SOFTWARE PARA SIMULAÇÃO DE SERVIÇOS WEB BASEADOS EM API'S RESTFUL

Lucas Torres Mezzari[1]
Silvio Cesar Viegas[2]
Paulo Tadeu Campos Lopes[3]

RESUMO
Este trabalho, que tem como objetivos demonstrar os conhecimentos estudados durante o curso de Análise e Desenvolvimento de Sistemas, trata-se de um projeto para o desenvolvimento de um sistema *web* que simula uma API REST. Tem como objetivo auxiliar desenvolvedores a desenvolverem aplicações clientes quando a aplicação servidor não se encontra disponível. Pela aplicação cliente, o usuário poderá criar, editar, deletar e testar rotas de forma individual ou cooperativa; Pela aplicação servidor, que pode ser utilizada sozinha, o usuário poderá simular as rotas cadastradas, validar requisições realizadas, criar registros de comunicação a cada chamada, e realizar todas as demais funcionalidades da aplicação cliente. O sistema é desenvolvido com JavaScript no *Front End*, utilizando-se do *framework* VueJS, HTML, CSS, *Bootstrap*; e, no *Back End,* TypeScript e NodeJS usando o *framework Hapi* e o banco de dados *NoSQL,* MongoDB. Tendo em vista o conceito de *Open Source*, o projeto foi desenvolvido, utilizando de princípios SOLID, para permitir futuras modificações e expansões em suas funcionalidades. As telas da aplicação cliente foram desenhadas usando um modelo como base e modificadas para se encaixar na visão do projeto, com suas paletas de cores e funcionalidades. Como resultado o sistema se encontra em sua versão *alpha* que poderia ser liberada para uso.

Palavras-chave: sistema *Web*; REST; modelo cliente-servidor; Vue JS; JavaScript.

1 INTRODUÇÃO

Este trabalho tem como objetivo documentar o desenvolvimento de um sistema *web* para simulação de serviços *RESTFUL* API, e como foco, elaborar uma ferramenta *Open Source* que possa fornecer auxílio a desenvolvedores de aplicações cliente que necessitem replicar uma aplicação servidor. Partindo dos

[1] Graduando do curso de Análise e Desenvolvimento de Sistemas da Faculdade QI Brasil (FAQI), Gravataí/RS. Contato: lucas.mezzari1@gmail.com
[2] Professor Mestre, coordenador dos cursos do Eixo Tecnologia e Educação da FAQI, Gravataí/RS. Contato: silvio.viegas@qi.edu.br
[3] Professor do Programa de Pós-graduação em Ensino de Ciências e Matemática da Universidade Luterana do Brasil -ULBRA. Canoas, 2022. Contato: pclopes@qi.edu.br

requisitos básicos de criar, editar, deletar, e listar rotas, serão analisadas soluções similares e, tendo em vista os modelos cliente-servidor e os princípios REST (*Representational State Transfer* ou Transferência de Estado Representacional), traçados os requisitos funcionais para aplicação final.

Baseando-se nos requisitos levantados e dos conceitos estudados, a aplicação foi modelada de forma a permitir futuras mudanças e modificações, proporcionando opções para desenvolvedores alterarem o comportamento do sistema como desejarem. Ao final deste artigo, será apresentado um sistema *web* composto de uma aplicação *Back End* e uma aplicação *Front End* com cumpra com todos os requisitos funcionais levantados.

1.1 TEMA

Desenvolvimento de uma ferramenta *web Open Source* para fornecer apoio aos desenvolvedores e permitir a simulação de serviços APIs REST de forma rápida, prática, e colaborativa.

1.2 DELIMITAÇÕES

A primeira versão deste sistema consiste em um sistema *Back End* que irá disponibilizar e organizar as rotas simuladas, e um sistema *Front End* que irá disponibilizar ao usuário ferramentas para acessar todas as funcionalidades do servidor de forma prática e simples.

O sistema *Back End*, aplicação no lado do servidor, será desenvolvida em *node js* utilizando-se da linguagem *TypeScript* e o framework *Hapi.dev* para criação de rotas. O sistema Front End, aplicação do lado do cliente, será desenvolvido com *JavaScript* e o framework *Vue.js* como uma aplicação *web* para navegadores.

Essa versão irá se limitar a rodar apenas em ambiente local e sua versão cliente se encontra nos estágios iniciais do desenvolvimento, abandonando conceitos de UX e usabilidade para priorizar as funcionalidades.

1.3 PROBLEMA

O desenvolvimento de software está se tornando cada vez mais complexo e interligado para garantir a integridade, segurança, escalabilidade e controle das informações. Com isso, muitas aplicações escolhem um modelo de arquitetura baseado em Cliente-Servidor para poder garantir que aplicações *web* e *mobile* (cliente) possam utilizar as informações da forma desejada. Em uma arquitetura como essa, como trazer o controle de volta para os desenvolvedores das aplicações cliente?

1.4 JUSTIFICATIVA

Com a popularização dos smartphones o modelo cliente-servidor se tornou ainda mais popular pela necessidade de que os novos aplicativos, construídos para as diversas plataformas (Android, iOS e Windows Phone), possam compartilhar as mesmas regras de negócio e informações das aplicações *Web*. Com esse modelo as aplicações clientes sofrem durante as etapas iniciais do desenvolvimento por não possuírem acesso às informações ou por possuírem muito pouco controle, ou até nenhum, sobre fluxos dentro da aplicação.

Alguns exemplos de comportamentos controlados pelo *Back End* poderiam ser:

- Erros em função de indisponibilidade dos servidores;
- Situações onde o usuário precisa ser deslogado da aplicação;
- Situações onde o usuário precisa aceitar os termos de uso;
- Situações onde o usuário precisa realizar algum procedimento no primeiro acesso;

Para isso será montado uma aplicação que tente fornecer aos desenvolvedores, das aplicações cliente, um pouco mais de controle e assim possibilitar o desenvolvimento de aplicações com maior qualidade.

1.5 OBJETIVO GERAL

Desenvolver uma ferramenta *web Open Source* onde um profissional de TI possa criar, editar, remover, configurar, e testar serviços simulados em um ambiente compartilhado ou individual.

1.6 OBJETIVOS ESPECÍFICOS

- Estudar as soluções já existentes e elaborar funcionalidades que possam suprir algumas das necessidades deixadas de fora das aplicações já existentes; Modelar um sistema *Back End* e *Front End*;

- Desenvolver um sistema *Back End* que possa ser modificado posteriormente para suprir novas funcionalidades;

- Desenvolver um sistema *Front End* que possa utilizar as funcionalidades desenvolvidas na aplicação servidor.

2 REFERENCIAL TEÓRICO

Nessa etapa serão apresentados alguns conceitos relacionados ao desenvolvimento dessa aplicação.

2.1 FRAMEWORKS

Durante a elaboração de um software é comum encontrar certos padrões ou costumes que são carregados de projeto a projeto evoluindo e se transformando. A partir desses padrões e necessidades é criado e elaborado um novo framework.

> *Framework* é um conjunto de técnicas, ferramentas ou conceitos pré-definidos usados para resolver um problema de um projeto ou domínio específico. É, basicamente, uma estrutura de trabalho que atua com funções pré-estabelecidas que se adaptam à situação e à organização em questão. D'AVILLAR, (2018).

2.2 LINGUAGENS DE PROGRAMAÇÃO

Uma linguagem de programação permite a comunicação entre os desenvolvedores e os computadores. Através de uma sintaxe pré-estabelecida um desenvolvedor pode elaborar uma série de instruções que serão transformadas para binário, através de um processo chamado compilação, e executadas pelo computador. Assim como as linguagens usadas para comunicação interpessoal, há uma gama de linguagens de programação com suas próprias sintaxes, particularidades e objetivos.

> Linguagens de programação e códigos de computador fizeram nossa vida mais simples. Seja pelos automóveis, bancos, aplicativos domésticos, ou hospitais, cada aspecto de nossas vidas depende de códigos. (BELANI, 2020).

2.2.1 Javascript

É uma linguagem de programação interpretada que é principalmente utilizada em aplicações *web* no lado do cliente. Foi criada por Brendan Eich em 1995 para o Netscape Navigator 2.0 e logo outros navegadores adicionaram suporte para ela. Atualmente a linguagem passou a ser usada tanto no lado cliente quanto no lado do servidor, permitindo assim, que todas as camadas de uma aplicação sejam desenvolvidas em uma única linguagem.

Figura 1 - Exemplo Javascript

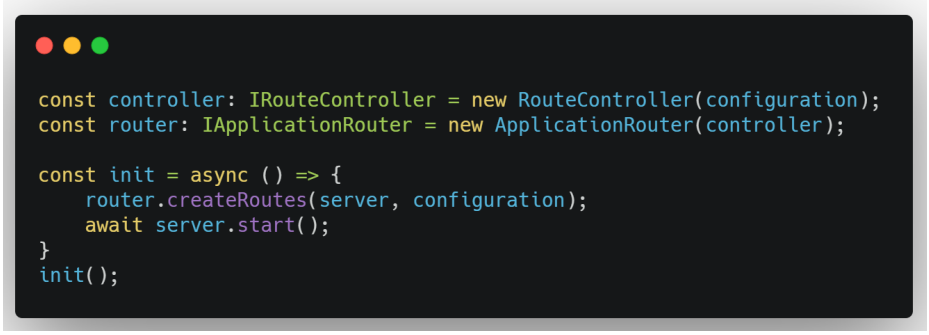
```
function postRoute(data) {
    return execute((resource) => api
        .post('route', data)
        .then((response) => {
            if (!response.data) {
                resource.postState(FAILED, response);
                return;
            }

            resource.postState(SUCCESSFUL, response.data.data);
        })
        .catch((error) => {
            resource.postState(FAILED, error);
        }));
}
```

Fonte: elaborado pelo autor, 2021.

2.2.2 Typescript

TypeScript é uma linguagem de programação de código aberto que permite os desenvolvedores a criar suas aplicações utilizando de todas as vantagens do JavaScript junto a uma estrutura utilizada em linguagens fortemente tipadas como Java e C. Desenvolvida pela Microsoft em 2012, o TypeScript possibilitou a aplicação de arquiteturas mais robustas e melhores práticas de programação, permitindo, inclusive, o paradigma de Programação Orientada a Objetos.

