
**VICTOR HUGO DE ANDRADE LANDIN**

# RELATÓRIO DE ESTÁGIO - DESENVOLVIMENTO E MANUTENÇÃO DE SISTEMAS WEB NA AGÊNCIA ZETTA

**LAVRAS – MG**

**2021**

**VICTOR HUGO DE ANDRADE LANDIN**

# RELATÓRIO DE ESTÁGIO - DESENVOLVIMENTO E MANUTENÇÃO DE SISTEMAS WEB NA 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. Paulo Afonso Parreira Júnior

Orientador

**LAVRAS – MG**

**2021**

**VICTOR HUGO DE ANDRADE LANDIN**

# RELATÓRIO DE ESTÁGIO - DESENVOLVIMENTO E MANUTENÇÃO DE SISTEMAS WEB NA 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 Abril de 2021.

| | |
|---|---|
| Prof. DSc. Paulo Afonso Parreira Júnior | UFLA |
| Prof. DSc. Renata Teles Moreira | UFLA |
| Antônio Armando De Oliveira Barbosa | ZETTA |

Prof. Paulo Afonso Parreira Júnior
Orientador

**LAVRAS – MG**
**2021**

## AGRADECIMENTOS

Primeiramente, agradeço meus pais, José dos Reis Landin e Sônia Maria Faustino de Andrade, e minha irmã, Gabrielle de Andrade Landin, por todo o apoio que permitiu à mim, buscar minha formação superior, por sempre me incentivar a estudar e fornecer as condições para isso. Agradeço a todos os amigos que me apoiaram e aconselharam em todo o período de graduação. Agradeço também a todos os professores da UFLA que se dedicaram a fornecer um ensino de alta qualidade, em especial aos professores Paulo Afonso Parreira Júnior e Erick Galani Maziero, por contribuírem com minha formação para além das salas de aula.

*"Se a educação sozinha não transforma a sociedade, sem ela tampouco a sociedade muda."*
*(Paulo Freire)*

## RESUMO

Este relatório de estágio relata as atividades desenvolvidas na Agência Zetta, na área de desenvolvimento de sistemas *Web*. Os projetos em que o estagiário atuou foram Gestão de Contratos (responsável por manter os contratos da agência), Análise Automatizada do SICAR (responsável por analisar informações de cadastro de imóveis rurais) e Retificação Automática do SICAR (responsável por retificar as informações apresentadas no Análise Automatizada, desde que haja inconsistências). Neste relatório, são apresentadas as atividades desenvolvidas e as tecnologias e metodologias adotadas no estágio. Dentre os resultados alcançados com o estágio, destaca-se o aprimoramento profissional por parte do estagiário, que pode aplicar na prática os conceitos vistos nas disciplinas do curso de graduação.



**Palavras-chave:** Sistemas Web. Estágio supervisionado. Desenvolvimento de Software. Manutenção de Software.

## ABSTRACT

This internship report describe the activities developed at *Agência Zetta*, in the area of web systems development. The projects in which the intern worked were *Gestão de Contratos* (responsible for maintaining the agency's contracts), *Análise Automatizada do Sicar* (responsible for analyzing information registration of rural properties) and *Retificação Automática do Sicar* (responsible for rectifying the information presented in the Automated Analysis, as long as there are inconsistencies). This report presents the activities developed and the technologies and methodologies adopted in the internship. Among the results achieved with the internship, the professional improvement on the part of the intern stands out, who was able to apply in practice the concepts seen during the disciplines of the undergraduate course.



**Keywords:** Web systems. Supervised internship. Software development. Software maintenance.

## LISTA DE FIGURAS

**SUMÁRIO**

# 1 INTRODUÇÃO

Este capítulo apresenta o contexto do estágio supervisionado, uma breve descrição da Agência Zetta, de seus projetos e de sua organização, bem como os principais objetivos para a realização deste trabalho.

## 1.1 Contextualização

Devido à chegada da pandemia do Coronavírus (COVID-19), houve uma aceleração no processo de digitalização de muitas empresas, fazendo com que o mercado de TI (Tecnologia da Informação) ganhasse destaque ainda maior no processo de transformação digital. Com isso, o papel do desenvolvedor de software se tornou ainda mais imprescindível para a criação de soluções em diferentes áreas de negócio. Associada a isso existe a necessidade de se ter profissionais capacitados e qualificados para atender a essa demanda com qualidade, uma vez que muitas vagas de trabalho nessa área não são preenchidas devido à falta de qualificação profissional (JORNAL CONTÁBIL, 2020).

Aliado a isso, o interesse do estagiário em adquirir conhecimento prático na área de desenvolvimento de software fez com que o mesmo buscasse a realização de estágio supervisionado na Agência Zetta, na área de desenvolvimento *Web*, com o intuito de obter conhecimento, experiência e vivência profissional.

O estágio relatado neste documento foi realizado na modalidade remota, em decorrência do distanciamento social adotado pela agência, com o intuído de colaborar com as autoridades do país para diminuir a propagação do coronavírus, causador da doença *Covid-19*.

## 1.2 Sobre a Agência Zetta

O estágio supervisionado relatado neste trabalho foi realizado na Agência Zetta, que é uma agência de inovação da UFLA (Universidade Federal de La-

vras), com enfoque em geotecnologias e sistemas inteligentes (ZETTA, 2020). A agência nasceu das experiências executadas pela UFLA de forma multidisciplinar, tendo o LEMAF (Laboratório de Estudos e Projetos em Manejo Florestal) como grande fonte de inspiração. Este laboratório trabalhou por mais de 25 anos integrado ao Departamento de Ciências Florestais, sendo responsável por grandes projetos, como o Sistema Nacional de Cadastro Rural (SICAR), entre outros (DCOM UFLA, 2020). A Agência Zetta surge a partir do LEMAF, no ano de 2020, buscando integração institucional que permita maior transversalidade entre docentes de diferentes unidades acadêmicas, especificamente em Geotecnologias e Sistemas Inteligentes vinculados ao agronegócio (DCOM UFLA, 2020).

A organização da agência em relação às equipes de desenvolvimento de software é caracterizada por tribos, que são equipes compostas por um grupo de sub-equipes, chamados *squads*. O termo *squad* faz alusão a esquadrões militares, que consistem de pequenos grupos de soldados com diferentes especialidades e que têm autonomia para planejar e executar tarefas de forma independente. Nos *squads*, há colaboradores que assumem papéis específicos, como os desenvolvedores, que trabalham na implementação do software, os analistas de qualidade, que realizam diferentes tipos de testes para verificar sua qualidade, e o *Product Owner*, que é o responsável por direcionar o projeto, de acordo com as necessidades do cliente e das demais partes interessadas.

## 1.3 Objetivos

Dentre os principais objetivos da realização do processo de estágio, pode-se citar a introdução do estagiário em um ambiente de trabalho em que o mesmo pode aplicar os conhecimentos aprendidos durante a graduação. Associado a esse objetivo, tem-se outros objetivos como (i) a adaptação às rotinas existentes no dia a dia de uma organização; (ii) contato com novas tecnologias; (iii) participação em

experiências de projetos reais, com ênfase no trabalho em equipe; e (iv) estabelecimento de novos objetivos profissionais após o término da graduação.

Este relatório tem como principal objetivo apresentar as atividades desenvolvidas durante o estágio supervisionado realizado na Agência Zetta, no período de Julho a Dezembro de 2020. Neste período, o estagiário atuou como desenvolvedor em 3 (três) projetos, sendo eles: Sistema de Controle de Contratos da Agência, Análise Dinamizada do SICAR (Sistema Nacional de Cadastro Ambiental Rural) e Retificação Automática do SICAR.

Algumas das atividades realizadas ao longo do estágio são resumidamente apresentadas a seguir; mais detalhes são apresentados na Seção 3. Dentre essas atividades, estão presentes a criação de novas funcionalidades e a manutenção e correção de erros. No Sistema de Controle de Contratos, que é um projeto para ser utilizado dentro da própria Agência, foram desenvolvidas diversas funcionalidades, como por exemplo: (i) a tela inicial, com menu lateral e listagem de contratos em tabela; e (ii) as telas de cadastro, consulta e alteração de contratos. Nos módulos de Análise Dinamizada e Retificação Automática, ambos do SICAR, não foram desenvolvidas novas funcionalidades, porém foram realizadas manutenções, tais como: (i) alterações de textos das telas, conforme demanda do cliente; (ii) alteração de mensagem de retorno quando o usuário solicita a geração de um relatório do tipo PDF; (iii) exibição de tabela com dados sobre documentos duplicados e inconsistências no cadastro de imóvel rural; entre outros.

## 1.4 Estrutura do trabalho

Este relatório está organizado como segue: (i) no Capítulo 2 é apresentado o referencial teórico deste trabalho, destacando-se as metodologias, tecnologias e ferramentas utilizadas ao longo do estágio; (ii) no Capítulo 3 são apresentadas em mais detalhes as atividades realizadas pelo estagiário na Agência Zetta; e (iii) por fim, o Capítulo 4 apresenta as considerações finais deste relatório.

# 2 REFERENCIAL TEÓRICO

Neste capítulo são relatados os conceitos referentes às metodologias gerenciais, tais como o *Scrum* e *Kanban*, às tecnologias de desenvolvimento *Front-end* e *Back-end* e às tecnologias de comunicação entre sistemas utilizadas ao longo do estágio.

## 2.1 Scrum e Kanban

Para que diversos indivíduos trabalhem em um mesmo projeto, é necessário adotar formas de organização de atividades e comunicação, com o intuito de se obter resultados positivos. Neste cenário, durante o estágio, foram utilizadas as metodologias de gerenciamento de projetos e de atividades *Scrum* e *Kanban*.

*Scrum* é uma metodologia ágil orientada para a gestão e planejamento de projetos, muito difundida no contexto de projetos de software. Nesta metodologia, há alguns ritos e papéis para auxiliar as equipes. Entre os papéis, estão: (i) o *Scrum Team*, que é a equipe de desenvolvimento do projeto; (ii) o *Scrum Master*, que é responsável por assegurar que a equipe respeite os ritos do *Scrum* e por remover possíveis obstáculos que possam surgir; e (iii) o *Product Owner* (PO), que é o membro responsável por direcionar o projeto de acordo com a necessidade do cliente e das demais partes interessadas (PAULA, 2016).

