



**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO**  
**CENTRO UNIVERSITÁRIO NORTE DO ESPÍRITO SANTO**  
**COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO**

Arthur Ferraz  
Christian Oliveira  
Eric Rabelo  
Lorenzo Alves de Sousa  
Luiz Gustavo Falqueto Baptista

# **Microserviços**

## **Documentação Autenticação**

São Mateus, ES

2023

# Lista de ilustrações

Figura 1 – Diagrama de Casos de Uso . . . . .	6
Figura 2 – Diagrama do Caso de Uso "Efetuar Cadastro" . . . . .	7
Figura 3 – Diagrama do Caso de Uso "Efetuar <i>Login</i> " . . . . .	8
Figura 4 – Diagrama do Caso de Uso "Atualizar Dados do Próprio Usuário" . . . .	9
Figura 5 – Diagrama do Classes . . . . .	9
Figura 6 – Diagrama do Arquitetura . . . . .	10
Figura 7 – Rota do Login . . . . .	12
Figura 8 – Rota para o registro de um novo usuário . . . . .	13
Figura 9 – Rota para a busca de um usuário . . . . .	14
Figura 10 – Rota para a atualização de um <i>token</i> . . . . .	15
Figura 11 – Rota para a obtenção da lista de usuários registrados . . . . .	16
Figura 12 – Rota para a obtenção das informações de um único usuário . . . . .	17
Figura 13 – Rota para deleção de um usuário por id . . . . .	18
Figura 14 – Rota para realização do <i>logout</i> . . . . .	19
Figura 15 – Rota para realização da atualização de um usuário existente com base no id . . . . .	20

# Sumário

<b>1</b>	<b>VISÃO GERAL</b>	<b>4</b>
<b>1.1</b>	<b>Models e Controllers</b>	<b>4</b>
1.1.1	Models	4
1.1.2	Controllers	5
<b>2</b>	<b>DIAGRAMAS</b>	<b>6</b>
<b>2.1</b>	<b>Diagrama de Caso de Uso</b>	<b>6</b>
<b>2.2</b>	<b>Diagramas de Sequência</b>	<b>7</b>
2.2.1	Efetuar Cadastro	7
2.2.2	Efetuar Login	8
2.2.3	Atualizar dados do próprio usuário	8
<b>2.3</b>	<b>Diagrama de Classes</b>	<b>9</b>
<b>2.4</b>	<b>Diagrama de Arquitetura</b>	<b>9</b>
<b>3</b>	<b>GUIA DE BORDO</b>	<b>11</b>
<b>3.1</b>	<b>Processo de desenvolvimento</b>	<b>11</b>
<b>3.2</b>	<b>Pipeline de Deployment</b>	<b>11</b>
<b>4</b>	<b>ROTAS DOS ENDPOINTS</b>	<b>12</b>
<b>4.1</b>	<b>[POST] '/login'</b>	<b>12</b>
<b>4.2</b>	<b>[POST] '/register'</b>	<b>13</b>
<b>4.3</b>	<b>[GET] '/me'</b>	<b>14</b>
<b>4.4</b>	<b>[GET] '/refresh-token'</b>	<b>15</b>
<b>4.5</b>	<b>[GET] '/users'</b>	<b>16</b>
<b>4.6</b>	<b>[GET] '/users/id'</b>	<b>17</b>
<b>4.7</b>	<b>[DELETE] /users/id</b>	<b>18</b>
<b>4.8</b>	<b>[DELETE] /logout</b>	<b>19</b>
<b>4.9</b>	<b>[PUT] /users/id</b>	<b>20</b>
<b>5</b>	<b>TECNOLOGIAS</b>	<b>21</b>
<b>5.1</b>	<b>Swagger.io</b>	<b>21</b>
<b>5.2</b>	<b>Laravel v.10</b>	<b>21</b>
<b>5.3</b>	<b>PHP 8.1</b>	<b>21</b>
<b>5.4</b>	<b>Biblioteca Laravel Sanctum</b>	<b>21</b>
<b>5.5</b>	<b>PHP Unit</b>	<b>22</b>
<b>5.6</b>	<b>MySQL</b>	<b>22</b>

5.7	Draw.io . . . . .	22
6	BOAS PRÁTICAS DE DESENVOLVIMENTO . . . . .	23
7	CONCLUSÃO . . . . .	24
	REFERÊNCIAS . . . . .	25

# 1 Visão geral

Este presente documento visa descrever sobre o microserviço de autenticação do nosso grupo. É um componente essencial para o sistema, responsável por garantir a segurança e o controle de acesso aos recursos. Ele oferece recursos robustos de autenticação e autorização, permitindo que os usuários se autenticem de forma segura e acessem os recursos apropriados de acordo com suas permissões.

Além da autenticação, nosso microserviço também lida com a autorização. Ele gerencia as permissões dos usuários e garante que eles só tenham acesso aos recursos e funcionalidades apropriados. Isso é particularmente útil em sistemas com diferentes níveis de acesso, onde é necessário controlar quem pode executar determinadas ações ou visualizar certas informações.

Ao utilizar o microserviço de autenticação, outros componentes do sistema podem se integrar a ele para verificar a autenticidade dos usuários. Essa integração é feita por meio de chamadas de API seguras, onde o microserviço valida as credenciais fornecidas e retorna um token de autenticação válido, que pode ser usado posteriormente para acessar os recursos protegidos.

Nosso microserviço de autenticação foi projetado levando em consideração os princípios de segurança e escalabilidade. Ele emprega práticas recomendadas de segurança, como criptografia de senhas, gerenciamento seguro de tokens. Além disso, é altamente escalável, permitindo que o sistema lide com um grande número de solicitações de autenticação simultaneamente.

Em resumo, o microserviço de autenticação que desenvolvemos é uma parte crucial do sistema, fornecendo autenticação segura e controle de acesso a recursos. Com sua ampla gama de recursos de autenticação e autorização, ele oferece aos usuários uma experiência confiável e protegida, garantindo a integridade e a segurança do sistema como um todo.

## 1.1 Models e Controllers

A estrutura do código neste microserviço é organizada em Models e Controllers.

### 1.1.1 Models

Os Models são classes que representam entidades no banco de dados. Eles são responsáveis por definir a estrutura das tabelas do banco de dados, bem como as relações entre elas. No microserviço de autenticação, temos o Model User, que representa um

usuário do sistema. O Model User pode conter campos como nome, e-mail e senha.