Figura 2 - Exemplo TypeScript



```
const controller: IRouteController = new RouteController(configuration);
const router: IApplicationRouter = new ApplicationRouter(controller);

const init = async () => {
    router.createRoutes(server, configuration);
    await server.start();
}
init();
```

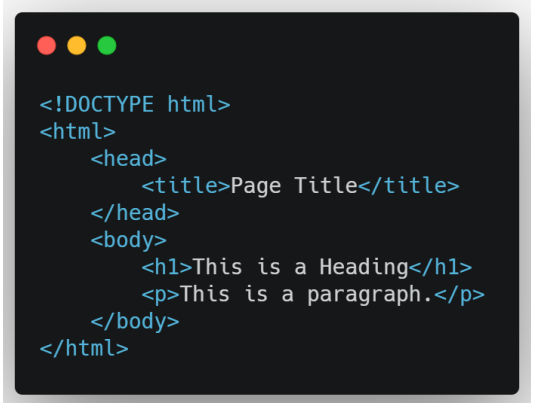
Fonte: elaborado pelo autor, 2021.

2.3 HTML

É uma linguagem de marcação padrão utilizada majoritariamente para a elaboração de interfaces *web*. Através dela é possível definir o conteúdo de uma página *web* utilizando-se de tags que são elementos entre “<” e “>”. Utilizando-se

apenas dela é possível criar páginas estáticas para apresentação de conteúdos sem muita dificuldade, e com a adição de outras tecnologias como CSS (*Cascading Style Sheets* ou Folhas de Estilo em Cascata) e JavaScript é possível criar páginas complexas e visualmente agradáveis.

Figura 3 - Exemplo HTML



Fonte: elaborado pelo autor, 2021.

2.4 VUEJS

É um framework JavaScript de código aberto para elaboração de interfaces *web*. Ela teve seu lançamento em Fevereiro de 2014, por Evan You, e foi elaborada para ser performática, versátil e de fácil aprendizado.

Com uma arquitetura dividida em três categorias: *Template* utilizando-se de HTML; *Scripts* utilizando-se da linguagem JavaScript para estabelecer os comportamentos e processos da página; e *Style* que utiliza CSS para modificar a apresentação de elementos HTML. Segundo VueJs.org (2021), o VueJS consegue se expandir de uma simples biblioteca até uma estrutura completa com um ecossistema adotável incrementalmente.

Figura 4 - Exemplo VueJs

```
<template>
  <div>
    <notifications></notifications>
    <router-view></router-view>
  </div>
</template>
<script>
export default {
  mounted: function () {
    // Do Something
  }
};
</script>
<style lang="scss">
.vm--overlay {
  backdrop-filter: blur(5px);
  background-color: rgba(0, 0, 0, 0.7) !important;
}
</style>
```

Fonte: elaborado pelo autor, 2021.

2.5 HAPI

Framework para JavaScript que permite criar aplicações do lado servidor com simplicidade e segurança. Criado por Eran Hammer para lidar com a escala de uma *Black Friday* no Walmart. Possui um rico ecossistema com plugins oficiais para resolver problemas críticos que surgirem durante o desenvolvimento da aplicação.

Com uma sintaxe simples e configuração intuitiva garante segurança e facilidade para a elaboração do sistema e suas rotas.

Figura 5 - Exemplo definição de rotas no Hapi

```
server.route({
    method: 'GET',
    path: '/hello',
    handler: (request: any) => {
        console.log('Hello World');
        return {
          code: 200,
          response: 'Hello World'
        };
    },
});
```

Fonte: elaborado pelo autor, 2021.

2.5.1 Rotas

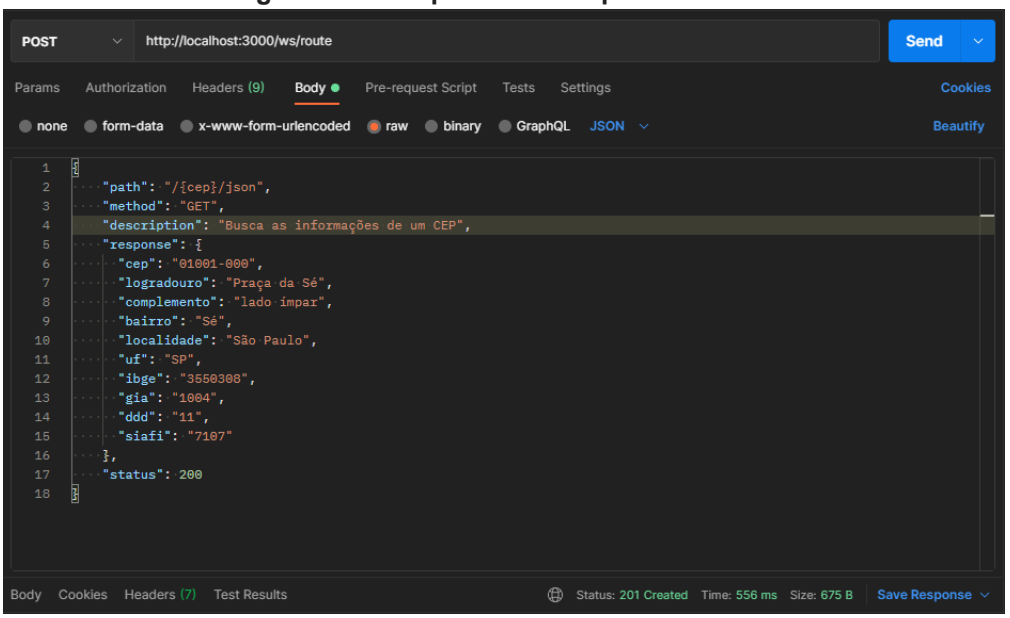
Uma rota é um endereço no servidor para onde o cliente poderá realizar requisições que irão executar um procedimento e retornar uma resposta. Uma rota possui um método que indica o que vai ser feito, como por exemplo *GET* irá buscar informações e um *POST* irá cadastrar informações a partir do corpo da requisição.

Um mesmo endereço poderá suportar diferentes tipos de métodos e executar procedimentos únicos de acordo com o método. A resposta irá variar de acordo com a aplicação podendo ser JSON (*JavaScript Object Notation*, ou Notação de Objeto JavaScript), XML, HTML, ou outro tipo que seja necessário.

As requisições podem ser divididas em: Método, caminho (endereço), cabeçalho e corpo. O cabeçalho de uma requisição é uma série de valores com suas chaves que irão carregar informações menos prioritárias e opcionais. O corpo de uma requisição poderá variar de acordo com a necessidade, sendo o mais comum o JSON.

Uma resposta irá possuir uma propriedade a mais para informar o status da requisição informando se foi bem sucedida ou não.

Figura 6 - Exemplo chamada pelo Postman



Fonte: elaborado pelo autor, 2021.

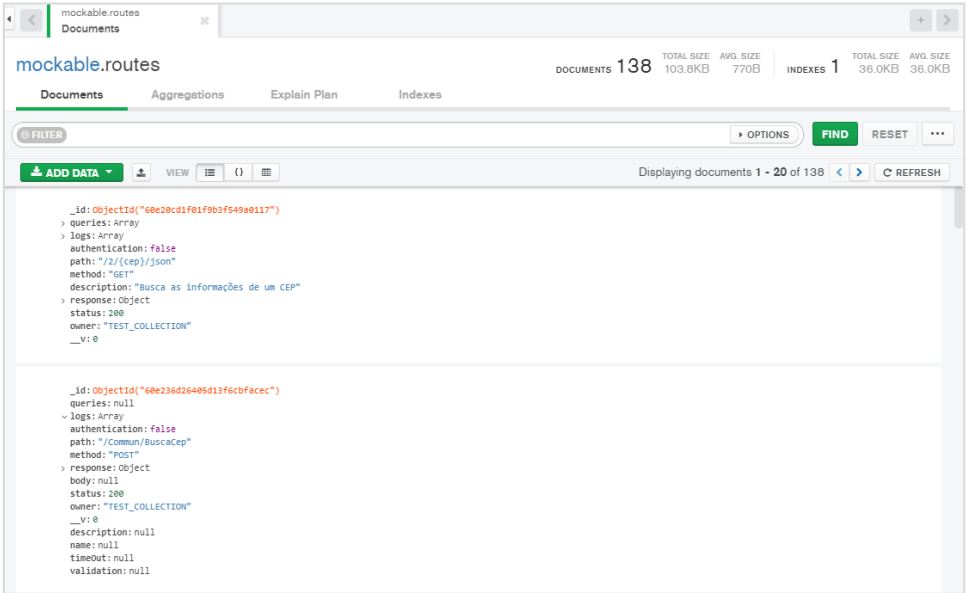
2.6 MONGODB

MongoDB é um banco de dados não relacional, ou para quem já conhece, um banco de dados *NoSQL*. O termo *NoSQL* (*Not Only SQL* ou Não apenas SQL) surgiu por volta de 1998 mas só foi se tornar popular mais recentemente com os conceitos de Big Data, onde um único servidor hospedando todo o banco de dados de uma aplicação tornou-se custoso. Com essa nova necessidade de armazenar e trafegar grandes quantidade de informação surgiram os bancos *NoSQL* que proporcionam maior flexibilidade, escalabilidade, e alta performance.

O MongoDB é um banco *NoSQL* orientado a documentos que armazena as informações em uma estrutura JSON. O modelo orientado a documentos "é especialmente eficiente para tratar dados não estruturados, já que uma única coleção pode contar com grupos de dados (documentos) de diversos formatos diferentes" (VICTÓRIA, 2019).

A maior vantagem dos bancos *NoSQL* são o fato de possuírem uma escalabilidade horizontal, o que significa, que ao invés de aumentarmos a potência do nosso servidor comprando peças melhores, podemos adicionar novas máquinas para compartilhar o processamento e armazenamento. Além disso, os bancos *NoSQL* permitem um alto desempenho nas consultas e são altamente flexíveis.

Figura 7 - Documentos salvos da aplicação



Fonte: elaborado pelo autor, 2021.

3 METODOLOGIA

“O mundo da ciência é movido por perguntas. Essas perguntas precisam de respostas e são as pesquisas o melhor caminho para consegui-las.”.(BRAZ, 2021).

Este projeto pode ser classificado, quanto à abordagem, uma pesquisa qualitativa. Quanto à natureza, uma pesquisa aplicada. Quanto a sua finalidade, uma pesquisa exploratória. E quanto ao procedimento, uma pesquisa experimental.

O desenvolvimento do projeto foi dividido em três etapas. A primeira abordando a definição e elaboração do projeto onde foram levantadas os requisitos necessários e analisadas as tecnologias. Com os requisitos levantados segue-se para a prototipação do produto final e a montagem da documentação. E como última etapa foi feito o desenvolvimento onde inicia-se a construção das aplicações *Front End* e *Back End* com as tecnologias estabelecidas.

