INTRODUÇÃO AO JAVASCRIPT NO BACKEND E NODE.JS

Prof. Me. Hélio Esperidião

O que é Node.js?

Node.js não é uma linguagem de programação

Você programa utilizando a linguagem JavaScript.

Possui semelhanças com linguagens compiladas, uma vez que uma máquina virtual faz etapas de pré-compilação e otimização antes do código entrar em operação.

O que é Node.js?

Node.js não é um framework Javascript

É uma aplicação na qual você escreve seus programas com Javascript, estes serão compilados, otimizados e interpretados pela **máquina virtual V8**.

A VM é a mesma que o Google utiliza para executar Javascript no Chrome, e foi a partir dela que surgiu do Node.js

Para que serve Node.js?

Permite utilizar javascript substituir java, c#, php, etc no backend de aplicações.

Node.js serve para fazer APIs.

Suporte a banco de dados

Bancos de Dados: você pode utilizar tanto com bancos relacionais quanto com não-relacionais:

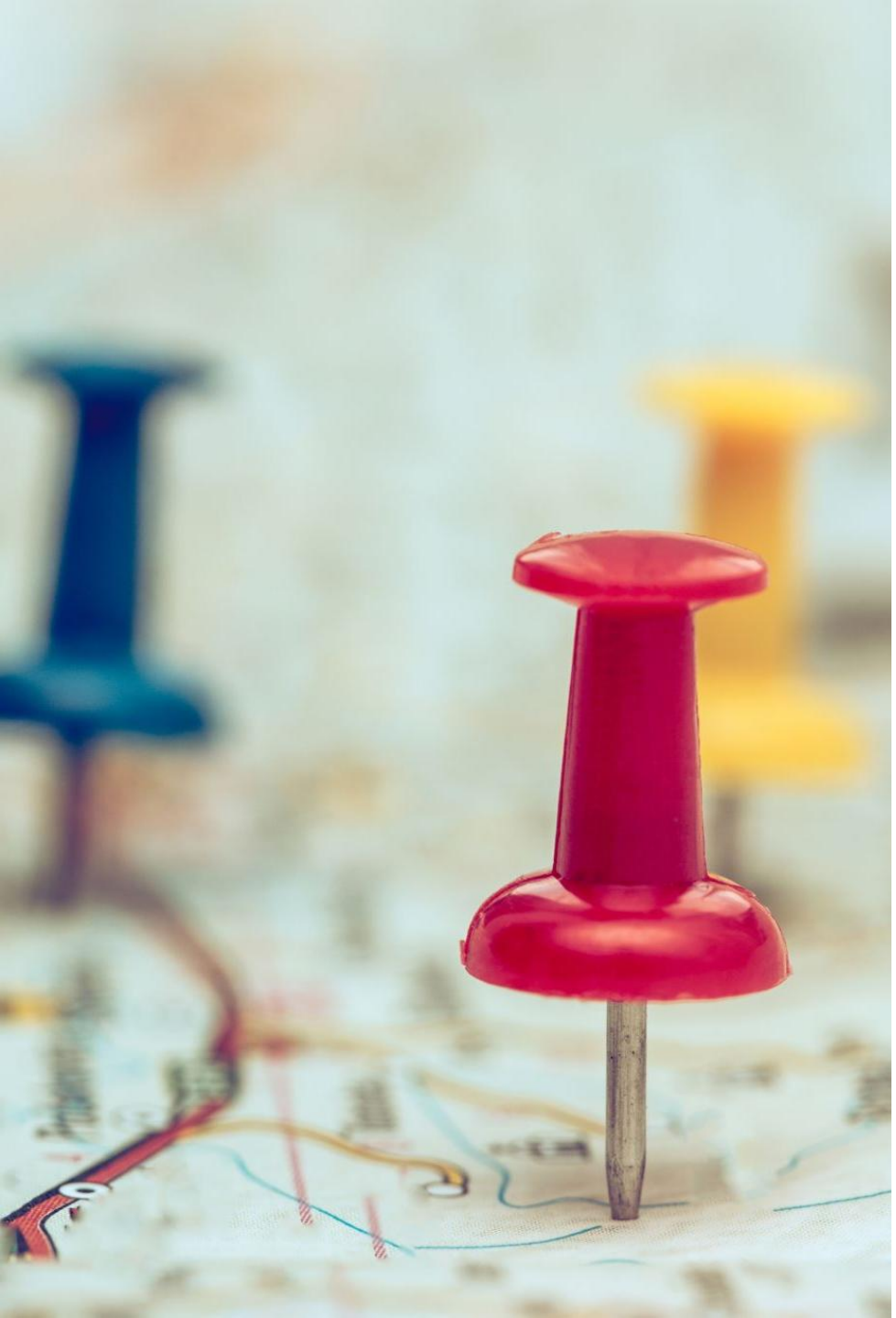
- MySQL
- PostgreSQL
- MS SQL Server
- MongoDB
- Redis
- SQLite

