

# **Usu-backend**

Equipe:Matheus Lourenço RM: 367182

## **1. Introdução**

### **Descrição do problema**

Na nossa região, um grupo de restaurantes decidiu contratar estudantes para construir um sistema de gestão para seus estabelecimentos. Essa decisão foi motivada pelo alto custo de sistemas individuais, o que levou os restaurantes a se unirem para desenvolver um sistema único e compartilhado. Esse sistema permitirá que os clientes escolham restaurantes com base na comida oferecida, em vez de se basearem na qualidade do sistema de gestão.

O objetivo é criar um sistema robusto que permita a todos os restaurantes gerenciar eficientemente suas operações, enquanto os clientes poderão consultar informações, deixar avaliações e fazer pedidos online. Devido à limitação de recursos financeiros, foi acordado que a entrega do sistema será realizada em fases, garantindo que cada etapa seja desenvolvida de forma cuidadosa e eficaz.

A divisão em fases possibilitará uma implementação gradual e controlada, permitindo ajustes e melhorias contínuas conforme o sistema for sendo utilizado e avaliado tanto pelos restaurantes quanto pelos clientes.

### **Objetivo do projeto**

Desenvolver um backend robusto utilizando Spring Boot para gerenciar usuários e atender aos requisitos definidos.

## **2. Arquitetura do Sistema**

### **Descrição da Arquitetura**

O sistema foi desenvolvido seguindo uma estrutura em camadas, um modelo bastante comum por deixar o código organizado e fácil de manter. As principais camadas são: Controller, Service, Repository, DTO, Entity, Exceptions, Handlers, Mappers, Security e Config.

A camada Controller recebe as requisições e envia as respostas ao cliente. A Service contém as regras de negócio do sistema, enquanto a Repository faz a comunicação com o banco de dados. A DTO serve para transferir dados sem expor diretamente as entidades, e as Entities representam as tabelas do banco. A parte de Exceptions define erros personalizados, tratados pelos Handlers. Os Mappers fazem a conversão entre entidades e DTOs, a Security cuida da autenticação e dos tokens, e a Config reúne as configurações gerais do projeto.

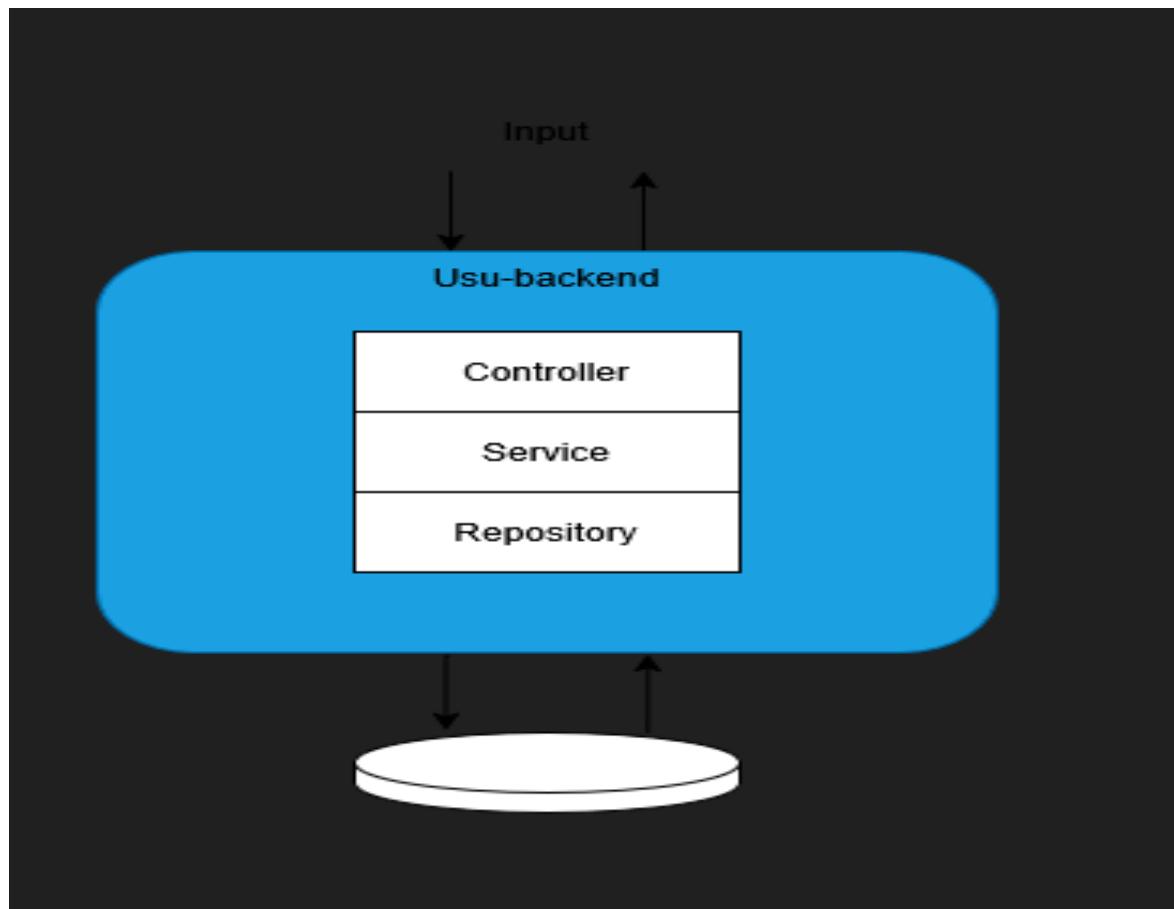
O banco de dados escolhido foi o postgres 16, por ser uma versão estável, bem documentada e com material de apoio disponível.

O Docker foi utilizado para facilitar a execução e o empacotamento do projeto, garantindo que todas as dependências e configurações sejam reproduzidas de forma padronizada em qualquer ambiente. A aplicação foi construída com um Dockerfile de múltiplas etapas (multi-stage build), o que reduz o tamanho final da imagem e melhora o desempenho, utilizando as versões Alpine do Java para ocupar menos espaço.

