

AGENDA

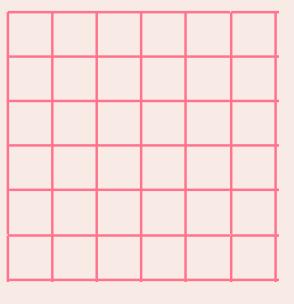
- Apresentação;
- Pra relembrar:
 - Protocolo HTTP e Verbos;
- CRUD;
- API;
- GET && POST;
- Criando o Server
- Atividade pra casa <3

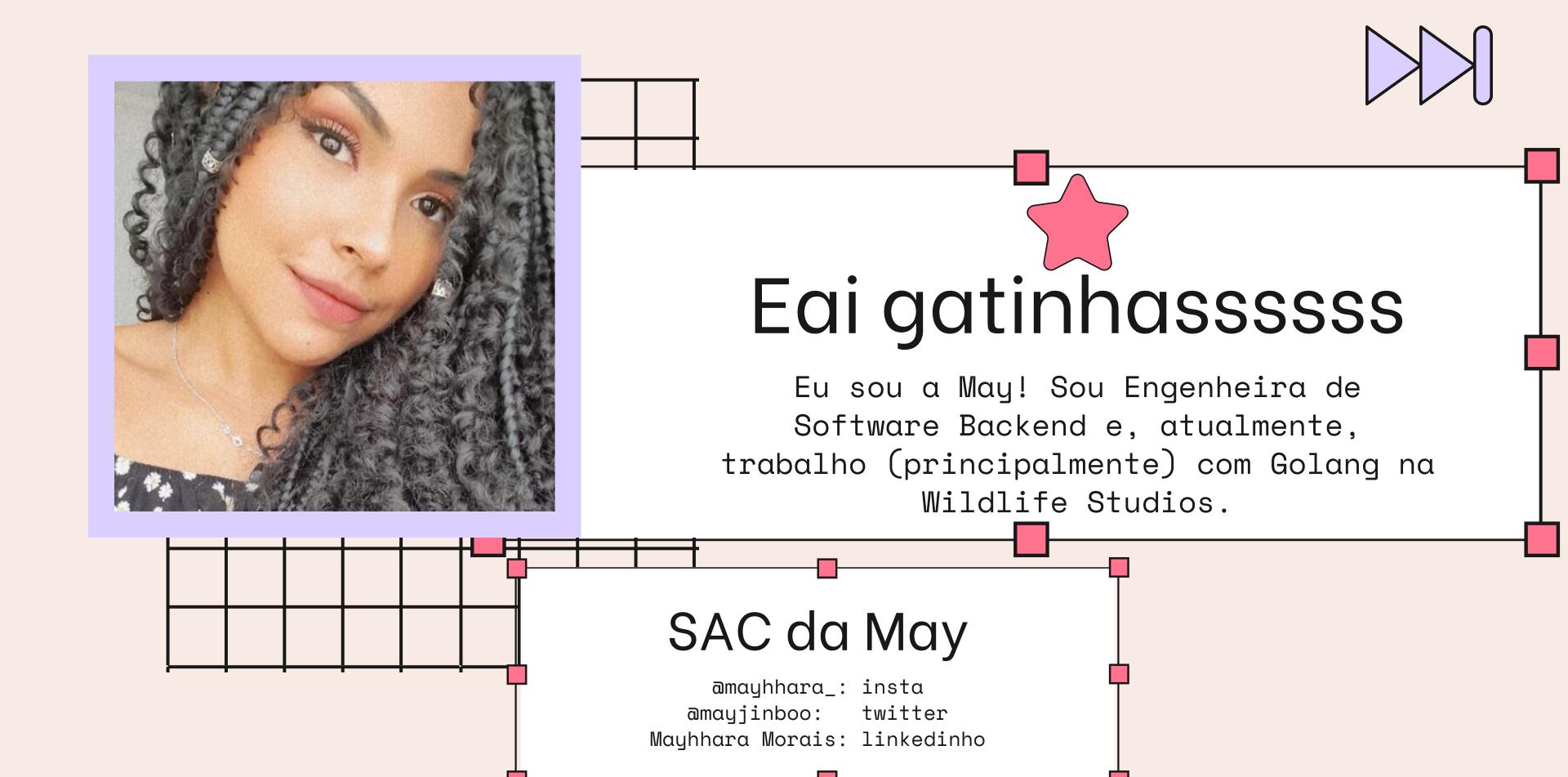


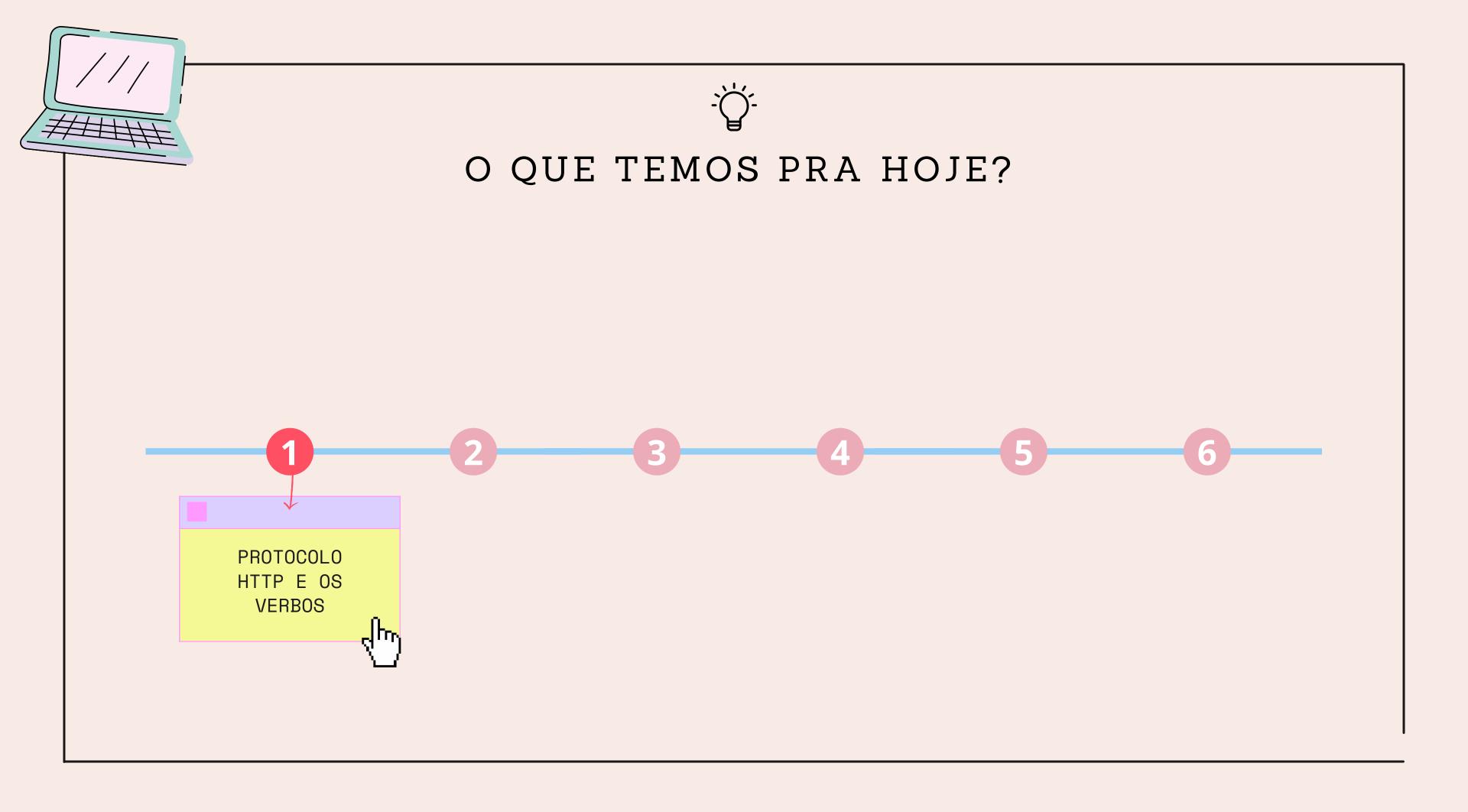












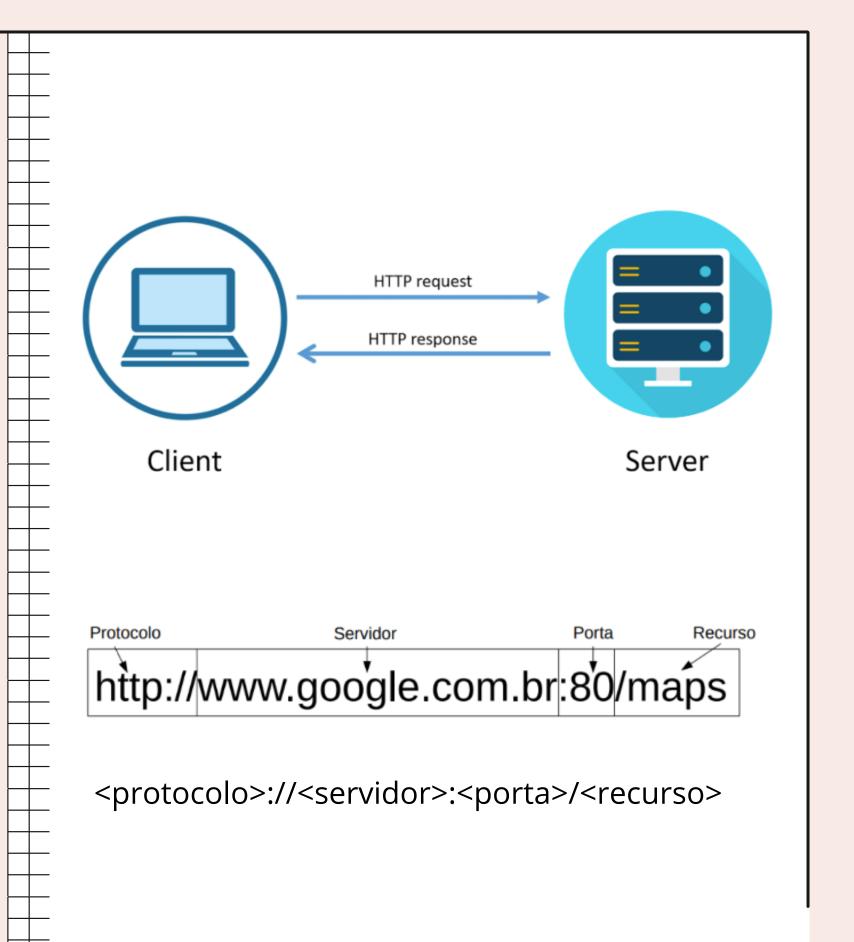


HTTP

Protocolo de Transferência de Hipertexto é um protocolo usado dentro do modelo Client/Server é baseado em pedidos (requests) e respostas (responses).

Ele é a forma em que o Cliente e o Servidor se comunicam.

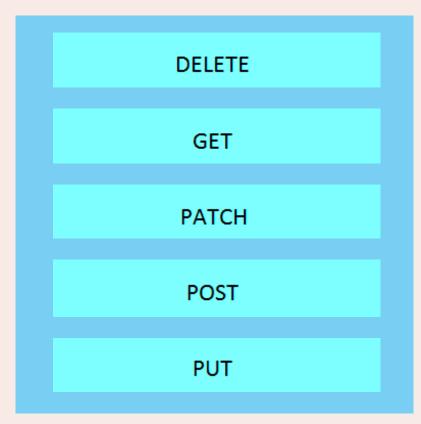
Pensando em uniformizar a comunicação entre servidores e clientes foram criados **códigos** e **verbos** que são usados por ambas as partes, e essas requisições são feitas em **URLs** que possuem uma estrutura específica.



