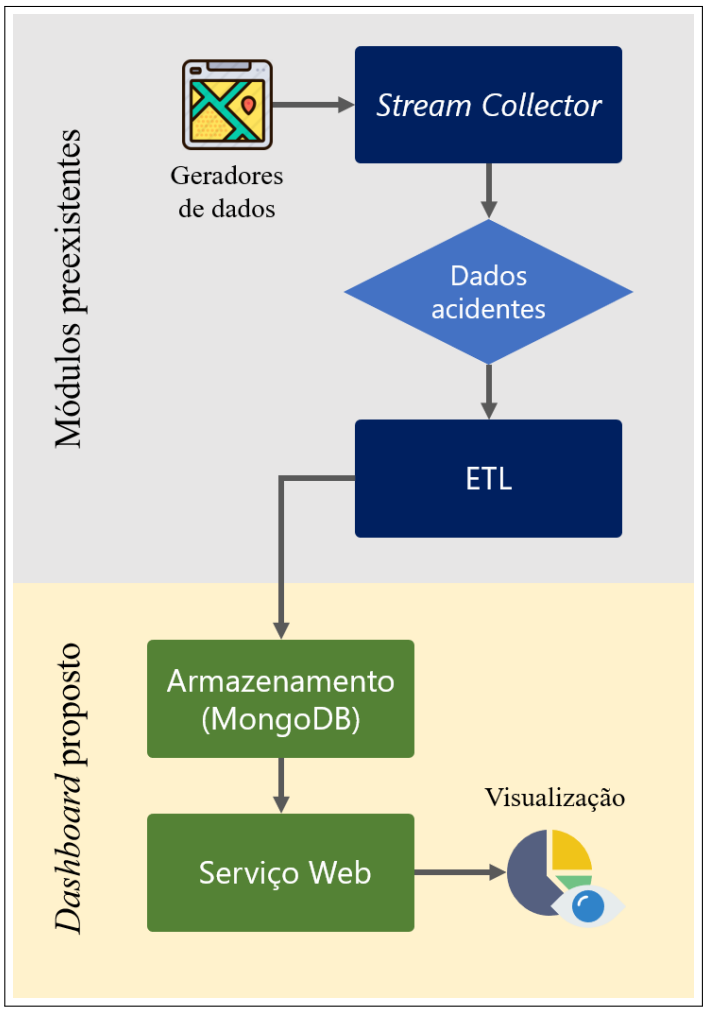
5.1 ARQUITETURA DO SISTEMA

Em uma visão macro, a aplicação proposta é uma interface de visualização e relacionamento da informação extraída a partir dos DAs dos acidentes das RFs. Em uma visão ainda mais macro, na Figura 6 é possível identificar a existência de duas áreas distintas: (i) Módulos preexistentes; e (ii) *Dashboard* proposto. Na qual, foi concentrado o esforço deste trabalho no desenvolvimento da área "*Dashboard* proposto".

A área de "Módulos preexistentes" mostra como seria o possível plano de fundo para qualquer serviço que queira consumir informações a partir da proposta deste trabalho. Nessa área, é possível encontrar quatro módulos: (i) geradores de dados, que são as fontes responsáveis por produzirem os dados a serem analisados; (ii) *stream collector*, que é responsável pela coleta de dados gerados por fontes específicas em tempo real; (iii) dados acidentes, que corresponde ao conjunto de dados relacionados, obtidos a partir do módulo anterior; e (iv) o módulo ETL (*Extract, Transform, Load*), que integra os dados originários de diversas fontes de acordo com as regras de negócio da entidade utilitária. Vale ressaltar que os *Módulos preexistentes* não foram abordados no escopo deste trabalho, ficando a cargo dos utilizadores do possível sistema, o fornecimento dos módulos explicados ou similares.

A área de "*Dashboard* proposto", é de fato, o escopo de pesquisa e construção deste trabalho. Nela é projetada a existência de três módulos: (i) Armazenamento, que disponibiliza uma API que se conecta com o módulo ETL e persiste os dados no Banco NoSQL MongoDB; (ii) Serviço Web, que culmina no *Dashboard* propriamente dito, onde ocorre o processamento, análise e seleção de relações entre os dados; e (iii) Visualização, onde o usuário pode interagir com a ferramenta, requisitando consultas pelos parâmetros desejados e recebendo as informações em elementos gráficos intuitivos (*e. g.* tabelas e gráficos).

Figura 6 – Arquitetura simplificada do sistema



Fonte: Próprio Autor.

5.2 DASHBOARD PROPOSTO

Estão Seção apresenta mais detalhamente a construção do três módulos referentes a área "*Dashboard* proposto" do modelo de arquitetura explicada na Seção 5.1. A Subseção 5.2.1 descreve o desenvolvimento da persitência da ferramenta e a Subseção 5.2.3 explica sobre o modelo visual do *Dashboard* desenvolvido.

5.2.1 Armazenamento e seleção do conjunto de dados

Na arquitetura visualizada na Figura 6, os dados armazenados são obtidos após a passagem por um anterior módulo ETL, porém, este trabalho não abrange nenhum módulo ETL disponível para o propósito da solução. Portanto, foi necessário popular o banco separadamente, com isso foi ganha uma simulação de existência dos "Módulos preexistentes" a partir da última

etapa.

Para tanto, os DAs fornecidos pela PRF, originalmente são contidos em formato CSV (*Comma-Separated Values*) e não em uma base de dados única, isso impede o uso de uma série de funções que um banco de dados possui. Dessa forma, os dados dos arquivos tiveram que ser importados para o Banco NoSQL MongoDB, esse banco foi escolhido, pois uma instância (ocorrência) de um acidente nos dados não apresenta relação direta com outras. O MongoDB oferece uma rica API, incluindo uma função de importação nativa, que recebe um arquivo em formato válido (CSV, por exemplo) e o converte em uma coleção de documentos. Uma vez transformado os DAs em uma coleção do MongoDB, foi possível ter acesso a API de manipulação de dados desse banco, trabalhando com diversas operações, como agregação e agrupamento.

