

# PRIMEIRO SERVIDOR

## API - GET - POST

INICIAR



# GAIA MARIA



/in/gaia-maria/



@olha.a.navalha



gaiamaria.melos@gmail.com

- Eu sou uma travesti pansexual;
- Apaixona por tecnologia;
- Graduada em historia (UNESP);
- Graduanda ADS (Descomplica);
- Bruxa Hekatina, magista do caos e pagã;
- Neurodiversa, gamer, viciada em RPG e desenvolvedora.

# PARA HOJE

- 1 Protocolo HTTP e Verbos
- 2 CRUD
- 3 API
- 4 GET & POST
- 5 Criando o Server
- 6 Atividade pra casa

MOMENTO PARA RELAXAR



PORQUE HOJE SÓ TEM SURTO...

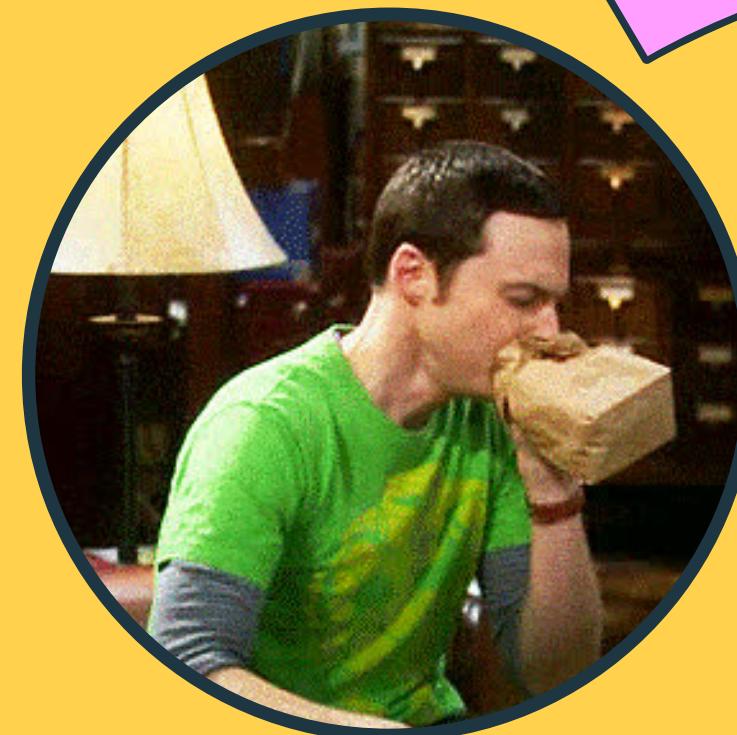
# ANTES DE COMEÇAR



Respire



Expire



Não pira!!!!!!