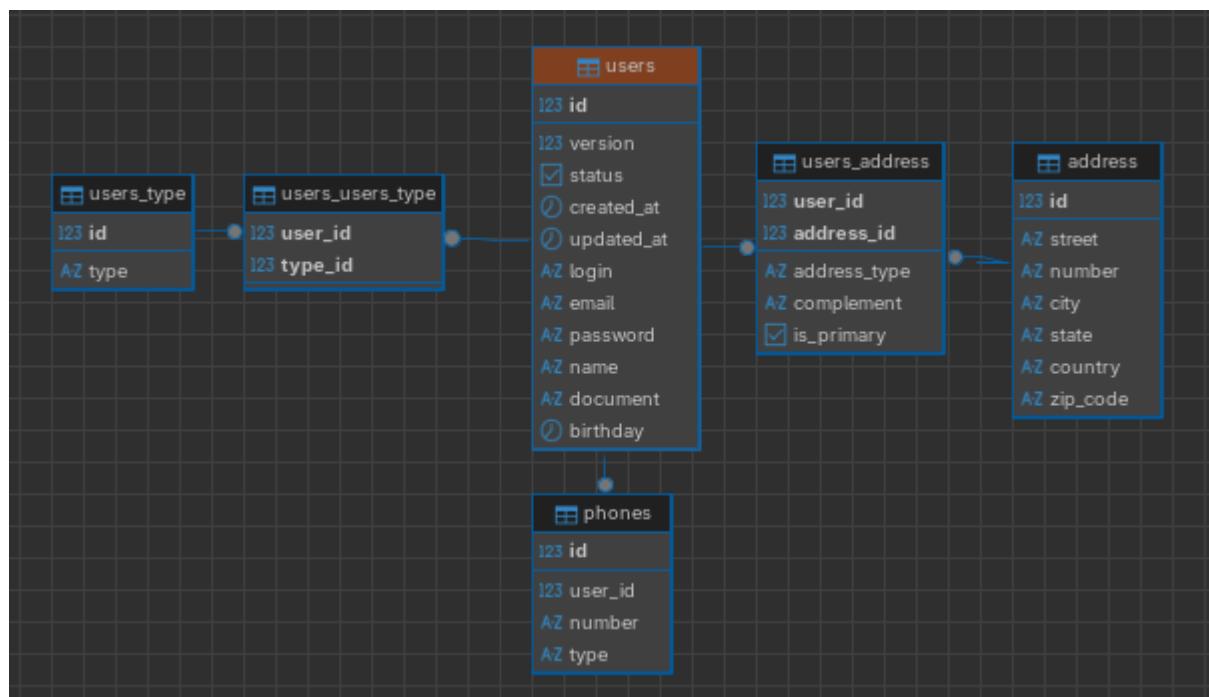
O arquivo docker-compose.yml define dois serviços principais: o backend, que executa a aplicação Java, e o postgres responsável pelo banco de dados. O Docker também realiza a comunicação entre esses serviços por meio de uma rede interna e mapeia as portas necessárias para o acesso externo.

As variáveis sensíveis, como as credenciais do banco e o secret usado para gerar o token JWT, são armazenadas em um arquivo .env, mantendo a segurança e a organização das configurações. Além disso, o uso de volumes garante a persistência dos dados do banco mesmo após a reinicialização dos containers.

## Diagrama da Arquitetura



## Diagrama das entidades



### 3. Descrição dos Endpoints da API

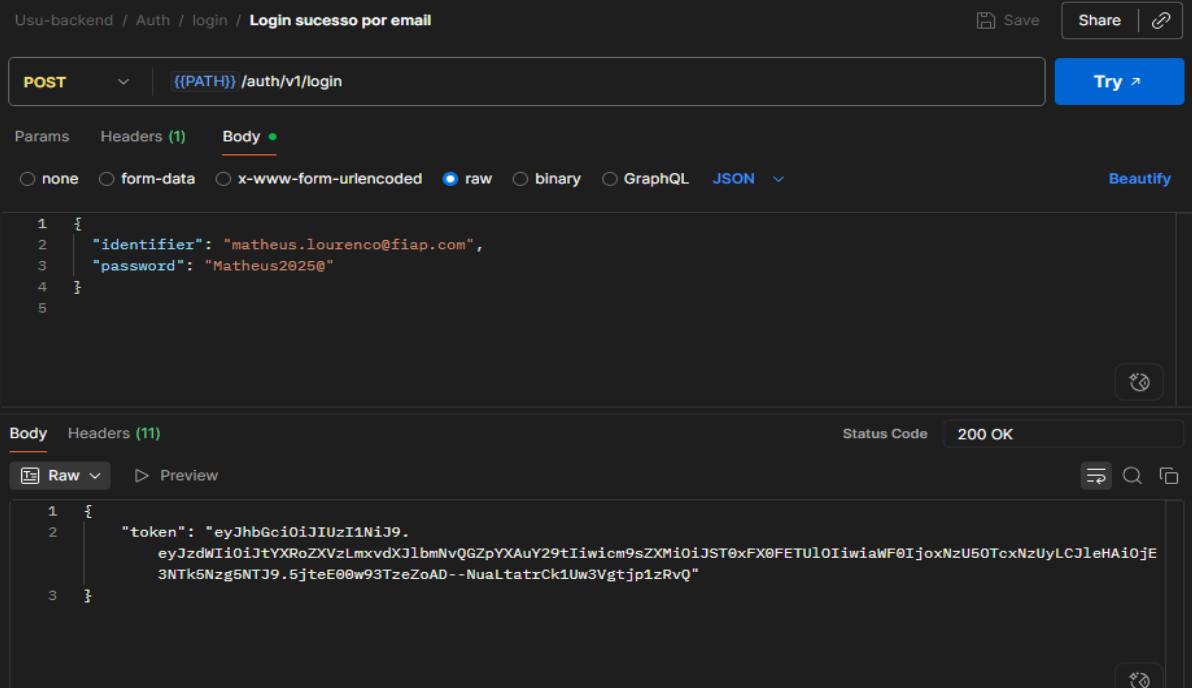
Tabela de Endpoints

Endpoint	Método	Descrição	Parâmetros
/auth/v1/login	<b>POST</b>	Gera um token de autenticação a partir do e-mail e senha do usuário.	-
/users/v1	<b>POST</b>	Cadastrar um novo usuário no sistema.	-
/users/v1	<b>GET</b>	Listar os usuários cadastrados(paginado e com filtro opcional por nome).	name
/users/v1/{id}	<b>GET</b>	Consultar dados de um único usuário pelo id.	id
/users/v1/{id}	<b>PATCH</b>	Atualizar dados de um usuário existente.	id
/users/v1/{id}	<b>DELETE</b>	Excluir um usuário do sistema.	id
/users/v1/password	<b>PATCH</b>	Atualizar a senha de um usuário autenticado.	-

