Pet Store

Índice

Introdução

OBJETIVO

Desenvolver uma aplicação baseada em bases de dados para uma loja de animais especializada em alimentos e equipamentos para animais de estimação.

PROJETO

Implementar uma API que funcione como back-end de uma loja online, lidando com várias funcionalidades como gestão de produtos, efetuação de compras e análise de vendas.

Introdução

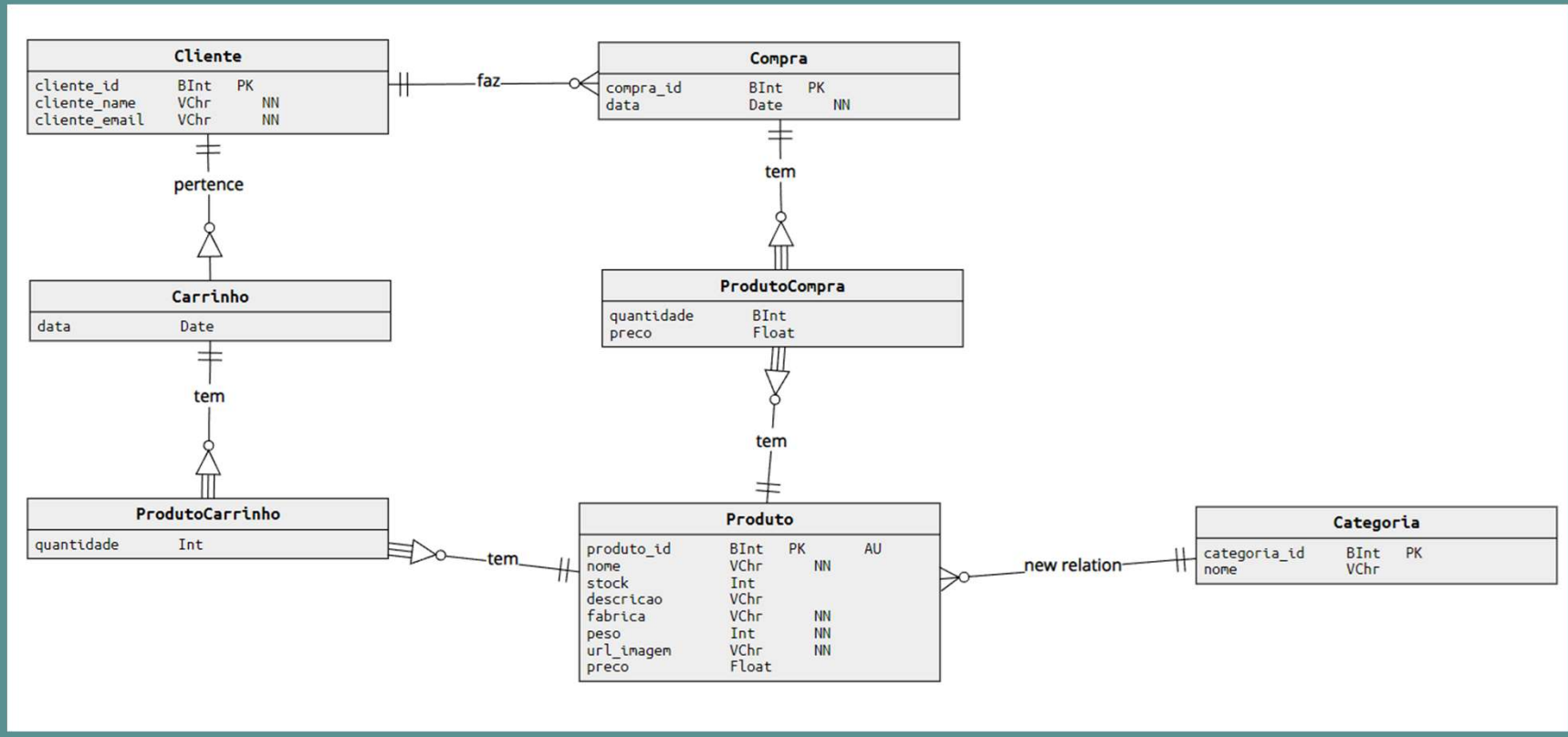
Características Principais

- ➔ Utilização de SGBD transacional e arquitetura de bases de dados distribuída.
- ➔ Enfoque na gestão eficiente de transações, segurança dos dados e resolução de conflitos.
- ➔ Documentação abrangente e abordagem inovadora na solução de problemas.