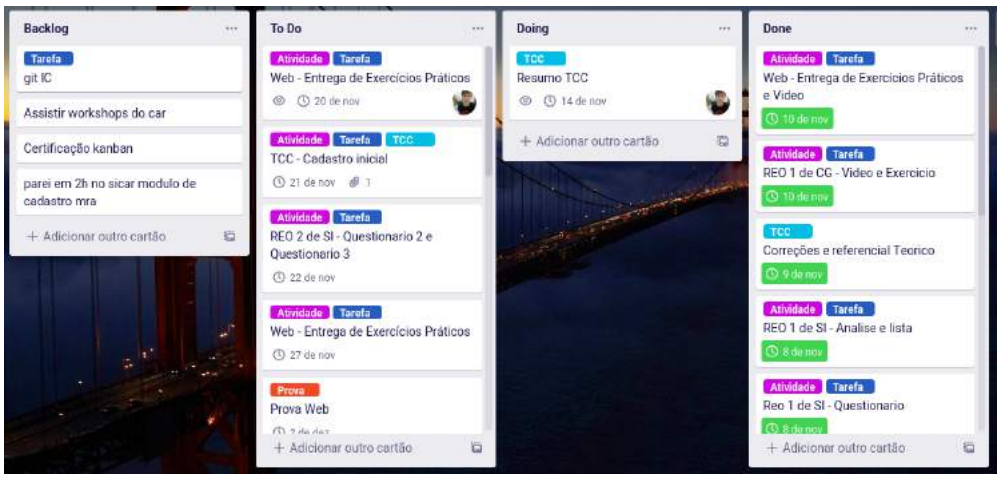
No *Scrum*, os projetos são divididos em ciclos, chamados de *Sprints*. Uma *Sprint* é um período de tempo (*time-box*), no qual um conjunto de atividades deve ser executado. Tipicamente, uma *Sprint* dura entre 2 e 4 semanas, ficando a cargo da equipe essa escolha da duração. As atividades a serem executadas são mantidas em uma lista, conhecida como *Product Backlog* e, em cada *Sprint*, é selecionada uma porção dessas atividades para serem implementadas (PAULA, 2016). Dentre os ritos do *Scrum*, há algumas reuniões para disseminar conhecimentos sobre as atividades, levantar possíveis erros e para planejar as próximas atividades e *Sprints*. Entre essas reuniões, têm-se: (i) *Sprint Planning*, que é uma reunião de

planejamento da *Sprint*, na qual o PO prioriza os itens do *Product Backlog* e o *Scrum Team* seleciona as atividades que ele será capaz de implementar durante a *Sprint*; (ii) *Daily Scrum*, que é uma breve reunião que ocorre diariamente e, nela, todos os membros do *Scrum Team* expõem as atividades que realizaram desde a última *Daily Scrum*, quais atividades realizarão após a reunião e quais impedimentos ou obstáculos ocorreram nesse período; (iii) *Sprint Review*, que é uma reunião que ocorre ao final de uma *Sprint* e, nela, são apresentadas as funcionalidades que foram desenvolvidas neste período, avaliando se os objetivos dessa *Sprint* foram atingidos ou não; e (iv) *Sprint Retrospective*, que é uma reunião ao final da *Sprint*, em que se identifica o que funcionou bem, o que pode ser melhorado e que ações devem ser tomadas para esse melhoramento (PAULA, 2016).

Já a metodologia *Kanban* propõe a utilização de cartões em um quadro para indicar e acompanhar, visualmente, o andamento da execução das atividades (ESPINHA, c2020). Em geral, esse quadro possui 4 (quatro) colunas ordenadas (conforme apresentado na Figura 2.1), que representam estados das atividades do projeto, sendo elas: (i) a lista de atividades que necessitam ser executadas, chamada de *Backlog*; (ii) a lista de atividades prontas para desenvolvimento, chamada de *To do*; (iii) a lista de atividades em andamento, chamada de *Doing*; e (iv) a lista de atividades finalizadas, chamada de *Done*. Conforme as tarefas são desempenhadas, os cartões referentes a essas atividades são colocados na lista correspondente ao seu estado (INVENTTI, 2018).

As metodologias *Scrum* e *Kanban* geralmente são utilizadas juntas, inserindo-se as atividades de cada *Sprint* em um quadro *Kanban* (INVENTTI, 2018). Os cartões *Kanban* levam informações sobre as atividades, como a descrição, o responsável, o tempo estimado, a prioridade, o tipo da atividade, entre outras informações (ESPINHA, c2020).

Figura 2.1 – Exemplo de um quadro Kanban.




Fonte: Autor

## 2.2 Tecnologias para *Front-end*

A parte da aplicação (sistema de software) em que o usuário pode interagir, geralmente por meio de interfaces gráficas, é denominada *Front-end* (SOUTO, 2019). Nesta seção, são apresentadas as tecnologias de *Front-end* denominadas HTML, CSS, *JavaScript* e *Vue.js*, destacando-se como elas foram utilizadas ao longo estágio supervisionado.

### 2.2.1 HTML

O HTML (*HyperText Markup Language*) é uma linguagem de marcação de hipertexto que permite a criação de documentos que podem ser lidos em um computador e transmitidos pela *Internet*. Nesses documentos, a linguagem HTML tem a função de inserir e estruturar seções, parágrafos, cabeçalhos, imagens, *hyperlinks*, entre outros (XAVIER, 2019).

A definição desses conteúdos é feita a partir de *tags*, que são blocos de construções das páginas, definidos por palavras reservadas entre os símbolos ’<’ e ’>’ (Figura 2.2). A maioria das *tags* HTML são compostas por uma estrutura

de abertura e fechamento, como por exemplo <*tag*> e <*/tag*>. Há também *tags* com apenas uma estrutura, como por exemplo <*br/*>, que descreve uma quebra de linha na página *Web* (MARQUES, 2020a).

As *tags* possuem atributos que definem alguns de seus comportamentos. Esses atributos possuem um nome e, possivelmente, um valor. O atributo *src* apresentado na Figura 2.2 indica o caminho da imagem para a *tag* <*img*>. Existem diversos outros atributos, como por exemplo o *id*, que define um identificador para a *tag*, entre outros (MARQUES, 2020b).

Figura 2.2 – Exemplo de código HTML.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Página de exemplo</title>
5  </head>
6  <body>
7      <h1>Titulo</h1>
8
9      <p>Aqui podemos inserir um parágrafo</p>
10
11     <img src="imagem.jpg" alt="uma imagem qualquer">
12
13 </body>
14 </html>
```

Fonte: Autor

A estrutura básica de um documento HTML começa com a *tag* <*!DOCTYPE*>, que indica ao navegador que se trata de um arquivo do tipo HTML. Em seguida, tem-se o par de *tags* <*html*> e <*/html*>, entre as quais todas as demais *tags* são inseridas, como por exemplo: (i) a <*head*>, que contém informações sobre a página, como o título que aparecerá na aba do navegador, vínculos para arquivos externos, entre outros; e (ii) a <*body*>, que contém os elementos visuais da página HTML (XAVIER, 2019).

A linguagem HTML foi empregada neste estágio para a inserção de elementos de interação com o usuário das aplicações citadas na Seção 1.3. Dentre esses elementos, pode-se citar tabelas, formulários, botões, parágrafos, títulos, imagens, menus, calendários e rodapés.

### 2.2.2 CSS

CSS (*Cascading Style Sheet*) é um recurso utilizado para estilizar elementos escritos em uma linguagem de marcação, como o HTML. Com o CSS, é possível realizar a decoração das páginas, alterando características visuais dos elementos, tais como cor, posicionamento, alinhamento, tamanho, margens, entre outros.

Figura 2.3 – Exemplo de código HTML com CSS embutido.

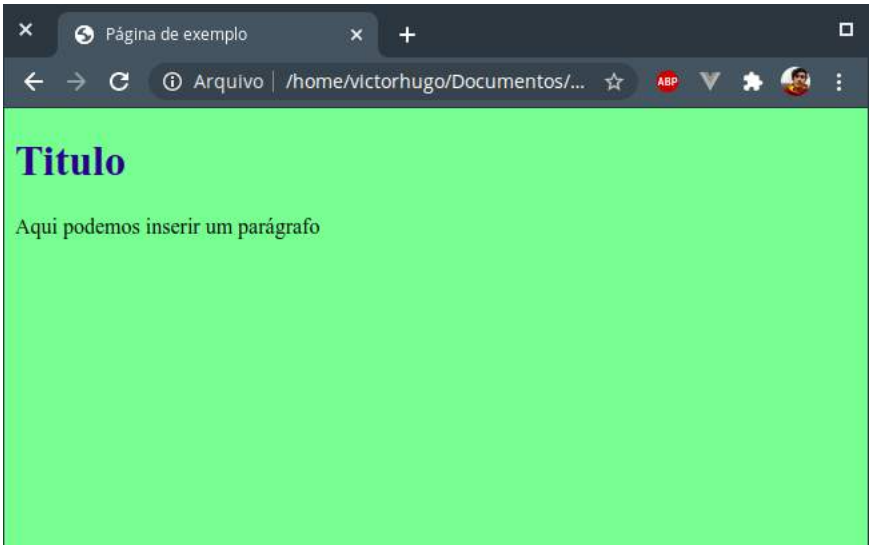
```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>Página de exemplo</title>
6      <style>
7          body{
8              background-color: lightGreen;
9          }
10         h1{
11             color: darkblue;
12         }
13     </style>
14 </head>
15 <body>
16     <h1>Titulo</h1>
17
18     <p>Aqui podemos inserir um parágrafo</p>
19
20 </body>
21 </html>
```

Fonte: Autor

A estrutura do CSS é bem simples, na qual tem-se um seletor, que seleciona um elemento do HTML, e um bloco de declaração, que declara o que será estilizado para esse elemento (GONÇALVES, 2019). Na Figura 2.3, é possível ver que, dentro da *tag* <*style*>, é declarada a estilização das *tags* <*body*> e <*h1*>. No bloco de declaração são inseridas propriedades e valores para elas.

A Figura 2.4 ilustra a página *Web* correspondente ao arquivo HMTL da Figura 2.3. Na linha 8 é estilizado o plano de fundo (*background-color*) de toda a página (<*body*>) para a cor *lightGreen*, que é uma tonalidade de verde claro. Na

Figura 2.4 – Exemplo de página *Web* com formatação via CSS.




Fonte: Autor

linha 11 é estilizada a cor (*color*) da fonte do texto da *tag* <*h1*> para *darkblue*, que é uma tonalidade de azul escuro.

Para os projetos em que o estagiário atuou, CSS foi utilizado para estilizar todas as páginas HTML, configurando cores, tamanhos, posicionamento, fonte, margens e alinhamento.

### 2.2.3 JavaScript

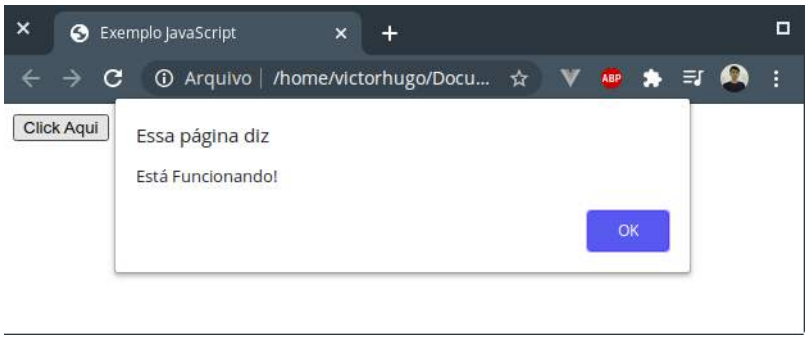
Figura 2.5 – Exemplo de código *JavaScript* integrado ao HTML.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Exemplo JavaScript</title>
5          <script type="text/javascript">
6              function exemploDeFuncao(){
7                  alert("Está Funcionando!");
8              }
9          </script>
10     </head>
11     <body>
12         <input type="button" onclick="exemploDeFuncao()" value="Click Aqui">
13     </body>
14 </html>
```

Fonte: Autor

*JavaScript* é uma linguagem de programação criada em 1995 e tem como propósito oferecer aos desenvolvedores formas de tornar as páginas *Web* mais dinâmicas. Essa linguagem, que é costumeiramente abreviada como JS, tem como principal característica ser executada do lado do cliente, ou seja, nos navegadores dos usuários, ao invés de ser executada em servidores remotos. O JS permite a transformação e o processamento de dados, interagindo com o conteúdo da linguagem HTML e com a estilização proporcionada pelo CSS nessas páginas (SILVA, 2015).