Quadro 1 - Cronograma do Desenvolvimento

Etapa	Item
1 Definição e Elaboração.	1.1 Funcionalidades; 1.2 Tecnologias;
2 Prototipação e Documentação.	2.1 Casos de Uso; 2.2 Prototipação;
3 Desenvolvimento da Aplicação	3.1 Aplicação Servidor; 3.2 Aplicação Cliente;

Fonte: elaborado pelo autor, 2021.

O Quadro 1 apresenta o cronograma que será seguido no desenvolvimento do projeto, mas para melhor entendimento ele será dividido da seguinte forma:

1 Definição e Elaboração: Nessa etapa serão levantadas as tecnologias a serem usadas para o desenvolvimento e levantado os requisitos funcionais.

1.1 Funcionalidades: Neste ponto serão apresentados os requisitos funcionais e não funcionais da aplicação com observações a respeito do funcionamento.

1.2 Tecnologias: Nesse primeiro segmento serão levantadas as tecnologias que serão usadas para o desenvolvimento da aplicação junto com uma breve explicação da escolha.

2 Prototipação e Documentação: Aqui serão apresentadas as documentações necessárias para entendimento do funcionamento do sistema assim como as telas criadas.

2.1 Casos de Uso: Esse tópico irá apresentar os casos de uso criados para a aplicação como um todo.

2.2 Prototipação: Esse tópico irá apresentar as telas desenhadas para o sistema Front End junto com breves explicações sobre o funcionamento dela.

3 Desenvolvimento da Aplicação: Irá relatar pontos importantes do desenvolvimento da aplicação como um todo. Será dividido em dois segmentos separados para cada parte da aplicação.

3.1 Aplicação Servidor: Nesse segmento será relatado o desenvolvimento da aplicação servidor, que pode ser considerada o núcleo principal do projeto.

3.2 Aplicação Cliente: Este segmento irá relatar pontos importantes do sistema *Web* desenvolvido e algumas de suas características.

4 DESENVOLVIMENTO

Partindo do cronograma estabelecido no Quadro 1, aqui serão descritas as etapas pelo qual o projeto passou durante seu desenvolvimento.

4.1 DEFINIÇÃO E ELABORAÇÃO

Para esta etapa foram analisadas outras soluções para o problema em questão e elaborado os requisitos. A análise foi limitada a soluções gratuitas para manter-se justo à concorrência.

Foram analisadas três soluções ao todo, cada uma delas com suas particularidades, e, partindo da análise, levantado os requisitos obrigatórios junto com a elaboração de requisitos diferenciais para garantir a identidade da aplicação.

4.1.1 Funcionalidades

Partindo da análise realizada foram levantados os requisitos e divididos em prioridades. Requisitos de alta prioridade significam que são funcionalidades básicas compartilhadas em todas as aplicações pesquisadas, tornando assim, funcionalidades primárias para qualquer solução decente para o problema. As funcionalidades de prioridade mais baixa são, em sua maioria, funcionalidades diferenciais.

Segue abaixo os requisitos funcionais do sistema.

Quadro 1 - Cadastro de Rotas

Identificador	RF01		
Nome	Cadastro de Rotas		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Alta
Descrição	O sistema deve permitir o cadastro de rotas a serem simuladas.		

Fonte: elaborado pelo autor, 2021.

Quadro 2 - Edição de Rotas

Identificador	RF02		
Nome	Edição de Rotas		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari

Versão	1.0	**Prioridade**	Alta
Descrição	O sistema deve permitir a edição de rotas já cadastradas.		

Fonte: elaborado pelo autor, 2021.

Quadro 3 - Apagar Rotas

Identificador	RF03		
Nome	Apagar Rotas		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Alta
Descrição	O sistema deve permitir que rotas já cadastradas possam ser apagadas.		

Fonte: elaborado pelo autor, 2021.

Quadro 4 - Listar Rotas

Identificador	RF04		
Nome	Listar Rotas		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Alta
Descrição	O sistema deve apresentar todas as rotas cadastradas e ativas.		

Fonte: elaborado pelo autor, 2021.

Quadro 5 - Importar Rotas

Identificador	RF05
Nome	Importar Rotas

Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Média
Descrição	O sistema deve permitir que rotas sejam importadas e cadastradas no sistema de forma simples. Obs.: Em um primeiro momento as rotas poderão ser importadas através de uma coleção do Postman ou de um link do Swagger.		

Fonte: elaborado pelo autor, 2021.

Quadro 6 - Exportar Rotas

Identificador	RF06		
Nome	Exportar Rotas		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Média
Descrição	O sistema deve permitir que as rotas possam ser exportadas para o formato desejado. Obs.: Para o sistema atual as rotas poderão ser exportadas para uma coleção do Postman.		

Fonte: elaborado pelo autor, 2021.

Quadro 7 - Cadastrar Logs

Identificador	RF07
Nome	Cadastrar Logs

Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Alta
Descrição	O sistema deve permitir que qualquer rota criada salve um log guardando as informações relevantes.		

Fonte: elaborado pelo autor, 2021.

Quadro 8 - Validar Requisição

Identificador	RF08		
Nome	Validar Requisição		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Média
Descrição	O sistema deve permitir que as rotas criadas possam possuir simples verificações. Obs.: Para o sistema atual serão suportadas validações simples de Autenticação pelo cabeçalho e / ou validação de integridade na requisição como um todo.		

Fonte: elaborado pelo autor, 2021.

Quadro 9 - Listar Logs

Identificador	RF09		
Nome	Listar Logs		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Alta
Descrição	O sistema deve permitir que todos os logs salvos possam ser listados.		

Fonte: elaborado pelo autor, 2021.

Quadro 10 - Simular Rotas

Identificador	RF10		
Nome	Simular Rotas		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Alta
Descrição	O sistema deve permitir que as rotas criadas possam ser acessadas por fora da aplicação, criando logs sempre que forem acessadas (RF07) e respeitando as validações configuradas (RF08).		

Fonte: elaborado pelo autor, 2021.

Quadro 11 - Testar Rotas

Identificador	RF11		
Nome	Testar Rotas		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari
Versão	1.0	**Prioridade**	Média
Descrição	O sistema deve permitir que as rotas possam ser testadas internamente. Sempre que uma rota for testada será executada a simulação de uma rota (RF10).		

Fonte: elaborado pelo autor, 2021.

Quadro 12 - Colaborar rotas entre usuários

Identificador	RF12		
Nome	Colaborar		
Data de Criação	04/07/2021	**Autor**	Lucas T. Mezzari

Versão	1.0	**Prioridade**	Média
Descrição	O sistema deve permitir que usuários possam trabalhar de forma colaborativa online.		

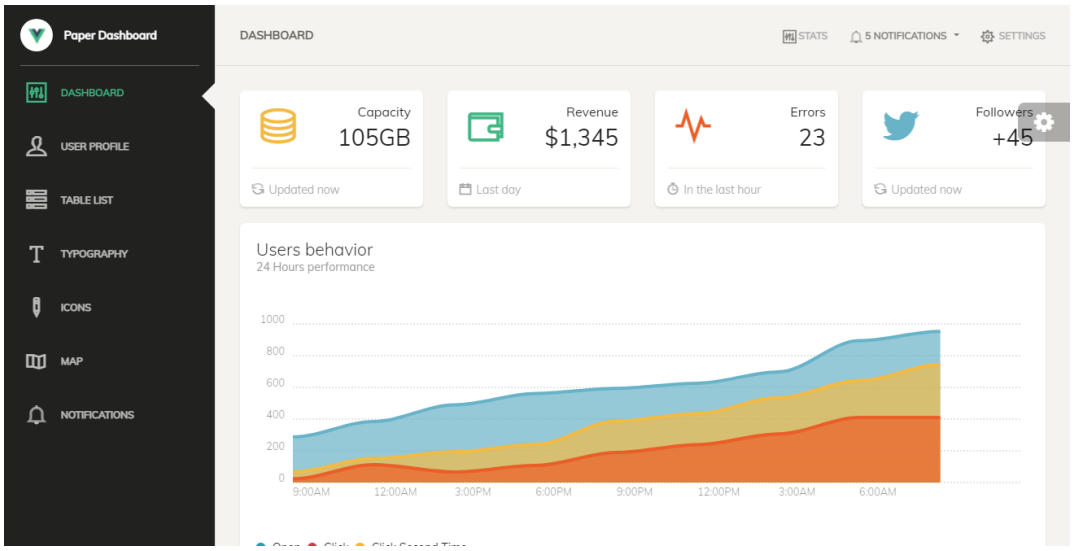
Fonte: elaborado pelo autor, 2021.

4.1.2 Tecnologias

Para dar seguimento ao desenvolvimento foi escolhido o *framework* Hapi.dev para criar a aplicação *Back End*, junto com a linguagem TypeScript e MongoDB, um banco de dados *NoSQL*. Essas tecnologias foram escolhidas em função da simplicidade, escalabilidade, e familiaridade. Outro ponto importante que pesou na decisão foi a possibilidade de usar a biblioteca Joi para realizar as validações das requisições. Com ela será possível realizar validações poderosas para cada chamada, desde simples validações de tipo a validações completas de objeto com regex, limite de caracteres, e outras funcionalidades.

Para a aplicação *Front End* foi escolhido o *framework* Vue JS e a linguagem JavaScript. A escolha do *framework* foi em função da estrutura, sintaxe, e escalabilidade. Para agilizar a criação das telas do projeto, foi escolhido um modelo gratuito desenvolvido por Cristi Jora e do grupo Creative Team.

Figura 8 - Modelo de layout disponível na Creative Team



Fonte: elaborado pelo autor, 2021.