Quem usa Node.js?

- Netflix
- Linkedin
- Walmart
- Trello
- Uber
- PayPal
- eBay
- NASA

Quais as desvantagens do Javascript no backend?

- Não implementa orientação objeto a objeto com todos os recursos de linguagens como java, c# ou php.



Download

- https://nodejs.org/en/download/
  - Escolha uma versão LTS
  - LTS significa "Long Term Support" (Suporte de Longo Prazo).
    - É um termo comum no mundo do software, especialmente em sistemas operacionais, frameworks, e bibliotecas. Versões LTS de um software são mantidas por um período mais longo, recebendo atualizações de segurança e correções de bugs, mas geralmente sem a inclusão de novos recursos. Essas versões são preferidas em ambientes de produção, onde a estabilidade e a segurança são prioritárias.

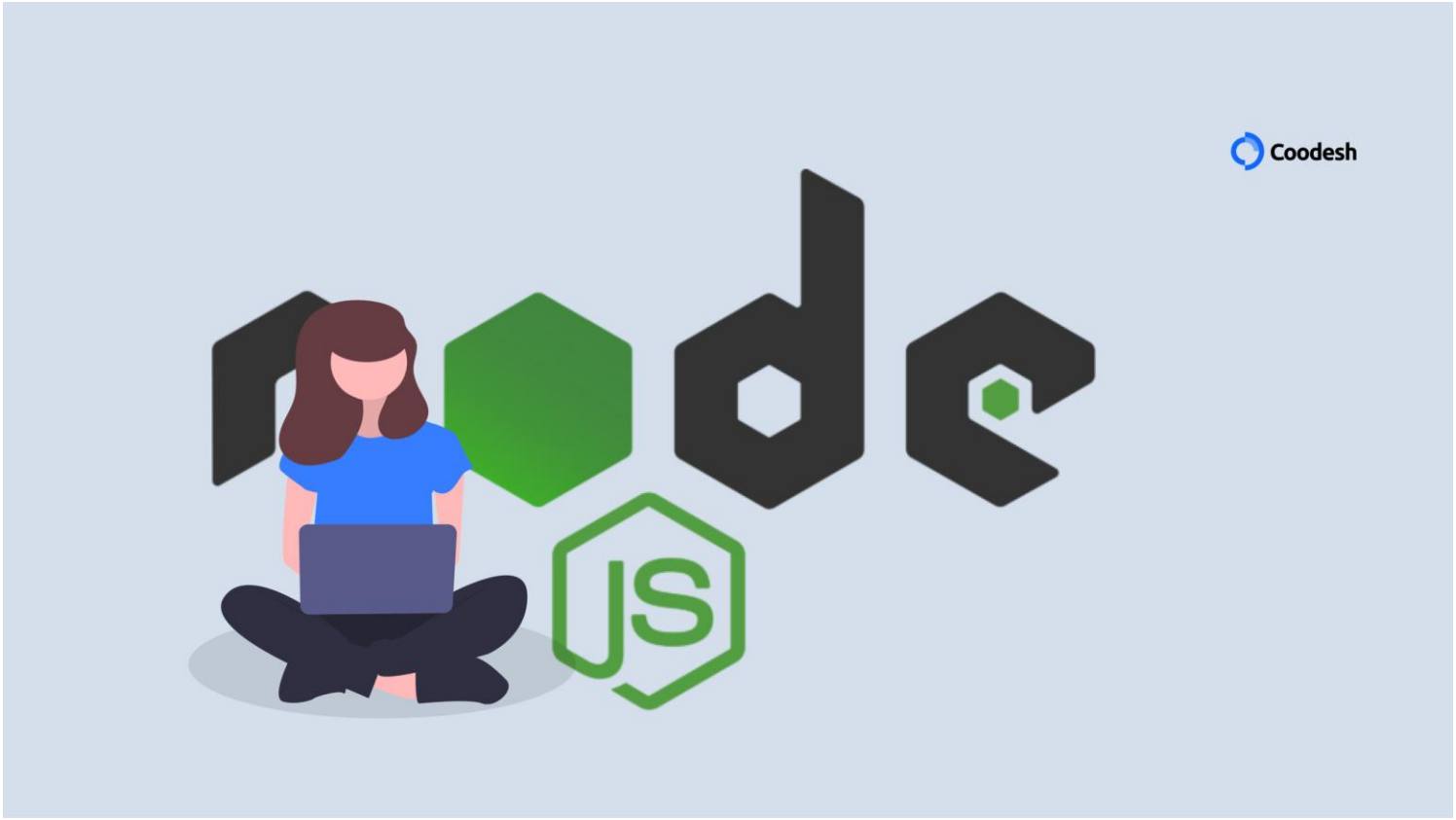
Express

- O Express é um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicações web.

Framework: Express

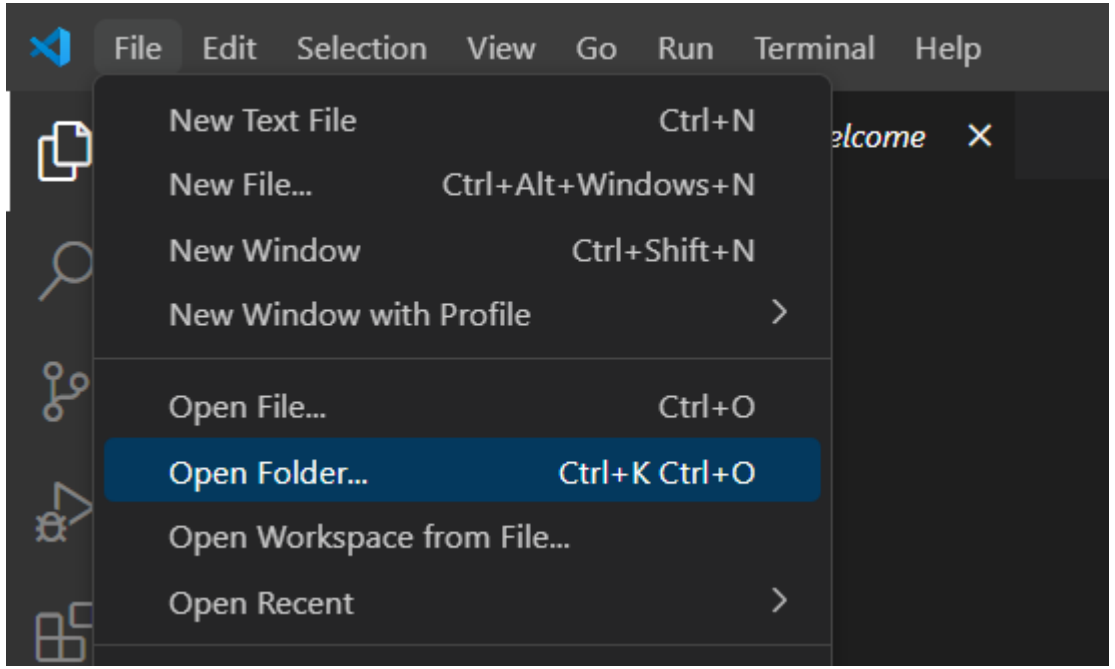
- O Express é um framework web minimalista e flexível para Node.js, projetado para criar aplicativos web e APIs de forma rápida e eficiente.
- Ele simplifica o desenvolvimento de servidores web ao fornecer um conjunto de recursos robustos e prontos para uso, sem adicionar complexidade desnecessária.
  - Rotas (Routing):
  - Middleware
  - Templates
  - Facilidade na Criação de APIs
  - Suporte a Múltiplos Protocolos

