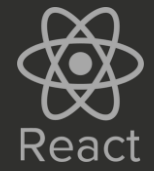


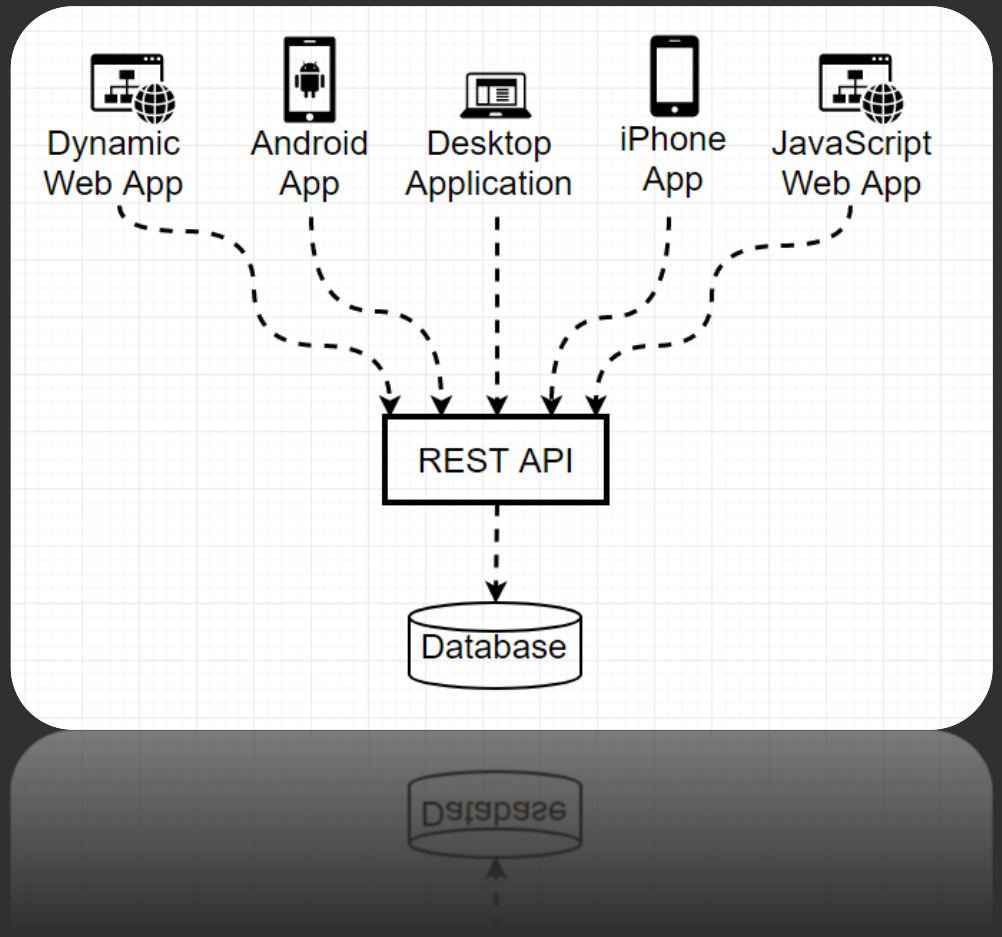
Aula 01.2:

Desenvolvendo uma API Rest com Node.js

API - Application Programming Interface

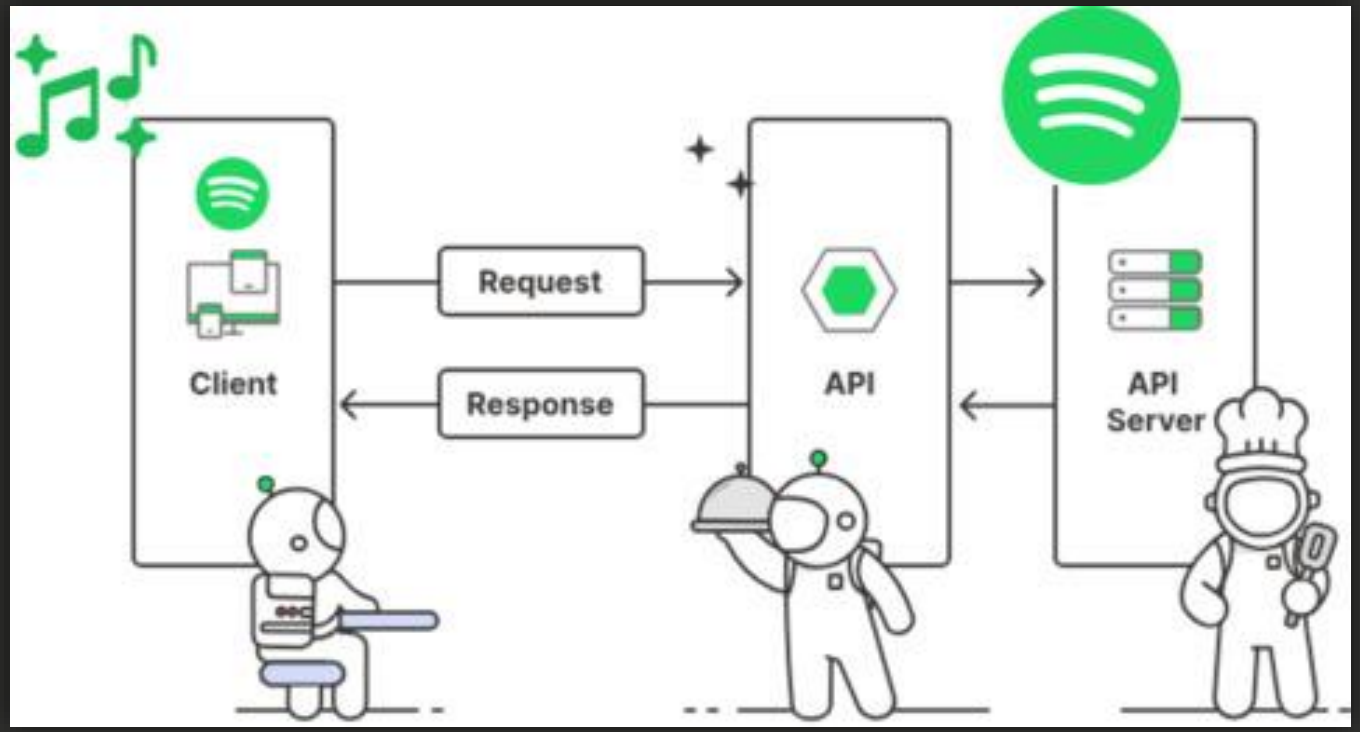
API significa Interface de Programação de Aplicações (Application Programming Interface).

