Proposta de uma API E-commerce Utilizando AdonisJs e WebSockets

Matheus Luiz Oliveira de Camargo, Paulo Eduardo Nogueira

Instituto Federal de Educação Ciência e Tecnologia de São Paulo – Campus Hortolândia, Avenida Thereza Ana Cecon Breda, N.º 1896, Vila São Pedro, Hortolândia – SP

matheus_luiz99@hotmail.com, nogueira.paulo@ifsp.edu.br

Abstract. *This article exposes the development of an e-commerce RESTful API. Research was carried out on Brazilian electronic commerce and to define which tools to handle in software development. According to Ebit Nielsen's expectation, e-commerce has to continue on the rise, after the considerable increase last year due to the Covid-19 pandemic. In 2021, the outlook in Brazil is for a growth of 26%. The objective is to have an application that will serve real-time data from an e-commerce and subsequently open a way to assemble the e-commerce front-end that will consume data from this platform. The API is presented in the article.*

Resumo. *Este artigo expõe o desenvolvimento de uma API RESTful de e-commerce. Realizou-se pesquisas a respeito do comércio eletrônico brasileiro e para definição de quais ferramentas manusear no desenvolvimento do software. Segundo expectativa da Ebit Nielsen, o e-commerce tem de continuar em alta, após o aumento considerável no ano passado devido à pandemia de Covid-19. Em 2021, a perspectiva no Brasil é de um crescimento de 26%. O objetivo é dispor uma aplicação que servirá dados em tempo real de um comércio eletrônico e subsequentemente abrir um caminho para montar o front-end do e-commerce que consumirá dados desta plataforma. A API é apresentada no artigo.*

1. Introdução

Com o crescimento da Internet a partir dos anos 2000, vários serviços começaram a ser oferecidos de forma *on-line*, dentre esses pode-se destacar o serviço de compras *on-line*, ou *e-commerce*.

Segundo expectativa da plataforma Ebit|Nielsen (E-COMMERCE BRASIL, 2020) em 2021, o *e-commerce* poderá avançar 26% em relação a 2020, movimentando aproximadamente cerca de 110 bilhões de reais. Em relação ao *e-commerce* brasileiro, a plataforma identifica um crescimento substancial. A Figura 1 apresenta a evolução do faturamento do *e-commerce* brasileiros nos últimos 20 anos. Como pode ser observado o primeiro semestre de 2020 apresenta uma evolução de 47% em relação ao primeiro semestre do ano anterior.

Após o isolamento social o comércio eletrônico cresceu cerca de 75%, de acordo com a análise da MasterCard (ALVES, 2021). No entanto, a infraestrutura tecnológica, na qual o ambiente de *e-commerce* está inserido, não é tão simples. Com o passar do tempo as ferramentas de desenvolvimento também evoluíram, isso se deve ao fato da necessidade de oferecer plataformas seguras e robustas para atender o alto número de

transações que há entre o cliente e o servidor durante as operações de compra *on-line*. Uma das formas de se atender essa necessidade é através do uso de API *Restful.*

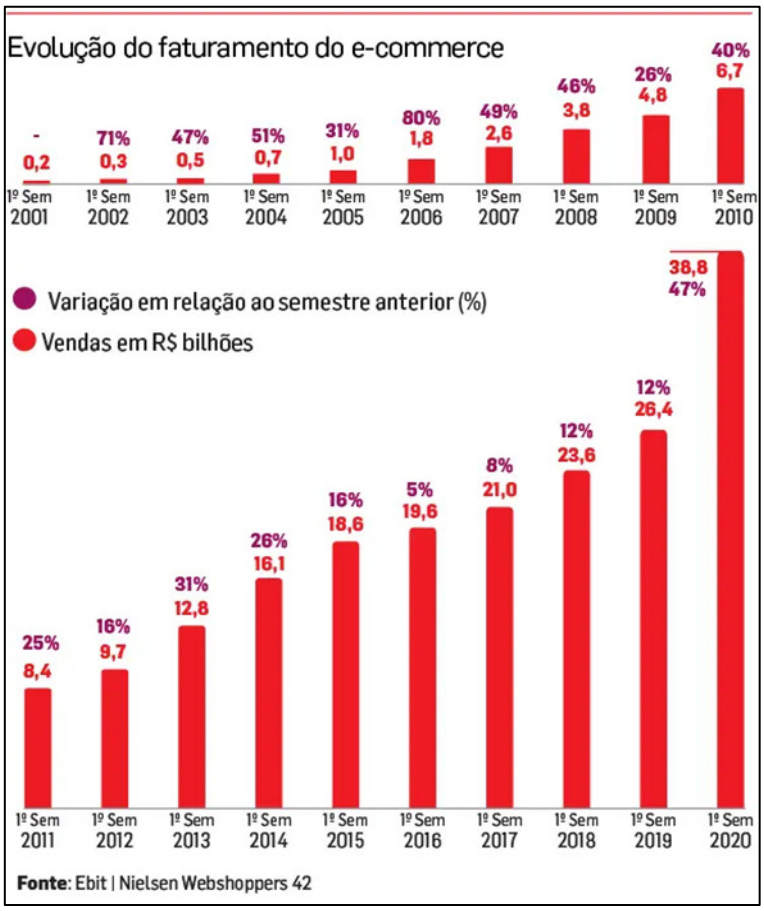

Figura 1. Evolução do faturamento do *e-commerce*.

Application Programming Interface (API), ou em português, Interface de Programação de Aplicações, se trata de um conjunto de rotinas e padrões estabelecidos e documentados por uma aplicação para que outras aplicações consigam utilizar as funcionalidades desta aplicação sem precisar conhecer detalhes da implementação do *software*. Ou seja, permitem a comunicação entre aplicações e entre os usuários.

Para isso, a API utiliza requisições HTTP que são responsáveis pelas operações básicas necessárias para manipulação dos dados. As principais requisições podem ser resumidas em criar dados no servidor, permitir leitura de dados no *host*, excluir informações e atualizar registros.

Desse modo delinearam-se os seguintes objetivos da pesquisa: o objetivo geral foi produzir uma API *RESTful* que sirva dados em tempo real de um comércio eletrônico, para gestores e consumidores. Mas para ter uma resposta mais eficaz, traçou-se os seguintes objetivos específicos: pesquisar sobre as tecnologias usadas nesse projeto, desenvolver o *layout* das páginas, desenvolver a API e acomodá-la em alguma empresa de hospedagem de sítio eletrônico. Posto isto, este projeto mostra a criação de uma API *RESTful*, com funcionalidades de comércio eletrônico, com dados em tempo real.

