

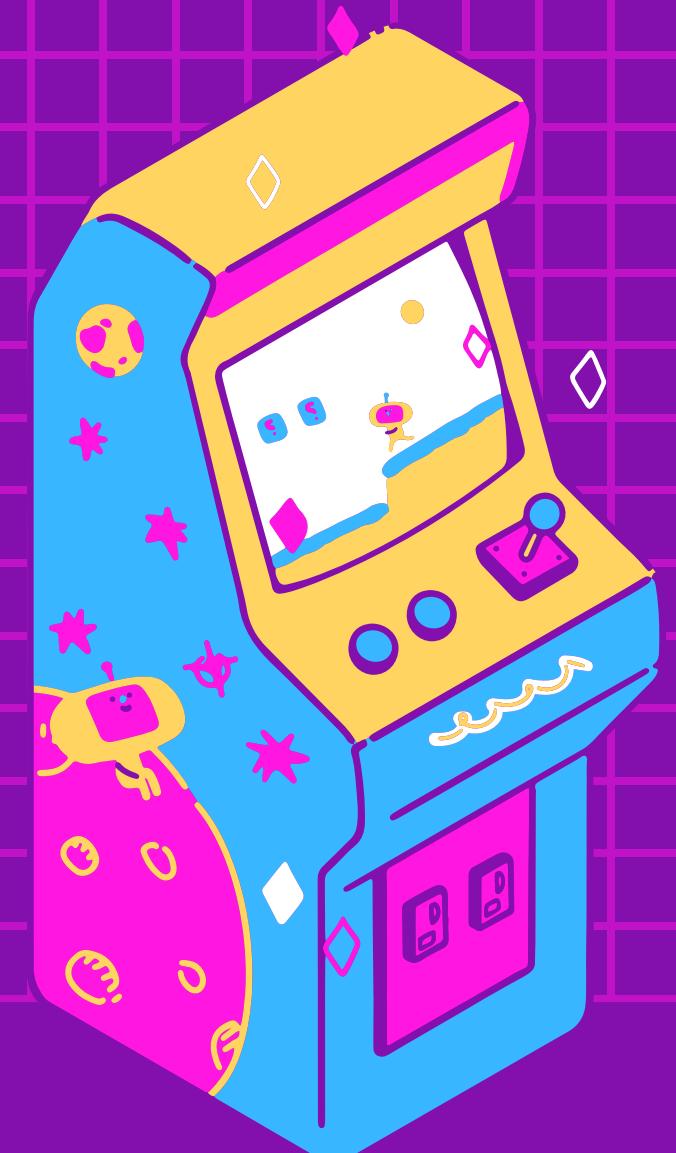
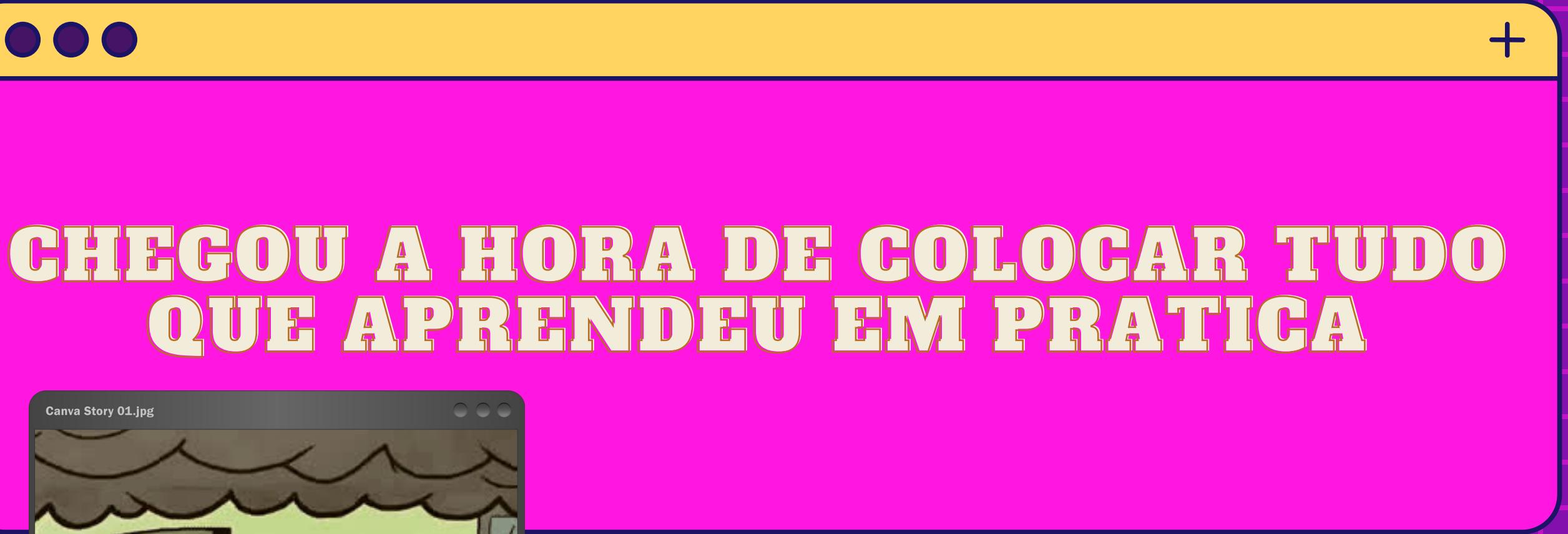
Projeto guiado 3: Crud - DB

LET'S
GO!

{REPROGRAMA}

on19 - semana 13





O que veremos hoje?

