IGOR EMANUEL CARVALHO DA CRUZ

DESENVOLVIMENTO DE APLICAÇÕES WEB:

ESTÁGIO AGÊNCIA ZETTA

LAVRAS – MG

2021

IGOR EMANUEL CARVALHO DA CRUZ

DESENVOLVIMENTO DE APLICAÇÕES WEB:
ESTÁGIO AGÊNCIA ZETTA

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Dr. Erick Galani Maziero
Orientador

LAVRAS – MG

2021

IGOR EMANUEL CARVALHO DA CRUZ

DESENVOLVIMENTO DE APLICAÇÕES WEB: ESTÁGIO AGÊNCIA ZETTA

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 05 de maio de 2021.

Prof. DSc. Erick Galani Maziero	UFLA
Prof. DSc. Ricardo Terra Nunes Bueno Villela	UFLA
Pâmela Andrade de Almeida	ZETTA

Prof. Dr. Erick Galani Maziero
Orientador

LAVRAS – MG
2021

AGRADECIMENTOS

Agradeço aos meus familiares que me apoiaram nesta caminhada em busca de minha formação acadêmica. Agradeço de forma mais especial a minha mãe Vera Lúcia Teixeira de Carvalho que me apoiou no início dessa jornada. Agradeço aos meus amigos, que me deram apoio, ajuda e conselhos, permanecendo ao meu lado em momentos fáceis e difíceis. Agradeço de forma especial a minha amiga Raiane Manoel e ao meu namorado Lucas Pacheco por me dar forças para nunca desistir. Agradeço também aos professores e colaboradores da UFLA que se dedicaram em tornar a graduação rica e de alta qualidade, agradeço de forma especial ao professor Erick Galani Maziero e ao colaborador Vitor Anacleto Rodarte Andrade por contribuir com meu aprendizado para além das salas de aula.

RESUMO

O presente documento descreve as atividades desenvolvidas durante o estágio supervisionado na área de desenvolvimento de software na Agência Zetta. Durante o estágio, o discente teve a oportunidade de aprimorar e incrementar os conhecimentos adquiridos no decorrer do curso de Ciência da Computação. O estágio proporcionou o aprendizado em sistemas web, e desta forma, preparou o discente para se tornar um profissional qualificado no mercado de trabalho. No decorrer deste documento, foram apresentadas as ferramentas, metodologias e tecnologias aprendidas no estágio. Dentre os *frameworks* utilizados, vale citar principalmente o SpringBoot para desenvolvimento *back-end*, e Vue.js e o ReactJS em desenvolvimentos *front-end*. Ao final desta monografia, são realizadas considerações sobre as contribuições de diversas disciplinas do curso para as atividades do estudante.

Palavras-chave: *back-end*, *front-end*, SpringBoot, ReactJs, VueJs, JavaScript, Java, SCRUM.

ABSTRACT

This document describes the activities developed during the supervised internship in the area of software development at Agência Zetta. During the internship, the student had the opportunity to improve and increase the knowledge acquired during the course of Computer Science. The internship provided the learning in web systems, and in this way, prepared the student to become a qualified professional in the job market. Throughout this document, the tools, methodologies and technologies learned in the internship will be presented. Among the frameworks used, it is worth mentioning SpringBoot for *back-end* development, and Vue.js and ReactJs in *front-end* developments. At the end of this monograph, considerations are made considering the course subjects.

Keywords: *back-end*, *front-end*, SpringBoot, ReactJs, VueJs, JavaScript, Java, SCRUM.

LISTA DE FIGURAS

Sumário

1 INTRODUÇÃO

Este capítulo tem o objetivo de realizar uma breve apresentação do assunto tratado neste trabalho. Esse capítulo aborda uma introdução da Agência Zetta, o objetivo do projeto desenvolvido durante o estágio, a atuação do estagiário dentro do projeto e, ao final, uma visão geral dos próximos capítulos deste documento.

1.1 Contextualização

Segundo a página de empreendedorismo Page (2020), a área de TI (Tecnologia de Informação) sofreu um grande impacto com o início da pandemia do Covid-19. A pandemia forçou empresas a acelerarem seus projetos engavetados na área de TI. Da mesma forma, incentivou essas empresas a se adaptarem à alta demanda de projetos. Em contrapartida, os profissionais de diversas áreas se viram obrigados a aperfeiçoar suas habilidades digitais. Por outro lado, a busca por profissionais de todas as áreas de desenvolvimentos cresceu exponencialmente.

Ainda segundo Page (2020), com essa revolução de trabalho remoto, a tecnologia passou a estar no centro do desenvolvimento de negócios. A adoção de novos sistemas de desenvolvimento, tecnologias, softwares, plataformas, sistemas, automações e máquinas foram necessárias para dar continuidade no trabalho dos colaboradores em *home office*. O crescimento da área de TI que vinha aumentando no decorrer dos anos subiu aceleradamente em 2020.

Em conjunto, as grandes oportunidades provindas na área da computação, somadas ao interesse em aprender e conhecer mais do desenvolvimento de software, incentivaram o estagiário a iniciar um estágio supervisionado na Agência Zetta. O estágio é na área de desenvolvimento de aplicações Web, o que proporcionará ao aluno conhecimento, experiência e vivência profissional.

1.1.1 A Agência Zetta

Segundo DCOM UFLA (2020) A Zetta - Agência UFLA de Inovação em Geotecnologias e Sistemas Inteligentes no Agronegócio foi apresentada pela UFLA como sendo sua quarta agência de inovação. A criação da Zetta foi proposta pela Pro-Reitoria de Pesquisa (PRP).

A Zetta nasceu a partir do antigo LEMAF - Laboratório de Estudos e Projetos em Manejo Florestal. O LEMAF foi responsável pelo desenvolvimento de vários projetos, como o Sistema Nacional de Cadastro Ambiental Rural (Sicar), o Zoneamento Ecológico-Econômico do Estado de Minas Gerais e também do Espírito Santo (ZEE), o Manejo Sustentável da Candeia, o Inventário Florestal de Minas Gerais e o Modelo Fitogeográfico da Bacia do Rio Grande e do São Francisco, o Manejo do Cerrado para Usos Múltiplos (DCOM UFLA, 2020). Segundo o DCOM UFLA (2020), o LEMAF continuará seguindo com a missão de desenvolver projetos na área de Manejo Florestal, enquanto, por outro lado, a Zetta focará com o desenvolvimento de sistemas inteligentes e geotecnologias.

A Zetta é organizada em quatro equipes de desenvolvimento de software, denominadas Tribos. Cada tribo é compostas por *squads*. *Squads* são pequenas divisões dentro da Tribo responsáveis por planejar e executar atividades de forma independente uma das outras. Os *squads* são compostos por um conjunto de colaboradores que realizam funções distintas, tais como: os desenvolvedores, os analistas de qualidades ou *testers* e o *Product Owner*.

1.1.2 Objetivos

Este trabalho tem o objetivo de descrever a trajetória do estagiário na Zetta, com o período de estágio de Agosto/2020 a Março/2021. Neste período, o estagiário atuou nos seguintes projetos: Sistema de Gestão de Sprints, Observatório da Agropecuária Brasileira e PronaSolos.

1.1.2.1 O projeto: Gestão de *Sprints*

O Gestão de *Sprints* é um projeto desenvolvido para uso interno da Zetta. *Sprints* possuem um prazo de conclusão definido previamente e trata-se de uma lista de funcionalidades que serão desenvolvidas neste prazo (SAVOINE et al., 2009). Dito isso, uma aplicação de gestão de *sprints* precisa conter dados como: projeto, datas de início e término, PO (*Product Owner*) responsável, custos e ordens de serviços, por exemplo. Desta forma, o projeto Gestão de *Sprints* visa desenvolver uma aplicação que gerencie as *sprints* da Agência Zetta (DCOM UFLA, 2020).

1.1.2.2 O projeto: Hotsite PronaSolos

Hotsite é um local de acesso rápido e prático para reunir informações de um assunto em específico. Para o projeto, PronaSolos a ideia é disponibilizar um local que garanta visibilidade e dê acesso facilitado a Plataforma PronaSolos. Outro objetivo do hotsite é trazer informações e curiosidades sobre o Programa PronaSolos, seus objetivos e impactos, e incluir de forma lúdica e educativa outras informações sobre os solos no Brasil.

1.1.2.3 O projeto: Observatório da Agropecuária Brasileira

O Observatório da Agropecuária Brasileira é um projeto que visa a integração das bases de dados geoespaciais do setor agropecuário, possibilitando uma visualização em painéis dinâmicos que reúnem informações (*Business Intelligence*), tais como: (i) imagens de satélite de resoluções variáveis, (ii) dados não espaciais que complementam o entendimento das cadeias produtivas do setor oriundas de múltiplas fontes e (iii) informações geradas por modelos espaciais complexos e simulação de cenários considerando variáveis do meio físico, socioeconômicas e ambientais em temas relevantes para o setor agropecuário.

1.2 Estrutura do trabalho

Na estrutura deste trabalho, o Capítulo 2 apresenta o referencial teórico, visando a apresentação das tecnologias e metodologias aprendidas. o Capítulo 3 descreve as atividades desenvolvidas pelo estagiário. Por fim, Capítulo 4 apresenta as considerações finais deste relatório de estágio.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta as principais bibliotecas, metodologias e tecnologias de desenvolvimento aderidas.

2.1 Metodologias de Desenvolvimento

No intuito de reunir e organizar as atividades da equipe em um mesmo projeto, é necessário adotar formas de organizar, dividir e comunicar o andamento de cada atividade do projeto. Desta forma, nesta sessão são apresentadas duas metodologias de desenvolvimento adotadas no estágio.

2.1.1 Kanban

Segundo Presotto (2021), o Kanban traduzido do japonês como **cartão** ou **sinal**, é comumente conhecido como uma metodologia de desenvolvimento organizacional e de gestão visual. O Kanban, em outras palavras, trata de um controle de tarefas para uma equipe e pode ser adaptado de acordo com a necessidade do usuário.

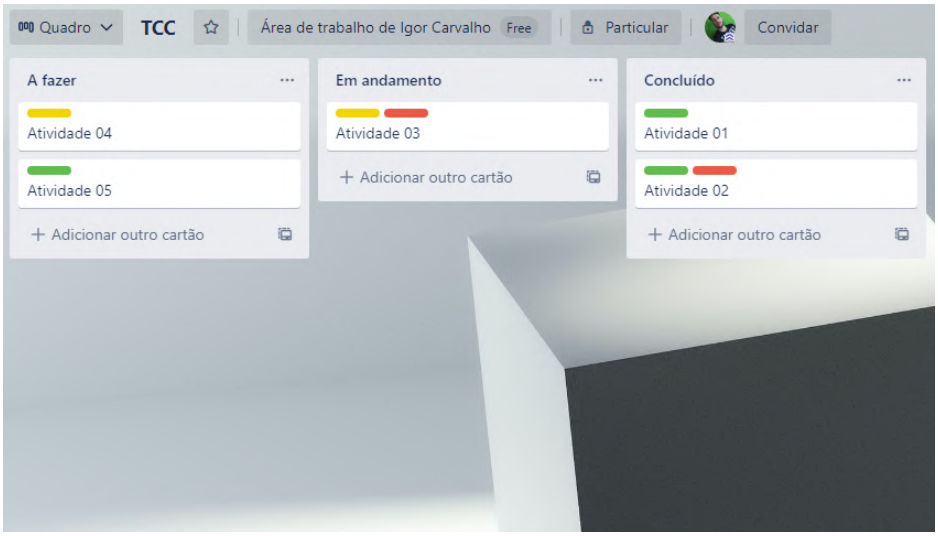
Ainda segundo Presotto (2021), o Kanban foi inicialmente empregado com uso de *post-its* ou cartões coloridos organizados em um quadro branco para gerenciar estoques. Atualmente é usado de diversas formas inclusive no uso de gestão de projetos e de produtividade.

O funcionamento do Kanban é provindo de três partes principais: O **cartão** em que são descritas as tarefas ou atividades; pelo menos três **colunas** - A fazer (*To Do*), Em execução (*Doing*) e Concluído (*Done*) - o que indica o estado da tarefa e por último o **quadro** em que são organizados as colunas e os cartões. Cada quadro pode indicar um ou mais projetos(ESPINHA, 2019).

A Figura 2.1 exemplifica um *Kaban* feito por meio da aplicativo de gerenciamento de projetos Trello[1]. Esse exemplo apresenta o quadro *TCC* composto

[1] https://trello.com/pt-BR

Figura 2.1 – Exemplo de Kanban



Fonte: Do Autor, 2021

pelas três colunas principais e cinco atividades dispostas, conforme seu desenvolvimento.

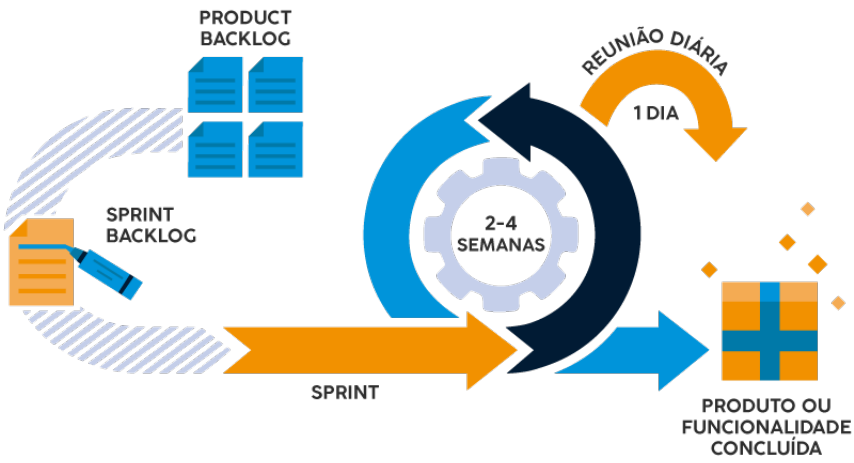
2.1.2 Scrum

O Scrum é uma metodologia de desenvolvimento ágil. Essa metodologia tem o objetivo principal de apresentar um processo de desenvolvimento para projetos complexos e adaptativos. O Scrum é amplamente utilizado em desenvolvimento de software, introduzindo as ideias de flexibilidade, adaptabilidade e produtividade (SOARES, 2004).

Ainda segundo Soares (2004), o Scrum é estruturado com equipes pequenas (*Squads*), constituídas de até dez pessoas, utilizado em projetos em que os requisitos de desenvolvimentos não são muito estáveis e desenvolvidos em curtas iterações de tempo, denominadas *Sprints*. Os *Squads* são formados pelos desenvolvedores, *Product Owner* (Dono do produto) e Analista de qualidade. É o *Squad* quem define o tempo e a demando atendida para cada *Sprint*. As *Sprints* podem ter um duração de até trinta dias e possuem uma demanda de atividades fixas.

A metodologia *Scrum* é composta por três ritos principais. O primeiro dos ritos é a Reunião de planejamento, ou *planning*. No *Planning*, são descritos os requisitos do produto, quebrados por atividades, em um documento nomeado *Product Backlog*. Após a quebra, são feitas as estimativas de esforço de desenvolvimento de cada item no *Sprint Backlog*. No segundo rito, o desenvolvimento, são identificadas novas variáveis até então não percebidas. O controle do *Scrum* é feito de forma contínua e flexível facilitando o acompanhamento das mudanças. Na próxima *Sprint* são inseridos os novos requisitos percebidos. E, por último, no rito *review* são feitas as reuniões para analisar o progresso do projeto e demonstrar o produto para o cliente (SOARES, 2004).

Figura 2.2 – Esquema de funcionamento SCRUM



Fonte: Tecnicon, 2021

No Scrum, são realizadas reuniões de acompanhamento diárias (*dailies*). Essas reuniões possuem duração de aproximadamente 15 minutos. Durante a *daily*, cada membro do *squad* expõe o trabalho realizado desde a última *daily*, bem como as dificuldades encontradas, impedimentos e um breve apontamento

das próximas atividades a ser realizadas. O esquema apresentado na Figura 2.2 ilustra o funcionamento e principais ritos do *Scrum*.

2.2 Bibliotecas e Tecnologias utilizadas

Na intenção de facilitar e auxiliar o desenvolvedor durante o desenvolvimento de produtos, são disponibilizados diversas *frameworks* e/ou bibliotecas na Internet. Essas bibliotecas podem ser disponibilizadas para uso no *front-end* ou no *back-end* de um aplicação. Desta forma, esta seção descreve as principais bibliotecas e tecnologias aprendidas e utilizadas no desenvolvimento dos projetos durante o estágio.

2.2.1 Tecnologias *front-end*

O *front-end* é responsável por realizar a interação do usuário no uso de alguma aplicação ou sistema. No intuito de melhorar a comunicação entre máquina e usuário, são disponibilizados *frameworks* de *front-end* que facilita o desenvolvedor em alocar, estilizar, ordenar e manipular objetos gráficos de forma livre enquanto interagem com *back-end* do sistema. A seguir, serão listados as principais tecnologias e *frameworks* de *front-end* utilizados.

2.2.1.1 HTML

O *HyperText Markup Language*, ou HTML como é mais comumente conhecido, foi desenvolvido no intuito de criar uma linguagem que fosse acessível e compreendida por todos os tipos de computadores. Desta forma, o HTML foi desenvolvido para permitir que os desenvolvedores incluam em suas páginas publicações, documentos, título, textos, tabelas, listas e imagens por exemplo (RAGGETT et al., 1999).

Um site pode ser formado por páginas HTML. As páginas HTML são escritas com base em diversas *tag's* pré-definidas que são os componentes que

compõem cada elemento da página. As *tag's* possuem a seguinte estrutura ***<tag-name>*** para dar inicio ao bloco e ***</tag-name>*** para fechar o bloco (MARQUES, 2020).

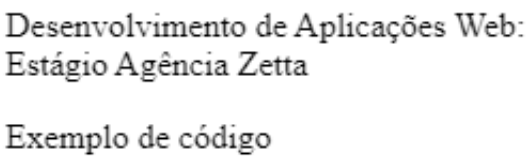
Figura 2.3 – Exemplo de código HTML

```
tcc.html > ...
1  <!DOCTYPE html>
2  <html>
3
4      <head>
5
6      </head>
7
8      <body>
9
10         <p class="titulo">
11             Desenvolvimento de Aplicações Web:
12             <br/>
13             Estágio Agência Zetta
14         </p>
15
16         <p id="body">
17             Exemplo de código
18         </p>
19
20     </body>
21
22 </html>
23
```

Fonte: Do Autor, 2021

A Figura 2.3 mostra um exemplo de código HTML, em que a *tag body* comporta o conteúdo da página. Esta página é composta por duas *tag's p*, a primeira sendo uma classe **titulo** e a segunda um *id* **body**. A Figura 2.4 mostra o resultado do código exibido em um navegador Web.