### 1.1.2 Controllers

Os Controllers são responsáveis por receber as requisições HTTP e coordenar as ações correspondentes. Eles servem como intermediários entre as rotas da aplicação e os Models, manipulando os dados e fornecendo as respostas adequadas.

No microserviço de autenticação, temos dois Controllers principais:

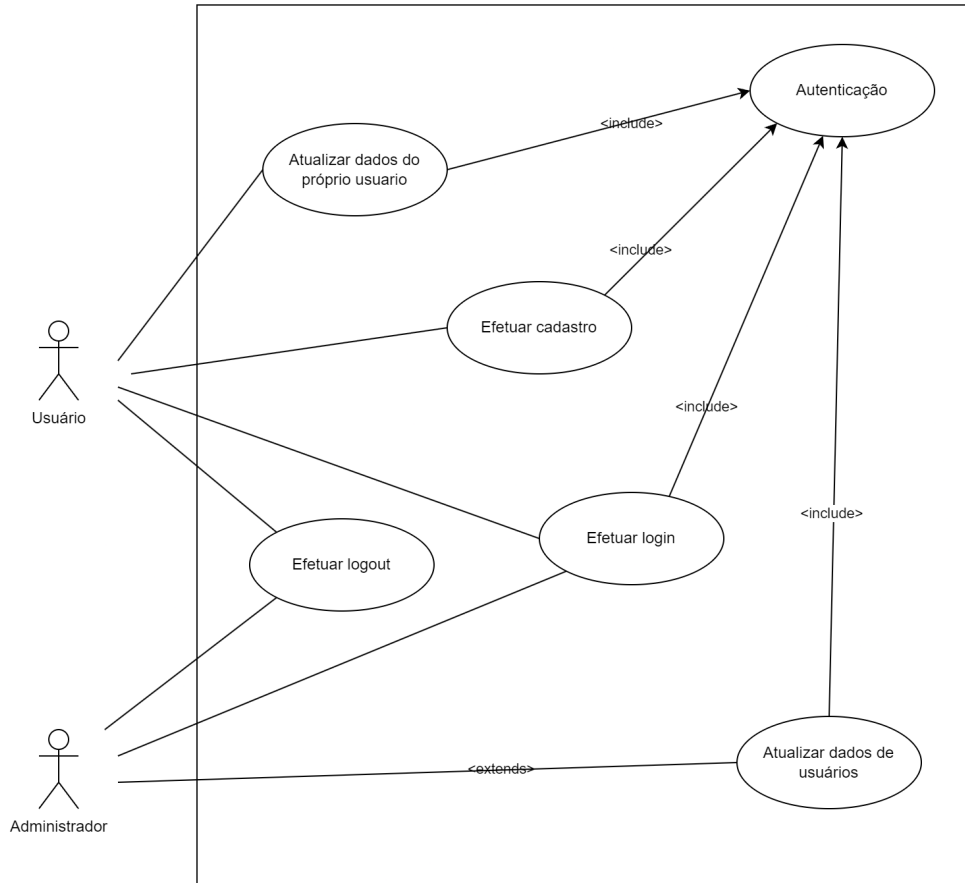
- UserController: responsável por operações relacionadas aos usuários, como criação, atualização, remoção e consulta de informações de usuários.
- AuthController: responsável por autenticação de usuários, incluindo registro, login e logout.

## 2 Diagramas

### 2.1 Diagrama de Caso de Uso

O diagrama de caso de uso do sistema apresenta dois atores principais: o Administrador e o Usuário. O Administrador possui o papel de gerenciar o sistema e tem acesso a funcionalidades adicionais, como "Atualizar dados de usuários". Por outro lado, o Usuário representa os usuários comuns do sistema. Os casos de uso incluem "Atualizar dados do próprio usuário", permitindo que os usuários atualizem suas informações pessoais; "Efetuar cadastro", para que novos usuários possam se registrar no sistema; "Efetuar logout", que permite aos usuários encerrarem suas sessões; "Efetuar login", para que os usuários possam acessar o sistema com suas credenciais; "Atualizar dados de usuários", exclusivamente disponível para o Administrador, permitindo a atualização das informações de outros usuários; e "Autenticação", que trata da validação e autenticação das credenciais dos usuários durante o processo de login.

Figura 1 – Diagrama de Casos de Uso



Fonte: Produção do Autor.

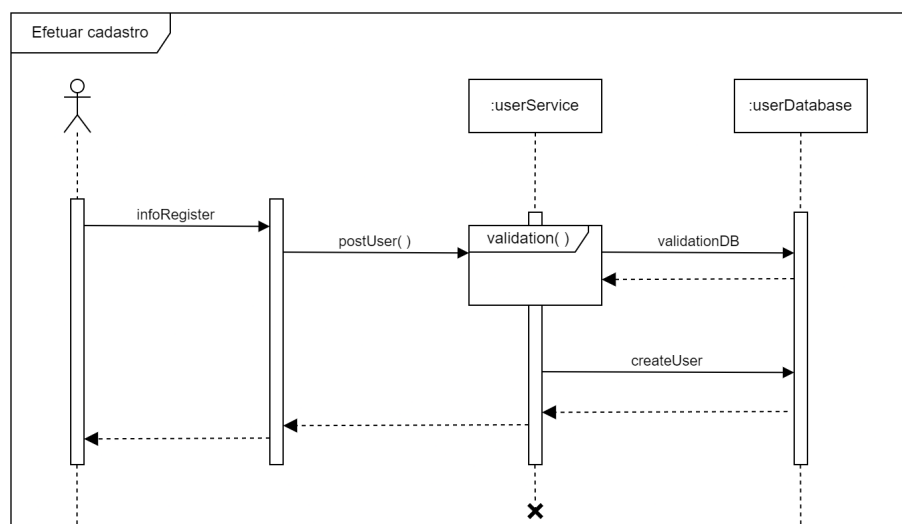
## 2.2 Diagramas de Sequência

Com base nos casos de uso identificados, foram selecionados três casos específicos que requeriam uma descrição mais detalhada da ordem de execução e sequência dos processos: "Efetuar cadastro", "Efetuar login" e "Atualizar dados do próprio usuário". Esses casos são cruciais para a interação dos usuários com o sistema, envolvendo etapas importantes que exigem uma compreensão precisa do fluxo de informações e das interações entre os atores e o sistema. Ao criar os diagramas de sequência para esses casos, foi possível representar visualmente as trocas de mensagens, as chamadas de métodos e a sequência de ações necessárias para a conclusão de cada processo. Esses diagramas fornecem uma representação clara e concisa da lógica e da ordem de execução, permitindo uma compreensão mais precisa do comportamento do sistema nessas funcionalidades específicas.