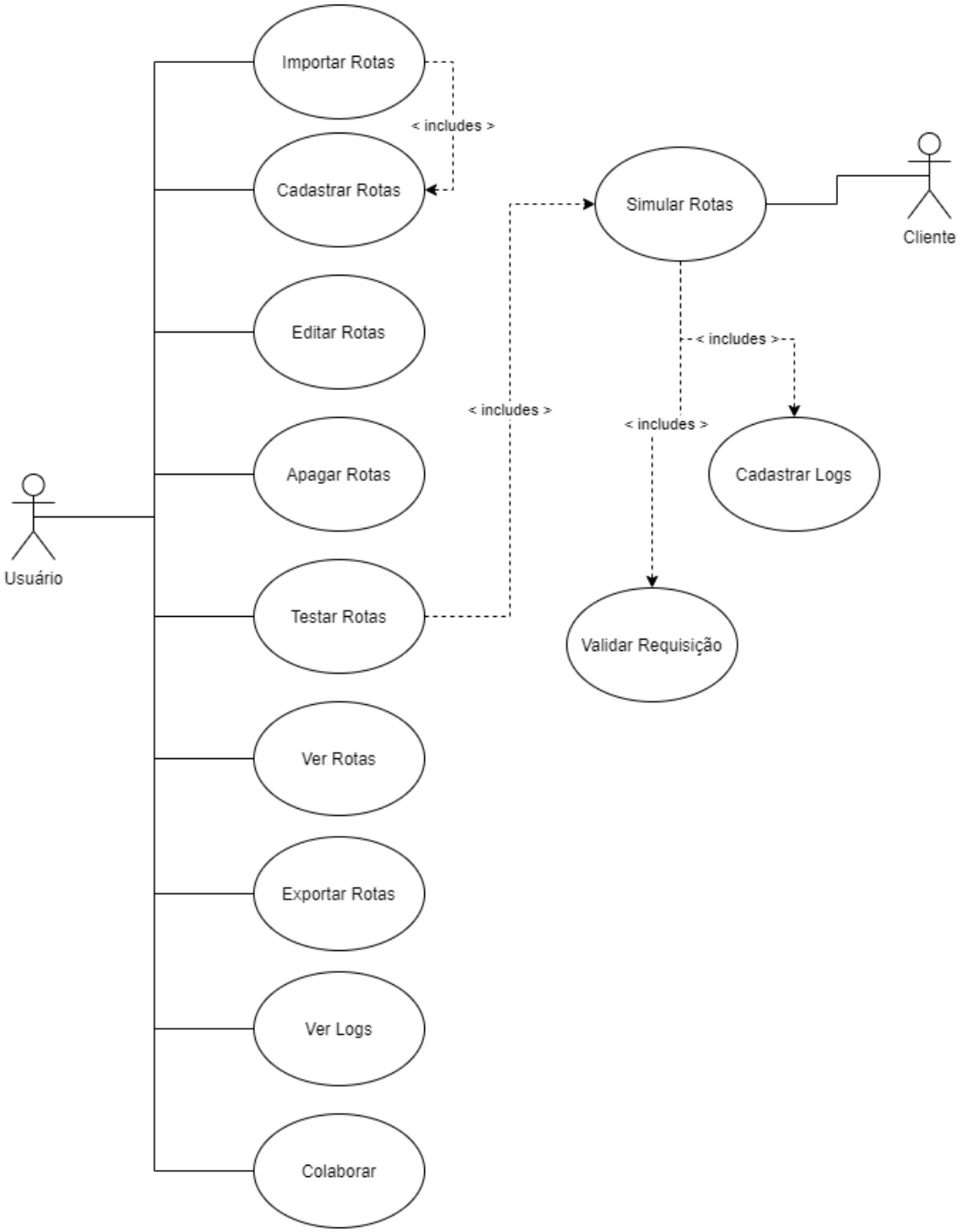
4.2 PROTOTIPAÇÃO E DOCUMENTAÇÃO

Neste capítulo será apresentado os casos de uso da aplicação e abordado quais os atores que utilizaram o software. Também será apresentada as telas criadas utilizando do modelo em Vue JS.

4.2.1 Casos de uso

O diagrama a seguir representa de forma visual quais são as funcionalidades da aplicação e os atores que irão atuar sobre elas. Este tipo de diagrama é utilizado regularmente na modelagem de *Software* pela forma clara de como o sistema pode ser representado. Cada elipse representa um requisito funcional apresentado no segmento 4.1.1 e cada boneco irá representar um ator no sistema. Toda ligação marcada como *"includes"* representa que, sempre ao executar a funcionalidade, ambas serão executadas.

Figura 9 - Diagrama de casos de uso da aplicação



Fonte: elaborado pelo autor, 2021.

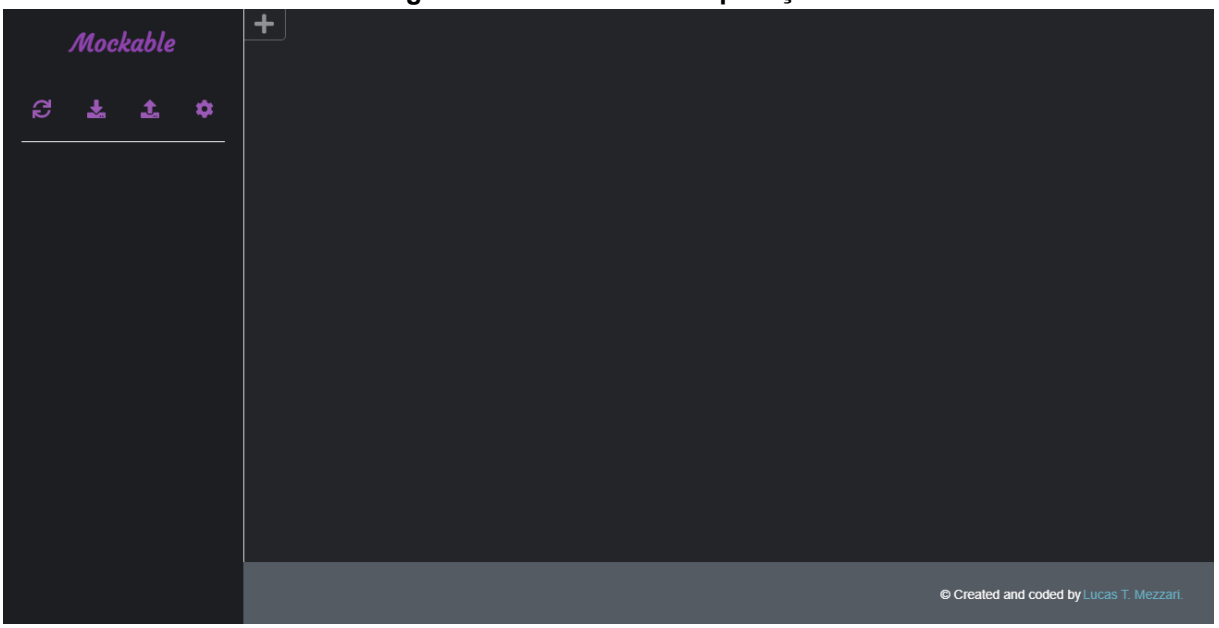
Acima na Figura 9 temos representados os dois atores da aplicação, onde o usuário será o desenvolvedor que estará utilizando a aplicação *Front End*, e o cliente será toda e qualquer aplicação que irá se conectar com o sistema *Back End* para acessar as rotas. Um exemplo prático, para melhor entendimento dos atores e suas funções, seria um aplicativo Android (O Cliente), que ainda não possui um sistema

Back End, irá acessar as rotas que o desenvolvedor (O Usuário) cadastrou no sistema *Front End*.

4.2.2 Prototipação

A aplicação *Web* do *Front End* será apenas uma única página que irá apresentar todas as funcionalidades da aplicação. A seguir temos a tela que um novo usuário irá ver ao entrar na aplicação:

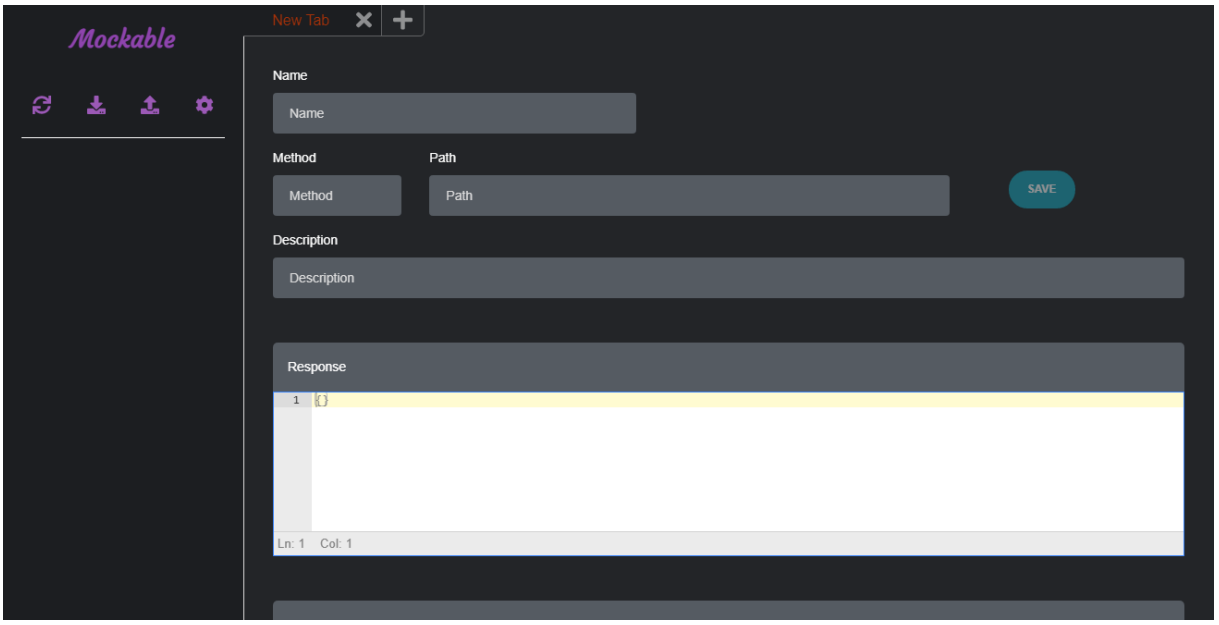
Figura 10 - Tela inicial da aplicação



Fonte: elaborado pelo autor, 2021.

A tela inicial irá apresentar o nome da aplicação, que para exemplo foi usado *Mockable*, com quatro (4) botões logo abaixo que irão dar acesso, respectivamente, às funcionalidades de atualizar, importar (RF05), exportar (RF06), e as configurações. Ao lado do menu lateral há um botão para abrir uma nova aba de cadastro que irá mudar para a figura abaixo.