HTTP - Verbos

Os verbos HTTP são um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada.

O Client manda um request solicitando um dos verbos e o Server deve estar preparado para receber e responde-lo com um response.



HTTP - Status Code

Quando o **Client** faz uma requisição o Server responde com um código de status numérico também padronizado.

Os códigos de status das respostas HTTP indicam se uma requisição HTTP foi concluída. As respostas são agrupadas em cinco classes:

Respostas de informação (100-199)

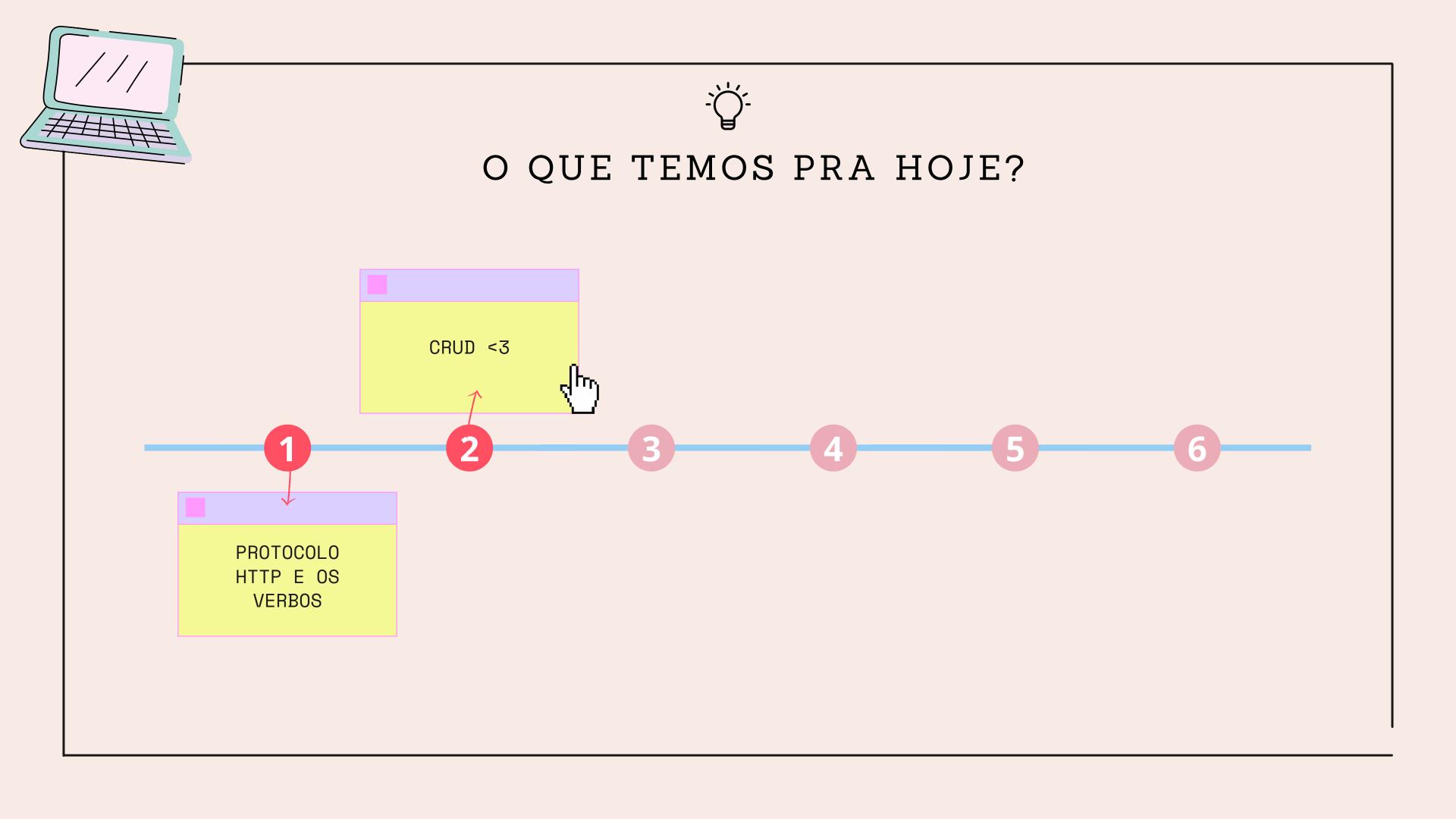
Respostas de sucesso (200-299)

