

API de Integração SofitView GraphQL

1. Visão geral

Esta documentação abrange alguns módulos disponíveis para busca de informações. Ela inclui informações sobre os parâmetros aceitos e os formatos de resposta correspondentes. Aqui você encontrará detalhes sobre como interagir com o graphql.

2. URL padrão:

Segue a URL base padrão para acesso às consultas em GraphQL, sendo ela:

Ambiente	URL padrão
Produção	https://sofitview.com.br/api/v2/graphql

3. Login - Autenticação na API

A autenticação na API do SofitView é realizada através de tokens. Cada requisição deve conter um token de acesso válido no cabeçalho "**Authorization**" seguindo um formato de autenticação "**Bearer**". O *token* de acesso deverá ser incluído em **todas** as requisições. O corpo (body) da requisição deve ser fornecido no formato JSON. Certifique-se de que o conteúdo enviado no corpo da requisição esteja corretamente estruturado.

Segue um exemplo:

Endpoint	<code>https://sofitview.com.br/api/v1/users/login</code>
Method	<code>POST</code>
Payload	<pre>{ "user_name": "<USUARIO_INTEGRACAO>", "password": "<SENHA_INTEGRACAO>" }</pre>
Response	<pre>{ "uuid": "<UUID>" "token": "<TOKEN_DE_ACESSO>" "user": "<USER>" }</pre>

Para realizar requisições à API, recomenda-se o uso de aplicativos especializados em fazer requisições HTTP, como Postman, Insomnia, entre outros. Essas ferramentas oferecem uma interface intuitiva e recursos avançados que facilitam a interação com a API, permitindo testes, depuração e visualização das respostas de forma eficiente.

3.1 Autenticação no GraphQL Playground

O token de acesso deve ser incluído em todas as queries, sendo enviado nos cabeçalhos HTTP seguindo o formato de autenticação Bearer. Se você preferir utilizar a IDE própria do GraphQL, é necessário informar o token da seguinte maneira:

Aqui está como você deve inserir o token de acesso no GraphQL Playground:



QUERY VARIABLES **HTTP HEADERS (1)**

```
1 {  
2   "Authorization": "Bearer <SEU_TOKEN>"  
3 }
```

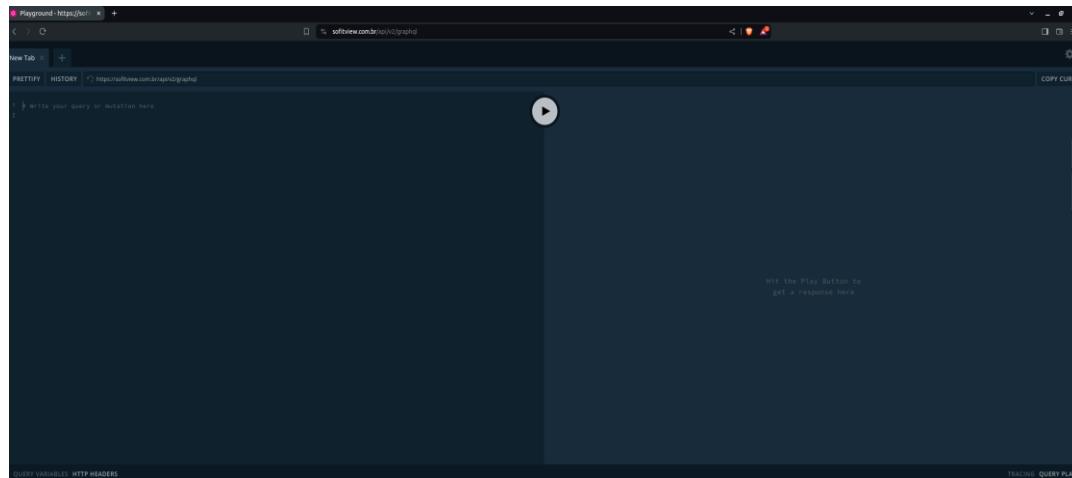
4. GraphQL Playground

- **O que é?**

GraphQL Playground é um IDE GraphQL criado e mantido pela Prisma. Ele é construído em cima do GraphQL com recursos adicionais, como recarregamento automático de esquema, suporte para assinaturas GraphQL, a capacidade de configurar cabeçalhos HTTP e muito mais. Ótimo para efetuar testes com consultas a serem utilizadas em APIs.