Figura 2.6 – Exemplo de página *Web* que faz uso de código *JavaScript*.




Fonte: Autor

Essa linguagem pode ser inserida diretamente em uma página *Web*, utilizando as *tags* <*script*> e </*script*>, como visto na Figura 2.5. A figura apresenta um exemplo de uma função que exibe um alerta com a mensagem *"Está funcionando"* quando o usuário clicar sobre o botão disponível na página HTML. O resultado desse comportamento pode ser visto na Figura 2.6.

A linguagem JS foi utilizada no estágio para implementar diversas funcionalidades. Uma dessas funcionalidades é a de processar os dados que são trazidos do servidor para serem apresentados na tela do usuário e os dados que são obtidos como entradas do usuário, que posteriormente são enviados para o servidor. Outras funcionalidades implementadas com JS foram o controle do fluxo e dos eventos das páginas, a validação de campos em formulários e a realização de requisições para se comunicar com outros componentes.

### 2.2.4 Vue.js

Segundo Galdino (2017), *Vue.js*, também conhecido como *Vue*, é um *framework JavaScript* de código aberto para a construção de interfaces de usuário.

Um *framework* é uma estrutura-base que contém um conjunto de funções e componentes pré-definidos, os quais se relacionam para disponibilizar funcionalidades específicas ao desenvolvimento de software. Estas funções e componentes genéricos p̈ré-prontosägilizam o processo de desenvolvimento, poupando tempo e evitando retrabalho por parte dos desenvolvedores (GUEDES, 2019).

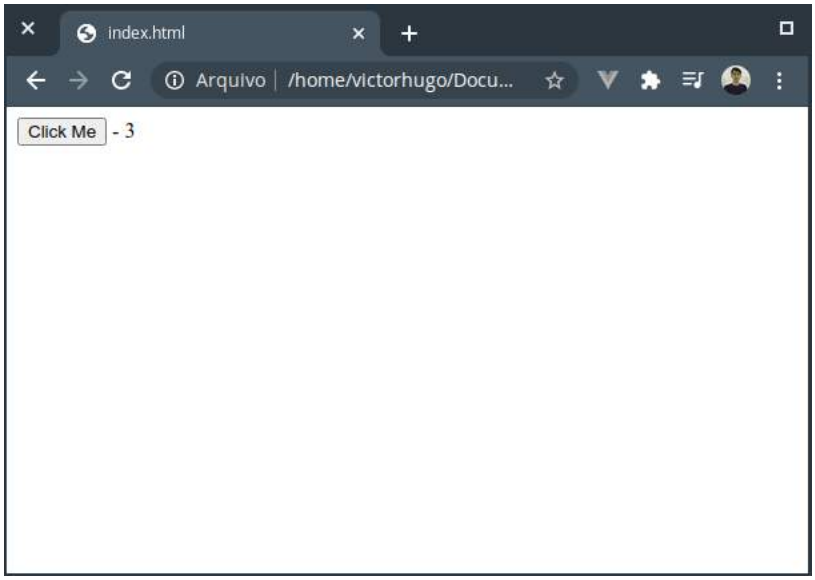
As aplicações que utilizam esse *framework* são constituídas de um conjunto de componentes criados com as linguagens HTML, CSS e *Javascript* em um único arquivo *.vue*. Cada componente possui um escopo isolado, tanto em lógica quanto em estilização, o que facilita a manutenção desses componentes (PICOLLO, 2020).

Figura 2.7 – Exemplo de código com *Vue.js*.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <div id="app">
        <div>
            <button @click="upCounter"> Click Me </button> - {{counter}}
        </div>
    </div>

    <script src="https://unpkg.com/vue/dist/vue.js"></script>
    <script>

        var app = new Vue({
            el: '#app',
            data: function(){
                return {counter:0}
            },
            methods:{
                upCounter: function(){
                    this.counter++;
                }
            }
        })

    </script>
</body>
</html>
```

Fonte: Autor

Figura 2.8 – Exemplo de página *Web* criada com *Vue.js*.




Fonte: Autor

No exemplo da Figura 2.7, tem-se um código em Vue que apresenta um botão e um valor numérico, o qual é incrementado quando o botão é pressionado. O resultado deste código é apresentado na Figura 2.8.

Nesse exemplo, para inserir o *Vue* em uma pagina HTML foi utilizado o código da linha 13, por meio do qual é importado o *Vue*, e o código da linha 14 e 16, que insere uma instância do *Vue* na página *Web*. O valor impresso na tela do usuário está armazenado na variável *counter*, definido na linha 19. Essa variável se encontra dentro do escopo *data*, que, no *Vue*, é o escopo no qual são definidas as variáveis da página. O valor de *counter* é incrementado quando o botão definido na linha 9 é pressionado. Ao clicar sobre este botão, o método *upCounter*, que contém uma função que incrementa o valor de *counter*, é executado. Este método se encontra dentro do escopo *methods*, no qual são inseridos as métodos da página. Para embutir o valor de uma variável do *Vue* no HTML, é utilizado o nome da variável dentro de chaves duplas, como na linha 9.

O *framework Vue* foi utilizado no estágio para as atividades de *Front-end* em cada um dos projetos em que o estagiário atuou.

## 2.3 Tecnologias para *Back-end*

*Back-end* é a parte da aplicação responsável pela integração dos dados que vêm do *Front-end* com o banco de dados, e também o caminho oposto. Além disso, é no *Back-end* que se aplica as devidas regras de negócio e validações (SOUTO, 2019). Nesta seção, são apresentadas as tecnologias e ferramentas de *Back-end* utilizadas no estágio, tais como *Java*, *Spring Boot* e *Postgresql*.

### 2.3.1 Bootstrap

Segundo Marques (2021), Bootstrap é um *framework front-end* utilizado por milhares de desenvolvedores web ao redor do mundo. Com ele, muitas etapas do desenvolvimentos web se tornam mais rápidas e dinâmicas, pois o *framework* traz consigo diversos elementos prontos e estilizados. Além disso, trata-se de uma ferramenta gratuita para desenvolvimento com HTML, CSS e JS. Os padrões do *Bootstrap* seguem os princípios de usabilidade e as tendências de *design* para interfaces (MARQUES, 2021)

Segundo Longen (2020), Bootstrap poupa tempo dos desenvolvedores para desenvolver componentes de interface, mas seu principal objetivo é a criação de *websites* responsivos. Ele permite, portanto, que a interface do usuário de um *website* seja otimizada para qualquer tamanho de tela, desde dispositivos móveis até telas maiores, de computadores de mesa. Com isso, os desenvolvedores não precisam se preocupar em fazer muitas versões de um mesmo *website* para diferentes tipos e tamanhos de telas.

Figura 2.9 – Exemplo de código em *Java*.

```java
public class Carro{
    // Atributos
    private String cor;
    private String placa;
    private String modelo;

    /* CONSTRUTOR PADRÃO */
    public Carro(){

    }

    /* CONSTRUTOR COM 3 PARÂMETROS */
    public Carro(String cor, String modelo, double placa){
        this.cor = cor;
        this.modelo = modelo;
        this.placa = placa;
    }
    // metodos para atribuir valores aos atributos
    void setCor(String cor){
        this.cor = cor;
    }
    void setModelo(String modelo){
        this.modelo = modelo;
    }
    void setPlaca(String placa){
        this.placa = placa;
    }
    // metodos para consultar os valores dos atributos
    String getCor(){
        return this.cor;
    }
    String getPlaca(){
        return this.placa;
    }
    String getModelo(){
        return this.modelo;
    }
}
```

Fonte: Autor

### 2.3.2 Linguagem de Programação *Java*

A Figura 2.9 apresenta um exemplo de código de uma classe nessa linguagem, que é utilizada para modelar um objeto carro. Essa classe contém os atributos cor, placa e modelo do carro. Além disso, ela contém métodos para consultar (*get*) e alterar (*set*) os valores destes atributos, além de um construtor. O construtor é um método responsável por inicializar o objeto em memória, atribuindo valores aos atributos do objeto. Os métodos *get* são usados para recuperar valores dos atributos do objeto e os métodos *set* para alterá-los.

A linguagem *Java* foi empregada no estágio para modelar as entidades específicas no *Back-end* de cada um dos projetos em que o estagiário atuou.

### 2.3.3 Spring Boot

O *Spring Boot* é uma ferramenta que tem como intuito facilitar o processo de inicialização de um projeto com o *framework Spring*, realizando processos de configuração e publicação do projeto (AFONSO, 2017).

Spring é *framework* que tem como objetivo facilitar o desenvolvimento de aplicações *Web* em *Java*. Ao adotá-lo, o desenvolvedor terá à sua disposição uma ferramenta que fornece os recursos necessários para grande parte das aplicações, como persistência de dados, segurança, testes, bem como um padrão de arquitetura a seguir, permitindo a criação de soluções menos acopladas, mais coesas e, consequentemente, mais fáceis de compreender e manter (DEVMEDIA, 2020).

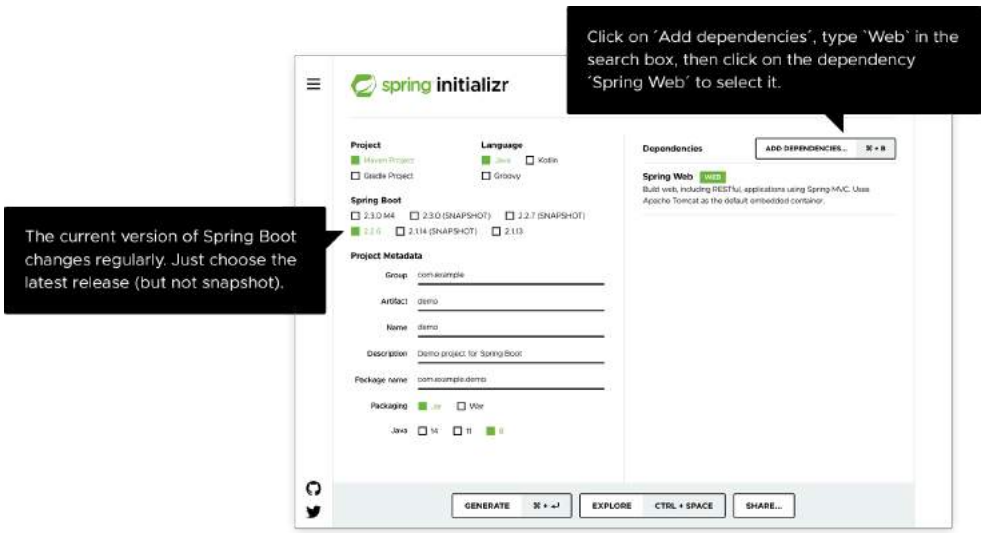
O *framework* Spring dispõe de diversos módulos responsáveis por diferentes funções. Dentre esses módulos, têm-se: (i) *Spring Data*, responsável pelas tecnologias de persistência de dados; (ii) *Spring MVC*, responsável por uma arquitetura *Web* com padrão MVC; (iii) *Spring Security*, responsável por tecnologias de segurança e autenticação; e (iv) *Spring Boot*, responsável pela criação de projetos baseados em Spring, controlando dependências, aequivos de configurações, entre outras coisas. Esses módulos podem ser utilizados de forma conjunta em uma mesma aplicação (SPRING, 2020).

Um dos principais objetivos do *Spring Boot* é gerenciar, de forma automática, as dependências de um projeto. Dessa forma, evitam-se preocupações desnecessárias com dependências e configurações, permitindo que o desenvolvedor mantenha o foco na resolução do problema para o qual o software está sendo desenvolvido (OKI, 2015).

Segundo Spring (2020), é possível iniciar facilmente um projeto com *Spring Boot* [1], indicando suas configurações e dependências com apenas alguns cliques (Figura 2.10). Algumas dessas dependências podem ser outros módulos do Spring, como o *Spring Data*, o *Spring MVC* e o *Spring Security*. Ao iniciar um projeto

[1] https://start.spring.io/

Figura 2.10 – *Website* para criação rápida de um projeto com *Spring Boot*.




Fonte: Spring (2020)

com *Spring Boot*, a ferramenta cria um arquivo principal, como pode ser visto na Figura 2.11. Na linha 10 desta figura, tem-se uma especificação do *Spring Boot* para inicializar a aplicação.

Figura 2.11 – Exemplo do arquivo principal de um projeto com *Spring Boot*.