2. Trabalhos correlatos

Nesta Seção apresentam-se trabalhos correlatos ao tema deste projeto.

O trabalho elaborado por RIBEIRO (2021), tem como pauta o *e-commerce*. Afim de esclarecer o assunto, explicando de uma forma direta e simples para que o leitor consiga entender o que é um comércio eletrônico. Para isso, além de demonstrar alguns conceitos importantes, como Marketing Digital, é contado sobre a história de Eduarda, que venceu a crise do desemprego com sua família através da venda de máscaras. Vale destacar que, com a chegada da pandemia o *E-commerce* está tendo uma demanda significativa, seja por pessoas que decidiram abrir seu próprio negócio ou por pessoas que buscam suprimentos.

O trabalho produzido por SOUZA (2016) tem como objetivo apresentar resultados para averiguar se o uso de Node.js e *WebSockets* dentre outras opções, pode render algo mais benéfico para a escalabilidade e desempenho da aplicação. Para isso, foram realizados estudos em artigos, livros para conhecer as características das plataformas e execuções de testes comparativos para verificar o desempenho destas plataformas atuando em conjunto. Após o estudo, conclui-se que comparado ao conjunto Apache e PHP, o Node.js e *WebSockets* apresenta um desempenho superior quando submetido a um número elevado de usuários, mantendo a estabilidade do sistema com uma baixa degradação no tempo de resposta.

3. Fundamentação Teórica

Este tópico apresenta o referencial teórico utilizado para a execução deste trabalho, contendo conceitos importantes para compreensão do trabalho.

3.1. Scrum

Scrum é uma metodologia ágil para administração e planejamento de projetos de *software*, nele os trabalhos são divididos em ciclos (normalmente mensais) chamados de *Sprints*.

As funções a serem introduzidas em um projeto são mantidas em uma lista que é conhecida como *Product Backlog*. No início de cada *Sprint*, faz-se uma reunião, conhecida como *Sprint Planning Meeting* (PEREIRA et al, 2007), no qual o *Product Owner* designa os itens do *Product Backlog* e a equipe opta pelas tarefas que ela será capaz de realizar durante o *Sprint*. As tarefas escolhidas são transferidas do *Product Backlog* para o *Sprint Backlog*.

A cada dia de uma *Sprint*, a equipe faz uma breve reunião chamada *Daily Scrum*. O objetivo é espalhar o conhecimento sobre o que foi feito no dia anterior, reconhecer objeções e favorecer o trabalho do dia que se inicia.

Ao final de um *Sprint*, a equipe apresenta as funções implementadas em uma *Sprint Review Meeting*. Finalmente, faz-se uma *Sprint Retrospective* e a equipe começa a preparar a próxima *Sprint*. Assim recomeça o ciclo.

3.2. MVC (Model-View-Controller)

MVC, acrônimo de *Model-View-Controller*, é um modelo de projeto de *software* muito utilizado em aplicações *Web*, sendo encarregada por dividir cada subsistema ou módulo em camadas. Segundo Cassimiro (2010), cada camada é responsável por produzir independentemente uma lógica do negócio com a interface com o usuário.

A *Model* é o objeto de aplicação, a *View* é a interface que o usuário visualiza e o *Controller* funciona em relação às entradas de uma *View* e como as mesmas reagirão (GAMMA et al, 2000).

A camada *Model* controla a forma como os dados se apresentam por meio das funções, lógica e regras de negócios determinadas. É apenas na *Model* que as operações de *create, reader, update e delete* (CRUD), operações simples em um banco de dados, podem acontecer. A *View* é a camada de apresentação da aplicação, ela é a responsável por demonstrar as informações de forma visual ao usuário, através de um arquivo *html, xml*. A *Controller* é responsável por controlar todo o fluxo da aplicação, é o que move a aplicação, a lógica lida com os dados de entrada da *View* e define qual operação utilizará da camada *Model*.

O reuso de *software* é um dos benefícios ao empregar o MVC, segundo Sommerville (2012), a reutilização de *software* tornou-se a principal abordagem para a construção de sistemas *Web*. Quando desenvolvemos esses sistemas, analisamos em como conseguimos montá-los a partir de componentes e sistemas de *software* preexistentes.

3.3. Vue.Js

Vue.js, começou a ser desenvolvido em 2014 e, pelo ex-funcionário da Google, Evan You, que lidera atualmente a equipa de desenvolvimento

"*Vue.js* é um *framework* progressivo para a construção de interfaces de usuário. Ao contrário de outros *frameworks* monolíticos, *Vue* foi projetado desde a sua concepção para ser adotado incrementalmente" (VUE.JS, 2019). A biblioteca essencial é focada unicamente na camada visual, sendo fácil assumir e integrar com outros projetos desenvolvidos ou bibliotecas. Ele dispõe características como (WOHLGETHAN, 2018):

- Estrutura baseada em componentes, sendo relevante para facilitar a manutenibilidade do sistema.
- Práticas aceitas, oferecendo-as em sua documentação para uma configuração de desenvolvimento completa.
- Facilidade de integração com outras tecnologias e bibliotecas de maneira progressiva.
- Pequena curva de aprendizado.

3.4. AdonisJs

O *AdoniJs* é um *framework* mais compacto com uma série de funções prontas, profundamente baseado em *frameworks* de outras linguagens como o *Django, Laravel ou Rails*. Desta forma quando é instalado, concede computar uma estrutura MVC previamente formada. Então, o desenvolvedor utiliza a estrutura de ordenação, que inclui funcionalidades, como: autenticação, ORM, validação, envio de *e-mail* e *logging* (FERNANDES, 2017).

Seu propósito é fazer o desenvolvedor produtivo portanto sua abordagem é de escrever pouco código e produzir bastante. Mesmo sendo robusto, ele é leve, porque por parâmetro ele vem com uma estrutura mínima, desse modo o programador vai injetando os pacotes de acordo com sua demanda. Suas vantagens são:

- Estrutura de arquivos bem padronizada. Para aplicações maiores, este *framework* possui uma estrutura de pastas que facilita muito o desenvolvimento, trabalhando com *dependency injection* de forma simples.
- *ORM* extremamente poderoso para trabalhar com *SQL*. O *Lucid* como é nomeado, trabalha com qualquer banco de dados *SQL*, utilizando o padrão *active record*, onde são geradas *migrations* onde serão definidos os campos das tabelas

e toda vez que precisar incluir ou excluir um campo de alguma tabela só é necessário gerar uma *migration* e configurar a ação a ser executada.

