



Centro Universitário SENAC – Santo Amaro

# Back-End

Integrantes: Matheus Ferreira, João Gabriel, Lucas Ryu, Bruno Gabriel, José Daniel e Matheus Oldoni.

Professor: Carlos Verissimo.

Matéria: Programação Web (TADS 2º Semestre).

04 de dezembro de 2022

# Desenvolvimento

## Configurações de ambiente:

```
1  import express, {Request, Response, NextFunction} from 'express';
2  import {router} from './routes';
3  import cors from 'cors';
4  import './config/env';
5
6  const app = express();
7
8  app.use(express.json());
9  app.use(cors());
10 app.use(router);
11 app.use((erro: Error, req: Request, res: Response, next: NextFunction)=>{
12     if(erro instanceof Error){
13         return res.status(400).json({Error: erro.message});
14     }
15
16     return res.status(500).json(
17         {
18             status: "Erro",
19             message: "Internal Error."
20         }
21     );
22 });
23
24 app.listen(3333, () => console.log("Servidor Ativo!"));
```

(Linhas 1 a 4): Para dar início a prova apresentada em aula, é necessário definir algumas configurações fundamentais para que possa ser realizado o projeto, para isso é necessário a importação do "express, Request, Response, NextFunction", linha 1, que irá servir para conseguir posteriormente trabalhar com rotas dentro da aplicação, que é o fundamental para uma API. É necessário também o "router" vindo diretamente do arquivo "routes.ts" que futuramente abordarei sobre, importo o cors, para poder fazer requisição independentemente da URL, e o config env que futuramente falei mais sobre.

# Desenvolvimento

## Continuação:

(Linha 6): Para conseguir fazer o servidor funcionar é necessário declarar uma constante onde receberá o método "express()" como valor inicial e padrão.

(Linha 8 a 22): Logo após definir a constante app, deve-se informar que em alguma parte da aplicação, pode ser recebido um JSON como parte da requisição, para que seja possível o recebimento de alguma requisição em forma de JSON é necessário informar para a constante, previamente definida e configurada, que ela deve usar um JSON, dessa forma, deve ser informado: "app.use(express.json())".

Logo após isso, é necessário informar para a constante que ela deve usar o método "cors()", que como dito anteriormente, permitirá qualquer URL solicitar uma requisição.

É necessário também informar para a constante que o arquivo Router, importado anteriormente, será usado para controlar a aplicação inteira, para isso define-se: "app.use(router)".

Também se faz necessário ter tratamento de erro para que caso alguma requisição seja feita de forma indevida retorne um erro amigável para o usuário, para isso, da linha 11 até a 22 tem um método específico só para isso.

(Linha 24): E por fim, é necessário declarar uma porta para a api escutar, para isso é necessário declarar o método "listen()" para a constante passando:

1. Obrigatoriamente a porta que ela irá escutar (Nesse exemplo, usando a porta 3333);
2. Alguma mensagem saber quando ela para realmente foi executada (Opcional).

# Desenvolvimento

Primeira parte:

```
1 import Router from 'express';
2 import 'express-async-errors';
3 import {CreateCategoryController} from './controllers/categoria/CreateCategoryController';
4 import {CreateUserController} from './controllers/user/CreateUserController';
5 import {CreateComandaController} from './controllers/comanda/CreateComandaController';
6 import {CreateProductController} from './controllers/produto/CreateProductController';
7 import {CreateItemController} from './controllers/item/CreateItemController';
8 import {auth} from './middlewares/auth';
```

```
10 const router = Router();
11 const createUserController = new CreateUserController();
12 const createCategoryController = new CreateCategoryController();
13 const createComandaController = new CreateComandaController();
14 const createProductController = new CreateProductController();
15 const createItemController = new CreateItemController();
```

(Linhas 1 e 2): É Definido algumas configurações para o ambiente onde servem justamente para garantir o pleno funcionamento do Back-end.

(Linhas 3 a 7): É importado também as classes "Controllers" de cada funcionalidade do Back-end.

(Linha 8): É importado a constante "auth" que fica na classe "auth.ts" que serve justamente para definir um nível de segurança a mais para a aplicação.