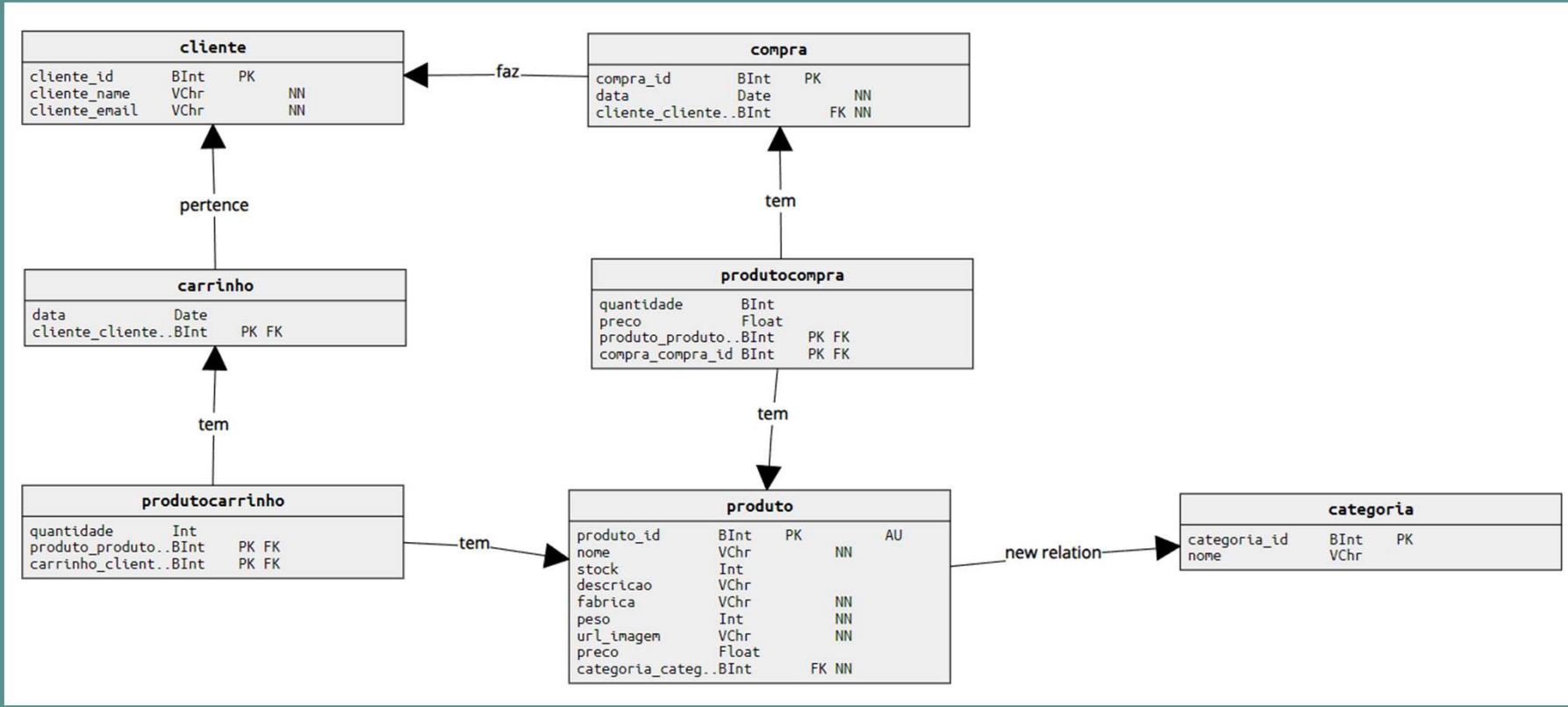
Descrição Técnica Resumida

- ➔ Desenvolvimento de uma API REST em Python com Flask, interligada a uma base de dados PostgreSQL.
- ➔ A API abrange a gestão de itens, operações de carrinho e análise de vendas.