```
1  package com.AgendaDeContatos.apirest;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class ApirestApplication {
8
9      public static void main(String[] args) {
10         SpringApplication.run(ApirestApplication.class, args);
11     }
12
13 }
14
```

Fonte: Autor

Para este estágio o *Spring Boot* foi empregado em todos os projetos citados. Nos módulos Análise Dinamizada e Retificação Automática, o *Spring Boot* já estava inserido e configurado quando o estagiário começou a trabalhar neles.

Assim, ele trabalhou realizando o controle de novas dependências inseridas nesses módulos; quanto ao sistema de Gestão de Contratos, o *Spring Boot* foi adicionado e configurado pelo estagiário, pois ele foi o responsável por criar o *Back-end* do mesmo.

### 2.3.4 PostgreSQL

Segundo Techtudo (2014), *PostgreSQL* é um Sistema Gerenciador de Banco de Dados (SGBD) poderoso e de código aberto. Esse sistema possui mais de 15 anos de desenvolvimento ativo e uma arquitetura com uma forte reputação, devido à sua estabilidade. *PostgreSQL* é uma ótima opção de banco de dados, inclusive para aplicações com grande volume de dados (TECHTUDO, 2014).

De acordo com Group (2006), *PostgreSQL* suporta grande parte do padrão SQL e oferece muitas funcionalidades, tais como chaves estrangeiras, visões, integridade transacional, entre outros. Além disso, ele pode ser ampliado pelo usuário de muitas maneiras como, por exemplo, adicionando novos tipos de dados, funções e operadores. Devido sua licença pouco restritiva, *PostgreSQL* pode ser utilizado, modificado e distribuído por qualquer pessoa para qualquer finalidade, seja particular, acadêmica ou comercial. SQL é uma linguagem padrão para trabalhar com bancos de dados relacionais. A sigla SQL significa *Standard Query Language*, literalmente a linguagem padrão para consultas (SILVEIRA, 2019).

Figura 2.12 – Exemplo de criação de uma tabela com SQL.

```
1  CREATE TABLE carro (
2      placa VARCHAR(7) PRIMARY KEY,
3      cor VARCHAR(20),
4      modelo VARCHAR(20),
5      numPortas INT
6  );
7
```

Fonte: Autor

Na Figura 2.12 é exemplificada a criação de uma tabela com a linguagem SQL. Nesse exemplo, cria-se uma tabela denominada carro, com as colunas placa,

cor, modelo e numPortas (número de portas). O SGBD *PostgreSQL*, bem como a linguagem SQL, foram empregados no estágio para manipular os bancos de dados de cada um dos projetos citados na introdução deste relatório. Em cada um dos projetos, com exceção do projeto de Gestão de Contratos, o estagiário somente manipulou dados e criou algumas tabelas, pois grande parte do banco de dados já estava desenvolvido.

## 2.4 Tecnologias de Comunicação

Esta seção aborda os conceitos de comunicação entre diferentes módulos de uma aplicação, contemplando as seguintes tecnologias e conceitos: JSON, Requisições *Web* e *API*.

### 2.4.1 JSON

Segundo Gonçalves (2012), JSON (*JavaScript Object Notation*) é um modelo para armazenamento e transmissão de informações no formato texto. JSON tem sido empregado principalmente em aplicações *Web*, devido à sua capacidade de estruturar informações de uma forma bem compacta.

A sintaxe do JSON é relativamente simples, na qual para cada valor representado atribui-se um nome ou rótulo, que tem o objetivo de descrever seu significado. Essa sintaxe é derivada da forma utilizada pelo *JavaScript* para representar informações. Os valores podem possuir apenas 3 tipos básicos: numérico (inteiro ou real), booleano e string. A partir de tipos básico é possível representar tipos complexos, como um *array*, que é delimitado por colchetes e objetos, que são compostos de pares de nome e valor, separados por vírgula e delimitados por chaves (GONÇALVES, 2012).

A Figura 2.13 apresenta um exemplo de uma lista de objetos no formato JSON. Esses objetos modelam 3 (três) produtos com seus atributos: id (identificador), nome, valor e quantidade. São utilizadas vírgulas para separar cada par de

Figura 2.13 – Exemplo de uma lista de objetos no formato JSON.

```
[
  {
    "id": 1,
    "nome": "Geladeira",
    "valor": 3000.99,
    "quantidade": 1500
  },
  {
    "id": 2,
    "nome": "Fogão",
    "valor": 1300.99,
    "quantidade": 5100
  },
  {
    "id": 3,
    "nome": "Microondas",
    "valor": 300.0,
    "quantidade": 700
  }

]

```

Fonte: Autor

atributo e valor e para separar cada produto. Cada um dos produtos está delimitado entre chaves e o nome de cada atributo é delimitado por aspas duplas. Como se trata de uma lista (*array*) de produtos, esses estão delimitados por colchetes.

O JSON foi empregado no estágio em cada um dos projetos como modelo de transmissão de dados entre *Front-end* e *Back-end*.

### 2.4.2 Requisições *Web*

O protocolo base para a troca de dados na *Web* é o HTTP (*HyperText Transfer Protocol*). Esse protocolo tem como base o modelo de comunicação cliente-servidor, o que significa que solicitações são iniciadas por um cliente e atendidas por um servidor. Dessa forma, clientes e servidores se comunicam trocando mensagens individuais, em que as mensagens enviadas pelo cliente são chamadas de requisições (*requests*) e as mensagens enviadas pelo servidor como resposta são chamadas de respostas (*responses*). O conteúdo de uma resposta pode

ser um código no formato HTML, XML ou JSON, figuras, textos, entre outros (ZAVADNIAK, 2017).

O protocolo HTTP define ainda um conjunto de métodos para requisições, chamados de *Request Methods*, que são responsáveis por informar as ações a serem executadas em relação a um determinado recurso. Cada um deles implementa uma semântica diferente e, dentre esses métodos, pode-se citar *GET, POST, PUT e DELETE* (MDN, 2019).

Figura 2.14 – Exemplo de uma requisição *Web* utilizando o método *GET*.


Fonte: Autor

Segundo Felipe (2016), um *Request Method* indica quais intenções o cliente tem ao fazer uma requisição ao servidor. Ao realizar uma requisição para buscar uma página ou alguma informação no servidor, o método utilizado é o *GET*. Para enviar dados ao servidor, de forma a armazená-los em uma base dados, utiliza-se o *POST*. O método *DELETE* tem a função de solicitar ao servidor a exclusão de determinadas informações. O método *PUT* é responsável por solicitar ao servidor a alteração de informações já existentes. Estes princípios fazem parte da arquitetura conhecida como REST.

REST, do inglês *Representational State Transfer*, trata-se de um padrão arquitetural, que consiste em princípios e regras para permitir a criação de um

projeto com interfaces de comunicação bem definidas, facilitado a comunicação entre aplicações (LEITE, 2020).

Figura 2.15 – Exemplo de um JSON de resposta à requisição *Web* da Figura 2.14 com dados fictícios.


```
[
  {
    "id": 80,
    "nome": "João Tavares",
    "email": "joao_tavares39129043@example.com",
    "telefone": "3912-9034",
    "rua": "Rua Paulino Constante",
    "complemento": "Casa",
    "numero": 31,
    "cep": "37200008",
    "bairro": "Centro",
    "cidade": "Lavras",
    "estado": "MG"
  }
]
```


Fonte: Autor

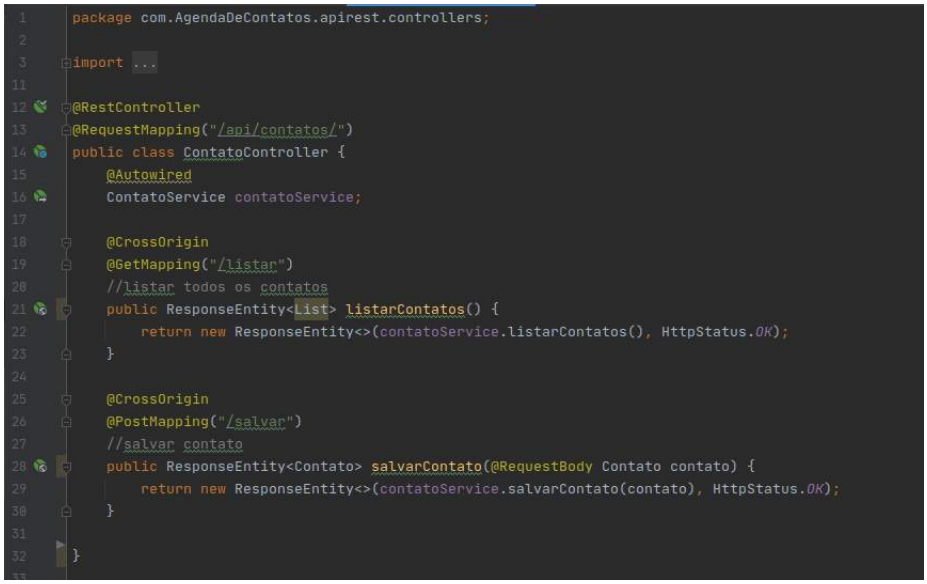
A Figura 2.14 apresenta o exemplo de uma requisição utilizando o método *GET*. Essa requisição está solicitando a listagem de um recurso denominado "contato". O item *Request URL* indica o endereço do servidor ao qual está se fazendo essa requisição. A resposta do servidor é exemplificada na Figura 2.15.

As requisições *Web* via protocolo HTTP foram utilizadas neste trabalho para realizar a comunicação entre diversos módulos de cada um dos projetos citados anteriormente, realizando principalmente a comunicação do *Front-end* (cliente) com o *Back-end* (servidor).

A sigla *API* é derivada da expressão inglesa *Application Programming Interface*. Uma *API* é um conjunto de padrões que faz parte de uma interface, permitindo a criação de plataformas de maneira simples, prática e independentes entre si (FABRO, 2020). Segundo Fabro (2020), utilizando uma *API* é possível criar aplicações diversas, capazes de se comunicar com outras aplicações, como por exemplo uma arquitetura cliente-servidor. Segundo Leite (2020), uma *API*

que segue os padrões definidos pela arquitetura *REST* é chamada de *API REST* ou *REST API*.

Figura 2.16 – Exemplo de um trecho de código responsável por retornar a resposta da requisição da Figura 2.14 .



