

{REPROGRAMA} - SEMANA 10 - BACKEND

PROJETO GUIADO 2

Profª Gaia Maria

RELEMBRANDO

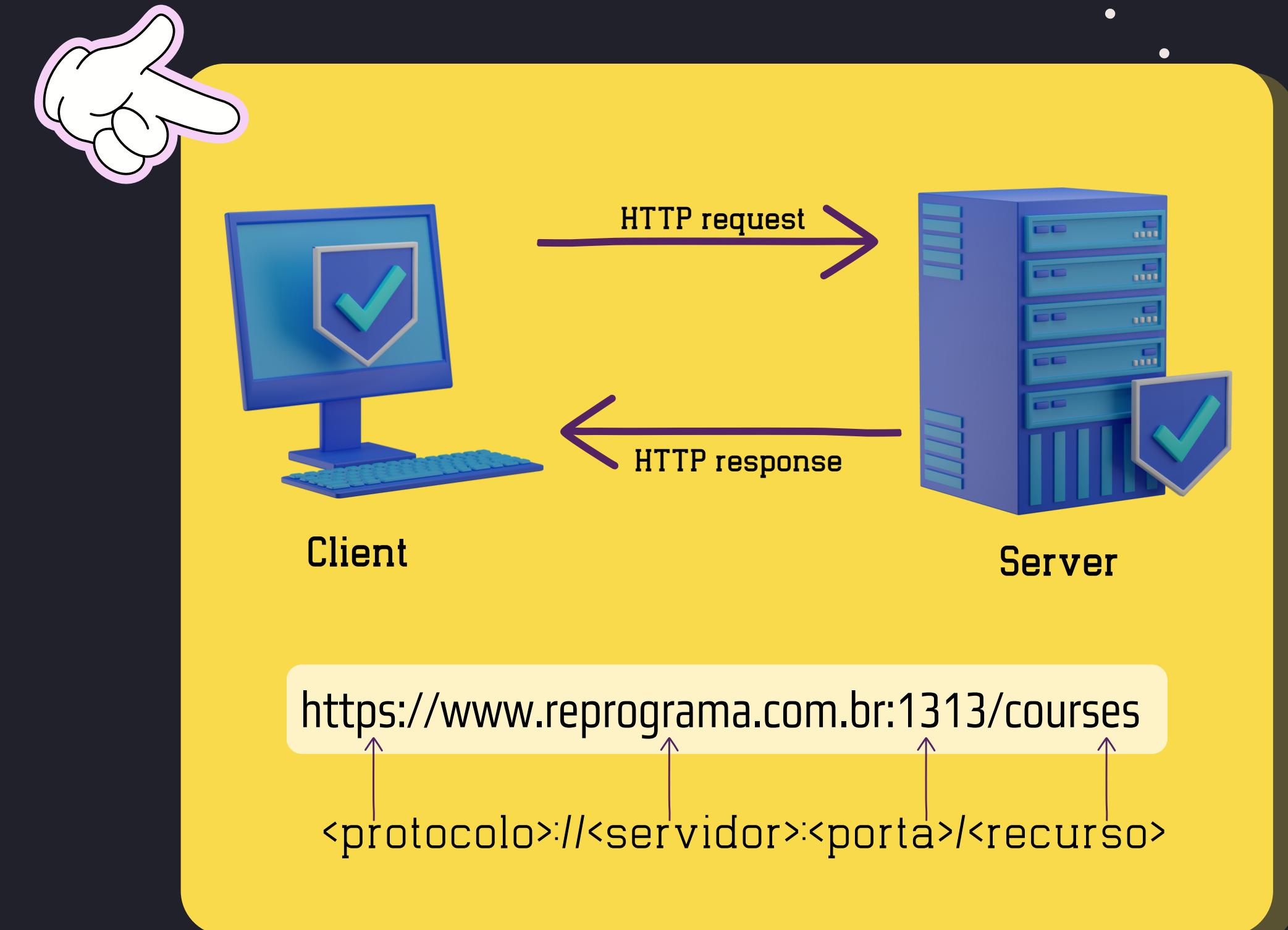
PORQUE LEMBRAR É SOFR... OPS... VIVER



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.