- **Como acessar?**

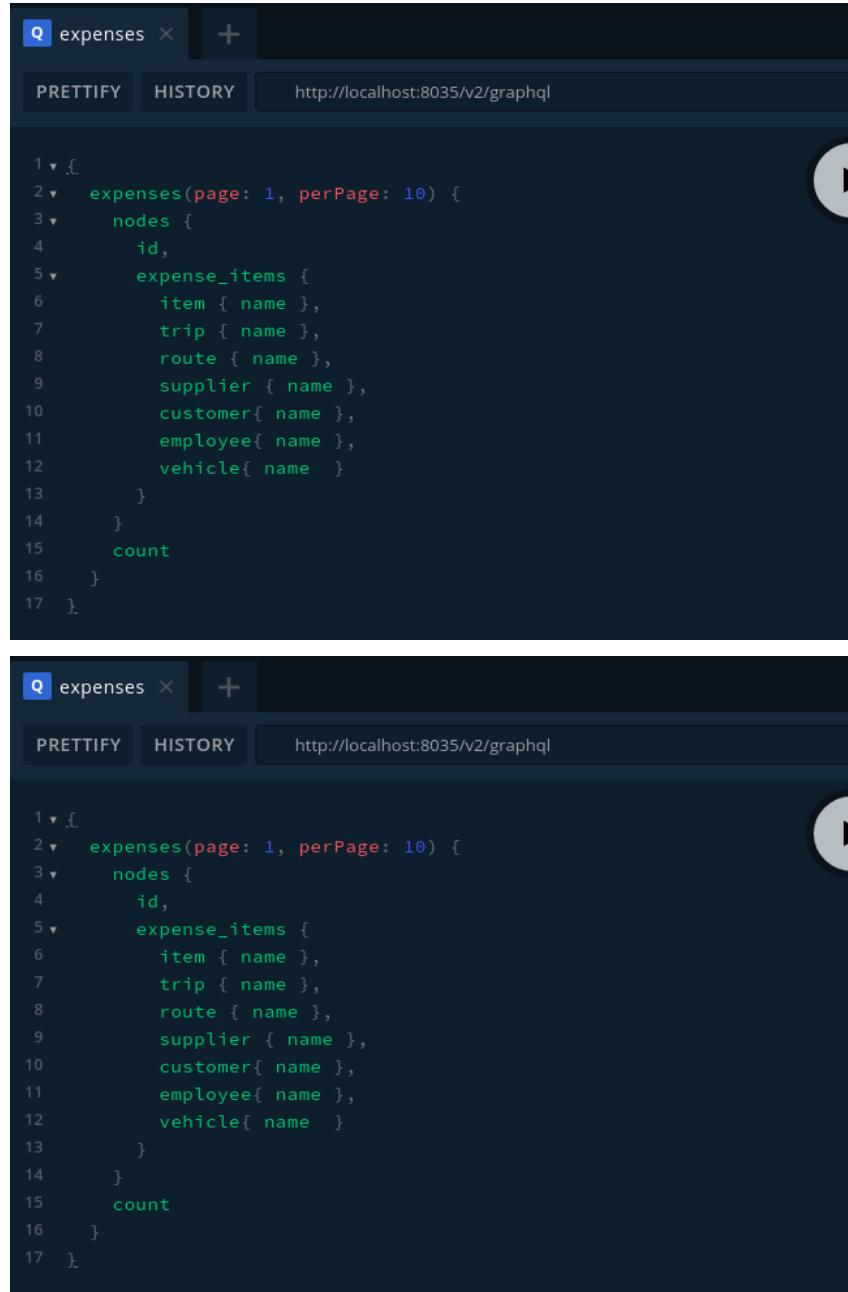
Para acessar o Playground basta abrir um navegador e digitar a url do ambiente, conforme imagem abaixo:



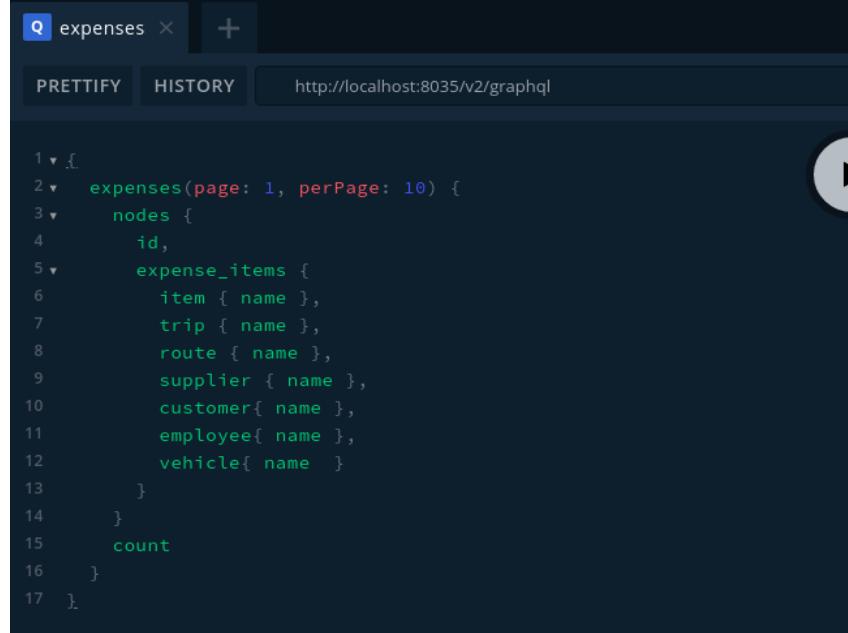
- **Como utilizar?**

URL: Na parte superior da tela, validar se a URL a ser chamada corresponde a URL do ambiente em questão, conforme imagem abaixo:

Query: Na parte superior esquerda da tela, inserir a query com os valores a serem buscados, conforme imagem abaixo:



```
1 ▼ [.
2 ▼   expenses(page: 1, perPage: 10) {
3 ▼     nodes {
4       id,
5       expense_items {
6         item { name },
7         trip { name },
8         route { name },
9         supplier { name },
10        customer{ name },
11        employee{ name },
12        vehicle{ name }
13      }
14    }
15    count
16  }
17 ].
```



```
1 ▼ [.
2 ▼   expenses(page: 1, perPage: 10) {
3 ▼     nodes {
4       id,
5       expense_items {
6         item { name },
7         trip { name },
8         route { name },
9         supplier { name },
10        customer{ name },
11        employee{ name },
12        vehicle{ name }
13      }
14    }
15    count
16  ]
17 ].
```

Result: Na parte direita da tela serão disponibilizados os resultados obtidos da query executada, conforme imagem abaixo:



The screenshot shows a dark-themed GraphQL interface. On the left, a large text area displays a JSON-like query result. The result includes a "data" field, an "expenses" field, and a "nodes" field containing a list of expense items. Each item has an "id" of 1 and a name of "Compra de pneu". To the right of the main content area, there are two vertical tabs: "DOCS" and "SCHEMA".

```
graph TD; A[""] --> B["data"]; B --> C["expenses"]; C --> D["nodes"]; D --> E["id: 1"]; E --> F["expense_items"]; F --> G["item"]; G --> H["name: Compra de pneu"]
```

Docs: Localizados à direita da tela, após os resultados da consulta, encontram-se os módulos que detalham os métodos a serem utilizados nas queries. Estes módulos contêm informações cruciais, como o nome do método e a tabela utilizada para a busca. Os parâmetros (argumentos) disponíveis para o método permitem filtrar as informações da tabela, como especificar a quantidade de itens a serem retornados, a paginação e a ordenação dos resultados. Além disso, os módulos fornecem detalhes sobre o retorno do método e os tipos de parâmetros e retorno. Para mais informações, consulte a imagem abaixo.

The screenshot shows the SofitView GraphQL documentation interface. On the left, there's a sidebar with tabs for 'DOCS' (selected) and 'SCHEMA'. The main area has a search bar at the top. Below it, under 'QUERIES', there are several items: 'employee(...): Employee!', 'expenses(...): PaginatedExpense!' (which is expanded), 'serviceOrders(...): PaginatedServiceOrder!', and 'vehicles(...): PaginatedVehicle!'. To the right of these queries, the 'SCHEMA' section is displayed. It includes sections for 'expenses' (with fields 'page: Int = 1', 'perPage: Int = 20', and 'lastIntegrationDate: DateTime'), 'TYPE DETAILS' (showing the type definition for 'PaginatedExpense'), and 'ARGUMENTS' (listing the arguments for the 'expenses' query). A 'Download' button is located at the bottom right of the schema panel.