### 2.2.1 Efetuar Cadastro

Nesse contexto, o cliente faz uma requisição de cadastro. Durante esse processo ocorre a entrada de dados, e autenticação do usuário verificando se suas credenciais já existem. De acordo com a situação, uma resposta é enviada de volta para ele, permitindo cadastro no sistema ou reprovando-o em caso dele já existir. Esse fluxo garante que usuários cadastrados sejam únicos, tenham permissão adequada e acesso às funcionalidades disponíveis.

Figura 2 – Diagrama do Caso de Uso "Efetuar Cadastro"



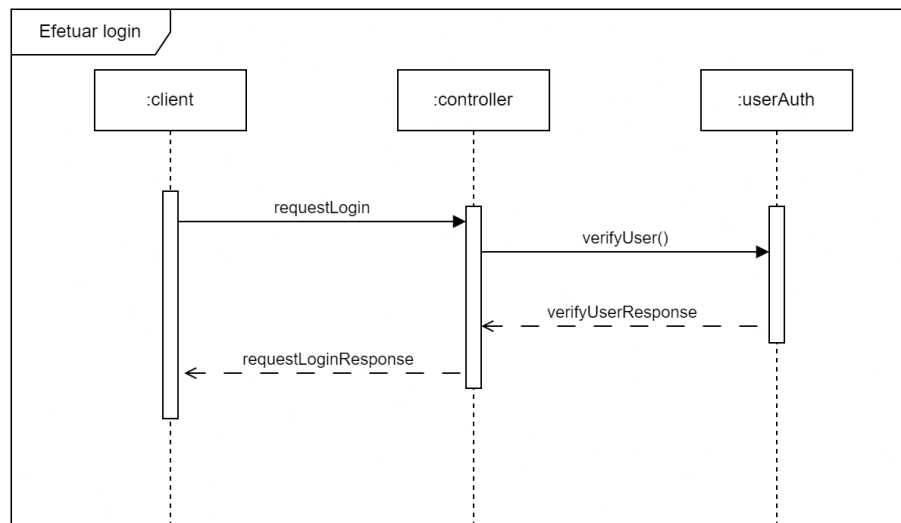
Fonte: Produção do Autor.



### 2.2.2 Efetuar Login

Aqui, o cliente faz uma requisição de login, solicitando acesso ao sistema. Durante esse processo, ocorre a autenticação do usuário, verificando suas credenciais. Se o usuário já estiver autenticado, uma resposta é enviada de volta para ele, permitindo o acesso ao sistema. Esse fluxo garante que usuários autenticados tenham permissão adequada e acesso às funcionalidades disponíveis.

Figura 3 – Diagrama do Caso de Uso "Efetuar *Login*"

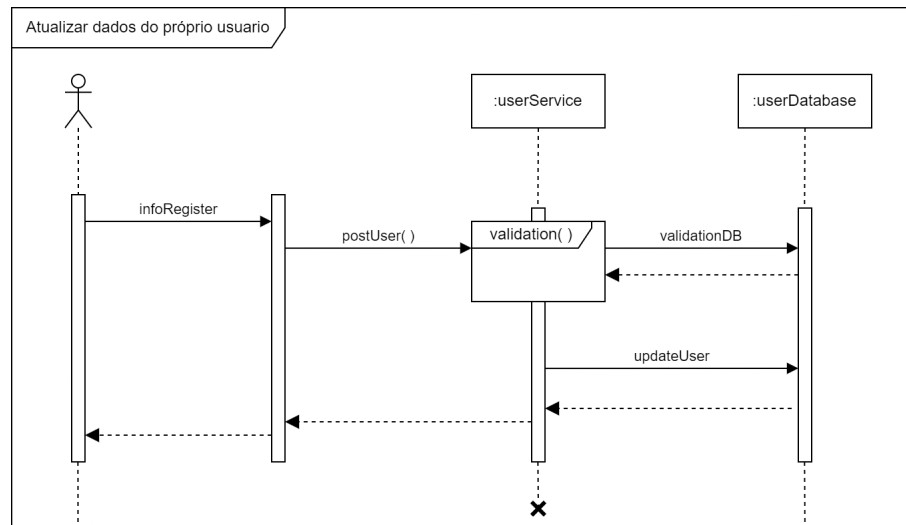


Fonte: Produção do Autor.

### 2.2.3 Atualizar dados do próprio usuário

Para este caso, quando o cliente deseja fazer uma requisição de alteração dos dados, ele envia os novos dados e, em seguida, é realizada a validação dessas informações. Caso o usuário esteja registrado no sistema, a validação é realizada com sucesso e os dados solicitados são alterados conforme especificado na requisição. Esse processo garante que apenas usuários autenticados e autorizados possam modificar seus dados, mantendo a integridade e segurança das informações armazenadas.

Figura 4 – Diagrama do Caso de Uso "Atualizar Dados do Próprio Usuário"

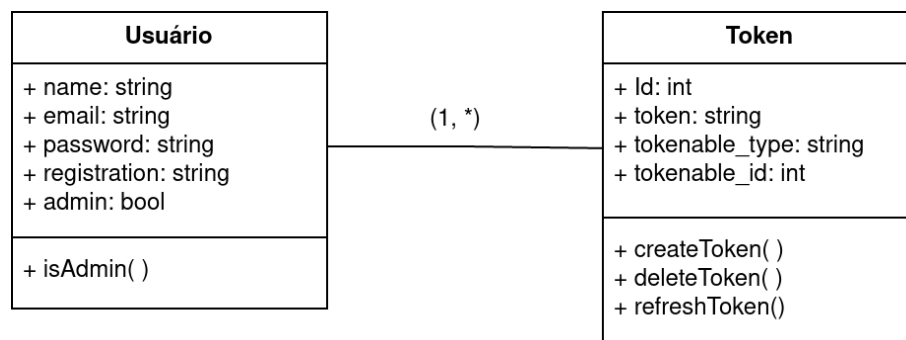


Fonte: Produção do Autor.

## 2.3 Diagrama de Classes

No nosso sistema, foram identificadas duas classes principais: a classe usuário e a classe token. Essas classes desempenham um papel fundamental na estrutura e funcionamento do sistema. A classe usuário representa os indivíduos ou entidades que interagem e utilizam o sistema, fornecendo informações e realizando ações específicas. Já a classe token refere-se aos elementos de autenticação e autorização, garantindo a segurança e a validação das operações realizadas pelos usuários. Ambas as classes são essenciais para garantir a integridade, confiabilidade e proteção dos dados e recursos do sistema.

