

SOAT 7 - FIAP

Júlio Augusto Silva - RM355432

Lucas Henrique de Oliveira Silva - RM354904

Getúlio Magela Silva - RM355427

Lucas Rego Lima - RM356101

Lilian Rosario de Jesus - RM355928

Tech Challenge

São Paulo

2024

Projeto desenvolvido durante a Fase I do curso de arquitetura de software da FIAP como requisito para avaliação do aprendizado.

O Desafio:

Problema:

Há uma lanchonete de bairro que está expandindo devido seu grande sucesso. Porém, com a expansão e sem um sistema de controle de pedidos, o atendimento aos clientes pode ser caótico e confuso. Por exemplo, imagine que um cliente faça um pedido complexo, como um hambúrguer personalizado com ingredientes específicos, acompanhado de batatas fritas e uma bebida. O atendente pode anotar o pedido em um papel e entregá-lo à cozinha, mas não há garantia de que o pedido será preparado corretamente.

Sem um sistema de controle de pedidos, pode haver confusão entre os atendentes e a cozinha, resultando em atrasos na preparação e entrega dos pedidos. Os pedidos podem ser perdidos, mal interpretados ou esquecidos, levando à insatisfação dos clientes e a perda de negócios.

Em resumo, um sistema de controle de pedidos é essencial para garantir que a lanchonete possa atender os clientes de maneira eficiente, gerenciando seus pedidos e estoques de forma adequada. Sem ele, expandir a lanchonete pode acabar não dando certo, resultando em clientes insatisfeitos e impactando os negócios de forma negativa.

Proposta de Solução

Para solucionar o problema proposto, a lanchonete irá investir em um sistema de autoatendimento de fast food, que é composto por uma série de dispositivos e interfaces que permitem aos clientes selecionarem e fazerem pedidos sem precisar interagir com um atendente, autoatendimento.

Com o objetivo de aprimorar a experiência do cliente e assegurar uma gestão eficiente dos pedidos, desenvolvemos um sistema de controle de pedidos. Sua arquitetura centraliza-se em um aplicativo de autoatendimento intuitivo, capacitando os clientes a personalizarem seus pedidos de forma ágil e simples, selecionando entre uma variedade de lanches, acompanhamentos, bebidas e sobremesas.

Inicialmente, o sistema integrará apenas com a opção de pagamento via QR Code do Mercado Pago.

Para a solução foi utilizada arquitetura hexagonal, garantindo baixo acoplamento do sistema com componentes externos, como o Mercado Pago por exemplo e de DDD para garantir consistência do domínio, ou seja, o problema resolvido pelo sistema. Também foi utilizada a técnica de modelagem Event Storming, para permitir um mapeamento preciso da jornada de um pedido dentro dos sistemas.

Ferramentas de Desenvolvimento:

- Java como linguagem base de programação
- Spring como framework para criação de aplicações web
- Postgres como banco de dados
- Postman como navegador de APIs
- Insomnia como navegador de APIs
- Swagger como ferramenta para documentação como código
- Docker como ferramenta para isolamento e empacotamento da aplicação
- Docker Compose como ferramenta de execução dos serviços multicontainer (banco de dados e API)

O projeto está disponível para apreciação no Github <https://github.com/augustojulio-code/techchallengeFIAP> e os diagramas como Event Storming e Domain Story Telling no Miro https://miro.com/app/board/uXjVKYNMy0E=/. Abaixo algumas evidências de documentação e funcionamento da aplicação:

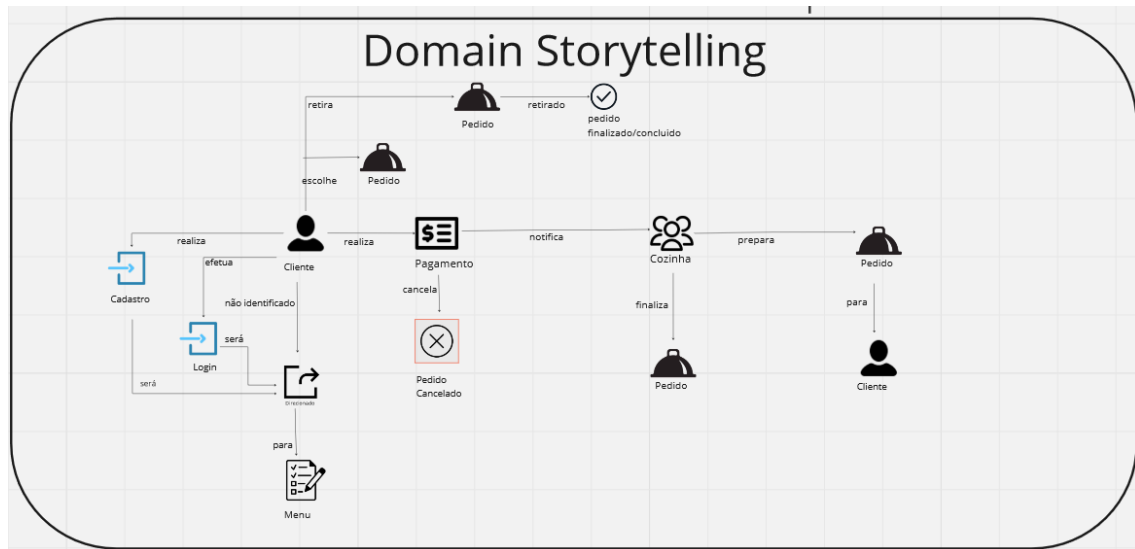
Demonstração:

Vídeo de demonstração disponível no Youtube:

<https://youtu.be/l0tNdbITFDc?si=XxHD3PUWE4XurWdN>

Documentação

Domain Storytelling



Linguagem Ubíqua

Linguagem Ubíqua

Usuário: Qualquer pessoa que faça uso do sistema

Cliente: Perfil utilizado para fazer pedidos

Funcionário: Perfil utilizado para atualizar status dos pedidos

Item: Lanche, acompanhamento, bebida, sobremesa

Pedido: Conjunto de produtos escolhidos

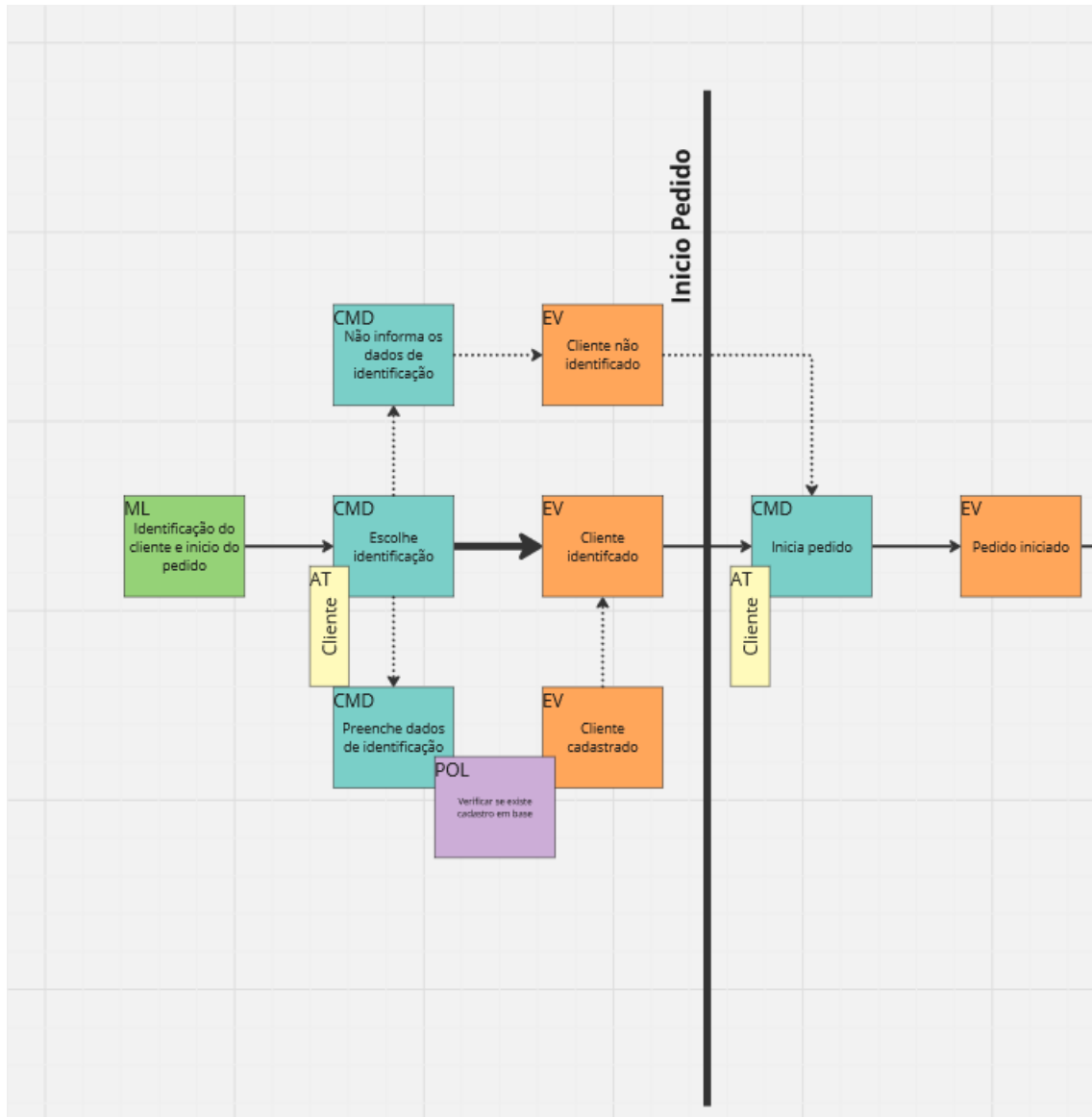
Mercado Pago: Sistema para efetuar pagamento do cliente

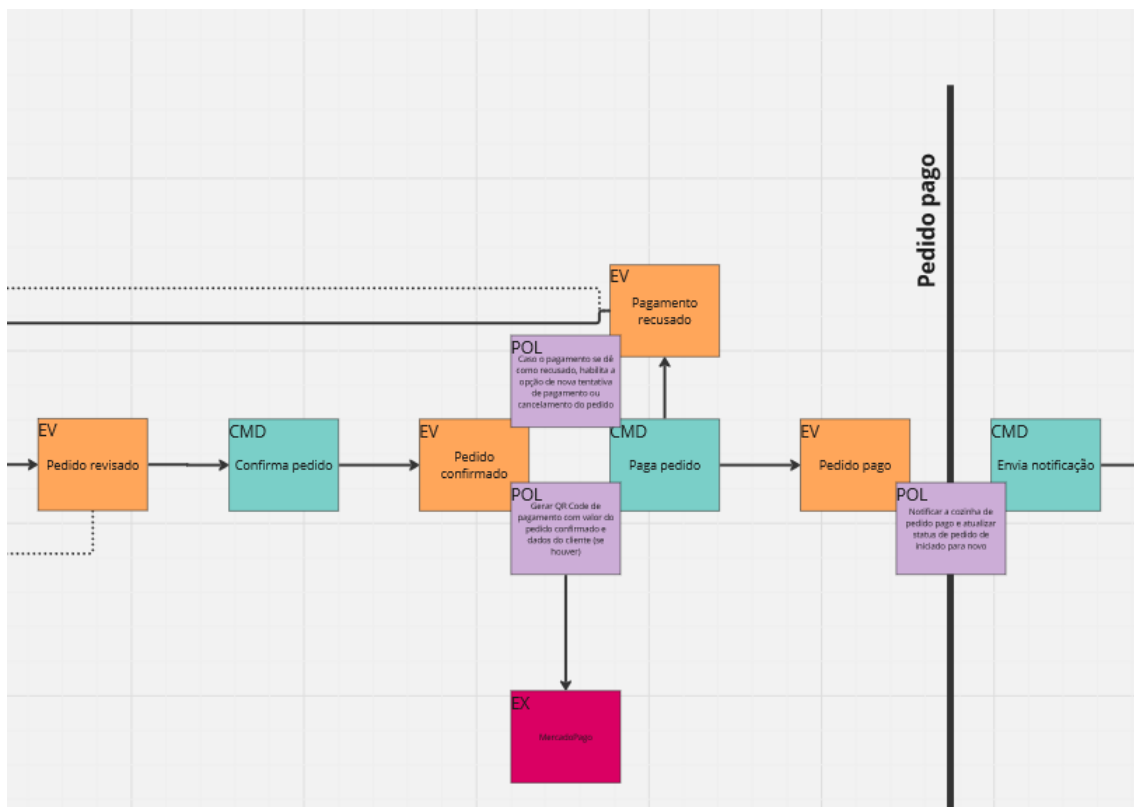
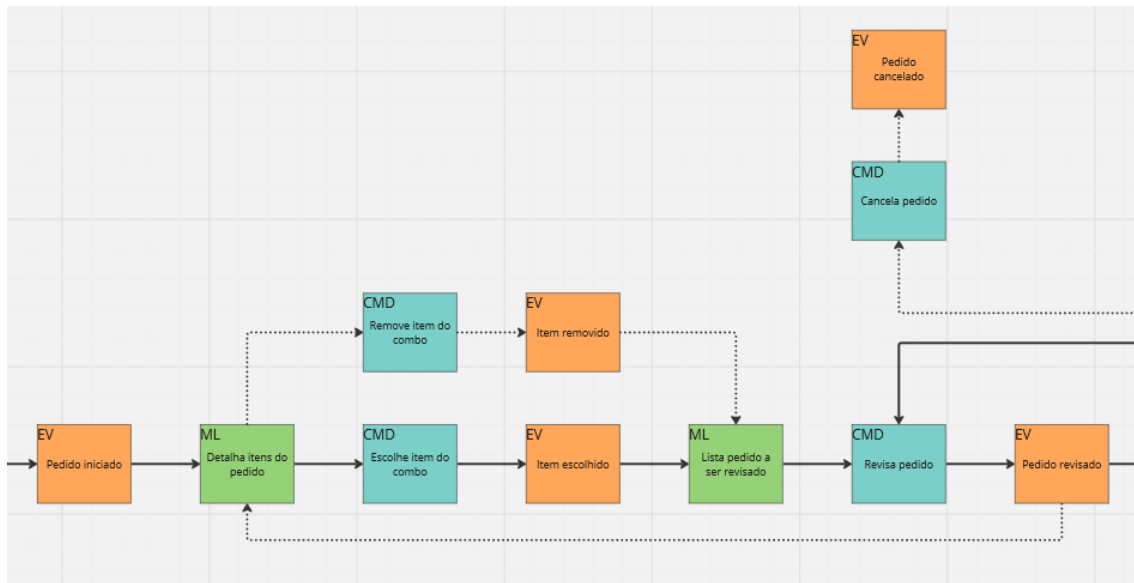
Classificação de Tipos de Domínio:

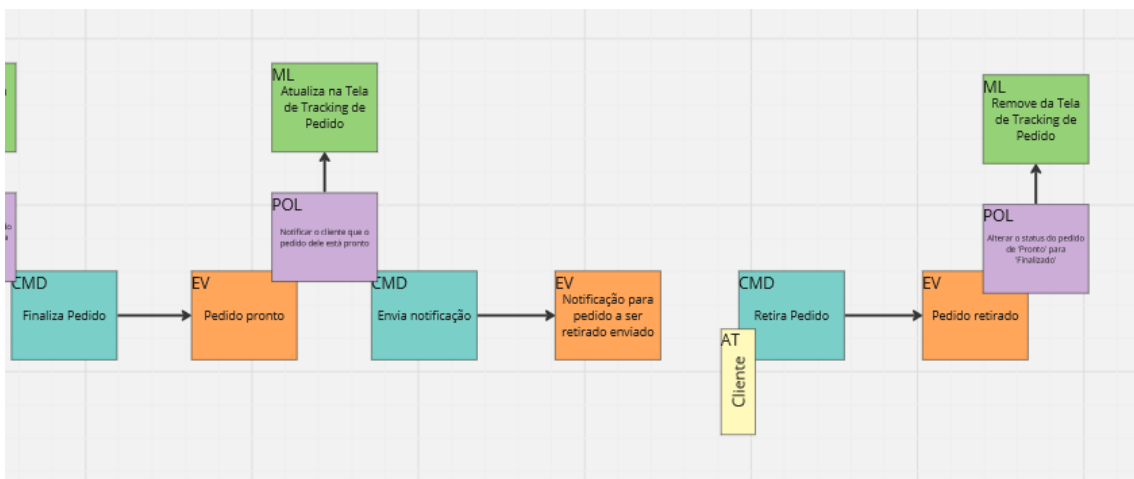
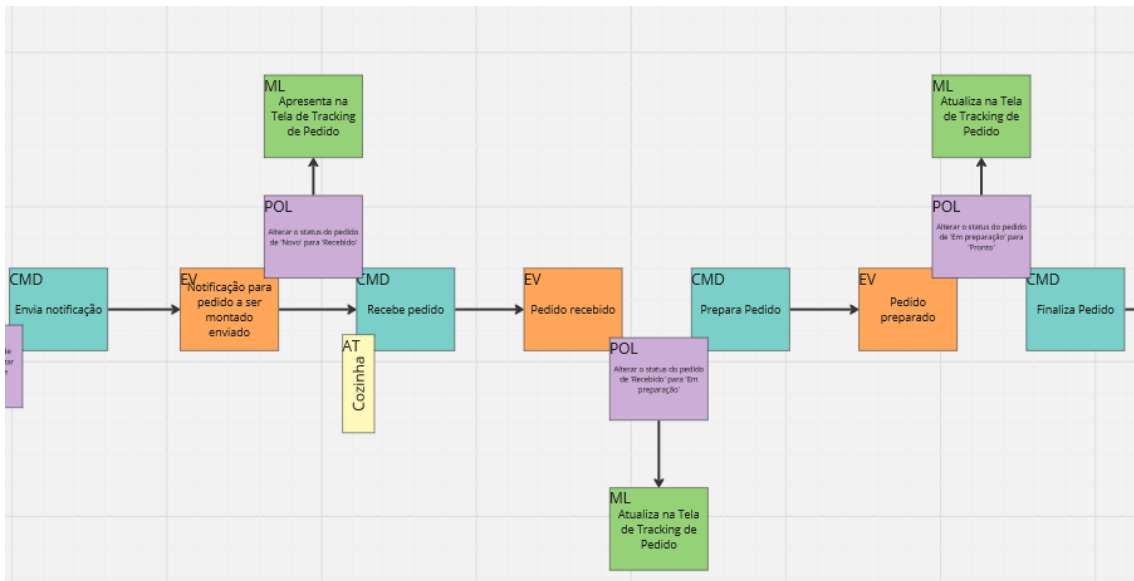


Event Storming:

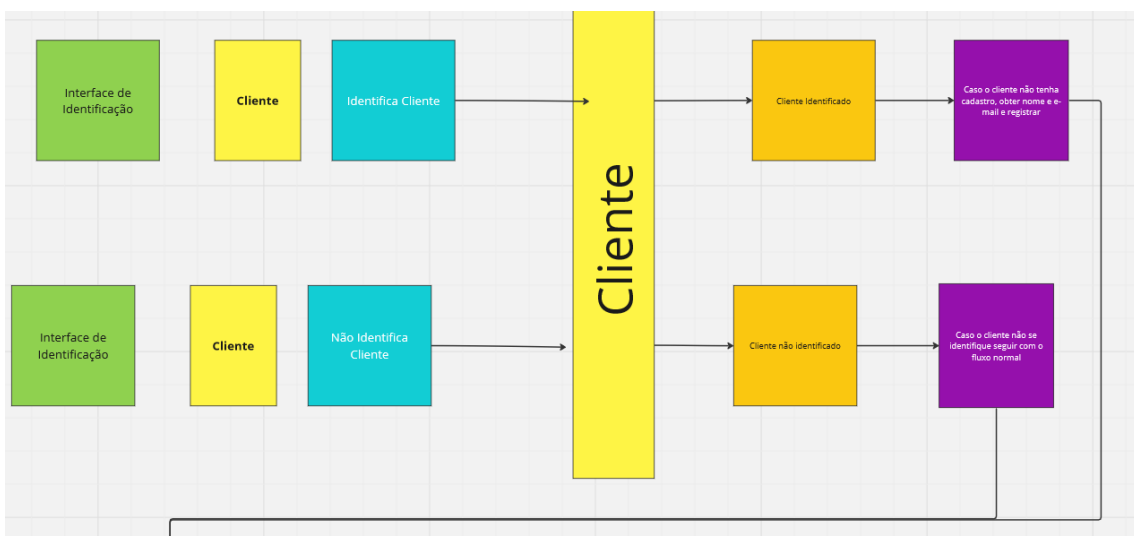
Como é um diagrama extenso, foi quebrado em partes. Deve ser lido de cima para baixo considerando a evolução ao longo do tempo.

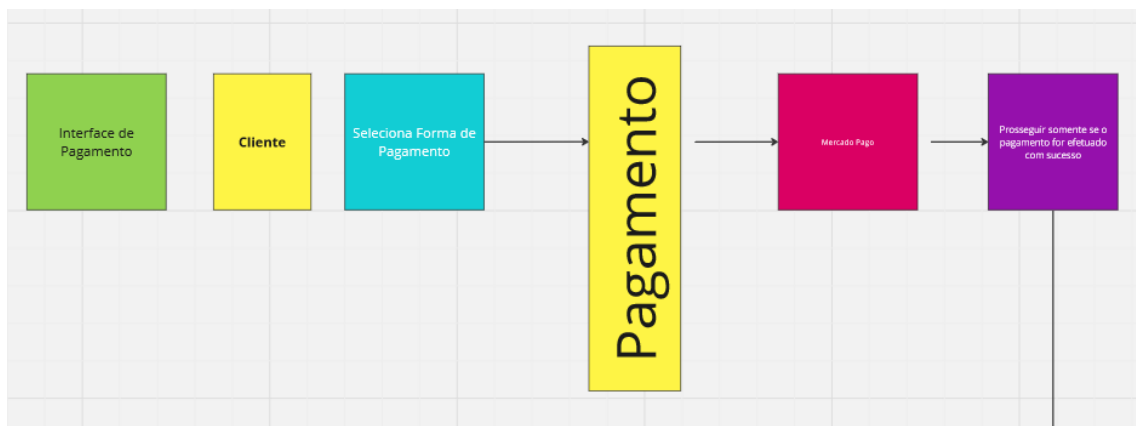
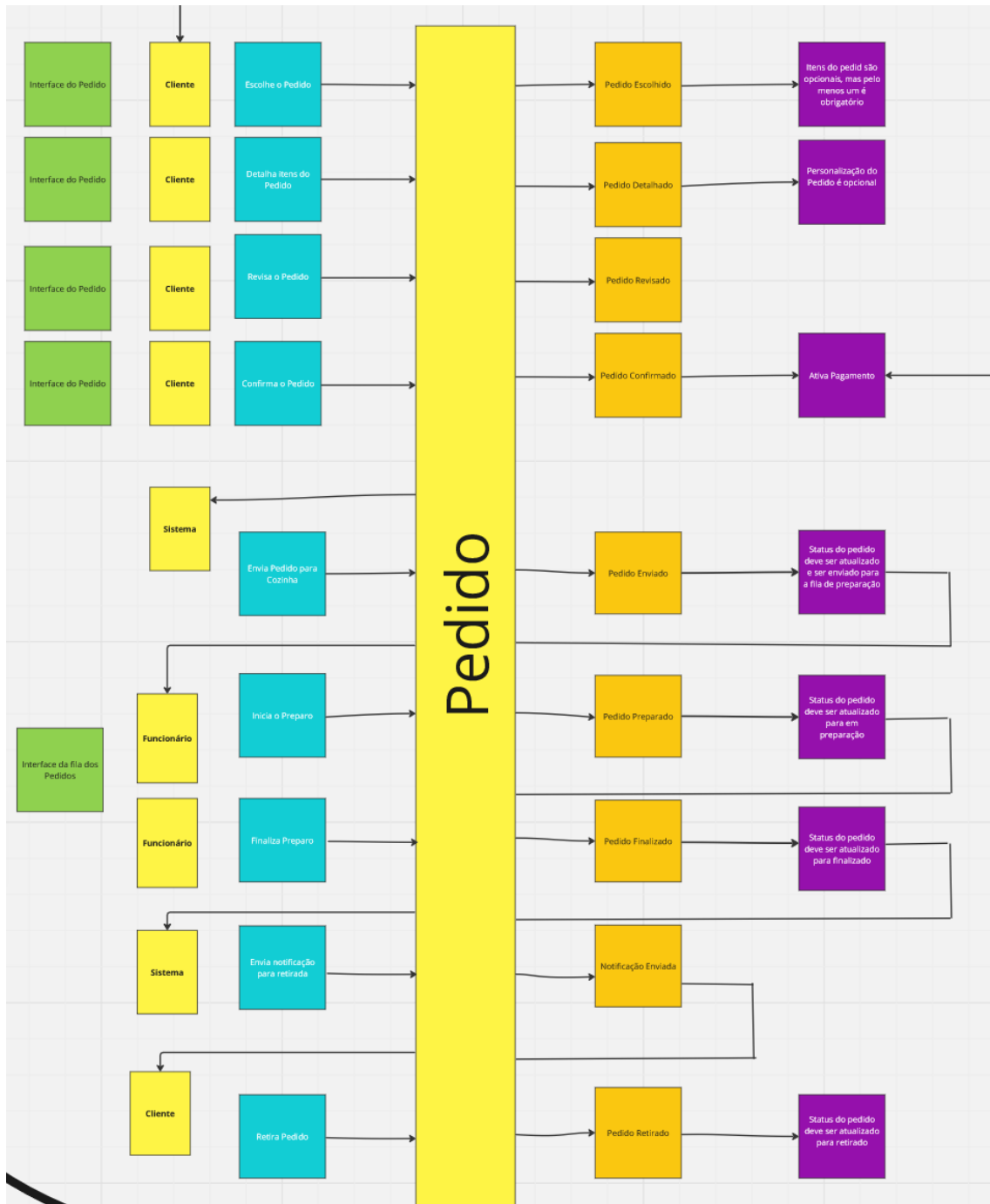