Redirecionamentos (300-399)

Erros do cliente (400-499)

Erros do servidor (500-599)

É a desenvolvedora Back-end que coloca na construção do servidor quais serão as situações referentes a cada resposta.





HTTP - CRUD

CRUD é a composição da primeira letra de quatro operações básicas de um banco de dados, e são o que a maioria das aplicações fazem.

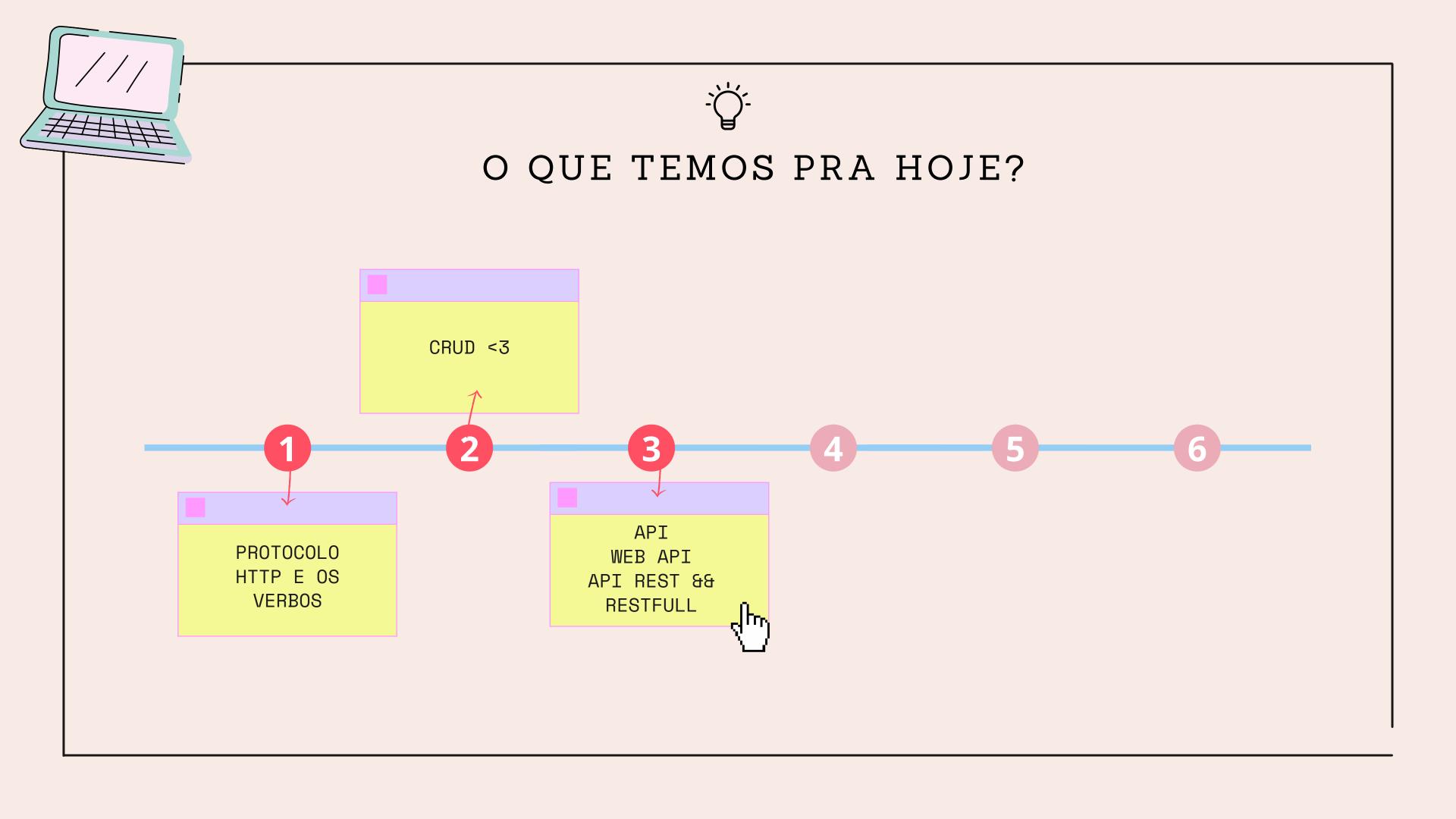
C: Create (criar) - criar um novo registro

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

U: Update (atualizar) - atualizar os dados do registro

D: Delete (apagar) - apagar um registro

Operações CRUD com HTTP DELETE Create Read GET Update **PATCH** Update POST PUT Delete



API

Interface de Programação de Aplicativos

API busca criar formas e ferramentas de se usar uma funcionalidade ou uma informação sem realmente ter que ''reinventar a tal função''.

Ela não necessariamente está num link na Web, ela pode ser uma lib ou um framework, uma função já pronta em uma linguagem específica por exemplo.

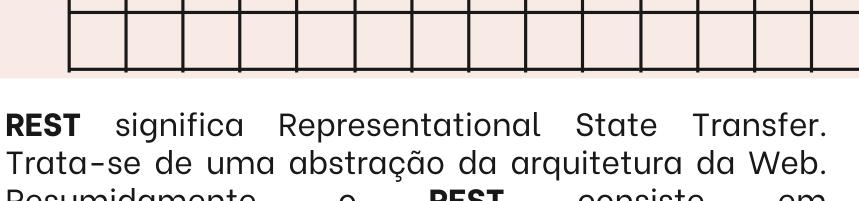
Short words: API são instruções sobre como se comunicar com um serviço



- YouTube possui uma API para listar vídeos, buscar, ver comentários…
- Instagram possui uma API para ver e enviar fotos...
- Uber possui uma API para chamar um motorista...
- Yahoo possui uma API para consultar a previsão do tempo...
- Mercado Livre possui uma API para pesquisar produtos, fazer compras e rastrear pedidos...