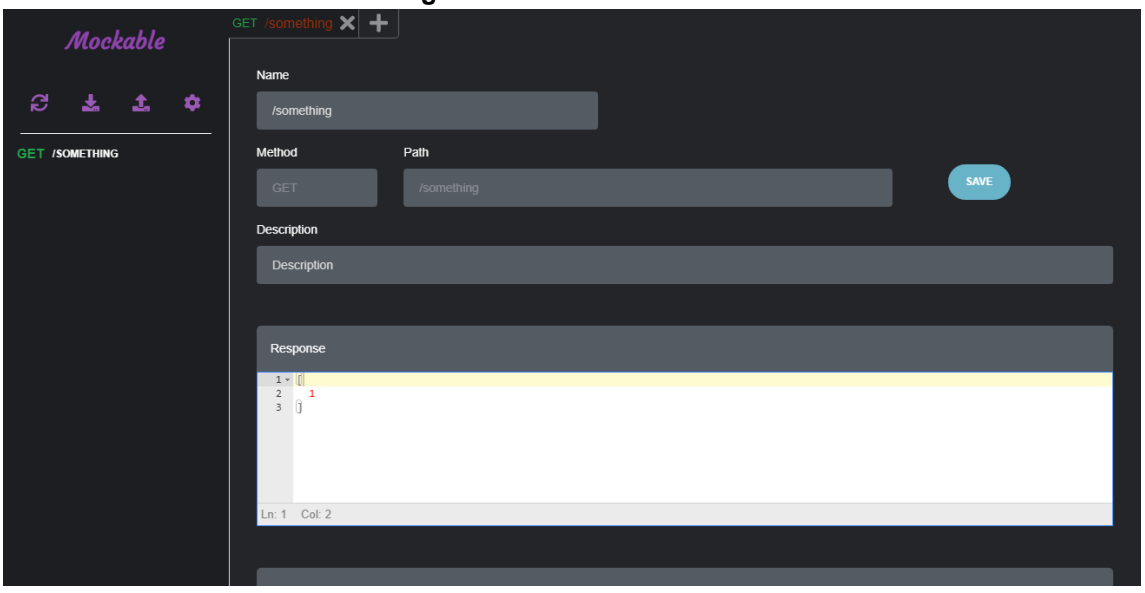
Figura 11 - Aba de cadastro e edição de rota



Fonte: elaborado pelo autor, 2021.

A figura acima mostra a tela de cadastro e edição de rotas. Nessa tela apenas os campos "*Method*" (Método) e *"Path"* (Caminho) serão necessários. Quando preenchidos irão habilitar o botão "*Save*". Ao salvar uma rota (RF01) haverá alterações na tela como a figura abaixo.

Figura 12 - Rota cadastrada

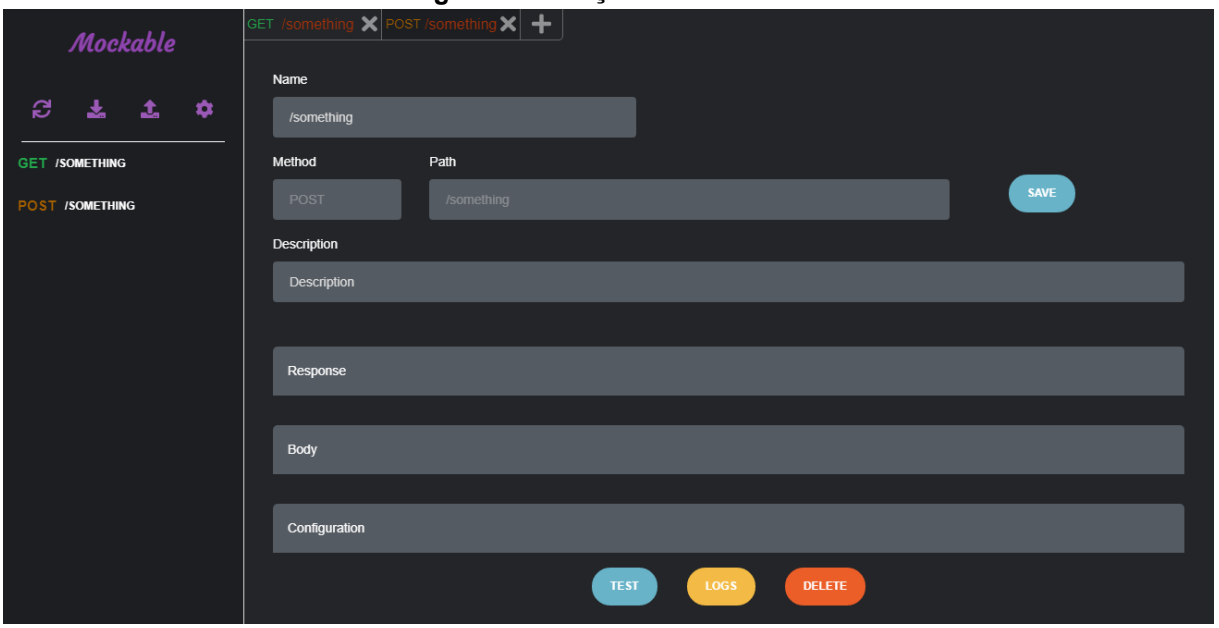


Fonte: elaborado pelo autor, 2021.

Quando uma rota é cadastrada o título da tab irá ser atualizado para uma junção do método com o nome ou caminho da rota. será listado no menu lateral as

rotas cadastradas (RF04), e novas funcionalidades irão ser adicionadas ao editar a rota (RF02). As funcionalidades serão, respectivamente, “*TEST*” (RF11), “*LOGS*” (RF09), “*DELETE*” (RF03). Segue abaixo as funcionalidades liberadas.

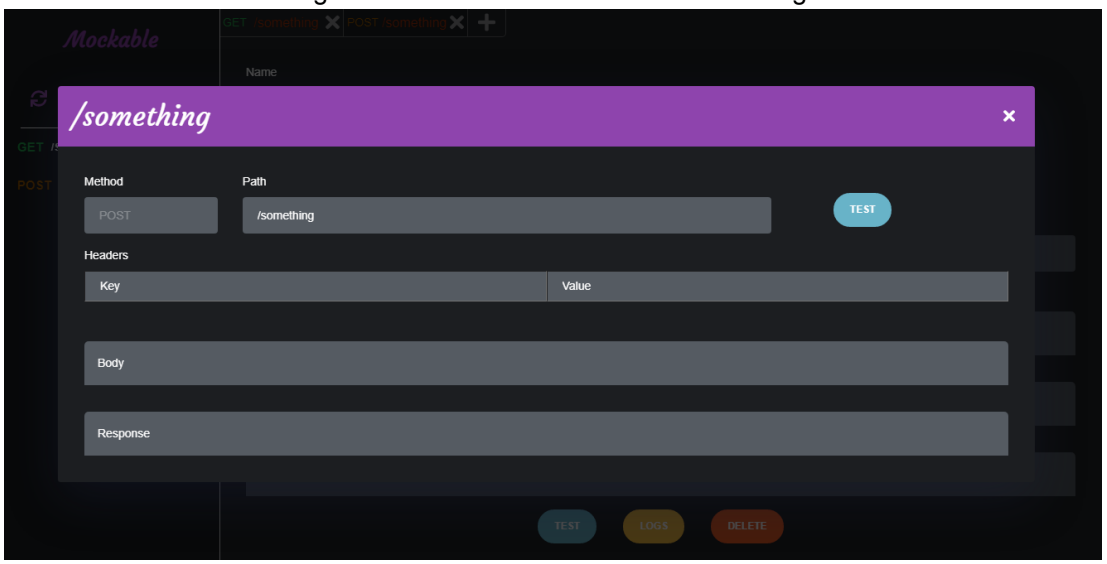
Figura 13 - Edição de uma rota



Fonte: elaborado pelo autor, 2021.

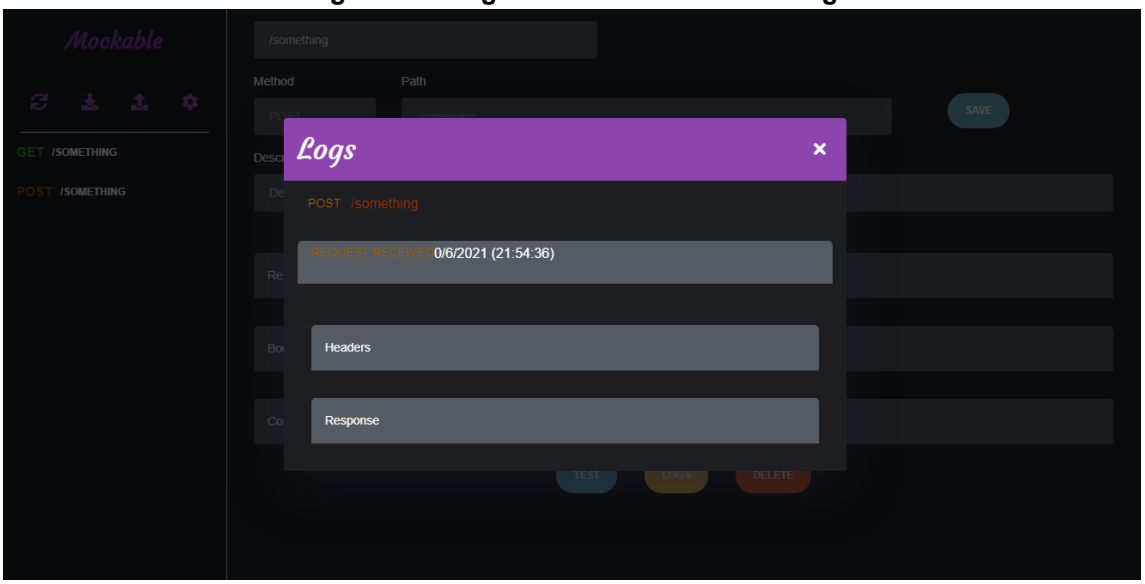
Os botões “*TEST*” e “*LOGS*” irão abrir uma modal permitindo que o usuário utilize a funcionalidade. O botão “*DELETE*” irá apagar a rota que está sendo editada. Serão listadas abaixo as modais.

Figura 14 - Teste da rota POST /something.



Fonte: elaborado pelo autor, 2021.

Figura 15 - Logs da rota POST /something

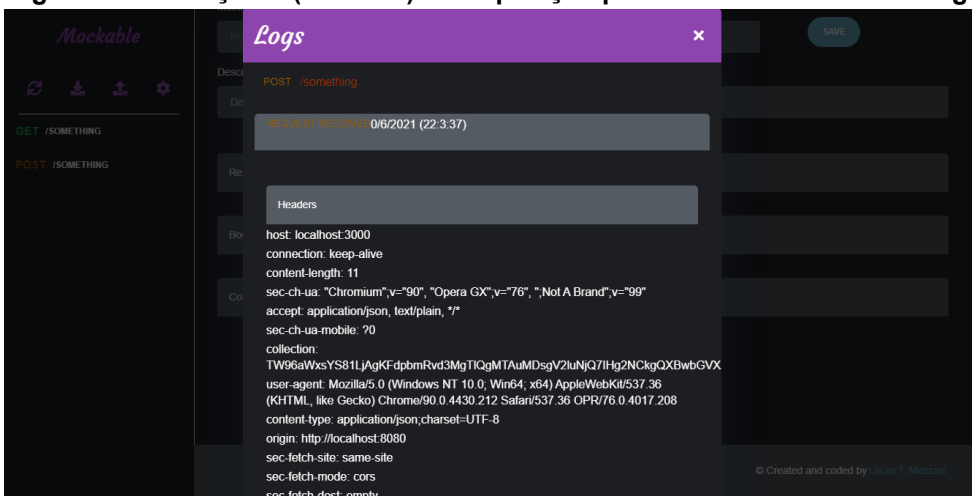


Fonte: elaborado pelo autor, 2021.