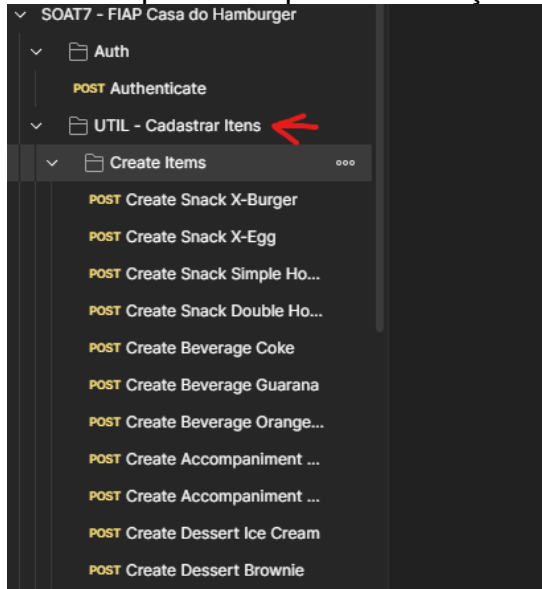
Modelagem de Agregados (DDD)



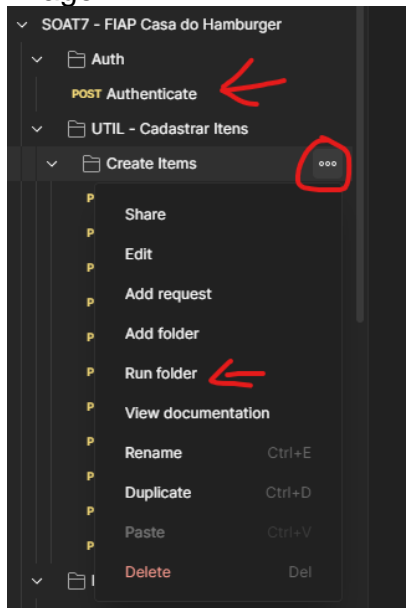


Funcionamento do Sistema

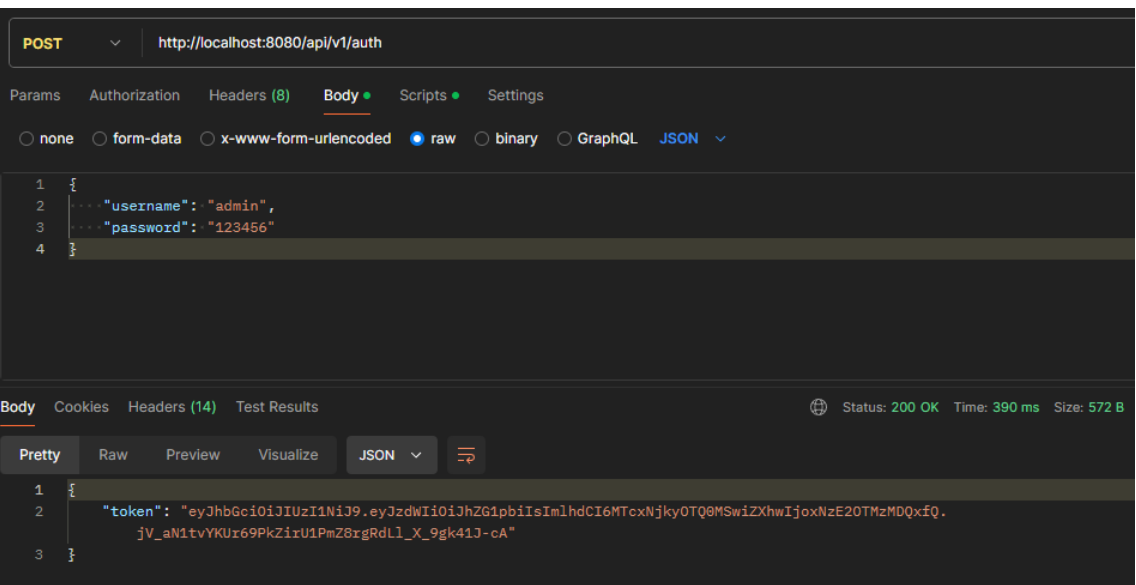
Para simplificar o processo de cadastro de itens durante a primeira utilização da plataforma, foi criada uma pasta dentro da coleção no Postman contendo vários itens disponíveis para realização de pedidos conforme imagem abaixo:



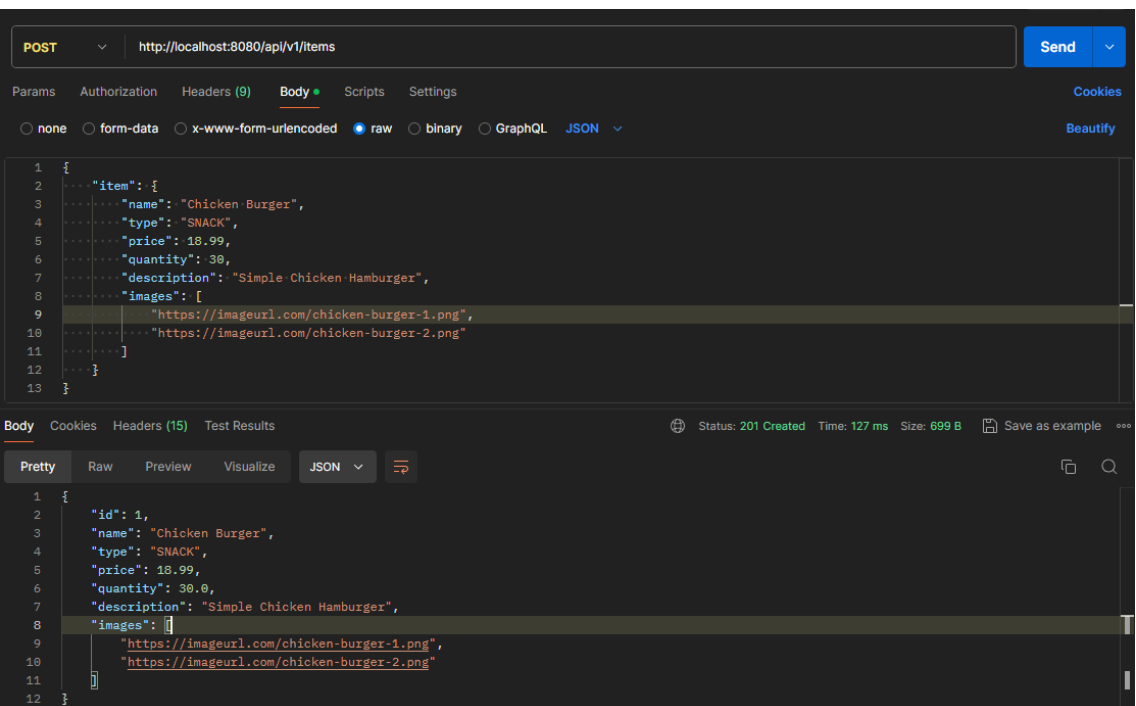
Esta pasta pode executada de uma vez com o Postman, bastando realizar a autenticação previamente e depois clicar em mais opções conforme esta imagem:



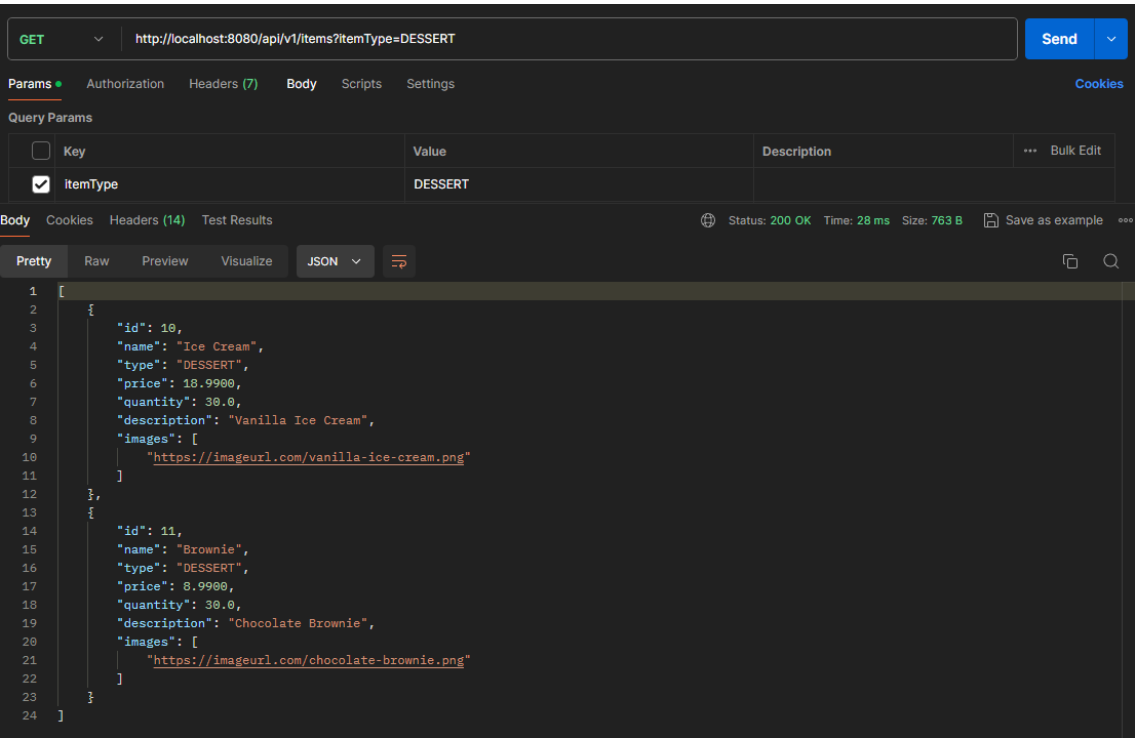
Processo de autenticação com Usuário e Senha para administração sistêmica:



Cadastro de Itens



Consulta de itens por categoria



GET <http://localhost:8080/api/v1/items?itemType=DESSERT> Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

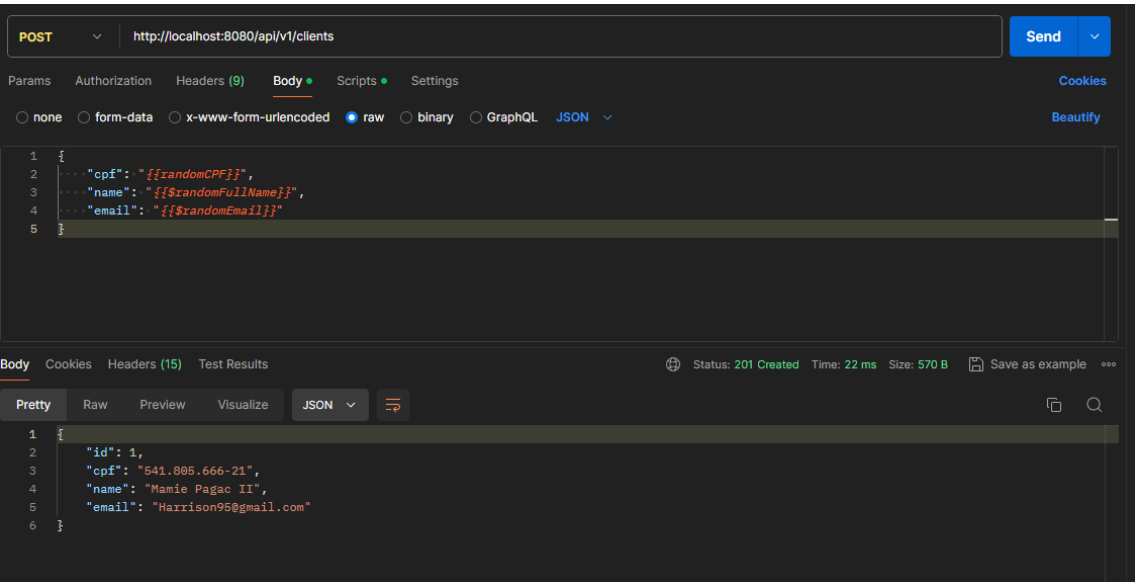
Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> itemType	DESSERT		