É um conjunto de regras e definições que permite que diferentes softwares se comuniquem entre si.

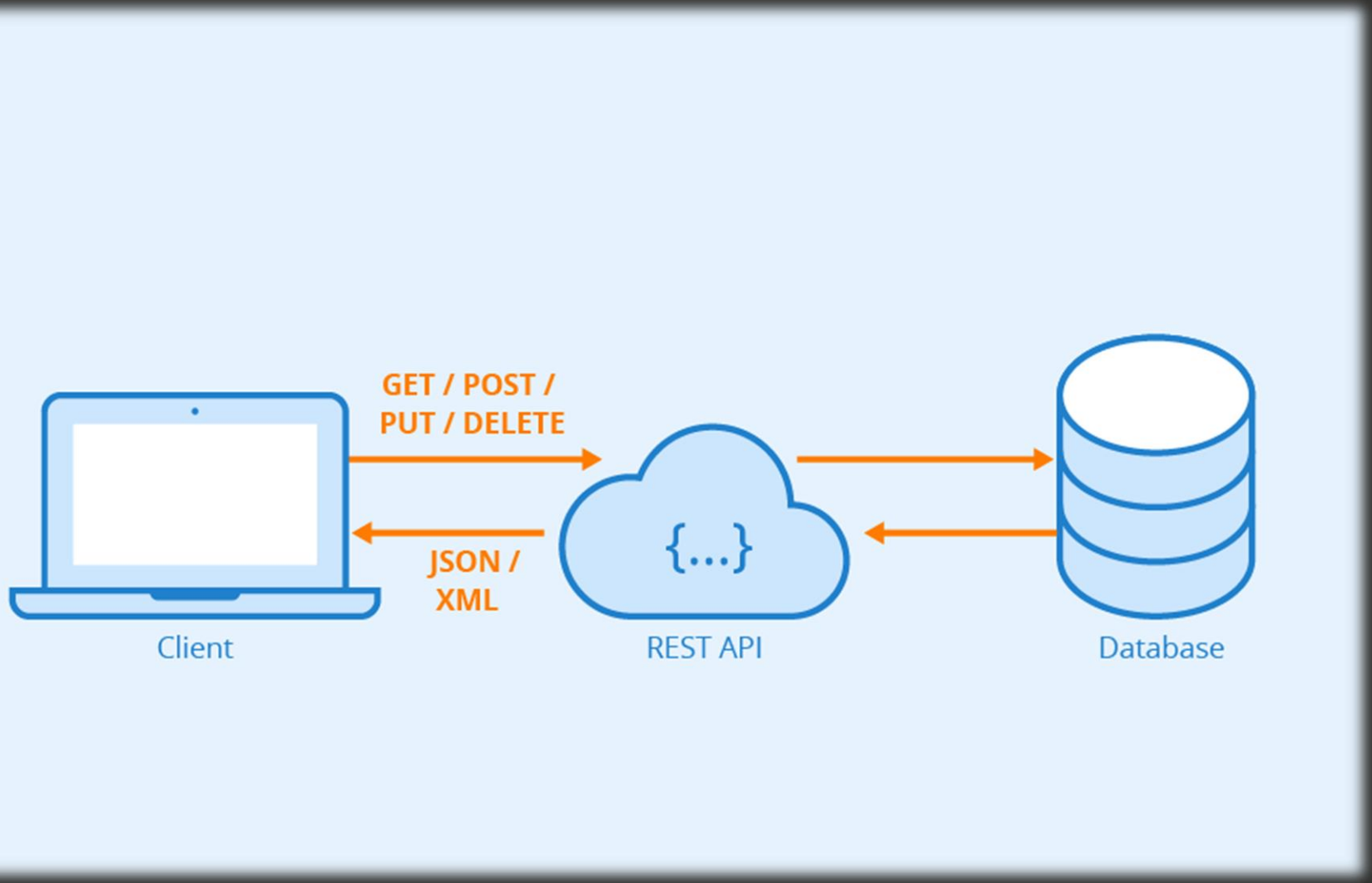


API - Application Programming Interface

Uma API é como um "conector" que deixa diferentes programas conversarem entre si. Em vez de precisar entender tudo que acontece dentro de um software, os desenvolvedores usam a API, que dá as regras e as maneiras corretas de se comunicar com o sistema.



