



API Testing

Application Programming Interfaces

Aula 9

QA

Tech Fixe
prof Herzio Pinto

Carmen Cabral

2023

Índice

O QUE É API?	3
Classes de serviços da web para API	3
SOAP	3
REST	3
Como entender os requisitos da API?	3
O que é JSON?	4
Status de saída / Output Status	4
O que precisa ser verificado no código de status das APIs?	4
Como acessar uma API REST?	5
Por que testar API?	5
SWAGGER	5
Swagger Petstore (exemplo)	6
Como testar uma API	8
Testar API/POST	8
Como fazer uma atualização/PUT com menos erros possíveis	9
Rest Fundamentals	10
O que é API Rest?	10
Quais são os métodos 'request' HTTP mais populares?	11
Postman: download	12
Alterar tema do Postman	12
Criar coleção no Postman	12
Adicionar um Request no Postman	13
Editar um Request	13
TESTES MANUAIS	13
GET: LISTAR TODOS OBJETOS/LIST OF ALL OBJECTS	13
TESTES AUTOMATIZADOS	13
GET: LISTAR OBJETOS POR ID/LIST OF OBJECTS BY IDS	14
GET: ÚNICO OBJETO/SINGLE OBJECT	15
POST: ADICIONAR OBJETO/ADD OBJECT	16
PUT: ATUALIZAR OBJETO / UPDATE OBJECT	17
PATCH: ATUALIZAR PARCIALMENTE / PARTIALLY UPDATE OBJECT	17
DELETE: DELETAR OBJETO	18

O QUE É API?

Interface de programação de aplicativos: **intermediário** de software que permite que 2 aplicativos se comuniquem.

Forma acessível de extrair e compartilhar dados dentro de um software.

Classes de serviços da web para API

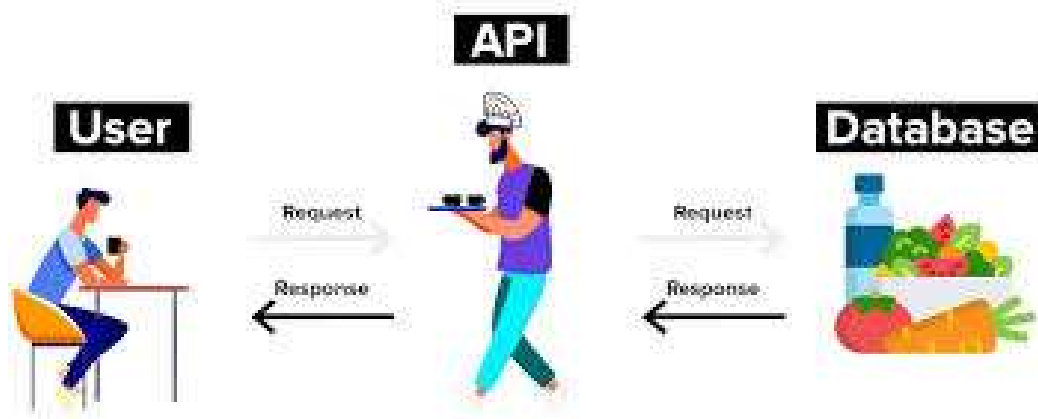
SOAP e REST

SOAP

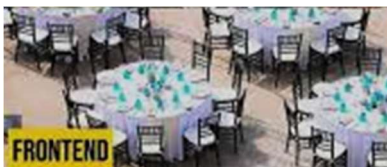
Simple Object Access Protocol. Protocolo padrão definido pelos padrões **W3C** para enviar e receber solicitações e respostas de serviços da web.

REST

Representational State Transfer. Arquitetura baseada em padrões da web que usa **HTTP**. Não existe um padrão oficial para APIs Web **RESTful**, como no SOAP. Será o mais utilizado na área de QA.



Teste de **Frontend** com a **API** e da **API** com o **Backend** para verificar se a ligação está correta.



Como entender os requisitos da API?

✓ **Entender qual é o propósito da API:** para preparar os dados de teste para entrada e saída e para definir a abordagem de verificação.

Ex. Uma API respondendo outra – numa compra on-line ao colocar o CEP, este será consultado (GET) no site/API dos Correios que irá responder.

API da compra para fazer um pagamento no Banco (POST)

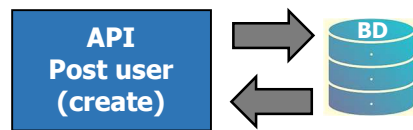
Métodos CRUD (operações do Banco de Dados)	Métodos HTTP/API
Create	Post
Read	Get
Update	Put (atualiza tudo) ou Patch (atualiza uma parte)
Delete	Delete

GET	/posts/ Get Posts
POST	/posts/ Create Posts
GET	/posts/{id} Get Post
PUT	/posts/{id} Update Post
DELETE	/posts/{id} Delete Post

✓ **Entender qual é o fluxo de trabalho da aplicação e onde está a API nesse fluxo:**

APIs são utilizadas para manipular seus recursos em leitura (GET), criação (POST), atualização (PUT) e exclusão (DELETE).