Modelo ER


Cliente
cliente_id BInt PK
cliente_name VChr NN
cliente_email VChr NN
faz
Compra
compra_id BInt PK
data Date NN
pertence
tem
Carrinho
data Date
ProdutoCompra
quantidade BInt
preco Float
tem
tem
ProdutoCarrinho
quantidade Int
tem
Produto
produto_id BInt PK AU
nome VChr NN
stock Int
descricao VChr
fabrica VChr NN
peso Int NN
url_imagem VChr NN
preco Float
new relation
Categoria
categoria_id BInt PK
nome VChr

Modelo Físico



Descrição da estrutura do Software

Para estruturar o nosso projeto, recorremos a 2 scripts: “load_data.py” e “api.py”

“api.py”	“load_data.py”
• add_produto() • update_produto() • delete_item_carrinho() • add_item_carrinho() • get_produtos_list() • get_produto() • search_produto_keyword() • get_top_sales_per_category() • add_purchase()	❑ Onde criamos as tabelas da base de dados e inserimos os dados dos ficheiros csv. ❑ (Nota: para duas das tabelas da base de dados inserimos os dados com queries no pgAdmin)

Transações, Concorrências, Falhas e Erros

Transações

Temos transações no nosso código “api.py” em funções como “add_produto”, “update_produto”, “add_item_carrinho”, “add_purchase”;

Em **“add_produto”** recebemos os parâmetros que vamos querer inserir na tabela “produto” e verificamos se é recebido produto_id, se o stock é maior ou igual a 0 e se a quantidade e o peso são maiores do que 0 e, caso as condições sejam todas reunidas, os parâmetros vão ser inseridos em “produto”;

Em **“update_produto”**, recebemos os parâmetros que vamos atualizar na tabela “produto” e verificamos se o stock é maior ou igual a 0 e a quantidade e o peso são maiores do que 0 e no URL do endpoint especificamos o produto_id do produto que queremos atualizar;

Em **“add_item_carrinho”**, recebemos os parâmetros que vamos querer inserir na tabela “produtocarrinho” e verificamos se é recebido item_id e se quantity é maior do que 0. Decidimos adicionar a funcionalidade de escolher a qual carrinho é que vamos adicionar o produto;

Transações

1. Em **“add_purchase”**, recebemos os parâmetros que vamos querer inserir em várias tabelas. Começamos por verificar se estamos a receber um cart e um client_id e também se dentro de cart temos item_id e quantity.

1. Verificamos se na tabela “carrinho” já existe aquele client_id e, se existir, eliminamos o que já lá estava e inserimos uma nova linha com o mesmo client_id e a data atualizada.

1. Passamos a validar se a quantidade que estamos a pedir é inferior ao número de stock do produto e, se for, inserimos os novos produtos na tabela produtocarrinho (aproveitamos também para calcular o valor total dos items que temos no cart).

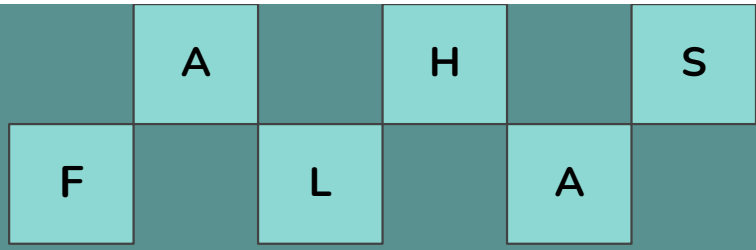
1. Inserimos agora na tabela “compra” o novo compra_id com o client_id que recebemos. Vamos agora inserir na tabela “produtocompra” o novo compra_id, com o produto_id e quantidade que recebemos e o respetivo preço do produto.

1. Por último, damos update à tabela “produto” para obtermos o novo stock de cada produto_id que foi comprado, ou seja, subtraímos do stock original a quantidade comprada e obtemos o novo stock do produto depois de efetuada a compra.