API - Application Programming Interface



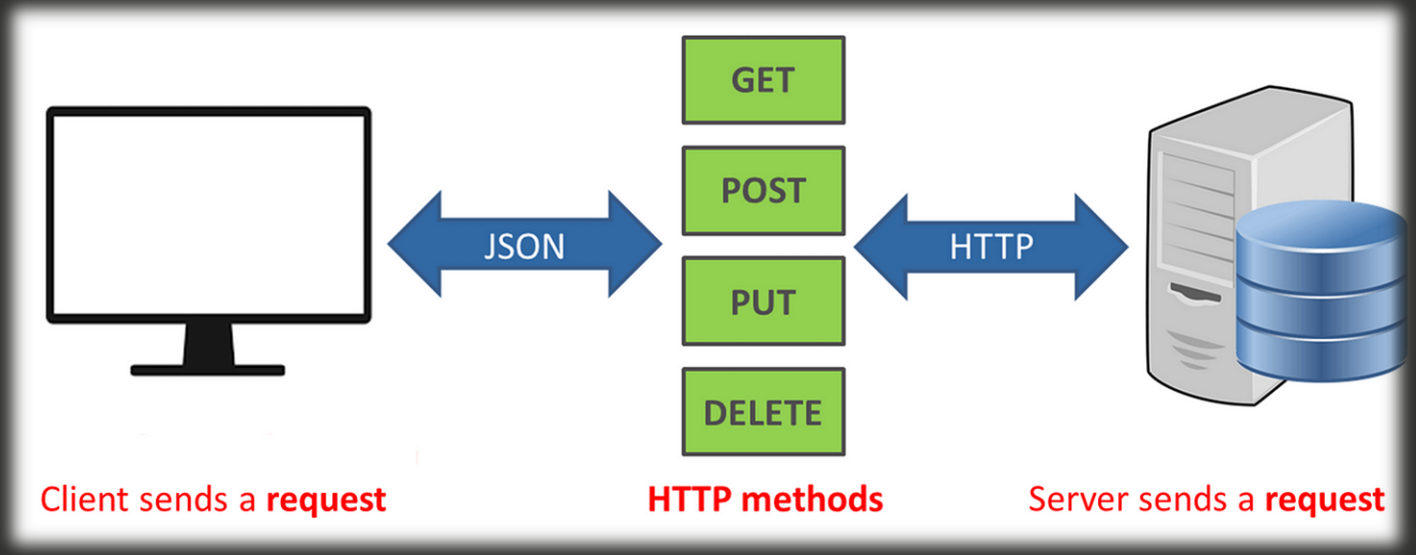
API - Application Programming Interface

Através das APIs é possível criar aplicativos que usam funcionalidades de outros lugares, facilitando a vida dos programadores.

Sendo assim as APIs são muito úteis para permitir que diferentes sistemas, mesmo que estejam em ambientes distintos, troquem informações e funcionem bem juntos.

APIs REST

APIs REST são APIs web que seguem os princípios da **arquitetura REST** (*Transferência de Estado Representacional*, que é um estilo de arquitetura de software), utilizando **HTTP** para comunicação e geralmente operando com os **métodos HTTP** (**GET**, **POST**, **PUT**, **DELETE**) para manipular recursos.



Formato JSON

Muitas vezes, os dados retornados por uma API estarão no formato **JSON** (Javascript Object Notation).

O **JSON** é um formato de dados leve e de fácil leitura utilizado para troca de informações entre sistemas computacionais. Ele é frequentemente usado para transmitir dados entre um servidor e um cliente em aplicações web e móveis, embora também seja utilizado em diversos outros contextos.

Ele é amplamente utilizado na web para representar dados estruturados de forma legível tanto para humanos quanto para máquinas. Em resumo, o JSON é uma forma popular de representar dados estruturados e transferi-los entre diferentes sistemas.

FileInfo.com Example.json

```
{
  "users": [
    {
      "userId": 1,
      "firstName": "Chris",
      "lastName": "Lee",
      "phoneNumber": "555-555-5555",
      "emailAddress": "clee@fileinfo.com"
    },
    {
      "userId": 2,
      "firstName": "Action",
      "lastName": "Jackson",
      "phoneNumber": "555-555-5556",
      "emailAddress": "ajackson@fileinfo.com"
    },
    {
      "userId": 3,
      "firstName": "Ross",
      "lastName": "Bing",
      "phoneNumber": "555-555-5557",
      "emailAddress": "rbing@fileinfo.com"
    },
```

Exemplo de API pública:

https://www.freetogame.com/api/games

Métodos HTTP

GET: O método GET é usado para solicitar dados de um recurso específico. Ele é usado para recuperar informações, e não para modificá-las. Por exemplo, ao fazer uma solicitação GET para uma URL de API de usuários, você pode obter uma lista de todos os usuários ou os detalhes de um usuário específico.

POST: O método POST é usado para enviar dados para o servidor criar um novo recurso. Ele é comumente usado para criar novos registros ou enviar dados que serão processados pelo servidor. Por exemplo, ao enviar um formulário de cadastro de usuário, você usaria o método POST para enviar os dados do novo usuário para o servidor.