```
1  package com.AgendaDeContatos.apirest.controllers;
2
3  import ...
11
12 @RestController
13 @RequestMapping("/api/contatos/")
14 public class ContatoController {
15     @Autowired
16     ContatoService contatoService;
17
18     @CrossOrigin
19     @GetMapping("/listar")
20     //listar todos os contatos
21     public ResponseEntity<List> listarContatos() {
22         return new ResponseEntity<>(contatoService.listarContatos(), HttpStatus.OK);
23     }
24
25     @CrossOrigin
26     @PostMapping("/salvar")
27     //salvar contato
28     public ResponseEntity<Contato> salvarContato(@RequestBody Contato contato) {
29         return new ResponseEntity<>(contatoService.salvarContato(contato), HttpStatus.OK);
30     }
31
32 }
```

Fonte: Autor

A Figura 2.16 apresenta o trecho de código de uma *API REST*, responsável por realizar o recebimento de requisições. Na linha 13, é definido parte do endereço, que é concatenado com o restante do endereço na linha 19. Na linha 19, o trecho *GetMapping* define que o endereço somado ao método *GET* executará a função da linha 21. Essa função é responsável por chamar outros módulos da aplicação para trazer todos os contratos persistidos e retorná-los como resposta a requisição.

O conceito de *API* e *API REST* foram utilizados neste estágio para fornecer serviços de uma aplicação e consumi-los em outra.

## 2.5 Git

Nesta seção são abordados, sucintamente, os principais conceitos sobre tecnologias de versionamento de arquivos, contemplando a ferramenta GIT.

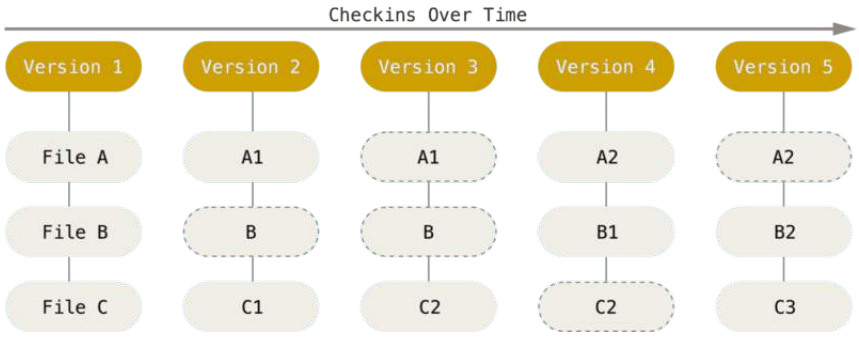
Sistemas de controle de versão são uma categoria de ferramentas de software que ajudam equipes de software a gerenciar alterações no código-fonte. Um software de controle de versão mantém registro de todas as modificações no código-fonte em um tipo especial de base de dados. Dessa forma, se um erro for cometido, os desenvolvedores podem voltar no tempo e comparar as diferentes versões do código. Além disso, esse tipo de software ajuda a rastrear cada alteração individual feita por cada membro da equipe, minimizando os conflitos causados pelo trabalho simultâneo (ATLASSIAN, 2020).

Segundo Gonçalves (2020), o Git é um dos mais conhecidos sistemas de controle de versão de código aberto do mundo. Desenvolvido em 2005 por Linus Torvalds, o Git é um projeto de código aberto maduro e com manutenção ativa. Por meio do GIT, é possível desenvolver projetos nos quais diversas pessoas podem contribuir simultaneamente, alterando e criando arquivos, minimizando o risco de suas alterações serem sobrescritas (SCHMITZ, 2015).

Em um projeto cujas versões são controladas com Git, tem-se um repositório em que se encontram os arquivos e pastas do projeto e o arquivo *.git*, em que todas as informações sobre cada arquivo e modificação são armazenados. No Git, todas as operações são atômicas. Com isso, um repositório não apresentará um estado de instabilidade ou corrupção, caso ocorra algum problema em uma ação. Além disso, esse software oferece aos desenvolvedores o recurso de ramificação, em que o usuário pode criar ramos (*branches*) de código independentes e depois mesclá-los ao ramo principal do projeto (GONÇALVES, 2020).

A Figura 2.17 ilustra o controle de versão em alguns arquivos denominados A, B e C ao longo do tempo, em que cada coluna representa uma versão. Em um primeiro momento, têm-se os arquivos em sua primeira versão. No segundo

Figura 2.17 – Exemplo de controle de versão de arquivos com Git.




Fonte: Chacon e Straub (2014)

momento, os arquivos A e C foram alterados e o arquivo B permanece da mesma forma. Em outro momento, C permanece com a última alteração, A é alterado novamente e B sofre sua primeira alteração. Na última coluna, B e C são alterados novamente.

O Git foi utilizado para gerenciar o versionamento de cada um dos projetos citados anteriormente, facilitando o trabalho colaborativo e simultâneo entre os membros da equipe de desenvolvimento.

# 3 ATIVIDADES DESENVOLVIDAS

Neste capítulo são descritas algumas das atividades desenvolvidas durante o estágio, lições aprendidas e dificuldades encontradas. As atividades são divididas em três projetos, sendo eles o sistema de Gestão de Contratos, o módulo de Análise Dinamizada do SICAR e o módulo de Retificação Automática do SICAR.

## 3.1 Atividades gerenciais

Para possibilitar o desenvolvimento das atividades dentro de uma equipe, foi adotado o *Scrum* como metodologia gerencial. Os ritos do *Scrum*, citados em 2.1, foram utilizados para distribuir o conhecimento sobre as atividades entre os membros da equipe. Dentro do *Scrum*, foi utilizado o quadro *Kanban*, para que fosse possível acompanhar o estado de cada atividade da *Sprint*, além de quais membros as estão realizando.

Devido ao trabalho remoto, as reuniões, que fazem parte dos ritos do *Scrum*, foram realizadas por meio de salas virtuais de vídeo-chamada. Dessa forma, foi possível realizar o trabalho em equipe de maneira satisfatória, mesmo com os membros estando em ambientes diferentes.

## 3.2 Atividades desenvolvidas no sistema de Gestão de Contratos

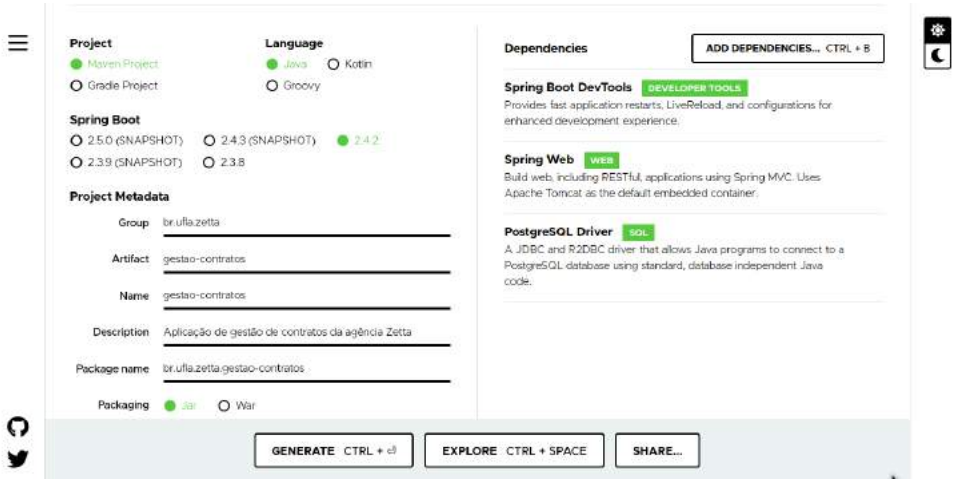
Nesta seção são descritas as atividades que foram desenvolvidas no sistema Gestão de Contratos, além das tecnologias utilizadas. O sistema de Gestão de Contratos é responsável pela gestão de contratos da agência, permitindo realizar cadastro, consulta, exclusão e edição de contratos.

### 3.2.1 Criação do *Back-end* e estrutura inicial do sistema

Como o estagiário participou do início desse projeto, uma de suas atividades foi a configuração inicial do *Back-end* do sistema. Para isso, foi utilizado o *framework Spring Boot*.

Na Figura 3.1, é ilustrada a configuração do *Back-end* deste projeto, utilizando o *Spring Boot*. Dentre as dependências adicionadas estão o *Java* em sua versão 11; o *PostgreSQL Driver*, para adicionar suporte ao sistema gerenciador de banco de dados *PostgreSQL*; e o *Spring Web*, para se trabalhar com o *framework* MVC para desenvolvimento de aplicações *Web*.