Figura 5 – Diagrama do Classes



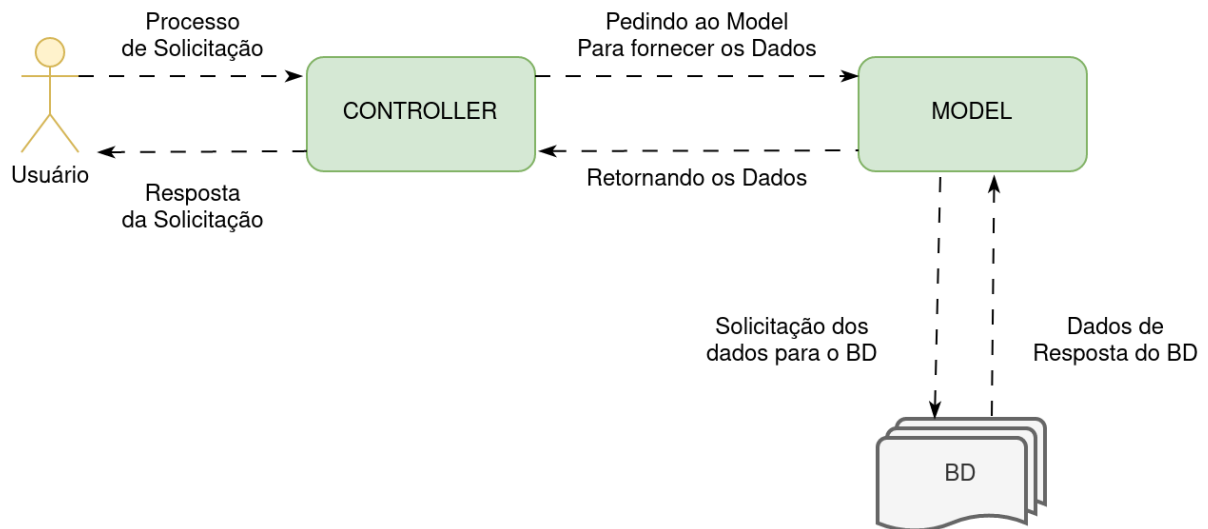
Fonte: Produção do Autor.

## 2.4 Diagrama de Arquitetura

O diagrama de arquitetura do sistema é baseado no padrão Modelo-Visão-Controlador (MVC), que é amplamente utilizado para projetar e desenvolver sistemas de software. No

entanto, neste estágio específico do projeto, decidimos focar principalmente nas partes Modelo e Controlador, deixando as Visões para uma fase posterior. Isso ocorre porque, neste momento, a implementação das Visões não é um ponto pertinente e pode demandar recursos adicionais. Ao adotar essa abordagem, nosso sistema se beneficia da clareza da separação de responsabilidades entre o Modelo, que lida com a lógica de negócios e o acesso aos dados, e o Controlador, que coordena as ações e interações do sistema. Essa arquitetura permite maior flexibilidade e facilita futuras iterações e adições de funcionalidades, incluindo a implementação das Visões quando se tornarem relevantes para os objetivos do projeto.

Figura 6 – Diagrama do Arquitetura



Fonte: Produção do Autor.

## 3 Guia de Bordo

### 3.1 Processo de desenvolvimento

Nosso processo de desenvolvimento seguiu uma abordagem ágil cujos ciclos, iterativos e incrementais, ocorriam no decorrer de uma semana. No entanto, houve um momento em que o time decidiu mudar de ferramentas e a migração ocorreu de forma desestruturada.

- Planejamento: Discutimos as tarefas mais importantes a serem realizadas na semana.
- Design: É feito um brainstorm das abordagens possíveis para resolução das tarefas escolhidas.
- Implementação: Alguém da equipe fica responsável pela codificação do design pensado, e nos preocupamos em garantir padronização e boas práticas de desenvolvimento.
- Testes: Os testes foram sendo planejados utilizando o software insomnia.
- Implantação: Nossa equipe utiliza workflows dedicados para testes unitários e para a criação e envio de imagens para o repositório.

### 3.2 Pipeline de Deployment

Nosso pipeline de deployment automatizado garante uma implantação suave e consistente do microsserviço em diferentes ambientes. Aqui está uma visão geral do pipeline:

- Compilação: O código é compilado para gerar os artefatos de implantação.
- Testes automatizados: Os testes automatizados são executados para garantir que o microsserviço esteja funcionando corretamente antes da implantação.
- Implantação em ambiente de teste: O microsserviço é implantado em um ambiente de teste, onde são realizados testes de integração e testes de aceitação para validar a funcionalidade.
- Revisão e aprovação: A equipe revisa os resultados dos testes e aprova a implantação em ambiente de produção.
- Implantação em ambiente de produção: O microsserviço é implantado no ambiente de produção, onde passa por mais testes e é monitorado para garantir a estabilidade e a disponibilidade.

## 4 Rotas dos endpoints

A API Auth - API MICRO é um serviço de autenticação e gerenciamento de usuários que fornece endpoints para registro, login, obtenção de informações do usuário, atualização, exclusão e logout. Ela é projetada para oferecer recursos de autenticação seguros e gerenciamento de usuários em um aplicativo ou sistema. A API possui as seguintes rotas principais:

Essas rotas fornecem funcionalidades essenciais para autenticação e gerenciamento de usuários, permitindo que aplicativos e sistemas implementem recursos de segurança e controle de acesso.

### 4.1 [POST] '/login'

Esta rota permite que um usuário faça login fornecendo suas credenciais de email e senha. Ela retorna um token de acesso que pode ser usado para autenticar solicitações futuras.

The image shows a Swagger UI interface for the POST /login endpoint. The interface is divided into several sections:

- Endpoint:** POST /login. Realiza login e retorna token de acesso.
- Parameters:** No parameters.
- Request body:** Required. Content type: application/json.
- Credenciais de login:** Example Value: 

```
{  "email": "user@example.com",  "password": "string"}
```
- Responses:**
  - 200:** Token de acesso. Example Value: 

```
{  "accessToken": {    "name": "John Doe",    "abilities": [      "a"    ],    "expiresAt": "2023-07-02T12:00:00Z",    "tokenableId": 123,    "tokenableType": "user",    "updatedAt": "2023-07-01T10:00:00Z",    "createdAt": "2023-07-01T10:00:00Z",    "id": 456  },  "plainTextToken": "random_token_example"}
```
  - 401:** Requisição não autorizada.
  - 500:** Erro interno no servidor.