(Linha 10): É criado a constante "router" que servirá justamente para controle de cada end-point e cada funcionalidade dos respectivos end-points.

(Linhas 11 a 15): São criados também os objetos de cada classe, para que fique mais fácil chamá-las em rotas que serão apresentadas futuramente.

# Desenvolvimento

## Funcionalidade de Usuários + Funcionalidade de Token:

```
17  router.post('/usuarios', createUserController.post);
18  router.post('/usuarios/login', createUserController.login);
19
20  router.use(auth);
```

(Linhas 17 e 18): É criado 2 rotas do tipo POST, a primeira delas é a “/usuarios” que serve justamente para criar novos usuários no sistema, e a segunda rota do tipo POST, “/usuarios/login” serve justamente para os usuários fazerem login, onde, retornará como resposta um Token de autenticação, que será requerido para qualquer outro tipo de requisição que desejar fazer para a API, dando assim um nível de segurança maior para a aplicação.

Observação 1: Esse token é gerado no arquivo “auth.ts”, que fica dentro da pasta middlewares, é possível ver mais sobre no link do repositório informado no final dessa documentação.

Observação 2: Todo tipo de requisição informada abaixo precisará necessariamente ser informado o token de usuário gerado na rota de login.

```
22  router.get('/usuarios', createUserController.get);
23  router.get('/usuario/:id', createUserController.getUnique);
24  router.put('/usuarios/:id', createUserController.put);
25  router.delete('/usuarios/:id', createUserController.delete);
```

(Linhas 22 e 23): São criadas duas rotas do tipo GET, onde a primeira delas na linha 22, serve para retornar todos os usuários cadastrados no sistema, e a segunda, linha 23, serve para retornar apenas os dados de um e somente um único usuário registrado, onde é pego através de uma requisição por parâmetros o ID do usuário.

# Desenvolvimento

## Continuação:

(Linha 24): É criada uma rota do tipo PUT para editar os dados do usuário. Será recebido por meio de uma requisição Body um novo nome, o novo perfil e a sua nova senha e um Id do usuário, que será informado através de uma requisição por Parâmetro e editará seus valores se:

1. Se o Usuário informado existir.

(Linha 25): É criado uma rota do tipo DELETE para deletar justamente um usuário previamente criado, onde será recebido por meio de uma requisição por parâmetro o id do usuário e a partir disso será feito sua remoção do sistema.

## Funcionalidade de Categorias:

```
27 router.post('/categorias', createCategoryController.post);
28 router.get('/categorias', createCategoryController.get);
29 router.get('/categoria/:id', createCategoryController.getUnique);
30 router.put('/categorias/:id', createCategoryController.put);
31 router.delete('/categorias/:id', createCategoryController.delete);
```

(Linha 27): É criado uma rota do tipo POST, “/categorias” que serve justamente para criar novos usuários no sistema, deve ser informado por requisição Body o nome da categoria nova que deseja criar.

(Linhas 28 e 29): São criadas duas rotas do tipo GET, onde a primeira delas, na linha 28, serve para retornar todas as categorias cadastradas no sistema, e a segunda, linha 29, serve para retornar apenas os dados de uma e somente uma única categoria registrada, onde é pego através de uma requisição por parâmetros o ID da categoria.

# Desenvolvimento

## Continuação:

(Linha 30): É criada uma rota do tipo PUT para editar os dados de uma categoria. Será recebido por meio de uma requisição Body um novo nome da categoria e um Id, da categoria, que será informado através de uma requisição por Parâmetro e editará seus valores se:

1. Se a Categoria informada realmente existir.

(Linha 31): É criado uma rota do tipo DELETE para deletar justamente uma categoria previamente criada, onde será recebido por meio de uma requisição por parâmetro o id da categoria e a partir disso será feito sua remoção do sistema.

## Funcionalidade de Comanda:

```
33 router.post('/comandas', createComandaController.post);
34 router.get('/comandas', createComandaController.get);
35 router.get('/comanda/:id', createComandaController.getUnique);
36 router.put('/comandas/:id', createComandaController.put);
37 router.delete('/comandas/:id', createComandaController.delete);
```

(Linha 33): É criado uma rota do tipo POST, "/comandas" que serve justamente para criar novas comandas no sistema deve ser informado por requisição Body o número da mesa, o status da mesa, um rascunho, e um id de usuário para a comanda que deseja criar.