• Estrutura para lidar com *WebSockets* (*real-time*).

• Funcionalidades pré-implementadas.

3.5. WebSockets

WebSockets é uma tecnologia que gera um canal bidirecional de tempo real entre um cliente e um *servidor*. É fundamentado no protocolo TCP e é uma real evolução do protocolo HTTP (RFC 2616). A API do *WebSockets* está sendo padronizada pelo W3C e a IETF (FETTE e MELNIKOV, 2011).

WebSockets possibilita um contato entre o navegador e o *servidor* que proporciona que os dados sejam enviados em qualquer direção a todo momento. Há diversos modos de *serverpush* em uso, mas o *WebSockets* compromete-se em substituir a maioria, se não todas, estas soluções (WANG et al, 2013).

WebSockets concede uma conexão bidirecional, direcionando a transmissão de mensagens de texto e dados binários entre o cliente e o *servidor*, e oferece uma série de serviços adicionais (GRIGORIK, 2013):

• Interoperabilidade com a infraestrutura HTTP existente.

• Comunicação orientada à mensagem e enquadramento de mensagens eficiente.

• Negociação com subprotocolo e extensibilidade.

3.6. Rest e Restful

A sigla *REST*, em português significa Transferência de Estado Representacional, é um estilo arquitetural para *softwares* hipermídia, que enfatiza a globalização das interfaces, a escalabilidade da comunicação entre os elementos e a instalação independente dos mesmos (FIELDING, 2000). Ele refere-se de um grupo de princípios e caracterizações necessários para a formação de um projeto com interfaces definidas.

Com o fim de ser considerada *RESTful*, a *API* necessita atender os princípios determinados no *REST*. São eles: *Uniform Interface* (Interface uniforme), *Stateless* (Sem estado), *Cacheable* (Armazenável em cache), *Client-Server* (Cliente-Servidor), *Layered system* (Sistema em camadas).

As seções seguintes delineiam cada um desses conceitos.

3.6.1. Interface Uniforme

Os modos são aplicados através de um grupo fixo de quatro operações: *PUT*, *GET*, *POST* e *DELETE*. O método *PUT* produz um novo modo, este modo pode ser apagado utilizando o modo *DELETE*. O método *GET* recobra o estado atual de um recurso em alguma representação. O método *POST* fornece um novo estado para um recurso.

3.6.2. Sem Estado

Um mesmo cliente é capaz de enviar várias requisições para o servidor, no entanto, cada uma delas devem ser autossuficientes, isto é, toda requisição deve conter todas as referências necessárias para que o servidor consiga entendê-la e processá-la adequadamente.

3.6.3. Armazenável em Cache

De modo que muitos clientes acessam um mesmo servidor, e muitas vezes buscando os mesmos recursos, é indispensável que estas respostas sejam capazes de ser cacheadas,

impossibilitando processamento desnecessário e aumentando consideravelmente o desempenho.

3.6.4. Cliente-Servidor

É a condição básica para uma aplicação *REST*. O propósito desta divisão é dividir a arquitetura e responsabilidades em dois ambientes. Assim, o cliente não se preocupa com funções do tipo: comunicação com banco de dados, gerenciamento de *cache*, *log*, etc. E o *servidor* não se preocupa com tarefas como: interface, experiência do usuário, etc. Permitindo, desse modo, a evolução independente das duas arquiteturas.

3.6.5. Sistemas em Camadas

O seu *software* deve ser composto por camadas, e elas devem ser fáceis de modificar, tanto para acrescentar mais camadas, quanto para removê-las. Por isso, um dos princípios desta restrição é que o cliente nunca deve acionar de forma direta o servidor da aplicação antes deve passar por um intermediador, no caso, pode ser um *load balancer* ou qualquer outra máquina. Isso garante que o cliente se preocupe exclusivamente com a comunicação com o intermediador e o intermediador fica encarregado por separar as requisições nos servidores. O fundamental é ficar esclarecido que gerando um intermediador, a sua estrutura fica mais adaptável as mudanças.

4. Metodologia

Em um primeiro momento, foi realizado uma pesquisa exploratória com o intuito de identificar qual a porcentagem que o *e-commerce* representa no comércio brasileiro. Além disso, foi realizado um estudo bibliográfico para identificar as ferramentas necessárias para o desenvolvimento de uma plataforma de *e-commerce* utilizando API.

Para gerir o projeto, utilizou-se a metodologia *Scrum*. Desta forma, o projeto iniciou-se com a definição do *Product Backlog*, que é feito pelo *Product Owner*, salienta-se que quem exerceu os papéis do *Scrum* foram apenas eu. A tabela 1 demonstra quais as funções que foram inseridas no *Product Backlog* para serem desenvolvidas

Com isso o trabalho foi dividido em *Sprints*, no início de cada uma delas eram escolhidas tarefas a serem desenvolvidas, e ao final eu verificava quais funcionalidades foram implementadas, para planejar a próxima *Sprint.*

Tabela 1. Demonstra as funções que foram colocadas no *Product Backlog* que precisavam ser desenvolvidas para API.

Funcionalidades
Crud Usuário
Crud dos Produtos
Crud Categorias
Crud Cupons
Funções de Cada Usuário
Receber Pedidos

4.1. Arquitetura da API

Com base no padrão MVC definido pelo *framework AdonisJs* podemos visualizar através da Figura 2 uma representação esquemática do funcionamento da API *RESTful* quando esta recebe um pedido HTTP do usuário.

Na Figura 2 podemos visualizar os componentes que fazem parte do projeto e como estes comunicam entre si no processo. Em seguida é apresentada uma explicação detalhada do seu funcionamento.