## Exemplos de requisição e resposta

### 3.1. Login

#### 3.1.1 Login usando email



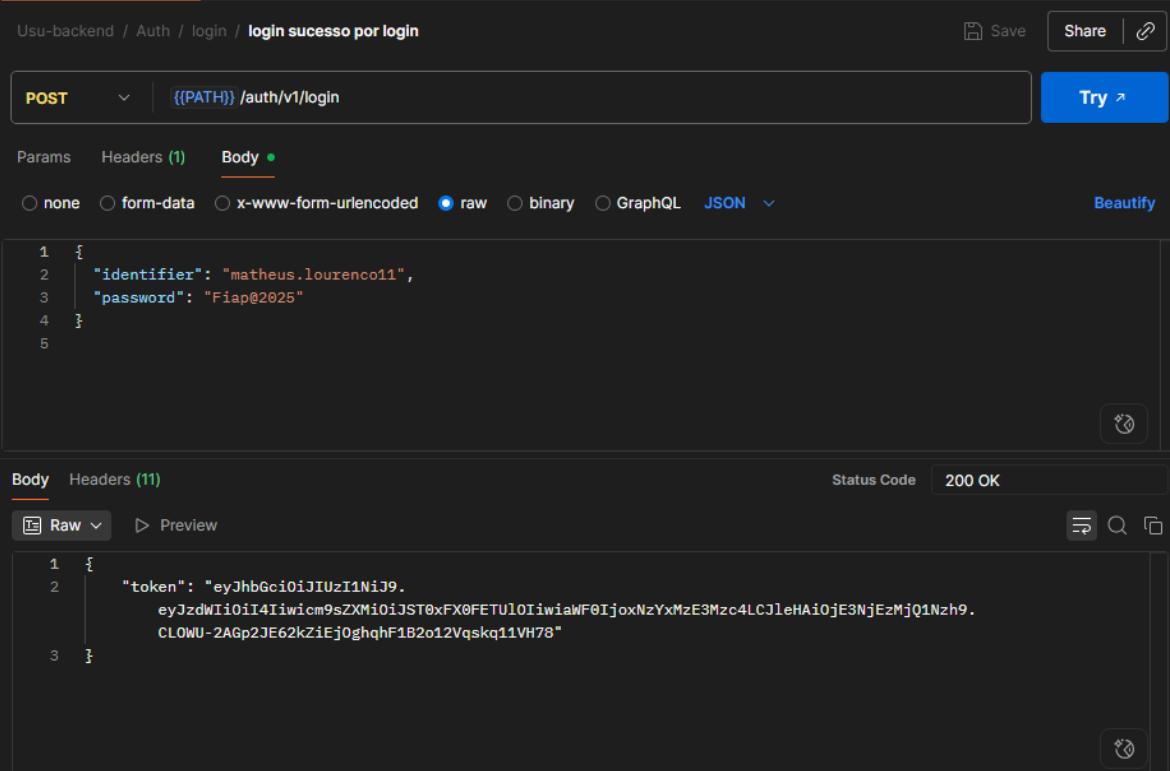
The screenshot shows the Postman application interface. At the top, the URL is `Usu-backend / Auth / login / Login sucesso por email`. Below it, the method is set to `POST` and the path is `((PATH)) /auth/v1/login`. The `Body` tab is selected, showing a raw JSON payload:

```
1 {  
2   "identifier": "matheus.lourenco@fiap.com",  
3   "password": "Matheus2025@"  
4 }  
5
```

On the right, the status code is `200 OK`. The response body contains a token:

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiJ9.  
3       eyJzdWIiOiJtYXR0ZXVzMmxvdXJlbmNvQGZpYXAuY29tIiwicm9sZXMiOiJST0xFX0FETUlOTiwiawF0IjoxNzU50TcxNzUyLCJleHAiOjE3NTk5Nzg5NTJ9.5jteE00w93TzeZoAD--NuaLatrCk1Uw3Vgtjp1zRvQ"  
3 }
```

#### 3.1.2 Login usando login



The screenshot shows the Postman application interface. At the top, the URL is `Usu-backend / Auth / login / login sucesso por login`. Below it, the method is set to `POST` and the path is `((PATH)) /auth/v1/login`. The `Body` tab is selected, showing a raw JSON payload:

```
1 {  
2   "identifier": "matheus.lourenco11",  
3   "password": "Fiap@2025"  
4 }  
5
```

On the right, the status code is `200 OK`. The response body contains a token:

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiJ9.  
3       eyJzdWIiOiI4Iiwicm9sZXMiOiJST0xFX0FETUlOTiwiawF0IjoxNzYxMzE3Mzc4LCJleHAiOjE3NjEzMjQ1Nzh9.  
3 CL0WU-2AGp2JE62kZiEj0ghqhF1B2o12Vqskq11VH78"  
3 }
```

### 3.1.3 Dados inválidos

The screenshot shows the Postman interface with the following details:

- URL:** {{PATH}}/auth/v1/login
- Method:** POST
- Headers:** (1)
- Body:** (1)
- Status Code:** 401 Unauthorized
- Response Body:**

```
1 {
2   "type": "https://example.com/problems/business-error",
3   "title": "Credenciais Inválidas",
4   "status": 401,
5   "detail": "Usuário ou senha incorretos.",
6   "instance": "/usu/auth/login",
7   "timestamp": "2025-10-09T01:44:43.653442601Z",
8   "path": "/usu/auth/login"
9 }
```

## 3.2. Salvar usuário

### 3.2.1 Persistir usuário

The screenshot shows the Postman interface with the following details:

- URL:** {{PATH}}/users/v1
- Method:** POST
- Headers:** (1)
- Body:** (1)
- Content Type:** raw
- Status Code:** 201 Created
- Response Body:**

```
2 {
3   "name": "Fiap Fiap",
4   "email": "fiap.fiap@fiap.com",
5   "login": "fiap",
6   "document": "11319526000740",
7   "password": "Fiap@2025",
8   "birthday": "1999-10-21",
9   "phones": [
10     {
11       "type": "HOME",
12       "number": "1"
13     }
14   ],
15   "addresses": [
16     {
17       "street": "Rua Nestor Victor",
18       "number": "317",
19       "city": "Curitiba",
20       "state": "PR",
21       "complement": "teste",
22       "addressType": "HOME",
23       "country": "Brazil",
24       "zipcode": "80620400",
25       "primary": true
26     }
27   ],
28   "idTypes": [
29     2
30   ]
31 }
```

### 3.2.2 Campos obrigatórios não informados

The screenshot shows the Postman interface with the following details:

- URL:** Usu-backend / Users / user / campos obrigatorios faltantes
- Method:** POST
- Body:** Raw JSON (selected)
- JSON Content:**

```
1 {
2     // "name": "joaozinho Lourenco",
3     // "email": "joaozinho.lourenco@fiap.com",
4     // "login": "joaozinho.lourenco",
5     // "document": "1",
6     // "password": "Matheus20250",
7     // "birthday": "1999-10-21",
8     "phones": [
9         {
10        }
11    ]
12}
```
- Status Code:** 400 Bad Request
- Response Body:**

```
2   {
3       "type": "https://example.com/problems/validation-error",
4       "title": "Erro de validação",
5       "status": 400,
6       "detail": "Um ou mais campos estão inválidos",
7       "instance": "/usu/users",
8       "timestamp": "2025-10-1ST23:03:29.617669999Z",
9       "path": "/usu/users",
10      "errors": [
11          {
12              "password": "must not be blank",
13              "idTypes": "must not be null",
14              "document": "must not be blank",
15              "name": "must not be blank",
16              "login": "must not be blank",
17              "email": "must not be blank"
18          }
19      ]
20  }
```

### 3.2.3 Documento já registrado

The screenshot shows the Postman interface with the following details:

- URL:** Usu-backend / Users / user / documento ja existente
- Method:** POST
- Body:** Raw JSON (selected)
- JSON Content:**

```
1 {
2     "name": "Fiap Fiap",
3     "email": "fiap.fiap@fiap.com",
4     "login": "fiap",
5     "document": "11319526900740",
6     "password": "Fiap@025",
7     "birthday": "1999-10-21",
8     "phones": [
9         {
10            "type": "HOME",
11            "number": "1"
12        }
13    ],
14     "addresses": [
15     ]
16 }
```
- Status Code:** 400 Bad Request
- Response Body:**

```
1   {
2       "type": "https://example.com/problems/business-error",
3       "title": "Erro de validação",
4       "status": 400,
5       "detail": "Document already registered.",
6       "instance": "/usu/users",
7       "timestamp": "2026-10-1ST23:36:59.618913609Z",
8       "path": "/usu/users"
9   }
```

### 3.2.4 Email já registrado

The screenshot shows the Postman interface with the following details:

- URL:** Usu-backend / Users / user / **email ja existente**
- Method:** POST
- Body (JSON):**

```
1 {
2     "name": "Fiap Fiap",
3     "email": "fiap.fiap@fiap.com",
4     "login": "fiap",
5     "document": "11319526000741",
6     "password": "Fiap@2025",
7     "birthday": "1999-10-21",
8     "phones": [
9         {
10            "type": "HOME",
11            "number": "1"
12        }
13    ],
14    "addresses": [
15    ]
}
```

- Status Code:** 400 Bad Request
- Response Body (Raw JSON):**

```
1 {
2     "type": "https://example.com/problems/business-error",
3     "title": "Erro de validação",
4     "status": 400,
5     "detail": "Email already registered.",
6     "instance": "/usu/users",
7     "timestamp": "2025-10-13T23:37:31.635311609Z",
8     "path": "/usu/users"
9 }
```

### 3.2.5 Login já registrado

The screenshot shows the Postman interface with the following details:

- URL:** Usu-backend / Users / user / **login ja existente**
- Method:** POST
- Query Params:**

Key	Value	Description	Bulk Edit
Key	Value	Description	

- Status Code:** 400 Bad Request
- Response Body (Raw JSON):**

```
1 {
2     "type": "https://example.com/problems/business-error",
3     "title": "Erro de validação",
4     "status": 400,
5     "detail": "Login already registered.",
6     "instance": "/usu/users",
7     "timestamp": "2025-10-13T23:38:03.591638586Z",
8     "path": "/usu/users"
9 }
```

### 3.2.6 Apenas usuário admin pode adicionar dono de restaurante e adm

The screenshot shows the Postman interface with the following details:

- URL:** {{PATH}} /users/v1
- Method:** POST
- Body:** (JSON) - Raw JSON payload:

```
28     "number": "39",
29     "city": "Curitiba",
30     "state": "PR",
31     "complement": "teste",
32     "addressType": "HOME",
33     "country": "Brazil",
34     "zipcode": "80620400",
35     "primary": true
36   },
37   ],
38   "idTypes": [
39     1
40   ]
```
- Status Code:** 403 Forbidden
- Response Body:** (Raw JSON)

```
1  {
2    "type": "https://example.com/problems/internal-error",
3    "title": "Access Denied",
4    "status": 403,
5    "detail": "You do not have permission to access this resource",
6    "instance": "/usu/users",
7    "timestamp": "2025-10-14T00:52:38.948024158Z",
8    "path": "/usu/users"
9 }
```

## 3.3. Listar usuários

### 3.3.1 Listar todos usuários

The screenshot shows the Postman interface with the following details:

- URL:** {{PATH}} /users/v1
- Method:** GET
- Params:** (1) - Query Params:

Key	Value	Description	Bulk Edit
name			
- Status Code:** 200 OK
- Response Body:** (Raw JSON)

```
14   {
15     "id": 4,
16     "name": "matheus lourenco",
17     "email": "matheus.lourenco@fiap.com",
18     "login": "matheus.lourenco",
19     "createdAt": "2025-10-09T00:50:14.991326Z",
20     "updatedAt": "2025-10-09T22:56:20.4291992Z",
21     "status": true,
22     "birthday": "1999-10-21",
23     "document": "1"
24   },
25   ],
26   "pageable": {
27     "pageNumber": 0,
28     "pageSize": 20,
29     "sort": {
30       "sorted": false,
31       "empty": true,
32       "unsorted": true
33     }
34   }
```

### 3.3.2 Listar usuários por nome

The screenshot shows the Postman interface for a GET request to the endpoint `/users/v1?name=matheus`. The 'Params' tab is selected, showing a query parameter `name` with the value `matheus`. The 'Body' tab shows the response in raw JSON format:

```
1 {  
2   "content": [  
3     {  
4       "id": 1,  
5       "name": "matheus lourenco",  
6       "email": "matheus.lourenco@fiap.com",  
7       "login": "matheus.lourenco",  
8       "createdAt": "2025-10-09T00:50:14.991326Z",  
9       "updatedAt": "2025-10-09T22:56:20.429199Z",  
10      "status": true,  
11      "birthday": "1999-10-21",  
12      "document": "1"  
13    }  
14  ],  
15  "pageable": {  
16    "pageNumber": 0,  
17    "pageSize": 20,  
18    "sort": {  
19      "sorted": false,  
20      "unsorted": true  
21    }  
22  }  
23}
```

The status code is 200 OK.