# PROTÓCOLOS HTTP E VERBOS

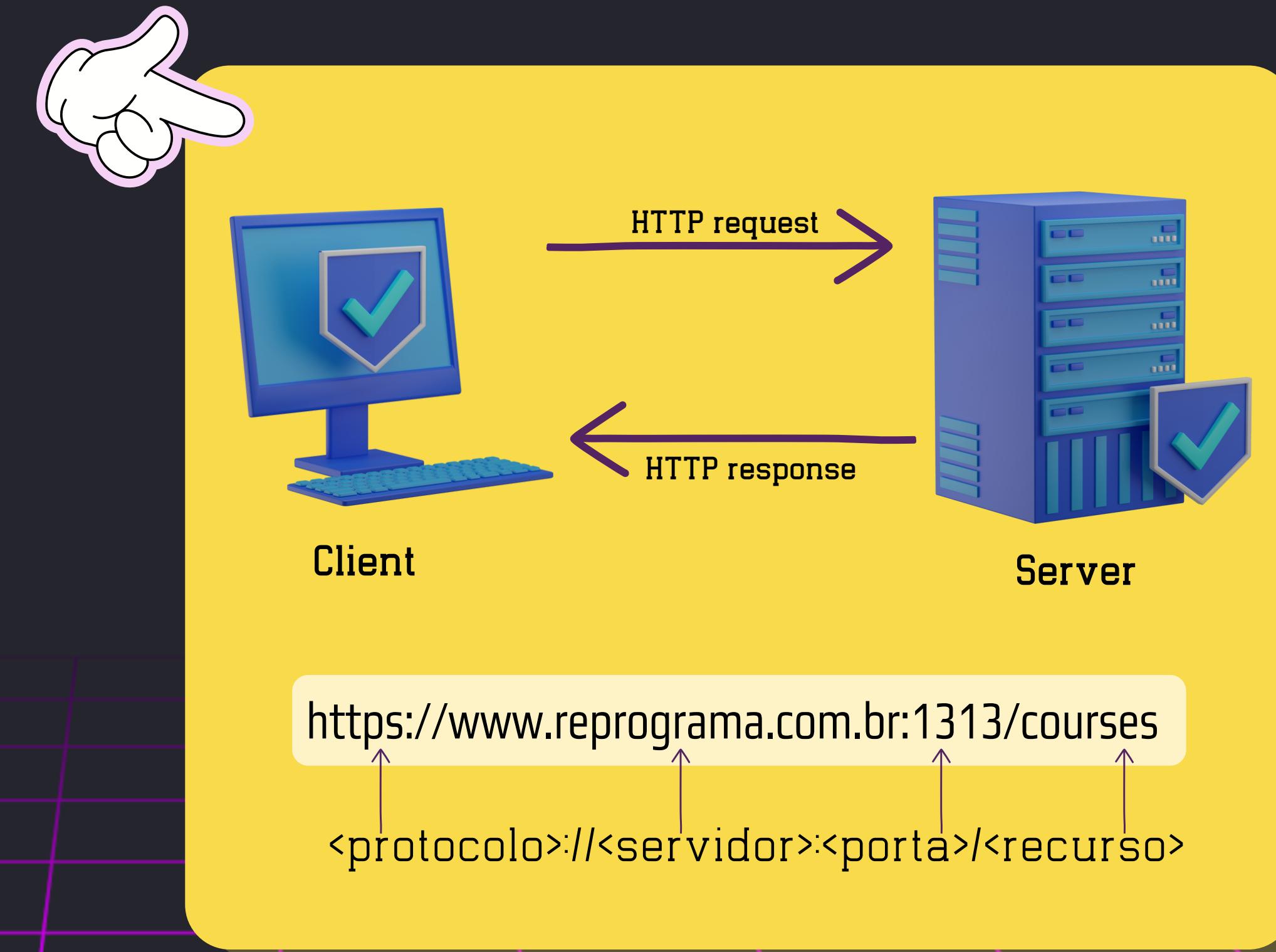


PARA RELEMBRAR...

## PROTOCOLOS HTTP

O Protocolo de Transferência de Hipertexto é um serviço utilizado dentro do modelo Client/Server e é baseado em requisições (requests) e respostas (responses).

Qualquer servidor que você escolha para hospedar um site tem um programa projetado para receber solicitações HTTP. Portanto, o navegador que você usa é um cliente HTTP que envia solicitações constantemente ao seu servidor.

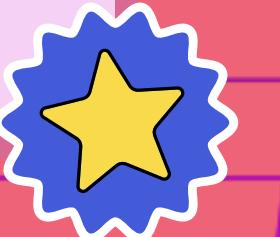


## VERBOS HTTP

São um conjunto de métodos de requisição responsáveis por indicar uma ação a ser executada.

O Client envia um request solicitando um dos verbos e o Server responde com um response.

A URL <https://www.reprograma.com.br/#courses> poderia ser usada para diferentes finalidades, dependendo do verbo enviado na requisição., No caso do GET, essa URL deveria nos retornar os cursos cadastrados no site. Já o verbo DELETE indicaria que desejamos remover esse registro.