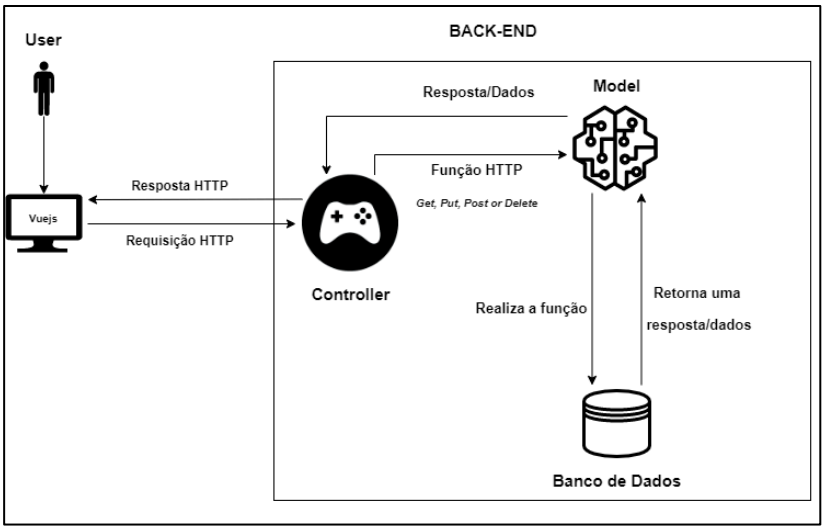

Figura 2. Arquitetura proposta para a API.

Quando um usuário acessa o site e executa uma ação, o *front-end* realiza uma requisição HTTP (*Get, Post, Put, Delete*) para a API, que recebe e através de um mecanismo de rotas, convertendo a requisição para um URL específico.

Dessa forma, é escolhido o método correspondente a URL para invocar o *Controller* que confere se tem alguma regra de negócio envolvida, como por exemplo se o usuário precisa estar logado para fazer aquela determina ação, e implementa de fato a lógica de negócio. Lembrando que, algumas rotas tem validações de dados para prevenção de erros e confiabilidade da API.

Consequentemente o *Controller* requisitado comunica o *Model*, e como o *Model* é a camada que faz a manipulação dos dados, ele faz uma comunicação com o banco de dados de acordo com a requisição. Sendo assim, ele pode criar, atualizar, excluir ou apenas buscar os dados.

Posto isso, o *Model* devolve para o *Controller* o que lhe foi solicitado, e assim o *Controller* transmite uma resposta HTTP através de um arquivo *json* para o *front-end*, que renderiza as informações ao usuário.

5. Desenvolvimento

5.1. Infraestrutura da API

5.1.1. Front-End

Para a construção da interface gráfica do usuário deste trabalho, foram utilizadas as linguagens HTML5, CSS3 e o *framework Vue.js*.

O HTML5, foi implementado com o objetivo de desenvolver a estrutura das páginas da aplicação, para que, seja possível a visualização da mesma no navegador. Para incorporar estilo ao documento *web* da API, foi inserida a linguagem *Cascading Style Sheets*, que é mais conhecida pela sua sigla CSS. Através de sua sintaxe simples, foi possível criar uma interface gráfica sofisticada para o usuário.

O *Vue.js*, foi incorporado ao projeto com o propósito de criar um *front-end* que será atualizado sempre que houver uma mudança nos dados da API.

5.1.2. Back-End

Com o objetivo de desenvolver as regras de negócio do *software*, ou seja, as funcionalidades do programa, fez-se o uso da plataforma *Node.js*, do *Node Package Manager* (NPM), da API *WebSocket*s, do *framework AdonisJs* (ADONIS, 2022) e para gerenciamento do banco de dados o sistema *MySQL*.

O *Node.js* neste projeto, foi instalado com o intuito de interpretar todo a codificação em *Javascript*, ou seja, ele foi utilizado como ambiente para execução das funcionalidades no *back-end*. E aplicando seu modelo de orientação a eventos e operações de I/O não bloqueantes, foi possível criar a aplicação em tempo real com intensa troca de dados entre cliente e *servidor*, de forma mais eficiente. (PEREIRA, 2014)

A figura 3, demonstra um pedaço da construção do *Controller* de cupom. Nela podemos observar que os *if's* estão averiguando por quem o cupom deverá ser utilizado.

```
1   // Clients e Products ( Pode ser utilizado somente em produtos e clientes específicos )
2       if(can_use_for.product && can_use_for.client){
3
4           coupon.can_use_for = 'product_client'
5
6   }
7   // Produto ( Pode ser utilizado apenas em produtos específicos )
8       else if(can_use_for.product && !can_use_for.client){
9
10          coupon.can_use_for = 'product'
11
12  }
13  // Clients ( Pode ser utilizado apenas por clientes específicos )
14      else if(!can_use_for.product && can_use_for.client){
15
16          coupon.can_use_for = 'client'
17
18  }
19  // Pode ser utilizado por qualquer cliente em qualquer pedido
20      else{
21
22          coupon.can_use_for = 'all'
23
24  }
```

Figura 3. Demonstra os If's do Controller de Cupom que averigua quem pode utilizá-lo.

O gerenciador de pacotes npm (*Node Package Manager*) foi instalado com a finalidade de possibilitar a instalação das dependências dos projetos e de algumas ferramentas que foram utilizadas, como exemplo o *AdonisJ*s e o *WebSockets*. Sempre que um projeto é gerado através deste gerenciador, é anexado um arquivo denominado *package.json*, que possui a relação dos pacotes instalados no ambiente. Na figura 4, é apresentado um exemplo de conteúdo deste arquivo, podemos observar qual o nome do projeto, a versão que ele está, as dependências que vão ser utilizadas nele, dentre outras coisas.

Para que o programa tenha uma comunicação em tempo real, foi preciso utilizar-se da API *WebSockets*. Visto que, com esta tecnologia temos uma comunicação bidirecional, ou seja, os dados podem ser enviados e recebidos ao mesmo tempo.

```
{
    "name": "appteste",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "test": "echo "Erro: no test specified" && exit 1"
    },
    "author": "",
    "license": "ISC",
    "dependencies": {
        "bootstrap: "^4.5.2"
    }
}
```

Figura 4. Exemplo de conteúdo de um arquivo package.json.

O *framework AdonisJs*, foi utilizado pois ele nos permite focar na regra de negócio da aplicação sem perder muito tempo com o funcionamento por trás de funcionalidades comuns como manipulação do banco de dados, envio de *e-mail*, autenticação etc. A figura 5 apresenta o código para instalar as ferramentas WebSockets e AdonisJs.

```
npm i --save adonis-websocket
npm i -g @adonisjs/cli
```

Figura 5. Código para instalação das ferramentas WebSockets e AdonisJs.

O *MySQL* é um sistema de gerenciamento de banco de dados, rápido e importante. Por esse motivo, ele foi implantado para armazenar e gerenciar os dados da aplicação.