Métodos HTTP

PUT: O método PUT é usado para atualizar ou substituir completamente um recurso existente. Quando você envia uma solicitação PUT, você está substituindo o recurso inteiro com os dados que você está enviando.
Por exemplo, ao enviar uma solicitação PUT para uma URL de API de usuário com novos dados, você está substituindo completamente os dados do usuário existente pelos novos dados.

DELETE: O método DELETE é usado para excluir um recurso específico do servidor. Ele é usado quando você deseja remover permanentemente um recurso. Por exemplo, ao enviar uma solicitação DELETE para uma URL de API de usuário com o ID do usuário que você deseja excluir, você está solicitando que o servidor remova esse usuário do banco de dados.

Códigos de status HTTP

200 (OK): Indica que a requisição foi bem sucedida. É o código que o servidor retorna quando a requisição foi processada com sucesso.

201 (Created): Indica que a requisição foi bem sucedida e resultou na criação de um novo recurso no servidor.

204 (No Content): Indica que a requisição foi bem sucedida, mas não há conteúdo para retornar ao cliente. É comum em respostas de requisições que não retornam dados, como atualizações de status.

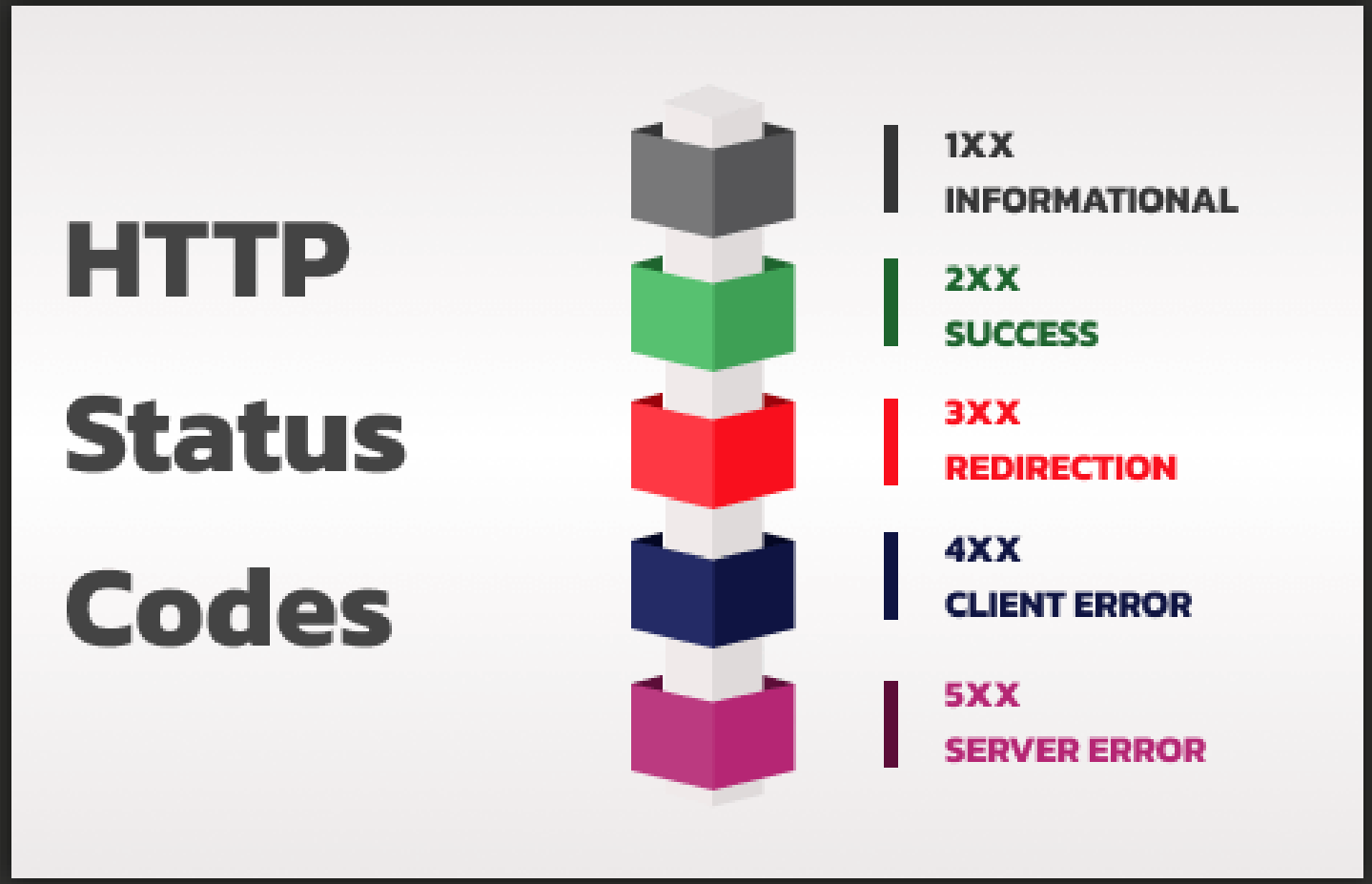
Códigos de status HTTP

400 (Bad Request): Indica que a requisição feita pelo cliente é inválida, geralmente devido a dados malformados ou ausentes.