Algumas APIs verificam respostas no banco de dados (BD)



Outras APIs verificam respostas / responses em relação a outras APIs



O que é JSON?

JavaScript Object Notation.



```

{
  "Curso": {
    "Categoria": "Testes de Software",
    "Seção": "Testes de API",
    "Público-alvo": ["Estudantes", "Tester Junior", "Mudar Carreira"]
  }
}
  
```

"Categoria" é um **atributo**. "Teste de Software" é um **valor**.

Status de saída / Output Status

É o código de status de resposta das APIs / *API Status response code*

Há 5 classes ou categorias com padrão global, onde o 1º dígito do código de status define a classe de resposta e os 2 últimos dígitos não possuem nenhuma função de classe ou categorização.

1xx	INFORMATIVO	A solicitação é recebida e continua a ser processada.
2xx	BEM-SUCEDIDO	A solicitação foi recebida, compreendida e aceita com sucesso.
3xx	REDIRECIONAMENTO	Outras ações precisam ser tomadas para concluir a solicitação.
4xx	ERRO DO CLIENTE	A solicitação contém a sintaxe errada ou não pode ser atendida.
5xx	ERRO DO SERVIDOR	O servidor não atende a uma solicitação aparentemente válida.

Pergunta de entrevista:

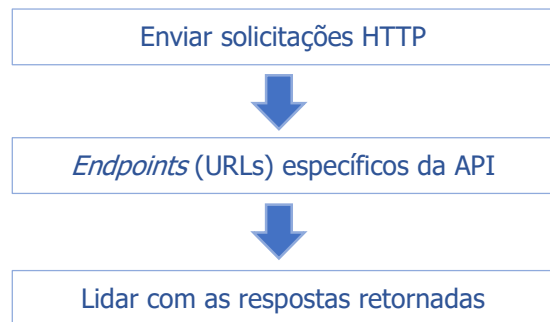
Qual é o erro 800? Não existe, pois só tem 5 tipos de erros.

O que precisa ser verificado no código de status das APIs?

- ✓ Se o código segue **classes padrão globais**.
- ✓ Se o código é especificado no **requisito**.

Apesar de existirem os padrões de 1xx a 5xx, os desenvolvedores irão colocar na documentação os valores esperados.

Como acessar uma API REST?



Roteamento		
Requisição	Path/Endpoints	Objetivo
GET	/api/products	Consultar todos produtos
GET	/api/products/4	Consultar o produto específico
POST	/api/products	Criar um produto
PUT	/api/products/7	Atualizar um produto específico
DELETE	/api/products/2012	Apagar um produto específico

Por que testar API?

- ✓ Frontend é 'burro';
- ✓ Regras de negócio estão no backend;
- ✓ Mais rápido para testar (execução e construção).

SWAGGER

Especificação para definir **REST API**.

<https://swagger.io/>

Schemes: HTTP Authorize

pet Everything about your Pets

- POST** /pet Add a new pet to the store
- PUT** /pet Update an existing pet
- GET** /pet/findByStatus Finds Pets by status
- GET** /pet/findByTags Finds Pets by tags
- GET** /pet/{petId} Find pet by ID
- POST** /pet/{petId} Updates a pet in the store with form data
- DELETE** /pet/{petId} Deletes a pet
- POST** /pet/{petId}/uploadImage uploads an image

store Access to Petstore orders

No **SWAGGER** estará a **documentação** da **API** que os desenvolvedores criaram.

Swagger Petstore (exemplo)

<https://petstore.swagger.io/>

Swagger Petstore

1.0.6 OAS 2.0

[Base URL: petstore.swagger.io/v2]
<https://petstore.swagger.io/v2/swagger.json>

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on <irc://freenode.net, #swagger>. For this sample, you can use the api key `special-key` to test the authorization filters.

[Terms of service](#)
[Contact the developer](#)
[Apache 2.0](#)
[Find out more about Swagger](#)

Schemes
HTTPS

Authorize

pet Everything about your Pets

Find out more ^

GET	/pet/{petId}	Find pet by ID	🔒	▼
POST	/pet/{petId}	Updates a pet in the store with form data	🔒	▼
DELETE	/pet/{petId}	Deletes a pet	🔒	▼
POST	/pet/{petId}/uploadImage	uploads an image	🔒	▼
POST	/pet	Add a new pet to the store	🔒	▼
PUT	/pet	Update an existing pet	🔒	▼
GET	/pet/findByStatus	Finds Pets by status	🔒	▼

store Access to Petstore orders

Find out more about our store ^

GET	/store/inventory	Returns pet inventories by status	🔒	▼
GET	/store/order/{orderId}	Find purchase order by ID		▼
DELETE	/store/order/{orderId}	Delete purchase order by ID		▼
POST	/store/order	Place an order for a pet		▼

user Operations about user

Find out more about our store ^

GET	/user/{username}	Get user by user name		▼
PUT	/user/{username}	Updated user		▼
DELETE	/user/{username}	Delete user		▼
POST	/user	Create user		▼
POST	/user/createWithList	Creates list of users with given input array		▼
GET	/user/login	Logs user into the system		▼
GET	/user/logout	Logs out current logged in user session		▼
POST	/user/createWithArray	Creates list of users with given input array		▼