Figura 2.4 – Página HTML simples

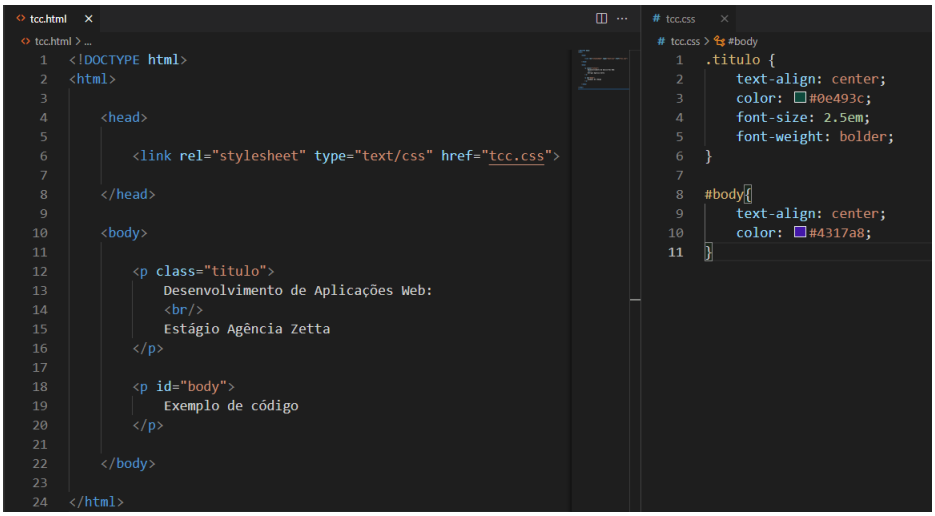


Fonte: Do Autor, 2021

2.2.1.2 CSS

O CSS ou *Cascade Style Sheet*, é uma estrutura utilizada para estilizar páginas HTML. A estrutura do CSS define a criação de uma regra. Cada regra é composta por um seletor e suas propriedades/valores. O seletor informa a qual componente a regra em questão referencia e as propriedades indicam a palavra-chave que especificam o efeito do estilo. (MILETTO; BERTAGNOLLI, 2014)

Figura 2.5 – Exemplo de código CSS



Fonte: Do Autor, 2021

Dando sequência ao código HTML apresentado na Figura 2.3 temos um exemplo da inserção de estilo CSS, apresentado na Figura 2.5. Neste código exibido no lado esquerdo pode-se notar a inserção da linha 4 com a *tag* <*link*>. Essa *tag* faz a referência ao arquivo *tcc.css*, ao lado direito, que contêm as propriedades CSS. Ainda olhando o código à esquerda temos a referência a um ID *body* na linha 18, indicado pelo *.(ponto)* antes do nome; e uma classe na linha 8 *titulo* indicado pelo *(cerquilha)* antes do nome. O id *titulo* personaliza o texto para centralizar o conteúdo alterando a cor da fonte, a largura da fonte e o tamanho. Já a classe *body* centraliza e altera a cor da fonte. Na Figura 2.6 é possível visualizar o resultado em um navegador.

Figura 2.6 – Página HTML estilizada com CSS

Fonte: Do Autor, 2021

2.2.1.3 JavaScript

O JavaScript é uma linguagem de programação criada em 1995 referência principalmente o desenvolvimento Web. O JavaScript faz parte de um trio de ferramentas para o desenvolvimento Web, juntamente com HTML e CSS. O HTML especifica o conteúdo das paginas WEB, o CSS cuida da estética destas paginas e o JavaScript as ações e comportamentos dos componentes na página (FLANAGAN, 2004).

O JavaScript é uma linguagem de alto nível, dinâmica, interpretada e não tipada, conveniente para uso na programação orientada a objetos e funcionais. O código JavaScript pode ser inserido no HTML entre as *tag* <script></script> ou importado de outro arquivo por meio da *tag* <*link*> (FLANAGAN, 2004).

O código apresentado na Figura 2.7 continua o desenvolvimento do exemplo, porém, desta vez, é acrescentada a *tag* <*script*></*script*>. Entre essa *tag* indica que há um código JavaScript. Para esse exemplo, foram inseridas duas linhas de código, a primeira cria uma variável e a inicializa com o valor retornado pelo construtor *Date()*. Esse construtor preenche a variável com as informações de data e horário atual. Na Figura 2.8 vê-se o resultado da substituição do conteúdo no ID *body* pelo valor da variável *time*.

Figura 2.7 – Exemplo de código JavaScript

Fonte: Do Autor, 2021

2.2.1.4 React

Criados por engenheiros do Facebook em 2011, React é uma biblioteca JavaScript desenvolvido para interfaces complexas em que a base de dados muda constantemente. O React teve seu código aberto para a comunidade em 2013, o que contribuiu para sua popularização e refinamento (BRASIL, 2018).

A função do React é organizar os componentes da interface, preocupando apenas com o que será exibido para o usuário, sem se preocupar com a implementação de cada componente. Outro ponto que vale ser citado é que o React trabalha

Figura 2.8 – Página com uso de função JavaScript

Fonte: Do Autor, 2021

com o DOM (*Document Object Model*) e atualiza os componentes de acordo com seu estado. O React cria seu proprio DOM, denominado *Virtual DOM*, e o compara com o estado do componente antes de atualização, desta forma, atualiza somente os componentes necessários e não a página como um todo (BRASIL, 2018).

Figura 2.9 – Exemplo de código React

```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Lista de compras para {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}

// Exemplo de uso: <ShoppingList name="Mark" />
```

Fonte: reactjs.org, 2021

A Figura 2.9 representa um código que implementa um componente *React*. O componente é uma lista de compras para uma determinada pessoa. Na linha 5, está o trecho ***this***.*props.name*. Este trecho indica que na criação do componente

existirá uma propriedade *name* com o nome do responsável pela lista de compras associado, conforme exemplo na última linha da Figura 2.9.

2.2.1.5 Material UI

Material-UI é um projeto de código aberto licenciado pelo MIT. O Material-UI é uma biblioteca para React, baseada no Google Material Design. Esse framework trabalha com uso de *grids* para organizar os componentes na tela. A biblioteca possui diversos componentes prontos facilitando o uso e estilização das páginas (AMBERJ, 2019).

Figura 2.10 – Guia de instalação e uso Material-UI

Fonte: material-ui.com, 2021

A Figura 2.10, retirada da documentação oficial do *Framework Material-UI* demonstra um breve guia de instalação e uso. O exemplo de código à direita da Figura 2.10 trata-se da inserção de um botão na tela.

2.2.1.6 Vue.js

O Vue.js foi criado por Evan You enquanto trabalhava em um dos projetos do Google Creative Labs. A repetição de vários trechos de código em páginas HTML motivou a criação da linguagem Vue. Desta forma, deu-se a criação do Vue.js, um *framework* de prototipagem rápida, que permite uma maneira fácil e

flexível de ligação dos dados reativos e componentes reutilizáveis (GALDINO, 2017).

O Vue.js é um *framework* progressivo desenvolvido para criação de interfaces de usuários. Diferente de *frameworks* monolíticos, o Vue.js foi criado para ser incremental e permite a integração de outras bibliotecas JavaScript. O Vue.js tambem é um projeto que segue a estruta de desenvolvimento *Single Page Application* (GALDINO, 2017).

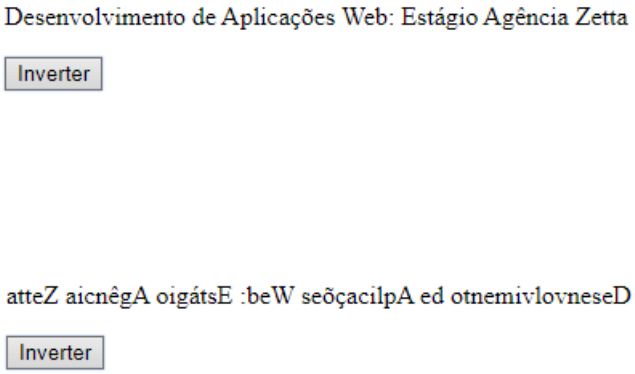
Figura 2.11 – Exemplo de código Vue.js

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5          <title>Minha página de teste</title>
6      </head>
7      <body>
8
9          <div id="app">
10             <p>{{ texto }} </p>
11             <button @click="inverterCaracteres">Inverter</button>
12         </div>
13
14         <script src="https://cdn.jsdelivr.net/npm/vue@2.x/dist/vue.js"></script>
15
16         <script>
17
18            var app = new Vue({
19                el: '#app',
20                data: {
21                    texto: 'Desenvolvimento de Aplicações Web: Estágio Agência Zetta'
22                },
23                methods: {
24                    inverterCaracteres: function (){
25                        this.texto = this.texto.split('').reverse().join('')
26                    }
27                }
28             })
29
30         </script>
31
32     </body>
33 </html>
```

Fonte: Do Autor, 2021

O código representado na Figura 2.11 exemplifica a utilização do Vue.js. Esse código inverte os caracteres contidos na variável *texto* na linha 24 ao clicar no botão na linha 13, vide Figura 2.12.

Figura 2.12 – Exemplo de componente Vue.js no navegador



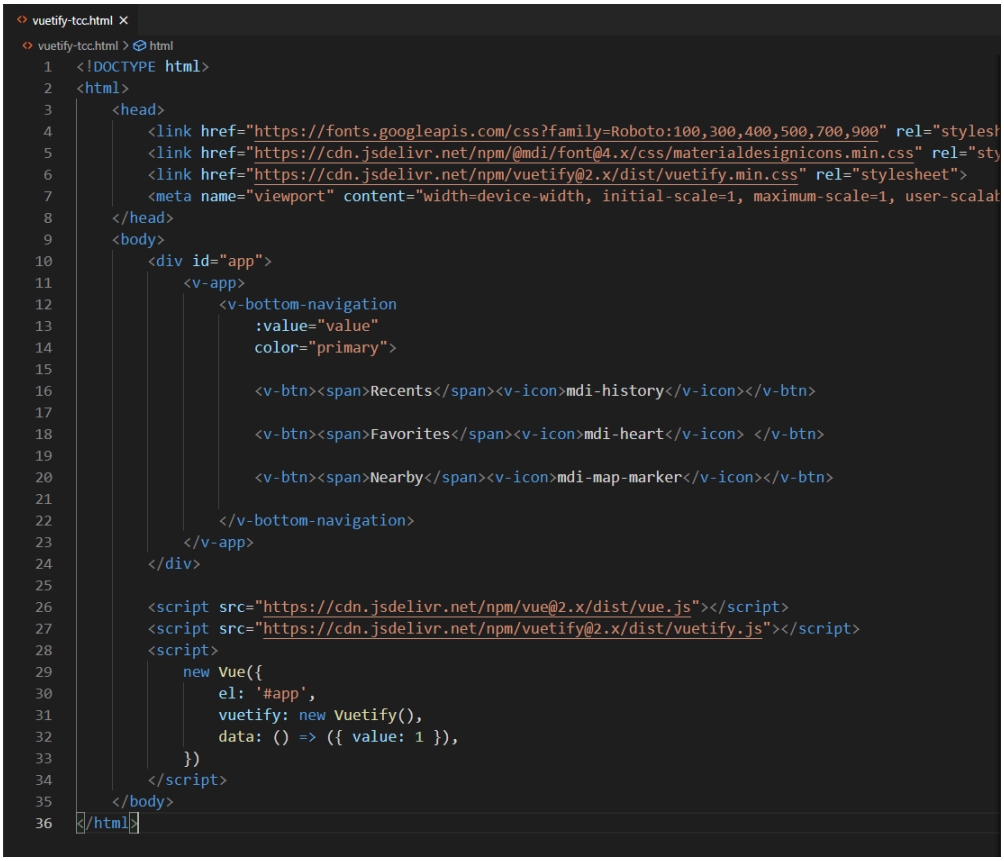
Fonte: Do Autor, 2021

2.2.1.7 Vuetify

O Vuetify é um *framework* criado para desenvolvimento de sites juntamente do Vue.js com base no Material Design. Esse *framework* compõe uma boa quantidade de componentes pré-criados que auxiliam na criação de páginas web (SCHMITZ, 2018).

A Figura 2.13 mostra um código Vue.js com uso do *framework* Vuetify. A importação da biblioteca do *Vuetify* está nas linhas de 4 a 7 e 44. Após importados, é possível o uso dos componentes pré prontos do Vuetify. Esse código utiliza o componente *v-bottom-navigation* para criar um menu de navegação, conforme exibido na Figura 2.14. Nas linhas 16 a 35 os botões são inseridos, seguido do respectivo ícone de cada item do menu.

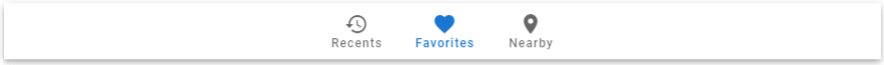
Figura 2.13 – Exemplo de código Vuetify



```
<!DOCTYPE html>
<html>
    <head>
        <link href="https://fonts.googleapis.com/css?family=Roboto:100,300,400,500,700,900" rel="stylesh
        <link href="https://cdn.jsdelivr.net/npm/@mdi/font@4.x/css/materialdesignicons.min.css" rel="sty
        <link href="https://cdn.jsdelivr.net/npm/vuetify@2.x/dist/vuetify.min.css" rel="stylesheet">
        <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalab
    </head>
    <body>
        <div id="app">
            <v-app>
                <v-bottom-navigation
                    :value="value"
                    color="primary">

                    <v-btn><span>Recents</span><v-icon>mdi-history</v-icon></v-btn>

                    <v-btn><span>Favorites</span><v-icon>mdi-heart</v-icon> </v-btn>

                    <v-btn><span>Nearby</span><v-icon>mdi-map-marker</v-icon></v-btn>

                </v-bottom-navigation>
            </v-app>
        </div>

        <script src="https://cdn.jsdelivr.net/npm/vue@2.x/dist/vue.js"></script>
        <script src="https://cdn.jsdelivr.net/npm/vuetify@2.x/dist/vuetify.js"></script>
        <script>
            new Vue({
                el: '#app',
                vuetify: new Vuetify(),
                data: () => ({ value: 1 }),
            })
        </script>
    </body>
</html>
```

Fonte: vuetifyjs.com, 2021

Figura 2.14 – Exemplo de página com o Vuetify



Fonte: Do Autor, 2021

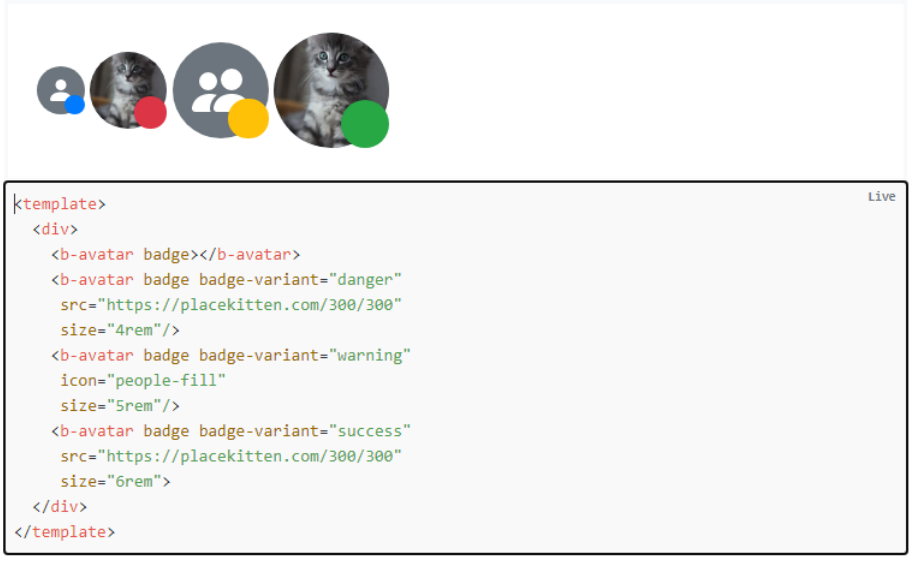
2.2.1.8 Bootstrap e BootstrapVue

Segundo Marques (2021), Bootstrap é um *framework front-end* utilizado por milhares de desenvolvedores web ao redor do mundo. Com uso do *boots-*

trap, as etapas do desenvolvimentos web se tornam mais rápidas e dinâmicas, pois esse *framework* compõe diversos elementos prontos e estilizados. O *Booststrap* é uma ferramenta para desenvolvimento com HTML, CSS e JS. Entretanto, segundo Marques (2021), o principal objetivo do *Bootstrap* é a criação de *websites* responsivos, ou seja, permite à interface do usuário ser otimizada para qualquer tamanho de tela, incluindo dispositivos móveis e telas de computadores de mesa.

Já o *Bootstrap-vue*, segundo Ceron (2020), é um *framework* que adaptou as funcionalidade já existentes no *Bootstrap* pra utilizá-las no Vue.js. Desta forma, pode-se utilizar as textitgrids para construir sistemas responsivos, importar componentes de maneira modular, usar a biblioteca de ícones padrão e/ou criar novos temas, por exemplo. Além disso, a principal vantagem é que o *Bootstrap-vue* está em constante evolução e possui muitos componentes preparados para o Vue que não existem ainda no *Bootstrap* puro.

Figura 2.15 – Exemplo *Bootstrap-Vue*



Fonte: https://bootstrap-vue.org, 2021

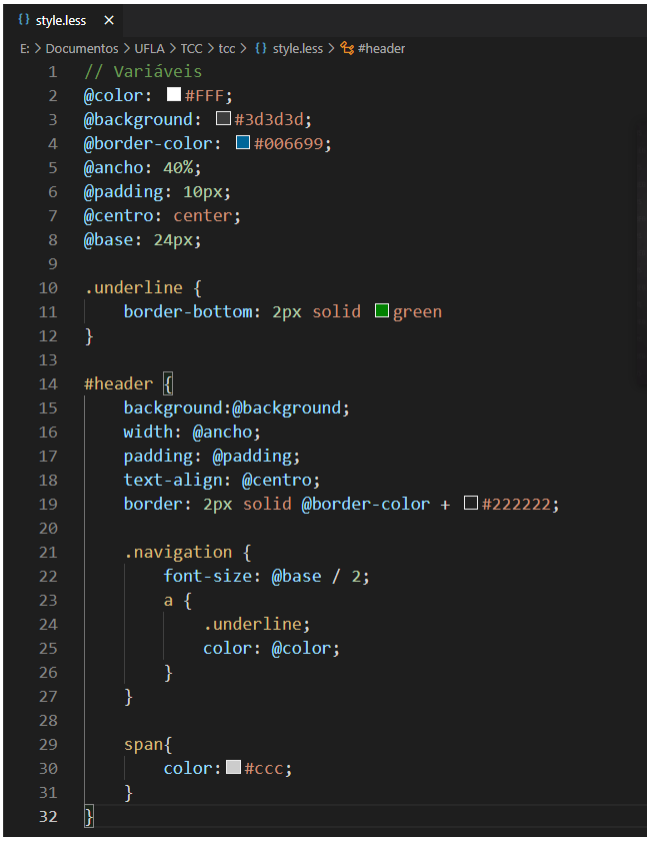
A Figura 2.15 ilustra o uso do componente avatar do *Booststrap-vue*. Na parte superior da figura tem-se a exemplificação gráfica e a na parte inferior o

código. O código mostra o uso de quatro componentes avatar com suas respectivas propriedades variando tamanho e cor da *badge*.

2.2.1.9 SASS e LESS CSS

Segundo Egidio (2020), SASS e LESS são pré-processadores que permitem ao desenvolvedor usar recursos que não existem no CSS, como variáveis, aninhamento e herança, por exemplo.

Figura 2.16 – Exemplo LESS CSS



Fonte: Do Autor, 2021

Segundo Riveros (2018), LESS foi criado para otimizar códigos CSS. Desta forma, o LESS auxilia o desenvolvedor a obter mais resultados com uma menor quantidade de código. O LESS não substitui o CSS, mas simplesmente

oferece melhorias ao desenvolvimento ao empregar o uso de variáveis, classes dinâmicas, funções de cor entre outras funcionalidades.

A Figura 2.16 ilustra um código CSS utilizando LESS. Neste código, tem-se a declaração de variáveis nas linhas de 2 a 8, a classe *underline* na linha 10 e a estilização do *id Header* a partir da linha 14. O *header* compõe o uso das variáveis declaradas no início do arquivo, operações nas linhas 19 e 22, heranças nas linhas 21, 23 e 29, além do uso da classe *underline* declarada acima.

2.2.1.10 Pug

Segundo Escudelario (2017), Pug é um facilitador para os desenvolvedores do *front-end*. O Pug é *template engine* com alto desempenho, rico em recursos e implementado em JavaScript. O Pug funciona como um intermediário para o HTML substituindo as variáveis do projeto por valores atuais e, em seguida, envia a sequência HTML resultante para o cliente. Semelhante ao SASS, o Pug é um pré processador que auxilia na realização tarefas repetitivas, fornecendo recursos não disponíveis no HTML.

Figura 2.17 – Exemplo de código HTML convertido para Pug