5.2. Realização da API

5.2.1. Front-End

Para produzir o *front-end*, primeiramente pensou-se no *design* do *dashboard* de nossa aplicação, para facilitar nesta parte do processo, pesquisou-se alguns *templates* de painéis de gerenciamento no Pinterest.

Dessa forma, após as definições das telas do painel, criou-se as interfaces gráficas utilizando-se apenas das linguagens HTML5 e CSS3, o *Vue.js* não se fez necessário até então, pois esta primeira etapa era para exemplificar como ficaria o *front-end,* quando a aplicação estivesse *on-line*. A figura 6, ilustra como ficou a criação do *design* do *dashboard* da nossa aplicação.

Desta maneira ilustrou-se o painel de gerenciamento, consequentemente inicializou-se a construção definitiva do *front-end*, através do *framework Vue.js*, para que o *dashboard* se atualize conforme tenha mudanças no *back-end*.

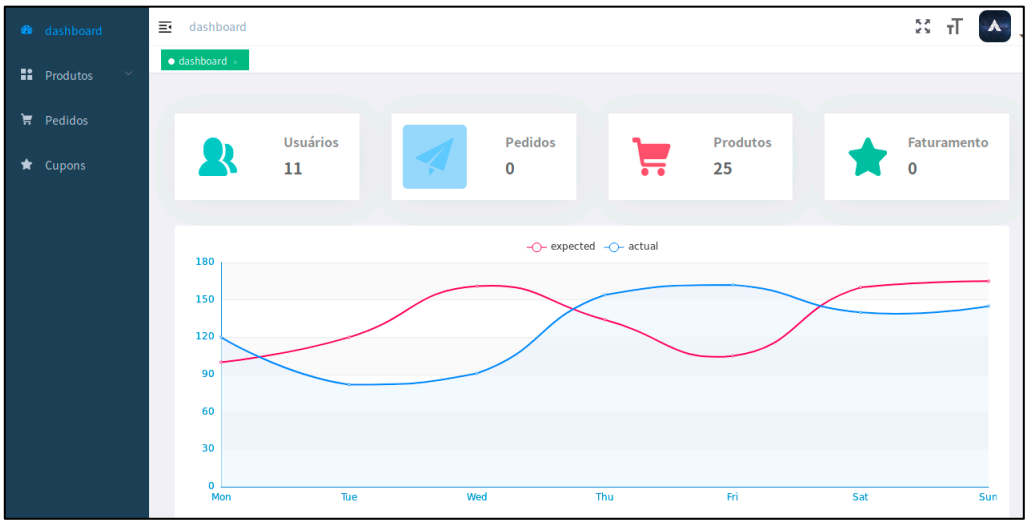

Figura 6. Visão geral da tela de *dashboard* da API.

5.2.2. Back-End

O trabalho de criação do back-end deu-se início com a reflexão e fixação de quais seriam as entidades para permanência dos dados.

Dessa forma, a tabela 2 demonstra as migrações que foram criadas e suas funcionalidades.

Dando continuidade ao projeto, foram criados os *Models*, que são classes responsáveis pela leitura, escrita e validação de qualquer dado e seus respectivos relacionamentos. Para ajudar na fase de teste criou-se *Model factories* e *Seeders*, que são dados pré-determinados que vão popular o banco de dados.

Tabela 2. Apresenta os nomes das *migrations* e suas funcionalidades.

Nome da Migration	Funcionalidade
User	Manusear os usuarios no banco de dados.
Token	Cria a tabela que irá gerar o token do usuário.
Create Permissions Table	Criação da tabela que contem as permissões.
Create Roles Table	Criação da tabela que armazena os papéis de usuários.
Create Permission Role Table	Gera a tabela com as devidas permissões de cada papél de usuário.
Create Permission User Table	Gera a tabela com as devidas permissões de cada usuário.
Create Role User Table	Cria a tabela que conecta o papel do usuario com o usuário.
Image Schema	Forma a tabela que armazena a imagem do usuário.
User Imagem FK Schema	Gera a tabela que conecta a imagem ao usuário.
Category Schema	Cria a tabela que irá armazenar as categorias.
Product Schema	Cria a tabela que irá armazenar os produtos.
Coupon Schema	Cria a tabela que irá armazenar os cupons.
Order Schema	Cria a tabela que irá armazenar os pedidos.
Order Item Schema	Gera a tabela de conectar os produtos ao pedido.
Coupon Order Schema	Cria a tabela de conectar o coupom ao pedido.
Coupon User Schema	Cria a tabela de conectar o coupom ao usuário.
Coupon Product Schema	Cria a tabela de conectar o coupom ao produto.
Password Reset Schema	Gera a tabela que vai auxiliar na redefinição de senha.

Posto isso, inicializou-se a construção dos *Controllers* de administradores e usuários, que é basicamente aonde é definido quais informações serão geradas, quais

regras serão acionadas e para onde as informações devem ir. Para organizar a aplicação desenvolve-se *Helpers*, posto que ele é uma classe estática, fora do grupo de *Controllers*, que possui lógica replicável para todo o restante do sistema. A figura 7, demonstra a codificação do *Helper* de gerenciamento de *uploads* únicos.

```
const manage_single_upload = async (file, path = null) =>{

    path = path ? path : Helpers.publicPath('uploads')

    //gera um nome aleatório
    const random_name = await str_random(30)
    let filename = `${new Date().getTime()}-${random_name}.${file.subtype}`

    //renomeia o arquivo e move ele para o path
    await file.move(path, {

        name: filename

    })

    return file

}
```

Figura 7. Código do *Helper* que gerencia os uploads únicos.

Então para transferir ao usuário uma resposta clara dos dados, foram criados os *Transformers*. Basicamente eles oferecem a flexibilidade de criar um formato de resposta json visto que através deles é possível fazer conversão de tipos, resultados de paginação e relacionamentos de aninhamento. Na reta final do desenvolvimento do *software*, foi integrado o *dashboard* e configurado o *WebSockets*, vale ressaltar que antes de qualquer novo progresso do trabalho, realizou-se testes para verificação e correção de *bugs*.

5.3. Apresentação da Interface

As interfaces do projeto foram projetadas com um *layout* simples e direto, para que o usuário tenha uma navegação facilitada referente as funcionalidades da API.

A tela de login (Figura 8) é de fácil entendimento, é para o cliente simplesmente inserir os seus dados nos campos solicitados, e prosseguir ao painel de controle (*dashboard*).