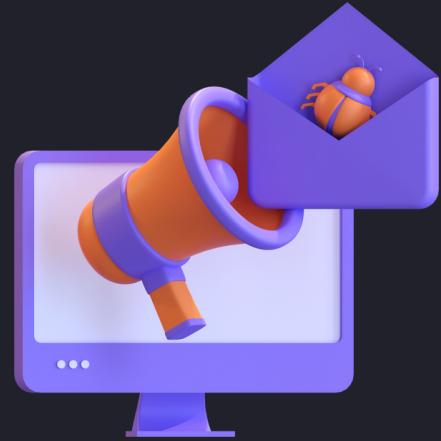


HTTP - STATUS CODE

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.

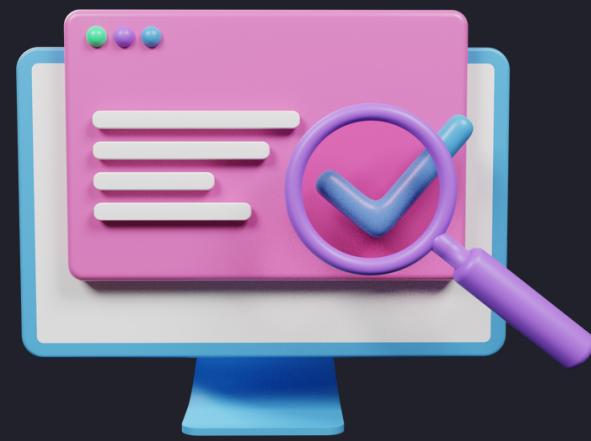
As respostas são divididas em cinco grupos:

RESPOSTAS DE INFORMAÇÕES



(100 - 199)

RESPOSTAS DE SUCESSO



(200 - 299)

RESPOSTAS DE REDIRECIONAMENTO



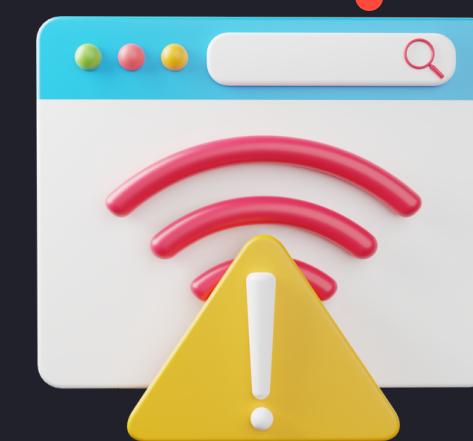
(300 - 399)

ERROS DO CLIENTE



(400 - 499)

ERROS DO SERVIDOR



(500 - 599)



CRUD 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}

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 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.

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]"  
}
```

PUT

O PUT substitui todos os atuais dados do recurso de destino pelos dados passados na requisição. Existe, a possibilidade de atualizar todo o recurso em apenas uma requisição. Por exemplo:

```
PUT /games/:idDoJogo
{
  "nome": "Shadows",
  "desenvolvedor": "Sony",
  "lançamento": 2015,
  "genero": "[aventura, tiro]",
  "console": "[PC]"
}
```

Patch

O Patch aplica modificações parciais em um recurso. Logo, é possível modificar apenas uma parte do recurso. O que torna as coisas mais fáceis. Por exemplo:

```
PATCH /games/:idDoJogo
{
  "nome": "novoTituloDoJogo"
}
```

DELETE

O método DELETE é usado para remover um recurso ou uma coleção de recursos. Quando apertamos o botão “Excluir”, o evento que está sendo disparado passa pelos recursos do método DELETE.