Figura 7 – Rota do Login

## 4.2 [POST] '/register'

Essa rota permite que um novo usuário seja registrado. O usuário precisa fornecer seu nome, email, senha e informações de registro. Após o registro bem-sucedido, o servidor responde com uma mensagem de confirmação.

**POST** /register Registra novo usuário

Parameters Try it out

No parameters

Request body *required* application/json

Dados do usuário a ser registrado

Example Value | Schema

```
{
  "name": "string",
  "email": "user@example.com",
  "password": "string",
  "registration": "string"
}
```

Responses

Code	Description	Links
200	Usuário registrado com sucesso	No links
422	Entidade não processável	No links
500	Erro interno no servidor	No links

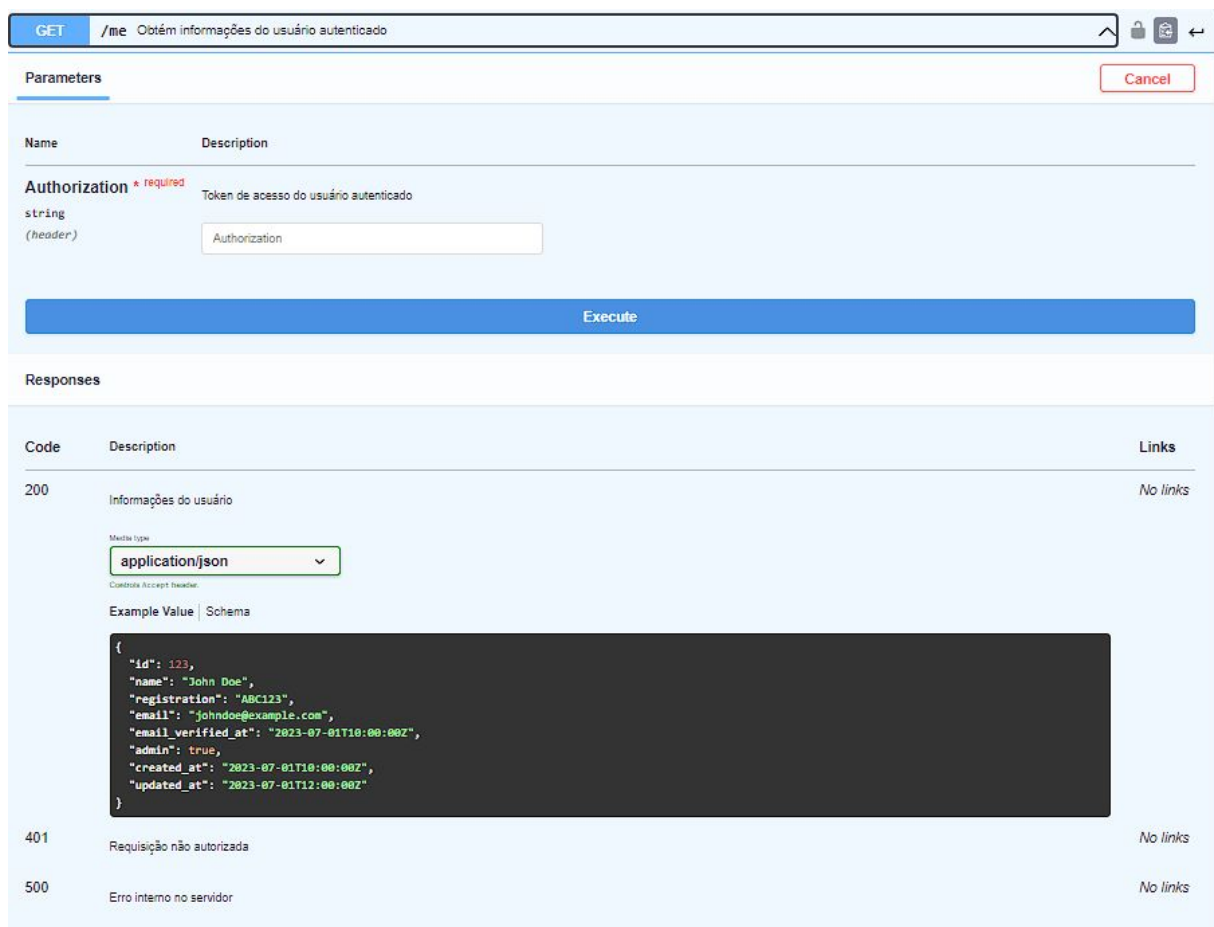
Extensions

Field	Value
x-openapi-router-controller	"swagger_server.controllers.auth_controller"

Figura 8 – Rota para o registro de um novo usuário

### 4.3 [GET] '/me'

Essa rota retorna informações do usuário autenticado. Ela requer que o token de acesso seja fornecido no cabeçalho da solicitação. As informações retornadas incluem o ID do usuário, nome, email, data de verificação do email, status de administrador, data de criação e data de atualização.



The screenshot displays a REST client interface for the endpoint `GET /me` with the description "Obtém informações do usuário autenticado".

**Parameters**

Name	Description
<b>Authorization</b> * required	Token de acesso do usuário autenticado
string (header)	<input type="text" value="Authorization"/>

**Execute**

**Responses**

Code	Description	Links
200	Informações do usuário	No links
401	Requisição não autorizada	No links
500	Erro interno no servidor	No links

Media type:

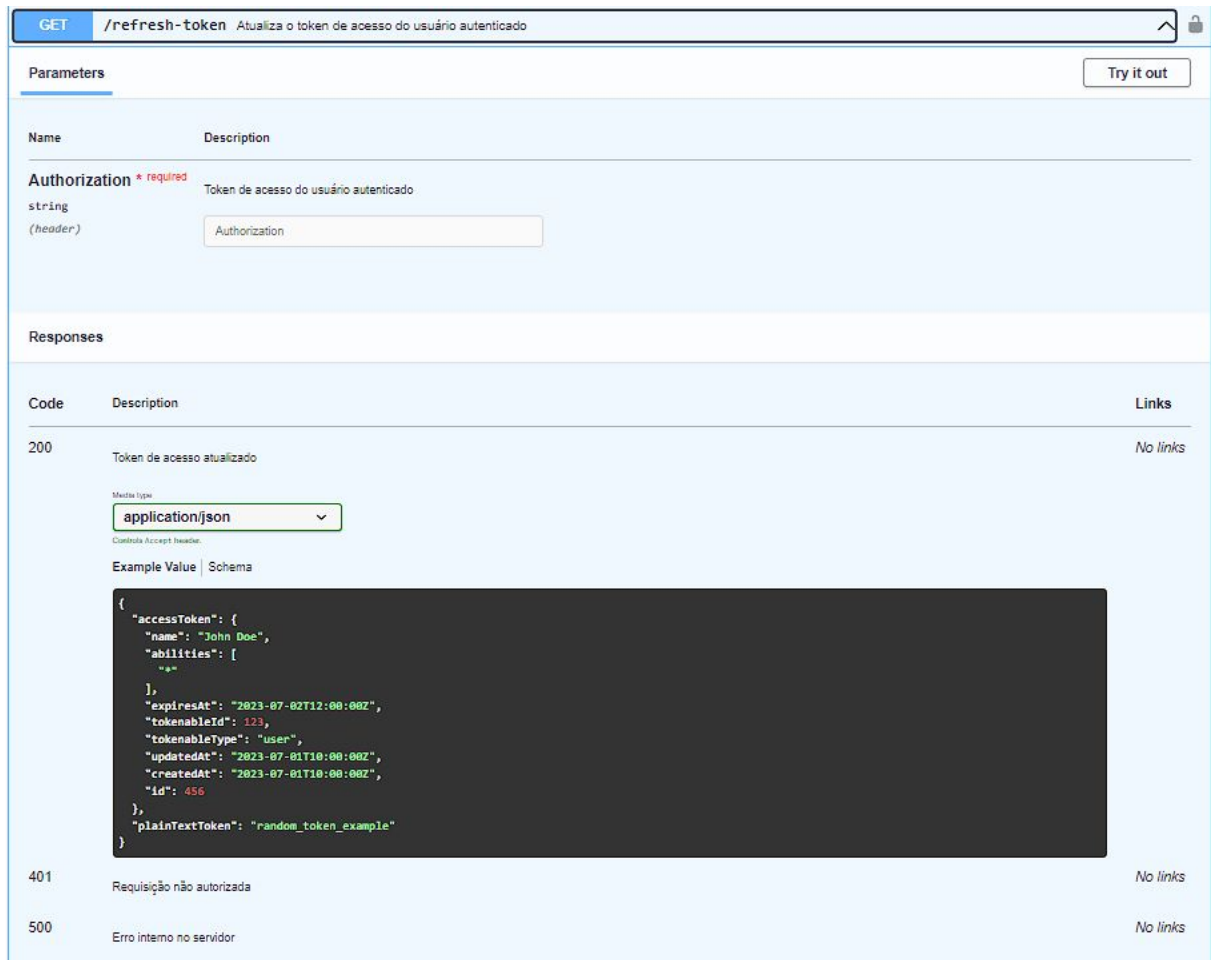
Example Value | Schema

```
{
  "id": 123,
  "name": "John Doe",
  "registration": "ABC123",
  "email": "johndoe@example.com",
  "email_verified_at": "2023-07-01T10:00:00Z",
  "admin": true,
  "created_at": "2023-07-01T10:00:00Z",
  "updated_at": "2023-07-01T12:00:00Z"
}
```

Figura 9 – Rota para a busca de um usuário

## 4.4 [GET] '/refresh-token'

Essa rota atualiza o token de acesso do usuário autenticado. O token de acesso atual deve ser fornecido no cabeçalho da solicitação. O servidor retorna um novo token de acesso atualizado.



GET /refresh-token Atualiza o token de acesso do usuário autenticado

Parameters

Try it out

Name	Description
<b>Authorization</b> * required	Token de acesso do usuário autenticado
string (header)	<input type="text" value="Authorization"/>

Responses

Code	Description	Links
200	Token de acesso atualizado	No links
401	Requisição não autorizada	No links
500	Erro interno no servidor	No links

Media type: application/json

Example Value

```
{  "accessToken": {    "name": "John Doe",    "abilities": [      ],    "expiresAt": "2023-07-02T12:00:00Z",    "tokenableId": 123,    "tokenableType": "user",    "updatedAt": "2023-07-01T10:00:00Z",    "createdAt": "2023-07-01T10:00:00Z",    "id": 456  },  "plainTextToken": "random_token_example"}
```

Figura 10 – Rota para a atualização de um *token*



## 4.5 [GET] '/users'

Essa rota obtém a lista de usuários registrados. É necessário fornecer o token de acesso no cabeçalho da solicitação. A resposta do servidor é uma lista de objetos que representam cada usuário, contendo informações como ID, nome, registro, email, data de verificação do email, status de administrador, data de criação e data de atualização.

The screenshot displays a REST client interface for the `GET /users` endpoint. The header bar indicates the method `GET` and the path `/users`, with a description "Obtém a lista de usuários". Below the header, the "Parameters" section shows a required `Authorization` header of type `string` with a value `Token de acesso do usuário autenticado`. The "Responses" section lists three status codes: `200` (Lista de usuários), `401` (Requisição não autorizada), and `500` (Erro interno no servidor). The `200` response is expanded, showing a media type of `application/json` and an example value of a JSON array containing one user object.

```
[
  {
    "id": 123,
    "name": "John Doe",
    "registration": "ABC123",
    "email": "johndoe@example.com",
    "email_verified_at": "2023-07-01T10:00:00Z",
    "admin": true,
    "created_at": "2023-07-01T10:00:00Z",
    "updated_at": "2023-07-01T12:00:00Z"
  }
]
```

Figura 11 – Rota para a obtenção da lista de usuários registrados

## 4.6 [GET] '/users/id'

Essa rota obtém informações de um usuário específico com base no ID fornecido. É necessário fornecer o ID do usuário no caminho da solicitação. O servidor retorna as informações do usuário correspondente.

The image displays the Swagger UI for the endpoint `GET /users/{id}`. The title bar indicates the method and path, along with a description: "Obtém informações de um usuário específico".

**Parameters**

Name	Description
<b>id</b> * required integer(\$int64) (path)	ID do usuário a ser obtido
<b>Authorization</b> * required string (header)	Token de acesso do usuário autenticado

Input fields for 'id' and 'Authorization' are provided. A "Try it out" button is located in the top right of the parameters section.

**Responses**

Code	Description	Links
200	Informações do usuário	No links
401	Requisição não autorizada	No links
404	Usuário não encontrado	No links
500	Erro interno no servidor	No links

For the 200 response, the media type is set to `application/json`. An example value is shown in a dark box:

```
{
  "id": 123,
  "name": "John Doe",
  "registration": "ABC123",
  "email": "johndoe@example.com",
  "email_verified_at": "2023-07-01T10:00:00Z",
  "admin": true,
  "created_at": "2023-07-01T10:00:00Z",
  "updated_at": "2023-07-01T12:00:00Z"
}
```

Figura 12 – Rota para a obtenção das informações de um único usuário

## 4.7 [DELETE] /users/id

Essa rota exclui um usuário existente com base no ID fornecido. É necessário fornecer o ID do usuário no caminho da solicitação. O servidor responde com uma mensagem indicando que o usuário foi excluído com sucesso.