### 3.3.3 Usuário sem permissão

The screenshot shows the Postman interface for a GET request to the endpoint `/users/v1?name=matheus`. The 'Params' tab is selected, showing a query parameter `name` with the value `matheus`. The 'Body' tab shows the response in raw JSON format:

```
1 {  
2   "type": "https://example.com/problems/internal-error",  
3   "title": "Access Denied",  
4   "status": 403,  
5   "detail": "You do not have permission to access this resource",  
6   "instance": "/usu/users",  
7   "timestamp": "2025-10-10T01:09:21.925010057Z",  
8   "path": "/usu/users"  
9 }
```

The status code is 403 Forbidden.

## 3.4. Consultar usuário por ID

### 3.4.1 Usuário encontrado

The screenshot shows the Postman interface for a successful user retrieval. The URL is `https://usu-backend/api/users/v1/6`. The response body is:

```
1 {  
2   "id": 6,  
3   "name": "Matheus Lourenco",  
4   "email": "teste.fiap@fiap.com",  
5   "login": "matheus.lourenco",  
6   "createdAt": "13/10/2025 21:30:58",  
7   "updatedAt": "28/10/2025 10:24:12",  
8   "document": "113495260000746",  
9   "status": true,  
10  "birthday": "2025-10-23",  
11  "phones": [  
12    {  
13      "id": 10,  
14      "type": "HOME",  
15      "number": "2"  
16    },  
17    {  
18      "id": 9,  
19      "type": "HOME",  
20      "number": "1"  
21  ]  
22}
```

### 3.4.2 Usuário não encontrado

The screenshot shows the Postman interface for a user not found error. The URL is `https://usu-backend/api/users/v1/10`. The response body is:

```
1 {  
2   "type": "https://example.com/problems/business-error",  
3   "title": "Recurso não encontrado",  
4   "status": 404,  
5   "detail": "User not found.",  
6   "instance": "/usu/users/10",  
7   "timestamp": "2025-10-10T02:43:08.962876433Z",  
8   "path": "/usu/users/10"  
9 }
```

### 3.4.3 Usuário não tem permissão para acessar o recurso

The screenshot shows a Postman request for a user resource. The URL is `https://usu-backend/users/v1/1`. The method is GET. The status code is 403 Forbidden. The response body is a JSON object:

```
1 {  
2   "type": "https://example.com/problems/internal-error",  
3   "title": "Access Denied",  
4   "status": 403,  
5   "detail": "You do not have permission to access this resource",  
6   "instance": "/usu/users/1",  
7   "timestamp": "2025-10-10T02:42:16.483909508Z",  
8   "path": "/usu/users/1"  
9 }
```

## 3.5. Atualizar usuário

### 3.5.1 Atualizar apenas um campo

The screenshot shows a Postman request to update a user's name. The URL is `https://usu-backend/users/v1/1`. The method is PATCH. The body contains the following JSON:

```
1 {  
2   "name": "Matheus Lourenco"  
3 }
```

The status code is 204 No Content.

### 3.5.2 Atualizar múltiplos campos

The screenshot shows the Postman interface with the following details:

- URL:** {{PATH}} /users/v1/6
- Method:** PATCH
- Body:** JSON (selected)
- Body Content:**

```
1 {  
2   "name": "Matheus Louzenco",  
3   "birthday": "2025-10-23",  
4   "email": "teste.fiap@fiap.com",  
5   "phones": []  
6 }
```
- Status Code:** 204 No Content

### 3.5.3 Atualizar email para um já existente

The screenshot shows the Postman interface with the following details:

- URL:** {{PATH}} /users/v1/6
- Method:** PATCH
- Body:** JSON (selected)
- Body Content:**

```
1 {  
2   "email": "fiap.fiap@fiap.com"  
3 }
```
- Status Code:** 400 Bad Request
- Raw Response:**

```
1 {  
2   "type": "https://example.com/problems/business-error",  
3   "title": "Erro de validação",  
4   "status": 400,  
5   "detail": "Email already registered.",  
6   "instance": "/usu/users/6",  
7   "timestamp": "2025-10-23T12:48:03.543139427Z",  
8   "path": "/usu/users/6"  
9 }
```

### 3.5.4 Atualizar login para um já existente

The screenshot shows the Postman interface with the following details:

- URL:** Usu-backend / Users / user / **validar login existente**
- Method:** PATCH
- Path:** {{PATH}} /users/v1/6
- Try:** A blue button to execute the request.
- Params:** Headers (2)
- Body:** Headers (10) - Status Code: 400 Bad Request
- Raw:** Preview of the response body:

```
1 {  
2   "type": "https://example.com/problems/business-error",  
3   "title": "Erro de validação",  
4   "status": 400,  
5   "detail": "Login already registered.",  
6   "instance": "/usu/users/6",  
7   "timestamp": "2025-10-25T12:48:40.995481994Z",  
8   "path": "/usu/users/6"  
9 }
```

### 3.5.5 Atualizar login para um já existente

The screenshot shows the Postman interface with the following details:

- URL:** Usu-backend / Users / **user**
- Method:** PATCH
- Path:** {{PATH}} /users/v1/2
- Send:** A blue button to execute the request.
- Params:** Authorization
- Headers:** (10)
- Body:** raw
- Body Content:**

```
1 {  
2   "login": "admin"  
3 }
```
- Body:** Cookies Headers (10) Test Results
- Status:** 400 Bad Request
- Raw:** Preview of the response body:

```
1 {  
2   "type": "https://example.com/problems/business-error",  
3   "title": "Erro de validação",  
4   "status": 400,  
5   "detail": "Login already registered.",  
6   "instance": "/usu/users/v1/2",  
7   "timestamp": "2025-11-02T17:42:54.050123632Z",  
8   "path": "/usu/users/v1/2"
```

## 3.6. Atualizar senha

### 3.6.1 Atualizar senha

The screenshot shows a POST request to `http://Usu-backend / Users / {{PATH}}/password update`. The method is set to `PATCH` and the URL is `{{(PATH)}}/users/v1/password`. The `Body` tab is selected, showing the following JSON payload:

```
1 {
2   "email": "matheus.lourenco@fiap.com",
3   "currentPassword": "Fiap@2025",
4   "newPassword": "Matheus2025@"
5 }
```

The response status is `204 No Content`, with a response time of `207 ms` and a size of `282 B`. The response body is empty.