# HTTP - STATUS CODE

Os códigos de status das respostas HTTP indicam se uma requisição HTTP foi concluída. É a pessoa desenvolvedora que adiciona na construção do servidor quais serão os casos referentes a cada resposta.

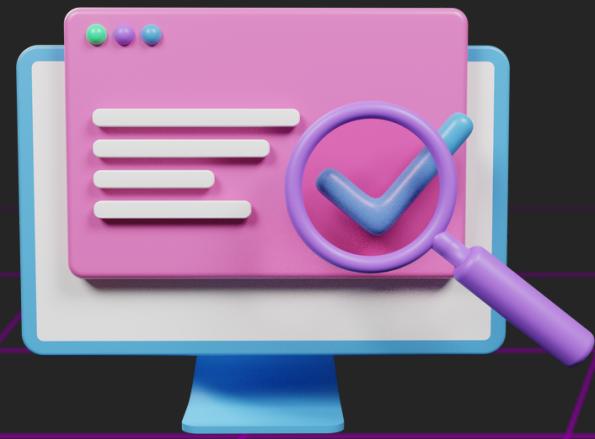
As respostas são divididas em cinco grupos:

RESPOSTAS DE  
INFORMAÇÕES



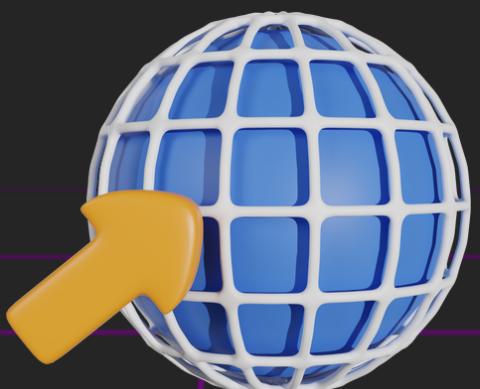
(100 - 199)

RESPOSTAS DE  
SUCESSO



(200 - 299)

RESPOSTAS DE  
REDIRECIONAMENTO



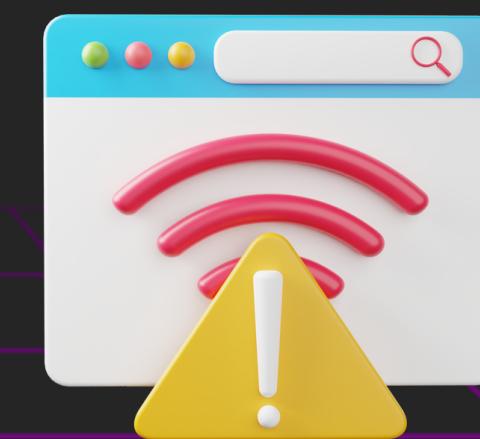
(300 - 399)

ERROS DO  
CLIENTE



(400 - 499)

ERROS DO  
SERVIDOR

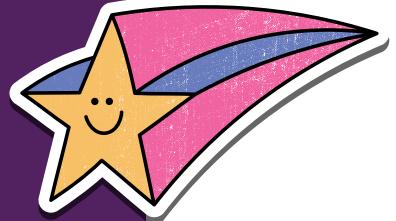


(500 - 599)

# CRUD



APRENENDENDO ALGO NOVO



CRUD é a composição da primeira letra de quatro operações básicas de um banco de dados,

Elas são o que a maioria das aplicações fazem.

**C: Create (criar)** – criar um novo registro

**R: Read (ler)** – exibir as informações de um registro

**U: Update (atualizar)** – atualizar os dados do registro

**D: Delete (apagar)** – apagar um registro

CRUD	HTTP	REST
Create	POST	/api/movie
Read	GET	/api/movie/{id}
Update	PUT	/api/movie
Delete	DELETE	/api/movie/{id}

API, WEB API, API REST & RESTFUL



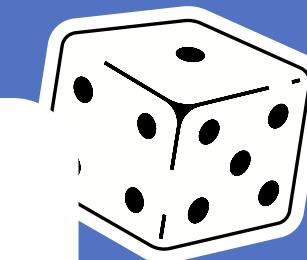
HELLO

QUE OS JOGOS COMECEM

## Resumo:

Uma API busca construir formas/ferramentas para se utilizar uma certa funcionalidade ou uma informação sem ter que "reinventar" algo novo.