```
DELETE /games/:idDoJogo
```

RESUMÃO:

O **REST API** funciona por meio de uma **requisição HTTP** enviada a partir do sistema de origem para realizar uma determinada ação em um sistema externo.

Para que o sistema de destino entenda qual é a solicitação, a requisição deve conter um verbo de ação e, em alguns casos, dados complementares, como as informações que serão gravadas no banco de dados.

Dessa forma, **o sistema que recebeu** a solicitação processa a requisição e retorna os dados solicitados, além de um código referente ao processamento da requisição HTTP para indicar se ela foi realizada com sucesso ou se encontrou algum tipo de problema. Os principais **verbos** utilizados na API são:

- **GET**: recuperar um ou mais dados do banco de dados;
- **POST**: criar um registro no sistema;
- **PATCH**: atualizar os dados sem a necessidade de passar todos os campos do registro, ou seja, apenas o campo de identificação e o atributo alterado;
- **PUT**: atualizar dados no sistema;
- **DELETE**: excluir um registro do sistema.



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

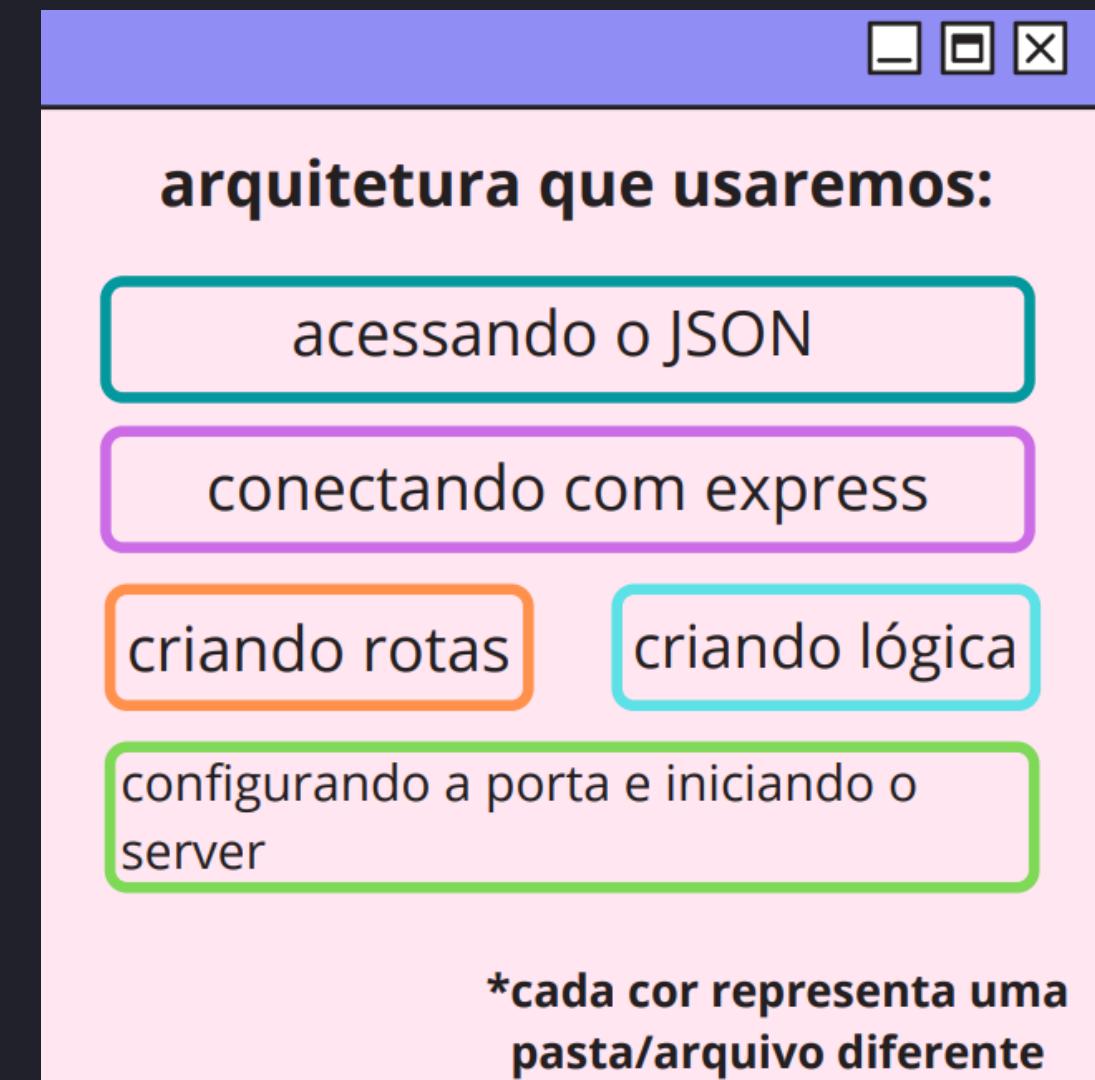
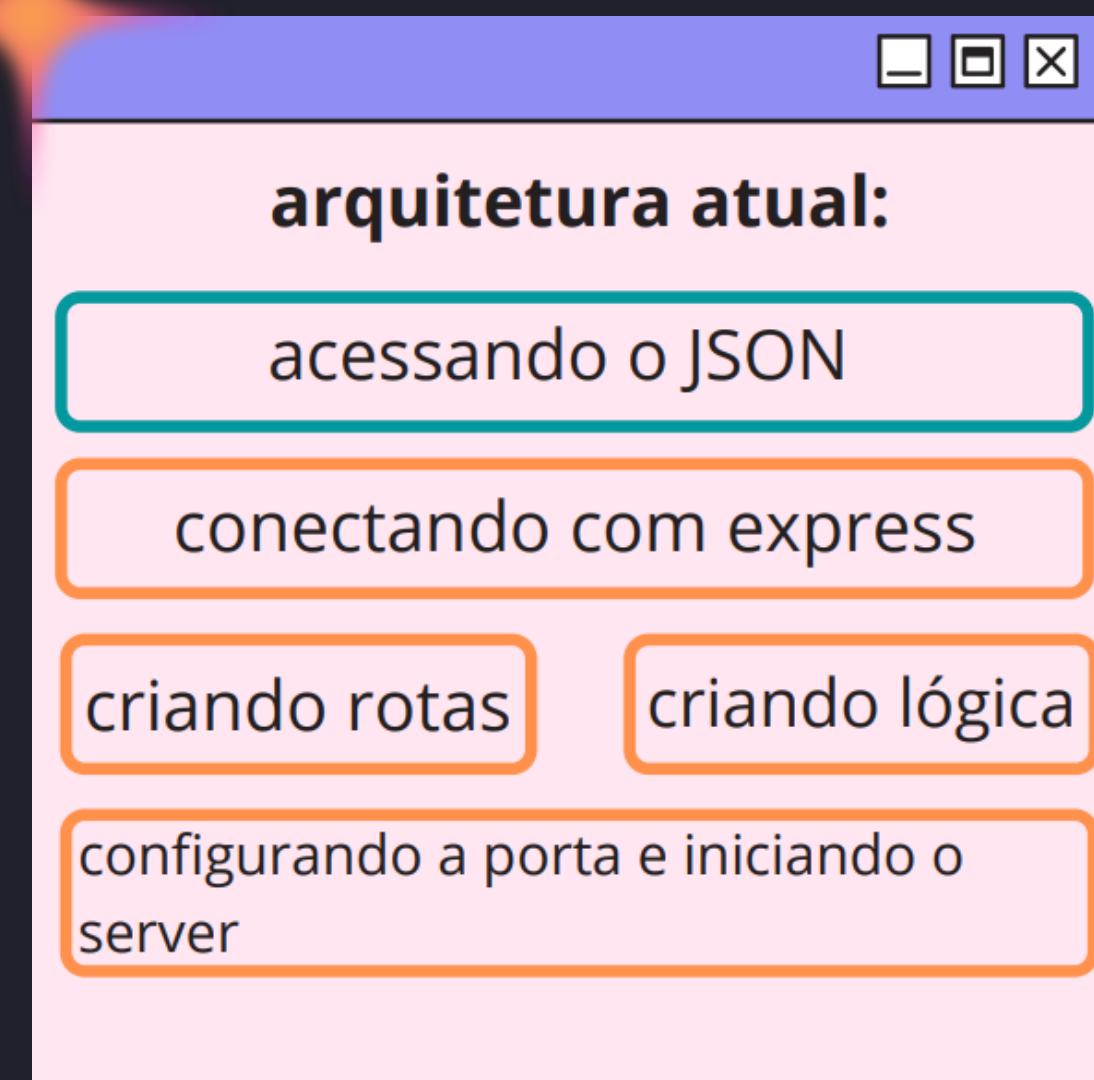
É utilizado para enviar dados que serão cadastrados no banco de dados, podem ser combinados com query ou path params.