APIREST && && RESTFULL



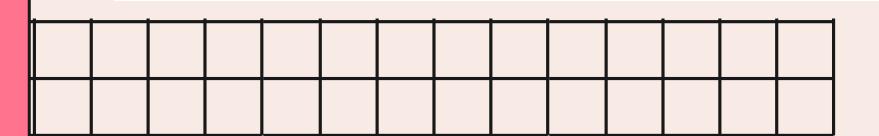


Trata-se de uma abstração da arquitetura da Web. Resumidamente, o **REST** consiste em princípios/regras/constraints que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas. Desta forma, permitindo, por exemplo, que aplicações se comuniquem.

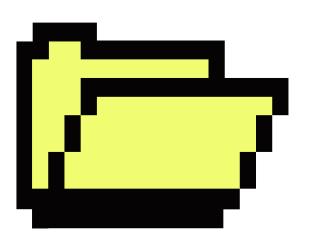
Short-words: A API Rest é uma forma padronizada de criar API's baseada no HTTP.

Existe uma certa confusão quanto aos termos **REST** e **RESTful**. Entretanto, ambos representam os mesmo princípios. A diferença é apenas gramatical. Em outras palavras, sistemas que utilizam os princípios **REST** são chamados de **RESTful**.

- **REST**: conjunto de princípios de arquitetura
- **RESTful**: capacidade de determinado sistema aplicar os princípios de REST.



Como se organiza uma API REST?



COLEÇÃO DE RECURSOS

CADA RECURSO POSSUI UM IDENTIFICADOR

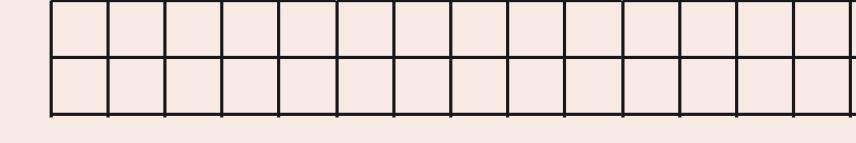
RECURSOS SÃO REPRESENTADOS COMO JSON

Coleções de de Recursos









Por exemplo: Uma API de uma biblioteca

Na API temos uma coleção de livros. "Livro" é um recurso nessa API.

Nós também temos uma coleção de autores. Dentro de um autor também temos uma coleção de livros.

LIVROS

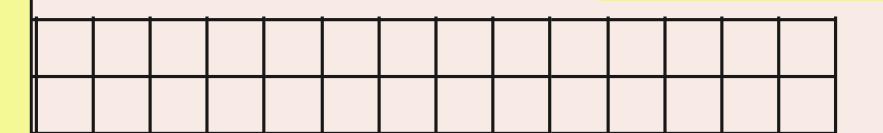
Título Editora Ano de lançamento

AUTORES

Nome País natal

LIVROS

Título Editora Ano de lançamento



Recursos possuem identificadores









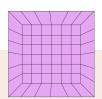
Ex: Uma pessoa pode ser identificada pelo seu CPF.

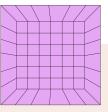
Ex: Um produto em uma loja é identificado pelo seu código de barras.

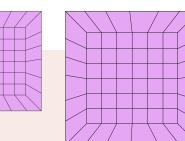
Ex: Um carro é identificado pela sua placa.

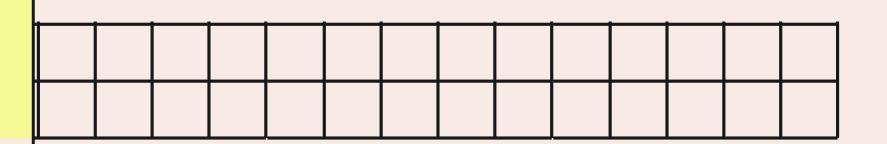
Podem ser qualquer coisa, desde que sejam únicos e imutáveis.











Recursos representados como JSON





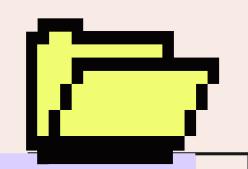


JSON é uma forma de representar dados em trânsito.

Suporta números, texto, objetos, listas, true/false e null.

```
"Title":"The Truman Show",
"Year":"1998",
"Rated":"PG",
"Released":"05 Jun 1998",
"Runtime":"103 min",
"Genre":"Comedy, Drama, Sci-Fi",
"Director":"Peter Weir",
"Writer":"Andrew Niccol",
"Actors":"Jim Carrey, Laura Linney, Noah Emmerich, Nata
"Plot":"An insurance salesman discovers his whole life
"Language":"English",
"Country":"USA",
"Awards":"Nominated for 3 Oscars. Another 40 wins & 66
```

Em toda coleção nós podemos:



Listar os **recursos** contidos nesta coleção

GET /livros

Ver um recurso dentro da coleção

GET /livros/37

Adicionar um novo recurso na coleção

POST /livros

Sobrescrever ou criar um recursos

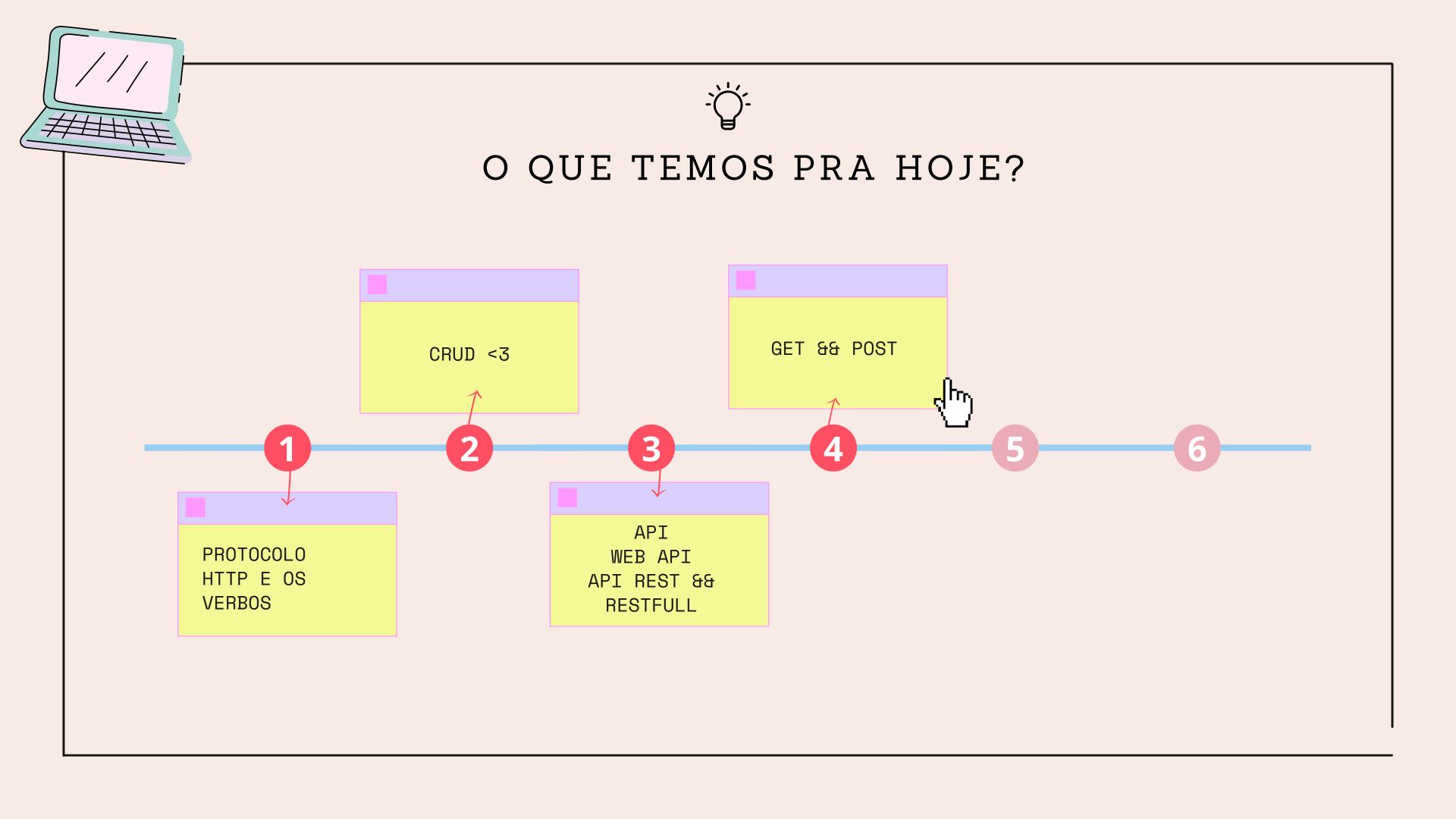
PUT /livros/37

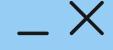
Editar um recurso dentro da coleção

PATCH /livros/37

Excluir um recurso

DELETE /livros/37





MÉTODO GET



Usamos GET para ler ou recuperar um recurso. Um GET bem-sucedido retorna uma resposta contendo as informações solicitadas.

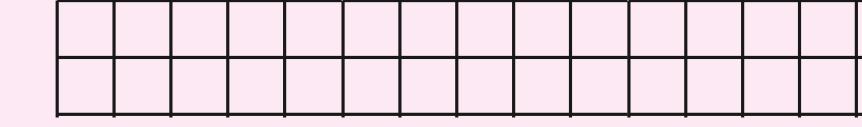
Em nossa biblioteca, poderíamos usar um GET para recuperar livros escritos por um autor específico.

GET /autor/:idautor/livros



MÉTODO POST





Usamos POST para criar um novo recurso. Uma solicitação POST requer um corpo no qual você define os dados da entidade a ser criada.

Uma solicitação POST bem-sucedida seria um código de resposta 200. Em nossa biblioteca, poderíamos usar um método POST para adicionar um livro.

POST /livros

```
{ "id": idDoNossoLivro,

"title": "tituloLivro",

"autor": "autorLivro"

"description": "descricaoLivro"

}
```



Body && &Body Parse

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

request.body

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 funçao analisa e transforma num JSON manipulável.



Params

Tanto o body quanto o query e o path são parâmetros enviados na requisição e podem ser acessados pelo servidos afim de definir a requisição e as ações.

request.params = NÃO faz parte do url e é passado no formato **key=value** esses parâmetros devem ser definidos pela desenvolvedora da API. É usado para pesquisa simples, enviado diretamente na rota

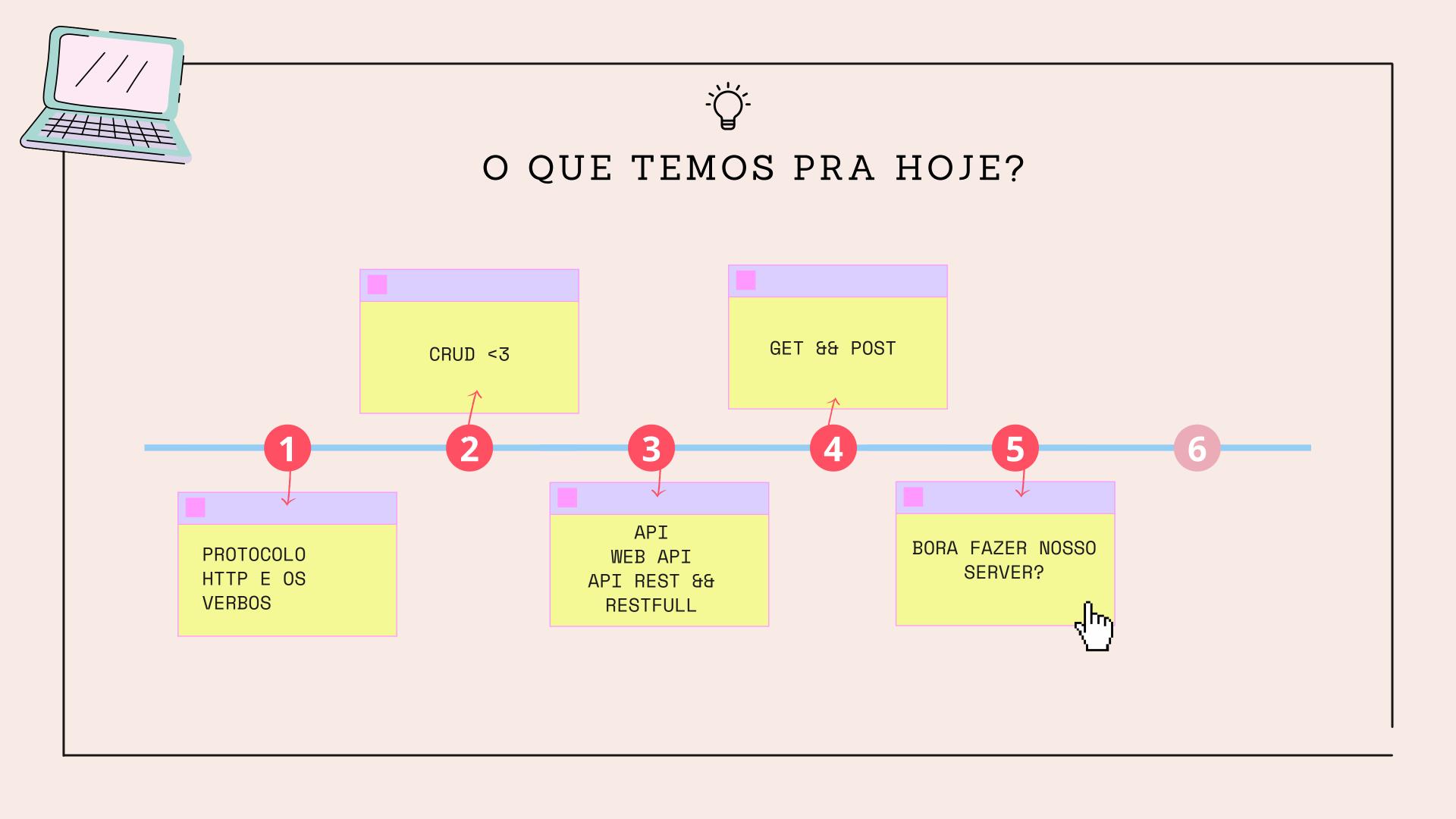
EX.: GET /livros/findByYear?year=2000

request.query= São partes variáveis de um caminho de URI. Eles são tipicamente usados para apontar para um recurso específico dentro de uma coleção, como um usuário identificado por ID. Um URL pode ter vários parâmetros de caminho, cada um

denotado com chaves { }.

EX.: GET /livros/{id}

request.body= É usado para enviar dados que serão cadastrados no banco, podem ser combinados com query ou path params.