O conjunto de dados usados para validar o *dashboard* refere-se ao modelo de DAs fornecidos pela PRF a partir do ano de 2017, cujo arquivo de análise contém ao todo 37 campos. Entretanto, nem todos os dados obtidos foram considerados necessários à análise ou visualização de informações. Sendo assim, foi realizada uma seleção a fim de identificar as variáveis que teriam propósito para o processamento dos dados, ou para a visualização da informação, ou seriam desconsiderados.

A análise e seleção dos campos para aplicação na ferramenta foi definida a partir de 3 critérios:

a) Campos de identificação - qualquer campo que tenha uma conotação de identificação (valor único, *e. g.* id), a priori, foi posto no escopo do processamento (*e. g.* operações de agrupamento), tanto na manipulação de servidor quanto lado cliente;
b) Campos de características - qualquer campo que mostre a característica relacionada a ocorrência de acidente ou pessoa(s) envolvida(s). A priori, foi posto no escopo de visualização e requisição (*e. g.* tipo de acidente);
c) Campos desconsiderados - qualquer campo que mostre valores não genéricos de um veículo (*e. g.* modelo do veículo), a priori, foram desconsiderados. Vale resaltar que esses campos são desconsiderados nos gráficos do *dashboard* até então, mas permanecem na base de dados para uma eventual aplicação de algoritmos mais poderosos (*e. g.* Mineração de Dados).

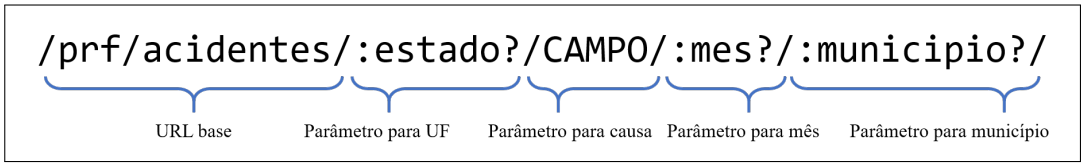
5.2.2 Estrutura e Montagem do Servidor

A estrutura do servidor atua como intermediário entre o módulo de Visualização e Armazenamento. Seu objetivo é receber as requições solicitadas, realizar uma consulta ao banco de dados ou algum tratamento e enviar uma resposta. A base do servidor foi construída com *Node.js*. Para inicialização do servidor foi utilizado o *Express.js* e para conexão com o MongoDB foi utilizado o ORM *Mongoose*.

Tendo em vista a natureza intrínseca aos dados de acidentes da PRF e ao universo de discurso que eles representam, o servidor ficou encarregado de realizar apenas operações "read" (leitura) de um sistema CRUD (*Create*, *Read*, *Update* e *Delete*) dentro da coleção do MongoDB. Isso é explicado devido ao fato das alterações no universo de discurso não serem sistêmicas. A realização de operação *create* estaria criando um falso acidente. A realização de operação *update* estaria quebrando a confiabilidade dos dados. A reliazação de operação *delete* sem o devido estudo de limpeza de dados, estaria reduzindo a acurácia da informação contida.

O servidor foi desenvolvido para o monitoramento de requisições para diferentes rotas que representam os diferentes campos dos dados, a Figura 7 mostra a estrutura básica de uma rota do *DASHCARE*, que é recebida via requisição pela instância do *Express.js*.

Figura 7 – Modelo base de uma rota do *DASHCARE*



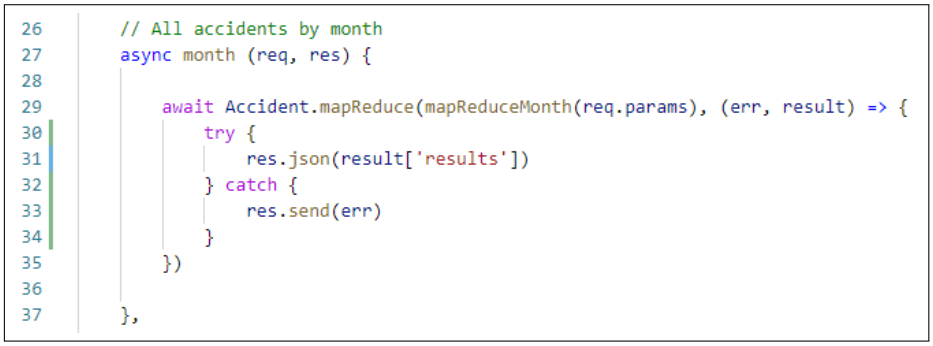
Fonte: Próprio Autor.

Na Figura 7, é mostrado o modelo de rota padrão do *DASHCARE*, a "URL base" é um prefixo obrigatório que identifica a API de destino; "Parâmetro para UF", é um atributo que recebe a sigla de um estado brasileiro e delimita os acidentes para aquele estado; "CAMPO", é um atributo obrigatório que recebe um título de um campo dos dados (*e. g.* causa, tipo); "mes", é um atributo que recebe as três iniciais de um mês e delimita os acidentes por mês e dias; e "municipio" recebe o nome de um município do respectivo estado passado e delimita a busca por região. O ponto de interrogação no fim de um parâmetro indica que o mesmo é opcional.

Com esse esquema de rota, quando uma requisição é feita, a instância do *Express.js* sabe qual é a função assíncrona específica dessa rota e a executa. A função assíncrona é encarregada de invocar algum método de acesso ao MongoDB da instância do *Mongoose* (Figura 8) e realiza

uma operação de busca e tratamento dos dados. O retorno dessa função é enviado como resposta a requisição. É importante frisar que as rotas são chamadas internamente aos componentes da aplicação, deixando-as “invisíveis” ao usuário final, onde esse não tem acesso direto a elas.

Figura 8 – Função da rota com parâmetro CAMPO igual a “mes”



Fonte: Próprio Autor.

Na Figura 8, a o nome da instância do *Mongoose* é “Accident” e o método de acesso que executa uma função é o “mapReduce”, ambas as declarações na linha 29.

5.2.3 Visualização das informações

A estrutura de Visualização é o último módulo da solução proposta, como pode ser visto na Figura 6. É esse módulo encarregado de mostrar os informações contidas nos dados ao usuário, ou seja, o *dashboard* propriamente dito. Apesar de ser executado totalmente no navegador e não estar integrado diretamente com os outros módulos, realizando apenas requisições, ainda faz parte do sistema, pois segundo Kopetz (2011), um dos elementos que compõe um STR é a “Interface homem-máquina” capaz de transformar comandosde maior abstração (menor granularidade dos dados) em comandos de mais baixo nível (maior granularidade dos dados, indicando uma maior riqueza de detalhe). O processo inverso também acontece, onde dados saem do sistema de computação e são convertidos em informações pela interface.