Body Cookies Headers (14) Test Results Status: 200 OK Time: 28 ms Size: 763 B Save as example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 10,
4     "name": "Ice Cream",
5     "type": "DESSERT",
6     "price": 18.9900,
7     "quantity": 38.0,
8     "description": "Vanilla Ice Cream",
9     "images": [
10      "https://imageurl.com/vanilla-ice-cream.png"
11    ]
12  },
13  {
14    "id": 11,
15    "name": "Brownie",
16    "type": "DESSERT",
17    "price": 8.9900,
18    "quantity": 38.0,
19    "description": "Chocolate Brownie",
20    "images": [
21      "https://imageurl.com/chocolate-brownie.png"
22    ]
23  }
24 ]
```

Cadastro de Clientes



POST <http://localhost:8080/api/v1/clients> Send

Params Authorization Headers (9) Body Scripts Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Beautify

```
1 {
2   "cpf": "{{$randomCPF}}",
3   "name": "{{$randomFullName}}",
4   "email": "{{$randomEmail}}"
5 }
```

Body Cookies Headers (15) Test Results Status: 201 Created Time: 22 ms Size: 570 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "cpf": "541.885.666-21",
4   "name": "Mamie Pagac II",
5   "email": "Harrison95@gmail.com"
6 }
```

Criação de Pedido

POST

http://localhost:8080/api/v1/orders

Send

Params

Authorization

Headers (9)

Body

Scripts

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

```
1 {
2   "combo": {
3     "items": [
4       {
5         "id": 1,
6         "name": "Hamburger",
7         "type": "SNACK",
8         "price": 7.99,
9         "quantity": 1
10      }
11    ]
12  }
13 }
```

Body

Cookies

Headers (15)

Test Results

Status: 201 Created

Time: 72 ms

Size: 742 B

Save as example

...

Pretty

Raw

Preview

Visualize

JSON

...

```
1 {
2   "id": 1,
3   "combo": {
4     "items": [
5       {
6         "id": 1,
7         "name": "X-Burger",
8         "type": "SNACK",
9         "price": 32.9900,
10        "quantity": 1.0,
11        "description": "Hamburger and Cheese",
12        "images": [
13          "https://imageurl.com/x-burger-1.png",
14          "https://imageurl.com/x-burger-2.png"
15        ]
16      }
17    ]
18  },
19  "amount": 7.990,
20  "status": "CREATED"
21 }
```

Atualização de Pedido – Inclusão de Mais Itens

The screenshot shows a REST client interface with a PATCH request to `http://localhost:8080/api/v1/orders/id`. The request body is a JSON object representing an order update. The top section shows the raw JSON, and the bottom section shows the same JSON formatted and beautified.

Request Method: PATCH
URL: `http://localhost:8080/api/v1/orders/id`
Send: [button]
Body: [selected tab]

Request Body (Raw):

```
1 {
2   "combo": {
3     "items": [
4       {
5         "id": 1,
6         "name": "Hamburger",
7         "type": "SNACK",
8         "price": 7.99,
9         "quantity": 1
10      }
11    ]
12  }
13 }
```

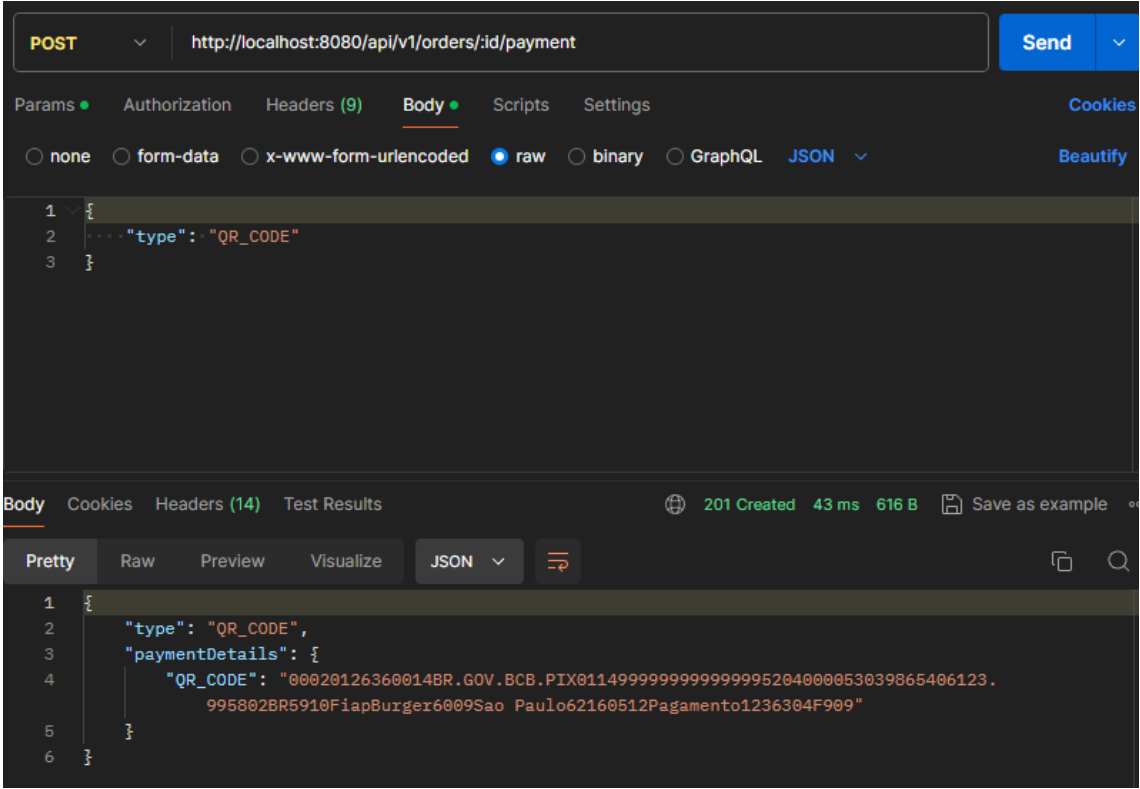
Response: 200 OK, 47 ms, 908 B
Body: [selected tab]

Response Body (Pretty):