**DELETE** /users/{id} Exclui um usuário existente

[Try it out](#)

Name	Description
<b>id</b> * required integer(\$int64) (path)	ID do usuário a ser excluído
<b>Authorization</b> * required string (header)	Token de acesso do usuário autenticado

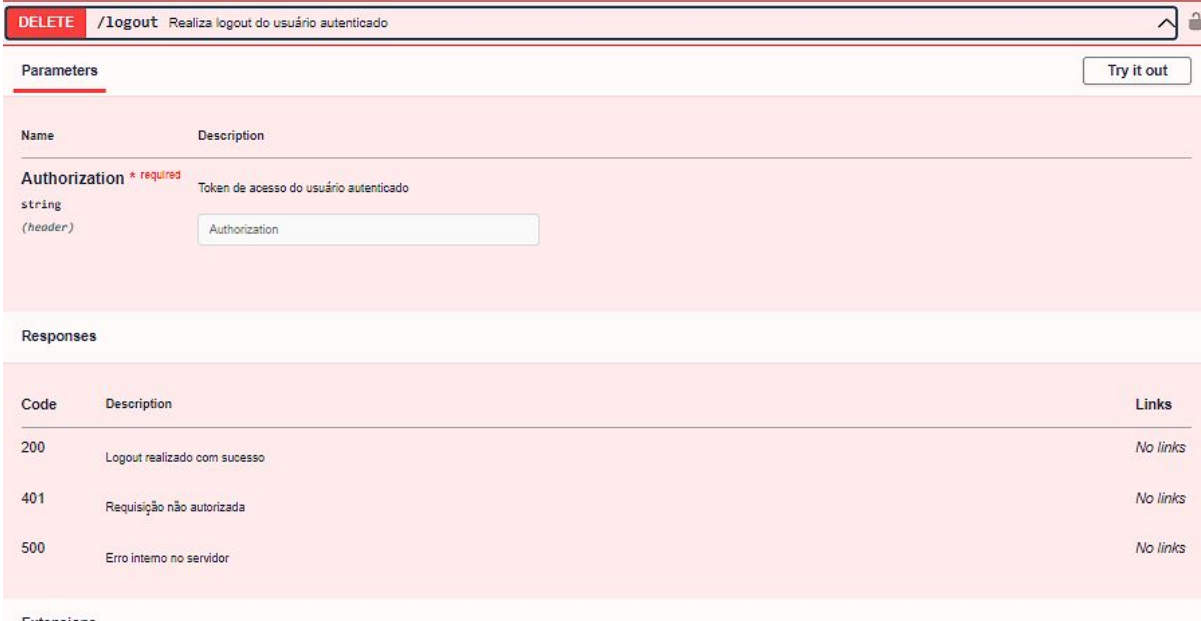
**Responses**

Code	Description	Links
200	Usuário excluído com sucesso	No links
401	Requisição não autorizada	No links
404	Usuário não encontrado	No links
500	Erro interno no servidor	No links

Figura 13 – Rota para deleção de um usuário por id

## 4.8 [DELETE] /logout

Essa rota realiza o logout do usuário autenticado. O token de acesso do usuário deve ser fornecido no cabeçalho da solicitação. Ao realizar o logout, o servidor responde com uma mensagem indicando que o logout foi realizado com sucesso.



**DELETE** **/logout** Realiza logout do usuário autenticado

**Parameters** [Try it out](#)

Name	Description
<b>Authorization</b> * <i>required</i>	Token de acesso do usuário autenticado
string (header)	<input type="text" value="Authorization"/>

**Responses**

Code	Description	Links
200	Logout realizado com sucesso	No links
401	Requisição não autorizada	No links
500	Erro interno no servidor	No links

**Extensions**

Figura 14 – Rota para realização do *logout*

### 4.9 [PUT] /users/id

Essa rota atualiza um usuário existente com base no ID fornecido. É necessário fornecer o ID do usuário no caminho da solicitação e fornecer os dados atualizados do usuário no corpo da solicitação. O servidor responde com os dados atualizados do usuário.

PUT

/user/{id} Atualiza um usuário existente

Try it out

Parameters

Name	Description
<b>id</b> <small>required</small>	ID do usuário a ser atualizado
integer(int64)	
(path)	<input type="text" value="id"/>
<b>Authorization</b> <small>required</small>	Token de acesso do usuário autenticado
string	
(header)	<input type="text" value="Authorization"/>

Request body required

application/json

Dados do usuário a serem atualizados

Example Value | Schema

```
{
  "id": 123,
  "name": "John Doe",
  "registration": "ABC123",
  "email": "johndoe@example.com",
  "email_verified_at": "2023-07-01T10:00:00Z",
  "admin": true,
  "created_at": "2023-07-01T10:00:00Z",
  "updated_at": "2023-07-01T12:00:00Z"
}
```

Responses

Code	Description	Links
200	Usuário atualizado com sucesso	No links
<div>Media type: application/json</div> <div>Content-type header:</div> <div>Example Value   Schema</div> <div><pre>{   "id": 123,   "name": "John Doe",   "registration": "ABC123",   "email": "johndoe@example.com",   "email_verified_at": "2023-07-01T10:00:00Z",   "admin": true,   "created_at": "2023-07-01T10:00:00Z",   "updated_at": "2023-07-01T12:00:00Z" }</pre></div>		
401	Requisição não autorizada	No links
404	Usuário não encontrado	No links
500	Erro interno no servidor	No links

Figura 15 – Rota para realização da atualização de um usuário existente com base no id

## 5 Tecnologias

### 5.1 Swagger.io

O Swagger.io é uma ferramenta popular para criar, documentar e testar APIs de forma eficiente. Ele permite que os desenvolvedores definam a estrutura da API usando a linguagem YAML ou JSON, facilitando a geração automática de documentação interativa. Com o Swagger.io, é possível visualizar os endpoints disponíveis, testar as requisições e obter exemplos de resposta. Além disso, ele oferece recursos avançados, como autenticação e autorização de API (SWAGGERDOCS, 2023).