Concorrências

Quando tentámos inserir dados nas tabelas “produtocarrinho” e “produtocompra” com o script “load_data.py” tínhamos um erro de concorrência, pelo que tivemos que contornar e inserir dados com queries pelo pgAdmin;

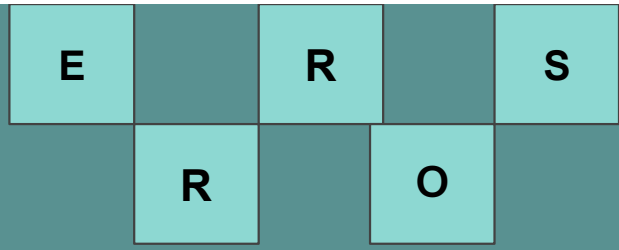
Também reparámos que se houvesse mais do que um cliente a tentar adicionar o mesmo produto ao carrinho em simultâneo poderíamos ter erros, dando por exemplo um erro no stock, em que poderia acontecer dois clientes estarem a comprar o mesmo produto ao mesmo tempo, o stock não estaria atualizado e, caso acabasse o stock, um dos clientes estaria a comprar um produto inexistente por estar fora de stock. Para isso, teríamos que fazer com que só pudesse ser efetuada uma compra depois da outra já ter dado commit para termos o stock atualizado.



FALHAS

Temos alguma validação nas nossas funções em relação aos parâmetros recebidos do postman, como por exemplo, na função de criar item, se não for recebido produto_id, stock <=0, preco e peso <0 vamos receber um aviso com aquilo que estamos a inserir mal. Porém, esta validação deveria ter sido feita para todos os outros parâmetros de uma forma semelhante, caso algum dos outros parâmetros não fosse inserido devíamos receber uma informação a dizer que faltou dar informação desse parâmetro.

Na função de dar update num item, além da validação similar da função de criar item, queríamos validar se o produto_id inserido no URL do endpoint está presente na tabela “produto”, que não seria feito da mesma maneira que foi feito anteriormente.



ERROS

Para a função de comprar um item, temos o erro de que se colocarmos o client_id de um cliente que não esteja na tabela “cliente” não vamos conseguir efetuar uma compra. Visto que não criámos nenhuma função para adicionar clientes, não conseguimos contornar este erro.