Ao executar o teste na Figura 14, será feita uma requisição onde ela será validada (RF08) e será retornada a resposta cadastrada (RF10), ou um erro, caso ocorra. A modal da Figura 15 mostra todas as requisições realizadas para a rota. Os logs são cadastrados automaticamente (RF07) sempre que houver um acesso a rota, seja pela modal de teste da Figura 14, ou através de outra aplicação que necessitar do acesso.

Na modal de logs cada item irá expandir para mostrar seu conteúdo separado em "*Headers*", "*Params*", "*Queries*", "Body", e *"Response*", que também irão expandir para mostrar seus conteúdos. Segue abaixo um exemplo.

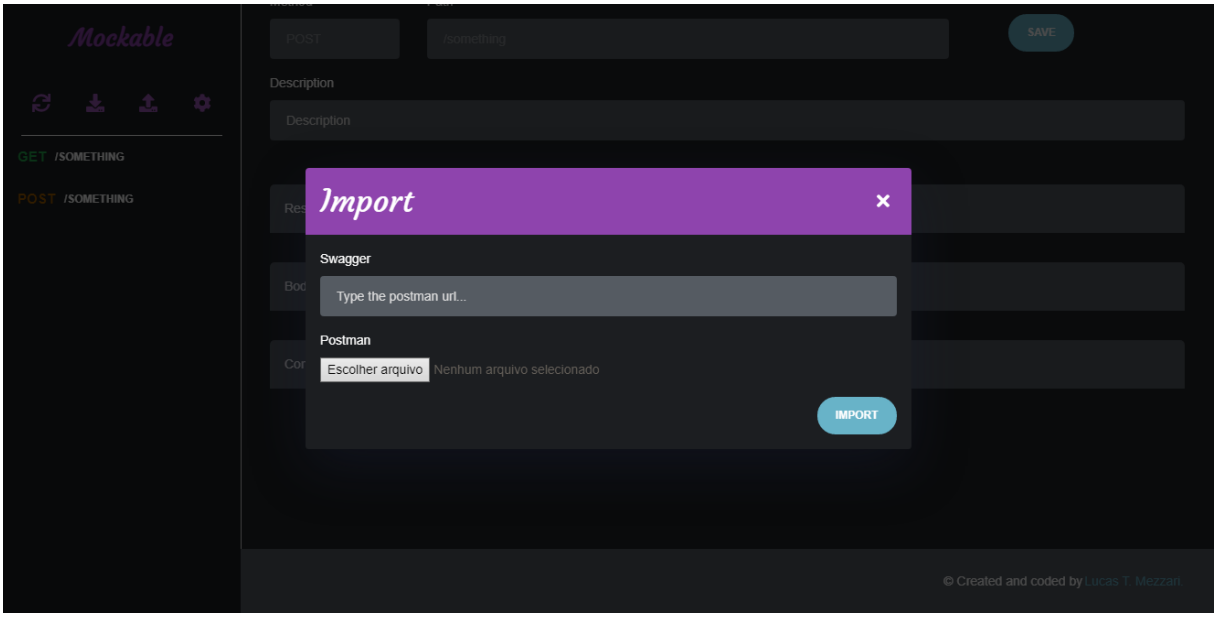
Figura 16 - Cabeçalho (*Headers*) da requisição para a rota POST /something



Fonte: elaborado pelo autor, 2021.

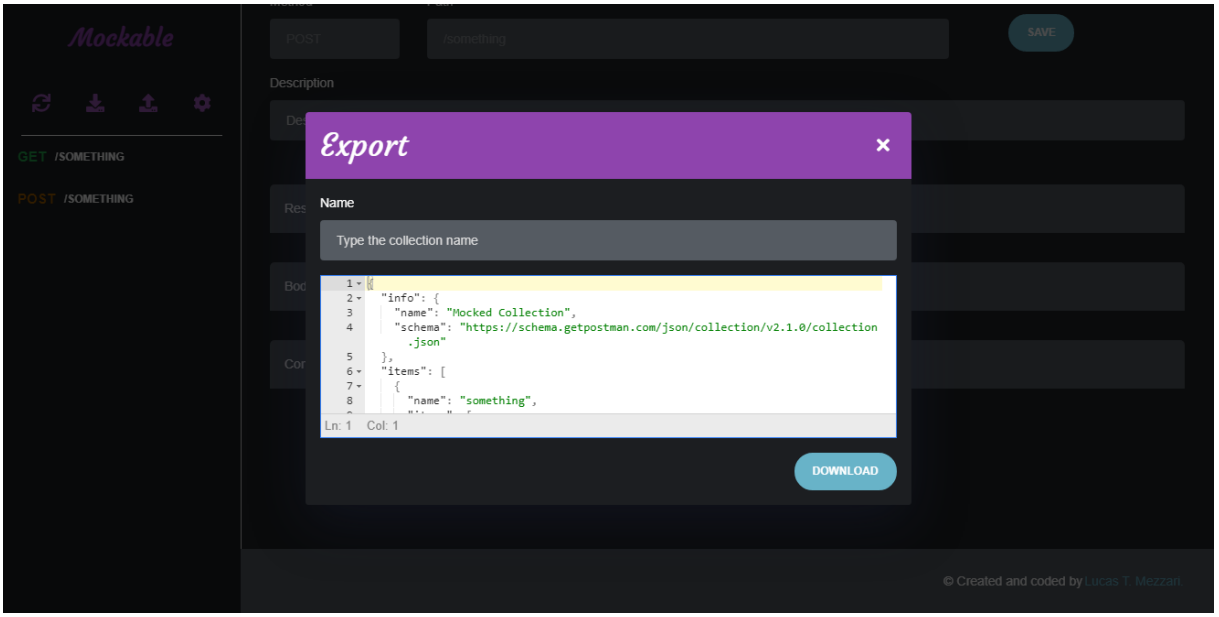
Para as funcionalidades de importar e exportar rotas, que são acessíveis nos botões do menu lateral, serão abertas as modais abaixo.

Figura 17 - Modal para importar rotas para a aplicação



Fonte: elaborado pelo autor, 2021.

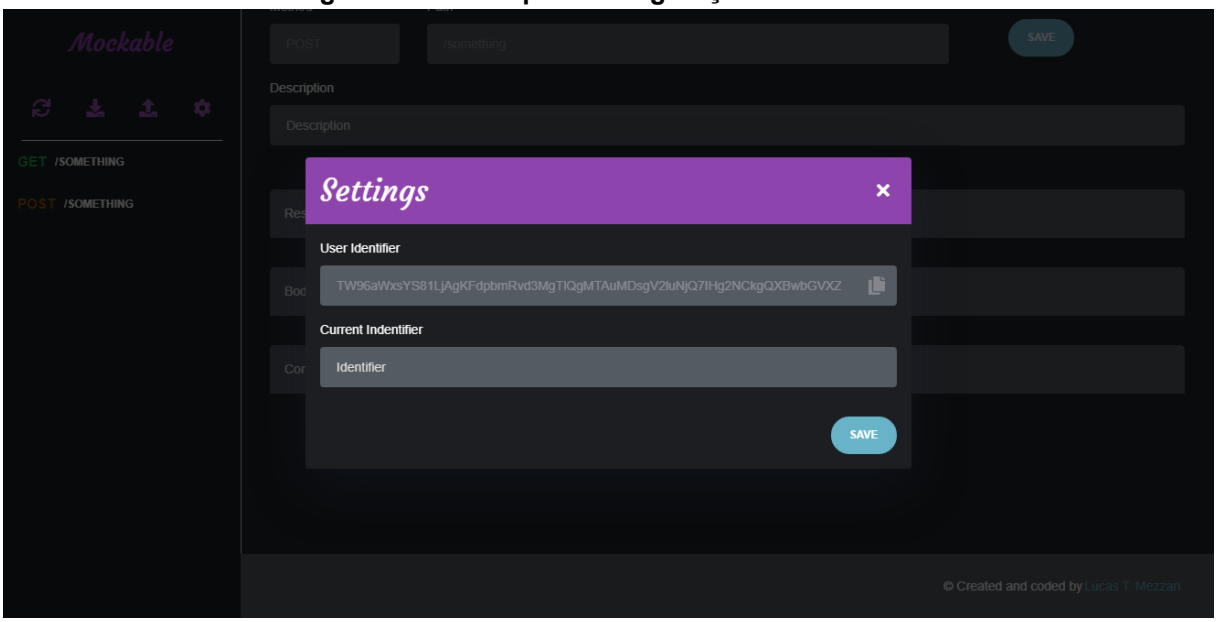
Figura 18 - Modal para exportar as rotas da aplicação



Fonte: elaborado pelo autor, 2021.

Para atender o último requisito do sistema, foi criada uma modal de configurações onde o usuário poderá manipular seu identificador, e com isso, poderá trabalhar de forma cooperativa com outros usuários (RF12).

Figura 19 - Modal para configurações do usuário



Fonte: elaborado pelo autor, 2021.

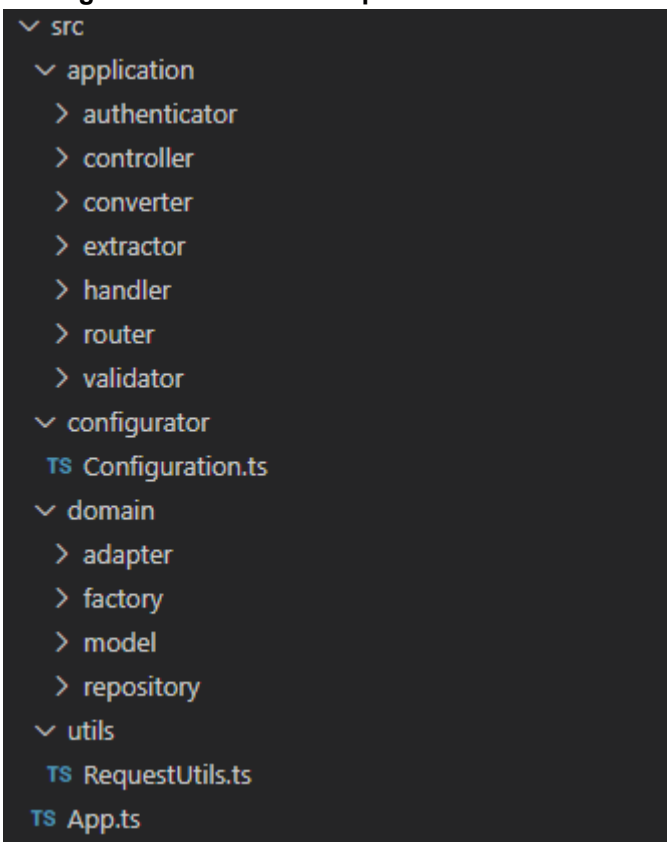
4.3 DESENVOLVIMENTO DA APLICAÇÃO

Para a última etapa do desenvolvimento será apresentado como as aplicações foram elaboradas. Começando pela aplicação servidor, onde será apresentado como a aplicação foi estruturada, e finalizando com a aplicação cliente, que irá complementar com as telas apresentadas no item 4.2.2. “Todos deveriam aprender a programar um computador, por que te ensina a pensar.”.(JOBS,1995).