Ela não precisa estar somente em um link na Internet, Mas também, pode ser uma lib ou um framework, uma função já pronta em uma linguagem.



## Exemplos:

- **YouTube** tem uma API que lista seus vídeos,
- **Uber** tem uma API para encontrar um motorista.
- **Mercado Livre** possui uma API para fazer compras e rastrear pedidos.
- **Steam** possui uma API para cadastrar e disponibilizar novos jogos



## REST - RESTful:

Uma API REST ou RESTful, é uma interface de programação de aplicações (API ou API web). Permitem a criação de um projeto com interfaces bem definidas. permitindo, que aplicações se comuniquem.

REST é a sigla em inglês para "Representational State Transfer", que em português significa transferência de estado representacional.



# ORGANIZANDO A API

## RECURSOS

Na API temos uma coleção, por exemplo, em uma API como a da Steam teríamos uma coleção de jogos. Logo "Jogos" é um recurso nessa API.

### Jogos

- Nome
- Desenvolvedor:
- Ano de lançamento:
- Gênero:
- Console jogavel:

## IDENTIFICADORES

Os recursos disponíveis em uma coleção são identificados pelo ID. No caso de jogos pode ser o numero de serie. Com pessoas é o CPF e com livros é o ISBN.



## JSON OU SCHEMA

O Json ou a Schema é a forma de apresentar os dados que estão em transito. Com o Schema as informações que estão indo ate o banco de dados. Aceita: boolean, numero, string...

```
1  {
2      "nome": "João da Silva",
3      "idade": 20,
4      "matricula": "2018123490",
5      "curso": "Sistemas de Informação",
6      "cadeiras": [
7          "Estrutura de Dados",
8          "Organização de Computadores",
9          "Matemática Discreta"
10     ]
11 }
```

# EM TODA COLEÇÃO NÓS PODEMOS:

- Listar os recursos contidos nesta coleção
- Ver um recurso dentro da coleção
- Adicionar um novo recurso na coleção
- Sobrescrever ou criar um recursos
- Editar um recurso dentro da coleção
- Excluir um recurso



# EM UM TWEET



**Namorada do Kratos**   
@MissRivotril

API são instruções sobre como se comunicar com um serviço. Ou seja, as APIs são um tipo de “ponte” que conectam aplicações, podendo ser utilizadas para os + variados tipos de negócio, por empresas de diversos nichos de mercado ou tamanho. Entendeu? Não? Nem eu kkkk

12:00 PM · Jun 1, 2021

**666** Retweets

**24** Quote Tweets

**69** Likes

# DIDATIZANDO EM 3 PASOS:



1- Você vai com suas amigas a um restaurante e precisa realizar um pedido

2- A cozinha tem tudo para preparar seu pedido, mas não pode levar até você. É necessário que alguém leve

3- Essa pessoa é o garçom. Ele sabe exatamente quando e como levar o seu pedido

Uma API funciona exatamente assim, ela é responsável por trazer a resposta necessária para o que está sendo pedido.

REQUISIÇÕES GET & POST



GET O TRANCO OU POST MUITO NO TWITTER

# GET

Utilizemos GET para ler ou informar um dado. Um GET funcionando corretamente irá retornar uma resposta contendo as informações solicitadas.

Se utilizássemos para saber todos os jogos da Sony na Steam, por exemplo, ficaria assim:

```
GET  
/games/desenvolvedor?desenvolvedor=Sony
```

# POST

Usamos o POST para adicionar um novo recurso a nossa coleção. Para solicitarmos essa adição precisamos adicionar no parâmetro Body todas as informações.