Schema: Disponíveis na parte direita da tela, abaixo dos **Docs**. Neles estão contidos todos os tipos de dados disponíveis nas **APIs**. Ele disponibiliza o nome dos schemas, suas propriedades e seus respectivos tipos. Também é possível fazer download dessas informações em formato JSON ou SDL. Mais detalhes na imagem abaixo:

This screenshot shows the 'SCHEMA' section of the SofitView interface. On the left, there's a sidebar with 'DOCS' and 'SCHEMA' tabs. The main area displays the GraphQL schema code. It includes a directive '@specifiedBy(url: String!) on SCALAR' and definitions for 'City' and 'CostCenter' types. The 'CostCenter' type has fields like 'id', 'name', 'external_id', 'state_id', and various timestamp and boolean fields. On the right side, there's a 'DOWNLOAD' section with 'JSON' and 'SDL' buttons. The 'SDL' button is currently selected.

5. Requisições

Agora iremos abordar as requisições para comunicação entre as APIs. Utilizaremos a ferramenta Postman para exemplificar. Toda requisição Http efetuada deve ser do tipo POST e deve conter o Authorization mencionado anteriormente.

5.1 Para criar o corpo da requisição, sugerimos copiar a query testada no Playground, segue o exemplo abaixo:

```
expenses(page: 1, perPage: 10, lastIntegrationDate: "2024-01-01T00:00:00Z") {  
  nodes {  
    id  
  
    expense_items {  
      item {  
        name  
      }  
  
      trip {  
        name  
      }  
  
      route {  
        name  
      }  
  
      supplier {  
        name  
      }  
  
      customer {  
        name  
      }  
  
      employee {  
        name  
      }  
  
      vehicle {  
        name  
      }  
    }  
  }  
  count  
}
```

```
{  
  employee(cpf: "Tão") {  
    name,  
    active,  
    cpf,  
    rg,  
    address  
  }  
}
```

5.2 Acessar o Postman, informar que é uma requisição POST para a URL do ambiente, mudar o tipo do body para GraphQL e passar o payload do exemplo acima.

The screenshot shows the GraphQL playground interface with the following details:

- Header:** GraphQL - New Request
- Method:** POST
- URL:** https://scikitview.com.br/api/v2/graphql
- Authorization:** Bearer [REDACTED]
- Headers:** Content-Type: application/json
- Body:** A GraphQL query for a trip with its route, supplier, and customer details.
- Pre-request Script:** No script is present.
- Tests:** No tests are present.
- Settings:** Schema Fetched
- Buttons:** Save, Send, Cookies

GraphQL Query:

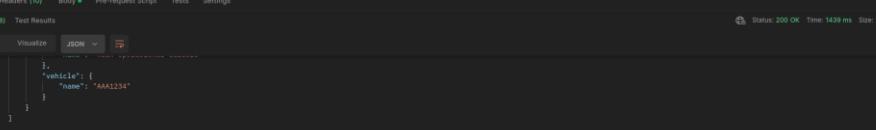
```
query {  
    expense(page: 1, perPage: 10, lastIntegrationDate: "2024-01-01T00:00:00") {  
        nodes {  
            id  
            name  
            expense_items {  
                item {  
                    name  
                }  
            }  
            trip {  
                name  
            }  
            route {  
                name  
            }  
            supplier {  
                name  
            }  
            customer {  
                name  
            }  
        }  
    }  
}
```

GraphQL Variables:

```
{  
    page: 1,  
    perPage: 10,  
    lastIntegrationDate: "2024-01-01T00:00:00"  
}
```

Status Bar: Status: 200 OK | Time: 14.99 ms | User: 174 kB | Save as example

5.3 Com o envio desta requisição, os resultados obtidos serão disponibilizados da seguinte forma:



The screenshot shows the Postman interface with a successful POST request to <https://softview.com.br/api/v2/graphql>. The response status is 200 OK, time 1439 ms, size 1.74 KB, and it includes a save as example button.

Request Details:

- Method:** POST
- URL:** https://softview.com.br/api/v2/graphql
- Headers:** (10) including Content-Type: application/json
- Body:** (4) showing a JSON payload with vehicle and expense_items data.
- Test Results:** (1) showing a success message: "Success! Response received from https://softview.com.br/api/v2/graphql".