```
1  doctype html
2  html
3    head
4    body
5      p.titulo
6        | Desenvolvimento de Aplicações Web:
7        br
8        | Estágio Agência Zetta
9      p#body
10       | Exemplo de código
11
```

Fonte: Do Autor, 2021

A Figura 2.17 mostra o primeiro código HTML na Figura 2.3 convertido para o Pug.

2.2.1.11 GeoServer

Segundo Quadro (2009), o Geoserver é um servidor de mapas de código aberto que facilita a publicação de dados geoespaciais. Além da publicação, o Geoserver permite a inserção, atualização ou deleção de elementos por meio do serviço WFS-T (*Transactional Web Feature Service*). O foco do Geoserver é facilitar o uso e suporte de informações geográficas.

Ainda segundo Quadro (2009), o GeoServer é um sistema multiplataforma, desenvolvido em Java possuindo integração com os principais formatos vetoriais, como SQL Server e PostGIS, por exemplo.

2.2.1.12 Leaflet

Segundo Leaflet (2020), o Leaflet é uma biblioteca de código aberto baseada em JavaScript para a criação de mapas interativos. O Leaflet foi desenvolvido para uma utilização simples que entrega desempenho, performance e usabilidade.

2.2.2 Tecnologias *back-end*

Em conjunto com o *front-end* tem-se o *back-end*. O *back-end* tem a função de realizar a persistência dos dados vindos do *front-end*. Da mesma forma que o *back-end* armazena as informações provindas do *front-end* no banco de dado, ele é responsável por pegar as informações no banco de dados e disponibilizá-las para uso do *front-end* (SOUTO, 2019).

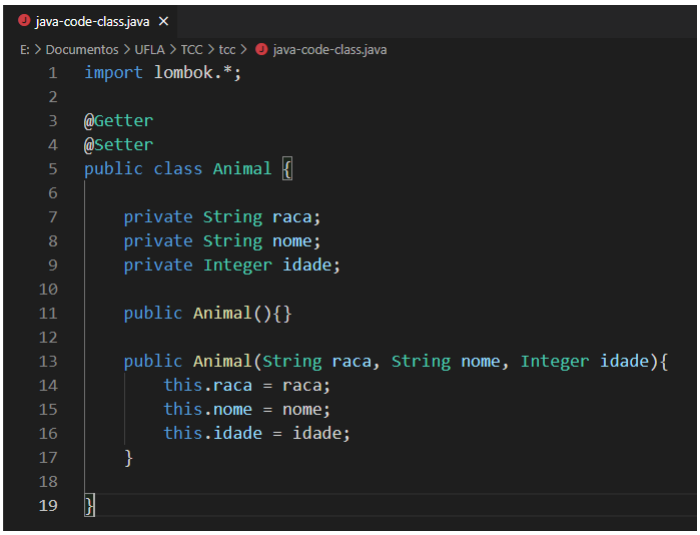
2.2.2.1 Java

Java é uma linguagem de programação orientada a objetos desenvolvido por James Gosling na antiga Sun Microsystem, hoje comprada pela Oracle. A linguagem Java é projetada para propósito geral de programação e para ter uma quantidade mínima de dependências. Um dos objetivos do Java é a portabilidade (ou seja, programas escritos nessa linguagem devem ser executados da mesma

forma em qualquer ambiente), combinação de *hardware* e *software*, com suporte para máquina virtual Java (GOSLING; HOLMES; ARNOLD, 2005).

Apesar da sintaxe do Java ter sido influenciadas pela linguagem C++ e C, o Java foi construído para funcionar exclusivamente coma uma linguagem orientada a objetos, ou seja, todo e qualquer código escrito em Java deve obedecer a estrutura de classes e objetos (GOSLING; HOLMES; ARNOLD, 2005).

Figura 2.18 – Exemplo de código Java



Fonte: Do Autor

O código Java representado na Figura 2.18 demonstra a criação de uma classe. Essa classe possui três atributos, linhas 3 a 7; dois construtores, um construtor vazio na linha 9 e outro construtor com atributos de inicialização dos atributos na linha 13. As anotações *@Getter* e *@Setter* fazem parte da biblioteca *Lombok* para facilitar a produtividade dos desenvolvedores com uso de anotações adicionadas ao código.

2.2.2.2 SpringBoot

O projeto Spring Framework foi publicado em 2002 por Rod Johnson pela editora Wrox. O projeto possuía mais de 30.000 linhas de código em linguagem

Java e inicialmente nomeado de Interface21, apenas futuramente foi renomeado para Spring Framework. Inicialmente, o projeto já continha os componentes básicos para a criação de uma aplicação Web CRUD[2]. Os componentes encontrados nessa versão eram um contêiner para Inversão de Controle[3] com BeanFactory[4], um ApplicationContext[5] e uma Injeção de Dependência[6]. Esses componentes eram os primórdios do SpringMVC (OKI, 2015).

O SpringBoot é um projeto Spring que facilita o processo de criação de uma aplicação Web. O SpringBoot consegue isso facilitando a configuração dos módulos e dependências necessárias no projeto. O módulos e dependências podem ser escolhidos o *start* do projeto criado na página *https://start.spring.io*, essas configurações serão inclusas no pom.xml do projeto. (AFONSO, 2017)

Conforme mencionado anteriormente, a Figura 2.19 representa a página web para inicialização de um projeto *Spring Boot*. Nessa página pode-se configurar as primeiras funcionalidades e características da aplicação. Na área do retângulo vermelho pode-se escolher o tipo de projeto (Maven ou Gradle), a linguagem de programação (Java, Kotlin ou Groovy) e a versão do projeto Spring Boot.

Delimitado por amarelo, tem-se os campos para o desenvolvedor preencher as informações do projeto, bem como escolher a versão do Java e o tipo de pacote que será gerado. O retângulo da cor verde delimita onde poderá adicionar as dependências necessárias para o projeto, como por exemplo, banco de dados

---

[2] "CRUD é o acrônimo de *Create*, *Read*, *Update* e *Delete*. Representa as quatro operações básicas de banco de dados relacional ou interface"(NASCIMENTO, 2018)

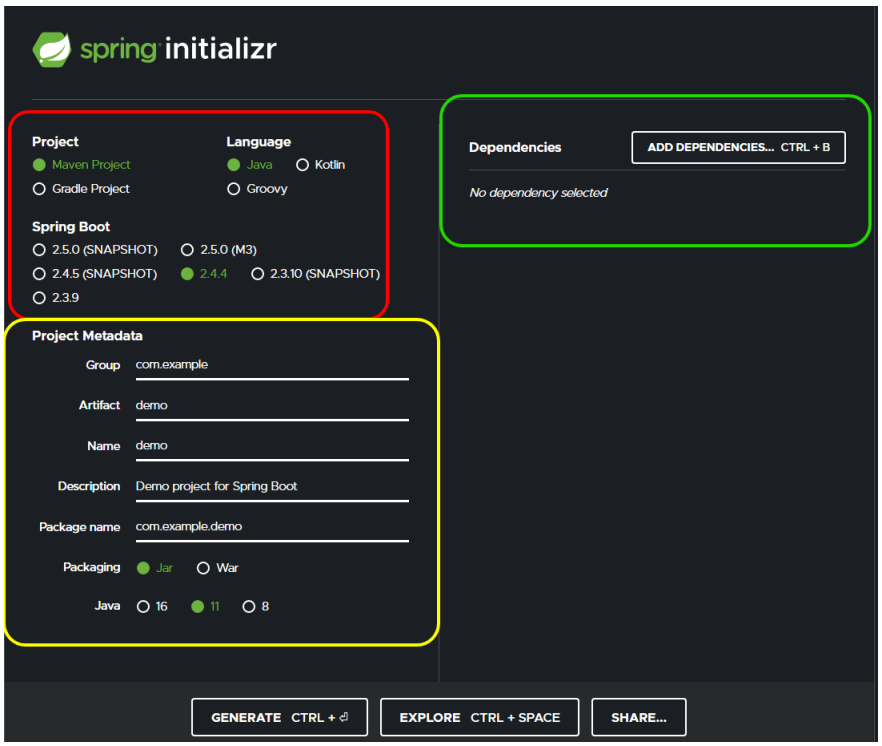
[3] Inversão de Controle é uma forma diferente para manipular o controle sobre um objeto. É uma a mudança de conhecimento que uma classe tem em relação com outra (GAMA, 2010).

[4] Responsável por realizar a inversão de controle (IoC) na injeção de dependências dos objetos instanciados e solicitados por uma aplicação(OKI, 2015)

[5] ApplicationContext é um objeto que carrega as configurações e auxilia o SpringBoot iniciar (OKI, 2015).

[6] Injeção de Dependência são princípios e padrões de *design* para desenvolvimento de software com intuito de desenvolver códigos evitando altos níveis de acoplamento (NIZZOLA, 2020).

Figura 2.19 – Inicializador Springboot



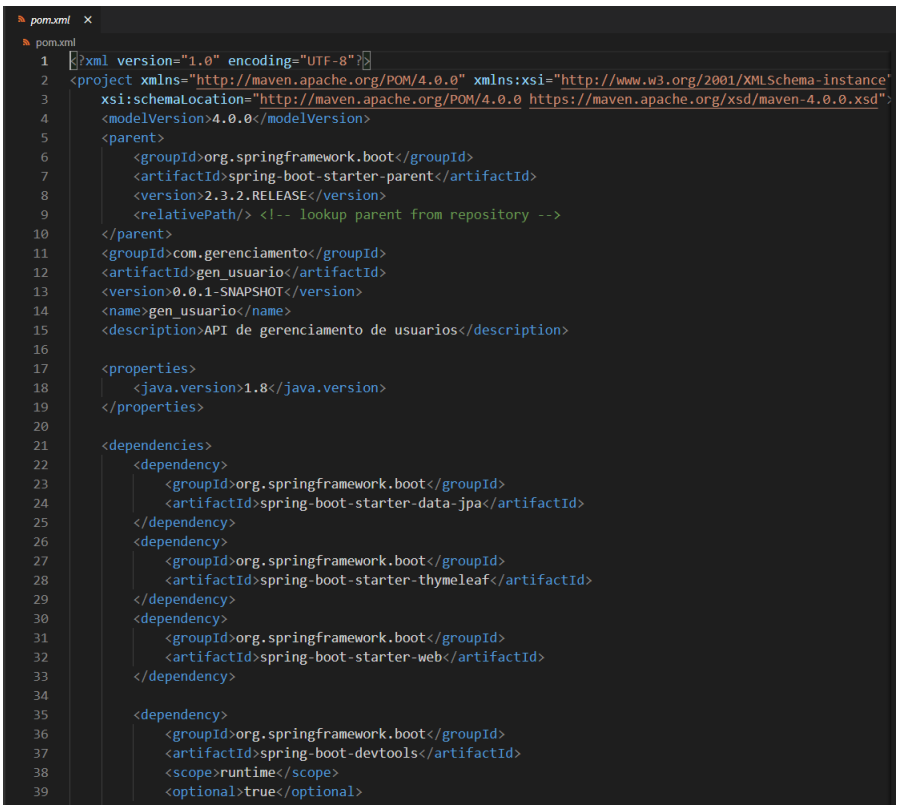
Fonte: start.spring.io, 2021

SQL ou NoSQL, ferramentas de segurança, ferramentas de teste, ferramentas web, ferramentas de desenvolvimento como o *Lombok*, dentre outras.

Ao final do preenchimento do formulário, e selecionadas as dependências o desenvolvedor pode realizar o *download* do pacote ou compartilha-lo em *Generate* e *Share* respectivamente. Dentre os arquivos baixados no pacote, tem o arquivo *pom.xml*. Esse arquivo conterá as dependências selecionadas para o projeto, bem como as demais informações do projeto. A Figura 2.20 exemplifica um trecho do *pom.xml*. Nas linhas de 4 à 19 estão inclusas as configurações do projeto como: nome, domínio, versão, descrição e a versão do java, por exemplo. A partir da linha 22 tem-se algumas das dependências utilizadas na aplicação.

Para realizar as conexões com banco é necessário o uso do Hibernate, implementado pela biblioteca JPA. “O Hibernate é uma ferramenta de mapeamento

Figura 2.20 – Exemplo arquivo pom.xml



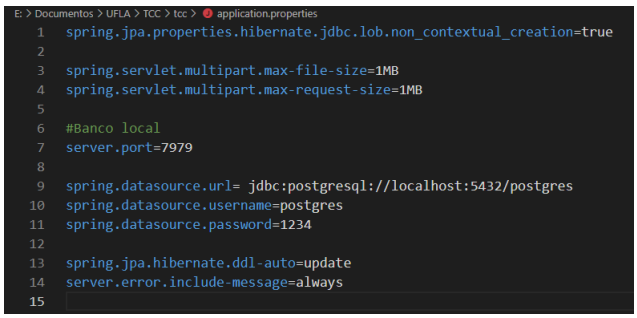
Fonte: Do Autor, 2021

objeto/relacional para Java. Ela transforma os dados tabulares de um banco de dados em um grafo de objetos definido pelo desenvolvedor." (LINHARES, 2016). No SpringBoot, a conexão com o banco de dados é configurada no arquivo *application.properties*. Esse arquivo é responsável por manter as informações referente a conexão com o banco de dados, como a localização na rede e as credenciais de acesso. A Figura 2.21 exemplifica as informações contidas neste arquivo.

2.2.2.3 PostgreSQL

O PostgreSQL nasceu em 1986, a partir do projeto POSTGRES na Universidade Berkeley, na Califórnia. O PostgreSQL é Sistema de Gerenciamento de Banco de Dados - SGBD Relacional, com suporte a operações de Atomicidade,

Figura 2.21 – Exemplo de arquivo application.properties



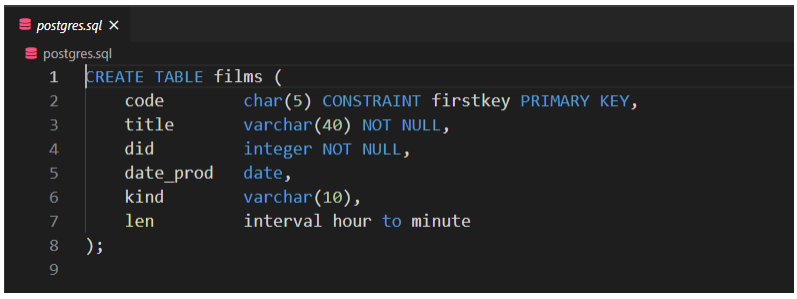
Fonte: Do Autor, 2021

Consistência, Isolamento e Durabilidade, conhecido como ACID. O Postgres fornece meios para realização de replicadas entre servidores de forma gratuita inclusive para fins comerciais, pondendo ser configurando para atuar como um *cluster* (MILANI, 2008).

Com uso de *multithreads* o PostgreSQL consegue gerenciar várias conexões com o banco de dados de uma só vez. Para acessos múltiplos em um mesmo dado, o PostgreSQL força o uso de filas. No intuito de manter a segurança, o PostgreSQL possui suporte nativo à SSL, *Secure Socket Layer*, tanto para gerenciar *login's* como conter informações sigilosas (MILANI, 2008).

Como um banco de dados relacional, o PostgreSQL armazena os dados em tabelas, sendo o única limitação a capacidade de armazenamento do hardware (MILANI, 2008).

Figura 2.22 – Exemplo de código PostgreSQL



Fonte: postgresql.org, 2021

A Figura 2.22 exemplifica a criação de uma tabela em linguagem PostgreSQL. Nesse exemplo, cria-se uma tabela *filmes* para armazenar as dados de diversos filmes. O atributo *code* representa a chave primária da tabela.

2.2.3 Sistema de controle de versão: Git

Um software normalmente é desenvolvidos em equipes, de forma simultânea. Desta forma, vários desenvolvedores podem estar trabalhando em um mesmo código fonte, mas na implementação de diferentes recursos. Dito isso, existem ferramentas de controle de versão que auxiliam as equipes a gerir as alterações feitas no código fonte. Por meio do controle de versão, é possível corrigir erros, enquanto diminui interrupções para os outros membros da equipe (ATLASSIAN, 2020).

O controle de versão ajuda os desenvolvedores a identificar e solucionar problemas de conflitos, gerados em trabalhos em uma mesma linha de código por diferentes desenvolvedores, por exemplo. Além de correção de conflitos, ferramentas de controle de versão ajudam as empresas de desenvolvimento a manter seus códigos-fonte seguros (ATLASSIAN, 2020).

A ferramenta de controle de versão utilizada foi o *Git*, com os códigos-fontes hospedados no *GitLab*. O *Git* utiliza um conceito de ramificações ou *branchs* demarcadas por marcos ou *commits*. Cada *branch* é disposta em uma linha de tempo em que os arquivos podem ser alterados livremente sem que ocasionem impactos nas outras *branchs*. *Commits* ou marcos são as descrições que cada desenvolver atribui para as modificações feitas em uma *branch* (BERTOLA, 2019).

O *GitHub* e *GitLab* são plataformas de hospedagem de código-fonte. Essas plataformas permitem que os desenvolvedores contribuam em projetos privados ou *open source*[7]. Cada projeto com código fonte são considerados repositórios. Contudo, a principal diferença entre essas duas plataformas é que o *GitLab*

[7] Projetos *Open Souce* são conhecidos como projetos que possuem licença de código aberto.

ferramentas CI/CD[8], métricas de acompanhamento de qualidade de código, performance e teste de usabilidade(BERTOLA, 2019).

Figura 2.23 – Exemplo de branch no GitLab

Fonte: zup.com.br, 2021

A Figura 2.23 ilustra quatro *branchs* de um software em desenvolvimento. Primeiro temos a *branch master* na versão *v0.1*, essa é dividida na *branch develop*. A *branch develop* em seu estado inicial gera uma *branch feature*, depois de uma motificação ou *commit* gera uma nova *branch feature*, ao mesmo tempo em que a *master* cria a versão *v0.2* e a primeira *feature* criada realiza sua primeira modificação. As *features* realizam mais dois *commits* cada uma, porém somente uma delas concatenam (*merge*) com a *develop*. As *branch's master* e *develop* seguem em desenvolvimento paralelo para a versão *v1.0*.

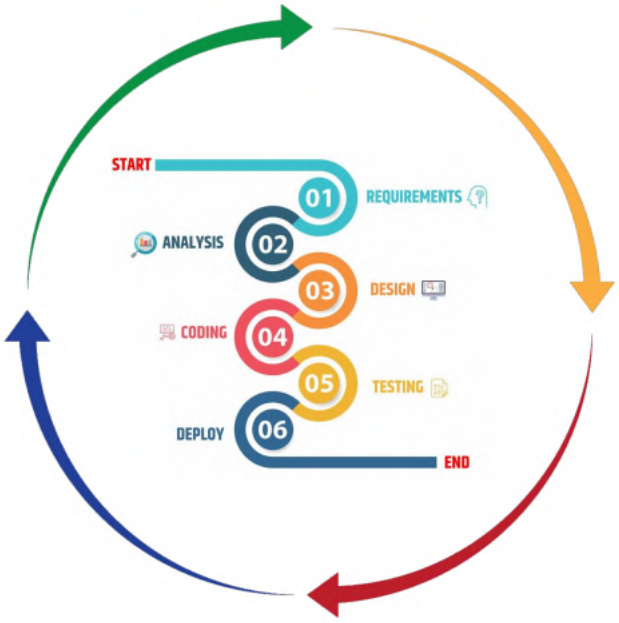
2.2.4 Processo de desenvolvimento de software Zetta

Segundo Pietro (2020) O processo de desenvolvimento de software da Zetta descreve que o fluxo de trabalho é uma sequência lógica de possíveis ações a serem tomadas durante todo o ciclo de vida do processo de desenvolvimento de software. O objetivo é ajudar as equipes a trabalhar de maneira coesa e eficaz,

[8] CI/CD, respectivamente *Continuous Integration* e *Continuous Delivery* ou em português Integração Contínua e Entrega Contínua são dois processos de inovação no desenvolvimento e entrega de projetos. Ambos são baseados em fórmulas modernas que permitem que ambas etapas sejam realizadas de forma mais rápida (FREITAS, 2020).

desde o primeiro estágio da implementação de algo novo (ideação) até a implantação do sistema em produção. A Figura 2.24 demonstra os principais ciclos durante o desenvolvimento de um projeto na Zetta.

Figura 2.24 – Ciclo de desenvolvimento de software



Fonte: Agência Zetta, 2021

A seguir será feita uma breve descrição sobre as etapas de desenvolvimento de software da ZETTA.

2.2.4.1 Requisitos (*Requirements*)

A etapa de requisitos é dividida em dois principais passos:

2.2.4.1.1 Ideia

Toda nova proposta começa com uma ideia, que geralmente aparece em um bate-papo, reunião com cliente. Nesta fase, a ZETTA irá armazenar todo esse

levantamento em um repositório para que possa ser acessível para todos os envolvidos (PIETRO, 2020).

2.2.4.1.2 Issues

Quando a ideia estiver madura e acordada entre todos, chega a hora de criar uma *Issue* no seu *Backlog*. Eessa *Issue* pode ser um problema no sistema, uma nova *feature* ou uma melhoria. Usa-se *Labels* para qualificar as *Issues*. O importante é que tudo esteja documentado e inserido em algum lugar de controle, para que no futuro tenha-se o histórico na mão. Nessa fase a ZETTA irá inserir as informações no *ISSUE Board* do GitLab de acordo com os *Labels* definidos. Todos os tipos de *Issues* tem um *template* a ser seguido e estão na raiz desse projeto (PIETRO, 2020).

2.2.4.2 Análise (*Analysis*) e Design

A etapa de Análise/Design é dividida nos cinco passos descritos a seguir.

2.2.4.2.1 Planejamento

Agora com as *Issues* armazenadas, categorizadas e agrupadas o momento é de organizar e prioriza-las no Backlog e escolher quais serão dispostas no Kan-Ban/Sprint. Nessa fase, a ZETTA irá priorizar essas informações no ISSUE Board do GitLab de acordo com seus *Labels* e prioridades(Da mais prioritária para a menos) (PIETRO, 2020).

2.2.4.2.2 *Design*

Depois realizado o planejamento, organizado e priorizado as *Issues*, chega o momento do Design. Nessa fase, o projeto se tornará mais visual, seja por meio de telas/protótipos ou arquitetura de sistemas. O Analista de Negócios da Zetta

juntamente com o Designer alinharão para as questões de telas/protótipos ou o Líder Técnico para tratar das questões técnicas (PIETRO, 2020).

2.2.4.2.3 Montando os ciclos de entrega

Depois do Planejamento e Design, serão montados os ciclos de entrega de valor para os clientes. Nesses ciclos será escolhido um grupo de atividades que farão parte da entrega, mas para isso juntamente com a equipe "são pesadas" as atividades. Pode-se usar para esse peso tanto a unidade de horas, como esforço, ou valor agregado, isso fica a cargo da equipe, sem esquecer de verificar o quanto cabe no ciclo de entregas (PIETRO, 2020).

Nessa fase, a ZETTA fará uso das *Milestones* para os Intervalos Regulares de entregas (KanBan/Sprint). As Milestones deverão conter em sua descrição a quantidade de pessoas envolvidas nas estimativas. As *Issues* podem ser agrupadas para formar um grupo de atividades(Histórias de Usuário) e serão utilizados os *Epics* para isso. Todas as *Issues* devem conter o tempo estimado("*/estimate*") e o tempo gasto("*/spend*") (PIETRO, 2020).

A Zetta utiliza um sistema de ponderação de *Issues* para ajudar no planejamento da capacidade de atuação da equipe. Esses pesos são usados para o planejamento da capacidade e o foco principal é garantir que a soma geral dos pesos seja razoável. Não há problema caso uma *Issue* demore mais do que o peso indica. Os pesos devem ser usados de forma agregada, e o que uma pessoa leva um dia, outra pessoa pode levar uma semana, dependendo do seu nível de conhecimento prévio sobre o problema. Isso é esperado (PIETRO, 2020).

2.2.4.2.4 Estimativa Contínua

Segundo Pietro (2020), para atribuir pesos às *Issues* de uma *Milestone* futura, os membros da equipe devem ponderar continuamente, e detalhar as *Issues* que estão no *Workflow:Planning*. Desta forma, os membros da equipe sempre

devem olhar e revisar as *Issues* que estão em Planejamento. Nas discussões das *Issues* deve-se tentar dar peso e detalhar as atividades exatamente como tratada uma "*Planning*"no SCRUM. Essa atividade pode ocorrer de forma Assíncrona e depois que se chega a um acordo quanto ao entendimento e "peso"da *Issue*, cabe ao GP(Gerente de projetos)/PO(Product Owner) inseri-la em uma *Milestone* futura.

Figura 2.25 – *Planning TimeBox*

Item	Tempo de duração	Ação
Passagem por todas as Issues que entrarão na Milestone	20 minutos	O PO revisará todas Issues, as quais as regras já estão definidas somente incluindo(se ainda não houver) o tempo estimado.
Selecionar quais Issues entrarão como Deliverable na Milestone	10 minutos	O PO marcará/confirmará com a label Deliverable as Issues
Repasse geral das Issues e ato de compromisso da equipe quanto aos entregáveis	5 minutos	PO repassa todos os itens e objetivos e o ato de compromisso da Milestone.

Fonte: Agência Zetta, 2021

A Figura 2.25 apresenta um quadro com os itens a ser discutido durante a *Planning*. Cada item possui uma duração aproximada e qual assunto tratado durante aquele tópico.

2.2.4.2.5 Retrospectivas

Segundo Pietro (2020), no último dia útil do mês ou quando o PO julgar necessário, será realizada a reunião retrospectiva. Para a reunião de retrospectiva será criado uma *Issue* com o label: ***retrospective*** pelo PO seguindo o *template* na Figura 2.26. Essa figura ilustra um modelo a ser usado como base para listar e avaliar os tópicos discutidos durante a retrospectiva.

Ainda segundo Pietro (2020), toda discussão será anotada e preenchida na *Issue*, essa reunião ocorrerá na ferramenta de transmissão ao vivo, Google Meet

Figura 2.26 – *Planning TimeBox*

Retrospectiva PO

Retrospectiva do Mês: `MÊS/ANO`

Por favor, olhe para as suas experiências de trabalho nesta release. Pergunte-se: 👍 O que correu bem neste mês?, 👎 O que não correu bem neste mês? E 📈 o que podemos melhorar?

Se houver algo que você não se sinta confortável compartilhando aqui, envie uma mensagem para seu PO/GP.

> Marquem como confidencial as Issues de `retrospective` para que os participantes possam comentar sem que os outros vejam

Issues entregues

- Enumere as Issues Entregues no mês

Issues não entregues

- Enumere as Issues NÃO entregues no prazo naquele mês

Totais

Total de Deliverable entregues	Total de issues fechadas	Total de MRs realizados
XX	YY	ZZ

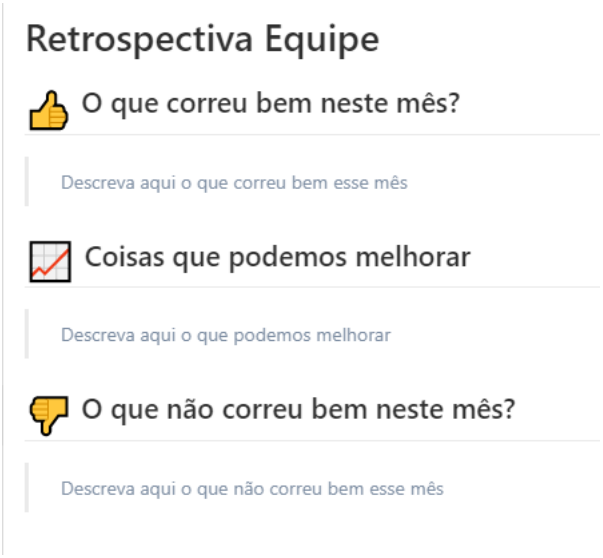
Fonte: Agência Zetta, 2021

por exemplo. Será discutido o que correu bem, o que deu errado e o que se pode melhorar relativo ao período da retrospectiva. Os participantes das equipes deverão preencher comentários na *Issue* seguindo o *template* na Figura 2.27.