Se utilizássemos para criar um novo jogo Steam, usariámos dessa forma dentro do Body:



POST /jogos

{

```
"nome": "Shadows of the Colossus",  
"desenvolvedor": "Sony",  
"lançamento": 2005,  
"genero": "[aventura, RPG]",  
"console": "[PlayStation, PC]"
```

}

# BODY & BODY PARSE

**Body** - São usados nos métodos POST, PATCH E PUT. Eles enviam dados a serem cadastrados no banco de dados.

- request.body

```
{  
  "descrição": "Exemplo"  
  "nome": "Gaiazinha"  
}
```

**Body Parse** - Quando recebemos um request os dados do body são enviados de uma forma que não conseguimos facilmente acessar e manipular.

Por isso, devemos "parsear" o body: essa função analisa e transforma num JSON manipulável.

# TIPOS DE PARAMETROS

Parâmetros são enviados na requisição e podem ser utilizados pelo serviços, com o objetivo de definir a requisição e as ações.

Eles são: Query, params e body

## **request.query**

É passado no formato key=value. Esses parâmetros são definidos por quem desenvolveu a API. Utiliza-se quando queremos criar filtros para fazer consultas na nossa aplicação, o ideal é sempre usar o req.query.

EX.: GET /jogos/findByDesenvolvedor?desenvolvedor=Sony

## **request.params**

Eles são usados para apresentar um recurso específico dentro de uma coleção. Um URL pode ter vários parâmetros, cada um denotado com chaves {} OU dois pontos .

EX.: GET /games/:id

## **request.body**

É utilizado para enviar dados que serão cadastrados no banco de dados, podem ser combinados com query ou path params.

EX.: { "disponivel": true}

CRIANDO O SERVER



BORA ABALAR?

# npm init

Esse comando nos permite iniciar um pacote, criando o arquivo package.json de acordo com certas respostas que damos às perguntas feitas.

```
mayhhara.morais@MacBook-Pro-de-Mayhhara:~/Desktop/On16-TodasEmTech-S8-API-G... ✘%1
→ On16-TodasEmTech-S8-API-Get-Post git:(main) npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (on16-todasemtech-s8-api-get-post)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository: (https://github.com/reprograma/On16-TodasEmTech-S8-API-Get-Post.git)
keywords:
author:
license: (ISC)
About to write to /Users/mayhhara.morais/Desktop/On16-TodasEmTech-S8-API-Get-Post/package.json

{
  "name": "on16-todasemtech-s8-api-get-post",
  "version": "1.0.0",
  "description": "1. Crie a pasta em que ficará seu servidor",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/reprograma/On16-TodasEmTech-S8-API-Get-Post.git"
  },
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/reprograma/On16-TodasEmTech-S8-API-Get-Post/issues"
  },
  "homepage": "https://github.com/reprograma/On16-TodasEmTech-S8-API-Get-Post"
}

Is this OK? (yes)
```

The screenshot shows a terminal window and a code editor side-by-side. The terminal window displays the command 'npm init' being run, asking for package details like name, version, and description. It also shows the generated package.json file content. The code editor shows the same package.json file with syntax highlighting, indicating the file has been created or modified.

```
package.json U ×
package.json > ...
1  {
2   "name": "on16-todasemtech-s8-api-get-post",
3   "version": "1.0.0",
4   "description": "1. Crie a pasta em que ficará seu servidor",
5   "main": "index.js",
6   ▷ Debug
7   "scripts": {
8     "test": "echo \\\"Error: no test specified\\\" && exit 1"
9   },
10  "repository": {
11    "type": "git",
12    "url": "git+https://github.com/reprograma/On16-TodasEmTech-S8-API-Get-Post.git"
13  },
14  "author": "",
15  "license": "ISC",
16  "bugs": {
17    "url": "https://github.com/reprograma/On16-TodasEmTech-S8-API-Get-Post/issues"
18  },
19  "homepage": "https://github.com/reprograma/On16-TodasEmTech-S8-API-Get-Post"
```

dependencias

# EXPRESS

O Express.js é um Framework rápido e um dos mais utilizados em conjunto com o Node.js, facilitando no desenvolvimento de aplicações back-end e até, em conjunto com sistemas de templates, aplicações full-stack.