Exemplo na prática



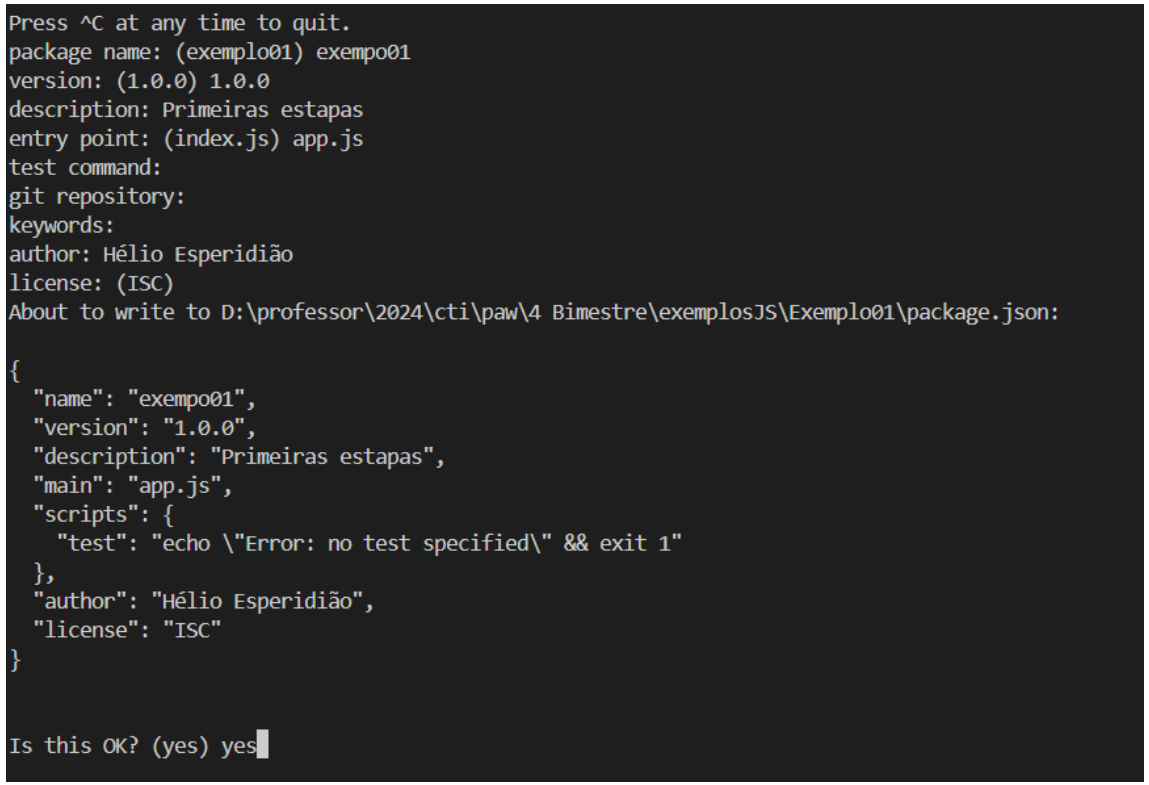
Faça a abertura de qualquer pasta no visual studio code

- Abra uma pasta vazia no visual studio code.