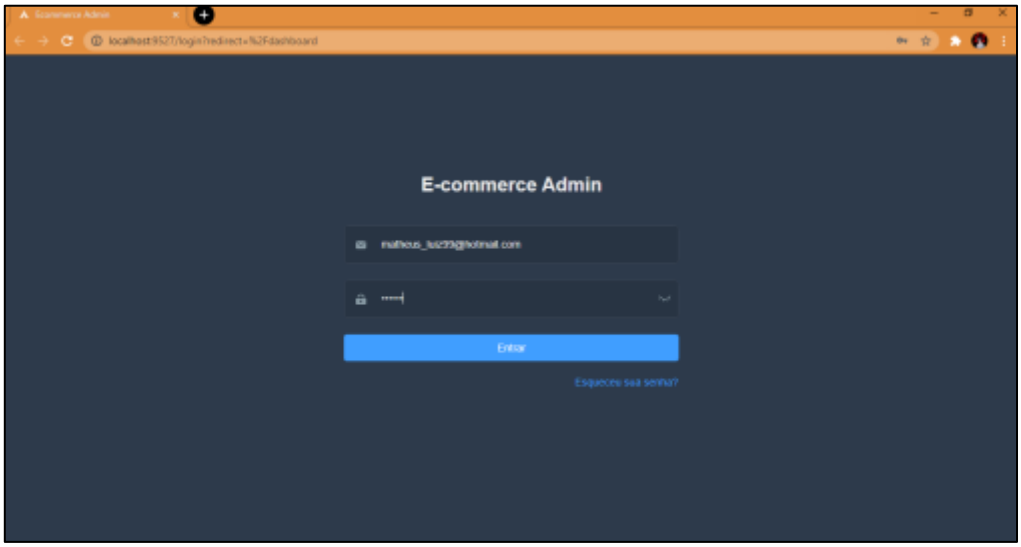

Figura 8. Tela de login da API desenvolvida.

A figura 9 demonstra que tem uma divisão de mercadorias por categorias, o que facilita ao usuário encontrar de forma mais eficiente o produto que procura.

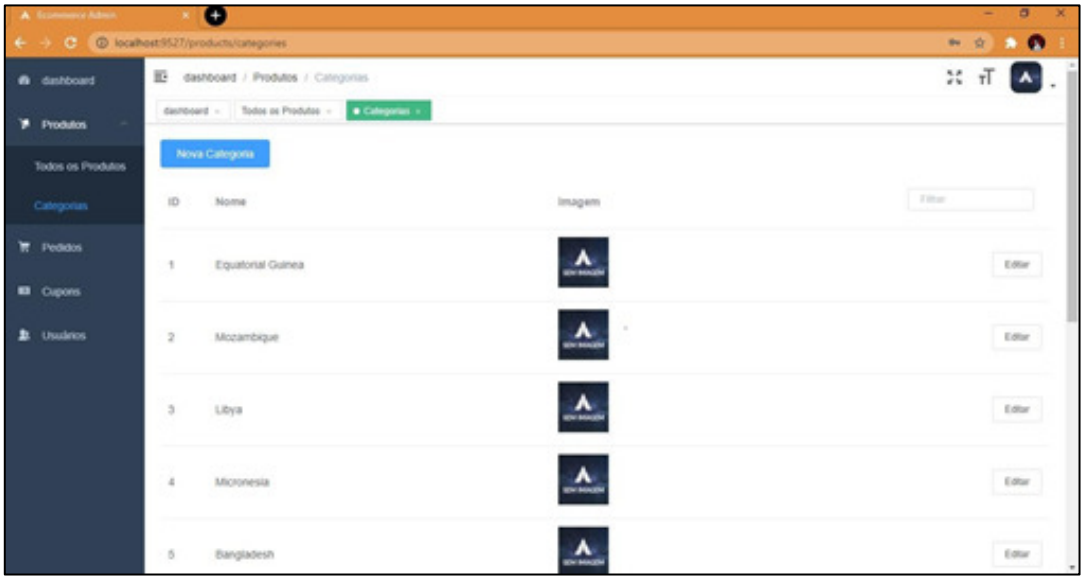

Figura 9. Tela de apresentação das categorias

De acordo com a figura 10, percebe-se que o cupom pode ter 3 tipos de desconto, o de porcentagem, um valor fixo ou até mesmo deixar o produto ou pedido que ele é aplicado gratuito. O mesmo também pode ser aplicado em cliente e produtos específicos e ter validade.

Pode-se observar a existência de um filtro no canto superior esquerdo da Figura 11, onde conseguimos pesquisar um pedido de forma rápida através do nome do cliente. Portanto, caso tenha acontecido algum *bug* com esse pedido, como uma mudança de *status* errada, o mesmo pode ser arrumado.

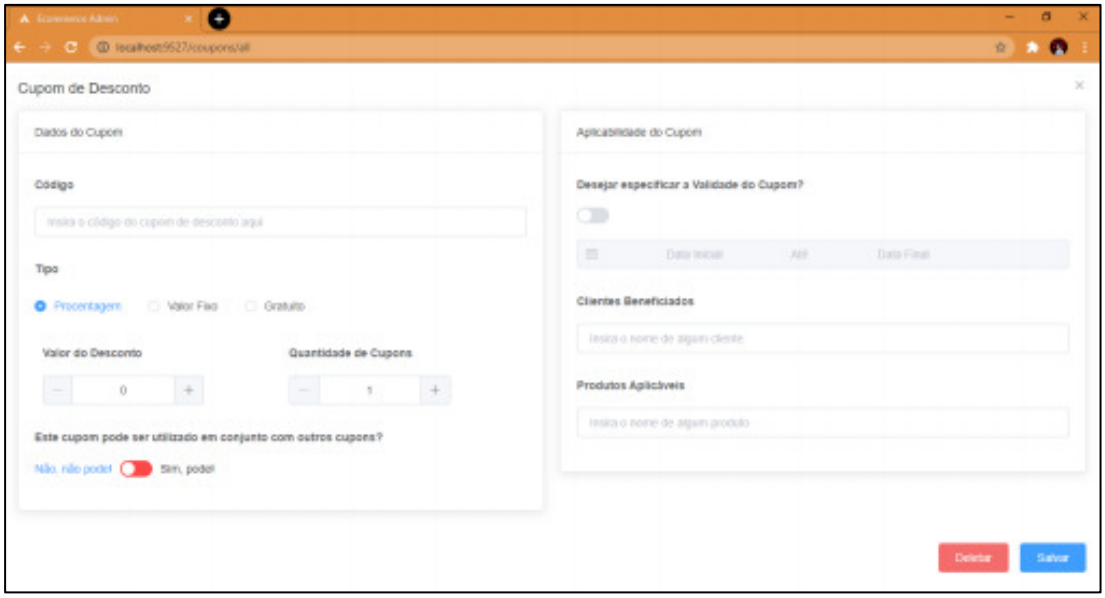

Figura 10. Visão da criação de cupom.