### 3.6.2 Senha atual inválida

The screenshot shows a POST request to `http://Usu-backend / Users / {{PATH}}/password`. The method is set to `PATCH` and the URL is `{{(PATH)}}/users/v1/password`. The `Body` tab is selected, showing the following JSON payload:

```
1 {
2   "email": "teste.lourenco@fiap.com",
3   "currentPassword": "Matheus2025@",
4   "newPassword": "Luana2025@"
5 }
```

The response status is `400 Bad Request`. The response body contains an error object:

```
1 {
2   "type": "https://example.com/problems/business-error",
3   "title": "Erro de validação",
4   "status": 400,
5   "detail": "invalid current password.",
6   "instance": "/usu/users/password",
7   "timestamp": "2025-10-09T02:29:20.088481385Z",
8   "path": "/usu/users/password"
9 }
```

### 3.6.3 Acesso invalido

The screenshot shows the Postman interface for a PATCH request to `/users/v1/password`. The status code is 403 Forbidden. The response body is a JSON object:

```
1 {  
2   "type": "https://example.com/problems/internal-error",  
3   "title": "Access Denied",  
4   "status": 403,  
5   "detail": "You do not have permission to access this resource",  
6   "instance": "/usu/users/password",  
7   "timestamp": "2025-10-09T02:32:43.999716121Z",  
8   "path": "/usu/users/password"  
9 }
```

### 3.6.4 Senha inválida

The screenshot shows the Postman interface for a PATCH request to `/users/v1/password`. The status code is 400 Bad Request. The response body is a JSON object:

```
1 {  
2   "type": "https://example.com/problems/business-error",  
3   "title": "Senha inválida",  
4   "status": 400,  
5   "detail": "Invalid password. It must have at least 6 characters, one uppercase letter, one number, and one special character.",  
6   "instance": "/usu/users/password",  
7   "timestamp": "2025-10-09T02:33:29.473050159Z",  
8   "path": "/usu/users/password"  
9 }
```

### 3.6.5 Campos obrigatórios não informados

The screenshot shows the Postman interface for a PATCH request to `/users/v1/password`. The 'Params' tab is selected, showing an empty table. The 'Body' tab is selected, showing a JSON response with a validation error. The response body is:

```
1 {  
2   "type": "https://example.com/problems/validation-error",  
3   "title": "Erro de validação",  
4   "status": 400,  
5   "detail": "Um ou mais campos estão inválidos",  
6   "instance": "/usu/users/password-update",  
7   "timestamp": "2025-10-09T02:34:15.794355156Z",  
8   "path": "/usu/users/password-update",  
9   "errors": {  
10     "newPassword": "must not be blank",  
11     "email": "must not be blank",  
12     "currentPassword": "must not be blank"  
13   }  
14 }
```

### 3.6.6 Email invalido

The screenshot shows the Postman interface for a PATCH request to `/users/v1/password`. The 'Params' tab is selected, showing an empty table. The 'Body' tab is selected, showing a JSON response with a business error. The response body is:

```
1 {  
2   "type": "https://example.com/problems/business-error",  
3   "title": "Recurso não encontrado",  
4   "status": 404,  
5   "detail": "User not found.",  
6   "instance": "/usu/users/password-update",  
7   "timestamp": "2025-10-09T02:45:06.764891266Z",  
8   "path": "/usu/users/password-update"  
9 }
```

## 3.7. Excluir usuário

### 3.7.1 Deletado

The screenshot shows the Postman interface with the following details:

- URL:** Usu-backend / Users / delete user / **deletado com sucesso**
- Method:** DELETE
- Try:** A blue button labeled "Try" is visible.
- Params:** Headers (2) is selected. There are no query parameters.
- Body:** Headers (9) is selected. Status Code: 204 No Content.

### 3.7.2 Não encontrado

The screenshot shows the Postman interface with the following details:

- URL:** Usu-backend / Users / delete user / **usuario não encontrado**
- Method:** DELETE
- Try:** A blue button labeled "Try" is visible.
- Params:** Headers (2) is selected. There are no query parameters.
- Body:** Headers (11) is selected. Status Code: 404 Not Found.
- Raw:** A JSON response is shown:

```
1 {  
2   "type": "https://example.com/problems/business-error",  
3   "title": "Recuzso não encontrado",  
4   "status": 404,  
5   "detail": "User not found.",  
6   "instance": "/usu/users/17",  
7   "timestamp": "2025-10-09T23:28:54.888156179Z",  
8   "path": "/usu/users/17"  
9 }
```

### 3.7.3 Sem permissão

The screenshot shows the Postman interface with the following details:

- URL:** Usu-backend / Users / **delete user**
- Method:** DELETE
- Send:** A blue button labeled "Send" is visible.
- Params:** Authorization, Headers (8), Body, Scripts, Settings, Cookies, Query Params.
- Body:** Cookies, Headers (11), Test Results.
- Status:** 403 Forbidden.
- Raw:** A JSON response is shown:

```
1 {  
2   "type": "https://example.com/problems/internal-error",  
3   "title": "Access Denied",  
4   "status": 403,  
5   "detail": "You do not have permission to access this resource",  
6   "instance": "/usu/users/v1/1",  
7   "timestamp": "2025-11-02T17:46:03.536391021Z",  
8   "path": "/usu/users/v1/1"
```

## 4. Configuração do Projeto

### Configuração do Docker Compose

O arquivo docker-compose.yml define e orquestra múltiplos serviços Docker. Neste projeto são dois serviços, a aplicação em Java (usu\_backend) e o banco de dados (postgres). Permitindo que ambos sejam executados e se comuniquem de forma integrada e automatizada.

Cada serviço representa um container que será iniciado com suas configurações específicas. Os serviços são configurados de formas independentes:

#### **Postgres:**

- Image: Usa a imagem oficial do postgres na versão 16, baseada em Alpine.
- Container\_name: Nome do container para facilitar a identificação.
- Restart: Garante que o container será reiniciado automaticamente caso pare por algum motivo.
- Env\_file: carrega o arquivo onde contém as variáveis de ambiente.
- Environment: Passa as variáveis para o banco, criando o banco e o usuário inicial.
- Ports: Faz o mapeamento entre a porta do host e a porta interna do container (5432).
- Volumes: utilizado para armazenar os dados do banco.
- Networks: Conecta na mesma rede Docker que o banco, permitindo que o backend acesse o postgres via hostname.