Figura 3.1 – Tela de configuração do sistema Gestão de Contratos com *Spring Boot*.

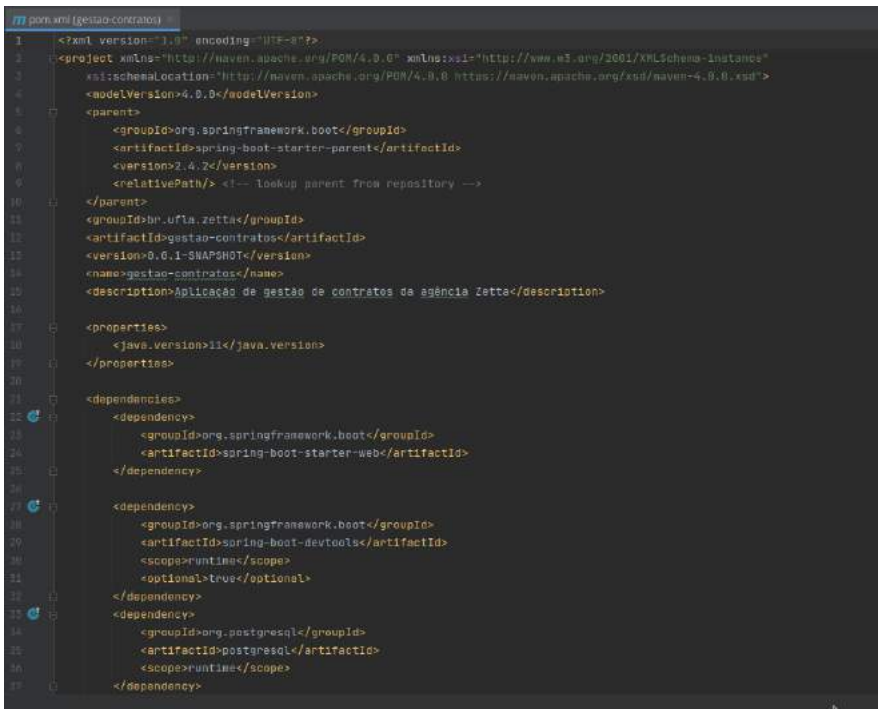



Fonte: Autor

A Figura 3.2 apresenta o arquivo *pom.xml*, que é o arquivo responsável por armazenar e estruturar as informações importantes sobre o projeto. Algumas dessas informações são as dependências, os detalhes de configurações, entre outras. Por exemplo, entre as linhas 17 e 19 é definida a versão 11 do *Java* e entre as linhas 33 e 37 é adicionada a dependência para o *PostgreSQL*.

Além disso, nesta atividade foi criada a estrutura inicial do *Back-end* do projeto. Essa estrutura, mostrada na Figura 3.3, está organizada em pacotes, os

Figura 3.2 – Arquivo *pom.xml* do *Back-end* do sistema Gestão de Contratos.



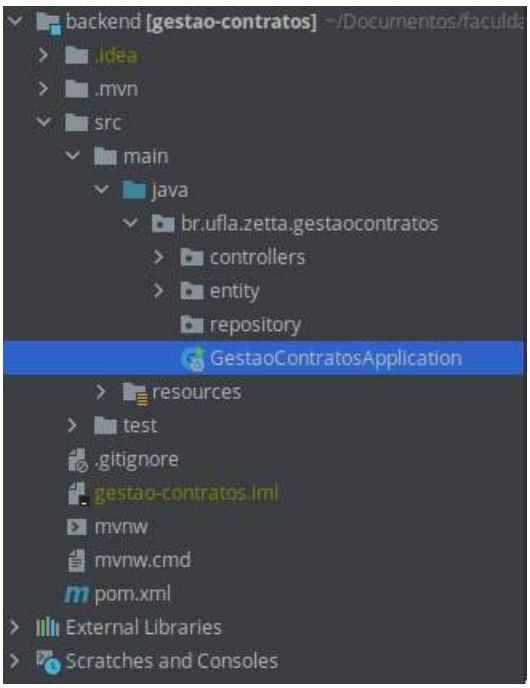

Fonte: Autor

quais são *controllers*, *entity* e *repository*. Os arquivos contidos no pacote *controllers* têm a função de receber e atender às requisições feitas a partir do *Front-end*. O pacote *entity* armazena arquivos que possuem a função de modelar os objetos da aplicação, como por exemplo um *Contrato*. Por último, os arquivos do pacote *repository* são responsáveis pela comunicação do sistema *Web* com o banco de dados.

### 3.2.2 Implementação do Menu Lateral do sistema

As Figuras 3.4 e 3.5 apresentam o menu do sistema, que é um menu expansível. Na Figura 3.4 pode-se perceber que o menu está retraído e na Figura 3.5, que ele está expandido. Esse menu é expandido ao posicionar o ponteiro do mouse sobre ele e retraído ao afastar o mesmo.

Figura 3.3 – Estrutura inicial do *Back-end* do sistema Gestão de Contratos.



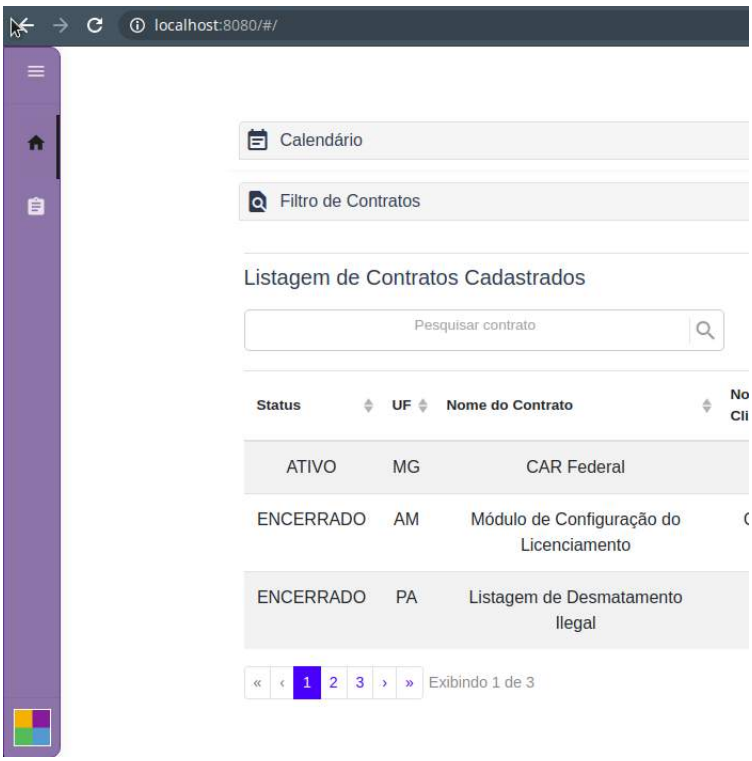

Fonte: Autor

Nesse menu têm-se o título do sistema em sua parte superior, a logo da Agência Zetta na parte inferior e duas opções: *Tela Inicial* e *Cadastrar Contrato*, posicionadas abaixo do título. Os textos são mostrados quando o menu está expandido; quando o mesmo está retraído, apenas somente os ícones são apresentados.

Esse menu permite que o usuário navegue entre as páginas ao clicar em suas opções. A opção selecionada é destacada com a fonte e ícone na cor preta. Nas Figuras 3.4 e 3.5 pode-se perceber esse efeito na opção da *Tela Inicial*, que está selecionada.

Para a criação do menu lateral foi utilizado o *framework Vue*, juntamente com o HTML, utilizado para inserir os textos e ícones, e o CSS, utilizado para estilizar o menu, inserindo cor, tamanho de fonte, posicionamento dos itens, ente outros. Por meio do *Vue*, foi possível inserir a animação de expandir e retrair o menu ao passar o mouse.

Figura 3.4 – Menu lateral retraído do sistema Gestão de Contratos.



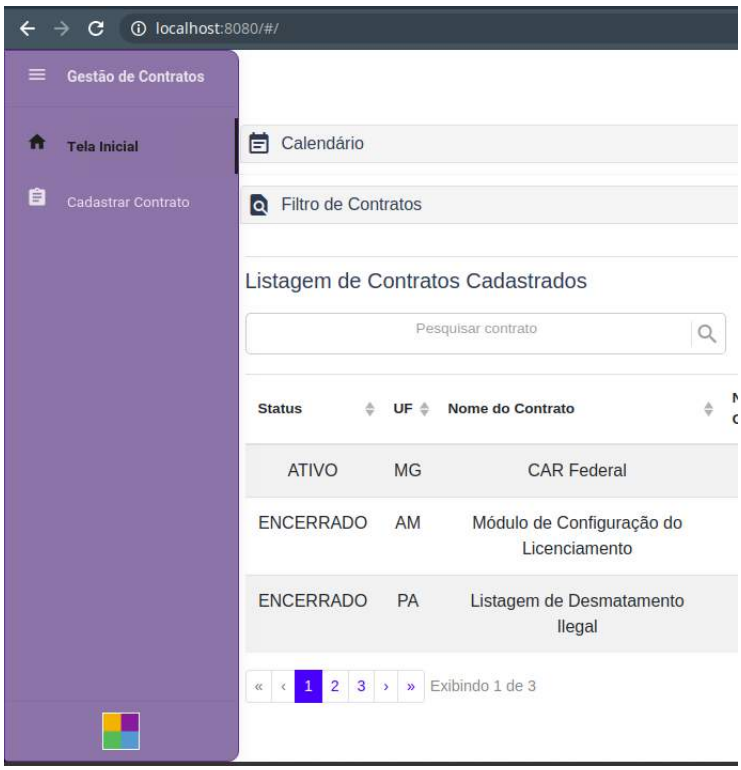

Fonte: Autor

Entre os desafios encontrados nesta atividade, pode-se citar as estilizações de (i) posicionar o menu no canto esquerdo; (ii) posicionar o menu ocupando toda a altura da página; (iii) animar o menu para retrair e expandir de modo a apresentar diferentes informações, como citado anteriormente; e (iv) indicar a página atual utilizando cor diferente no respectivo item.

### 3.2.3 Implementação de uma tabela para exibição dos contratos

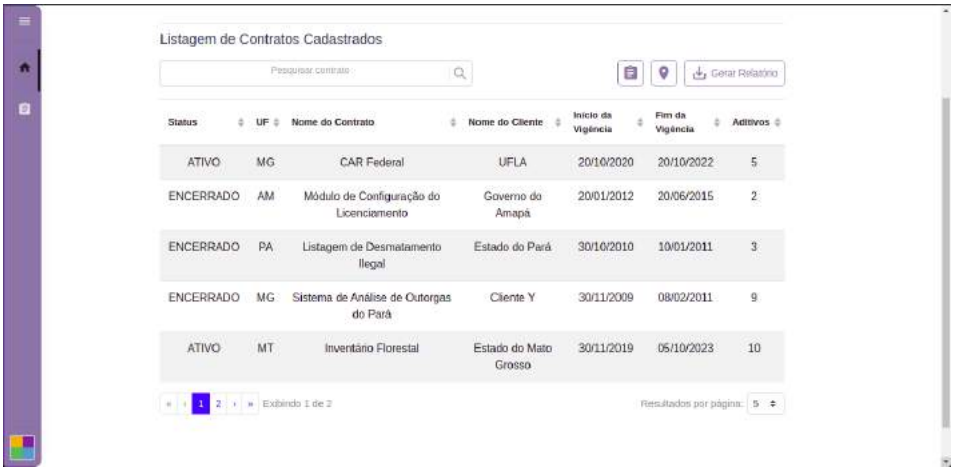
Para apresentar os contratos do sistema, foi projetada uma tabela com as informações dos contratos. Nesta tabela (ver Figura 3.6), é apresentada uma lista de contratos trazida do *Back-end* da aplicação, por meio de uma requisição HTTP.

Figura 3.5 – Menu lateral expandido do sistema Gestão de Contratos.




Fonte: Autor

Figura 3.6 – Tabela de listagem de contratos do sistema de Gestão de Contratos (os dados são fictícios).




Fonte: Autor

Nessa tabela, tem-se o título de cada coluna em seu topo e em negrito. Além disso, abaixo da tabela, têm-se algumas informações e opções sobre paginação. Da esquerda para a direita, na parte inferior da tabela, como mostrado nas Figura 3.6, têm-se um conjunto de botões para navegar entre as páginas, um texto informando o número da pagina atual e quantas páginas existem e um seletor para a escolha de quantos itens serão mostrados por página. Além disso, na tabela, tem-se a opção de ordenar os contratos, de forma crescente ou decrescente. Essa ordenação pode ser feita por meio de qualquer uma das colunas, apenas clicando nos cursores do lado direito de cada título de coluna.

Os dados dos contratos são buscados por meio de uma requisição HTTP do tipo GET. A partir do retorno do servidor, os dados dos contratos são armazenados em um *array* da linguagem JavaScript e os dados desse *array* são impressos na tabela. A Figura 3.7 apresenta um exemplo de *array* no formato JSON, que é retornado em uma requisição para listagem de contratos.

Para a criação dessa tabela, foi utilizado o *Vue* em conjunto com o HTML e CSS. O *Vue* foi utilizado para (i) realizar a requisição *Web*; (ii) manipular o *array* de contratos, apresentando os dados formatados, como por exemplo a data; (iii) realizar a ordenação dos dados dos contratos e (iv) integrar o CSS e o HTML. O HTML foi utilizado para criar a tabela com os dados dos contratos e da paginação. O CSS foi utilizado para estilizar a tabela, posicionando os itens, aplicando cor ao fundo de cada linha da tabela e selecionando as bordas da tabela.

Entre os desafios encontrados nesta atividade, pode-se citar (i) a realização da paginação; (ii) a formatação dos dados; (iii) a ordenação dos dados; e (iv) a estilização da tabela.

### 3.2.4 Implementação de tela para edição de contratos

Para alterar os dados dos contratos foi projetada uma tela de edição, na qual são mostrados os dados do contrato em um formulário. Essa tela de edição é

Figura 3.7 – Exemplo de resposta de uma requisição para listagem de contratos no formato JSON.