O *DASHCARE* foi construído como um projeto *React* em uma abordagem SPA (Single Page Aplication), onde só existe um aquivo HTML principal e todas as páginas são renderizadas de forma dinâmica de acordo com as ações do usuário e estados internos. As páginas são modeladas através da componentização de elementos de UI com reposabilidade única, os quais são estilizados com a biblioteca *styled-components*.

A maior parte da informação no *DASHCARE* é plotada em gráficos interativos, utilizando

as bibliotecas *Google GeoCharts* e *Recharts*. Cada página possui seus respectivos gráficos que são atualizados de acordo com o estado da página, a qual muda para mostrar a resposta de uma requisição via *Axios*.

A abordagem de SPA remove do lado servidor o ônus de renderizar os elementos (deixando totalmente a cargo do navegador) e enviar um nova página a cada requisição. Assim, o servidor ganha espaço de processamento, focando apenas na manipulação dos dados. Como o *React* e suas dependências são feitas em JS, o *DASHCARE* se torna portátil, podendo ser executado em todo navegador moderno ou *desktop*, se aplicado em uma abordagem baseada em *Node.js*.

5.3 CONCLUSÃO

Neste Caítulo, foi apresentado uma visão geral sobre os três módulos que formam a aplicação *DASHCARE* proposta neste trabalho. Os módulos são três e correspondem ao "Armazenamento", "Servidor" e "Visualização". Foi explicado como realizou-se a população do banco MongoDB com os DAs da PRF e quais os critério que decidiram qual o domínio de cada campo dos dados. Foi discorrido sobre como módulo "Servidor" executa uma ação via rotas que permitem o acesso aos dados e foi explanado sobre o módulo de interface e como ele age por meio de componentização.

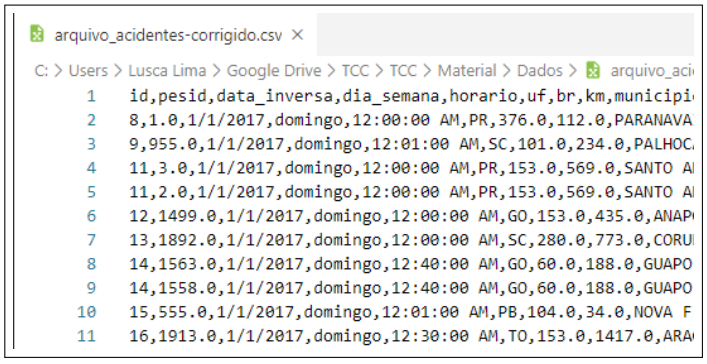
6 RESULTADOS E DISCUSSÕES

Neste Capítulo são mostrados os resultados obtidos a partir do desenvolvimento dos módulos de Armazenamento, Serviço Web e Visualização do *DASHCARE*, bem como uma breve discussão sobre esses. O Capítulo é composto por duas seções: a Seção 6.1 que descreve os resultados para a persitência dos dados e o servidor que os manipula; e a Seção 6.2 que mostra as informações exibidas pelo *dashboard* construído.

6.1 MÓDULOS DE ARMAZENAMENTO E SERVIDOR WEB

O armazenameto dos dados foram realizados em uma coleção do MongoDB, a partir de um arquivo CSV disponibilizado pela PRF, como Dado Aberto. Na Figura 9, é possivel visualizar um trecho dos dados utilizados para validar a ferramenta, nele cada elemento entre vírgulas é um valor e cada linha é uma instância de um acidente. Na Figura 10, é mostrado o resultado da operação de importação para o banco de dados. No MongoDB, cada linha do banco de dados (instância) transforma-se em um documento da coleção, aqui chamada de “dadosprf”.

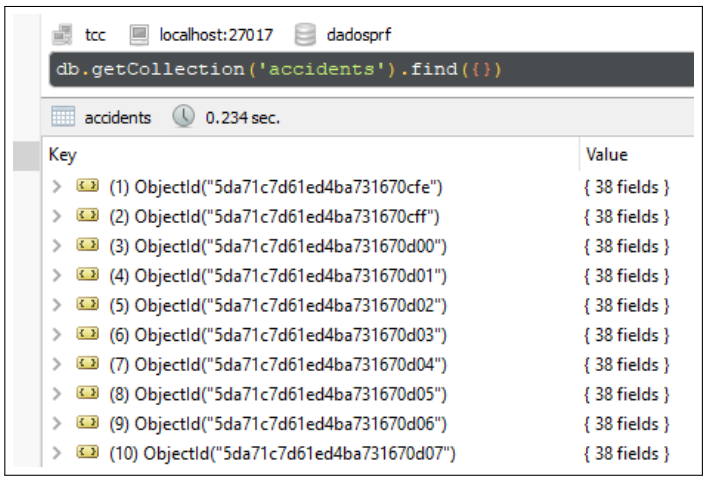
Figura 9 – Modelo original dos dados fornecidos pela PRF



Fonte: Próprio Autor.

As tarefas de construção do servidor, resultaram em uma API simplificada, onde só é possível realizar operações de leitura sobre os dados. As informações acerca dos acidentes podem ser acessadas seguindo a rota base, explicada na Subseção 5.2.2, por meio de um navegador ou API de requisções como a *Axios*, como ocorre no módulo de visualização dos dados. Para exemplificar, a Figura 11 mostra o acesso dos dados filtrados pelos seguintes atributos: o estado do RN, o tipo de acidente, a cidade de Mossoró e o mês, que foi definido o valor de Dezembro. Nesse caso, o servidor acessa o banco de dados e retorna um *array* de objetos com o campo de