**express**  
4.17.1 • Public • Published a year ago

Readme Explore BETA 30 Dependencies 46.033 Dependents 264 Versions

# express

Fast, unopinionated, minimalist web framework for [node](#).

npm v4.17.1 downloads 58M/month linux passing windows passing coverage 100%

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

Install  
> npm i express

Weekly Downloads 13.961.907

Version 4.17.1 License MIT

Unpacked Size 208 kB Total Files 16

Issues 97 Pull Requests 52

Homepage

Possui um sistema de rotas completo;  
Possibilita o tratamento de exceções dentro da aplicação;  
Permite a integração de vários sistemas de templates que facilitam a criação de páginas web para suas aplicações;  
Gerencia diferentes requisições HTTP com seus mais diversos verbos;  
Feito para a criação rápida de aplicações utilizando um conjunto pequeno de arquivos e pastas;

dependencias

# NODEMON

O nodemon é uma biblioteca que ajuda no desenvolvimento de sistemas com o Node.js reiniciando automaticamente o servidor. Ele fica monitorando a aplicação em Node, e assim que houver qualquer mudança no código, o servidor é reiniciado automaticamente.

**nodemon**

2.0.4 • Public • Published 4 months ago

[Readme](#) [Explore BETA](#) [10 Dependencies](#) [2.477 Dependents](#) [215 Versions](#)



**nodemon**

nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

Install

```
> npm i nodemon
```

Fund this package

Weekly Downloads  
2.893.116

Version  
2.0.4

Unpacked Size  
107 kB

License  
MIT

Total Files  
43

# ARQUITETURA MVC

MVC é um padrão de arquitetura de software, separando sua aplicação em 3 camadas. A camada de interação do usuário(view), a camada de manipulação dos dados(model) e a camada de controle(controller)

Já que estamos lidando com um projeto que tem somente backend, não lidaremos com as views, porém lidamos com as rotas(routes).

O MVC nada mais é que uma forma de organizar o nosso código

```
const express = require('express')
const app = express()

app.listen(3000, () => {
  console.log('Servidor rodando na porta 3000')
})

app.get("/", (request, response) => {
  response.status(200).json([
    "nome": "oiiiiiiii"
  ])
})
```

# POSTMAN

O Postman é uma ferramenta que dá suporte à documentação das requisições feitas pela API. Ele possui ambiente para a documentação, execução de testes de APIs e requisições em geral.