```
[
  {
    "id": 1,
    "nomeContrato": "CAR Federal",
    "numeroContrato": "BR23213324",
    "inicioVigencia": "2020-10-20T03:00:00.000+00:00",
    "fimVigencia": "2022-10-20T03:00:00.000+00:00",
    "estado": "MG",
    "nomeCliente": "UFLA",
    "cnpj": "974.589.720-50",
    "status": "ATIVO",
    "aditivos": 5,
    "observacao": "Aqui vai uma observação sobre o contrato do Car federal",
    "objeto": "Tem como objetivo.....",
    "valor": 10000.0
  },
  {
    "id": 8,
    "nomeContrato": "Sistema de Fauna e pesca.",
    "numeroContrato": "MG815921",
    "inicioVigencia": "2010-11-30T02:00:00.000+00:00",
    "fimVigencia": "2011-02-08T02:00:00.000+00:00",
    "estado": "MG",
    "nomeCliente": "Cliente Z",
    "cnpj": "521.252.820-89",
    "status": "ENCERRADO",
    "aditivos": 6,
    "observacao": "contrato elaborado para o sistema.....",
    "objeto": "esse contrato fornece infos sobre.....",
    "valor": 1919.02
  }
]
```

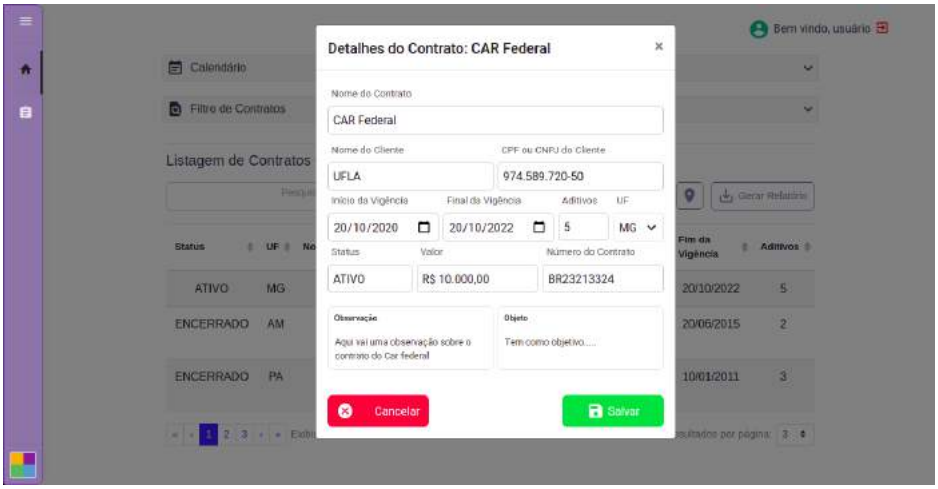

Fonte: Autor

iniciada com os dados do contrato preenchidos nos seus respectivos campos. Além disso, na parte inferior da tela de edição, têm-se um botão para cancelar a edição e um botão para salvar.

A Figura 3.8 apresenta o resultado da implementação da tela de edição. Ao clicar no botão salvar, os dados são enviados no formato JSON para o *Back-end* do sistema por meio de uma requisição HTTP do tipo PUT, conforme mostra a Figura 3.9.

Para a criação dessa tela de edição, foi utilizado o *Vue* em conjunto com o HTML, CSS e *Bootstrap*. O *Vue* foi utilizado para (i) manipular o objeto com os dados do contrato; (ii) inserir os dados do contrato no formulário; (iii) realizar a requisição HTTP, enviando os dados do contrato editado; e (iv) integrar o CSS e o HTML. O HTML foi utilizado para criar o formulário com os campos e botões, e

Figura 3.8 – Tela de edição de contratos do sistema de Gestão de Contratos (os dados são fictícios).




Fonte: Autor

Figura 3.9 – Exemplo de dados no formato JSON enviados em uma requisição para alterar um registro de contrato.