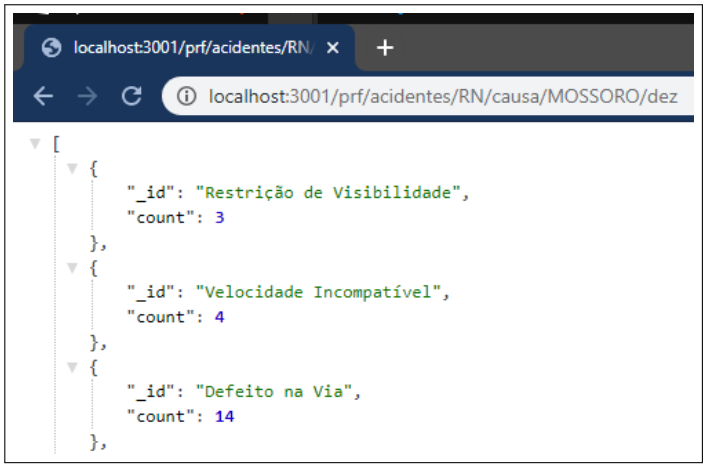
Figura 10 – Modelo dos dados armazenados no MongoDB



Fonte: Próprio Autor.

dados especificado, e se os demais parâmetros são passados, a busca é filtrada por esses.

Figura 11 – Acesso dos dados via rotas no servidor



Fonte: Próprio Autor.

Na API oferecida pelo servidor desenvolvido, também é disponibilizada uma rota especial que retorna todos os dados referentes a uma ocorrência de acidentes, sua assinatura é "/prf/acidentes/dados/:estado/:municipio", que segue os mesmos princípios da rota base.

Seguindo as definições explanadas do Capítulo 2, o servidor do *DASHCARE*, atende os princípios do universo de discurso abordado neste trabalho. Dessa forma, mesmo não assumindo a existência dos "Módulos preexistentes", os "3Vs" do *Big Data* são asegurados, pois o *volume* de dados fornecidos pela PRF é ligeiramente grande, a *variedade* é disposta entre 37 campos distintos, e a *velocidade* reside nos ambientes que processam esses dados, como *Node.js* e MongoDB, que realizam operações eficentes como *Map Reduce*, por exemplo e possuem

responsabilidade única.

O servidor resultante, está de acordo com o Teorema CAP (Subseção 2.3.1), pois não há garantia que na presença dos "Módulos preexistentes", o módulo de ETL espelhe a adição de novos dados pela PRF em todas as intâncias do banco de dados utilizado. Mas, caso exista mais de uma instância, a capacidade de não bloqueio de recursos do *Node.js* mediante muitas requisições garante a disponiblidade das informações.

6.2 MÓDULO DE VISUALIZAÇÃO

O *DASHCARE*, é um sistema intuitivo que foi desenvolvido para o monitorar e visualizar as informações contidas nos DAs da PRF acerca dos acidentes no Brasil. O fluxo de execução da ferramenta é iniciado via requisões HTTP ao módulo servidor. O *dashboard* realiza sua primeira requisição ao ser renderizada a primeira tela no navegador e mostrar as informações dos acidentes em um escopo nacional. Esse estágio incial do *DASHCARE* é ilustrado pela Figura 12.

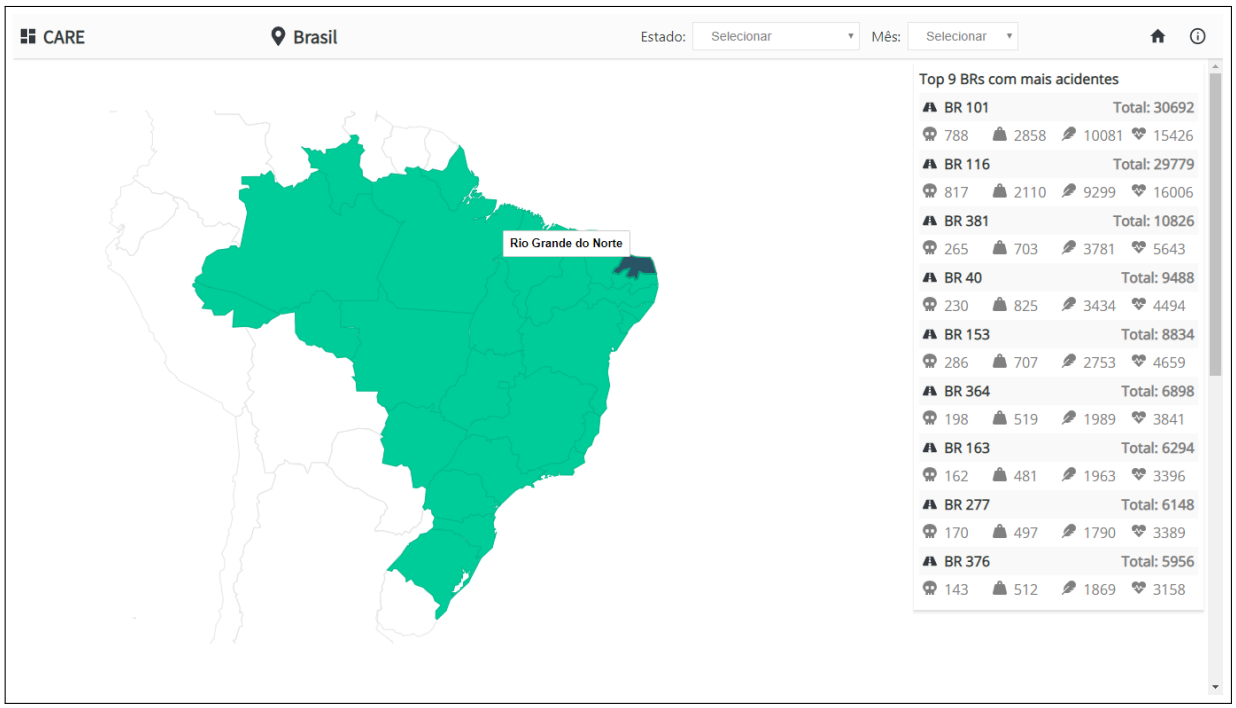
Na Figura 12, é mostrado a parte superior da tela incial do *DASHCARE*, que possui três componentes principais distintos:

1. Barra de navegação: nela é possível ter o nome da região de foco dos dados, no caso da tela incial, é "Brasil". Mais a direita, são encontradas duas caixas de seleção, a primeira definie o foco para um dos estados e a segunda seleciona o mês em que os dados devem ser buscados.
2. Mapa do Brasil: esse componente renderiza o mapa político definido pelos estados, onde cada um é uma área selecionavél que aumenta a granularidade da busca de acordo com o que foi selecionado;
3. *Ranking* BRs: esse componentes renderiza uma tabela que ordena de forma decrescente as BRs com maior número de vítimas. Os dados são mostrados em absoluto, gravidade dos ferimentos e óbitos.