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...
  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)
author:
license: (ISC)
About to write to /Users/mayhhara.morais/Desktop/Or

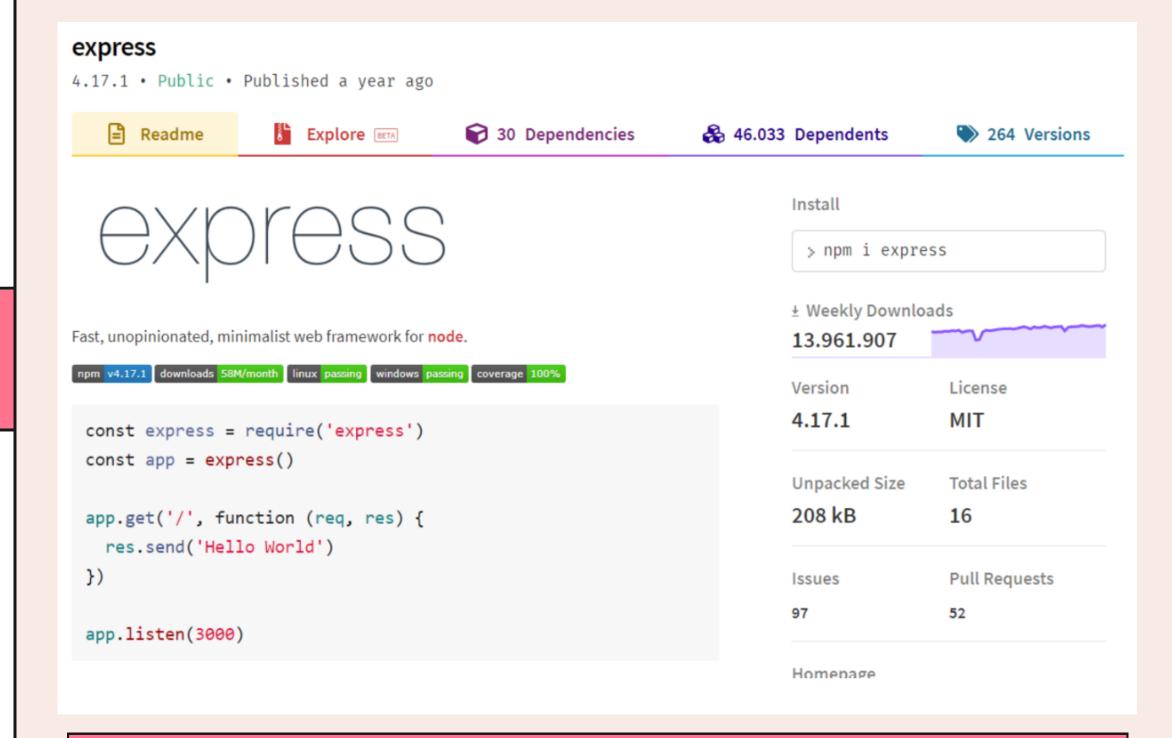
package.json U X

                                                     package.json > ...
  "name": "on16-todasemtech-s8-api-get-post",
  "version": "1.0.0",
                                                              "name": "on16-todasemtech-s8-api-get-post",
  "description": "1. Crie a pasta em que ficará seu
  "main": "index.js",
                                                              "version": "1.0.0",
   "scripts": {
                                                              "description": "1. Crie a pasta em que ficará seu servidor"
    "test": "echo \"Error: no test specified\" && e
                                                              "main": "index.js",
                                                              Debug
  "repository": {
                                                              "scripts": {
    "type": "git",
                                                                "test": "echo \"Error: no test specified\" && exit 1"
    "url": "git+https://github.com/reprograma/0n16-
                                                              "repository": {
  "author": "",