EX.: { "disponivel": true}

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 back-end, não lidaremos com as views, porém lidamos com as rotas(routes). O MVC nada mais é que uma forma de organizar o nosso código.



*cada cor representa uma pasta/arquivo diferente

ARQUITETURA MVC



arquitetura que usaremos:

acessando o JSON

conectando com express

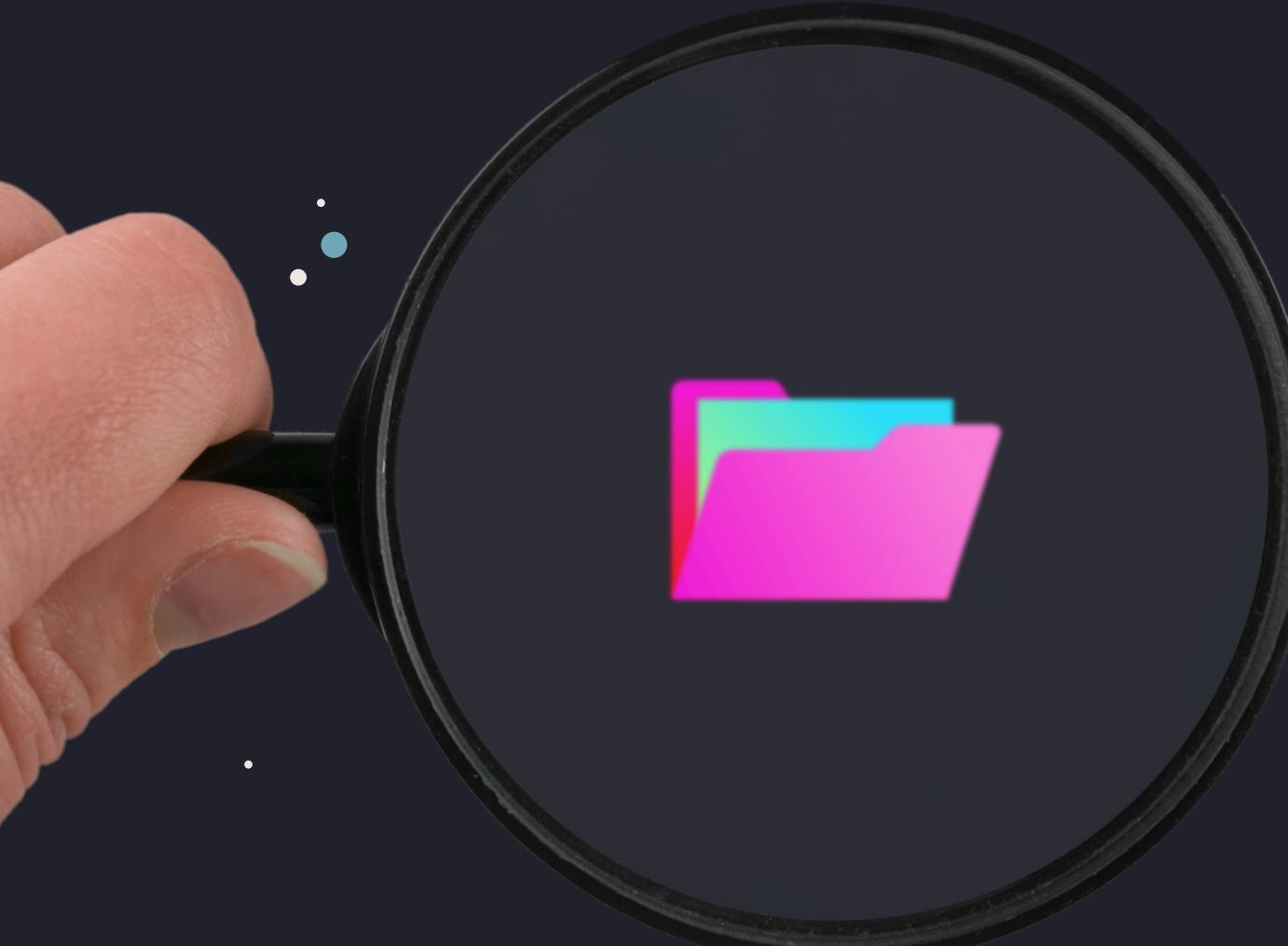
criando rotas

criando lógica

configurando a porta e iniciando o server

*cada cor representa uma pasta/arquivo diferente

ARQUITETURA MVC



ARQUITETURA MUC - EXEMPLO

PROJETO (ESSA É A PASTA RAIZ)