A Figura 13 revela a parte inferior da tela incial no *DASHCARE*. Ela contém nove componentes que plotam gráficos de acordo com o campo especificado, todos eles são explicados a seguir:

1. Acidentes por causa: nesse gráfico é possível ter acesso aos acidentes de acordo com a causa dele (*e. g*. velocidade incompatível), ações relacionadas ao condutores dos veículos;
2. Acidentes por região: esse gráfico mostra o número de acidentes por região, sendo ele exclusivo dessa tela. Os dados são computados dentro do *dashboard* e não em lado

Figura 12 – Tela incial do *DASHCARE* (superior)



Fonte: Próprio Autor.

servidor;

3. Acidentes por veículo: esse gráfico mostra quais os veículos mais envolvidos com acidentes no Brasil;
4. Acidentes por sexo: esse gráfico mostra quais os sexos mais presentes nos acidentes;
5. Acidentes por mês: esse gráfico mostra o número de acidentes em ordem cronológica dos meses. Quando existe algum mês específico selecionado, o componentes passa a renderizar a ordem cronológica dos dias do mês escolhido;
6. Acidentes por tipo: esse gráfico mostra quais os tipo de acidentes mais comuns no Brasil. Essa informação está relacionada a forma do acidente conduzida pelo veículo;

Na tela que mostra as informações pelo estado, é possível encontrar quase todos os mesmos componentes que na tela incial, porém com alguns novos (Figura 14), como:

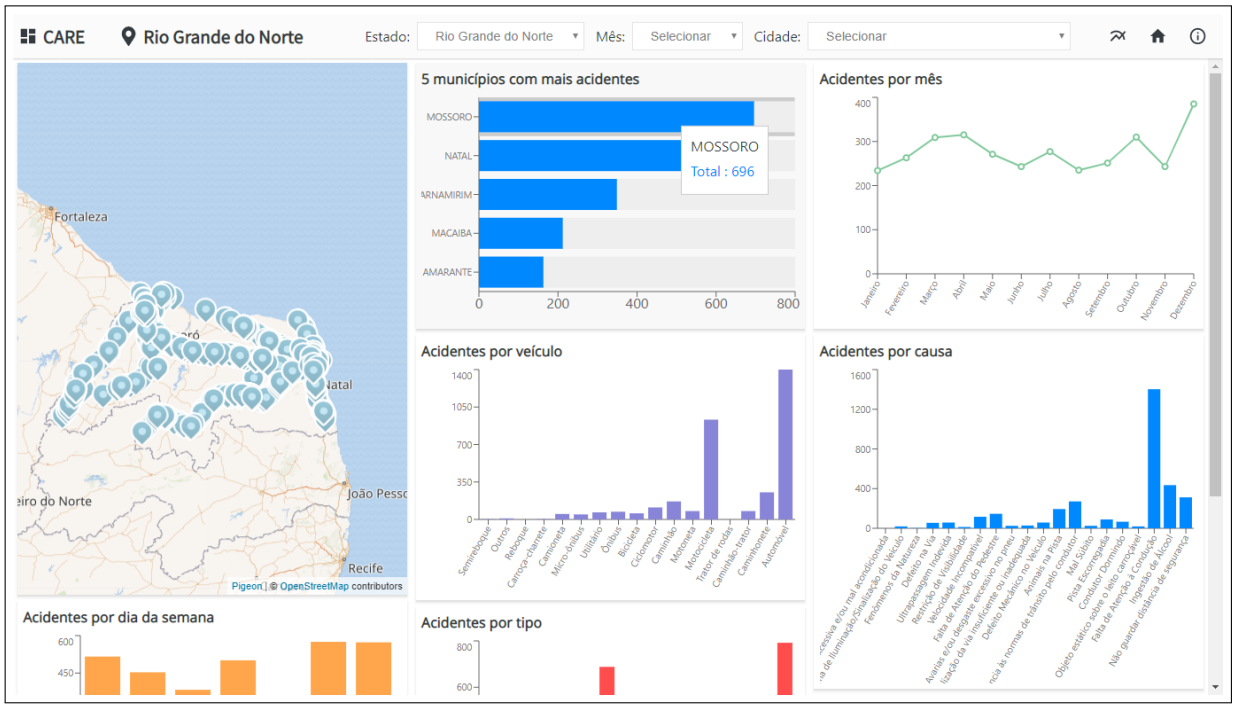
1. Mapa: esse componente renderiza um mapa que plota os pontos das coordenadas de cada acidente ocorrido. O número de pontos altera conforme a granularidade da informação;
2. Barra de navegação: esse componente, quando renderizado uma tela de estado, ganha uma terceira caixa de seleção para município, a fim de diminuir a granularidade do dado;
3. *Ranking* municípios: esse gráfico de barras mostra quais são os cinco municípios do estado selecionado com maior número de acidentes;
4. Acidentes por dia da semana: esse gráfico revela quais os dias da semana com maior

Figura 13 – Tela incial do *DASHCARE* (inferior)

Fonte: Próprio Autor.

número de acidentes ocorridos. Esse componente é exclusivo da tela de estado e município.