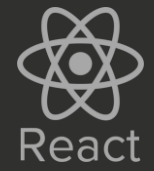
404 (Not Found): Indica que o recurso requisitado não foi encontrado no servidor. É o código de erro padrão quando uma URL não corresponde a nenhum recurso disponível.

500 (Internal Server Error): Isso significa que algo deu errado no servidor enquanto tentava processar a requisição, mas o servidor não conseguiu especificar a natureza exata do problema.

Códigos de status HTTP



Mais informações: https://www.devmedia.com.br/http-status-code/41222



Aula 01.2:

Criando a API

Criando a API

Iniciaremos agora a criação da nossa API. Após iniciar o projeto com o comando **npm init**, faça as devidas configurações no arquivo **package.json**, conforme a seguir:

```
{
  "name": "api-games",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "npx nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "module"
}

```

Criando a API

Feito isso, devemos criar o arquivo principal do projeto que recebe o nome de **index.js**. Nele iremos importar e configurar o express, bem como criar a inicialização do servidor na porta 4000.

```
1  import express from "express";
2  const app = express();
3
4  // Configurações do Express
5  app.use(express.urlencoded({ extended: false }));
6  app.use(express.json());
7
8  // Rodando a API na porta 4000
9  const port = 4000;
10 app.listen(port, (error) => {
11   if (error) {
12     console.log(error);
13   }
14   console.log(`API rodando em http://localhost:${port}.`);
15 });
16
```

Lembre-se que antes de fazer essas configurações você já deve ter instalado o **express** através do comando **npm install express**.

Aproveite também para instalar o **nodemon** através do comando **npm install nodemon**.

Por fim, inicie o servidor com o comando **npm start**. Agora a cada nova modificação no código o servidor será reiniciado automaticamente.

Testando a API

Agora iremos testar o retorno de dados da API, para isso iremos criar uma rota principal e em seguida, definiremos um array de objetos chamado **games**, e incluiremos nele alguns jogos para que sejam retornados pela API através de um JSON.

Para enviar as requisições para API é possível utilizar ferramentas como o **Insomnia** ou **Postman**. No caso do Postman é possível instalar sua extensão no VS Code e fazer os testes direto do editor de código.

```
// Configurações do Express
app.use(express.urlencoded({ extended: false }));
app.use(express.json());

app.get("/", (req, res) => {
  const games = [
    {
      title: "Game 1",
      year: "2020",
      platform: "PC",
      price: 20
    },
    {
      title: "Game 2",
      year: "2021",
      platform: "Console",
      price: 30
    }
  ];
  res.json(games);
});

// Rodando a API na porta 4000
```

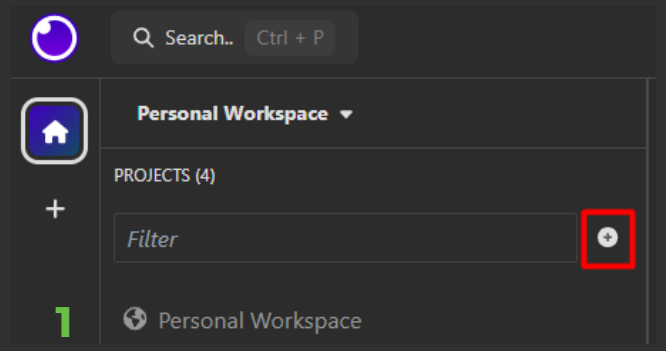
Testando a API com Insomnia

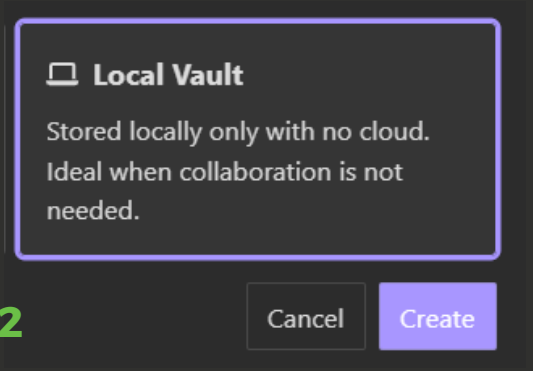
1, 2. Dentro do Insomnia deve ser criado um novo projeto que será salvo localmente.

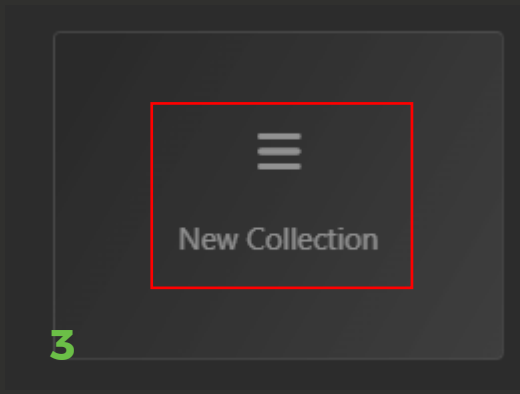
3. Após criar o projeto crie também uma nova coleção onde será organizado os testes das requisições.