# Desenvolvimento

## Continuação:

(Linhas 34 e 35): São criadas duas rotas do tipo GET, onde a primeira delas, na linha 34, serve para retornar todas as comandas cadastradas no sistema, e a segunda, linha 35, serve para retornar apenas os dados de uma e somente uma única comanda registrada, onde é pego através de uma requisição por parâmetros o ID da comanda.

(Linha 36): É criada uma rota do tipo PUT para editar os dados de uma comanda. Será recebido por meio de uma requisição Body um novo número da mesa, um status, um rascunho e um Id, da comanda, que será informado através de uma requisição por Parâmetro e editará seus valores se:

1. A mesa nova informada não estiver sendo ocupada;
2. E Se a comanda informada por parâmetros realmente existir.

(Linha 37): É criada uma rota do tipo DELETE para deletar justamente uma comanda previamente criada, onde será recebido por meio de uma requisição por parâmetro o id da comanda e a partir disso será feito sua remoção do sistema.

## Funcionalidade de Produtos:

```
39 router.post('/produtos', createProductController.post);
40 router.get('/produtos', createProductController.get);
41 router.get('/produto/:id', createProductController.getUnique);
42 router.put('/produtos/:id', createProductController.put);
43 router.delete('/produtos/:id', createProductController.delete);
```

(Linha 39): É criada uma rota do tipo POST, “/produtos” que serve justamente para criar novas comandas no sistema deve ser informado por requisição Body o nome do produto, preço do produto, a sua descrição e o id da categoria que ele pertence para o produto que deseja criar.



# Desenvolvimento

## Continuação

(Linhas 40 e 41): São criadas duas rotas do tipo GET, onde a primeira delas, na linha 40, serve para retornar todos os produtos cadastrados no sistema, e a segunda, linha 41, serve para retornar apenas os dados de um e somente um produto registrado, onde é pego através de uma requisição por parâmetros o ID do produto.

(Linha 42): É criada uma rota do tipo PUT para editar os dados de um produto. Será recebido por meio de uma requisição Body um novo nome do produto, preço do produto, descrição e o id da categoria e um Id, do produto, que será informado através de uma requisição por Parâmetro e editará seus valores se:

1. O Produto informado realmente exista;
2. A categoria informada realmente existir.

(Linha 37): É criada uma rota do tipo DELETE para deletar justamente um produto previamente criada, onde será recebido por meio de uma requisição por parâmetro o id do produto e a partir disso será feita sua remoção do sistema.

## Funcionalidade de Itens:

```
45  router.post('/itens', createItemController.post);
46  router.get('/itens', createItemController.get);
47  router.get('/item/:id', createItemController.getUnique);
48  router.put('/itens/:id', createItemController.put);
49  router.delete('/itens/:id', createItemController.delete);
```

(Linha 45): É criada uma rota do tipo POST, "/itens" que serve justamente para criar novos itens no sistema deve ser informado por requisição Body a quantidade de itens, o id de comanda e do produto que ele pertence para o item que deseja criar.

# Desenvolvimento

## Continuação

(Linhas 46 e 47): São criadas duas rotas do tipo GET, onde a primeira delas, na linha 46, serve para retornar todos os itens cadastrados no sistema, e a segunda, linha 47, serve para retornar apenas os dados de um e somente um item registrado, onde é pego através de uma requisição por parâmetros o ID do item.

(Linha 48): É criada uma rota do tipo PUT para editar os dados de um item. Será recebido por meio de uma requisição Body uma nova quantidade, um id de categoria e produto e um Id, do produto, que será informado através de uma requisição por Parâmetro e editará seus valores se:

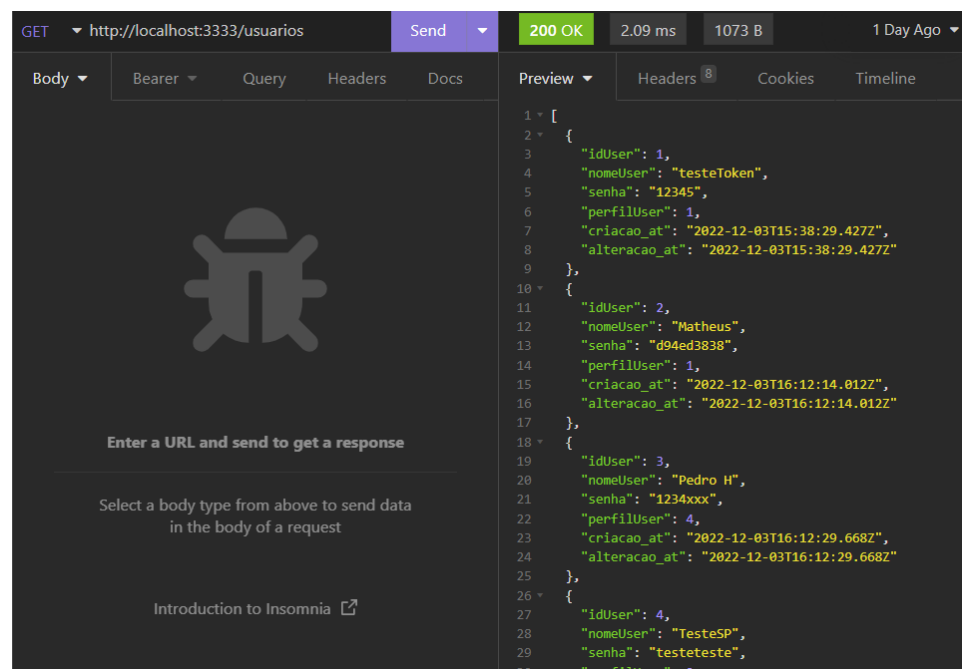
1. O Item informado realmente exista;
2. A categoria informada realmente existir;
3. O produto informado realmente existir.

(Linha 49): É criado uma rota do tipo DELETE para deletar justamente um item previamente criado, onde será recebido por meio de uma requisição por parâmetro o id do item e a partir disso será feito sua remoção do sistema.

# Imagens

Abaixo algumas imagens da API funcionando

```
yarn run v1.22.19
$ ts-node-dev src/server.ts
[INFO] 16:39:59 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 4.9.3)
Servidor Ativo!
```

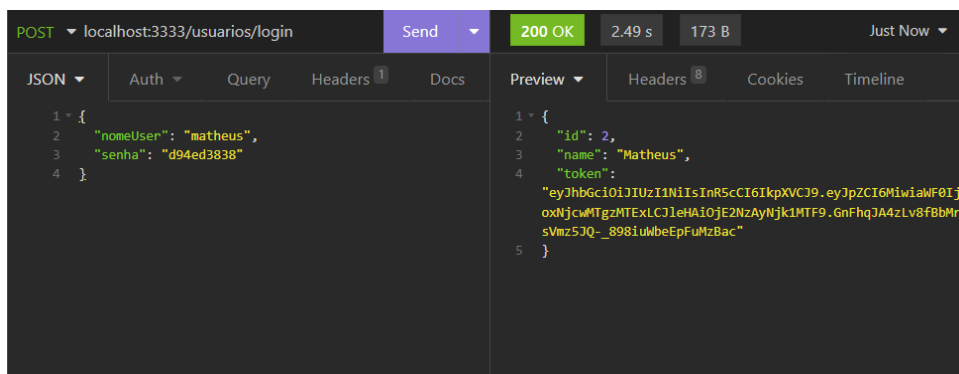
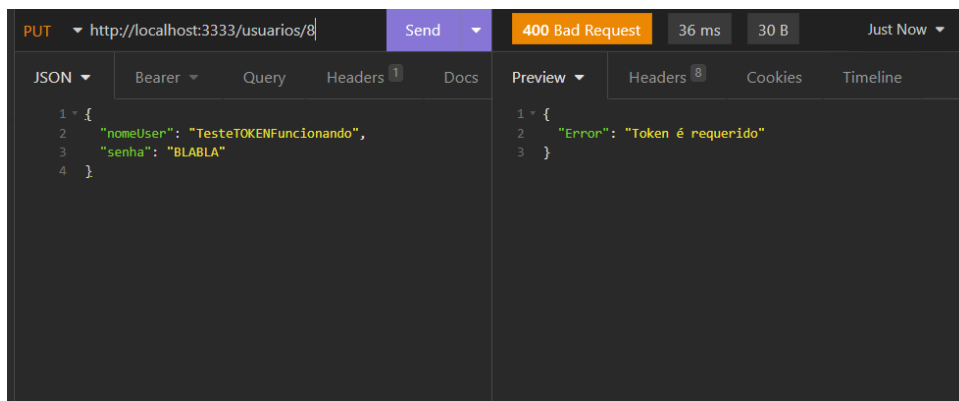
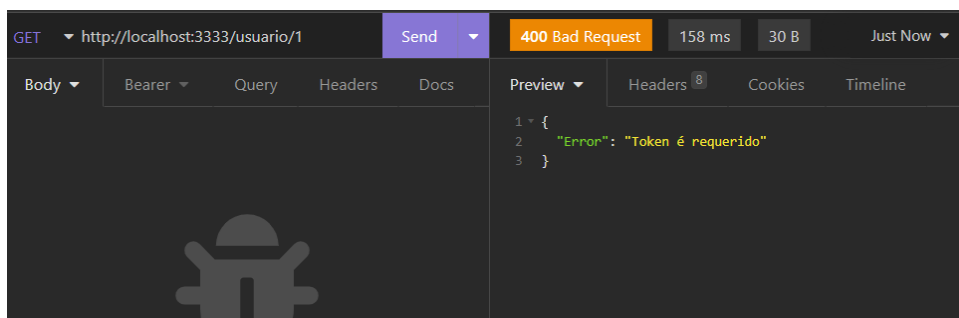