2.2.4.3 Codificando (*Coding*) /Testando (*Testing*) /Entregando (*Deploy*)

A etapa de Codificando/Testando/Entregando é dividida nos oito passos descritos a seguir.

Figura 2.27 – Quadro de retrospectiva da equipe



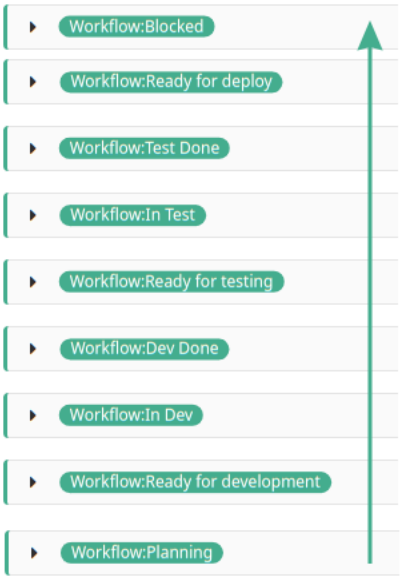
Fonte: Agência Zetta, 2021

2.2.4.3.1 *Code*

Segundo Pietro (2020), para o passo de Codificação ou *code*, inicia-se a escrita do código, uma vez que tudo esteja organizado conforme as etapas anteriores. A Figura 2.28 ilustra o fluxo de desenvolvimento que seguirá a ordem: *workflow: Planning* → *Ready for development* → *In Dev* → *Dev Done* → *Ready for testing* → *In Test* → *Test done* → *Ready for deploy*. A as *Issues* que por alguma razão se encontrem impossibilitadas de dar prosseguimento em alguma das fases devem ser colocadas no *workflow* como *Bloqued*;

Toda *issue* de desenvolvimento deverá ser uma *branch* no repositório de códigos e deverá ser nomeada da seguinte maneira **#ISSUE_TEXTO**, por exemplo *Issue:* #1 - Cadastro de usuário. As Issues que forem ligadas a Ordens de Serviço deverão apresentar nelas o número da OS dentro de uma *Tag* HTML <*cite*> (PIETRO, 2020).

Figura 2.28 – Fluxo de codificação



Fonte: Agência Zetta, 2021

2.2.4.3.2 *Merge Request*

Segundo Pietro (2020), todas as *issues* de desenvolvimento devem passar por *Merge Request* que terá dois atores principais(*Reviewer* e o *Maintainer*), em que o *Reviewer* é um membro aleatório da equipe, podendo ser qualquer um, aqui acontecerá a disseminação de conhecimento no projeto, e o *Maintainer* é o responsável por validar e garantir padrão e qualidade de código gerado antes de aceitar o *Merge Request*.

2.2.4.3.3 *Tests*

Segundo Pietro (2020), os desenvolvedores deverão escrever Testes Unitários e de Integração, quando aplicável, inclusive no *Front-End*. Esses testes entrarão no *Pipeline* de Integração contínua. Desta forma, no repositório ou no ambiente de produção, são executados testes de construção e automatizados que

garantem que não haja problemas de integração e identificação precoce de problemas.

2.2.4.3.4 *Stagging*

Segundo Pietro (2020), nesta etapa é implantado o código em um ambiente de teste para verificar se tudo funcionou como se esperava ou se ainda precisa de ajustes. Para isso, usa-se o recurso do Pipeline de CD(*Continuous Delivery*), o arquivo *.gitlab-ci.yml* deverá ser criado para os projetos e nele implementado a integração com o ambiente de testes. O ambiente de testes deverá ter o Sentry[9] habilitado e integrado ao GitLab.

2.2.4.3.5 *Homologation*

Se o projeto possui um ambiente onde o cliente pode realizar a homologação das entregas também tem-se um CD para o ambiente de homologação. O ambiente de homologação também deverá ter o *Sentry* habilitado e integrado ao GitLab (PIETRO, 2020).

2.2.4.3.6 *Production*

Após realizados os testes nos ambientes de teste e homologação, a próxima etapa é implantar o projeto no ambiente de produção. Para isso o CD do projeto deve contemplar o *deploy* em produção. A ideia aqui é que esses atos de entrega do sistema sejam todos automáticos e não necessitem de intervenção e não exista tempo gasto nesses ritos do sistema (PIETRO, 2020).

2.2.4.3.7 *QA - Quality Analyst*

Afim de prezar por padrão, boas práticas, códigos limpos, que facilitam a manutenção, evolução e correção dos sistemas, o *Maintainer* juntamente com

[9] https://sentry.io/about/

ferramentas de *code analysis* e *Linters* farão uma dupla. Os merges do sistema só serão aceitos se os dois atores estiverem de acordo, a ferramenta de *Analysis/Linter* com as suas checagens e o *Maintainer* com boas práticas de um *merge request* (PIETRO, 2020).

2.2.4.3.8 Comunicação

Para manter a efetividade é preciso que além das conversas entre os membros da equipe, devem ser escritas as decisões, manter histórico das conversas para que as mesmas não sejam perdidas. Hoje a Zetta utiliza o *Rocket Chat*, que se trata do canal de comunicação oficial da ZETTA além das próprias *Issues* que podem ser usadas para discussões (PIETRO, 2020).

Neste capítulo, foram apresentadas as principais tecnologias utilizadas durante o desenvolvimento de três projetos nos quais o estagiário atuou. Dentre elas, pode ser destacada a metodologia *Scrum* como a principal abordada pelas equipes de desenvolvimento. Para o *front-end* tem maior destaque no*Vue*, tecnologia presente nos projetos mais novos da Zetta acompanhado pelo *CSS* e *JavaScript*. SpringBoot no *back-end* enfatizando a linguagem de programação Java e finalizando com o Postgres, o principal sistema gerenciador de banco de dados utilizado. Os projetos em geral são armazenados e versionados pelo GitLab.

3 ATIVIDADES DESENVOLVIDAS

Neste capítulo, serão descritas as atividades desenvolvidas durante o estágio, lições aprendidas e dificuldades encontradas. As atividades são divididas em três projetos, sendo eles o sistema de Gestão de *Sprints*, o Hotsite PronaSolos e a Plataforma Geoespacial do Observatório da Agropecuária Brasileira.

O Sistema de Gestão de Sprints foi um projeto desenvolvido durante o treinamento, desta forma, foi possível detalhar como funciona cada componente bem como as regras de negócio. Entretanto, os projetos Hotsite PronaSolos e a Plataforma Geoespacial do Observatório da Agropecuária Brasileira por serem projetos oficiais da Agência Zetta, as atividades desenvolvidas pelo estagiário foram descritas tomando devido cuidado para não infligir as regras de confidencialidade.

3.1 Imersão Z e o Sistema de Gestão de Sprints

O Sistema de Gestão de Sprints foi a segunda etapa da fase de treinamento e integração do estagiário na Agência Zetta. A primeira fase do treinamento consistiu em uma série de *workshops* juntamente com time de inovação da Zetta, denominado **Imersão Z**. A Imersão Z foi realizada com auxilio da plataforma Miro[1] e tinha a finalidade de apresentar ao estagiário um pouco da história e funcionamento da Zetta.

Durante os *workshops*, o estagiário pode conversar e interagir com colaboradores mais experientes, aprender sobre as três tribos, ARA, IG e YBY, que junto ao time de ciência de dados compõem a agência, ter conhecimento de alguns dos grandes projetos desenvolvidos e aprender um pouco sobre a estrutura do desenvolvimento de software. Na última fase da Imersão Z, foi apresentado ao estagiário qual seria seu *squad* de treinamento e uma proposta de desenvolvimento. Essa proposta consistia na elaboração e implementação de um módulo ERP da Agência Zetta, o **Sistema de Gestão de Sprints**.

[1] https://miro.com/about/

3.1.1 Sistema de Gestão de Sprints

O Sistema de Gestão de Sprints[2] foi descrito como sendo a implementação de um módulo para controle gerencial e financeiro das *sprints* executadas na operação, de modo a eliminar a utilização de planilhas de controle, centralizar as informações em um único local e facilitar o compartilhamento e acesso aos dados. O *squad* desenvolvimento foi composto por três desenvolvedores, um PO (*Product Owner*) e um *Scrum Master*. O *Scrum Master* era um colaborador mais experiente e os demais membros da equipe eram novos estagiários e colaboradores.

Para adaptar os novos colaboradores e estagiários às rotinas e ritos do Scrum, o Scrum Master sugeriu que fosse seguido esse *framework* no decorrer do projeto. Desta forma, alguns dos ritos do Scrum foram utilizados durante a execução das atividade, como divisão das tarefas em um *sprint*, reuniões diárias e estimativas de tempo.

A primeira parte do desenvolvimento do sistema foi uma série de cursos preparatórios na plataforma TreinaWeb[3]. Os cursos abrangiam diversas tecnologias e *frameworks* de desenvolvimento, desde cursos de Scrum até cursos de desenvolvimento prático como *SpringBoot* para *back-end* e *VueJs* para *front-end*. Após a conclusão dos cursos sugeridos, foi iniciada a segunda parte: o desenvolvimento do Sistema de Gestão de Sprints.

Foi repassado para o *squad* um documento contendo as regras de negócio e as histórias de usuário de forma mais detalhada sobre o que era necessário conter na entrega final. Com as regras de negócio descritas, o PO do *squad* criou um protótipo do sistema. Com os integrantes do *squad* com as ideias alinhadas, atividades foram geradas a partir das regras e análise das telas do protótipo. Essas atividades foram dispostas na planilha de atividades, e dessa forma eram feitos os controles do que já havia sido feito e do que ainda era necessário.

[2] https://gestao-sprints-zetta.herokuapp.com
[3] https://www.treinaweb.com.br

3.1.1.1 Tecnologias utilizadas

Para seguir os cursos feitos durante a etapa de treinamento, juntamente com as orientações do *ScrumMaster*, o *squad* optou pelas seguintes escolhas de tecnologias para o desenvolvimento da aplicação:

- SGBD Postgres para o banco de dados;
- Spring Boot para o *back-end*;
- Vue.js e Vuetify para o *front-end*.

3.1.1.2 Modelagem do banco de dados e criação do *back-end*

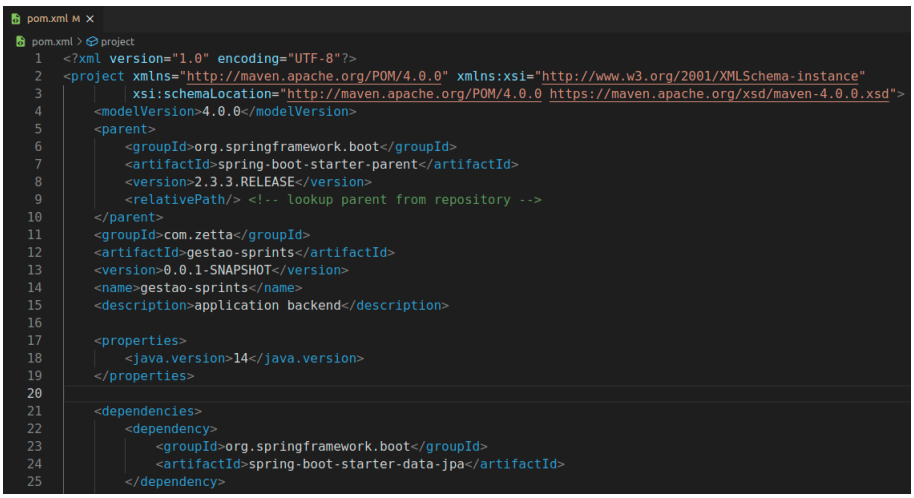
O estagiário juntamente com os outros desenvolvedores do *squad* iniciaram a análise das regras de negócio para decidir quais seriam as primeiras tabelas a existir no banco de dados. Com uso dos conceitos visto na disciplina de banco de dados, foi modelado um banco via diagramas de entidade-relacionamento (ER)[4]. Após a diagramação do banco de dados, foi decidido que o sistema inicialmente conteria as tabelas *Sprint*, *Projeto* e *Ordem de serviço*. O relacionamento entre as tabelas seria que cada projeto poderia conter uma ou mais *sprints* e cada *sprint* poderia abordar uma ou mais *ordens de serviços* relacionadas ao projeto.

Com o banco de dados estruturado iniciou-se a criação do *back-end*. O *back-end* foi criado utilizando o *framework SpringBoot*. Para as configurações iniciais do sistema foi utilizado o Spring Initializer. A Figura 3.1 mostra o arquivo *pom.xml* do sistema contendo suas configurações, como nome, grupo descrição e versão do Java utilizado. A partir da linha 21 estão listadas as dependências necessárias para o funcionamento do sistema.

A Figura 3.2 mostra as configurações no arquivo *application.properties* do sistema. Neste arquivo estão as configurações de conexão com o banco de dados,

[4] Um diagrama entidade relacionamento (ER) é um tipo de fluxograma que ilustra como "entidades", p. ex., pessoas, objetos ou conceitos, relacionam-se dentro de um sistema (LUCIDCHART, 2020).

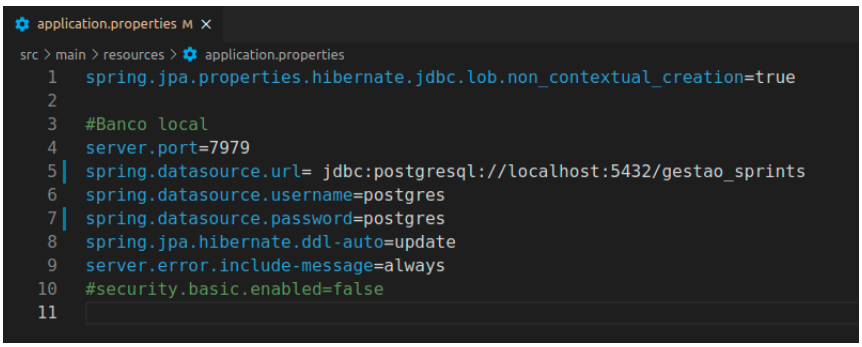
Figura 3.1 – pom.xml do Sistema Gestão de Sprints



Fonte: Do Autor, 2021

como o *host*, o nome do banco, as credenciais de acesso e a porta na qual estarão disponíveis os serviços fornecidos pelo *back-end*.

Figura 3.2 – application.properties do Sistema Gestão de Sprints

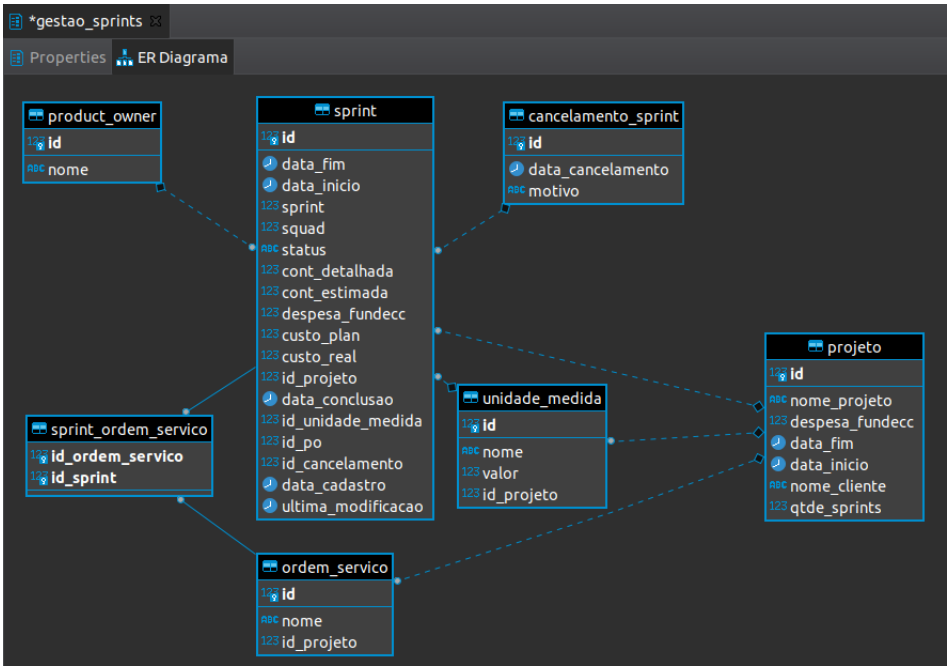


Fonte: Do Autor, 2021

À medida em que o sistema foi evoluindo, foram necessárias a expansão do banco de dados com adição de novas tabelas e relacionamentos. As novas tabelas adicionadas foram: *Product Owner*, responsável por armazenar os PO's; a tabela de *Unidade Medida*; responsável por conter as informações relacionadas as unidades de medida, como valor e *id* do projeto e por último, a tabela de *Cancelamento Sprint*, responsável por armazenar a lista de *sprint's* canceladas. A Figura

3.3 mostra o diagrama final gerado pelas tabelas e relacionamentos necessários para criação do sistema.

Figura 3.3 – Diagrama entidade-relacionamento final do Sistema Gestão de Sprints



Fonte: Do Autor, 2021

A Figura 3.4 mostra como foram organizados os arquivos que compõe o *back-end*. Os arquivos foram distribuídos nos seguintes pacotes de acordo com a função que cada um exerce:

1. *configurações*: possui pacotes de arquivos de necessários para configurações de componentes do sistema, como *deserializadores* de data por exemplo;

2. *controllers*: contêm os arquivos com função de receber e atender as requisições feitas a partir do *front-end*;

3. *enuns*: possui arquivos que enumeram *status* ou atributos de entidades;

4. *exceptions*: estão armazenados neste pacotes os arquivos de tratamento de exceção;

5. *interfaces*: possui as interfaces que controlam os métodos as serem implementados pelos serviços (*services*);

6. *jobs*: ficam os arquivos responsáveis por realizador *trabalhos* extras para as entidades, como verificação ou atualização de status, por exemplo;

7. *models*: os arquivos contidos neste pacote possuem a função de modelar os objetos da aplicação;

8. *repositories*: estão os arquivos responsáveis pela comunicação do sistema Web com o banco de dados;

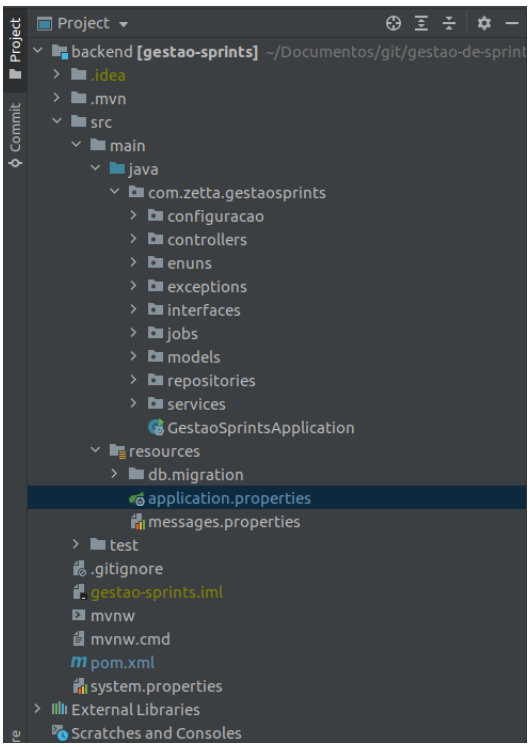
9. *services*: estão as classes responsáveis por implementar os as interfaces. Essas classes implementam os métodos que são utilizados pelas *controllers* de acordo com a requisição realizada pelo *front-end*.

3.1.1.3 Implementação do *front-end*

A implementação do *front-end* foi onde o autor teve maior participação. A etapa inicial da criação do *front-end* foi a criação de um projeto *Vue.js* com a estrutura do *Vuetify* acoplada. Para a criação de um projeto *Vue.js* utiliza-se o comando `vue create <nome-do-projeto>` no terminal ou utilizando o comando `vue ui` para criação com uso de interface gráfica, conforme Figura 3.5.

Após criado o projeto *vue*, foi adicionado o *framework Vuetify*. Para adicionar o *Vuetify*, a primeira etapa é a instanciá-lo dentro do projeto *vue* com o uso do comando `vue add vuetify`. A segunda etapa é a criação do diretório `src/plugins/vuetify.js` com configurações conforme exibido na Figura 3.6. Com as configurações iniciais realizadas, pôde dar inicio a estruturação das telas do sistema.

Figura 3.4 – Estrutura *back-end* final do Sistema Gestão de Sprints



Fonte: Do Autor, 2021

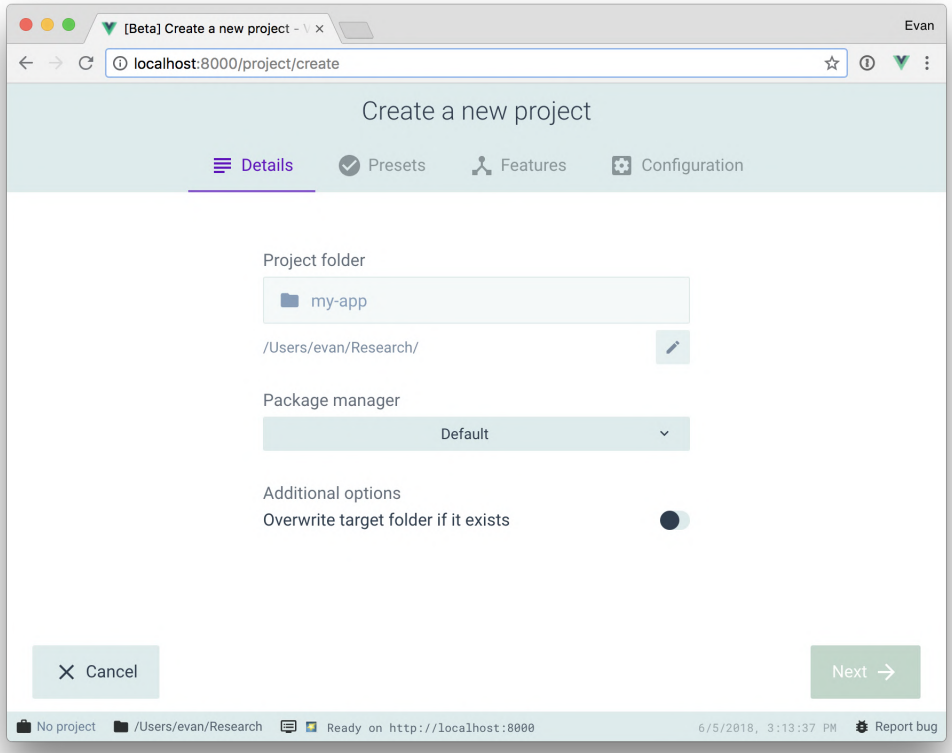
3.1.1.3.1 A primeira página do sistema

A primeira tela criada do sistema era uma tabela que listava *sprints* com dados mocados[5] contendo alguns dos atributos listados na tabela *sprints* do *back-end*. O cabeçalho é um componente de *header* simples do *Vuetify* com um ícone de menu adicionado a esquerda, o nome do sistema e um ícone de usuário a direita, vide Figura 3.7.

O menu lateral é um componente retrátil. Este menu ao ser clicado escurece o restante da tela, mantendo o foco do usuário nele. Este menu foi projetado para facilitar a navegação entre a página listagem de projetos e a página de lista-

[5] Dados mocados são dados genéricos utilizados apenas para visualização e exemplificação do funcionamentos dos componentes.

Figura 3.5 – Interface gráfica de criação de projeto Vue



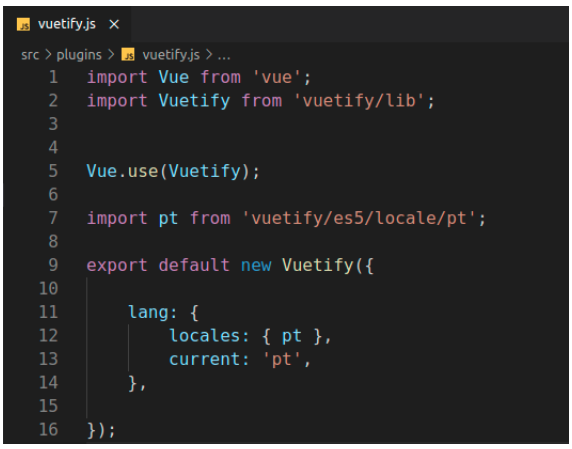
Fonte: cli.vuejs.org, 2021

gem de todas as *sprints* de responsabilidade do usuário. A Figura 3.8 exibe o menu expandido e parte do restante da tela esmaecida.

O rodapé é um componente fixo na parte debaixo da página. Ele é composto por duas divisões, a primeira com os ícones que redirecionam para a rede social equivalente e a segunda com o ano e o número do *squad* de treinamento, vide Figura 3.9.

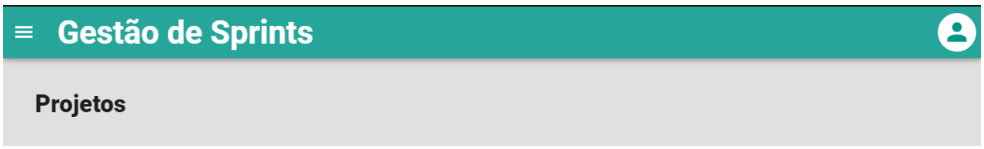
A Figura 3.10 ilustra a versão final da tela que conterá a lista de projetos, o menu lateral expandido, o cabeçalho e o rodapé do sistema.

Figura 3.6 – Arquivo vuetify.js



Fonte: Do Autor, 2021

Figura 3.7 – Cabeçalho do Sistema Gestão de Sprints



Fonte: Do Autor, 2021

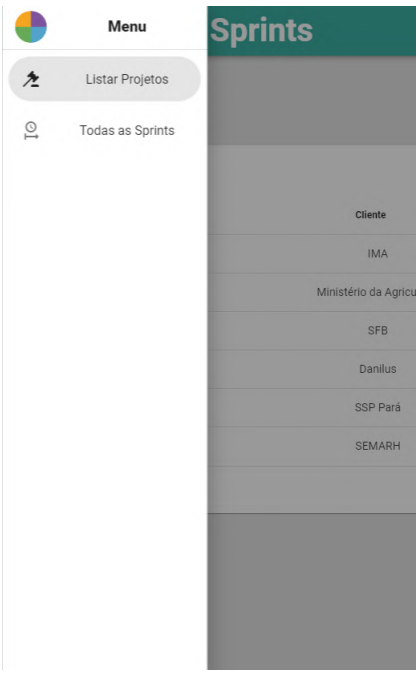
3.1.1.3.2 Implementação da página de listagem de *sprints* por projeto

Essa página lista somente as *sprints* relacionadas com o projeto selecionado na página de projetos. Nesta página, é possível a criação, edição, cancelamento e conclusão de *sprints*. Essa página exibe o status de cada *sprint* bem como valores e data de inicio e fim por exemplo. A Figura 3.11 demonstra a página final do *Sprints* por projeto do sistema.

Para a listagem das *sprints* foi utilizado um serviço fornecido pelo *back-end* que lista as *sprints* de acordo com o *id* do projeto selecionado. A Figura 3.12 mostra o método que realiza a listagem. Nas linhas 416 e 421 do método estão *toast*[6] que fornecem um *feedback* do serviço prestado pelo método.

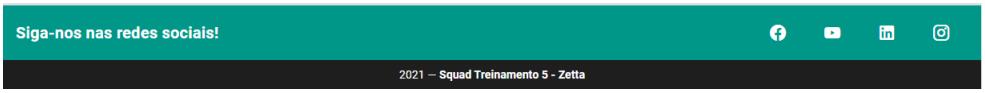
[6] Toast são pequenas *pop-up's* que exibem notificações simples.

Figura 3.8 – Menu do Sistema Gestão de Sprints



Fonte: Do Autor, 2021

Figura 3.9 – Rodapé do Sistema Gestão de Sprints



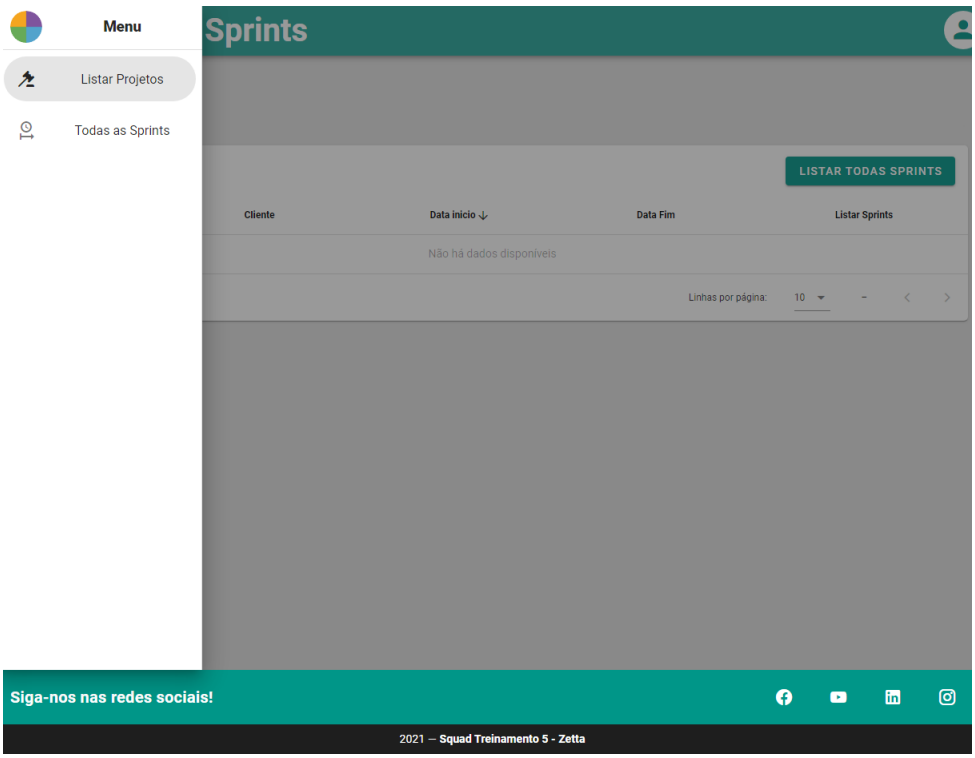
Fonte: Do Autor, 2021

A conclusão e cancelamento de *sprints* também são serviços fornecidos pelo *back-end*. A Figura 3.13 ilustra os dois métodos citados. O serviço de cancelar espera que o *front-end* forneça tanto o *id* da *sprint* como uma justificativa de cancelamento, enquanto o método de conclusão espera somente o *id*. A visualização, criação e edição de *sprints* são realizadas em outras páginas.

3.1.1.3.3 Implementação da página de criação e edição de *sprints*

A página de criação de *sprints* é um formulário com duas abas, a aba **Planejado** e a aba **Realizado**. A aba **Planejado** é composta por três painéis expan-

Figura 3.10 – Tela de listagem de projetos do Sistema Gestão de Sprints

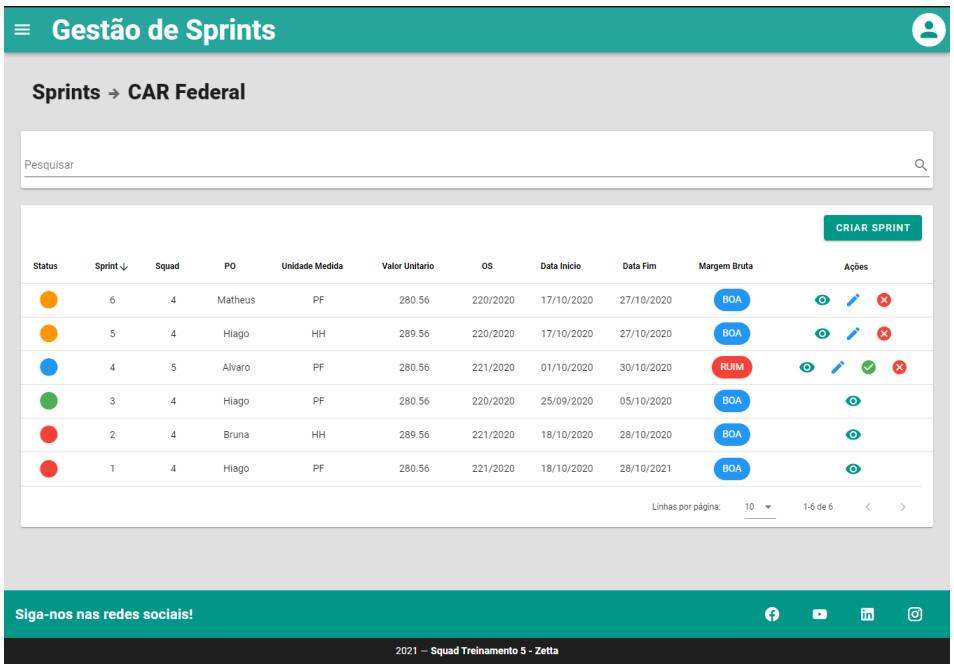


Fonte: Do Autor, 2021

síveis, cada painel possui os campos para preenchimento, alguns destes campos são regras que utilizam fórmulas apresentadas nas regras de negócio e valores dos campos preenchidos anteriormente. A Figura 3.14 mostra a aba **Planejado** com o painel de **Informações Planejadas da Sprint** expandido. Os campos de *Product Owner*, Unidade de Medida e Ordem de Serviço são listas fornecidas pelo *back-end* por meio de serviços.

Ao preencher todos os campos do formulários, atendendo os requisitos listado na regra de negócio, é possível realizar o cadastro da *sprint* clicando no botão **Enviar**. O botão **Enviar** aciona o método `submit`. Conforme exibido na Figura 3.15, o método verifica se o campos do formulário estão devidamente pre-

Figura 3.11 – Tela de listagem de *sprints* por projeto do Sistema Gestão de Sprints



Fonte: Do Autor, 2021

Figura 3.12 – Método que chama o serviço de listagem de *sprints*