9h ás 10h30

Revisão

10h30 ás 10h45 - intervalo

Revisão
10h45 ás 12h

12h ás 13h - almoço

13h ás 14h30

Projeto

14h30 ás 14h45 - intervalo

Projeto
14h45 ás 17h



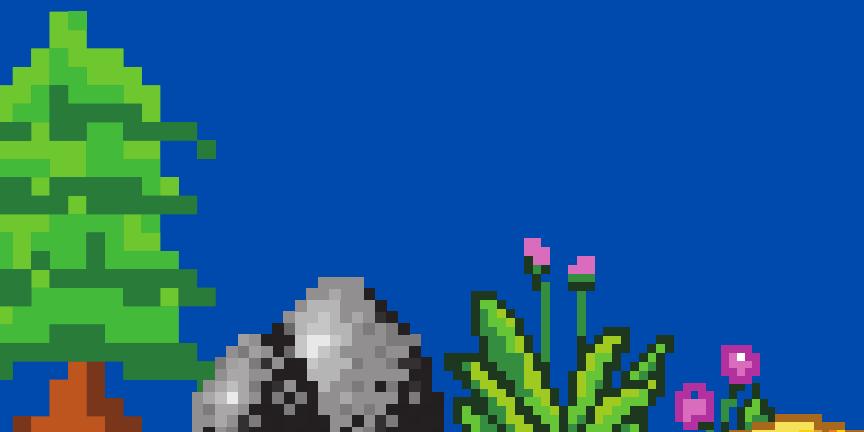
Recadinhos ✓

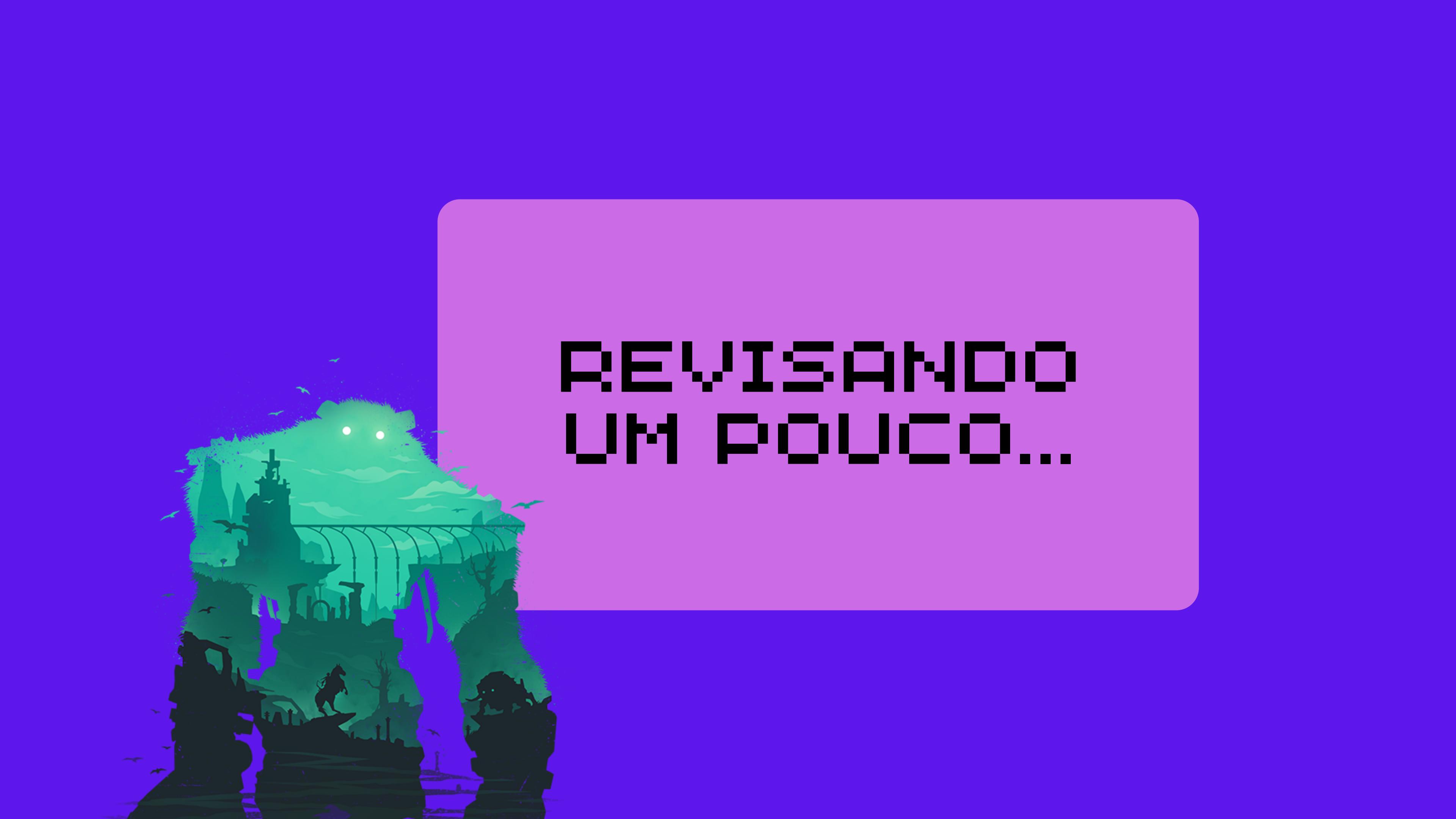
- Meus pronomes (a/ ela/ dela)
- Se hidratem
- Perguntem tudo que quiserem saber
- Deixem a camera ligada
- Levantem a mão no zoom
- A duvida da sua colega pode ser a sua
- Relaxe e divirta-se