4.3.1 Aplicação servidor

A aplicação foi desenvolvida com a intenção de ser modificada e atualizada pelo desenvolvedor que precisasse, e por isso sua arquitetura foi separada em módulos tentando evitar o acoplamento de funcionalidades. Utilizando-se da tipagem do TypeScript a estrutura foi separada da forma apresentada abaixo.

Figura 20 - Estrutura de pastas do Back End



Fonte: elaborado pelo autor, 2021.

Como mostrado na figura acima, o projeto foi separado em quatro (4) pastas: *application* que agrupa todos os arquivos que irão alterar as funcionalidades do servidor; *configurator* que possui um único arquivo que é responsável por todo o funcionamento da aplicação; *domain* onde ficam os arquivos relacionados a banco de dados e modelos; *utils* que agrupa funcionalidades diversas para facilitar o desenvolvimento.

As subpastas que ficam dentro de *application* e *domain* agrupam interfaces e implementações que são usadas para determinar o comportamento do sistema. Seguindo o princípio da segregação da interface, que sugere a ideia de que muitas interfaces específicas são melhores que uma única, e o princípio da inversão da dependência, que orienta para depender sempre de uma abstração e nunca de uma implementação, a classe *Configuration* foi elaborada seguinte forma:

Figura 21 - Classe Configuration

Fonte: elaborado pelo autor, 2021.

Utilizando-se dessa estrutura é possível modificar o funcionamento completo da aplicação apenas adicionando alguns arquivos e trocando na *Configuration*. Segue abaixo uma breve explicação das responsabilidades de cada parte:

Factory: É responsável por criar objetos para a aplicação ao extrair informações relevantes das requisições. Uma de suas funções mais importantes é criar a rota que será salva na aplicação.

Handler: Fica responsável por registrar e controlar todas as rotas que forem criadas na aplicação (RF10). Também será ele que irá validar as requisições através do *authenticator* e *validator* (RF08).

Repository: Se responsabiliza de salvar, listar, alterar e remover todas as rotas da aplicação. Será ele que irá se comunicar com o banco de dados.

Authenticator: Irá realizar uma validação simples de autenticação nas rotas que estiverem configuradas.

Validator: Utilizando-se da biblioteca Joi, ele irá validar toda e qualquer requisição que for feita para as rotas cadastradas, desde que tenha sido configurada para ela. Sua configuração possui uma sintaxe baseada em JSON que será interpretada e executada durante a aplicação.

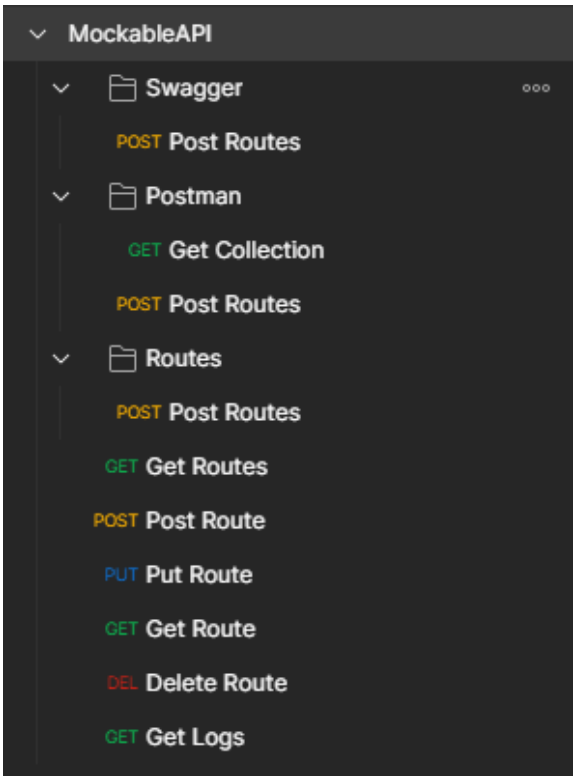
Extractor: É responsável por criar rotas dentro da aplicação (RF05). A partir dele é possível adicionar rotas pelo Swagger ou Postman como foi apresentado na Figura 17. Permite que mais de um esteja ativo ao mesmo tempo.

Converter: É responsável por converter e exportar as rotas em um formato que seja necessário (RF06). Como apresentado na Figura 18, com ele é possível extrair todas as rotas criadas para uma coleção que pode ser importada no Postman. Também permite que mais de um esteja ativo ao mesmo tempo.

Além dos sete (7) itens listados acima, existem mais três (3) pontos importantes a serem abordados. São eles: O *adapter* que permite pequenas modificações nos objetos criados pela *factory*; O *controller* que é responsável por lidar com as funcionalidades padrões da aplicação (RF01, RF02, RF03, RF04, e RF09); E o *router* que define as rotas padrões.

Demonstrando a força dessa estrutura, precisa apenas observar como a funcionalidade de colaboração entre usuários (RF12) funciona. Ela é controlada por uma chave que deve ser passada no cabeçalho de todas as requisições. Essa chave, que possui o nome de "*collection*", é extraída por um *adapter* e entregue para o *repository* guardar no banco. O *repository* filtra todas as rotas a partir da "*collection*" e com isso nenhum outro arquivo precisa conhecer esse atributo. Segue abaixo uma figura apresentando todas as funcionalidades (rotas) reservadas da aplicação.

Figura 22: Coleção da aplicação no Postman



Fonte: elaborado pelo autor, 2021.

4.3.2 Aplicação cliente

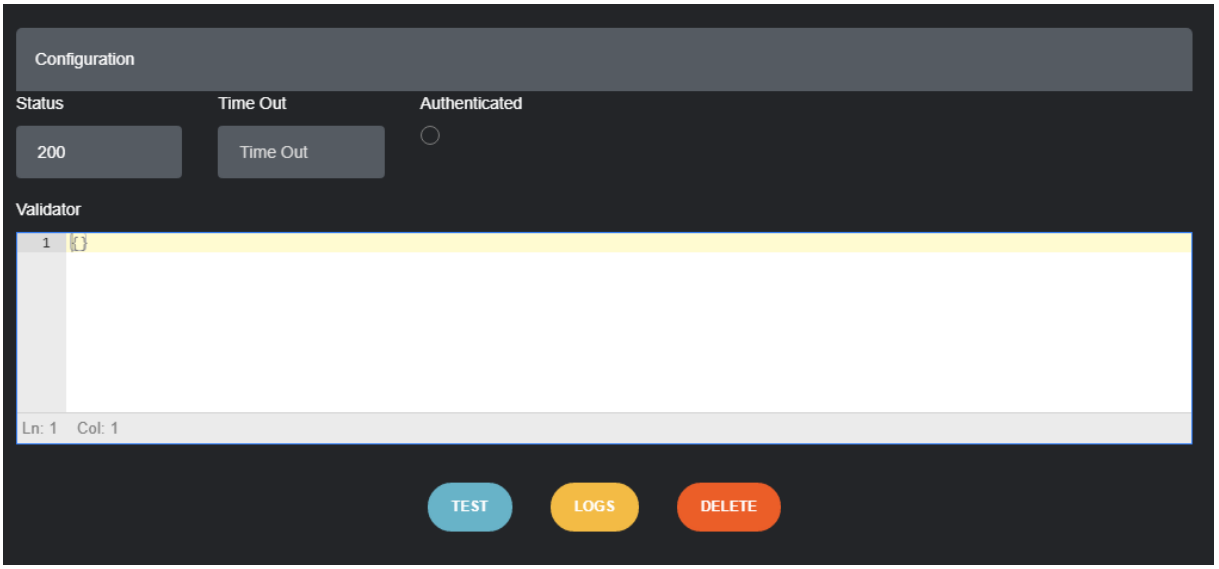
Com a maior parte da aplicação coberta nos tópicos 4.2.2 e 4.3.1, apenas alguns pontos ficam em aberto. O cadastro e edição possuem um editor de JSON para facilitar o preenchimento dos campos de resposta e corpo, assim como possuem campos opcionais guardados dentro do componente expansível de configuração.

Nesses campos é possível configurar qual o status que será usado para no retorno; O tempo de espera para o servidor retornar a resposta; Se a rota em questão precisa ter autenticação; E um outro campo JSON para configurar as validações da requisição.

Figura 23 - Campos para descrição, resposta, e corpo de uma rota

Fonte: elaborado pelo autor, 2021.

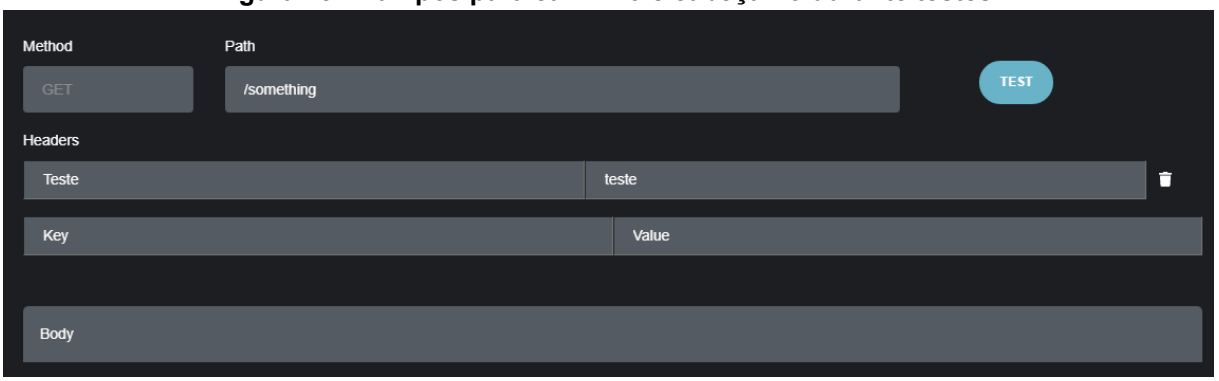
Figura 24 - Campos opcionais de configuração



Fonte: elaborado pelo autor, 2021.