### 5.2 Laravel v.10

Laravel é um framework de desenvolvimento web em PHP que oferece uma sintaxe elegante e expressiva, além de uma ampla gama de recursos poderosos. Ele segue o princípio MVC (Model-View-Controller) e promove uma codificação limpa e organizada. O Laravel 10 inclui recursos como roteamento simplificado, autenticação de usuário, ORM (Object-Relational Mapping), suporte a banco de dados e cache, entre outros. Com ele, os desenvolvedores podem criar aplicativos web robustos de maneira eficiente (LARAVELDOCS, 2023).

### 5.3 PHP 8.1

O PHP 8.1 é a versão mais recente da linguagem de programação PHP. Ele traz melhorias significativas em termos de desempenho, segurança e recursos. Com o PHP 8.1, os desenvolvedores podem se beneficiar de recursos avançados, como tipos mistos, tipos de argumentos enumerados, sintaxe mais concisa, melhorias na manipulação de erros e muito mais. Além disso, essa versão oferece suporte a bibliotecas e frameworks populares, tornando-a uma escolha poderosa para o desenvolvimento de aplicativos web (PHPDOCS, 2023).

### 5.4 Biblioteca Laravel Sanctum

Esta é uma ferramenta de autenticação de API leve e poderosa para aplicativos Laravel. Ela permite a autenticação de usuários e a geração de tokens de acesso para proteger as rotas da API. Com o Laravel Sanctum, os desenvolvedores podem implementar autenticação baseada em tokens com facilidade, fornecendo segurança para suas APIs. Ele

suporta vários métodos de autenticação, incluindo autenticação de sessão, autenticação por token e autenticação por cookie (SANCTUMDOCS, 2023).

## 5.5 PHP Unit

O PHPUnit é uma estrutura de teste unitário para PHP. Ele fornece uma série de recursos e ferramentas para escrever e executar testes automatizados em código PHP. Com o PHPUnit, os desenvolvedores podem criar testes para verificar se suas classes, métodos e funções estão funcionando corretamente. Ele permite a criação de testes independentes, a definição de conjuntos de testes e a geração de relatórios detalhados sobre os resultados dos testes. O PHPUnit é amplamente utilizado na comunidade de desenvolvimento PHP para garantir a qualidade do código (PHPUNITDOCS, 2023).

## 5.6 MySQL

MySQL é um sistema de gerenciamento de banco de dados relacional (RDBMS) amplamente utilizado para armazenar e gerenciar dados. Ele fornece uma solução escalável e de alto desempenho para aplicativos web e outras aplicações que requerem um armazenamento estruturado de dados. O MySQL oferece recursos como suporte a consultas SQL, replicação de dados, segurança avançada, otimização de consultas e gerenciamento de transações. Ele é compatível com várias linguagens de programação, incluindo PHP, e é uma escolha popular para o desenvolvimento de aplicativos web (MySQL, 2023).

## 5.7 Draw.io

O Draw.io é uma ferramenta de diagramação online que permite criar diagramas e fluxogramas de forma intuitiva. Com uma interface fácil de usar, o Draw.io oferece uma ampla gama de formas, ícones e recursos de formatação para criar diagramas profissionais. É possível criar diagramas de rede, organogramas, fluxogramas, diagramas de processo e muito mais. Além disso, o Draw.io oferece a possibilidade de colaboração em tempo real, compartilhamento de diagramas e exportação em vários formatos, tornando-o uma escolha popular para profissionais que precisam visualizar e comunicar ideias de forma visual (DRAWIODOCS, 2023).

## 6 Boas práticas de desenvolvimento

Boas práticas de desenvolvimento a serem seguidas ao contribuir com código para o nosso Microserviço:

- Princípio da Responsabilidade Única (SRP): Cada classe ou componente deve ter apenas uma responsabilidade claramente definida. Evite classes ou métodos que tenham múltiplas responsabilidades, pois isso dificulta a manutenção e o entendimento do código.
- Princípio Aberto/Fechado (OCP): O código deve estar aberto para extensão, mas fechado para modificação. Utilize interfaces e padrões de design que permitam adicionar novas funcionalidades sem alterar o código existente.
- Princípio de Segregação de Interfaces (ISP): As interfaces devem ser segregadas em conjuntos coesos de operações relacionadas. Evite interfaces muito genéricas e que exijam implementação de métodos desnecessários.
- Princípio de Inversão de Dependência (DIP): Dependenda de abstrações e não de implementações concretas. Utilize injeção de dependência para reduzir o acoplamento e facilitar a substituição de componentes.
- DRY (Don't Repeat Yourself): Evite duplicação de código. Extraia lógica comum em métodos ou componentes reutilizáveis. Utilize herança, composição e funções utilitárias para evitar repetições desnecessárias.
- Testes Unitários: Escreva testes unitários para cada unidade de código (métodos, classes, componentes) isoladamente. Os testes devem verificar o comportamento esperado e ajudar a detectar erros ou regressões no código.
- Documentação e Comentários: Documente o código e sua estrutura de forma clara. Utilize comentários para explicar partes complexas ou de difícil entendimento. Mantenha a documentação atualizada à medida que o código evolui.



## 7 Conclusão

Este documento apresentou a implementação de um microsserviço de autenticação utilizando o Laravel 10 e PHP 8.1. Exploramos o uso da biblioteca Sanctum do Laravel para fornecer recursos de autenticação stateless. Além disso, discutimos a estrutura do código, organizada em Models e Controllers, e a importância do Laravel como um framework robusto e eficiente para o desenvolvimento de aplicações web em PHP.

# Referências

DRAWIODOCS. 2023. Software. JGraph Ltd, OpenSource. Disponível em: <https://www.drawio.com/doc/>. Citado na página 22.

LARAVELDOCS. 2023. Web framework. Taylor Otwell. Disponível em: <https://laravel.com/docs/10.x/readme>. Citado na página 21.

MySQL. 2023. Software. Oracle Corporation. Disponível em: <https://www.mysql.com/>. Citado na página 22.

PHPDOCS. 2023. Software. Rasmus Lerdorf, OpenSource. Disponível em: <https://www.php.net/docs.php>. Citado na página 21.

PHPUNITDOCS. 2023. Software. Rasmus Lerdorf, OpenSource. Disponível em: <https://phpunit.de/documentation.html>. Citado na página 22.

SANCTUMDOCS. 2023. Web framework. Taylor Otwell, OpenSource. Disponível em: <https://laravel.com/docs/10.x/sanctum>. Citado na página 22.

SWAGGERDOCS. 2023. Software. Software SmartBear, Somerville, Massachusetts. Disponível em: <https://swagger.io/docs/>. Citado na página 21.