Monitoras da manhã





**REVISANDO
UM POUCO...**

CRUD REQUISIÇÕES HTTP API'S

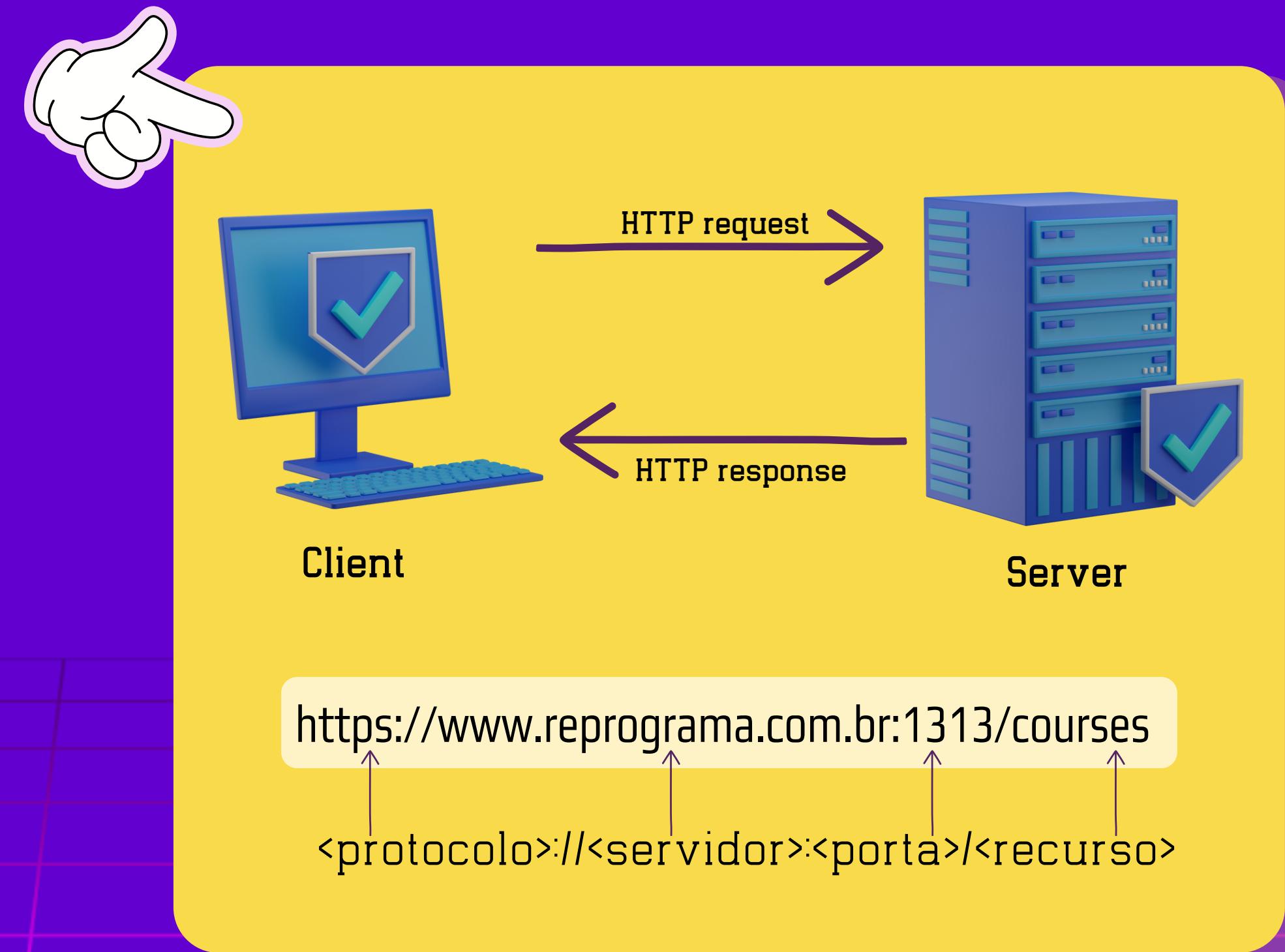
INICIAR



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.

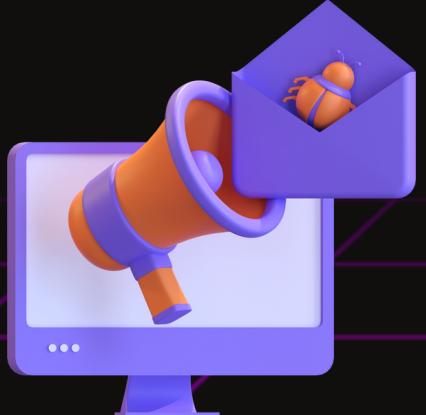


HTTP - STATUS CODE

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)

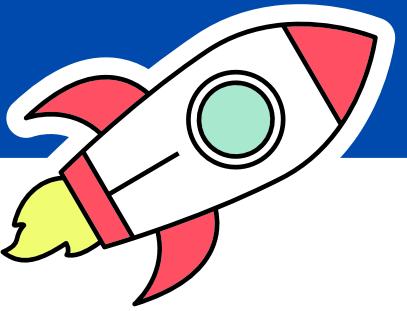


CRUD é a composição da primeira letra de quatro operações básicas de um banco de dados.

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}

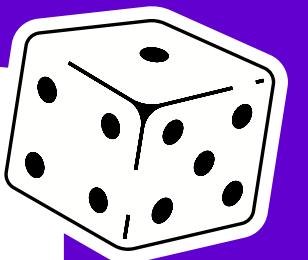


**ENTENDENDO
UMA API**

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.

Aceta: boolean, numero, string...

```
{  
  "_id": new ObjectId("62b0df5fa494af18319efae7"),  
  "name": "God of War",  
  "developer": "Santa Monica Studio",  
  "releaseData": 2018,  
  "genre": ["Action-adventure", "hack and slash"],  
  "mode": ["Single-player"],  
  "available": true,  
  "description": "While the first seven games were loosely",  
  "console": new ObjectId("62b0c3860a5912f473d73c0f")  
  "__v": 0  
},
```

DIDATIZANDO EM 3 PASOS:



1 - Voce vai com suas amigas há 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 necessaria para o que está sendo pedido

MAS COMO FAZER ESSE PEDIDO?

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 .

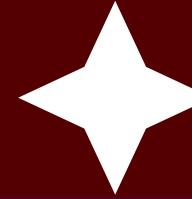
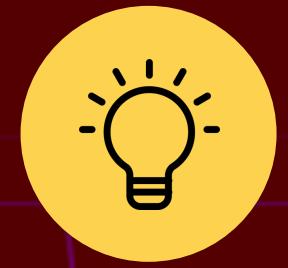
EX: GET /games/:id

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}

READY AS
REQUISICOES



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, usaríamos 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

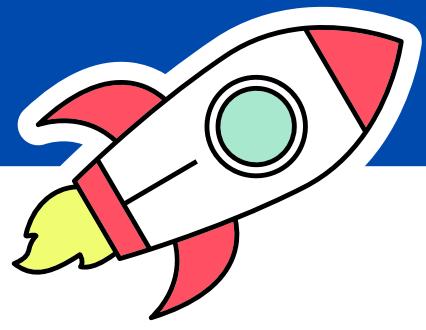
O Patch aplica modificações parciais em um recurso. Logo, é possível modificar apenas uma parte do recurso. O que torna as coisas mais fáceis. Por exemplo:

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

DELETE

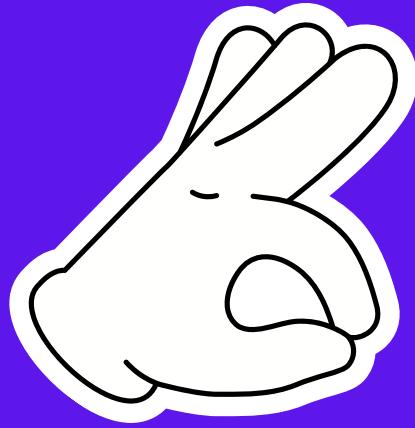
O método DELETE é usado para remover um recurso ou uma coleção de recursos. Quando apertamos o botão “Excluir”, o evento que está sendo disparado passa pelos recursos do método DELETE.

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



O COOIGO

NO CODIGO:



```
const controller = require('../controller/gamesController');
const express = require('express');

const router = express.Router();

router.get("/games", controller.findAllGames);

router.get("/games/:id", controller.findGameById);

router.post("/games/add", controller.addNewGame);

router.patch("/games/:id", controller.updateGame);

router.delete("/games/:id", controller.deleteGame);

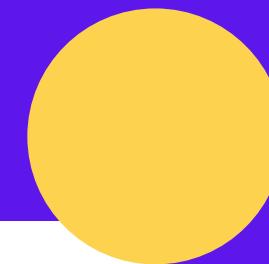
module.exports = router
```

TRADUZINDO:

Aqui estão nossas rotas. para elas funcionarem nós importamos EXPRESS (Framework) e conectamos ao nosso controller e por fim exportamos nossas rotas.