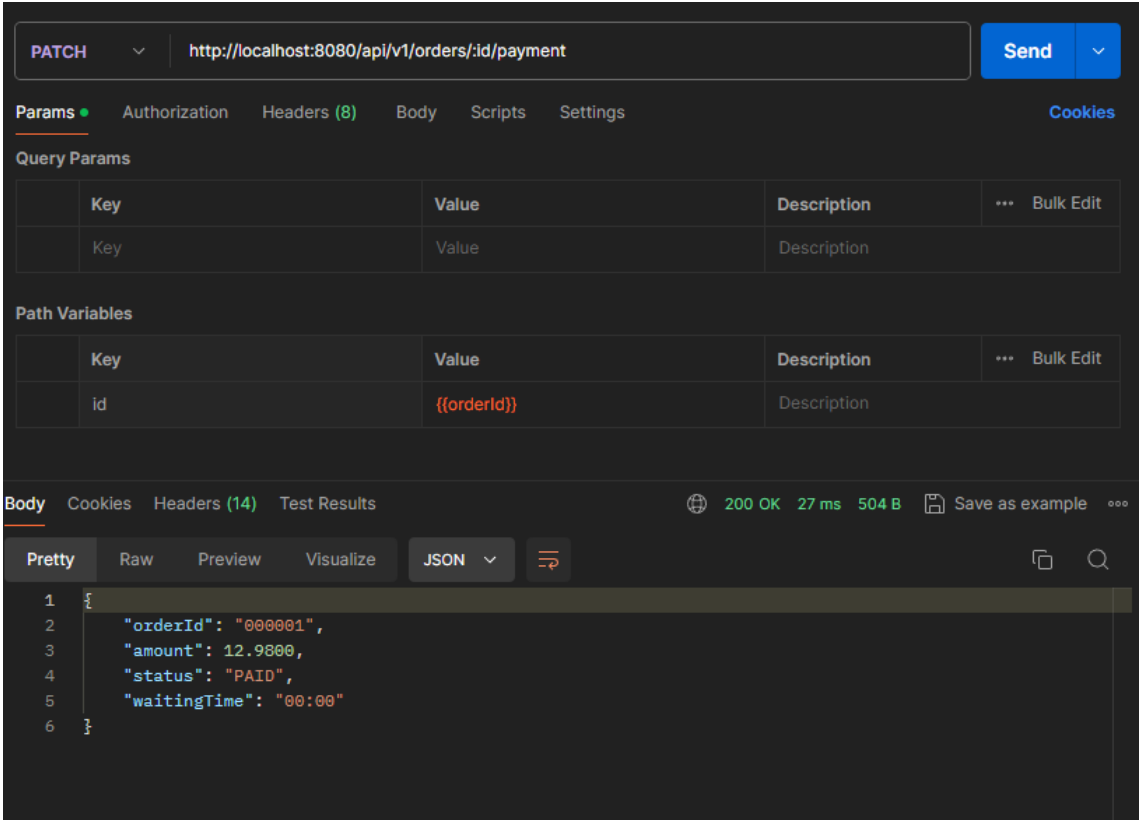
```
1 {
2   "id": 1,
3   "combo": {
4     "items": [
5       {
6         "id": 1,
7         "name": "X-Burger",
8         "type": "SNACK",
9         "price": 32.9900,
10        "quantity": 1.0,
11        "description": "Hamburger and Cheese",
12        "images": [
13          "https://imageurl.com/x-burger-1.png",
14          "https://imageurl.com/x-burger-2.png"
15        ]
16      },
17      {
18        "id": 2,
19        "name": "X-Egg",
20        "type": "SNACK",
21        "price": 42.9900,
22        "quantity": 1.0,
23        "description": "Hamburger with Cheese and Friend Eggs",

```

Confirmação de Pedido



Realização de Pagamento (Fake Checkout)



Follow Up

GET

http://localhost:8080/api/v1/followup

Send

Params

Authorization

Headers (7)

Body

Scripts

Settings

Cookies

Headers

7 hidden

	Key	Value	Description	...	Bulk Edit	Presets
	Key	Value	Description			

Body

Cookies

Headers (14)

Test Results

200 OK

18 ms

523 B

Save as example

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "RECEIVED": [
3     {
4       "orderId": "000003",
5       "amount": 37.9800,
6       "status": "RECEIVED",
7       "waitingTime": "00:02"
8     }
9   ]
10 }
```

Avançando o status de Pedido (até Pronto para Retirada)

PATCH

http://localhost:8080/api/v1/orders/id/status

Send

Params

Authorization

Headers (8)

Body

Scripts

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Path Variables

	Key	Value	Description	...	Bulk Edit
	id	{{orderId}}	Description		

Body

Cookies

Headers (14)

Test Results

200 OK

24 ms

505 B

Save as example

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "orderId": "000001",
3   "amount": 12.9800,
4   "status": "READY",
5   "waitingTime": "00:00"
6 }
```


API de Busca de Pedidos

GET

http://localhost:8080/api/v1/orders?from=2024-05-04&to=2024-05-31

Send

Params

Authorization

Headers (7)

Body

Scripts

Settings

Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input type="checkbox"/>	client_id	1			
<input checked="" type="checkbox"/>	from	2024-05-04			
<input checked="" type="checkbox"/>	to	2024-05-31			
<input type="checkbox"/>	status	CREATED			
	Key	Value	Description		

Body

Cookies

Headers (14)

Test Results

200 OK

19 ms

1.15 KB

Save as example

Pretty

Raw


Preview

Visualize

JSON

```
1 [
2   {
3     "id": 1,
4     "combo": {
5       "items": [
6         {
7           "id": 1,
8           "name": "X-Burger",
9           "type": "SNACK",
10          "price": 32.9900,
11          "quantity": 1.0,
12          "description": "Hamburger and Cheese",
13          "images": [
14            "https://imageurl.com/x-burger-1.png",
15            "https://imageurl.com/x-burger-2.png"
16          ]
17        },
18        {
19          "id": 2,
20          "name": "X-Egg",
21          "type": "SNACK"
```

Swagger

 **Swagger**
Supported by SMARTBEAR

/api-docs

Explore

OpenAPI definition v0 OAS 3.0

/api-docs

Servers

http://localhost:8080 - Generated server url

Get Order By Id Controller

Controller for return order data by his id

^

GET

/api/v1/orders/{id}

Return order data by his id

v

Update Order Controller

Controller for update order and save in database

^

PATCH

/api/v1/orders/{id}

Update Order

v

Update Item Controller

Controller for update item and save in database

^

PUT

/api/v1/items/{id}

Update Item

v

Create Item Controller

Controller for receiving item data and save it in the database

^

POST

/api/v1/items

Create item data

v

Search Order Controller

Controller for return order data

^

Schemas



```
ItemDto ▾ {  
  id           integer($int64)  
  name         string  
  type         string  
  price        number($float)  
  quantity     string  
  description  
  images       ▾ [string]  
}
```

RequestCreateItemDto >

RequestCreateUserDto >

UserDto >

ClientDto >

ComboDto >

RequestCreateOrderDto >

OrderDto >