JSON Payload (Body):

```
POST https://softview.com.br/api/v2/graphql

{
  "query": "query { vehicles { id, name } expense_items { item { name } vehicle { id, name } } }",
  "variables": {
    "vehicles": [
      {
        "id": 127,
        "name": "Gasolina comum"
      }
    ],
    "expense_items": [
      {
        "item": {
          "name": "Gasolina comum"
        },
        "vehicle": {
          "id": 1248,
          "name": "Compria de pneu"
        }
      }
    ]
  }
}
```

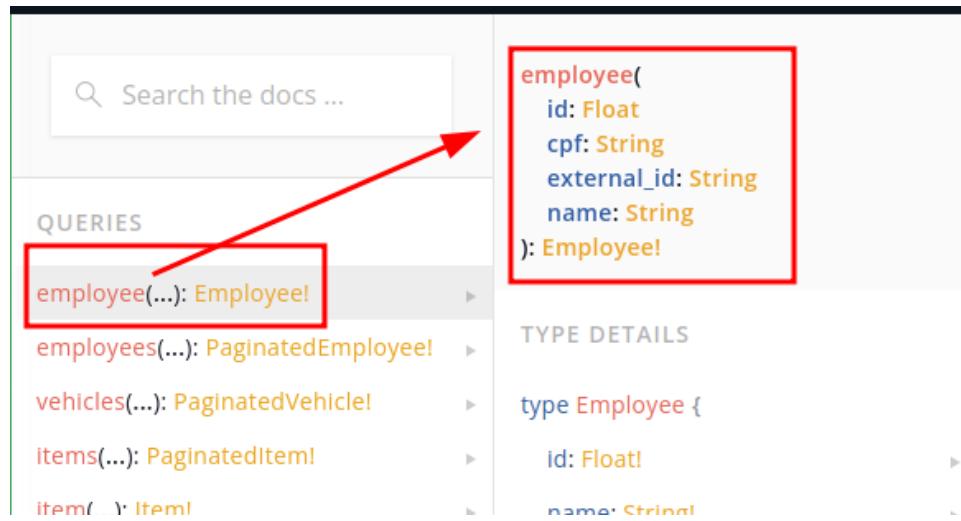
6. Funcionalidades disponíveis e Utilização

As funcionalidades que podem retornar mais de um registro recebem os mesmos parâmetros (argumentos), sendo eles:

Parâmetro	Rótulo	Tipo	Obrigatório?	Observações
page	Página	Inteiro	Não	Caso não seja informado, assume o valor 1. Exemplo: 5
perPage	Registros por página	Inteiro	Não	Caso não seja informado, assume o valor de 20 e possui um limite máximo de 20. Exemplo 10
lastIntegrationDate	Data da última integração	Data	Não	Caso não seja informado, assume o valor referente a data do dia anterior. Exemplo: "2020-10-22T00:00:00Z"

6.1 Observação

Nem todas as funcionalidades as tabelas recebem os parâmetros acima, tabelas que pesquisam apenas por um registro como no exemplo a seguir:



The screenshot shows a GraphQL playground interface. On the left, under 'QUERIES', there is a red box around the query `employee(...): Employee!`. An arrow points from this box to the 'employee' type definition on the right. The 'employee' type is defined with fields: `id: Float`, `cpf: String`, `external_id: String`, and `name: String`. Below the type definition, under 'TYPE DETAILS', is the type definition for `Employee`:

```

type Employee {
  id: Float!
  name: String
}

```

As tabelas que estão no singular, como por exemplo "**employee**", os parâmetros disponíveis se limitam aos argumentos necessários para filtrar ou localizar um único registro no sistema. Conforme ilustrado no print acima, esses parâmetros incluem "id", "external_id", "cpf" ou "name".

Já o retorno das funcionalidades, são no mesmo formato. Todos retornam duas propriedades, sendo elas:

- **nodes**: contém uma lista com valores do seu respectivo tipo. Exemplo: uma lista de veículos;
- **count**: contém o contador do total de registros recebidos da query.

Exemplos de queries abaixo:

6.2 Veículos

Método: vehicles

```
vehicles(  
    page: Int = 1  
    perPage: Int = 20  
    lastIntegrationDate: DateTime  
) : PaginatedVehicle!
```

Retorno:

```
type PaginatedVehicle {  
    nodes: [Vehicle!]!  
    count: Int!  
}
```

GraphQL Query:

```
{  
    vehicles(page: 1, perPage: 10, lastIntegrationDate: "2020-10-22T00:00:00Z") {  
        nodes {  
            id,  
            model { name },  
            model_version { name },  
            group { name },  
            subsidiary { name },  
            cost_center { name },  
            employee { name }  
        }  
        count  
    }  
}
```

6.3 Despesas

Método: expenses

```
expenses(  
    page: Int = 1  
    perPage: Int = 20  
    lastIntegrationDate: DateTime  
) : PaginatedExpense!
```