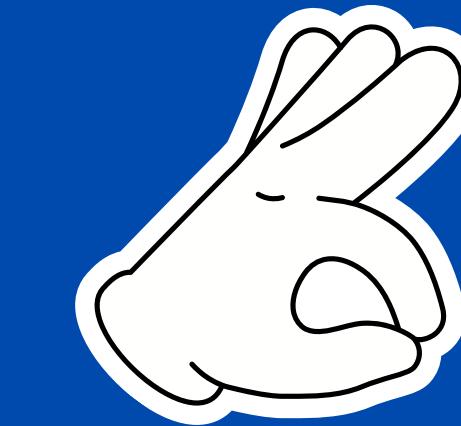
Em suma estamos falando:

Computador utilize a rota Get e vá ate o recurso "games". Utilize o código presente no meu controller para me mostre todos os jogos cadastrados



NO CODIGO:

```
const findGameById = async (req, res) => {
  try {
    const findGame = await GamesModel.findById(req.params.id).populate(
      "console"
    );
    if (findGame == null) {
      res.status(404).json({ message: "Game not available" });
    }
    res.status(200).json(findGame);
  } catch (error) {
    res.status(500).json({ message: error.message });
  };
};
```



TRADUZINDO:

Aqui o que estamos fazendo é basicamente pedindo para que o Computador utilize a rota Get e vá ate o recurso "games". mas dessa vez utilizando o parâmetro do ID.

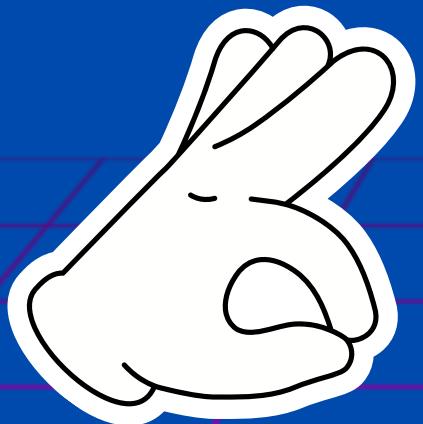
- Caso ele não encontre: mensagem de erro
- Caso ele encontre: mostre o jogo encontrado

Obs: Utilizamos o Async/Await, pois queremos que nosso código execute um processo por vez. Ou seja, que ele receba uma Promise e a transforme em um valor de retorno ou uma mensagem de erro caso não encontre

NO CODIGO:

```
const deleteGame = async (req, res) => {
  try {
    const { id } = req.params;
    const findGames = await GamesModel.findById(id);

    if (findGames == null) {
      return res.status(404).json({ message: `Game with id ${id} not found` })
    };
    await findGames.remove();
    res.status(200).json({ message: `Game with id ${id} was successfully deleted` });
  } catch (error) {
    res.status(500).json({ message: error.message });
  };
};
```



TRADUZINDO:

Assim como o exemplo anterior estamos utilizando o ID como parâmetro. mas dessa vez para excluirmos um jogo.

Computador utilize a rota delete > através do parâmetro ID > encontre o jogo > caso não encontre, me diga > caso encontre remova ele e me diga qual removeu.

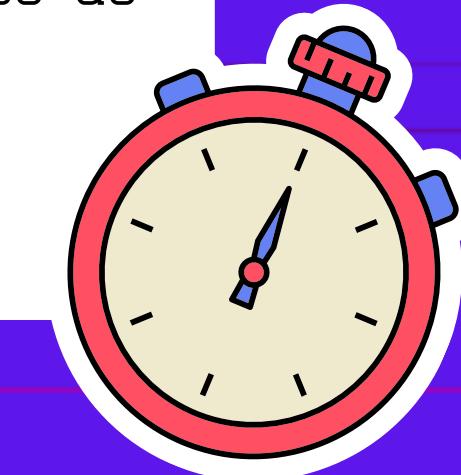
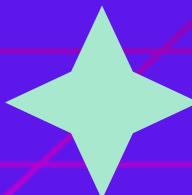
RESUMÃO:

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.



ARQUITETURA MVC:

MVC É UM PADRÃO DE ARQUITETURA DE SOFTWARE, SEPARANDO SUA APLICAÇÃO EM 3 CAMADAS:

- 1- A CAMADA DE INTERAÇÃO DO USUÁRIO(VIEW);
- 2- A CAMADA DE MANIPULAÇÃO DOS DADOS(MODEL);
- 3- 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



SERVER.JS

configurando a porta e iniciando o servidor

APP.JS

indicação das rotas e requisições

gamesRoutes.js

rotas e verbos
pasta routes

gamesController.js

toda a logica que estamos utilizando

ARQUITETURA MUC - EXEMPLO

PROJETO (ESSA É A PASTA RAIZ)

```
|- NODE_MODULES (DENTRO DA PASTA RAIZ)  
|- SRC (DENTRO DA PASTA RAIZ)  
|   |- DATABASE (DENTRO SRC)  
|     |- MONGOOSECONNECT.JS  
|- 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)  
|- SERVER.JS (PASTA RAIZ)
```

DEPENDENCIAS:

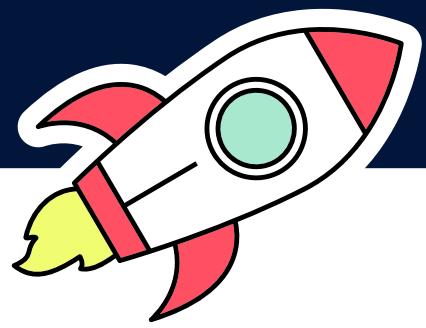


O Express.js é um Framework rápido e um dos mais utilizados em conjunto com o Node.js, facilitando no desenvolvimento de aplicações back-end e até, em conjunto com sistemas de templates, aplicações fullstack.



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.



BANCO DE
OABOS



O que é um B. D. ?

Segundo o site oficial da Oracle: Um banco de dados é uma coleção organizada de informações - ou dados - estruturadas, normalmente armazenadas eletronicamente em um sistema de computador. Um banco de dados é geralmente controlado por um sistema de gerenciamento de banco de dados (DBMS).

Juntos, os dados e o DBMS, juntamente com os aplicativos associados a eles, são chamados de sistema de banco de dados, geralmente abreviados para apenas banco de dados



Utilizamos bancos de dados o tempo todo

Notas

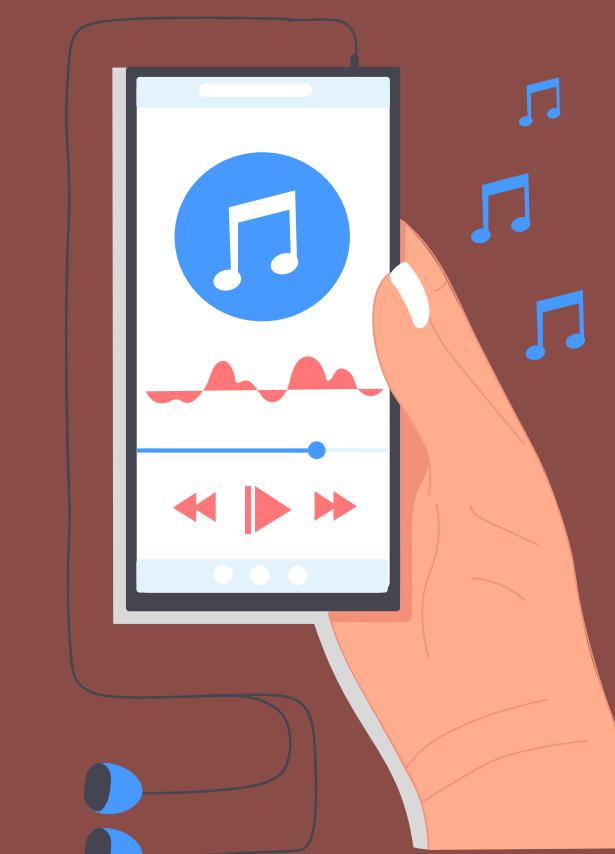
Uma professora inserindo notas em um computador.