The screenshot displays the Insomnia API client interface. The top bar shows a GET request to `http://localhost:3333/usuarios` with a status of **200 OK**, a response time of **2.09 ms**, and a body size of **1073 B**. The left sidebar contains a search icon, a prompt to "Enter a URL and send to get a response", and a link to "Introduction to Insomnia". The main area is divided into two panels: "Body" on the left and "Preview" on the right. The "Body" panel shows a large bug icon and instructions to select a body type. The "Preview" panel displays the JSON response of the API call.

```
1 * [
2 * {
3   "idUser": 1,
4   "nomeUser": "testeToken",
5   "senha": "12345",
6   "perfilUser": 1,
7   "criacao_at": "2022-12-03T15:38:29.427Z",
8   "alteracao_at": "2022-12-03T15:38:29.427Z"
9 },
10 * {
11   "idUser": 2,
12   "nomeUser": "Matheus",
13   "senha": "d94ed3838",
14   "perfilUser": 1,
15   "criacao_at": "2022-12-03T16:12:14.012Z",
16   "alteracao_at": "2022-12-03T16:12:14.012Z"
17 },
18 * {
19   "idUser": 3,
20   "nomeUser": "Pedro H",
21   "senha": "1234xxx",
22   "perfilUser": 4,
23   "criacao_at": "2022-12-03T16:12:29.668Z",
24   "alteracao_at": "2022-12-03T16:12:29.668Z"
25 },
26 * {
27   "idUser": 4,
28   "nomeUser": "TesteSP",
29   "senha": "testeteste",
30 }
```

# Imagens

Continuação:



# Imagens

## Continuação:

GET ▼ http://localhost:3333/usuario/1 Send ▼ 200 OK 273 ms 149 B Just Now ▼

Body ▼ Bearer ▼ Query Headers Docs

ENABLED ☒

TOKEN eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImMi


PREFIX

Preview ▼ Headers 8 Cookies Timeline

```
1 {
2   "idUser": 1,
3   "nomeUser": "testeToken",
4   "senha": "12345",
5   "perfilUser": 1,
6   "criacao_at": "2022-12-03T15:38:29.427Z",
7   "alteracao_at": "2022-12-03T15:38:29.427Z"
8 }
```