Figura 14 – Tela por estado no *DASHCARE*



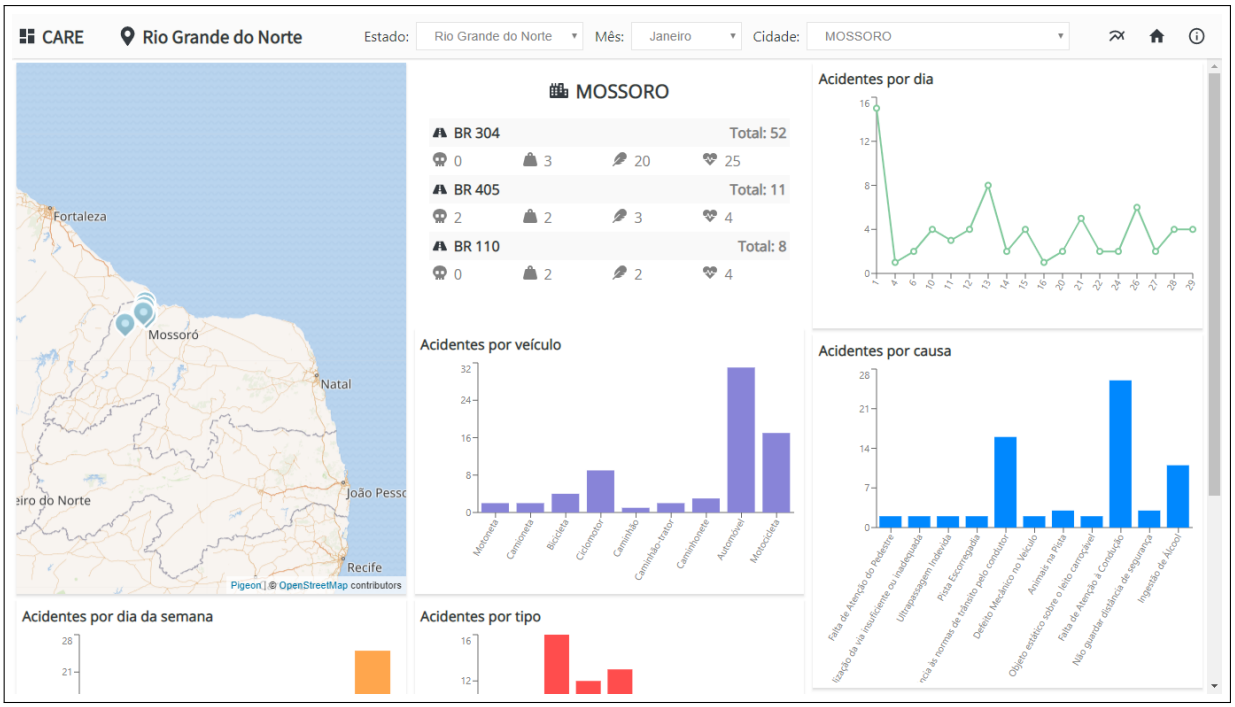
Fonte: Próprio Autor.

A partir da tela de um estado, é possível ter acesso aos seus municípios, via caixa de

seleção da barra de navegação. A Figura 15 mostra as alterações ocorridas nos componentes, quando é selecionado um município específico.

1. Tabela: esse componente que renderiza a tabela é o mesmo componente da tela incial, porém os dados mostrado são agrupados por município. Esse componente assume o lugar do componente "*Ranking* municípios";
2. Acidente por dia: esse componente é o mesmo que "Acidentes por mês", mudando apenas os dados passados ao gráfico e o título;
3. Mapa: o componente de mapa contínua da mesma forma, exceto que seu estado agora possui apenas coordenadas referentes ao município selecionado.

Figura 15 – Tela por município de um estado do *DASHCARE*



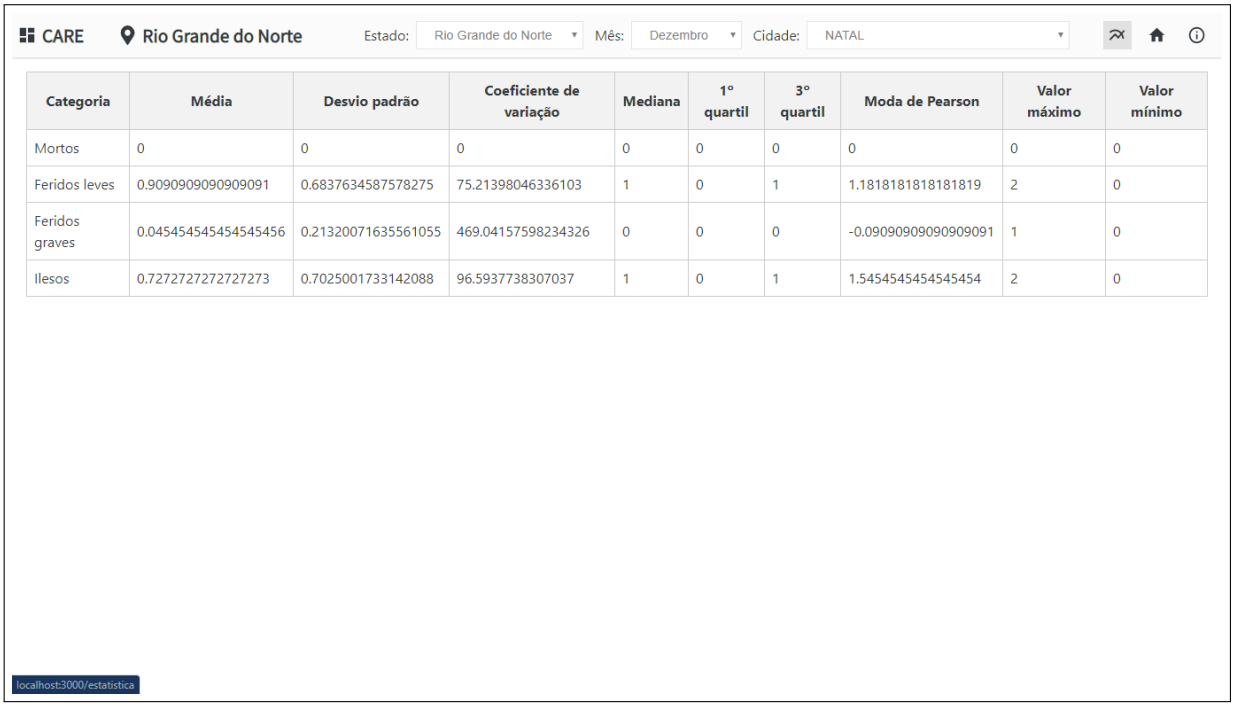
Fonte: Próprio Autor.

A ferramenta *DASHCARE*, também disponbiliza uma tela para a o cálculo de algumas funções estatístcas (Figura 16) para os dados do tipo quantidade presentes no universo de discurso. Essa tela pode ser acessada a partir da tela do município, onde um opção de "Estatísticas" é desbloqueada no componente de barra de navegação.