```
411    listar(){
412        ProjetoServices.getProjetosSprint(this.projetoId)
413                    .then( resposta => {
414                        this.sprints = resposta.data
415                        if (this.sprints.length == 0) {
416                            this.$toast.info('Projeto ' + this.projeto.nomeProjeto + ' não possui sprints cadastradas.')
417                        }
418                    })
419                    .catch(error => {
420                        console.log(error)
421                        this.$toast.error('Projeto ' + this.projeto.nomeProjeto + ' não possui sprints cadastradas!')
422            })
423    },
```

Fonte: Agência Zetta, 2021

enchidos com informações válidas, em caso positivo, será enviado ao *back-end* as informações da *sprint* em formato JSON[7] para persistência no banco de dados.

A página de edição de *sprint* segue a mesma estrutura da página de criação. Contudo, a página de edição carrega e preenche o formulário de acordo com as informações provindas da *sprint* selecionada. Na edição, também é possível

[7] JSON é acrônimo de *JavaScript Object Notation*, e em suma é um formato compacto, de padrão aberto independente, de troca de dados simples e rápida entre sistemas (FERNANDES, 2020).

Figura 3.13 – Métodos que chama os serviços de conclusão e cancelamento de *sprints*

```
440         cancelarSprint (id) {
441
442             SprintServices.cancelar(id, this.justificativaCancelamento)
443                 .then(() => {
444                     this.justificativaCancelamento.motivo = ''
445                     this.$toast.success('Sprint cancelada com sucesso!')
446                     this.atualizar()
447                 })
448                 .catch(error => {
449                     this.$toast.error("Erro: " + error.response.data.message)
450                     this.atualizar()
451                 })
452
453         },
454
455         concluirSprint (id) {
456
457             SprintServices.concluir(id)
458                 .then(() => {
459                     this.$toast.success('Sprint concluída com sucesso!')
460                     this.atualizar()
461                 })
462                 .catch(error => {
463                     this.$toast.error("Erro: " + error.response.data.message)
464                     this.atualizar()
465                 })
466         },
```

Fonte: Agência Zetta, 2021

visualizar o status e a data de cadastro da *sprint*. A Figura 3.16 ilustra uma edição de *sprints*.

3.1.1.3.4 Estrutura do *front-end*

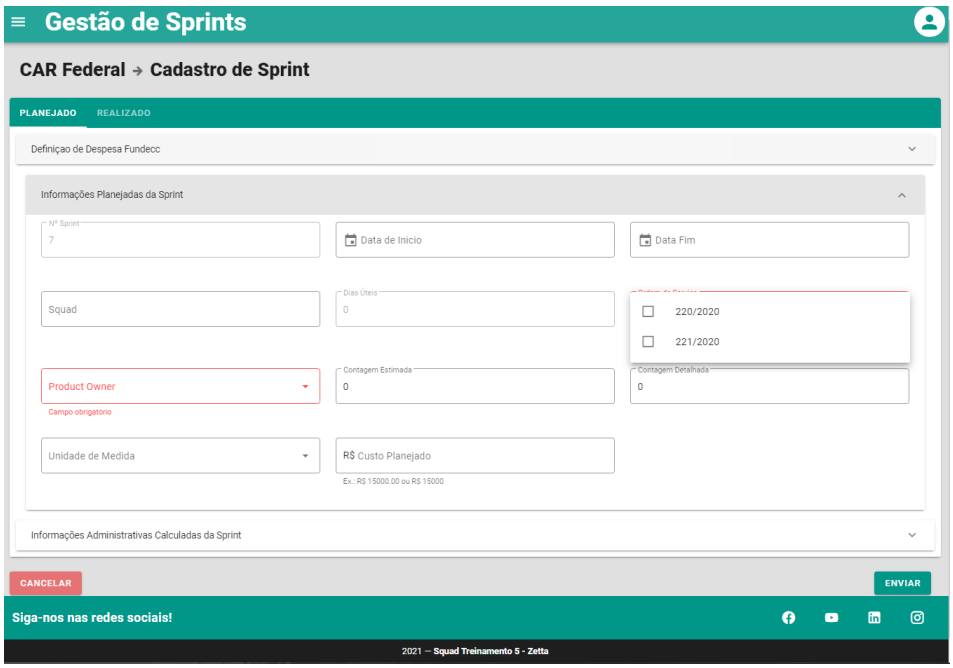
A estrutura do *front-end* foi planejada de maneira a organizar as arquivos em pacotes conforme sua funcionalidade. A Figura 3.17 ilustra essa organização do código-fonte. A seguir será descrita a funcionalidade de cada uma das pastas.

1. *assets*: contém os logos e imagens;

2. *components*: possui componentes que são frequentemente utilizados, como o cabeçalho e o rodapé, por exemplo;

3. *plugins*: é constituída do arquivo de configurações do *Vuetify*;

Figura 3.14 – Página de cadastro de *sprints*



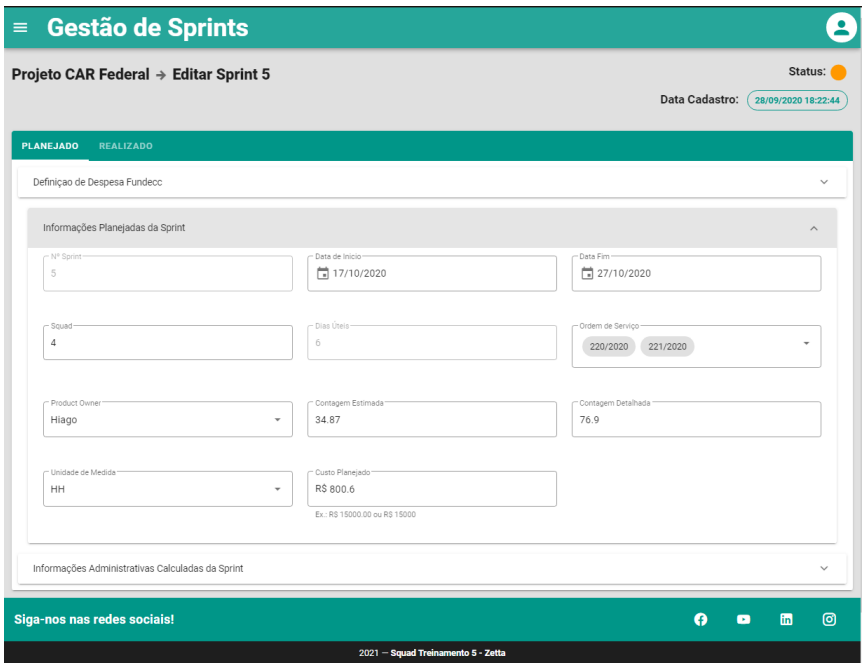
Fonte: Do Autor, 2021

Figura 3.15 – Método de submissão de *sprints*