                                                                "type" "git",
  "license": "ISC",
                                                                "url": "git+https://github.com/reprograma/On16-TodasEmTech
  "bugs": {
    "url": "https://github.com/reprograma/0n16-Tode
                                                              "author": "",
  "homepage": "https://github.com/reprograma/0n16-7
                                                              "license": "ISC",
                                                              "bugs": {
                                                                "url": "https://github.com/reprograma/On16-TodasEmTech-S8-A
Is this OK? (yes)
                                                              "homepage": "https://github.com/reprograma/On16-TodasEmTech-S
```

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 fullstack.





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;

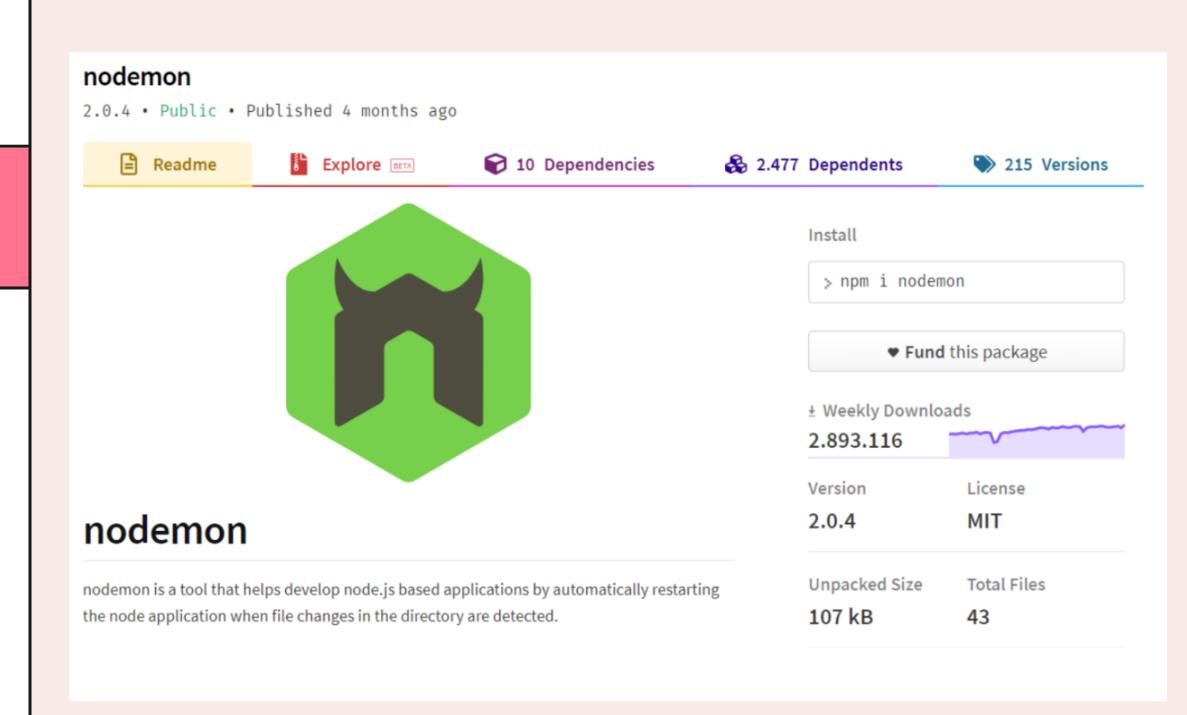
Gerencia diferentes requisições HTTP com seus mais diversos verbos;

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.



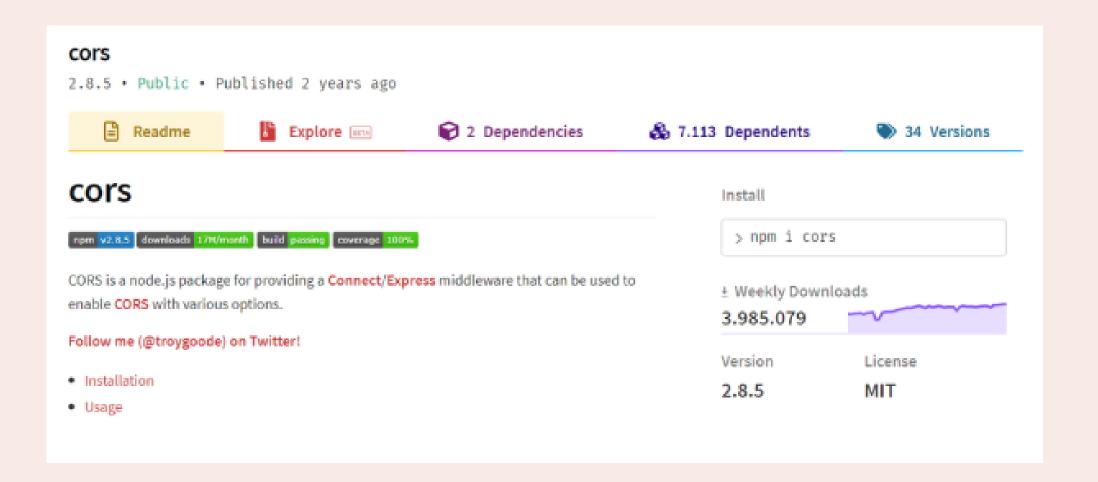
dependencias

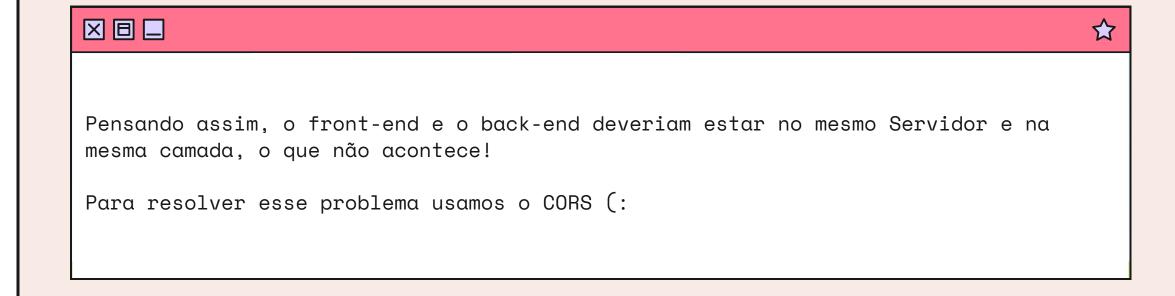
CORS

CORS (Cross-Origin Resource Sharing) é uma especificação que permite que um site acesse cursos de outro site mesmo estando em domínios diferentes.