As estatísticas apresentadas são definidas a seguir de acordo com (SPIEGEL; STEPHENS, 2000): (i) *média*, um valor representativo ou típico de um conjunto de dados; (ii) *desvio padrão*, indica quanto um conjunto de dados é uniforme; (iii) *coeficiente de variação*, fornece a variação dos dados com relação a *média*; (iv) *mediana*, é o valor central de um conjunto de dados

ordadenos por grandeza; (v) *quartis*, a extensão lógica da *mediana* em quatro partes iguais; (vi) *moda de Pearson*, valor que maior frequência em um conjunto de dados; e (vii) *mínimo* e *maximo*, menor valor e maior valor, respectivamente, de um conjunto de dados.

Figura 16 – Tela das estatítiscas do *DASHCARE*



Categoria	Média	Desvio padrão	Coeficiente de variação	Mediana	1° quartil	3° quartil	Moda de Pearson	Valor máximo	Valor mínimo
Mortos	0	0	0	0	0	0	0	0	0
Feridos leves	0.9090909090909091	0.6837634587578275	75.21398046336103	1	0	1	1.1818181818181819	2	0
Feridos graves	0.045454545454545456	0.21320071635561055	469.04157598234326	0	0	0	-0.09090909090909091	1	0
Ilesos	0.7272727272727273	0.7025001733142088	96.5937738307037	1	0	1	1.5454545454545454	2	0

Fonte: Próprio Autor.

Os componentes do *DASHCARE* foram projetados para encapsularem a lógica de comunicação com o servidor. Assim, cada componente, ao ser renderizado, faz uma requisição internamente, requerendo os dados específicos de seu domínio.

6.3 CONCLUSÃO

Este Capítulo apresentou os resultados obtidos, dada a implementação do *DASHCARE*. Primeiro, foi discorrido o resultado da importação dos dados para o MongoDB. Num segundo momento, foi apresentado o funcionamento das rotas da API para requisição e obtenção dos dados, conseguidos com o resultado do módulo de serviço. E por último, foi descrito as informações visuais, que aparecem em forma de gráficos, descrição textual e cálculos estatísticos.

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste estudo, foi explorado o objetivo do desenvolvimeto de um *dashboard* entitulado *DASHCARE*, visando a criação de uma ferramenta que possa contribuir para um melhor entendimento dos Dados Abertos (DAs) da Polícia Rodoviária Federal (PRF) no Brasil, colaborando com a tomada de decisão em soluções que possam vir a combater com o alto índice de acidentes. Este trabalho teve como indagação "A partir do conjunto de DAs da PRF, como criar um dashboard flexível e eficiente, que provenha informação clara e objetiva acerca desses dados, e também seja acessível a qualquer entidade interessada". Para resolução desta pergunta, foi pensado em um processo que envolveu o desenvolvimento de três módulos: um para armazenamento dos dados, um de servidor e outro para visualização das informações.

O *DASHCARE* foi desenvolvido como uma ferramenta Web extensível. O modelo básico de sua arquitetura seguiu uma estrutura baseada em Sistema de Tempo Real (STR) com uma abordagem em *Big Data*. A partir disso, com as ferramentas apropriadas, foi unificado o processo de criação, o que tornou o resultado final confiável, extensível e rápido. Dessa forma, a partir dos resultados apresentados, conclui-se que o objetivo foi alcançado, porém não se pode atestar sua total eficiência, pois não foi possível implantar a solução em um ambiente de utilização real.

7.1 CONTRIBUIÇÕES DO TRABALHO

As principal contribuição deste trabalho é disponibilização de uma ferramenta gratuita que pode ser utilizada como subsídio para ajudar no processo de tomadas de decisões de um problema real do Brasil, que é o alto número de acidentes nas Rodovias Federais, tudo isso a partir de uma base de dados pública e de propósito específico. Além disso, é esperado que este trabalho possa: (i) propiciar um melhor entendimento sobre o problema da mobilidade nas estradas federais; (ii) permitir o acompanhamento do cenário da problemática de um sistema em tempo real, com a devida aplicação da solução; e (iii) estimular a criação de mais ferramentas que utilizem bases de DAs e crie o fomento pela inovação.

7.2 LIMITAÇÕES

As limitações envolvidas no decorrer deste trabalho, estão relacionadas a impossibilidade de processamento dos dados em uma abordagem distribuída, e dessa forma realizar experimentos

mais sofisticados quanto a um volume maior de dados. Outra limitação também, se deu a fato do não acesso a um módulo ETL real, para observar como o sistema lidaria com a população do banco de forma automática.

7.3 TRABALHOS FUTUROS

Este trabalho criou um base tecnológica para monitoramento de DAs de nicho específico, então fica como ideia para trabalhos futuros: (i) criação de mais modalidades de visualização de dados; (ii) aplicação de técnicas estatíscas para correlação dos dados; (iii) integração de um módulo para mineração de dados sobre o universo de discurso abordado; e (iv) adaptação do módulo de servidor para outras bases de DAs.

REFERÊNCIAS

AGGARWAL, C. C. **Data streams: models and algorithms**. [S.l.]: Springer Science & Business Media, 2007. v. 31.

ANDRADE, F. R. d.; ANTUNES, J. L. F. Tendência do número de vítimas em acidentes de trânsito nas rodovias federais brasileiras antes e depois da década de ação pela segurança no trânsito. **Cadernos de Saúde Pública**, SciELO Public Health, v. 35, p. e00250218, 2019.

BEGOLI, E.; HOREY, J. Design principles for effective knowledge discovery from big data. In: IEEE. **2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture**. [S.l.], 2012. p. 215–218.

BRASIL. **Institui o Código de Trânsito Brasileiro**. 2019. Disponível em: <https://www2.camara.leg.br/legin/fed/lei/1997/lei-9503-23-setembro-1997-372348-publicacaooriginal-1-pl.html>. Acesso em: 3 out. 2019.

CHEN, C. P.; ZHANG, C.-Y. Data-intensive applications, challenges, techniques and technologies: A survey on big data. **Information sciences**, Elsevier, v. 275, p. 314–347, 2014.