Models

Category ▾ {	
id	integer(\$int64)
name	string
}	
Pet ▾ {	
id	integer(\$int64)
category	Category > {...}
name*	string
photoUrls*	example: doggie
tags	▾ [xml: OrderedMap { "wrapped": true } > [...]
status	▾ [xml: OrderedMap { "wrapped": true } Tag > {...}] string pet status in the store Enum: ▾ [available, pending, sold]
}	
Tag ▾ {	
id	integer(\$int64)
name	string
}	
ApiResponse ▾ {	
code	integer(\$int32)
type	string
message	string
}	
Order ▾ {	
id	integer(\$int64)
petId	integer(\$int64)
quantity	integer(\$int32)
shipDate	string(\$date-time)
status	string
complete	Order Status Enum: ▾ [placed, approved, delivered] boolean
}	
User ▾ {	
id	integer(\$int64)
username	string
firstName	string
lastName	string
email	string
password	string
phone	string
userStatus	integer(\$int32)
User Status	
}	

Como testar uma API

Authorize

Available authorizations

api_key (apiKey)
Name: api_key
In: header
Value:

Authorize **Close**

Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes. API requires the following scopes. Select which ones you want to grant to Swagger UI.

petstore_auth (OAuth2, implicit)
Authorization URL: <https://petstore.swagger.io/oauth/authorize>
Flow: implicit
client_id:

Scopes: [select all](#) [select none](#)
☐ read:pets
read your pets
☐ write:pets
modify pets in your account
Authorize **Close**

Testar API/POST

<https://petstore.swagger.io/#/pet/addPet>

POST /pet Add a new pet to the store

Parameters

Try it out

O que será testado?

Endpoint: /pet

Método: POST

1 Clicar em:

POST /pet Add a new pet to the store

2 Clicar em

Try it out

Antes de clicar em 'Try it out'

Parameters

Name	Description
body * required object (body)	Pet object that needs to be added to the store Example Value Model

```
{  "id": 0,  "category": {    "id": 0,    "name": "string"  },  "name": "doggie",  "photoUrls": [    "string"  ],  "tags": [    {      "id": 0,      "name": "string"    }  ],  "status": "available"}
```

Parameter content type
application/json

Depois de clicar em 'Try it out'

Parameters

Name	Description
body * required object (body)	Pet object that needs to be added to the store Edit Value Model

```
{  "id": 0,  "category": {    "id": 0,    "name": "string"  },  "name": "doggie",  "photoUrls": [    "string"  ],  "tags": [    {      "id": 0,      "name": "string"    }  ],  "status": "available"}
```

Cancel

Parameter content type
application/json

DTO: Data Transfer Object

Padrão de objeto usado para transportar dados de um local para outro na solução.

Test procedure

User history: As a [role] I want to use API to ...

Conditions of satisfaction:

GIVEN ...

AND ...

Fazer as alterações e clicar em **Execute**

The screenshot shows a REST client interface. On the left, a JSON request body is defined with the following structure:

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "Lulu",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Below the request is a blue button labeled "Execute". On the right, the "Server response" is displayed. It shows a status code of 200 (labeled "Undocumented") and a response body that is a duplicate of the request JSON, with the "id" field updated to "9223372036854776000".

Consultar pelo id o pet criado

Copiar o id '9223372036854776000'

Clicar em **GET** `/pet/{petId}` Find pet by ID e depois em **Try it out**
 Colar o id '9223372036854776000' e clicar em **Execute**

The screenshot shows a REST client interface. On the left, under "Parameters", a parameter named "petId" is defined as "ID of pet to return" with a type of "integer (\$int64)" and a path. Its value is set to "9223372036854776000". On the right, the "Server response" shows a status code of 404 with the message "Error: response status is 404". The response body contains an error object:

```
{
  "code": 404,
  "type": "unknown",
  "message": "java.lang.NumberFormatException: For input string: \"9223372036854776000\""
}
```

404: Neste caso, é um possível bug porque o pet foi criado anteriormente.