Elas são armazenadas e no final do bimestre/ano podem ser consultadas.



Musicas

Uma playlist que armazena todas as suas musicas. E que pode ser consultada de diversas localidades com o uso do seu aparelho eletronico.



Memoria

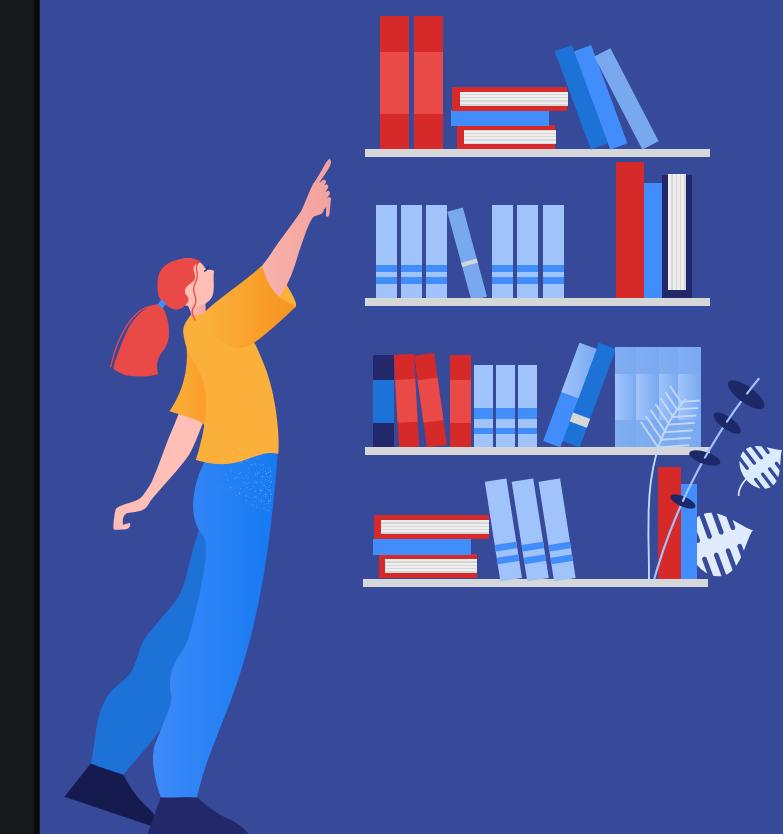
Seu próprio cérebro é um grande banco de dados.

Principalmente a parte do hipocampo que armazena novas memorias e informações.



Livros

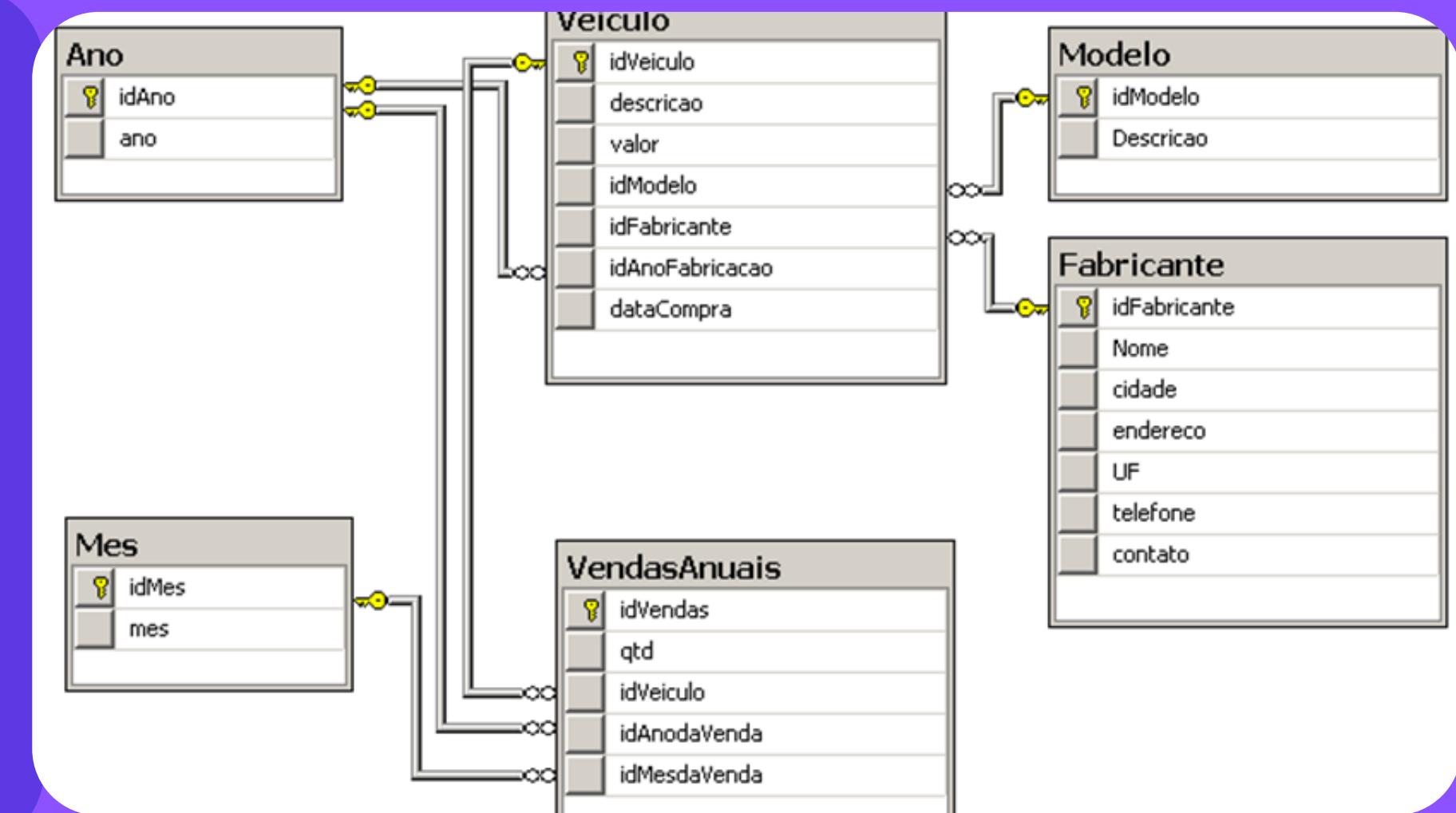
Uma biblioteca é uma especie de banco de dados. Mas, atualmente todos os livros e seus IDs são armazanados em um unico sistema. Facilitando o acesso.