GET ▼ http://localhost:3333/categorias Send ▼ 200 OK 4.97 ms 634 B Just Now ▼

Body ▼ Bearer ▼ Query Headers Docs



Enter a URL and send to get a response

Select a body type from above to send data in the body of a request

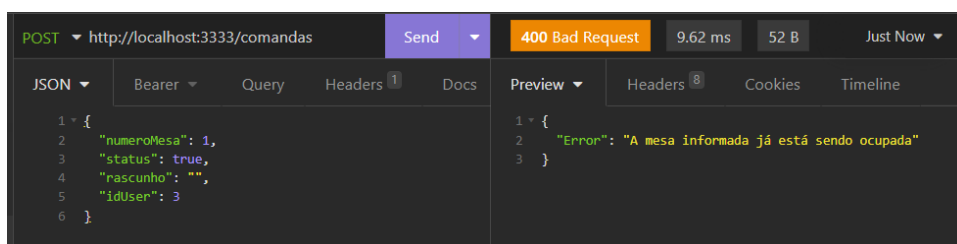
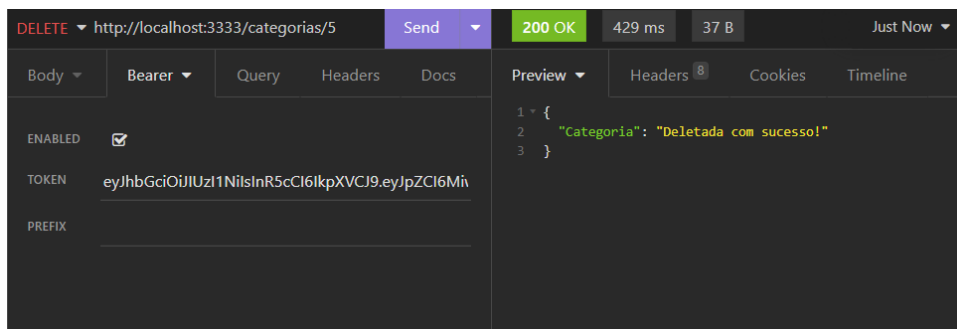
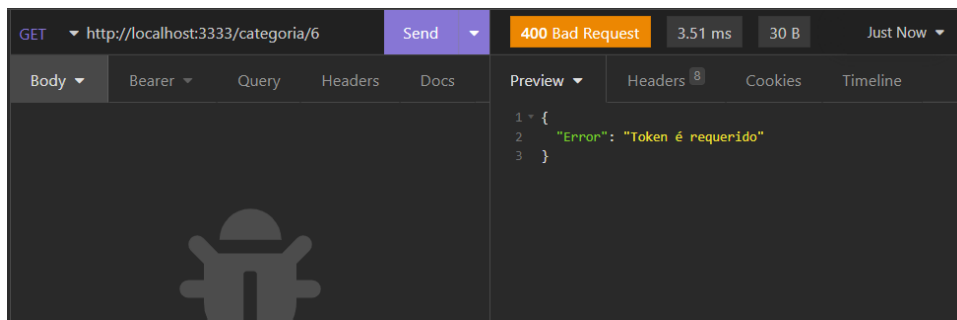
[Introduction to Insomnia](#) 

Preview ▼ Headers 8 Cookies Timeline

```
1 [
2   {
3     "idCategoria": 1,
4     "nomeCategoria": "testeSPSPSP",
5     "criacao_at": "2022-12-04T19:48:03.304Z",
6     "alteracao_at": "2022-12-04T19:48:03.304Z"
7   },
8   {
9     "idCategoria": 2,
10    "nomeCategoria": "Massas",
11    "criacao_at": "2022-12-04T19:48:17.620Z",
12    "alteracao_at": "2022-12-04T19:48:17.620Z"
13  },
14  {
15    "idCategoria": 3,
16    "nomeCategoria": "Frutas",
17    "criacao_at": "2022-12-04T19:48:21.667Z",
18    "alteracao_at": "2022-12-04T19:48:21.667Z"
19  },
20  {
21    "idCategoria": 4,
22    "nomeCategoria": "Bebidas",
23    "criacao_at": "2022-12-04T19:48:30.070Z",
24    "alteracao_at": "2022-12-04T19:48:30.070Z"
25  },
26  {
27    "idCategoria": 5,
28    "nomeCategoria": "Plantas",
29    "criacao_at": "2022-12-04T19:48:39.050Z",
30    "alteracao_at": "2022-12-04T19:48:39.050Z"
31  }
32 ]
```