Como fazer uma atualização/PUT com menos erros possíveis

1 Consultar/GET o pet a ser alterado**2 Copiar o objeto**

```
{
  "id": 11,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "Puff",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
},
```

3 Clicar em **PUT** /pet Update an existing pet, depois em **Try it out**, fazer as alterações e clicar em **Execute**

Antes da alteração do nome

PUT /pet Update an existing pet

Parameters

Name	Description
body * required	Pet object that needs to be added to the store

object (body)

Edit Value | Model

```
{
  "id": 11,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "Puff",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Cancel

Parameter content type
application/json

Depois da alteração do nome

```
{
  "id": 11,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "Puff Daddy",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Após executar

Code Details

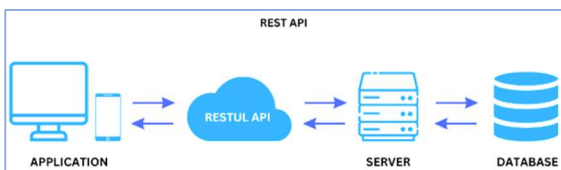
200 Undocumented

Response body

```
{
  "id": 11,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "Puff Daddy",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Rest Fundamentals

<https://restful-api.dev/rest-fundamentals#rest>



O que é API Rest?

REST: Representation State Transfer.

É um estilo de **arquitetura** para uma API que usa o método 'request' HTTP para acessar e manipular dados na internet.

Quais são os métodos *'request'* HTTP mais populares?

GET	Obtém dados de um banco de dados. A resposta/ <i>response</i> pode conter uma lista de itens, um item simples ou até mesmo uma mensagem de status. Pode ser repetido várias vezes sem ter efeitos colaterais porque somente recupera dados, não os modifica.
POST	É usado para enviar dados para o servidor de um cliente para criar um novo recurso. Os dados que são enviados como parte de um POST são codificados/ <i>encoded</i> no corpo e não está visível na URL, ao contrário de/ <i>unlike with</i> uma solicitação/ <i>request</i> GET.
PUT	Permite atualizar recursos existentes. O corpo solicitado deve conter uma representação completa do recurso, ou seja, todos os campos deverão ser colocados, mesmo que queira alterar só um ou alguns, caso contrário / <i>otherwise</i> , os campos que faltam serão atualizados com NULL ou a solicitação irá falhar. Se quiser atualizar parte de um recurso, use PATCH.
DELETE	É usado quando se pretende excluir um recurso existente do servidor. Geralmente, é especificado um recurso que você quer excluir fornecendo o ID de um recurso como parte de um parâmetro da URL.
PATCH	Parecido com o PUT HTTP request, pode ser usado para atualizar um recurso existente com uma representação parcial do recurso.