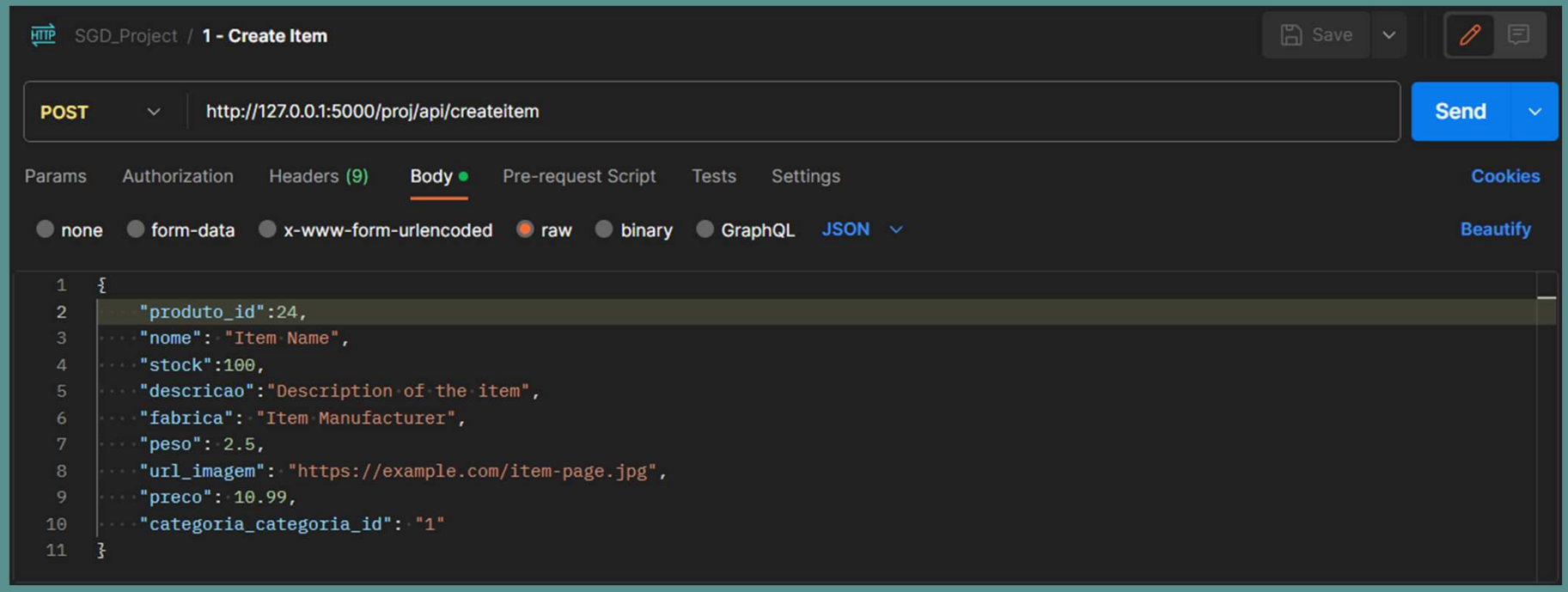
Ainda para a função de comprar um item, temos outro erro quando utilizamos um client_id que já está na coluna carrinho_cliente_cliente_id da tabela “produtocarrinho”, em que o erro consiste na unicidade de “produtocarrinho_pkey”(exemplo para “client_id”:1).

```
127.0.0.1 - - [29/Dec/2023 23:14:29] "POST /proj/api/createitem HTTP/1.1" 200 -
23:21:22 [INFO]:  POST /proj/api/purchase
23:21:22 [DEBUG]:  POST /proj/api/purchase - Data: {'cart': [{'item_id': 1, 'quantity': 2}, {'item_id': 2, 'quantity': 1}], 'client_id': 1}
23:21:22 [ERROR]:  POST /proj/api/purchase - error(carrinho): ERRO:  atualização ou exclusão em tabela "carrinho" viola restrição de chave estrangeira "produtocarrinho_fk2" em "pro
dutocarrinho"
DETAIL:  Chave (cliente_cliente_id)=(1) ainda é referenciada pela tabela "produtocarrinho".

23:21:22 [ERROR]:  POST /proj/api/purchase - error(produtocarrinho): ERRO:  duplicar valor da chave viola a restrição de unicidade "produtocarrinho_pkey"
DETAIL:  Chave (produto_produto_id, carrinho_cliente_cliente_id)=(1, 1) já existe.
```