Os navegadores fazem uso de uma funcionalidade de segurança chamada Same-Origin Policy: um recurso de um site só pode ser chamado por outro site se os dois sites estiverem sob o mesmo domínio.

Isso porque o navegador considera recursos do mesmo domínio somente aqueles que usam o mesmo protocolo (http ou https), a mesma porta e o mesmo endereço.





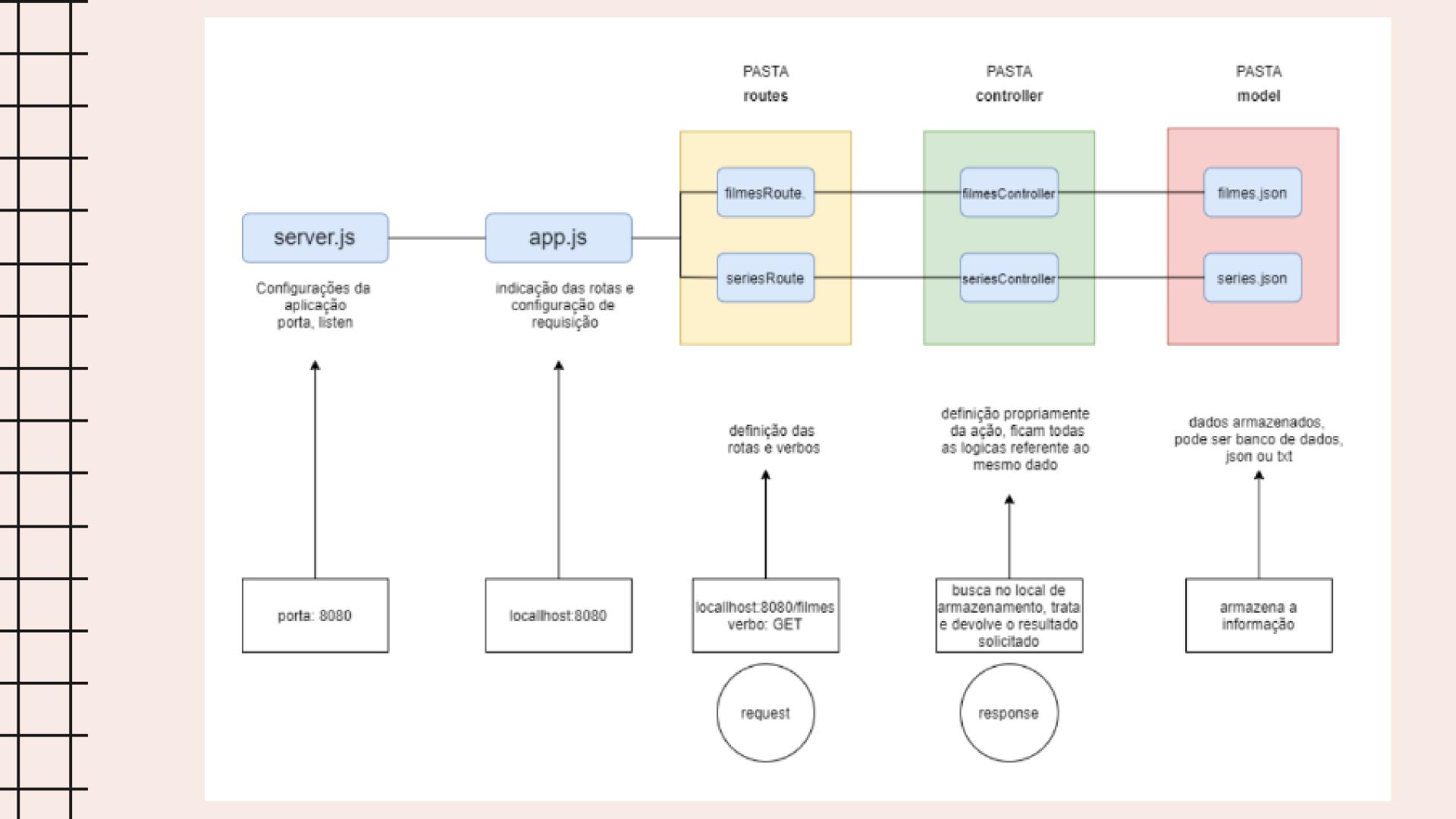
ARQUITETURA MVC

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).

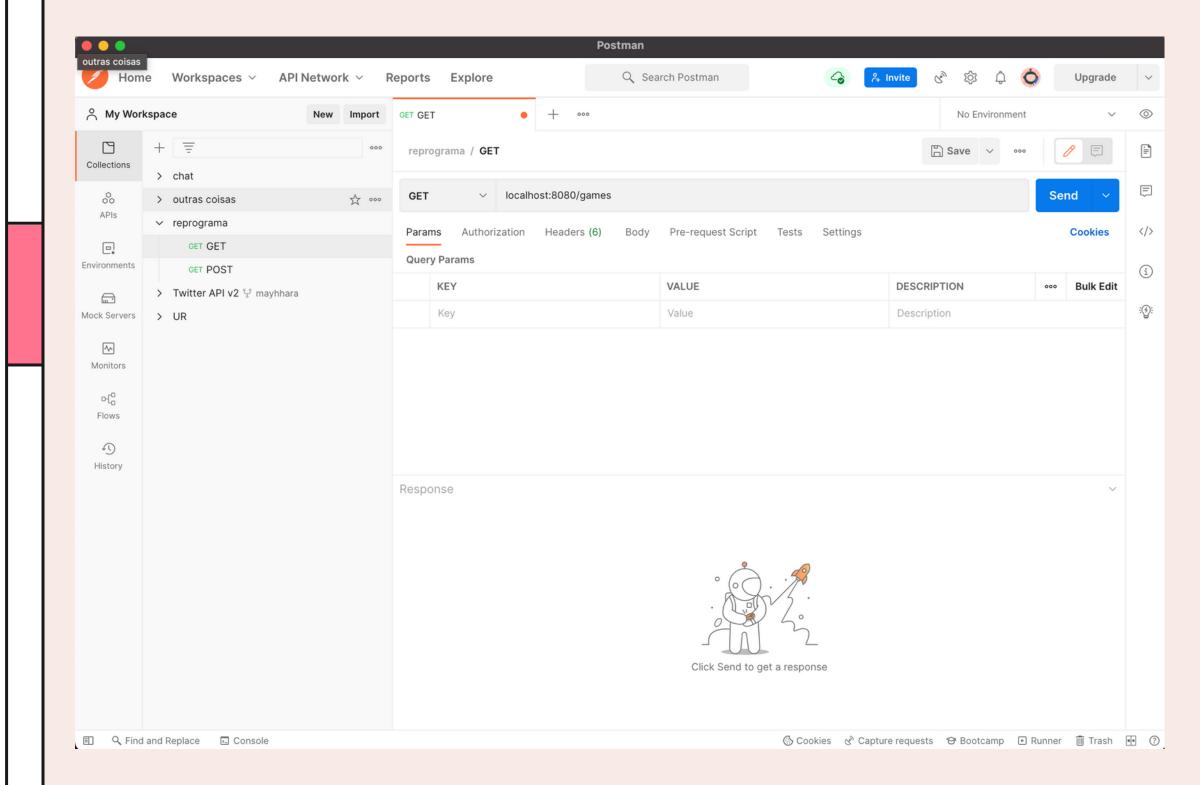
O MVC nada mais é que uma forma de **organizar** o nosso código

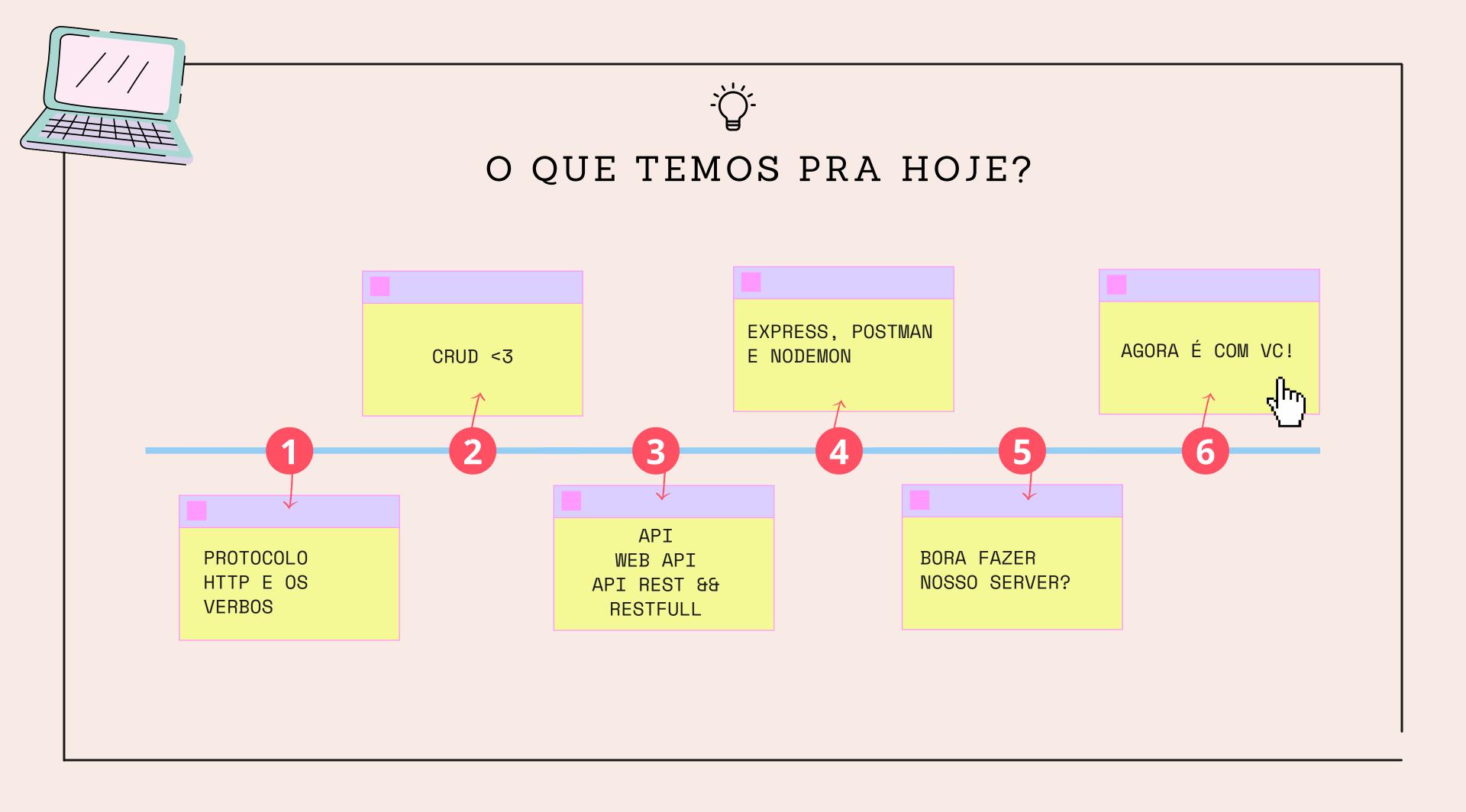
```
\-- NOME-DO-SEU-SERVIDOR
          | server.js
           \-- src
           app.js
         ---controller
     | NOMEController.js
           ---model
         NOME.json
           ---routes
       NOMERoute.js
```



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.





TAREFINHA DE CASA



- Quero uma rota que venha todos os filmes e séries
- Duas rotas a /filmes e a /series
- /filmes deve retornar todos os filmes
- /series deve retornar todas as series
- Devo conseguir filtrar por título, id e genero
- Devo conseguir cadastrar filmes e series.



H

ROTAS FILMES

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

ROTAS SÉRIES

- [GET] /series
 - retorna todas as series
- [GET] /series/{id}
 - o retorna uma serie pelo id