HEAD	É usado para buscar/ <i>fetch</i> os cabeçalhos que seriam retornados se uma solicitação/ <i>request</i> GET correspondente fosse feita. Pode ser útil em situações com alguma largura de banda/ <i>bandwidth</i> quando você precisa recuperar/ <i>retrieve</i> alguns metadados retornados pelo HEAD, pode ser usado para validar a informação sobre os recursos, como Content-length, Content-type, Last-modified date. Pode também ser usado para verificar se o recurso existe, antes de procurá-lo ou quando quiser verificar quando o recurso foi modificado pela última vez.
TRACE	Pode ser usado para depuração quando se quer determinar o que está acontecendo realmente com sua solicitação HTTP. Não deve ser habilitado num ambiente de produção porque em alguns cenários eles podem revelar algumas informações sensíveis sobre o servidor .
CONNECT	Pode ser usado para estabelecer uma conexão à rede entre um cliente e um servidor para criar um túnel seguro. Após a conexão, cliente e servidor podem se comunicar diretamente e encaminhar pacotes de dados um para o outro.
OPTIONS	Pode ser usado para solicitar opções de comunicação disponível do servidor para o recurso alvo. O servidor pode incluir a lista de métodos HTTP permitidos para o recurso de destino como parte do <i>header</i> <i>'Allow'</i> na solicitação.

<https://restful-api.dev/rest-fundamentals#rest>

Postman: download

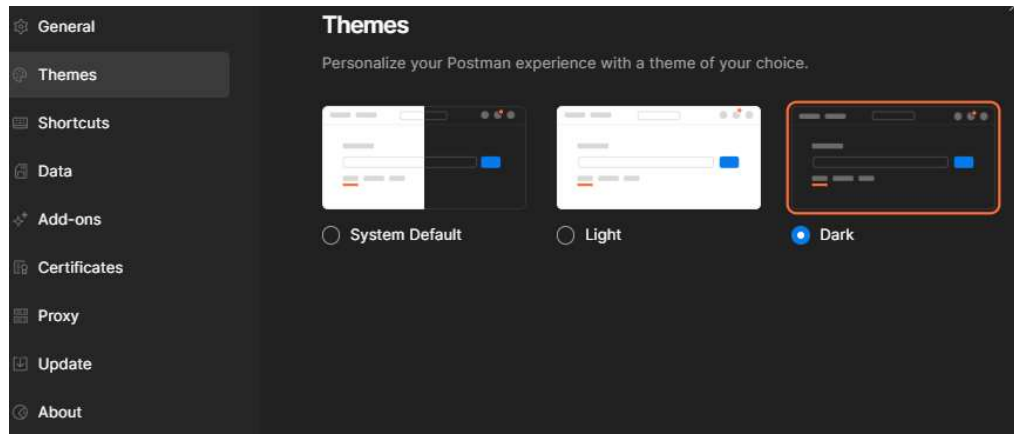
<https://www.postman.com/downloads/>

The Postman app

Download the app to get started with the Postman API Platform.

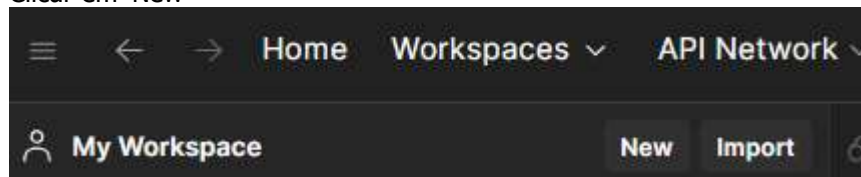


Alterar tema do Postman

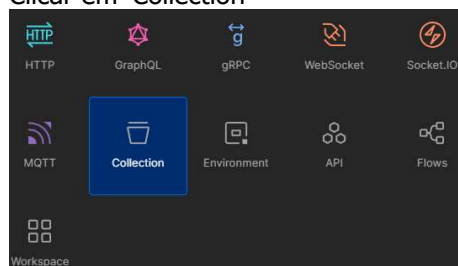


Criar coleção no Postman

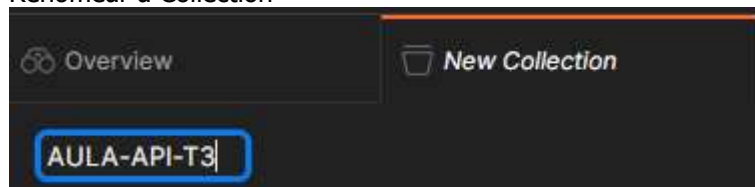
Clicar em 'New'



Clicar em 'Collection'



Renomear a Collection



O que é uma Collection?

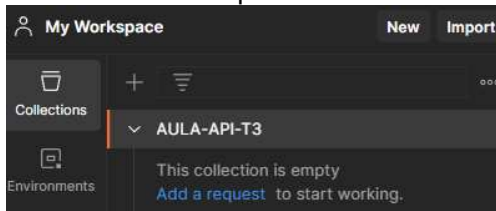
É uma coleção dos métodos request: POST, GET, PUT/PATCH, DELETE

A **Collection** é o **caso de teste** (describe do Cypress)

Cada **método 'request'** é um **cenário de teste** (it do Cypress)

Adicionar um Request no Postman

Clicar em 'Add a request'



Editar um Request

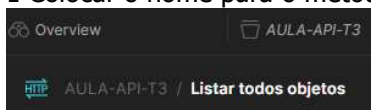
Entrar no site <https://restful-api.dev/>

Consultar a lista de **endpoints**.

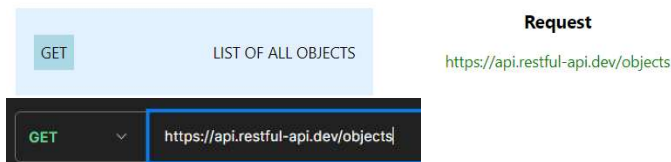
TESTES MANUAIS

GET: LISTAR TODOS OBJETOS/LIST OF ALL OBJECTS

1 Colocar o nome para o método GET



2 Copiar a URL (<https://api.restful-api.dev/objects>) + endpoint (objects) do site <https://restful-api.dev/>

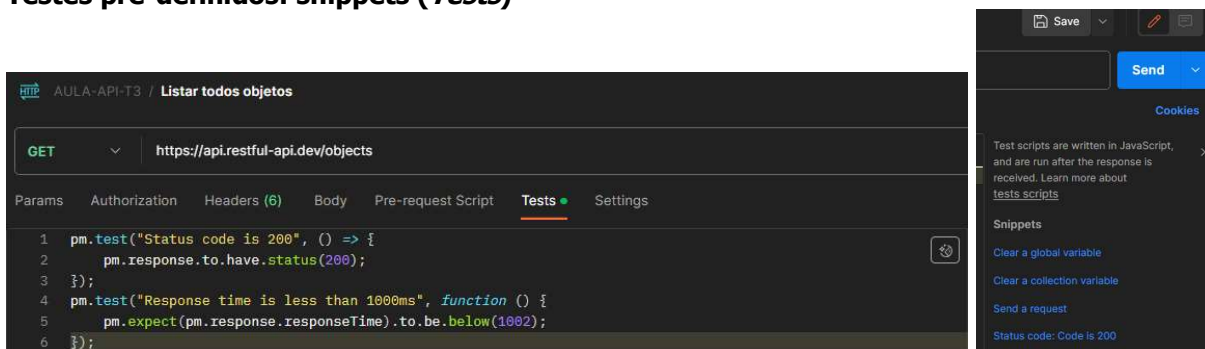


3 Clicar em

Send

TESTES AUTOMATIZADOS

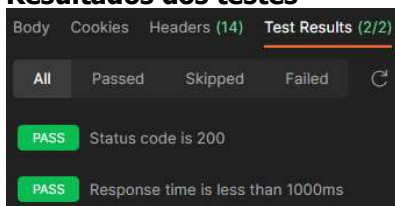
Testes pré-definidos: snippets (Tests)



Salvar alterações

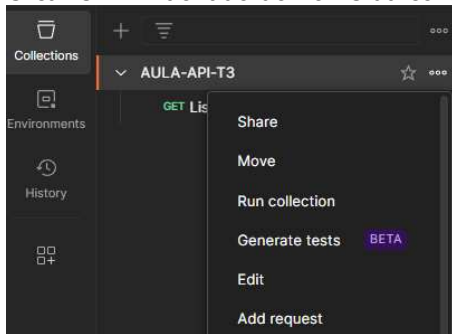


Resultados dos testes



CRIAR UM NOVO REQUEST

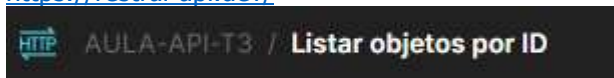
Clicar em '...' ao lado do nome da collection e depois em 'Add request'



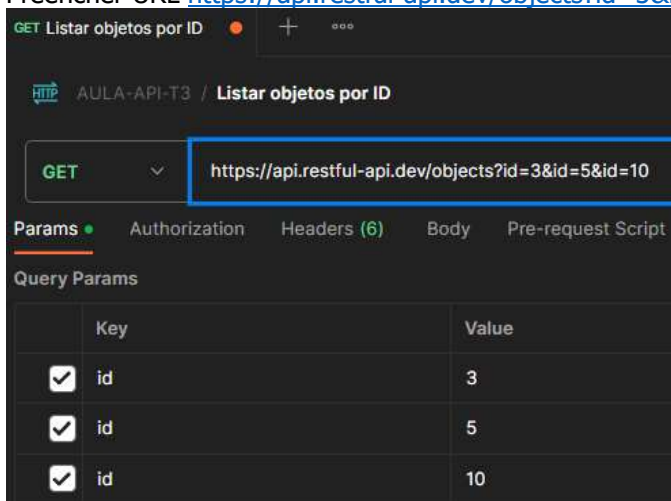
GET: LISTAR OBJETOS POR ID/LIST OF OBJECTS BY IDS

TESTE MANUAL

<https://restful-api.dev/>



Preencher URL <https://api.restful-api.dev/objects?id=3&id=5&id=10>



Clicar em **Send**