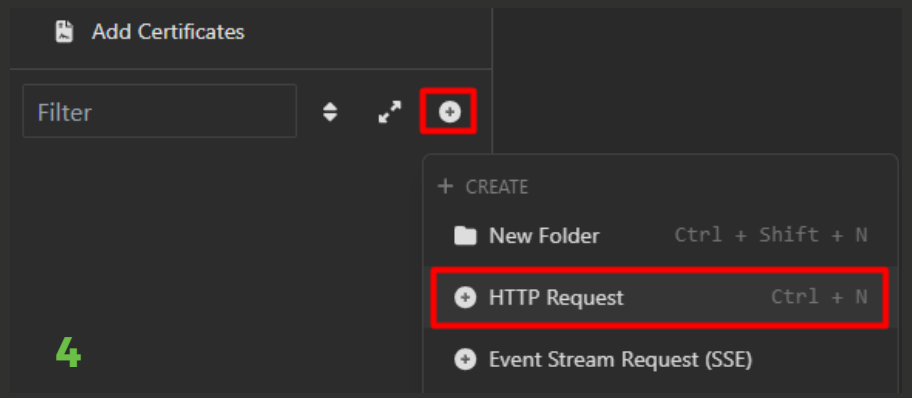
4. Dentro da coleção crie um novo teste de requisição clicando no botão de **+** ou pelo atalho **CTRL + N**.

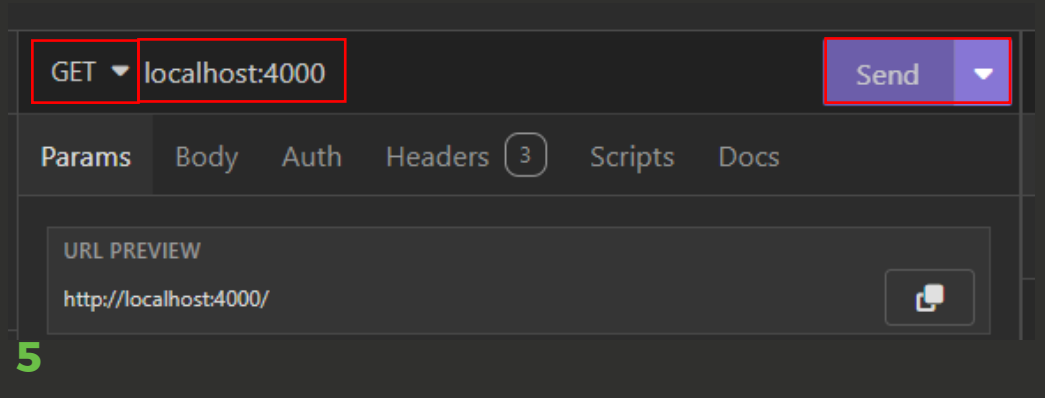
5. Defina a requisição a como **GET**, coloque o **endereço da API** e clique em **SEND**.