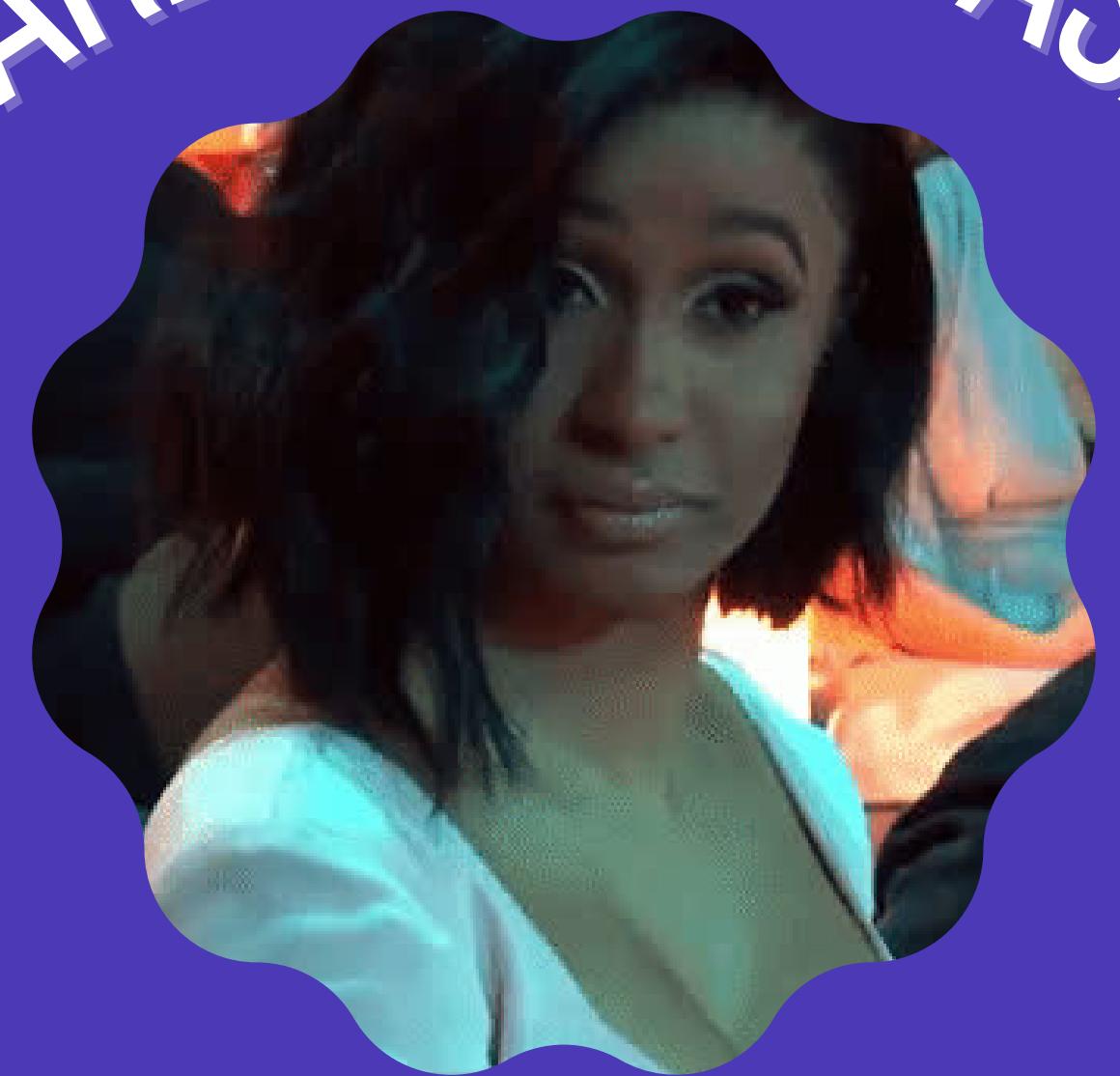
The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The 'My Workspace' section is selected. In the main area, a collection named 'reprograma' is expanded, showing two requests: 'GET GET' and 'GET POST'. The 'GET GET' request is currently selected, with its URL set to 'localhost:8080/games'. Below the URL, there are tabs for Params, Authorization, Headers (6), Body, Pre-request Script, Tests, and Settings. The Headers tab is selected, showing a table with one row: 'Key' and 'Value'. A note says 'Query Params' and 'Description'. At the bottom of the request panel, there's a button labeled 'Send' with a dropdown arrow. To the right of the request panel, there's a 'Response' section with a small illustration of an astronaut launching a rocket and the text 'Click Send to get a response'. At the very bottom of the interface, there are buttons for 'Find and Replace', 'Console', and various system icons.

LEVAMOS DE VS CODE



BORA CODAR?

TAREFA PARA CASA



CUIDADO PARA NÃO ACUMULAR

\_ X

- Quero uma rota que venha todos os filmes Ghibli;
- Uma rota /filmes;
- /filmes deve retornar todos os filmes;
- Devo conseguir filtrar por título, id e diretor;
- Devo conseguir cadastrar novos filmes.



## ROTAS FILMES

- [GET] /filmes
  - retorna todos os filmes
- [GET] /filmes/{id}
  - retorna um filme pelo id
- [GET] /filmes?{diretor}
  - retorna um filme pelo diretor
- [POST] /filmes/criar
  - cria novo filme