```
[
  {
    "id": "3",
    "name": "Apple iPhone 12 Pro Max",
    "data": {
      "color": "Cloudy White",
      "capacity GB": 512
    }
  },
  {
    "id": "5",
    "name": "Samsung Galaxy Z Fold2",
    "data": {
      "price": 689.99,
      "color": "Brown"
    }
  },
  {
    "id": "10",
    "name": "Apple iPad Mini 5th Gen",
    "data": {
      "Capacity": "64 GB",
      "Screen size": 7.9
    }
  }
]
```

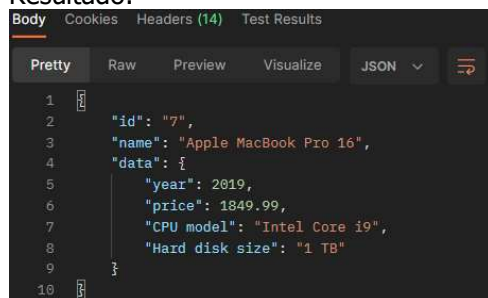
GET: ÚNICO OBJETO/SINGLE OBJECT

TESTE MANUAL

<https://restful-api.dev/>

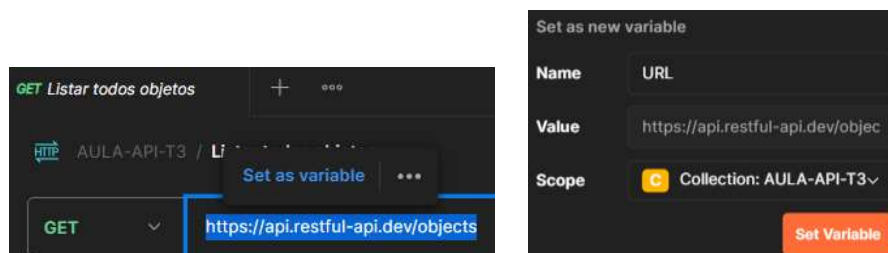
<https://api.restful-api.dev/objects/7>

Resultado:



```
1  {
2    "id": "7",
3    "name": "Apple MacBook Pro 16",
4    "data": {
5      "year": 2019,
6      "price": 1849.99,
7      "CPU model": "Intel Core i9",
8      "Hard disk size": "1 TB"
9    }
10 }
```

Criar variável para URL que se repete em vários Request



Alterar a URL para a variável {{URL}} criada nos *Request* existentesDe <https://api.restful-api.dev/objects?id=3&id=5&id=10>

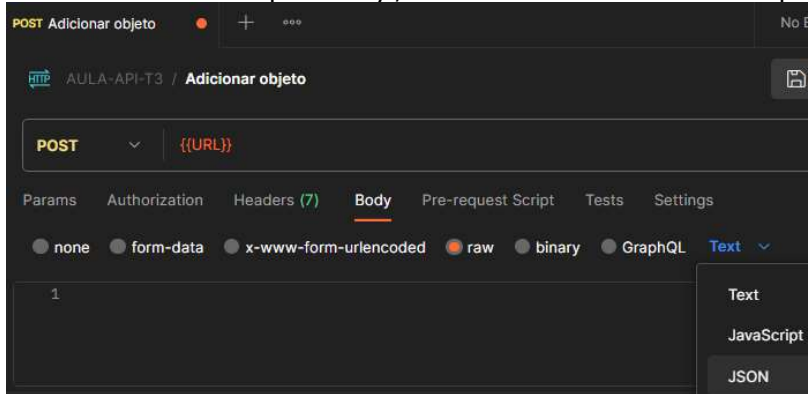
Para {{URL}}?id=3&id=5&id=10

De <https://api.restful-api.dev/objects/7>

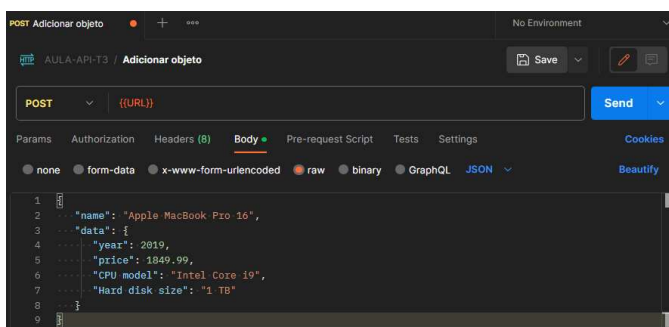
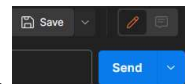
Para {{URL}}/7

**POST: ADICIONAR OBJETO/ADD OBJECT****TESTE MANUAL**

Selecionar 'POST' e depois 'Body'; Marcar 'raw' e alterar de 'Text' para 'JSON'.

No site <https://restful-api.dev/>:

Selecionar a opção 'Add object' e depois copiar o objeto. Depois: alterar, salvar e enviar.



PUT: ATUALIZAR OBJETO / UPDATE OBJECT

TESTE MANUAL

The image shows a Postman PUT request and its response. The request is for the URL `{{URL}}/ff8081818c01d7ae018c51e32d815468` with a JSON body. The response shows the updated object with an `updatedAt` timestamp.

```
PUT {{URL}}/ff8081818c01d7ae018c51e32d815468
```

```
{  "name": "BMW M3",  "data": {    "year": 2024,    "price": 49999.99,    "CPU model": "2.0 Turbo",    "Hard disk size": "50 lt",    "color": "giz"  }}
```

```
{  "id": "ff8081818c01d7ae018c51e32d815468",  "name": "BMW M3",  "updatedAt": "2023-12-10T04:06:43.468+00:00",  "data": {    "year": 2024,    "price": 49999.99,    "CPU model": "2.0 Turbo",    "Hard disk size": "50 lt",    "color": "giz"  }}
```

VERIFICAR SE O OBJETO ACIMA FOI ALTERADO

GET: ÚNICO OBJETO

Copiar o ID na URL.

The image shows a Postman GET request and its response. The request is for the URL `{{URL}}/ff8081818c01d7ae018c51e32d815468`. The response shows the object retrieved from the database.