```
894            submit () {
895                if (!this.$v.sprint.$invalid) {
896
897                    SprintServices.cadastrar(this.sprint)
898                        .then(() => {
899                            this.$toast.success('Sprint salva com sucesso!')
900                            window.location.href = this.cancelar + this.projetoId
901                        })
902                        .catch(error => {
903                            this.$toast.error("Erro: " + error.response.data.message)
904                            this.dialog_submit = false
905                            this.dialog_sucess = false
906                        })
907
908                } else {
909
910                    this.dialog_submit = false
911                    this.dialog_sucess = false
912                    this.$v.$touch()
913
914                }
915
916            },
```

Fonte: Agência Zetta, 2021

Figura 3.16 – Página de edição de *sprints*



Fonte: Do Autor, 2021

4. *router*: armazena o arquivo que configura as rotas e quais páginas serão exibidas quando acessadas;

5. *services*: possui todos os arquivos que controlam as requisições que o *front-end* faz aos serviços disponibilizados pelo *back-end*, vale ressaltar que estas conexões são realizadas com auxílio do Axios[8];

6. *store*: contém as configurações do *vuex*[9];

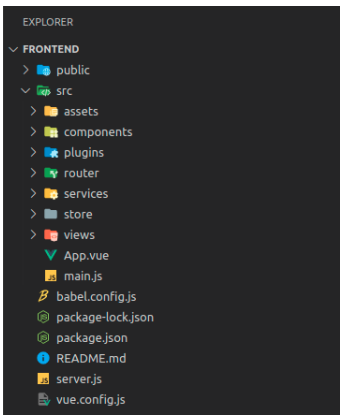
7. *views*: é composto pelas páginas do sistema.

Abaixo das pastas tem o arquivo *package.json*. Este arquivo funciona de forma semelhante ao *pom.xml* do *back-end*. O *package.json* armazena as infor-

[8] Axios é um cliente HTTP baseado em Promises para fazer requisições. Pode ser utilizado tanto no navegador quando no Node. js.(MARINHO, 2021)

[9] O Vuex é um padrão de gerenciamento de estado + biblioteca para aplicações Vue. js (VUEX, 2020)

Figura 3.17 – Estrutura *front-end* final do Sistema Gestão de Sprints



Fonte: Agência Zetta, 2021

mações de configuração para compilação do projeto e as dependências necessárias para o funcionamento do *front-end*. A Figura 3.18 mostra esse arquivo para o projeto Gestão de Sprints.

Figura 3.18 – Arquivo *package.json* do Sistema Gestão de Sprints