# Imagens

## Continuação:



# Imagens

## Continuação:

POST http://localhost:3333/comandas Send 200 OK 123 ms 151 B Just Now

JSON Bearer Query Headers 1 Docs Preview Headers 8 Cookies Timeline

```
1 {
2   "numeroMesa": 2,
3   "status": true,
4   "rascunho": "",
5   "idUser": 3
6 }
```

```
1 {
2   "idComanda": 2,
3   "numeroMesa": 2,
4   "status": true,
5   "rascunho": "",
6   "idUser": 3,
7   "criacao_at": "2022-12-04T19:50:24.284Z",
8   "alteracao_at": "2022-12-04T19:50:24.284Z"
9 }
```

DELETE http://localhost:3333/comandas/3 Send 200 OK 297 ms 31 B Just Now

Body Bearer Query Headers Docs Preview Headers 8 Cookies Timeline

ENABLED ☒

TOKEN eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImMh

PREFIX

```
1 "Comanda deletada com sucesso!"
```

POST http://localhost:3333/produtos Send 200 OK 272 ms 196 B Just Now

JSON Bearer Query Headers 1 Docs Preview Headers 8 Cookies Timeline

```
1 {
2   "nomeProduto": "Limao",
3   "precoProduto": 9999.99,
4   "descricaoProduto": "Limão de ouro",
5   "idCategoria": 2
6 }
```

```
1 {
2   "idProduto": 1,
3   "nomeProduto": "Limao",
4   "precoProduto": "9999.99",
5   "descricaoProduto": "Limão de ouro",
6   "idCategoria": 2,
7   "criacao_at": "2022-12-04T19:51:12.831Z",
8   "alteracao_at": "2022-12-04T19:51:12.831Z"
9 }
```

POST http://localhost:3333/itens Send 400 Bad Request 7.75 ms 46 B Just Now

JSON Bearer Query Headers 1 Docs Preview Headers 8 Cookies Timeline

```
1 {
2   "idComanda": 3,
3   "idProduto": 3,
4   "quantidade": 10
5 }
```

```
1 {
2   "Error": "A comanda e o produto não existem"
3 }
```

# Imagens

## Continuação:

POST http://localhost:3333/itens Send 200 OK 189 ms 138 B Just Now

JSON Bearer Query Headers 1 Docs Preview Headers 8 Cookies Timeline

```
1 {
2   "idComanda": 1,
3   "idProduto": 1,
4   "quantidade": 10
5 }
```

```
1 {
2   "idItem": 1,
3   "idComanda": 1,
4   "idProduto": 1,
5   "quantidade": 10,
6   "criacao_at": "2022-12-04T19:53:48.132Z",
7   "alteracao_at": "2022-12-04T19:53:48.132Z"
8 }
```

GET http://localhost:3333/itens Send 200 OK 5.45 ms 140 B Just Now

Body Bearer Query Headers Docs Preview Headers 8 Cookies Timeline

ENABLED ☒

TOKEN eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Mi41bnR5cCI6IkpXVCJ9.eyJpZCI6Mi41bnR5cCI6IkpXVCJ9

PREFIX

```
1 [
2   {
3     "idItem": 1,
4     "idComanda": 1,
5     "idProduto": 1,
6     "quantidade": 10,
7     "criacao_at": "2022-12-04T19:53:48.132Z",
8     "alteracao_at": "2022-12-04T19:53:48.132Z"
9   }
10 ]
```

GET http://localhost:3333/item/1 Send 200 OK 5.82 ms 138 B Just Now

Body Bearer Query Headers Docs Preview Headers 8 Cookies Timeline

ENABLED ☒

TOKEN FhqJA4zLv8fBbMrsVmz5JQ-\_898iuWbeEpFuMzBac

PREFIX

```
1 {
2   "idItem": 1,
3   "idComanda": 1,
4   "idProduto": 1,
5   "quantidade": 10,
6   "criacao_at": "2022-12-04T19:53:48.132Z",
7   "alteracao_at": "2022-12-04T19:53:48.132Z"
8 }
```



# Repositório

Repositório: [SENAC-PWA107-1142496616-Matheus/Prova 02 at main · Matheus-FSantos/SENAC-PWA107-1142496616-Matheus · GitHub](#)