```
GET {{URL}}/ff8081818c01d7ae018c51e32d815468
```

```
{  "id": "ff8081818c01d7ae018c51e32d815468",  "name": "BMW M3",  "data": {    "year": 2024,    "price": 49999.99,    "CPU model": "2.0 Turbo",    "Hard disk size": "50 lt",    "color": "giz"  }}
```

PATCH: ATUALIZAR PARCIALMENTE / PARTIALLY UPDATE OBJECT

TESTE MANUAL

Verificar alteração parcial (nome)

The image shows a Postman PATCH request and its response. The request is for the URL `{{URL}}/ff8081818c01d7ae018c51e32d815468` with a JSON body containing only the `name` field. The response shows the object with the `name` field updated to "Porsche 911".

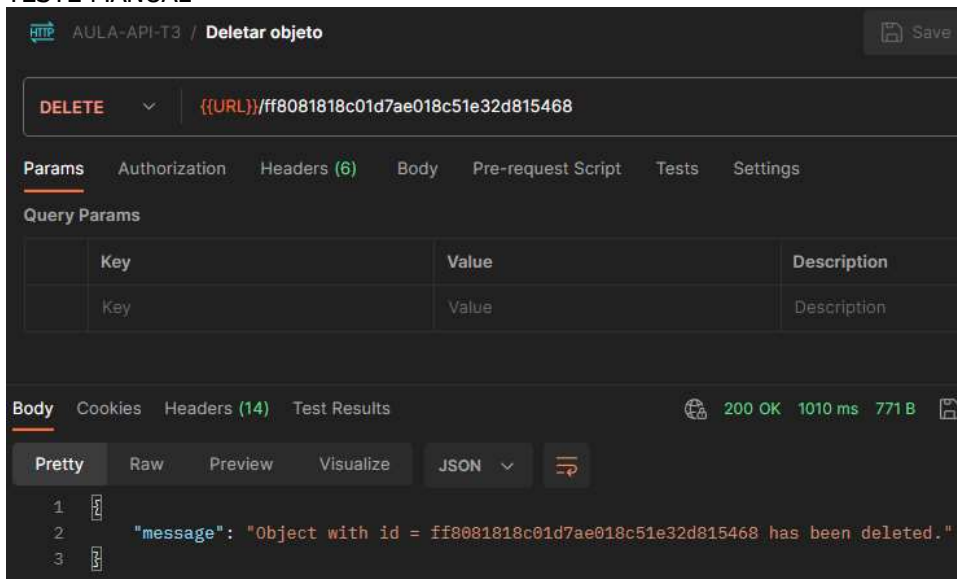
```
PATCH {{URL}}/ff8081818c01d7ae018c51e32d815468
```

```
{  "name": "Porsche 911"}
```

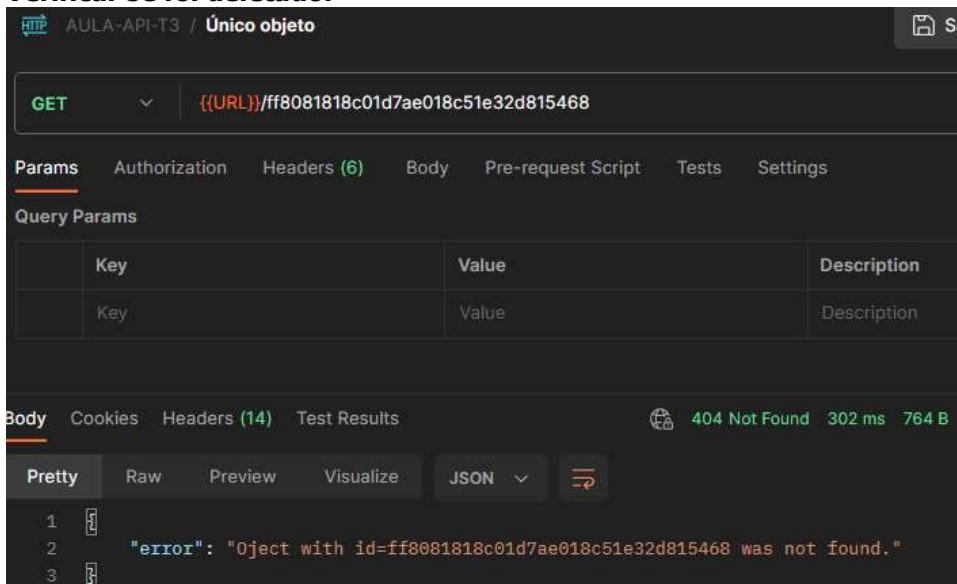
```
{  "id": "ff8081818c01d7ae018c51e32d815468",  "name": "Porsche 911",  "updatedAt": "2023-12-10T04:16:35.843+00:00",  "data": {    "year": 2024,    "price": 49999.99,    "CPU model": "2.0 Turbo",    "Hard disk size": "50 lt",    "color": "giz"  }}
```

DELETE: DELETAR OBJETO

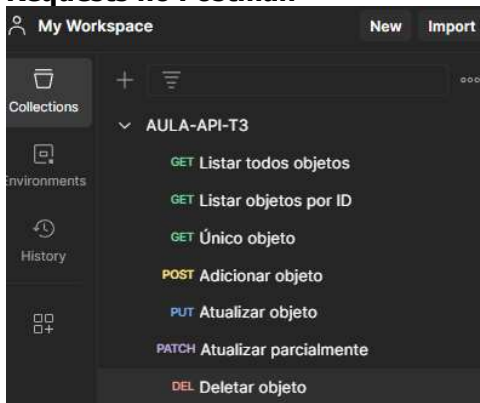
TESTE MANUAL



Verificar se foi deletado:



Requests no Postman



Tipos de Testes

Teste de Integração: entre APIs.

Teste de Regressão: nova API.

Smoke Test: testa só a API que entrou.