```
package.json
package.json > { } devDependencies
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  Debug
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint",
    "postinstall": "npm run build",
    "start": "node server.js"
  },
  "dependencies": {
    "axios": "^0.20.0",
    "connect-history-api-fallback": "^1.6.0",
    "core-js": "^3.6.5",
    "express": "^4.17.1",
    "moment": "^2.29.0",
    "moment-business-days": "^1.2.0",
    "v-tooltip": "^2.0.3",
    "vue": "^2.6.11",
    "vue-router": "^3.2.0",
    "vue-toast-notification": "^0.5.4",
    "vuelidate": "^0.7.5",
    "vuetify": "^2.2.11",
    "vuex": "^3.5.1"
  },
```

Fonte: Agência Zetta, 2021

3.1.1.4 O *deploy* do Sistema Gestão de *Sprints*

Após o término completo ou parcial do desenvolvimento de um sistema é comum disponibilizá-lo seja para a equipe de Analista de qualidade, seja para os clientes/usuários. Disponibilizar o sistema para uso ou testes é o que se chama de *deploy*. Para realização do *deploy*, foi utilizada a plataforma Heroku[10].

O Heroku é uma plataforma ou serviço em nuvem baseada em contêiner. O Heroku é comumente utilizado entre os desenvolvedores para implantar ou gerenciar aplicações e sistemas web (HEROKU, 2020).

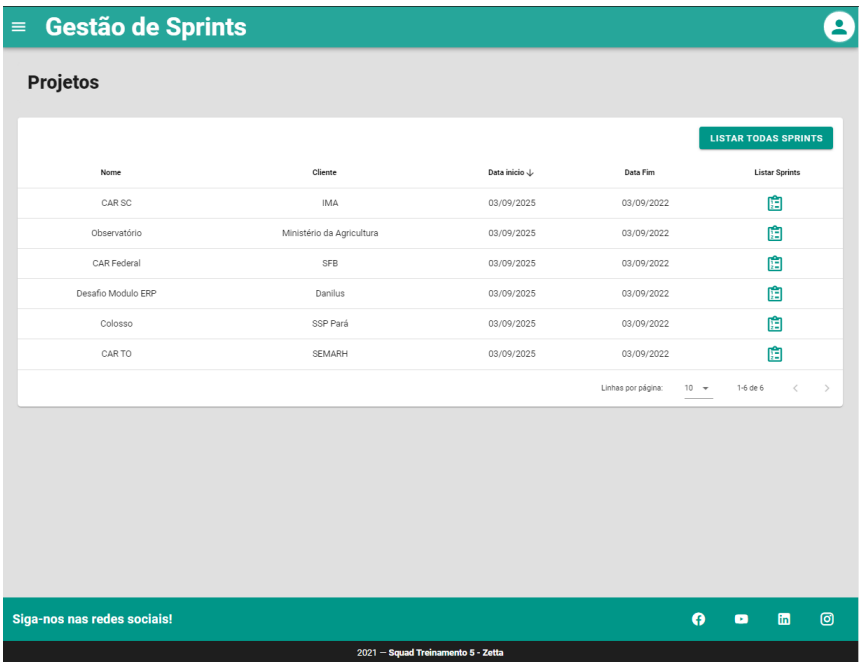
O *deploy* consistiu em três etapas. A primeira etapa foi o *deploy* do *back-end*. Para essa, etapa foi necessário adicionar o arquivo *system.properties*, o arquivo contém informações da versão do *Maven* e do Java do projeto. Com o *deploy* do *back-end* concluído foi gerado uma URL na qual o *back-end* do sistema estava hospedado, com isso foi inciada a segunda etapa.

A segunda etapa é o *deploy* do *front-end*. Para essa, foi necessário adicionar o arquivo *server.js* na raiz do *front-end*. Esse arquivo comporta uma série de configurações para que o *front-end* funcione corretamente após a realização do *deploy*. A outra modificação necessária foi no arquivo `src/services/api.service.js`. Neste arquivo o *axios*, que até então utilizava a rota de `localhost` do *back-end* irá utilizado a *url* do *back-end* disponibilizada pela *Heroku*.

A terceira etapa foi a criação do banco de dados no servidor do *Heroku*. Para isso, o *Heroku* disponibiliza o host, porta e credenciais de acesso ao banco de dados remoto. Desta forma foi possível com o uso de *scripts* criar as tabelas e populá-las com os dados necessários. A Figura 3.19 mostra a página inicial do sistema com dados populados após *deploy*.

[10] https://www.heroku.com/about

Figura 3.19 – Página inicial do Sistema Gestão de Sprints



Fonte: Do Autor, 2021

3.2 Hotsite PronaSolos

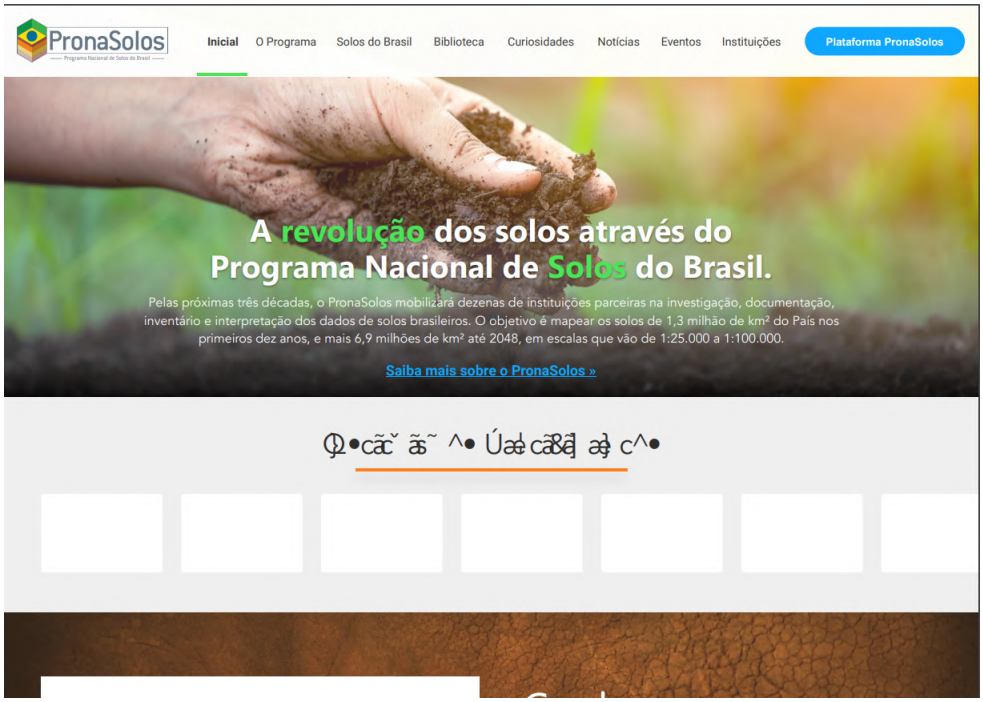
O PronaSolos[11] foi o primeiro projeto oficial da Zetta em que o estagiário teve participação. Este projeto foi desenvolvimento apenas com uso de tecnologias *front-end*, suas informações escritas são armazenadas em arquivos JSON que realiza a função de um mini banco de dados. O PronaSolos foi desenvolvido em um *squad* composto por três desenvolvedores, dois analistas de qualidade e um PO. Nesse projeto também foi seguida a metodologia *Scrum*.

No *planning* do projeto foi analisada de maneira minuciosa cada página do protótipo. Essa análise teve a intenção de identificar e quebrar cada atividade necessária para o desenvolvimento do hotsite. As atividades foram dispostas em um *board* do *Gitlab*. Este *board* funciona como um *Kanban* em que é possível visualizar todas as atividades prontas para desenvolvimento, em desenvolvimento,

[11] http://pronasolos.agenciazetta.ufla.br

desenvolvimento finalizado, pronta para teste e as atividades que voltaram para correções após análises dos analistas de qualidade. A Figura 3.20 ilustra o protótipo de alta fidelidade da página inicial do PronaSolos.

Figura 3.20 – Protótipo de alta fidelidade da tela inicial do PronaSolos



Fonte: Agência Zetta, 2021

3.2.1 Tecnologias utilizadas

Em consenso da equipe e seguindo orientações do *ScrumMaster*, o *squad* optou pelas seguintes escolhas de tecnologias para o desenvolvimento da aplicação:

- ReactJs e Material UI para o *front-end*.

3.2.2 Implementação da barra de navegação

A barra de navegação é um componente fixo no topo e esta presente em todas as páginas do hotsite. Esse componente é constituído de três elementos principais: o logo do PronaSolos, o menu de navegação e um botão que redireciona para a plataforma. A barra de navegação ainda possui a peculiaridade de possuir um pequeno tracejado verde que indica em qual página o usuário está acessando. A Figura 3.21 ilustra este componente.

Figura 3.21 – Barra de navegação do hotsite PronaSolos

Fonte: Pronasolos, 2021

A atividade de criação da barra de navegação consistia também na criação do arquivo que controla as rotas em que cada página faz acesso. A Figura 3.22 demostra o componente que controla as rotas do PronaSolos. A linha 35 indica que qualquer outra rota que não exista nas mapeadas anteriormente redirecionará o usuário para a página inicial.

Figura 3.22 – Arquivo de configurações de rotas do PronaSolos