```
|  
|- NODE_MODULES ( DENTRO DA PASTA RAIZ)  
|  
|- SRC (DENTRO DA PASTA RAIZ)  
|  
|   |- CONTROLLERS (DENTRO SRC)  
|   |   |- XCONTROLLER.JS  
|   |   |- YCONTROLLER.JS  
|  
|   |- MODELS (DENTRO SRC)  
|   |   |- XMODEL.JS  
|   |   |- YMODEL.JS  
|  
|   |- ROUTES (DENTRO SRC)  
|   |   |- XROUTES.JS  
|   |   |- YROUTES.JS  
|  
|   |- APP.JS (DENTRO SRC)  
|  
|- .GITIGNORE (PASTA RAIZ)  
|- PACKAGE-LOCK.JSON (PASTA RAIZ)  
|- PAKAGE.JSON (PASTA RAIZ)  
|- README.MD (PASTA RAIZ)  
|- SERUER.JS (PASTA RAIZ)
```

**ENTENDEU? NÃO?
ENTÃO BORA DE DUVIDAS!**



{REPROGRAMA}FY



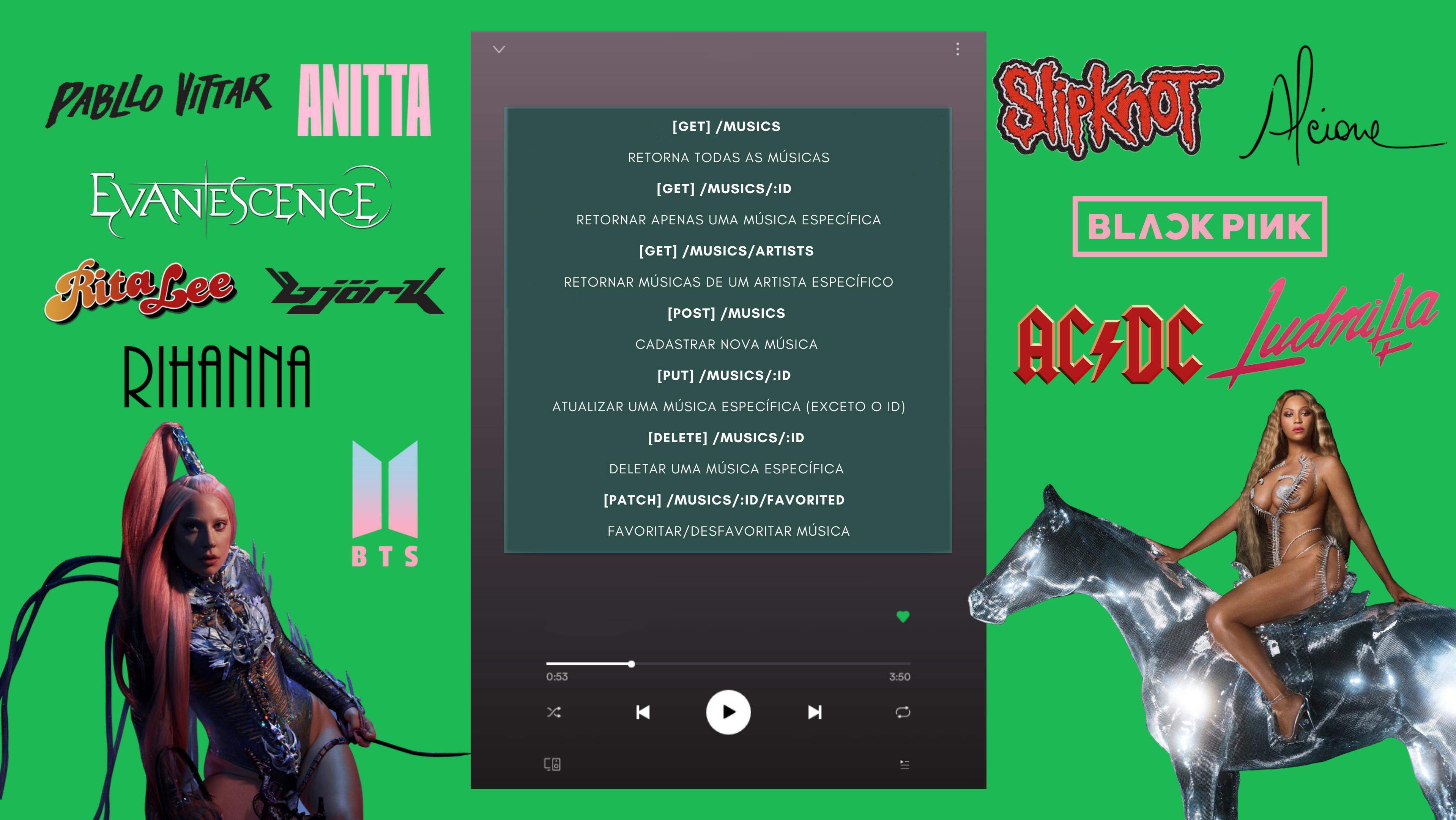