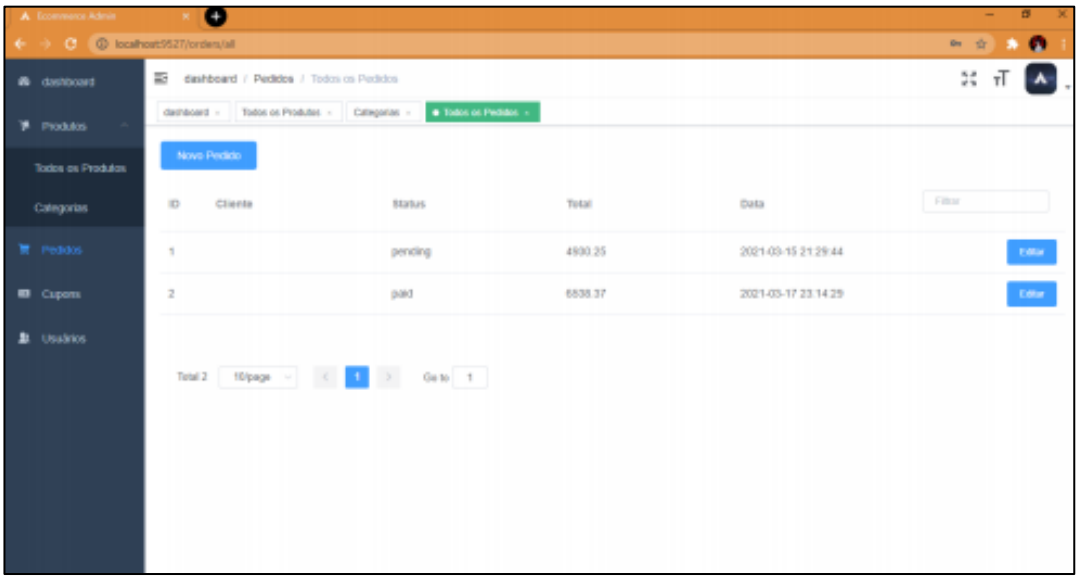

Figura 11. Visão geral dos pedidos.

A figura 12, demonstra a tela de produtos. Ela e a de usuários são parecidas, claramente o que se difere é a informação que vem em cada uma delas. Essencialmente conseguimos ver os dados em que ambas as páginas carregam, sendo possível fazer a edição e correção dos dois.

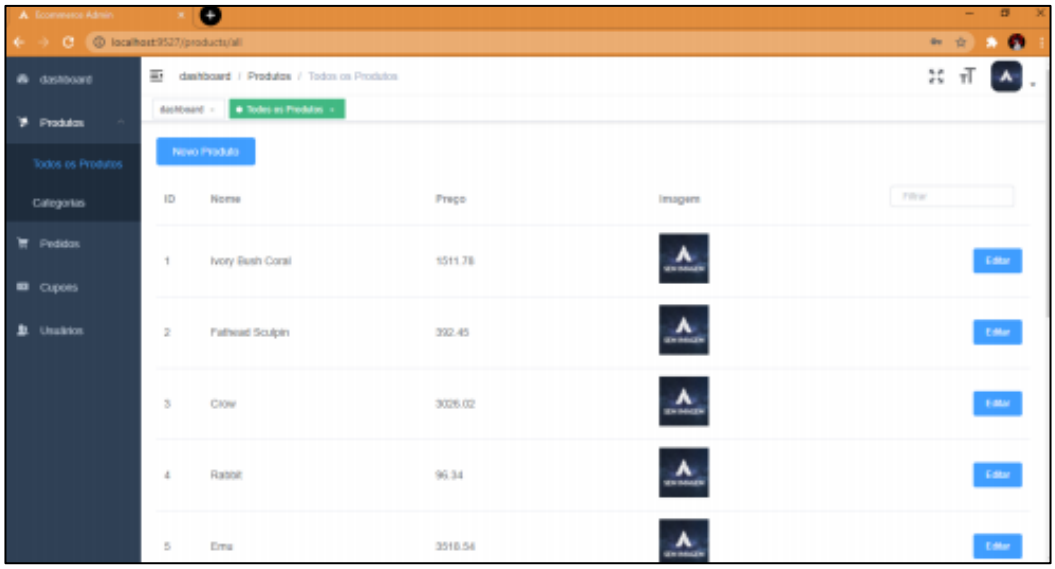

Figura 12. Tela que apresenta os produtos cadastrados na API

5.4. Teste

Para realizações de testes do projeto usou-se o aplicativo *Insomnia*, ele é uma ferramenta para projetar, depurar e testar APIs como um ser humano, não um robô, nele podemos criar solicitações, ver respostas e organizar espaços de trabalho, pastas, ambientes entre outras funções (INSOMNIA, 2020). Neste projeto, o *Insomnia* foi utilizado para realizar os testes de inserção, atualização, remoção e visualização dos dados da aplicação durante o desenvolvimento.

A figura 13 apresenta um teste de registro de usuário, a direita da imagem apresenta o *feedback* do teste realizado, isto significa o *status* da resposta (201 *Created*), o tempo decorrido (*Time* 1.67m) e seu tamanho (*Size* 151B). Esse tipo de informação é

muito relevante para certificar-se que sua API realmente está retornando os códigos recomendados na arquitetura *REST*, além de servir para verificar o desempenho da sua API.