CHEN, J.; CHEN, Y.; DU, X.; LI, C.; LU, J.; ZHAO, S.; ZHOU, X. Big data challenge: a data management perspective. **Frontiers of Computer Science**, Springer, v. 7, n. 2, p. 157–164, 2013.

DEAN, J.; GHEMAWAT, S. Mapreduce: a flexible data processing tool. **Communications of the ACM**, ACM, v. 53, n. 1, p. 72–77, 2010.

DIANA, M. D.; GEROSA, M. A. Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. In: **IX Workshop de Teses e Dissertaç oes em Banco de Dados**. [S.l.: s.n.], 2010. v. 9.

ELMASRI, R.; NAVATHE, S. B.; PINHEIRO, M. G. *et al.* Sistemas de banco de dados. Pearson Addison Wesley São Paulo, 2005.

ELOQUENT JAVASCRIPT. **Introduction**. 2019. Disponível em: <https://eloquentjavascript.net/00_intro.html>. Acesso em: 5 de novembro de 2019.

FARINES, J.-M.; FRAGA, J. d. S.; OLIVEIRA, R. d. Sistemas de tempo real. **Escola de Computação**, v. 2000, p. 201, 2000.

G1. **Acidentes em rodovias federais mataram mais de 83 mil pessoas no Brasil em 10 anos**. 2018. Disponível em: <https://g1.globo.com/df/distritofederal/noticia/2018/12/06/aidentes-em-rodovias-federais-mataram-mais-de-83-mil-pessoasno-brasl-em-10-anos.ghtml>. Acesso em: 3 out. 2019.

GANDOMI, A.; HAIDER, M. Beyond the hype: Big data concepts, methods, and analytics. **International journal of information management**, Elsevier, v. 35, n. 2, p. 137–144, 2015.

GIL, A. C. Como elaborar projetos de pesquisa. **São Paulo**, v. 4, 2002.

GILLENSON, M. L. Fundamentos de sistemas de gerência de banco de dados. **Rio de Janeiro: LTC**, 2006.

GOLDSCHMIDT, R.; PASSOS, E. **Data mining: um guia prático**. [S.l.]: Gulf Professional Publishing, 2005.

JOAZEIRO, S. A. Desenvolvimento de dashboard dinâmico de projetos utilizando o processo de business intelligence sobre um sistema de solicitação de serviço em um ambiente governamental. Universidade Federal de Mato Grosso, 2015.

KAISLER, S.; ARMOUR, F.; ESPINOSA, J. A.; MONEY, W. Big data: Issues and challenges moving forward. In: IEEE. **2013 46th Hawaii International Conference on System Sciences**. [S.l.], 2013. p. 995–1004.

KOPETZ, H. **Real-time systems: design principles for distributed embedded applications**. [S.l.]: Springer Science & Business Media, 2011.

MARQUESONE, R. **Big Data: Técnicas e tecnologias para extração de valor dos dados**. [S.l.]: Editora Casa do Código, 2016.

MCAFEE, A.; BRYNJOLFSSON, E.; DAVENPORT, T. H.; PATIL, D.; BARTON, D. Big data: the management revolution. **Harvard business review**, v. 90, n. 10, p. 60–68, 2012.

MCCREARY, D.; KELLY, A. Making sense of nosql. **Shelter Island: Manning**, p. 19–20, 2014.

MDN. **CSS**. 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>. Acesso em: 5 de novembro de 2019.

MDN. **HTML: Linguagem de Marcação de Hipertexto**. 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em: 5 de novembro de 2019.

MDN. **Sobre JavaScript**. 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript>. Acesso em: 5 de novembro de 2019.

MINISTÉRIO DA INFRAESTRUTURA. **Frota de Veículos - 2019**. 2019. Disponível em: <https://infraestrutura.gov.br/component/content/article/115-portal-denatran/8559-frota-de-veiculos-2019.html>. Acesso em: 8 out. 2019.

MONGODB. **MongoDB Architecture**. 2019. Disponível em: <https://www.mongodb.com/mongodb-architecture>. Acesso em: 8 out. 2019.

MONIRUZZAMAN, A.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. **arXiv preprint arXiv:1307.0191**, 2013.

NODE.JS. **Sobre Node.js**. 2019. Disponível em: <https://nodejs.org/pt-br/about/>. Acesso em: 5 de novembro de 2019.

POLÍCIA RODOVIÁRIA FEDERAL. **Dados Abertos**. 2019. Disponível em: <https://portal.prf.gov.br/dados-abertos>. Acesso em: 7 out. 2019.

REACT. **React**. 2019. Disponível em: <https://pt-br.reactjs.org/>. Acesso em: 5 de novembro de 2019.

SADALAGE, P. J.; FOWLER, M. **NoSQL Essencial: Um guia conciso para o Mundo emergente da persistência poliglota**. [S.l.]: Novatec Editora, 2019.

SANTOS, L. M. d. Criação de um dashboard para monitoramento de perfis de qualidade de software. 2017.

SCHMIDT, E. L. *et al.* O sistema de transporte de cargas no brasil e sua influência sobre a economia. Florianópolis, 2011.

SILVA, R. F. d. **Desenvolvimento de dashboard para análise de dados de agronegócio**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2018.

SPIEGEL, M. R.; STEPHENS, L. J. **Estatística: Coleção Schaum**. [S.l.]: Bookman, 2000.

STACK OVERFLOW. **Developer Survey Results 2018**. 2018. Disponível em: <https://insights.stackoverflow.com/survey/2018>. Acesso em: 5 de novembro de 2019.

WANG, L.; WANG, G.; ALEXANDER, C. A. Big data and visualization: methods, challenges and technology progress. **Digital Technologies**, v. 1, n. 1, p. 33–38, 2015.

ZHENG, K.; ZHENG, Y.; YUAN, N. J.; SHANG, S.; ZHOU, X. Online discovery of gathering patterns over trajectories. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 26, n. 8, p. 1974–1988, 2013.

ZUIDERWIJK, A.; JANSSEN, M.; CHOENNI, S.; MEIJER, R. Design principles for improving the process of publishing open data. **Transforming Government: People, Process and Policy**, Emerald Group Publishing Limited, v. 8, n. 2, p. 185–204, 2014.