#### **Usu\_backend:**

- Build:Constrói a imagem da aplicação Java usando o Dockerfile presente na raiz.
- Image: Nome da imagem resultante.
- Restart: Garante que o container será reiniciado automaticamente caso pare por algum motivo.
- env\_file: Carrega o arquivo onde contém as variáveis de ambiente.
- Depends\_on: Garante que o container do postgres seja iniciado antes do backend.
- ports: Expõe a aplicação.

## Instruções para execução local

O projeto pode ser executado de duas formas distintas: localmente sem o uso do Docker ou utilizando containers via Docker Compose. Abaixo, são descritos os procedimentos necessários para cada abordagem.

### Execução Local (sem Docker)

#### 1. Instalação da versão java

Caso o java 21 não esteja instalado na máquina, é necessário realizar sua instalação.

#### 2. Instalação do postgres

Caso o postgres não esteja instalado na máquina, é necessário realizar sua instalação.

Uma alternativa seria a executar o banco de dados em um container Docker com o comando:

```
docker run --name my-postgres -e POSTGRES_PASSWORD=fiap -p 5432:5432 -d postgres:16-alpine
```

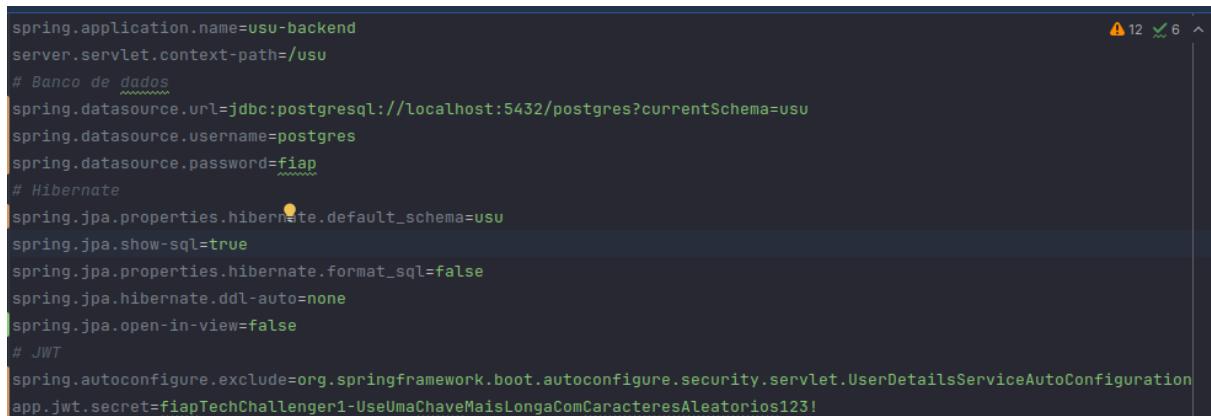
#### 3. Criação das tabelas e dados iniciais

Após a instalação do banco de dados, deve-se executar o script init.sql, localizado na pasta postgres do projeto.

Esse script é responsável por criar as tabelas, índices e o usuário padrão da aplicação.

#### 4. Configuração do arquivo application.properties

Atualize as variáveis relacionadas à conexão com o banco e ao segredo utilizado para geração do token JWT.



```
spring.application.name=usu-backend
server.servlet.context-path=/usu
# Banco de dados
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres?currentSchema=usu
spring.datasource.username=postgres
spring.datasource.password=fiap
# Hibernate
spring.jpa.properties.hibernate.default_schema=usu
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=false
spring.jpa.hibernate.ddl-auto=none
spring.jpa.open-in-view=false
# JWT
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.servlet.UserDetailsServiceAutoConfiguration
app.jwt.secret=fiapTechChallenger1-UseUmaChaveMaisLongaComCaracteresAleatorios123!
```

#### 5. Execução da aplicação

Execute a aplicação na sua idle.

## **Execução via Docker Compose**

### **1. Instalação do docker e docker compose**

Caso o docker não esteja instalado na máquina, é necessário realizar sua instalação.

### **2. Abrir o terminal na pasta do projeto**

Navegue até o diretório raiz do projeto onde se encontra o arquivo docker-compose.yml.

### **3. Subir o Container da aplicação**

Execute o seguinte comando para construir a imagem e iniciar os containers da aplicação e do banco de dados:

```
docker compose up --build
```

Por se tratar de um projeto acadêmico, tanto a execução via Docker quanto a execução local utilizam as mesmas portas padrão: 8080 para a aplicação e 5432 para o banco de dados postgres.

## 5. Qualidade do Código

### 5.1. Injeção de dependências

O serviço utiliza injeção de dependências via construtor, evitando a criação direta de objetos dentro da classe. Isso torna o código mais modular, desacoplado e testável.

```
public UserService(  
    UserRepository userRepository,  
    UserPhoneService phoneService,  
    UserAddressService userAddressService,  
    UserTypeRepository userTypeRepository,  
    SecurityValidator securityValidator  
) {
```

### 5.2. Transações declarativas

Operações críticas no banco de dados estão anotadas com `@Transactional`, garantindo que todas as alterações sejam aplicadas de forma atômica, prevenindo inconsistências.

```
@Transactional 1 usage  ↗ Matheus Lourenco  
public Long create(UserCreateDto dto) {
```

### 5.3. Controle de acesso e segurança

A anotação `@PreAuthorize` é utilizada para garantir que apenas usuários autorizados possam executar determinadas ações, aumentando a segurança do sistema.

```
@Transactional 1 usage  ↗ Matheus Lourenco  
@PreAuthorize("#id == principal.id or hasAnyRole('ADMIN')")  
public Long update(Long id, UserUpdateDto dto) {
```

### 5.4. Validações centralizadas

Regras de validação são encapsuladas em métodos específicos e em classes de validação (`SecurityValidator`), seguindo o princípio DRY (Don't Repeat Yourself).

```
private void validePersistUser(UserCreateDto dto) { 1 usage  ↗ Matheus Lourenco  
    // Apenas admin pode adicionar ADMIN e RESTAURANT_OWNER
```

## 5.5. Uso de logs

O código registra eventos importantes utilizando diferentes níveis de log (info, debug, warn), o que auxilia na depuração, monitoramento e auditoria.

```
log.info("Iniciando criação do usuário: {}", dto.email());
log.debug("Validação do usuário concluída para: {}", dto.email());
```

## 5.6. Mapeamento entre DTO e entidade

A separação entre DTOs e entidades é realizada por meio de Mappers, evitando o acoplamento entre a camada de apresentação e a persistência.

```
User user = UserMapper.mapPersistenceEntity(dto);
validadePersistUser(dto);
```