```
[
  {
    "id": 1,
    "nomeContrato": "CAR Federal",
    "numeroContrato": "BR23213324",
    "inicioVigencia": "2020-10-20T03:00:00.000+00:00",
    "fimVigencia": "2022-10-20T03:00:00.000+00:00",
    "estado": "MG",
    "nomeCliente": "UFLA",
    "cnpj": "974.589.720-50",
    "status": "ATIVO",
    "aditivos": 5,
    "observacao": "Aqui vai uma observação sobre o contrato do Car federal",
    "objeto": "Tem como objetivo.....",
    "valor": 10000.0
  }
]
```


Fonte: Autor

o CSS, utilizado para estilizar esses itens. Já o *Bootstrap* foi utilizado para criar a componente *modal*, uma pequena tela que contém o formulário e está sobreposta em destaque na Figura 3.8.

Entre as dificuldades encontradas na implementação desta atividade, pode-se citar (i) a inserção dos dados formatados nos campos do formulário, como por

exemplo a data, o valor em reais e o CNPJ, (ii) a distribuição dos campos na tela, por meio do CSS; e (iii) o envio dos dados no formato correto por meio da requisição.

## 3.3 Atividades desenvolvidas no sistema de Retificação Automática do SICAR

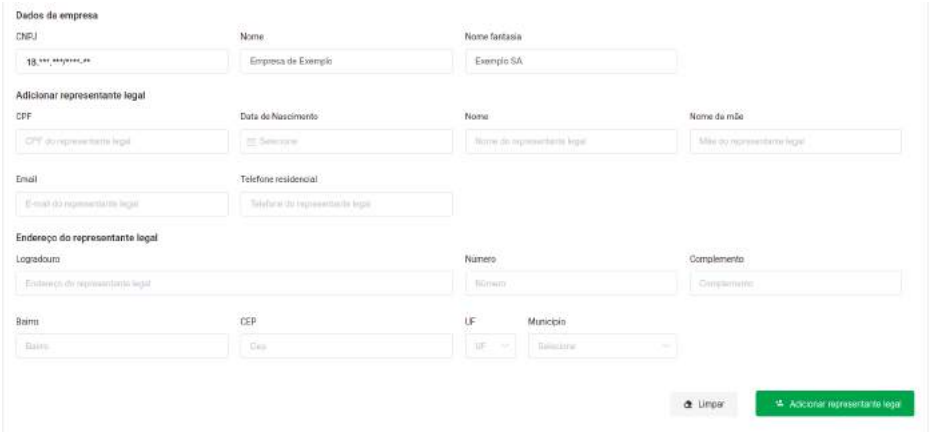
Nesta seção são apresentadas as atividades desenvolvidas no módulo de Retificação Automática do SICAR (Sistema Nacional de Cadastro Ambiental Rural). Este módulo tem como objetivo permitir ao cadastrante do imóvel rural retificar o cadastro de imóveis em que foram constatadas irregularidades. As atividades realizadas pelo estagiário neste módulo foram unicamente de manutenção.

### 3.3.1 Manutenção na tela de domínio dos imóveis

Nessa tela são cadastrados os proprietários ou possuidores do imóvel, que podem ser Pessoas Físicas (PF) e/ou Pessoas Jurídicas (PJ). Ao cadastrar pessoas jurídicas, segundo a regra de negócios da aplicação, deveria haver a opção de cadastrar um ou mais representantes legais para cada pessoa jurídica para o imóvel. A demanda que gerou essa atividade consiste na impossibilidade de se cadastrar mais de um representante legal para uma PJ. Esse problema foi identificado e reportado pelos analistas de qualidade em seus testes, que foram realizados antes de uma entrega ao cliente.

Para solucionar este problema, foram criados métodos em *Vue* para salvar os representantes legais de um imóvel em um *array*, verificando a duplicidade de números CPF, não permitindo o salvamento de representantes legais com o mesmo CPF. Os dados são, então, enviados para o servidor por meio de uma requisição. O serviço no *Back-end* responsável por atender a esta requisição já estava implementado e funcionando corretamente.

Figura 3.10 – Exemplo da tela de domínio do Retificação com o formulário de cadastro de representante legal.




Fonte: Autor

Figura 3.11 – Exemplo da tela de domínio do Retificação após o cadastro de representantes legais.


Fonte: Autor

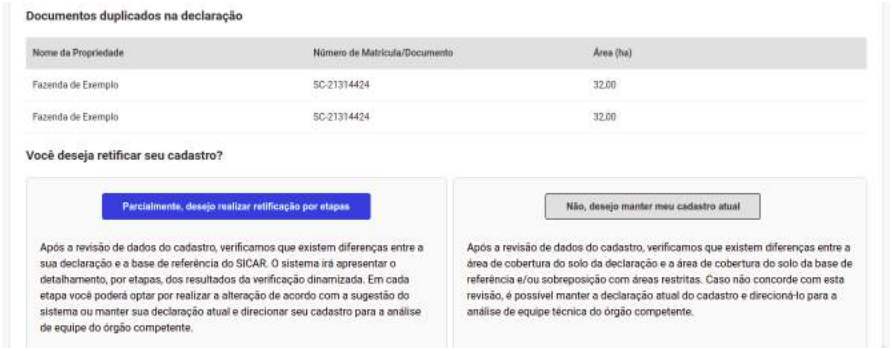
As Figuras 3.10 e 3.11 apresentam partes da tela de domínio do Retificação Automática. A Figura 3.10 apresenta o formulário de cadastro de um representante legal para uma PJ e a Figura 3.11 apresenta a tabela de representante legais de uma PJ.

Dentre os desafios encontrados nesta atividade, pode-se citar (i) o entendimento de como funciona a dinâmica dessa tela, principalmente o comportamento dos componentes visuais e dos objetos que armazenam os dados apresentados; (ii) a criação de uma função para salvar novos representantes legais, verificando a duplicidade de CPF; e (iii) o adicionamento de regras para a apresentação correta dos componentes, conforme a interação do usuário com a tela.

### 3.3.2 Criação de tabela para exibição de documentos duplicados

No módulo de Retificação Automática há uma página inicial em que são apresentadas informações sobre o imóvel para o qual está sendo realizado o cadastro. Dentre elas, têm-se informações sobre inconsistências no cadastro do imóvel em questão. A partir de uma demanda do cliente, foi solicitado que nessa página fosse apresentada uma tabela com os documentos duplicados do imóvel. Esses documentos fazem parte do cadastro do imóvel e se referem a registros e certidões dos imóveis, como por exemplo a matrícula de um imóvel junto a um cartório. A Figura 3.12 ilustra o resultado da implementação dessa tabela.

Figura 3.12 – Tabela de documentos duplicados na tela inicial do módulo de Retificação Automática do SICAR.




Fonte: Autor

Para realizar essa tarefa, foi criado um serviço no *Back-end* para receber uma requisição, realizar uma consulta no banco de dados e retornar os dados encontrados como resposta para a requisição. Já no *Front-end*, foi criado um método em *Vue* para realizar a requisição ao servidor, assim que a página é iniciada. Caso o imóvel tenha documentos duplicados, estes são apresentados em uma tabela.

Para a implementação dessa atividade, foram utilizadas as linguagens *Java*, *Vue*, *Bootstrap*, HTML e CSS. A linguagem *Java* foi utilizada para modelar o objeto com os dados dos documentos duplicados, realizar a comunicação com o banco de dados para trazer os dados e retornar esses dados por meio de uma *API*

*REST*. Já o *Vue* foi utilizado para realizar a requisição junto ao *Back-end* do sistema, recebendo como resposta o arquivo JSON com os dados dos documentos duplicados. Esses dados são armazenados e mostrados por meio de uma tabela implementada com Bootstrap, HTML e CSS.

Dentre os desafios encontrados nesta atividade, pode-se citar a criação de um serviço no *Back-end* para buscar os dados duplicados e a comunicação do *Front-end* com o *Back-end*.

## 3.4 Atividades desenvolvidas no sistema Análise Dinamizada do SICAR

Nesta seção são apresentadas as atividades desenvolvidas no módulo de Análise Dinamizada do SICAR (Sistema Nacional de Cadastro Ambiental Rural). Este módulo tem como objetivo realizar análises dos imóveis cadastrados. Essas análises envolvem verificações sobre os dados que foram declarados, como a análise da regularidade ambiental e revisões dos dados.

### 3.4.1 Manutenção na tela de resultado da análise de regularidade

No módulo de Análise Dinamizada, tem-se uma página na qual são mostradas as informações sobre o resultado da análise de regularidade ambiental. Nessa página é apresentado o mapa da área do imóvel e, ao lado direito desse mapa, uma tabela com características do imóvel. As características apresentadas nessa tabela são referentes à área do imóvel, como os tipos de área e o tamanho em hectare (ha) de cada tipo. Alguns exemplos desses tipos de área podem ser classificados quanto à cobertura do solo, à área de preservação permanente e à reserva legal.

A partir disso, por uma demanda do cliente, foi definido que, dessas características, só deveriam ser apresentadas as que fossem declaradas durante o cadastro do imóvel, ou seja, as características definidas com área superior a 0 (zero) ha.

Figura 3.13 – Tela de resultado da análise da regularidade ambiental do módulo de Análise Dinamizada do SICAR.


Fonte: Autor

A Figura 3.13 apresenta essa tela na forma como estava antes da correção, em que todas as características eram citadas, mesmo as com valor de área igual a zero ou com um traço (-), indicando que não foi definida.

Figura 3.14 – Tela de resultado da análise da regularidade ambiental do módulo de Análise Dinamizada do SICAR atualizada.


Fonte: Autor

A Figura 3.14 apresenta a tela, após a implementação dessa correção, em que apenas características com área superior a 0 (zero) ha são mostradas. Para

realizar essa correção, foram utilizadas funções no *Vue* para verificar se a área de cada umas das características foi definida e se era maior do que zero.

Dentre os desafios encontrados nessa atividade, pode-se citar a criação da verificação para ocultar os itens e a identificação de onde inserir essa verificação no código.

Análogo a essa atividade, foi desenvolvida uma correção na tela de resultado da revisão de dados. Nessa página, é apresentado um comparativo entre os dados declarados e os dados da base de referência, em relação aos dados sobre a área do imóvel. A correção nessa tela refere-se ao ocultamento dos dados de uma das características da área do imóvel. A característica a ser ocultada é a denominada 'Área Antropizada' e deve ser ocultada quando a mesma não foi declarada ou declarada como 0 (zero) ha.

A Figura 3.15 apresenta o exemplo dessa tela, antes de realizar a correção e a Figura 3.16 apresenta a tela após a correção.

Figura 3.15 – Tela de resultado da revisão de dados do módulo Análise Dinamizada do SICAR.


Fonte: Autor

Figura 3.16 – Tela de resultado da revisão de dados do módulo de Análise Dinamizada do SICAR atualizada.


Fonte: Autor

# 4 CONSIDERAÇÕES FINAIS

O estágio tem um valor financeiro mensurável, ao mesmo tempo em que esse tem um valor imensurável em relação ao aprendizado. É uma experiência engrandecedora em diversos sentidos, na qual pode-se enfatizar o crescimento profissional e pessoal. Além disso, o estágio é o primeiro passo na inserção do aluno no mercado de trabalho, agregando experiência para o currículo do mesmo.

Para possibilitar o desenvolvimento das atividades e solução dos desafios encontrados durante o período de estágio, diversas disciplinas foram importantes, as quais podem ser separadas em alguns grupos: (i) lógica e estruturas de programação, como por exemplo as disciplinas de Introdução aos Algoritmos, Estrutura de dados, Práticas de Programação Orientada a Objetos e Paradigmas de Linguagens de Programação, que foram importantes para auxiliar o estagiário a desenvolver soluções lógicas para solucionar problemas; (ii) processos de software, como por exemplo as disciplinas de Engenharia de Software, Interação Humano-Computador, Processos de Software e Gerência de Projetos de Software, que foram importantes para o desenvolvimento de sistemas utilizando boas práticas; e (iii) banco de dados, como por exemplo as disciplinas de Introdução a Banco de Dados e Sistemas Gerenciadores de Banco de Dados, que foram importantes para modelar bancos para armazenar e consultar dados dos sistemas durante o estágio.

O desenvolvimento dos sistemas mencionados anteriormente trouxe novas experiências ao aluno. Essas experiências tiveram uma grande importância no aprendizado e na aplicação dos saberes adquiridos durante a graduação, com ênfase no trabalho em equipe e soluções de problemas por meio de sistemas de software. Em virtude do que foi mencionado anteriormente, pode-se concluir que o estágio proporcionou um grande crescimento na carreira profissional do aluno. A participação em um estágio, anterior à entrada no mercado de trabalho, proporciona diversos impactos positivos no preparo prático do estagiário, aplicando

o conteúdo teórico aprendido na graduação. Além disso, o mesmo proporciona a absorção de saberes em torno de tecnologias e boas práticas utilizadas no desenvolvimento, aprimorando a qualificação profissional, técnica e interpessoal.

# REFERÊNCIAS

AFONSO, A. **O que é Spring Boot?** 2017. Disponível em: <https://blog.algaworks.com/spring-boot/>. Acesso em: 28 nov. 2020.

ATLASSIAN. **O que é controle de versão**. 2020. Disponível em: <https://www.atlassian.com/br/git/tutorials/what-is-version-control>. Acesso em: 19 dez. 2020.

CHACON, S.; STRAUB, B. **Pro git**. [S.l.]: Springer Nature, 2014.

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>. Acesso em: 27 out. 2020.

DEVMEDIA. **Como começar com Spring?** 2020. Disponível em: <https://www.devmedia.com.br/exemplo/como-comecar-com-spring/73>. Acesso em: 26 dez. 2020.

ESPINHA, R. G. **Kanban: O que é e tudo sobre como gerenciar fluxos de trabalho**. c2020. Disponível em: <https://artia.com/kanban/>. Acesso em: 11 nov. 2020.

FABRO, C. **O que é API e para que serve? Cinco perguntas e respostas**. 2020. Disponível em: <https://www.techtudo.com.br/listas/2020/06/o-que-e-api-e-para-que-serve-cinco-perguntas-e-respostas.ghtml>. Acesso em: 13 dez. 2020.

FELIPE, A. **Métodos de requisição do HTTP**. 2016. Disponível em: <https://www.alura.com.br/artigos/metodos-de-requisicao-do-http?gclid=Cj0KCQiA8dH-BRD_ARIsAC24umbQ_PEKEADGwAI-fUdJAf4NMDEKgBdtJLhtcx6zpNiN5A_lyoVxjuwaAoU0EALw_wcB>. Acesso em: 12 dez. 2020.

GALDINO, F. **Vue.js Tutorial**. 2017. Disponível em: <https://www.devmedia.com.br/vue-js-tutorial/38042>. Acesso em: 23 nov. 2020.

GONÇALVES, A. **O que é CSS? Guia Básico para Iniciantes**. 2019. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-css-guia-basico-de-css/>. Acesso em: 17 nov. 2020.

GONÇALVES, A. **GIT Tutorial Para Iniciantes**. 2020. Disponível em: <https://www.hostinger.com.br/tutoriais/tutorial-do-git-basics-introducao/>. Acesso em: 19 dez. 2020.

GONÇALVES, E. C. **JSON Tutorial**. 2012. Disponível em: <https://www.devmedia.com.br/json-tutorial/25275>. Acesso em: 12 dez. 2020.

GROUP, P. G. D. **Documentação do PostgreSQL 8.2.0**. [s.n.], 2006. Disponível em: <http://pgdocptbr.sourceforge.net/pg82/intro-whatis.html>.

GUEDES, M. **Para que serve um framework?** 2019. Disponível em: <https://www.treinaweb.com.br/blog/para-que-serve-um-framework/>. Acesso em: 27 nov. 2020.

INVENTTI. **Metodologia Ágil: Scrum X Kanban**. 2018. Disponível em: <https://inventti.com.br/scrum-x-kanban/>. Acesso em: 11 nov. 2020.

JORNAL CONTÁBIL. **Mercado de TI: Conheça a importância da profissionalização nesse setor**. 2020. Disponível em: <https://www.jornalcontabil.com.br/mercado-de-ti-conheca-a-importancia-da-profissionalizacao-nesse-setor/>. Acesso em: 22 out. 2020.

LEITE, D. **O que é REST API? Entenda mais sobre integração de sistemas**. 2020. Disponível em: <https://blog.vindi.com.br/o-que-e-rest/>. Acesso em: 26 dez. 2020.

LONGEN, A. **O Que é Bootstrap? Guia para Iniciante**. 2020. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-bootstrap>. 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: 17 nov. 2020.

MARQUES, R. **Tags HTML: as principais tags para criar sua página HTML**. 2020. Disponível em: <https://www.homehost.com.br/blog/tutoriais/tags-html/>. Acesso em: 17 nov. 2020.

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.

MDN, C. da. **Métodos de requisição HTTP**. 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods>. Acesso em: 12 dez. 2020.

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: 28 nov. 2020.

PAULA, G. D. **Tudo sobre Metodologia Scrum: o que é e como essa ferramenta pode te ajudar a poupar tempo e gerir melhor seus projetos**. 2016. Disponível em: <https://www.treasy.com.br/blog/scrum/>. Acesso em: 6 nov. 2020.

PICOLLO, L. **Vue JS: o que é, como funciona e vantagens**. 2020. Disponível em: <https://blog.geekhunter.com.br/vue-js-so-vejo-vantagens-e-voce/>. Acesso em: 23 nov. 2020.

SCHMITZ, D. **Tudo que você queria saber sobre Git e GitHub, mas tinha vergonha de perguntar**. 2015. Disponível em: <https://tableless.com.br/tudo-que-voce-queria-saber-sobre-git-e-github-mas-tinha-vergonha-de-perguntar/>. Acesso em: 26 dez. 2020.

SILVA, G. **O que é e como funciona a linguagem JavaScript?** 2015. Disponível em: <https://canaltech.com.br/internet/O-que-e-e-como-funciona-a-linguagem-JavaScript/>. Acesso em: 22 nov. 2020.

SILVEIRA, P. **O que é SQL?** 2019. Disponível em: <https://www.alura.com.br/artigos/o-que-e-sql?> Acesso em: 4 dez. 2020.

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: 16 nov. 2020.

SPRING. **Spring Quickstart Guide**. 2020. Disponível em: <https://spring.io/quickstart>. Acesso em: 29 nov. 2020.

TECHTUDO. **PostgreSQL Database: gerencie o banco de dados em diversas plataformas**. 2014. Disponível em: <https://www.techtudo.com.br/tudo-sobre/postgresql.html>. Acesso em: 29 nov. 2020.

XAVIER, T. **O que é HTML e qual sua funcionalidade?** 2019. Disponível em: <https://rockcontent.com/br/blog/html/>. Acesso em: 16 nov. 2020.

ZAVADNIAK, C. **Como funciona uma requisição HTTP**. 2017. Disponível em: <https://medium.com/clebertech/como-funciona-uma-requisi%C3%A7%C3%A3o-http-cf76f66fe36e>. Acesso em: 12 dez. 2020.

ZETTA. **Zetta - Agência UFLA de Inovação, Geotecnologia e Sistemas Inteligentes**. 2020. Disponível em: <https://agenciazetta.ufla.br>. Acesso em: 30 out. 2020.