SQL

SQL é uma linguagem de programação usada por quase todos os bancos de dados relacionais para consultar, manipular e definir dados e fornecer controle de acesso. O SQL foi desenvolvido pela primeira vez na IBM com a Oracle como principal contribuinte, o que levou à implementação do padrão SQL ANSI; o SQL estimulou muitas extensões de empresas como IBM, Oracle e Microsoft.

1. Usado principalmente no trabalho com vendas, marketing, negócios, e etc.
2. As linguagens SQL têm variações de sintaxe. Diferentes empresas seguem distintos conjuntos de bancos de dados, são apenas pequenas variações, mas é essencial estar ciente deles.
3. SQL APENAS se comunica com bancos de dados relacionais. Ou seja, qualquer banco de dados com uma organização tabular (com linhas e colunas).

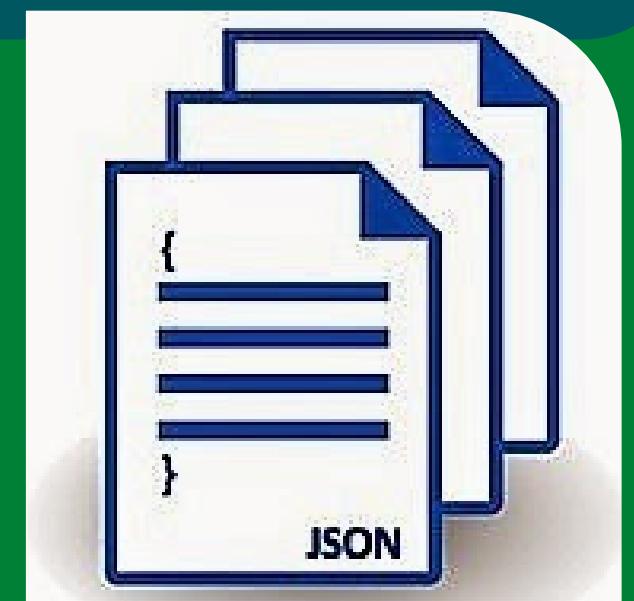


NoSQL

O termo 'NoSQL' se refere a tipos não relacionais de bancos de dados, e esses bancos de dados armazenam dados em um formato diferente das tabelas relacionais. No entanto, os bancos de dados NoSQL podem ser consultados usando APIs de linguagem idiomática, linguagens de consulta estruturadas declarativas e linguagens de consulta por exemplo, razão pela qual também são chamados de bancos de dados "não apenas SQL".

Os bancos de dados NoSQL também são a escolha preferida dos desenvolvedores, pois eles naturalmente aceitam um paradigma de desenvolvimento ágil, adaptando-se rapidamente aos requisitos em constante mudança.

- 1 - NoSQL permite que os dados sejam armazenados de maneiras mais intuitivas e fáceis de entender, ou mais próximas da maneira como os dados são usados pelos aplicativos - com menos transformações necessárias ao armazenar ou recuperar usando APIs no estilo NoSQL .
- 2 - os bancos de dados NoSQL podem aproveitar ao máximo a nuvem para oferecer tempo de inatividade zero.



MongoDB ➤

O MongoDB é um banco de dados orientado a documentos que possui código aberto (open source) e foi projetado para armazenar uma grande escala de dados, além de permitir que se trabalhe de forma eficiente com grandes volumes. Ele é categorizado no banco de dados NoSQL (not only SQL) pois o armazenamento e a recuperação de dados no MongoDB não são feitas no formato de tabelas.



MongoDB



01

O MongoDB é um servidor de banco de dados, onde as informações são armazenadas, mas o processo é mais fluido, independente, com os elementos tendo identificações únicas. Resumindo, o ambiente fornece um servidor que se pode iniciar e, em seguida, criar vários bancos de dados usando o MongoDB.



02

- O MongoDB contém coleções.
- A sua vantagem é a permissão para criar vários bancos de dados e várias coleções dentro do principal.
- Na coleção, estão documentos que contêm os dados que vamos armazenar no banco do MongoDB, e uma única coleção pode conter vários documentos.
- Não existe esquema de tipo, isso significa que não é necessário que um documento seja semelhante ao outro.
- Nos documentos, pode-se armazenar dados aninhados dentro de SCHEMAS.
- Essa conexão de dados permite criar relações complexas entre eles e armazená-los no mesmo documento, o que torna o trabalho e a busca mais eficientes em comparação com o SQL.

COMPASS

Compass é uma ferramenta interativa para consultar, otimizar e analisar os dados do MongoDB. A ferramenta MongoDB Compass, disponível nas assinaturas Enterprise e Professional, permite visualizar planos de execução de uma maneira bem prática e didática, sendo que cada estágio do pipeline de execução é exibido como um nó em uma árvore, como mostra a figura abaixo:

SHELL

O shell Javascript do Mongodb lhe permite uma interação com a instância do banco de dados Mongo a partir da linha de comando. O shell é uma ferramenta para desempenhar funções de administração, monitoramento de uma instancia em execução ou inserindo documentos. Ou seja ele funciona como um terminal para o MongoDB. Sendo possível inserir dados, organizar schemas e orientar os documentos a partir dele.

ROBO3T

O Robo 3T (anteriormente Robomongo) é uma ferramenta de gerenciamento MongoDB de plataforma cruzada e concêntrica. Diferentemente da maioria das outras ferramentas de interface do usuário do MongoDB, o Robo 3T incorpora o shell mongo real em uma interface com guias, com acesso a uma linha de comando do shell.