## 5.7. Tratamento de exceções específico

O código lança exceções customizadas (`UserAlreadyExistsException`, `InvalidPasswordException`, `ResourceNotFoundException`) para situações específicas, proporcionando mensagens de erro claras e tratamento adequado.

```
if (userRepository.existsByDocument(dto.document())) {
    throw new UserAlreadyExistsException(ValidationMessages.DOCUMENT_EXISTS);
}
```

## 5.8. Separação de responsabilidades

Serviços específicos, como `UserPhoneService` e `UserAddressService`, são responsáveis apenas por suas respectivas funcionalidades, seguindo o princípio Single Responsibility do SOLID.

```
User saved = userRepository.save(user);
log.info("Usuário salvo com ID: {}", saved.getId());

userAddressService.persistUserAddresses(user, dto.addresses());
phoneService.saveUserPhone(user, dto.phones());
```

## 6. Collections para Teste

### Link para a Collection do Postman

[https://app.getpostman.com/join-team?invite\\_code=b6e9a6e7714d2866e63797164ad534ddca0687c9ec8ac06df6ee1155c6c2ad93&target\\_code=b4df7c400c8c96fc2b7b66906fb57e7c](https://app.getpostman.com/join-team?invite_code=b6e9a6e7714d2866e63797164ad534ddca0687c9ec8ac06df6ee1155c6c2ad93&target_code=b4df7c400c8c96fc2b7b66906fb57e7c)

### Descrição dos Testes Manuais

Para a validação manual dos endpoints do sistema, foi desenvolvida uma collection no Postman contendo todos os cenários necessários para testar as funcionalidades da API. Essa collection utiliza variáveis de ambiente para padronizar as requisições e facilitar a execução dos testes.

A variável PATH representa o caminho base da aplicação, definido como `http://localhost:8080/usu`. Já a variável token é preenchida automaticamente por um script interno após a autenticação bem-sucedida, sendo utilizada nos endpoints que exigem autenticação.

A autenticação é necessária para todos os endpoints, com exceção do POST /users/v1. O token de acesso é obtido através do endpoint /auth/v1/login, responsável por autenticar o usuário e retornar um token JWT.

Para simplificar o processo de validação, foi criado um usuário padrão no sistema com as seguintes credenciais:

- Login: admin
- E-mail: admin@fiap.com
- Senha: Fiap@2025

O token gerado por meio do login é automaticamente armazenado na variável token e incluído no cabeçalho das requisições no formato:

```
Authorization: Bearer {{token}}
```

O endpoint POST /users/v1 requer autenticação apenas quando utilizado para criar usuários do tipo ADMIN ou RESTAURANT\_OWNER.

Por padrão, as respostas da API são retornadas em inglês. Para exibir as mensagens de erro e respostas em português, deve-se adicionar o cabeçalho:

```
Accept-Language: pt_BR
```

Qualquer dúvida sobre os endpoints é possível validar através do Swagger do projeto <http://localhost:8080/usu/swagger-ui/index.html>.

## **6.1. Login(POST /auth/v1/login)**

O endpoint de autenticação tem como objetivo gerar um token JWT a partir das credenciais informadas.

Foram realizados testes para os seguintes cenários:

- Login com e-mail: autenticação utilizando o campo identifier como e-mail do usuário.
- Login com login de usuário: autenticação utilizando o campo identifier como login.
- Credenciais inválidas: tentativa de autenticação com dados incorretos, resultando em resposta 401 Unauthorized com mensagem padronizada.

Após um login bem-sucedido, o token é automaticamente armazenado na variável {{token}}, que passa a ser utilizada nos demais endpoints autenticados.

## **6.2. Criar Usuário (POST /users/v1)**

O endpoint de criação de usuários permite cadastrar novos registros no sistema. Os testes realizados incluíram:

- Cadastro bem-sucedido: criação de um usuário completo, retornando 201 Created.
- Campos obrigatórios ausentes: validação de erro 400 Bad Request com retorno detalhado por campo inválido.
- Campos aninhados inválidos: verificação de inconsistências em listas de telefones e endereços.
- Regras de negócio: bloqueio de duplicidade para document, email e login.
- Controle de acesso: tentativa de criação de usuários do tipo ADMIN e RESTAURANT\_OWNER por perfis não autorizados, retornando 403 Forbidden.

### **6.3. Listar Usuários (GET /users/v1)**

Esse endpoint permite consultar usuários cadastrados de forma paginada. Foram validados os seguintes cenários:

- Listagem geral: retorno de todos os usuários com status 200 OK.
- Filtro por nome: uso do parâmetro name para pesquisa específica.
- Usuário sem permissão: bloqueio de acesso a perfis não autorizados, retornando 403 Forbidden.

### **6.4. Consultar Usuário por ID (GET /users/v1/{id})**

Permite a obtenção dos detalhes de um usuário específico. Os testes realizados incluíram:

- Usuário encontrado: retorno das informações completas, incluindo endereços e telefones.
- Usuário inexistente: resposta 404 Not Found.
- Acesso restrito: tentativa de acesso por perfil não autorizado, resultando em 403 Forbidden.

### **6.5. Atualizar Usuário (PATCH /users/v1/{id})**

Responsável pela atualização parcial ou total dos dados de um usuário existente. Foram testados os seguintes casos:

- Atualização de um único campo: alteração de um atributo isolado, como e-mail.
- Atualização múltipla: modificação de mais de um campo na mesma requisição.
- Conflito de dados: tentativa de atualização com e-mail ou login já cadastrados, retornando 400 Bad Request.

## **6.6. Atualizar Senha (PATCH /users/v1/password)**

Permite que o usuário autenticado altere sua senha. Foram contemplados os seguintes testes:

- Atualização bem-sucedida: senha alterada com sucesso, respeitando os critérios de segurança.
- Senha atual inválida: erro 400 Bad Request.
- Campos ausentes: verificação de validação de obrigatoriedade.
- Formato inválido: tentativa de uso de senha fora do padrão exigido.
- Acesso sem token: erro 401 Unauthorized.

## **6.7. Excluir Usuário (DELETE /users/v1/{id})**

Realiza a exclusão de um usuário existente. Os cenários testados foram:

- Exclusão com sucesso: resposta 204 No Content.
- Usuário inexistente: erro 404 Not Found.
- Usuário sem permissão: tentativa de exclusão por perfil não autorizado, resultando em 403 Forbidden.

## **7. Repositório do Código**

### **URL do Repositório**

<https://github.com/MLourenco111/usu-backend> ou use a collection que esta salva na pasta resource do projeto.