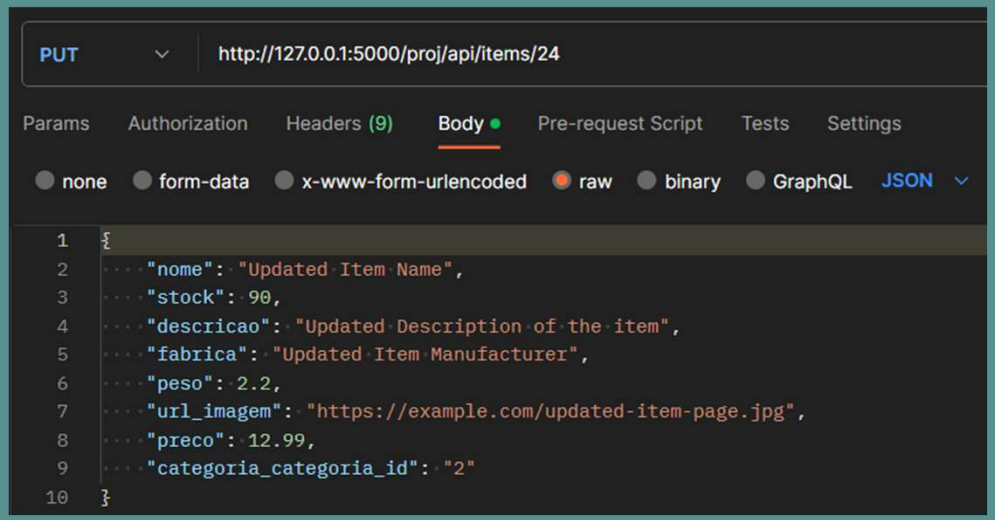
Exemplos

- Adicionar um produto

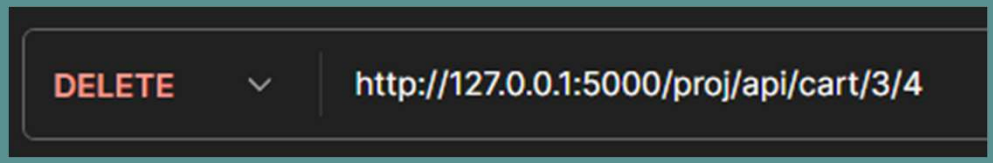


Exemplos

- Atualizar um produto

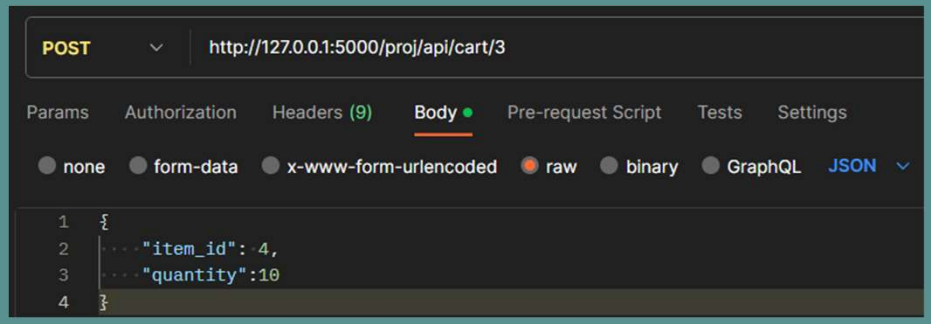


- Eliminar um produto de um carrinho

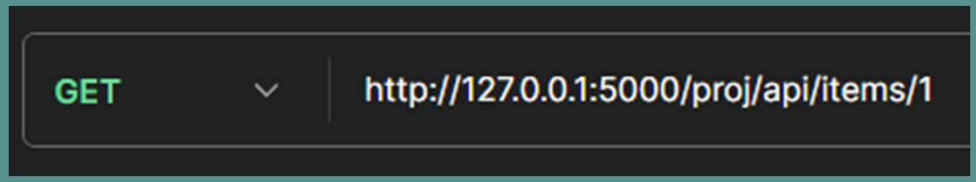


Exemplos

- Adicionar um produto a um carrinho

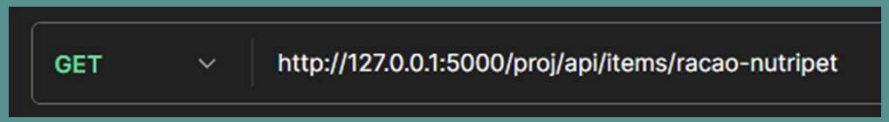


- Obter os detalhes de um produto:

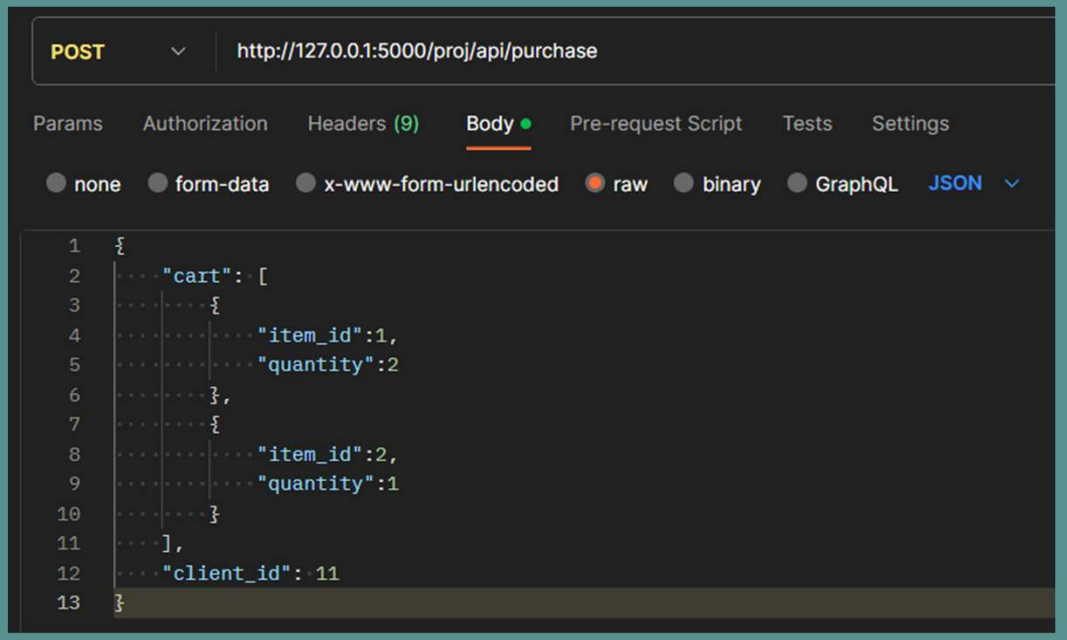


Exemplos

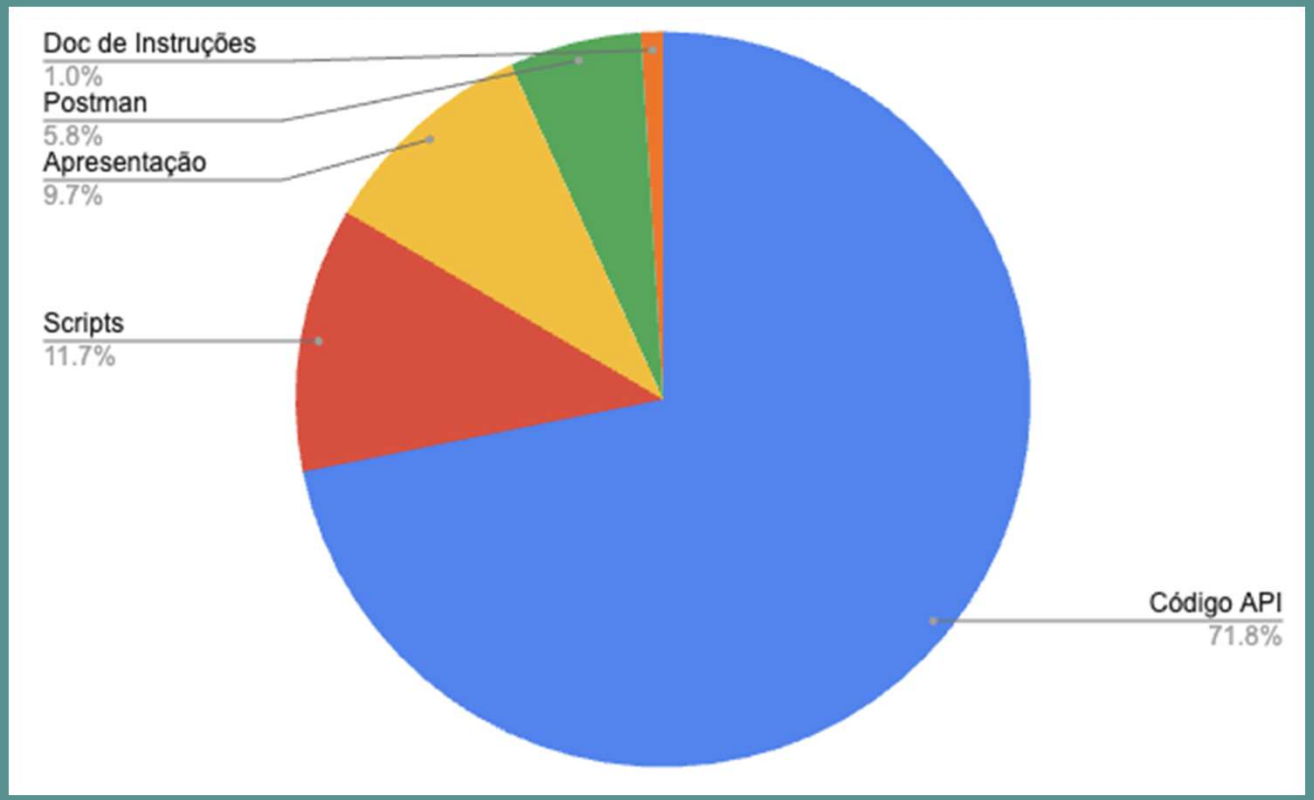
- Pesquisar um produto



- Compra de produtos



Tempo demorado na execução das tarefas



Tempo demorado na execução das tarefas

Pedro

Scripts

6 Horas

Ambos

API

Apresentação

Howto