adicionar musicas
e criar podcasts



atualizar e editar
musicas/podcasts



acessar, favoritar ou excluir
musica(s) e podcast(s)



[GET] /MUSICS
RETORNA TODAS AS MÚSICAS

[GET] /MUSICS/:ID
RETORNAR APENAS UMA MÚSICA ESPECÍFICA

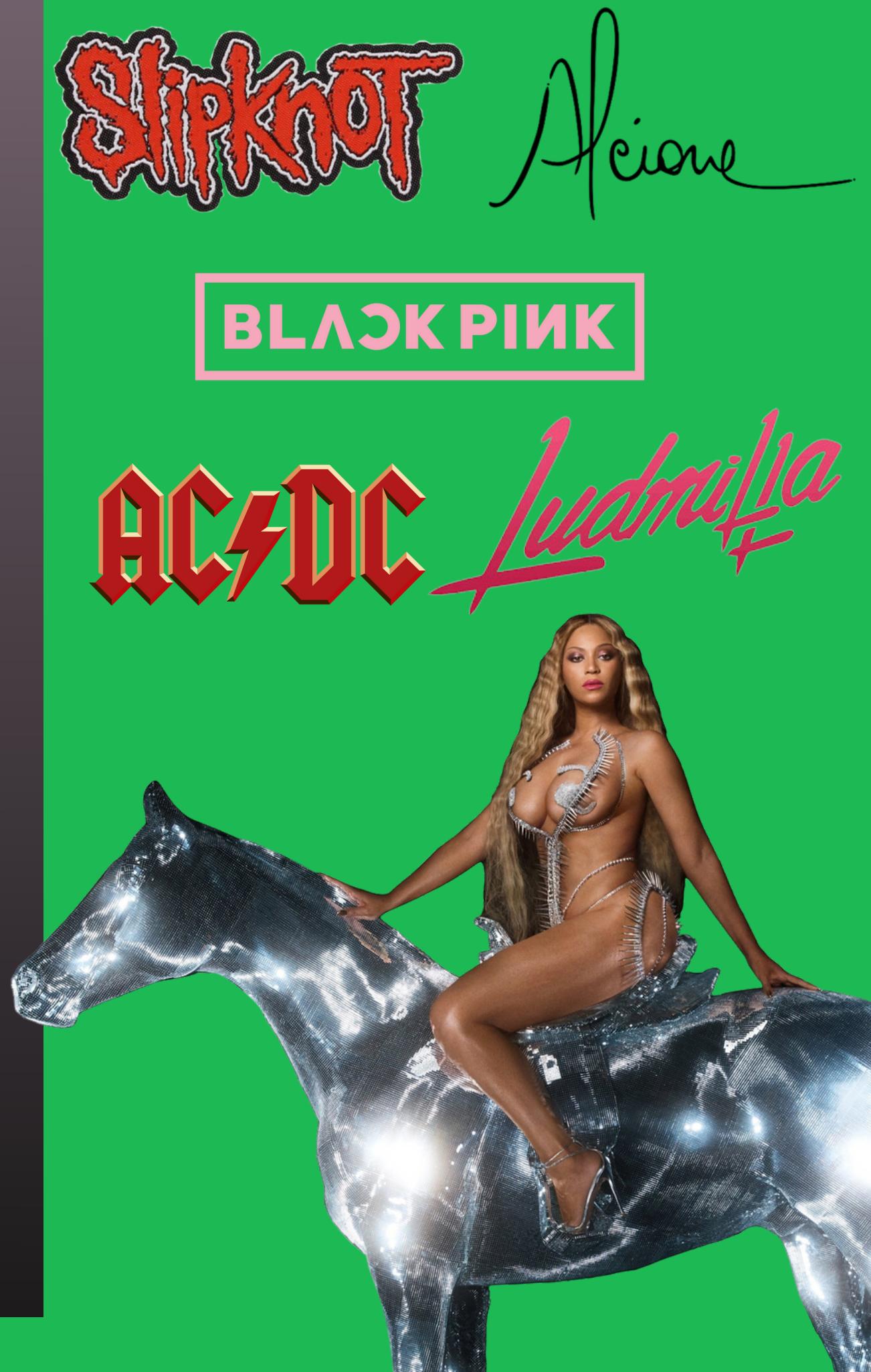
[GET] /MUSICS/ARTISTS
RETORNAR MÚSICAS DE UM ARTISTA ESPECÍFICO

[POST] /MUSICS
CADASTRAR NOVA MÚSICA

[PUT] /MUSICS/:ID
ATUALIZAR UMA MÚSICA ESPECÍFICA (EXCETO O ID)

[DELETE] /MUSICS/:ID
DELETAR UMA MÚSICA ESPECÍFICA

[PATCH] /MUSICS/:ID/FAVORITED
FAVORITAR/DESFAVORITAR MÚSICA





The screenshot shows a dark-themed mobile application interface. At the top, there is a navigation bar with a back arrow and a three-dot menu icon. Below the navigation bar is a large purple rectangular button containing several menu items:

- [GET] /PODCASTS**
RETORNA TODOS OS PODCASTS
- [GET] /PODCASTS/TOPIC**
RETORNAR APENAS OS PODCASTS DE UM TÓPICO ESPECÍFICO
- [POST] /PODCASTS**
CADASTRAR NOVO PODCAST
- [DELETE] /PODCASTS/:ID**
DELETAR UM PODCAST ESPECÍFICO
- [PATCH] /PODCASTS/:ID/STARS**
ALTERAR A NOTA DE UM PODCAST

At the bottom of the screen, there is a playback control bar with a progress bar showing '0:53' and '3:50', a play/pause button, and other standard media controls.