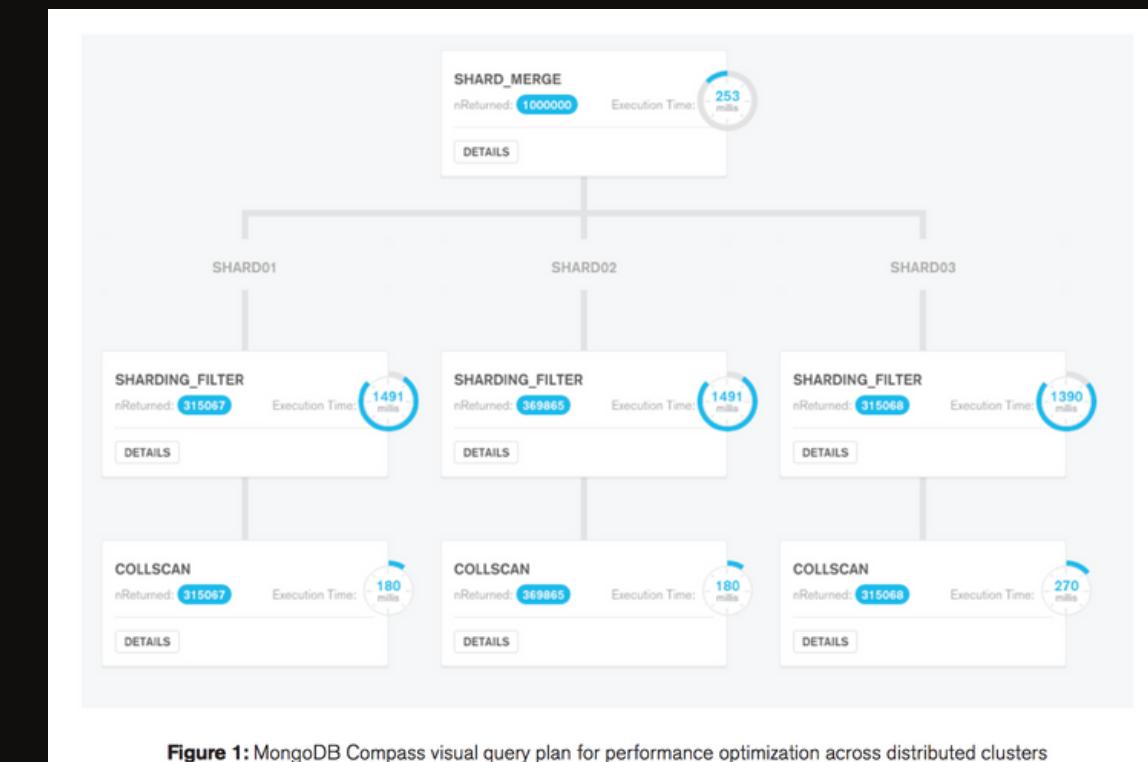


Figure 1: MongoDB Compass visual query plan for performance optimization across distributed clusters

```
C:\Program Files\MongoDB\Server\3.2\bin>mongo
MongoDB shell version: 3.2.6
connecting to: test
>
```

The screenshot displays the Robo 3T application window titled "Robo 3T - 1.1". On the left is a sidebar with a tree view of databases and collections, including "Mongo Local" with "15" collections like System, blog, course, etc., and "vdb" with "2" collections like categorias_vistorias and modelos_vistorias. The main area shows a query result for the "modelos_vistorias" collection with the command "db.getCollection('modelos_vistorias').find({})". The results are listed in a table with columns "Key", "Value", and "Type". The table contains 14 rows of document data. At the bottom, there is a "Logs" tab.

TERMINOLOGIA

COLEÇÕES ("COLLECTIONS")

- As coleções (ou 'Collections', em inglês) no MongoDB são equivalentes às tabelas dos bancos de dados relacionais, podendo guardar múltiplos documentos JSON.

DOCUMENTOS ("DOCUMENTS")

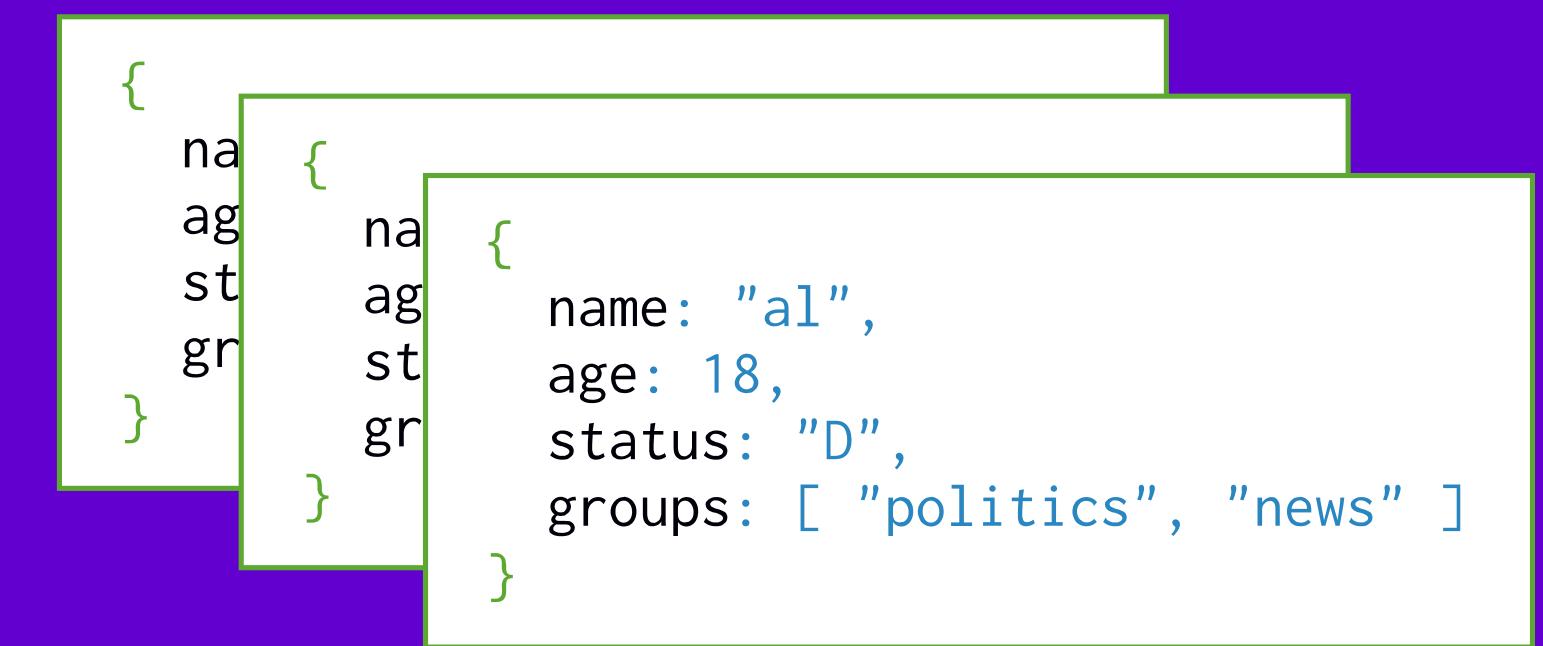
- Os documentos (ou 'Documents', em inglês) equivalem aos registros ou às linhas de dados no SQL. Enquanto uma linha em um banco SQL pode referenciar dados em outras tabelas, os documentos do MongoDB normalmente combinam isso dentro de um único documento.

ESQUEMA ("SCHEMA")

- Embora o MongoDB não possua esquemas, o SQL define esquemas por meio da definição de uma tabela. Um "esquema" no Mongoose é uma estrutura de dados de documento (ou a forma de um documento), que é aplicada por meio da camada da aplicação.

MODELOS ("MODELS")

- Os modelos (ou 'Models', em inglês) são construtores de ordem superior, que utilizam um esquema e instanciam um documento, equivalente aos registros de um banco de dados relacional.