37 Horas

6 Horas

30 Minutos

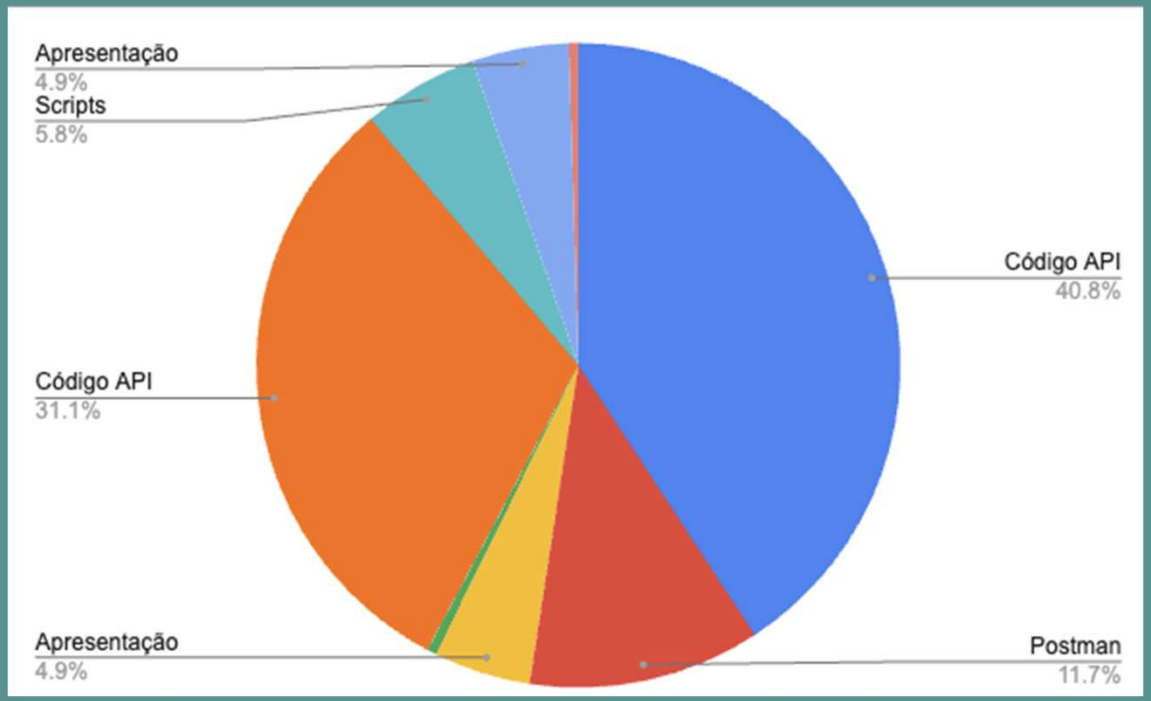
Rodrigo

Postman

3 Horas

Tempo demorado na execução das tarefas

Pedro



Rodrigo

Conclusão e Perguntas

Pedro Gaspar • 2021222324 • pedrogaspar2004@gmail.com

Rodrigo Santos • 2021236556 • rodrimiguelsantos12@gmail.com