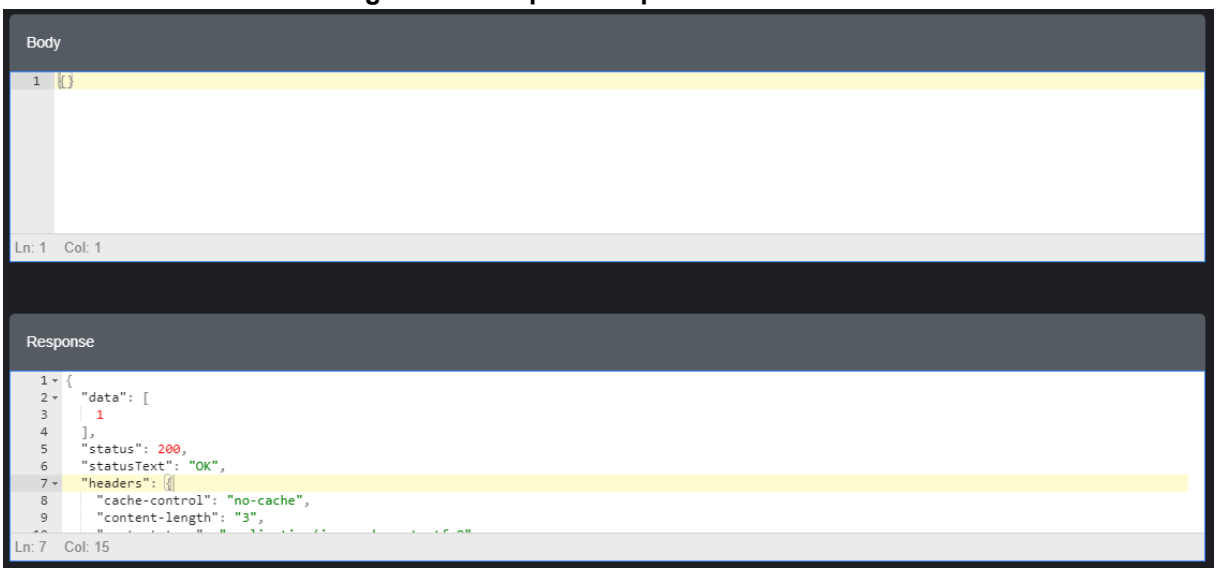
Ao testar uma rota é possível alterar o caminho, ou "*path" como é apresentado,* para poder adicionar "*queries*" ou algum outro parâmetro. Também é possível alterar o corpo e adicionar propriedades no cabeçalho. A resposta do servidor virá dentro do campo "*Response*" com informações relevantes.

Figura 25 - Campos para caminho e cabeçalho durante testes



Fonte: elaborado pelo autor, 2021.

Figura 26 - Corpo e Resposta de um teste



Fonte: elaborado pelo autor, 2021.

Para importar rotas do Swagger ou Postman, só é necessário abrir a modal da Figura 17 e preencher o link do Swagger, ou adicionar uma coleção exportada do Postman. Os campos são exclusivos, se preencher o *link* do Swagger ele terá prioridade sobre o arquivo do Postman.

Na Figura 18 as rotas da aplicação são exportadas para um arquivo JSON, um arquivo estruturado para ser uma coleção válida do Postman. Ao apertar o botão "*DOWNLOAD*" será gerado o arquivo e o navegador irá abrir uma janela para o usuário escolher onde deseja salvar.

Nas configurações (Figura 19) é apresentada ao usuário dois campos, um estará bloqueado e contém o identificador do usuário, o outro estará vazio para que o usuário possa digitar. Ao adicionar um valor no campo "*Current Identifier*" e salvar, as rotas serão recarregadas e listadas a partir deste novo identificador. Utilizando-se dessa mecânica um usuário pode copiar seu identificador, apertando o botão de copiar ao final do campo "*User Identifier*", e fornecer a um colega para terem acesso às mesmas rotas.

Esse identificador é gerado uma única vez e salvo para permanecer fixo enquanto o usuário não limpar as informações do navegador.

5 CONSIDERAÇÕES FINAIS

Este projeto, desenvolvido utilizando de diversas tecnologias, foi idealizado para auxiliar os profissionais de TI na elaboração de suas próprias aplicações, fornecendo ferramentas de apoio para solucionar situações comuns no dia a dia. A partir de um teste de conceito, o sistema evoluiu para uma versão *MVP* (*Minimum Viable Product* ou Produto Viável Mínimo) que já poderia ser liberada em *alpha*.

Considerando os objetivos gerais e específicos a aplicação serviu ao seu propósito. Como sistema completo, ela ainda possui pontos a serem melhorados ou corrigidos, principalmente na aplicação cliente, que se encontra em um estágio de desenvolvimento imaturo. Para trabalhos futuros, há várias possibilidades, desde formas para transformar rotas em blocos de código, que funcionam como sugestões

para agilizar o desenvolvimento, até uma reestruturação completa nas interfaces do *Front End*.

Uma dessas próximas funcionalidades a serem implementadas seria uma camada para canais de notificações no *Back End*. Essa camada de notificações ficaria no nível de *application* e iria permitir que qualquer mudança nas rotas poderia notificar a todos os interessados. Utilizando-se dessa camada, poderia ser logadas toda e qualquer chamada no terminal, ou realizar mudanças, através de *Web Sockets*, em tempo real no *Front End*.

Por fim, a aplicação, durante seus testes de desenvolvimento, tornou-se uma ferramenta poderosa que poderia auxiliar muitos desenvolvedores e educadores em suas necessidades, e por isso, o desenvolvimento será continuado.

REFERÊNCIAS

BATTISTI, Júlio. **Criando aplicações em 3, 4 ou n Camadas**. Disponível em: https://www.juliobattisti.com.br/artigos/ti/ncamadas.asp. Acesso em: 26 Jun.2021.

BELANI, Gaurav. **Linguagens de Programação que você deveria aprender em 2020**. Disponível em: https://www.computer.org/publications/tech-news/trends/programming-languages-you-should-learn-in-2020. Acesso em: 26 Jun. 2021.

BRAZ, Aline. **Guia completo sobre metodologia científica**. Disponível em: https://doity.com.br/blog/metodologia-cientifica/. Acesso em: 1 Jul.2021.

CAVALCANTE, Pablo Henrique Aguiar. Introdução ao TypeScript: o que é e como começar? **Geek Hunter**, 2021. Disponível em: https://blog.geekhunter.com.br/introducao-a-typescript/#O_que_e_TypeScript. Acesso em: 26 Jun.2021.

COMPUTER SCIENCE ORG. Linguagens de Programação para computadores. **Computer Science**, 2021. Disponível em: https://www.computerscience.org/resources/computer-programming-languages/. Acesso em: 26 Jun.2021.

D'AVILLAR, Priscila. Frameworks: Você sabe o que são? E para que serve? **DNC**, 2019. Disponível em: https://dnc.group/blog/projetos/o-que-sao-frameworks/. Acesso em: 26 Jun.2021.

ENCICLOPÉDIA BRITANNICA (ed.). Arquitetura cliente servidor. **Britannica**, 2021. Disponível em: https://www.britannica.com/technology/client-server-architecture. Acesso em: 26 Jun.2021.

GRUPPETTA, Stephen. Steve Jobs 1995: “Everyone should learn to program computers, because it teaches you how to think”. **Code Today**, 2018. Disponível em: https://www.codetoday.co.uk/post/2018/07/25/steve-jobs-1995-everybody-should-learn-to-program-a-computer-because-it-teaches-you-how. Acesso em: 04 Jul.2021.

GUEDES, Júlio. Tutorial VueJs. **Geek Hunter**, 2020. Disponível em: https://blog.geekhunter.com.br/tutorial-vue-js> . Acesso em: 26 Jun.2021.

HAPI. Disponível em: https://hapi.dev/. Acesso em: 26 Jun.2021.

JORA, Cristi. Paper Dashboard. **Creative Team**, 2018. Disponível em: https://demos.creative-tim.com/vue-paper-dashboard/#/. Acesso em: 4 Mar.2021.

MACÊDO, Diego. **Arquitetura de Aplicações em 2, 3, 4, ou N Camadas**. Disponível em: https://www.diegomacedo.com.br/arquitetura-de-aplicacoes-em-2-3-4-ou-n-camadas/ . Acesso em: 26 Jun.2021.

MICROSOFT. Estilo de arquitetura de microsserviços. **Microsof**t, 2019. Disponível em: https://docs.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/microservices. Acesso em: 26 Jun.2021.

MICROSOFT. Estilos de arquitetura. **Microsoft**, 2019. Disponível em: https://docs.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/. Acesso em: 26 Jun.2021.

MOZILLA. CSS. **Mozilla**, 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/CSS . Acesso em: 26 Jun.2021.

MOZILLA. HTML: Linguagem de Marcação de Hipertexto. **Mozilla**, 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML. Acesso em: 26 Jun. 2021.

MOZILLA. **Javascript.** Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript . Acesso em: 26 Jun.2021.

MOZILLA. **Javascript Tutoriais**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 26 Jun.2021.

MOZILLA. **Javascript:** Sumário**.** Disponível em: https://developer.mozilla.org/pt-BR/docs/Glossary/JavaScript. Acesso em: 26 Jun.2021.

MOZILLA. Uma visão geral do HTTP. **Mozilla**, 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview. Acesso em: 26 Jun.2020.

PACIEVITCH, Yuri. **HTML**. Disponível em: https://www.infoescola.com/informatica/html/ Acesso em: 26 Jun. 2021.

PICOLLO, Lucas. Vue JS: O que é, como funciona e vantagens. **Geek Hunter**, 2020. Disponível em: https://blog.geekhunter.com.br/vue-js-so-vejo-vantagens-e-voce/. Acesso em: 26 Jun.2021.

REVELO. Framework: saiba como usar e quais são os mais populares. **Revelo**, 2020. Disponível em: https://blog.revelo.com.br/o-que-e-framework-exemplos-e-aplicacoes/. Acesso em: 26 Jun.2020.

VICTÓRIA, Penélope. Banco de dados NoSQL: um manual prático e didático. **Geek Hunter**, 2019. Disponível em: https://blog.geekhunter.com.br/banco-de-dados-nosql-um-manual-pratico-e-didatico/. Acesso em: 04 Jul.2021.

VUE JS. Disponível em: https://vuejs.org/. Acesso em: 26 Jun. 2021.