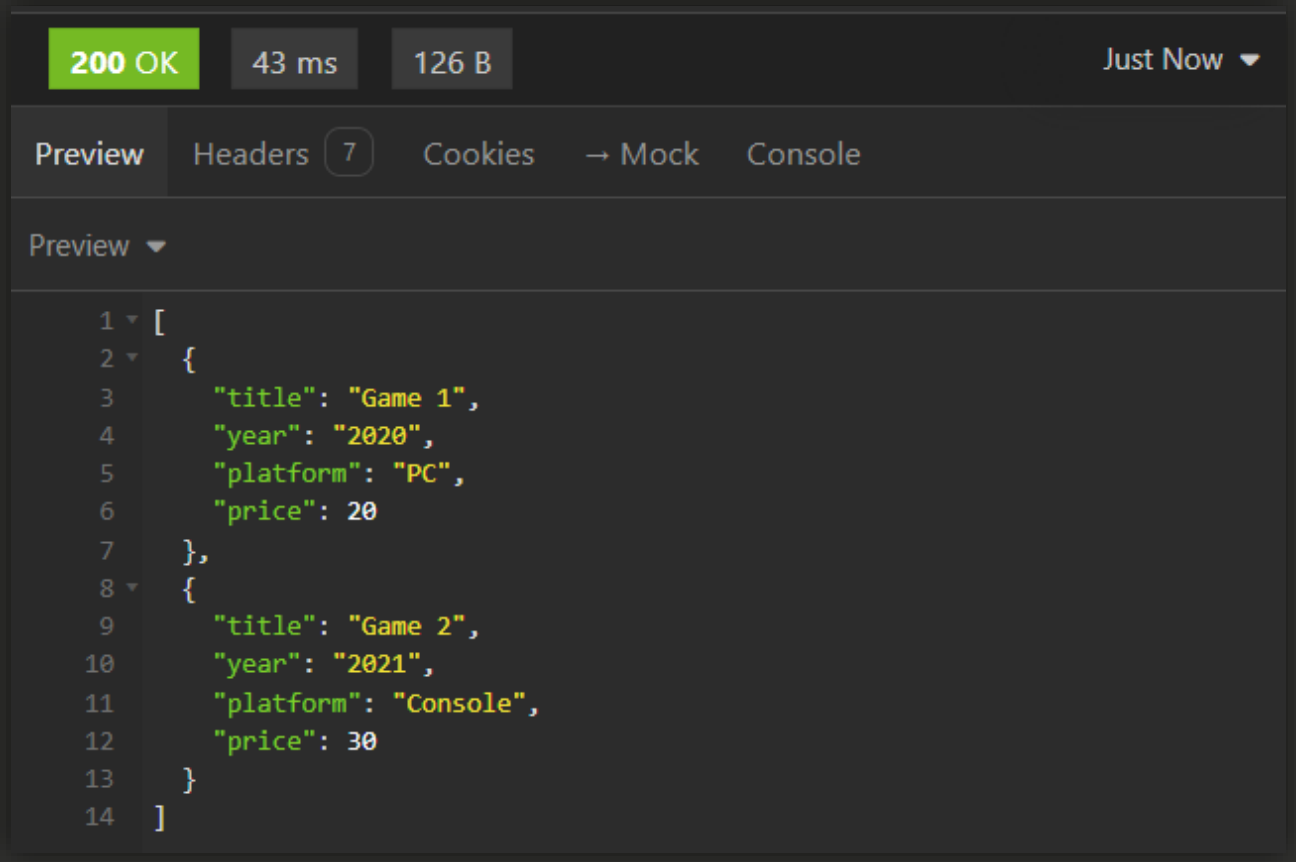






Testando a API com Insomnia

Ao enviar a requisição, deve ser retornado o código de status **200 (OK)** informando que a requisição foi bem sucedida bem como um JSON contendo a **lista de jogos**.



Feito isso, o próximo passo será conectar nossa API com o banco de dados **MongoDB**.

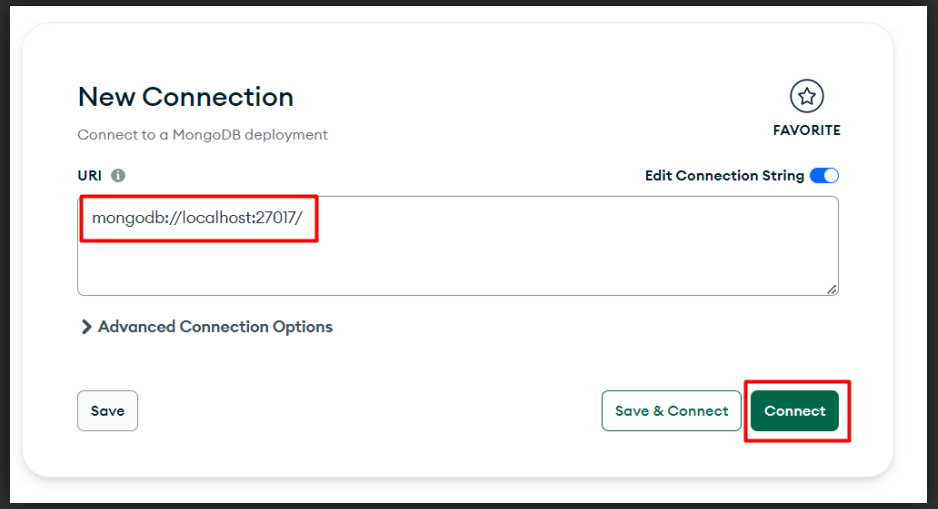
Iniciando com o MongoDB Compass

Antes de criarmos a conexão com o banco, lembre-se que o servidor do MongoDB já deve estar instalado no sistema.

Link para download do MongoDB:
https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-7.0.2-signed.msi

Com o MongoDB já instalado, iremos o utilizar a GUI (Graphical User Interface) MongoDB Compass, para nos ajudar a consultar as alterações em nosso banco.
Link para download da GUI MongoDB Compass:
https://www.mongodb.com/try/download/compass

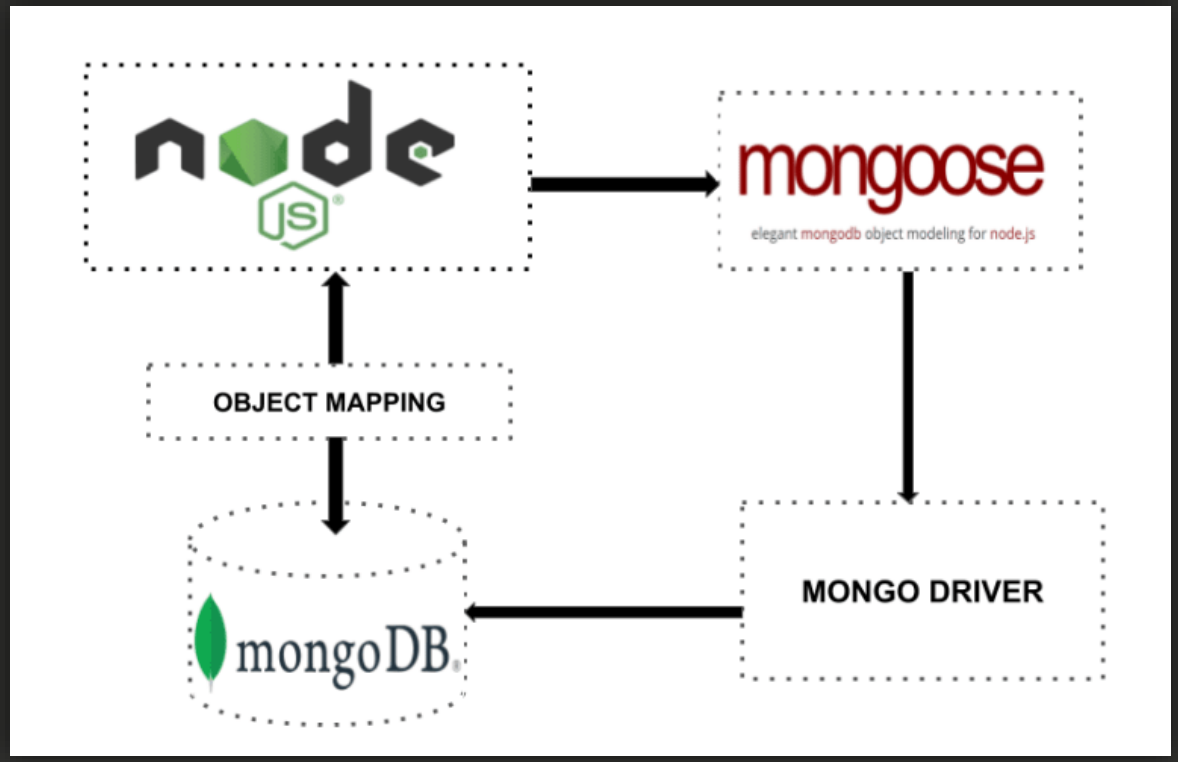
Após a instalação, basta abrir o **MongoDB Compass** e abrir um conexão:



Conhecendo a biblioteca Mongoose

Mongoose é um biblioteca de Modelagem de Dados de Objeto (ou **ODM**, do inglês: Object Data Modeling) para **MongoDB** e Node.js.

Ele gerencia o relacionamento entre dados, fornece a validação de esquemas e é usado como tradutor entre objetos no código e a representação desses objetos no MongoDB.



Mapeamento de objetos entre o Node e o MongoDB, gerenciado por meio do Mongoose.

Instalando e importando o Mongoose

Para instalar o Mongoose no seu projeto utilize o comando: **npm install mongoose**
Com o Mongoose instalado no seu projeto, faremos sua importação agora no arquivo **index.js**, conforme a seguir:

```
// Importando o Mongoose
import mongoose from "mongoose"
```

Criando a conexão com o banco de dados

Em seguida, iremos criar a conexão com o banco de dados do MongoDB, para isso, iremos incluir a seguinte linha no nosso arquivo **index.js**:

```
// Iniciando conexão com o banco de dados do MongoDB
mongoose.connect("mongodb://127.0.0.1:27017/api-thegames")
```

Criando o Model

Criaremos agora o modelo que irá representar a entidade Games em nosso sistema. Para isso, criamos uma pasta com o nome **Models** na raiz do projeto e dentro da pasta o arquivo **Games.js**.

O model terá a seguinte estrutura:

```
import mongoose from 'mongoose'

const gameSchema = new mongoose.Schema({
    title: String,
    platform: String,
    year: Number,
    price: Number
})

const Game = mongoose.model('Game',
gameSchema)

export default Game
```

1 - Importação da biblioteca Mongoose;

2 - Criação de um novo **Schema***

3 - Criação do model. Aqui informamos que deve ser criado uma coleção que receberá o nome de games quando for para o banco de dados.

4 - Eportação do módulo.

** Um Schema define a estrutura e o conteúdo dos seus dados. Schemas são a especificação do modelo de dados do seu aplicativo.*

Criando o Model

Os seguintes tipos de dados são permitidos em um Schema:

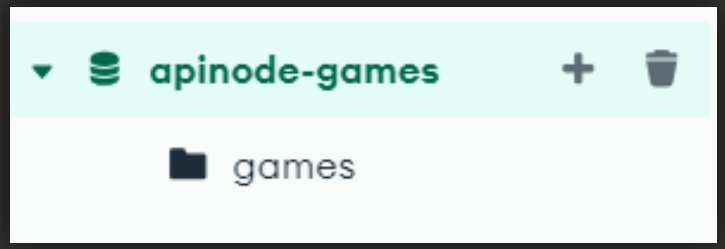
- Array
- Boolean (ou booleano, em português)
- Date (ou formato de data, em português)
- Mixed (um tipo genérico/flexível de dados)
- Number (ou numérico, em português)
- ObjectId
- String

Criação do banco de dados

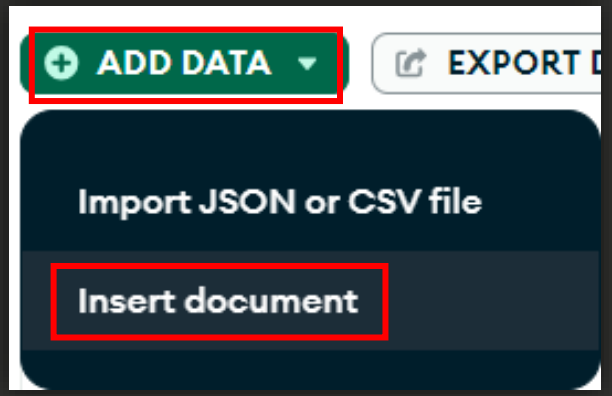
Com o model criado basta importá-lo no arquivo **index.js**:

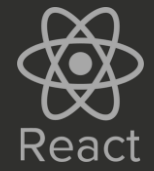
```
import Game from "./models/Games.js"
```

Agora, se necessário, rode o projeto novamente com o comando **npm start**. Feito isso o banco já deve ter sido criado. Basta abrir o MongoDB Compass e conferir.



Com o banco criado aproveite para inserir o registro de um novo jogo manualmente, com os campos “**tittle**”, “**year**”, “**price**” e “**platform**”.





Aula 01.2:

Desenvolvendo uma API Rest com Node.js