Retorno:

```
type PaginatedExpense {  
    nodes: [Expense!]!  
    count: Int!  
}
```

GraphQL Query:

```
{  
    expenses(page: 1, perPage: 10, lastIntegrationDate: "2020-10-22T00:00:00Z") {  
        nodes {  
            id,  
            trip { name },  
            route { name },  
            supplier { name },  
            customer { name },  
            employee { name }  
        }  
        count  
    }  
}
```

GraphQL Query 2:

```
{  
    expenses(page: 1, perPage: 10, lastIntegrationDate: "2020-10-22T00:00:00Z") {  
        nodes {  
            id,  
            expense_items {  
                item { name },  
                trip { name },  
                route { name },  
                supplier { name },  
                customer { name },  
            }  
        }  
    }  
}
```

```
        employee{ name },
        vehicle{ name }
      }
    }
    count
  }
}
```

6.4 Ordens de Serviço

Método: serviceOrders

```
serviceOrders(
  page: Int = 1
  perPage: Int = 20
  lastIntegrationDate: DateTime
): PaginatedServiceOrder!
```

Retorno:

```
type PaginatedServiceOrder {
  nodes: [ServiceOrder!]
  count: Int!
}
```

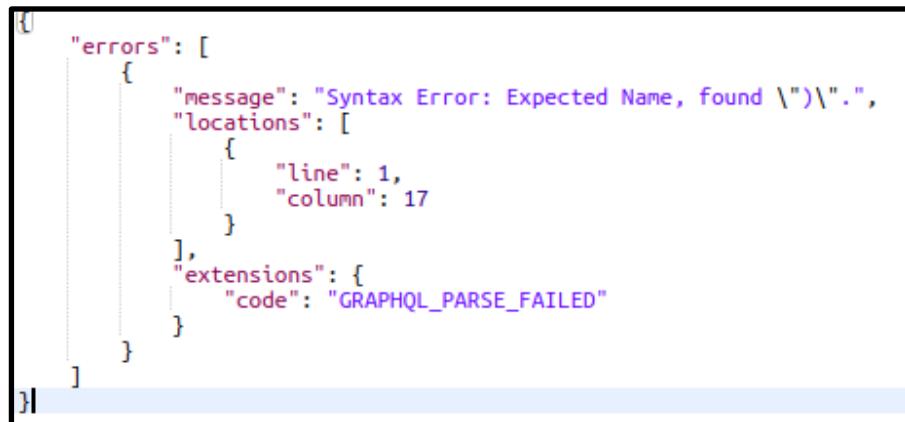
GraphQL Query:

```
{  
  serviceOrders(page: 1, perPage: 10, lastIntegrationDate: "2020-10-22T00:00:00Z") {  
    nodes {  
      id,  
      vehicle { name },  
      employee { name },  
      supplier { name },  
      foreseen_service_order_items {  
        name,  
        item { name }  
      }  
    }  
    count  
  }  
}
```

7. Possíveis erros

Todas as funcionalidades descritas anteriormente podem disparar os seguintes erros:

- **Necessidade de, pelo menos, um parâmetro (argumento):** Com o playground isto não é possível pois o mesmo não realiza a query caso não seja passado pelo menos um único parâmetro. Já por uma requisição pelo Postman é possível e é retornado erro de **PARSE** provindo do próprio GraphQL:



- **Propriedade perPage deve receber até um máximo de 20:** A propriedade perPage, que é passada por parâmetro pode disparar um erro caso ultrapasse seu total de 20:

```
{
  "errors": [
    {
      "message": "perPage precisa ter no máximo 20.",
      "locations": [
        {
          "line": 1,
          "column": 3
        }
      ],
      "path": [
        "serviceOrders"
      ],
      "extensions": {
        "code": "INTERNAL_SERVER_ERROR",
        "exception": {
          "response": {
            "statusCode": 422,
            "message": "perPage precisa ter no máximo 20.",
            "error": "Unprocessable Entity"
          },
          "status": 422,
          "message": "perPage precisa ter no máximo 20."
        }
      }
    ],
    "data": null
  ]
}
```

8. Logout

Após o término da integração dos dados, deve ser realizado o *logout* do usuário de integração, a fim de fechar sua conexão e invalidar o *token* gerado previamente:

Endpoint	https://sofitview.com.br/api/v1/users/logout
Method	POST
Headers	Authorization → Bearer <TOKEN_DE_ACESSO>
Payload	None
Response (status)	200 → OK 401 → Invalid signature received for JSON Web Token validation.