```
19  ReactDOM.render(
20      <React.Fragment>
21          <ScrollUpButton />
22
23          <BrowserRouter>
24              <Header />
25              <ScrollTop />
26              <Switch>
27                  <Route path={process.env.REACT_APP_PATH} exact={true} component={PaginaInicial} />
28                  <Route path={process.env.REACT_APP_PATH + "o-programa"} exact={true} component={Programa} />
29                  <Route path={process.env.REACT_APP_PATH + "noticias"} exact={true} component={Noticias} />
30                  <Route path={process.env.REACT_APP_PATH + "eventos"} exact={true} component={Eventos} />
31                  <Route path={process.env.REACT_APP_PATH + "biblioteca"} exact={true} component={Biblioteca} />
32                  <Route path={process.env.REACT_APP_PATH + "curiosidades"} exact={true} component={Curiosidades} />
33                  <Route path={process.env.REACT_APP_PATH + "solos-do-brasil"} exact={true} component={SolosDoBrasil} />
34
35                  <Redirect from='*' to={process.env.REACT_APP_PATH} />
36
37              </Switch>
38              <Footer />
39          </BrowserRouter>
40      </React.Fragment>,
41      document.getElementById('root')
42  );
```

Fonte: Agência Zetta, 2021

3.2.3 Implementação da página O Programa

A página "O Programa"foi dividido em sete componentes. O **banner** da página constituído de uma foto e uma frase. A **apresentação do programa PronaSolos** composto por um textos e um vídeo. Os **eixos de atuação**, para esse componente foi criado um *card* em *css*, esse card foi replicado na quantidade de eixos existentes com suas respectivas informações, vide Figura 3.23. Os *Impactos do PronaSolos* também composto por *cards* estilizados em *css*.

Figura 3.23 – Eixo do PronaSolos - Página O programa

Fonte: PronaSolos, 2021

A **Linha do Tempo** trata-se de um componente do Material UI. Este componente recebe *card's* estilizados em css, cada um destes *card's* comporta dados de um acontecimento, bem como arquivos ou *links* para redirecionamento em outras páginas. A Figura 3.24 ilustra parte dessa linha do tempo.

Por ultimo tem-se os ultimos dois componentes, **Números do PronaSolos** e **Legislações relacionadas ao PronaSolos**. Ambos componentes são formados por *cards* estilizados em *css* e preenchidos com informações relacionadas aos temas de cada um.

Figura 3.24 – Linha do tempo do PronaSolos - Página O programa

Fonte: PronaSolos, 2021

3.2.4 Implementação da página Solos do Brasil

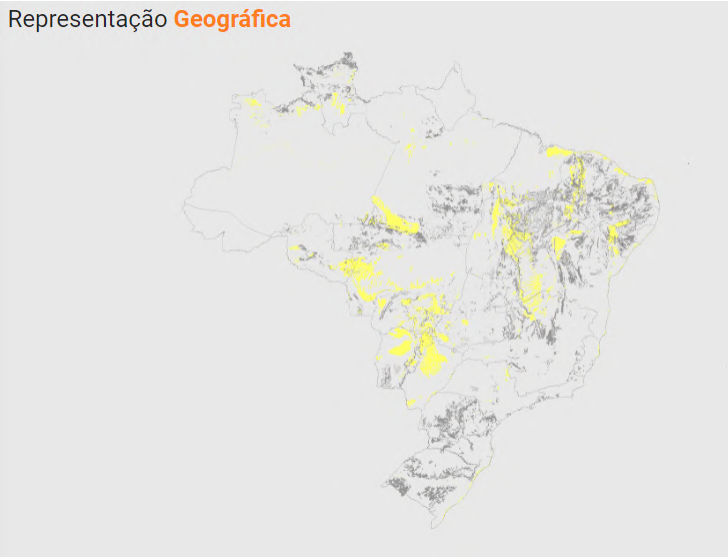
A página "Solos do Brasil"possui um menu lateral, este menu controla qual tipo de solo será exibido na página. No corpo da página tem-se mais cinco componentes. No primeiro componente tem uma imagem de fundo, o título do solo e uma frase representativa do solo.

O segundo componente possui uma descrição mais detalhada do solo e o local onde pode ser encontrado. Ao lado direito desse componente temos o terceiro. O terceiro componente apresenta os solos de segundo nível em forma de legenda para a representação gráfica do solo.

A Figura 3.25 representa o quarto componente. Este componente ilustra o mapa do Brasil apresentando os locais em que o solo em questão pode ser encontrado. O quinto e último componente são imagens representativas do solo.

Os componentes que formam esta página foram estilizados e implementados com auxílio de *css*. Com o *css* foi possível manter os elementos mais res-

Figura 3.25 – Representação gráfica do Neossolo - Página Solos do Brasil



Fonte: PronaSolos, 2021

ponsivos a adaptar cada componente da melhor maneira a apresenta-los para o usuário.

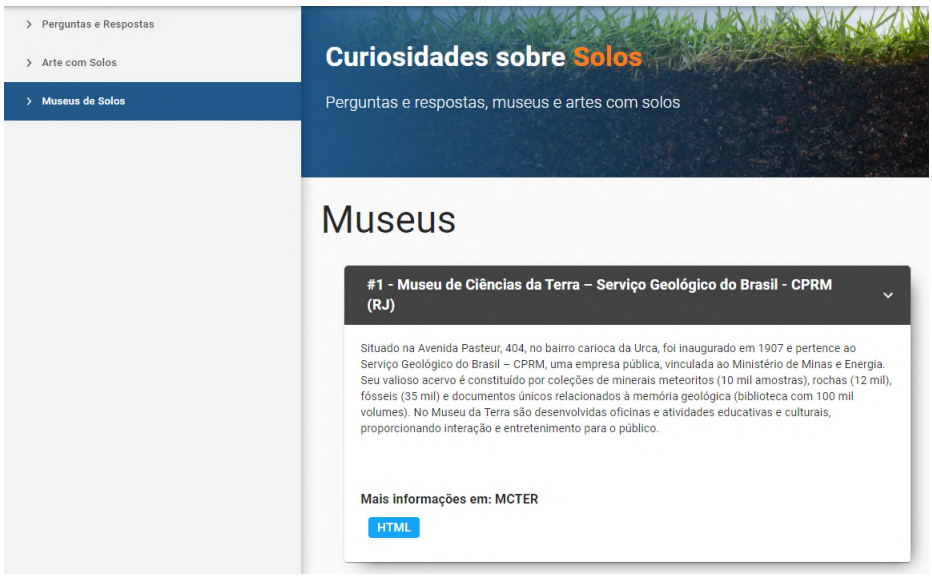
3.2.5 Implementação da página Curiosidades

A página "Curiosidades"é composta por um menu lateral com três sessões e outros três componentes. O primeiro componente é um *banner* fixo de apresentação da página, o qual foi estilizado em *css*. O outro componente altera de acordo com a sessão clicada. Esse componente é formado por *card's*, cada *card* é um painel expansível, componente do Material UI. Um painel pode comportar até cinco componentes, sendo eles: um título, um texto, imagens, um subtítulo e/ou botões. A Figura 3.26 apresenta parte dessa página com o painel do *card* expandido.

3.2.6 Implementação das página Notícias e Eventos

Essas duas páginas funcionam de forma semelhante, modificando apenas o conteúdo exibido. As páginas são composta por apenas dois componentes o *Título*

Figura 3.26 – Página Curiosidades



Fonte: PronaSolos, 2021

e uma sequência de *cards*. Os *cards* foram estilizados também em *css*. Cada *card* e composto por até quatro componentes: uma imagem, uma data, um título que também é um *hiperlink*[12] e um subtítulo. A Figura 3.27 ilustra parte da página de notícias do hotsite PronaSolos.

Figura 3.27 – Página Notícias

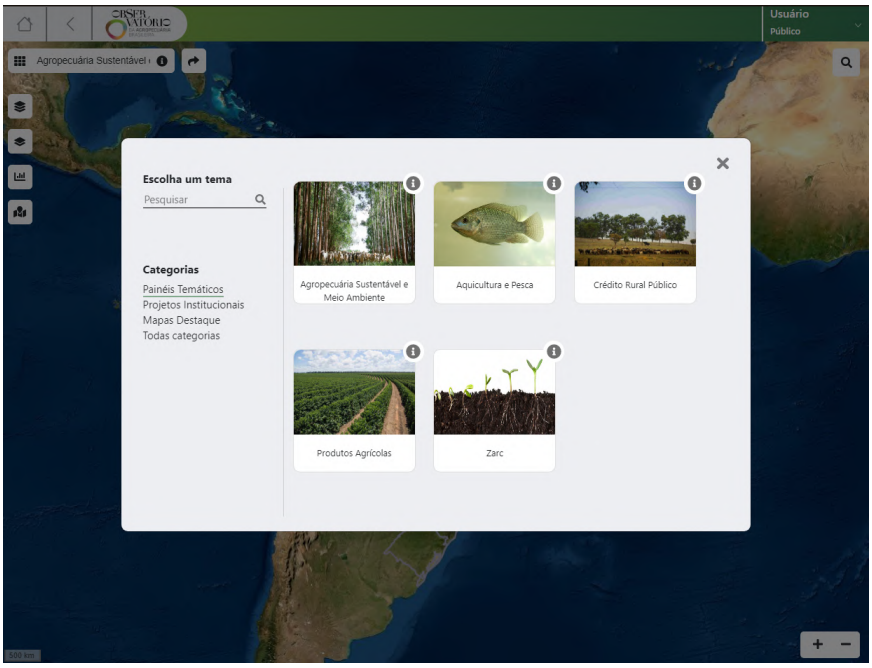
Fonte: PronaSolos, 2021

[12] Hiperlink ou hiperligação é qualquer elemento de uma página web que faça referência a outro texto ou página web (MIRANDA, 2005)

3.3 Observatório: Plataforma Geoespacial

A Plataforma geoespacial é um dos projetos que compõe o Projeto Observatório. Esse projeto é um mapa interativo que permite ao usuário consultar de forma detalhada uma grande diversidade de dados e informações da agropecuária brasileira no decorrer dos anos. O mapa possui cinco temas distintos, sendo eles: *Agropecuária Sustentável e Meio Ambiente*, *Aquicultura e Pesca*, *Crédito Rural Público*, *Produtos Agrícolas* e *Zarc*. Cada tema possui mapas e características diferentes, porém, a estrutura de camadas, relatórios e filtros contendo semelhanças no funcionamento. A Figura 3.28 mostra a página inicial da Plataforma Geoespacial.

Figura 3.28 – Plataforma Geoespacial - página inicial



Fonte: Agência Zetta, 2021

A participação do estagiário nesse projeto foi iniciado quando o mesmo já estava em andamento. Portanto, nesse capítulo é abordado apenas os principais pontos de participação do estagiário. O desenvolvimento deste projeto ocorreu

adotando ambas metodologias de desenvolvimento, ora era utilizado Scrum, ora utilizava Kanban e em determinados momentos eram utilizados características de ambas metodologias. O *squad* optava pela melhor metodologia que comportava a cada demanda de atividades solicitada pelos clientes.

3.3.1 Tecnologias utilizadas

O *squad* trabalhou com as seguintes tecnologias para o desenvolvimento da aplicação:

- Java e SpringBoot para o *back-end*;
- VueJs, BootstrapVue, Less , Pug e GeoServer, Leaflet para o *front-end*;

3.3.2 Implementação dos relatórios de cruzamentos de camadas

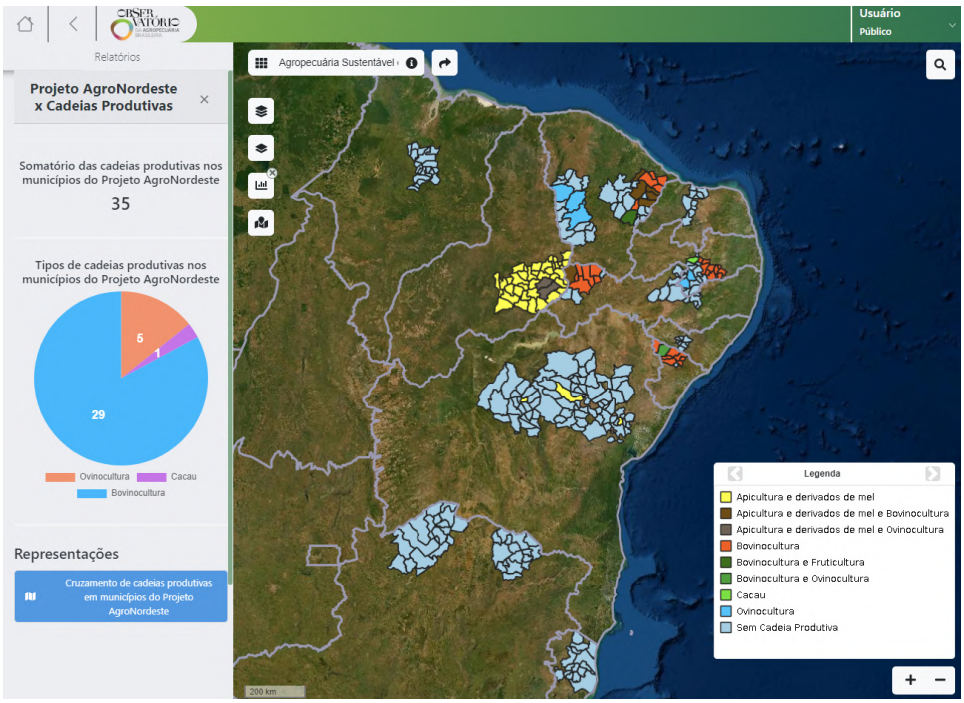
Camada é o termo adotado pelo *squad* para referir a quais características serão apresentadas no mapa, por exemplo: a camada de estados apresenta delimitação geográfica dos estados brasileiro no mapa. A Plataforma permite que as camadas sejam ativadas ou desativadas a qualquer momento. As camadas são armazenadas e configuradas no GeoServer. Entender e aprender a trabalhar nessa plataforma foi de suma importância para que o estagiário conseguisse desenvolver atividades na Plataforma Geoespacial.

Como o nome sugere, os relatórios de cruzamento de camadas consiste em unir duas ou mais camadas e obter informações relevantes para o usuário. Esses relatórios podem ser encontrados no menu **Relatórios** dos temas. A Figura 3.29 ilustra parte do relatório **Projeto AgroNordeste x Cadeias Produtivas**.

3.3.3 Implementação da ferramenta de busca

A ferramenta de busca trata de um componente retrátil presente no mapa. Este componente facilita a navegação do usuário pelo mapa permitindo a busca

Figura 3.29 – Relatório Projeto AgroNordeste x Cadeias Produtivas



Fonte: Agência Zetta, 2021

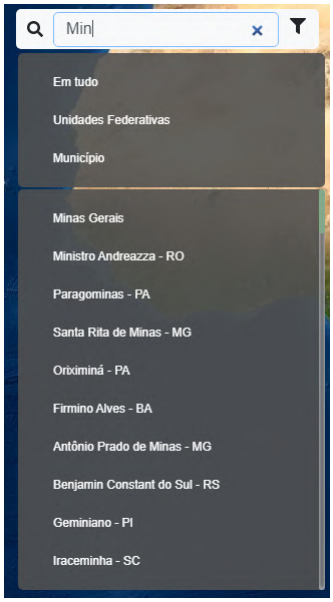
por município ou estado. Esse componente conta com um filtro que permite ao usuário filtrar a busca em Unidades Federativas, Municípios ou em ambos.

Esse componente conta com quatro elementos principais. Dois ícones sendo eles um ícone de lupa e um ícone de filtro, uma caixa de texto e um menu *dropdown*. O menu *dropdown* é preenchido de acordo com o resultado dos caracteres digitados na caixa de texto. A Figura 3.30 ilustra o componente em questão.

3.3.4 Implementação da *Pop-up* de relatório rápido

Popup's são janelas ou *card's* que aparecem na tela após alguma interação do usuário ou configuração pré programada do sistema. Neste caso, o uso de *pop-up* foi uma funcionalidade projetada para o sistema exibir informações da região clicada no mapa, esta região é limitada pelo território brasileiro. A *pop-up* apresenta informações de todas as camadas ativas no momento do *click*.

Figura 3.30 – Ferramenta de busca

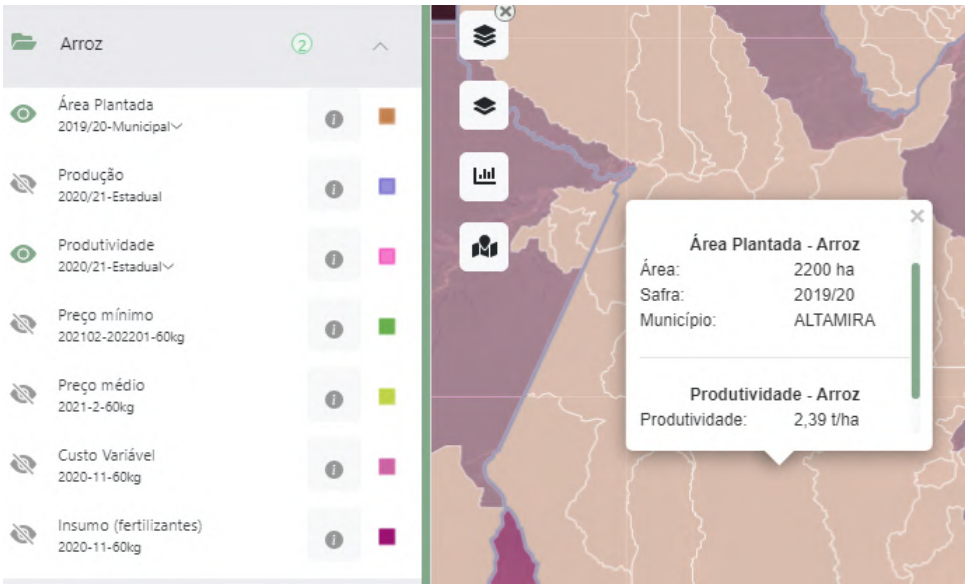


Fonte: Agência Zetta, 2021

A *popup* é um componente simples que preenche uma tabela HTML com as informações recebidas do *back-end*. Para que o *back-end* retorne as informações de forma correta, o *front-end* armazena as informações das camadas ativas. O mapa é um componente *Leaflet*, este componente possui inúmeras informações e métodos do mapa em si, dentre estes métodos temos um que retorna informações de localização. Portanto é possível calcular a localização do *click* a partir da latitude e longitude correspondente a região do clicada no mapa.

A Figura 3.31 mostra o comportamento do componente ao se clicar no mapa com três camadas ativas: **Unidades Federativas**, **Área Plantada** e **Produtividade** do produto Arroz. Nessa figura, pode-se extrair informações das três camadas ativas no momento do *click*, o **estado**, a **área plantada** de arroz em estratificação municipal e a **produtividade** de arroz em estratificação estadual. A barra de rolagem indica que é possível visualizar estas informações, rolando verticalmente o *scroll*.

Figura 3.31 – *Popup* de relatório rápido



Fonte: Agência Zetta, 2021

4 CONSIDERAÇÕES FINAIS

O estágio agregou inestimável valor para a formação do aluno. Mesmo possuindo uma bolsa com certo valor financeiro, o principal papel do estágio é fornecer conhecimento além das salas de aula, experiência prática e ingressar o aluno no mercado de trabalho. Por meio do estágio, o aluno tem a oportunidade de colocar em prática os conhecimentos adquiridos nas disciplinas ofertadas pelo curso de Ciência da Computação.

Em cada etapa de desenvolvimento, durante o estágio, o aluno pôde associar os desenvolvimentos às disciplinas cursadas durante a graduação. A disciplina de Introdução aos Algoritmos foi a primeira oportunidade do aluno de ver e entender a lógica de programação. Disciplinas como Estruturas de dados e Paradigmas de Linguagem de Programação contribuíram solidificando a lógica de programação do aluno ao apresentá-lo para resolver e implementar diversos problemas.

Criação de *scripts*, armazenamento, busca, edição e remoção de informações em banco de dados se tornaram mais compreensíveis e solucionáveis graças às disciplinas relacionadas, como: Introdução a Sistemas de Banco de Dados e Sistemas Gerenciadores de Banco de Dados. Práticas de Programação Orientada a Objetos e Engenharia de Software introduziram o desenvolvimento de sistemas com interface gráfica, bem como analisar, descrever e implementar os requisitos de um sistema, posteriormente vistos de forma mais aprofundada e teórica em Interação Humano-Computador.

Devido a pandemia do Covid-19 foi necessário acesso remoto com VPN[1] aos computadores e servidores da Zetta. Disciplinas como Redes de computadores, Sistemas Distribuídos e Programação Paralela e Concorrente auxiliaram no compreendimento e funcionamento da rede, bem como acessar e interagir remotamente com outros computadores.

[1] VPN ou Virtual Private Network (Rede Privada Virtual) trata-se de uma rede privada construída sobre a infraestrutura de uma rede pública. Essa é uma forma de conectar dois computadores através de uma rede pública, como a Internet(CIPOLI, 2020)

Em Metodologia de Pesquisa o aluno pôde conhecer quais os tipos de trabalhos de conclusão de curso oferecidos pelo Curso de Ciência da Computação na UFLA, sua estrutura e as principais regras de escrita em cada um.

As atividades e os projetos nos quais o estagiário teve participação contribuíram de forma gratificante e exponencial para seu desenvolvimento profissional e pessoal. A **Imersão Z** introduziu o estagiário no ambiente da Zetta ajudando-o a compreender o funcionamento de seus setores, conhecer os projetos, os colaboradores e a se sentir uma parte em construção importante da agência. No projeto **Sistema de Gestão de *Sprints*** o estagiário pôde colocar em prática a carga de conhecimento adquirido durante os cursos ofertados, treinar o trabalho em uma equipe direcionada pela metodologia *Scrum* com todos seus ritos, a compreender o funcionamento das tecnologias utilizadas em conjunto e a função que cada uma exerce no sistema. O **Hotsite PronaSolos** foi importante para o estagiário explorar e aprofundar seu conhecimento em *layouts* e *css* compondo um site do início ao fim com base em um protótipo de alta fidelidade. No projeto **Plataforma Geoespacial** diferentemente dos anteriores, quando o estagiário ingressou este estava em desenvolvimento. Nesse projeto o estagiário pôde aprofundar ainda mais os conhecimentos e habilidade adquiridas nos projetos anteriores devido ao alto nível de complexidade que exigia no desenvolvimento de cada novo componente ou manutenção e refatoração de componentes existentes.

Portanto, ao somar toda essas experiências, o estagiário adquiriu uma bagagem de aprendizado e experiências imensuráveis. Em cada *framework* de desenvolvimento, o trabalho em equipe e as soluções empregadas nas atividades tiveram grande importância em seu crescimento profissional e pessoal. O estágio provou ser um elemento muito importante no desenvolvimento do aluno anterior ao seu ingresso no mercado de trabalho. Por meio do estágio, o aluno possui uma experiência de desenvolvimento de aplicações web estando apto para novos desafios em sua vida profissional.

REFERÊNCIAS

AFONSO, A. **O que é Spring Boot?** 2017. Disponível em: <https://blog.algaworks.com/spring-boot/>. Acesso em: 13 mar. 2021.

AMBERJ. **5 Delightful Things about Material-UI**. 2019. Disponível em: <https://dev.to/amberjones/5-delightful-things-about-material-ui-5402>. Acesso em: 10 mar. 2021.

ATLASSIAN. **O que é controle de versão**. 2020. Disponível em: <https://www.atlassian.com/br/git/tutorials/what-is-version-control>. Acesso em: 06 abr. 2021.

BERTOLA, F. **Git, Github e Gitlab: o que são e principais diferenças**. 2019. Disponível em: <https://www.zup.com.br/blog/git-github-e-gitlab>. Acesso em: 06 abr. 2021.

BRASIL, U. **React: o que é e como funciona essa ferramenta?** 2018. Disponível em: <https://tableless.com.br/react-o-que-e-e-como-funciona-essa-ferramenta/>. Acesso em: 10 mar. 2021.

CERON, V. **O que é Bootstrap: Tudo sobre este Framework**. 2020. Disponível em: <https://www.cupcom.com.br/programacao/como-utilizar-o-bootstrap-vue.html>. Acesso em: 15 mai, 2021.

CIPOLI, P. **O que é VPN?** 2020. Disponível em: <https://canaltech.com.br/internet/o-que-e-vpn-23748/>. Acesso em: 17 abr. 2021.

DCOM UFLA. **UFLA apresenta a Zetta, nova Agência de Inovação focada em geotecnologia e sistemas inteligentes**. 2020. Disponível em: <https://ufla.br/noticias/institucional/13712-ufla-apresenta-a-zetta-nova-agencia-de-inovacao-focada-em-geotecnologia-e-sistemas-inteligentes>. Acesso em: 27 out. 2020.

EGIDIO, A. F. **O que é SASS e LESS na Web?** 2020. Disponível em: <https://www.javaavancado.com/o-que-e-sass-e-less-na-web/>. Acesso em: 15 mai, 2021.

ESCUDELARIO, B. de F. **Códigos HTML mais organizados e limpos com Pug**. 2017. Disponível em: <https://imasters.com.br/desenvolvimento/codigos-html-mais-organizados-e-limpos-com-pug>. Acesso em: 15 mai, 2021.

ESPINHA, R. G. **Kanban: o que é e como funciona**. 2019. Disponível em: <https://artia.com/kanban>. Acesso em: 06 abr. 2021.

FERNANDES, H. M. **O que é JSON e para que serve?** 2020. Disponível em: <https://marquesfernandes.com/tecnologia/o-que-e-json-e-para-que-serve/>. Acesso em: 18 abr. 2021.

FLANAGAN, D. **JavaScript: o guia definitivo**. [S.l.]: Bookman Editora, 2004.

FREITAS, L. R. **CI/CD, Entenda o que é, e como Estruturar esses Processos com a Accurate**. 2020. Disponível em: <https://blog.accurate.com.br/ci-cd-entenda-o-que-e/>. Acesso em: 18 abr. 2021.

GALDINO, F. **Vue.js Tutorial**. 2017. Disponível em: <https://www.devmedia.com.br/vue-js-tutorial/38042>. Acesso em: 13 mar. 2021.

GAMA, A. **Inversão de Controle x Injeção de Dependência**. 2010. Disponível em: <https://www.devmedia.com.br/inversao-de-controle-x-injecao-de-dependencia/18763>. Acesso em: 05 abr. 2021.

GOSLING, J.; HOLMES, D. C.; ARNOLD, K. **The Java programming language**. [S.l.]: Addison-Wesley, 2005.

HEROKU. **What is Heroku?** 2020. Disponível em: <https://www.heroku.com/about>. Acesso em: 11 abr. 2021.

LEAFLET. **Leaflet**. 2020. Disponível em: <https://leafletjs.com/index.html>. Acesso em: 15 mai, 2021.

LINHARES, M. Introduçao ao hibernate 3. **Grupo de usuários Java. Disponível em< http://www. guj. com. br/content/articles/hibernate/intruducao_hibernate3_guj. pdf>. Acesso em**, v. 12, 2016.

LUCIDCHART. **O que é um diagrama entidade relacionamento?** 2020. Disponível em: <https://www.lucidchart.com/pages/pt/o-que-e-diagrama-entidade-relacionamento>. Acesso em: 10 abr. 2021.

MARINHO, T. **Axios - um cliente HTTP Full Stack**. 2021. Disponível em: <https://blog.rocketseat.com.br/axios-um-cliente-http-full-stack/>. Acesso em: 10 abr. 2021.

MARQUES, R. **O que é HTML? Entenda de forma descomplicada**. 2020. Disponível em: <https://www.homehost.com.br/blog/tutoriais/o-que-e-html/>. Acesso em: 10 mar. 2021.

MARQUES, R. **O que é Bootstrap: Tudo sobre este Framework**. 2021. Disponível em: <https://www.homehost.com.br/blog/tutoriais/o-que-e-bootstrap/>. Acesso em: 10 abr, 2021.

MILANI, A. **PostgreSQL-Guia do Programador**. [S.l.]: Novatec Editora, 2008.

MILETTO, E. M.; BERTAGNOLLI, S. de C. **Desenvolvimento de Software II: Introdução ao Desenvolvimento Web com HTML, CSS, JavaScript e PHP-Eixo: Informação e Comunicação-Série Tekne**. [S.l.]: Bookman Editora, 2014.

MIRANDA, R. **Hyperlinks / Link – Visão Geral**. 2005. Disponível em: <http://www.otimizacao-sites-busca.com/links/hlink.htm>. Acesso em: 18 abr. 2021.

NASCIMENTO, T. **Usabilidade em interfaces CRUD: o guia completo**. 2018. Disponível em: <http://thiagonasc.com/usabilidade/usabilidade-interfaces-crud>. Acesso em: 04 abr. 2021.

NIZZOLA, M. **Inversão de Controle x Injeção de Dependência**. 2020. Disponível em: <https://marcionizzola.medium.com/afinal-o-que-é-injeç~ao-de-dependência-68131c864a79>. Acesso em: 06 abr. 2021.

OKI, W. **Primeiros passos com o Spring Boot**. 2015. Disponível em: <https://www.devmedia.com.br/primeiros-passos-com-o-spring-boot/33654>. Acesso em: 13 mar. 2021.

PAGE, M. **Perspectivas para o mercado de TI em 2021**. 2020. Disponível em: <https://www.michaelpage.com.br/advice/liderança-e-gest~ao/integraç~ao-e-engajamento/perspectivas-para-o-mercado-de-ti-em-2021>. Acesso em: 06 abr. 2021.

PIETRO, L. **Processo de desenvolvimento de Software Agência Zetta**. 2020. Disponível em: <https://gitlab.ti.lemaf.ufla.br/zetta/processo-de-desenvolvimento-de-software/-/wikis/home>. Acesso em: 15 mai, 2021.

PRESOTTO, A. **Kanban: o que é e como funciona**. 2021. Disponível em: <https://www.alura.com.br/artigos/metodo-kanban>. Acesso em: 06 abr. 2021.

QUADRO, F. S. de. **Por dentro do GeoServer**. 2009. Disponível em: <https://mundogeo.com/2009/07/09/por-dentro-do-geoserver/>. Acesso em: 15 mai, 2021.

RAGGETT, D. et al. Html 4.01 specification. **W3C recommendation**, v. 24, 1999.

RIVEROS, F. **LESS CSS – O que é e como Otimizar seu CSS**. 2018. Disponível em: <https://www.scriptcaseblog.net/pt/development-pt/less-css-o-que-e-e-como-otimizar-seu-css/>. Acesso em: 15 mai, 2021.

SAVOINE, M. et al. Análise de gerenciamento de projeto de software utilizando metodologia ágil xp e scrum: Um estudo de caso prático. **XI Encontro de Estudantes de Informática do Tocantins**, p. 93–102, 2009.

SCHMITZ, D. **Conheça o Vuetify - parte 1**. 2018. Disponível em: <https://vuejs-brasil.com.br/conheca-o-vuetify-tutorial-dicas-parte-1/>. Acesso em: 13 mar. 2021.

SOARES, M. dos S. Metodologias ágeis extreme programming e scrum para o desenvolvimento de software. **Revista Eletrônica de Sistemas de Informação**, v. 3, n. 1, 2004.

SOUTO, M. **O que é front-end e back-end?** 2019. Disponível em: <https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end>. Acesso em: 13 mar. 2021.

VUEX. **O que é Vuex?** 2020. Disponível em: <https://vuex.vuejs.org/ptbr/>. Acesso em: 18 abr. 2021.