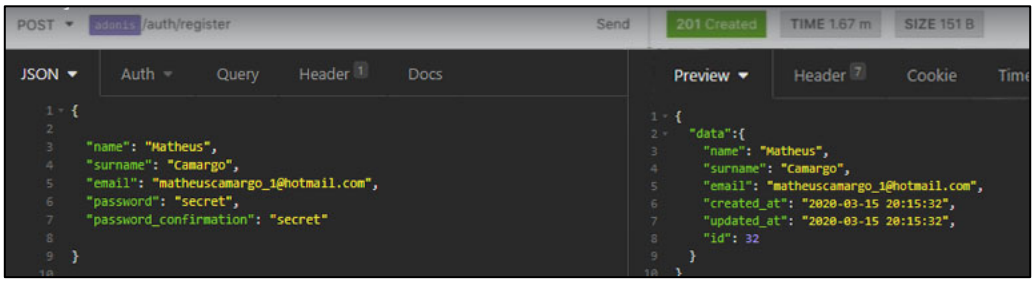

Figura 13. Teste feito no Insomnia para verificar o método de registro da aplicação.

6. Conclusão

O objetivo deste trabalho foi construir uma Interface de Programação de Aplicação (API) *RESTful* que sirva dados em tempo real de um *e-commerce*. Iniciou-se estudando sobre o crescimento do comércio eletrônico. Depois disto, realizou-se um estudo bibliográfico para constatar as ferramentas necessárias para o desenvolvimento do *software*.

Utilizando-se da metodologia ágil Scrum, construiu-se o *Product Backlog* aonde foi definido as funcionalidades que seriam desenvolvidas ao longo do projeto e, posteriormente, dividiu-se o projeto em 7 *Sprints*. Dessa forma, estabeleceu-se uma ordem de prioridade para construção das funcionalidades, fazendo com que consequentemente essas tarefas passam a compor nosso *Sprint Backlog*.

Na realização deste projeto foram utilizados conhecimentos das disciplinas de Banco de Dados I, Banco de Dados II, Desenvolvimento *Web*, Lógica de Programação, Análise Orientada a Objetos, Projetos de Sistemas I, Projeto de Sistemas II e Metodologia de Pesquisa Científica. Além disto, houve um estudo através da plataforma udemy com o objetivo de aprender ferramentas que não foram apresentadas no curso, tal como o *Vue.js*, *AdonisJs*, *WebSockets*, *Node.js* e *Insomnia*.

Como trabalhos futuros sugere-se a construção de uma funcionalidade de promoção, fazer atualizações ou até mesmo a criação do *front-end* de comércio eletrônico que consumirá os dados desta API. Também propõe-se realizar trabalhos que visam melhorar a segurança e o desempenho da aplicação.

Referências

ADONIS. **Adonis docs**. Disponível em: <https://adonisjs.com/docs/4.1>. Acesso em: 02. Fev. 2022.

ALVES, Pedro. **Com crescimento de 75% em 2020, E-commerce brasileiro chegou a representar 11% das vendas do varejo, revela estudo da Mastercard**. 2021. Disponível em: https://www.mastercard.com/news/latin-america/pt-br/noticias/comunicados-de-imprensa/pr-pt/2021/abril/com-crescimento-de-75-em-2020-e-commerce-brasileiro-chegou-a-representar-11-das-vendas-do-varejo-revela-estudo-da-mastercard/. Acesso em: 9 nov. 2021.

CASSIMIRO, M. H. de O. **Padrões arquiteturais e seus benefícios no processo de manutenção de software**. Belo Horizonte. 2010.

E-COMMERCE BRASIL. **E-commerce brasileiro deve crescer 26% em 2021, aposta Ebit|Nielsen**. 2020. Disponível em: https://www.ecommercebrasil.com.br/noticias/ebitnielsen-e-commerce-brasil-2021/.

Acesso em: 22 ago. 2021.

FERNANDES, D. **AdonisJS vs ExpressJS: Quando utilizar cada um?** [S. l.]: - RocketSeat, 2018. Disponível em: https://blog.rocketseat.com.br/adonis-vs-express/. Acesso em: 04 fev. 2022.

FIELDING, T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. 180 p. Tese (Doutorado). University of California, Irvine, 2000.

Fette I. e Melnikov A. (2011) "**RFC 6455 - The WebSocket Protocol**". Disponível em: <http://tools.ietf.org/html/rfc6455>. Acesso em: 30 jan. 2022.

GAMMA, Erich et al. **Padrões de Projeto: soluções reutilizáveis de software Orientado a Objetos.** Porto Alegre: Bookman, 2000.

INSOMNIA. **Insomnia**, 2022. Disponível em: <https://insomnia.rest/>. Acesso em: 17 fev. 2022.

PEIRERA, Caio. **Aplicações web real-time com Node.js**. Editora Casa do Código, 2014.

PEREIRA, Paulo et al. **Entendendo Scrum para Gerenciar Projetos de Forma Ágil**. **Mundo PM**, *[S. l.]*, p. 1–11, 2007. Disponível em: http://www.siq.com.br/DOCS/EntendendoScrumparaGerenciarProjetosdeFormaAgil.pdf.

RFC 791 (1981). **Internet Protocol.** Disponível em: <https://tools.ietf.org/html/rfc2616>. Acesso em: 01 fev. 2022.

RIBEIRO, Frantchesca Eduarda Soares. **2º Seminário de Tecnologia Gestão e Educação** – Faculdade Alcides Maya – Rua Dr. Flores, 396 – Centro Histórico de Porto Alegre – Rio Grande do Sul – novembro – 2019. *[S. l.]*, p. 5–10, 2021.

SOMMERVILLE, I. **Engenharia de Software**. 9ªed. São Paulo: Editora Pearson Brasil, 2012.

SOUZA, Thiago Oliveira De. **COMUNICAÇÃO REAL-TIME COM NODE.JS E WEBSOCKETS.** Angewandte Chemie International Edition**, 6(11), 951–952.**, *[S. l.]*, 2016.

VUE.JS. **Gerenciamento de Estado.** Guia do Vue.js, 2019. Disponível em: <https://br.vuejs.org/v2/guide/state-management.html>. Acesso em: 09 fev. 2022.

Wang, V., Salim, F., and Moskovits, P. (2013) "**The Definitive Guide HTML5 WebSocket**" Berkely., Apress

WOHLGETHAN, E. **Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js.** Bachelor's Thesis — IT department, Haw Hamburg, 2018

Documento Digitalizado Público

Artigo do Trabalho de Conclusão de Curso

Assunto: Artigo do Trabalho de Conclusão de Curso
Assinado por: Paulo Nogueira
Tipo do Documento: Outro
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Paulo Eduardo Nogueira**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 24/03/2022 11:39:11.

Este documento foi armazenado no SUAP em 24/03/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 925816
Código de Autenticação: 2f379adc7b