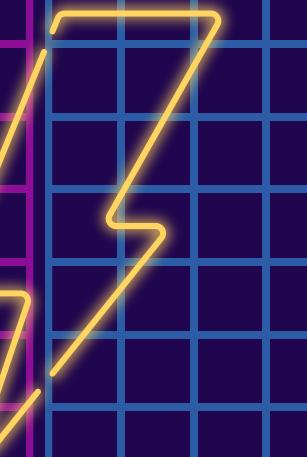
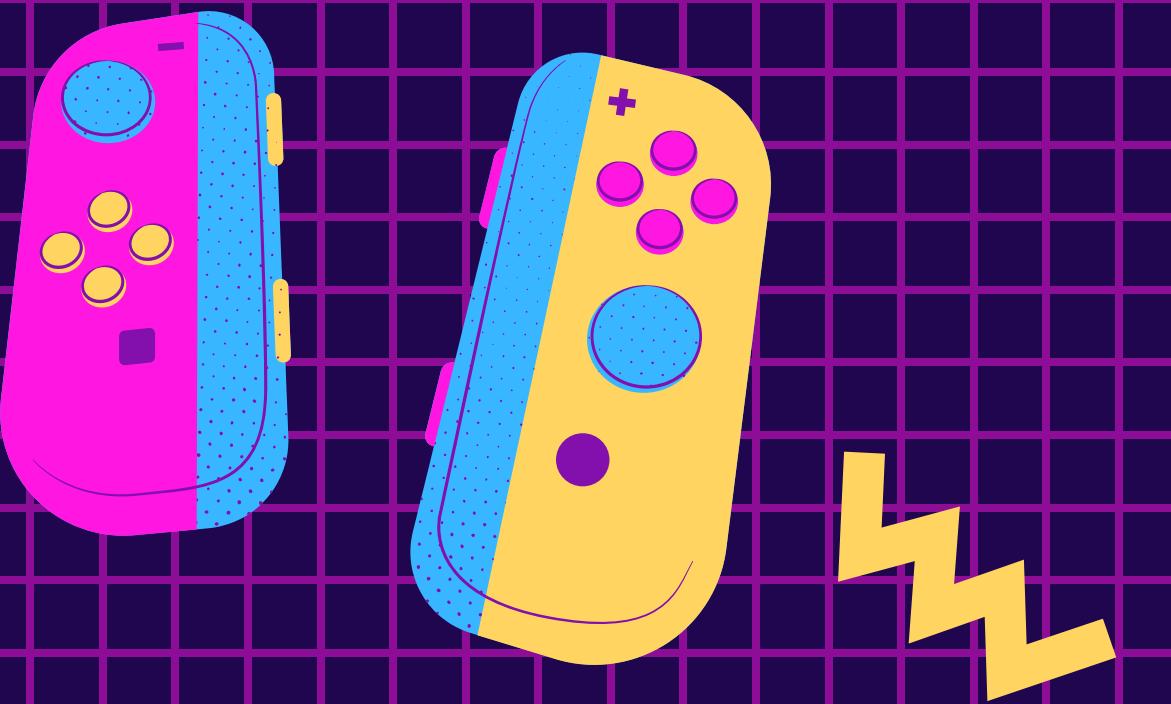


VAMOS ALMOÇAR?

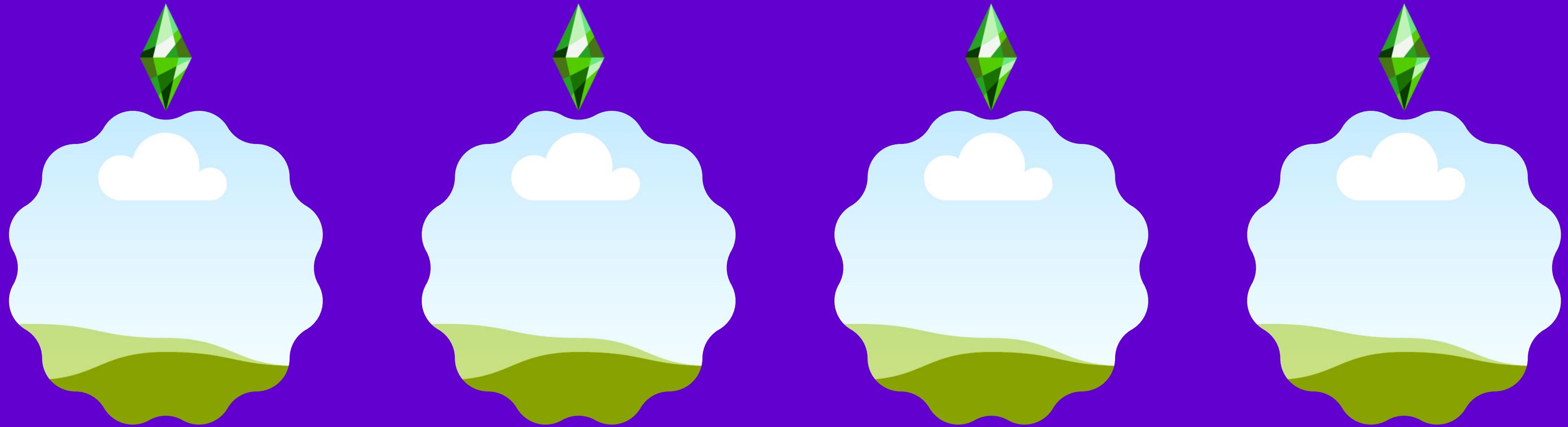


PROJETO

3



Monitoras da tarde



GAME STORE

CRUD COMPLETO:
PODEREMOS
CADASTRAR,
ACESSAR, ALTERAR
E EXCLUIR
CONSOLES OU
GAMES

NESSA API
TEREMOS 2
SCHEMAS
AMBAS SERÃO
CONECTADAS

LET'S **FINISH HIM !!**





[GET] /games

- Retorna todos os jogos

[GET] /games/:id

- Retornar apenas um jogo específico

[POST] /games

- Cadastrar novo jogo

[PATCH] /games/:id

- Atualizar um jogo específico

[DELETE] /games/:id

- Deletar um jogo específico

[PATCH] /games/:id/liked

- Atualizar se gostou ou não do jogo.

[GET] /consoles

- Retorna todos os consoles

[GET] /consoles/:id

- Retornar apenas um console específico

[POST] /consoles

- Cadastrar novo console

[PATCH] /consoles/:id

- Atualizar um console específico

[DELETE] /consoles/:id

- Deletar um console específico

[PATCH] /consoles/:id/available

- Atualizar se o console está ou não disponível





Trabalharemos com as seguintes tipos em uma schema:

String, Number e Boolean

Utilizando alguns desses elementos:
Id, array e limite de caracteres.

Nossa Schema conseguira obter informações
de contato com a desenvolvedora



S.A.C. DAS BONEKAS



"Gaiacita posso cadastrar meus jogos favoritos"

- Logico mor, segue teu critério;



"Tia Gaia não conheço jogos da geração atual"

- Gata pode cadastrar até o jogo da cobrinha ou xadrez se tu quiser;



"Prof. Gaia deu erro no banco de dados"

- Fique calma, no fim da aula vou olhar tudo e revisar com vocês ao longo da semana;



"Gaiazinha dos astros não gostei da aula"

- Imediatamente tenho crise de ansiedade



"Namorada do Kratos ainda tenho duvidas"

- Pois tire elas *-*

GAME
OVER