npm init

- Abra o terminal do visual studio code e digite: **npm init.**
- Caso o terminal do visual studio não funcione:
  - Abra o cmd do Windows
  - Digite: cd c:\local\da\pasta
  - Digite:npm init
- Configura o para rodar no node.
  - Preencha o “formulário” com as informações requisitadas



package.json

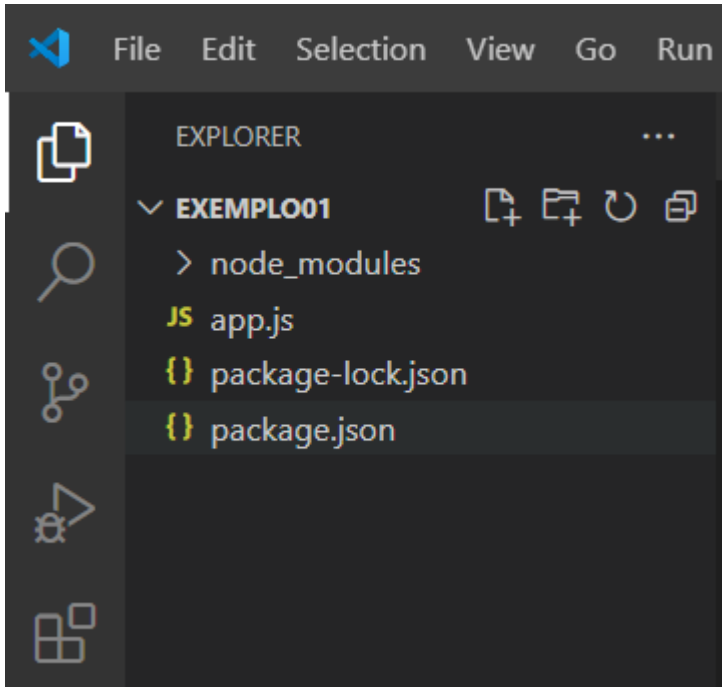
- O arquivo package.json é um arquivo fundamental em projetos Node.js.
- Ele serve como uma espécie de "manual" do seu projeto, contendo metadados sobre o projeto, como o nome, versão, autor, licença, etc
- Possui também a lista de dependências do projeto.

package.json

```
{
  "name": "restpapijsvideoaula",
  "version": "1.0.0",
  "main": "app.js",
  "directories": {
    "doc": "docs"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "bcrypt": "^6.0.0",
    "express": "^4.21.1",
    "jsonwebtoken": "^9.0.2",
    "md5": "^2.3.0",
    "mysql2": "^3.11.3"
  }
}
```

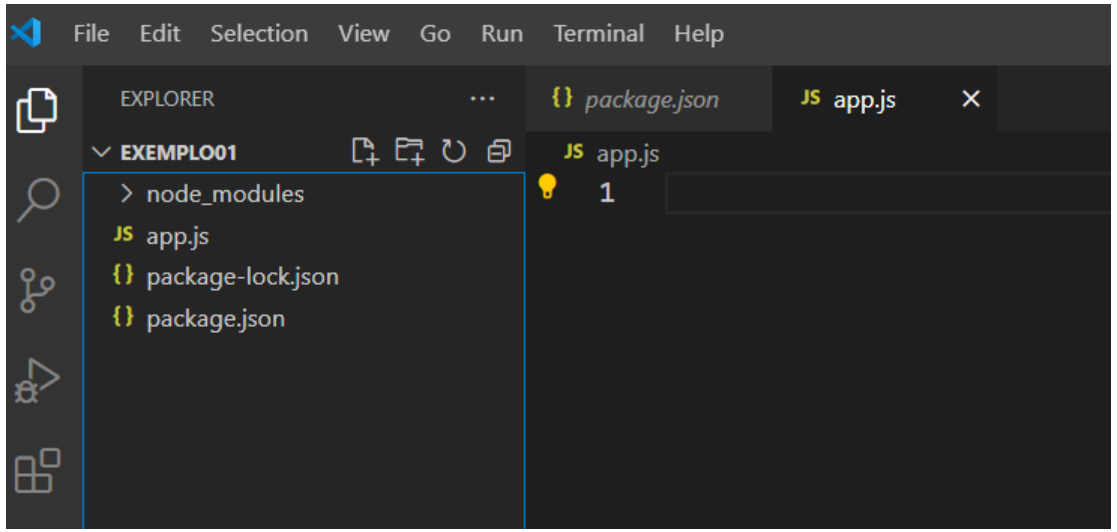
Para instalar o express

- Digite no terminal : **npm install express --save**
- Após a instalação de qualquer pacote é criada uma pasta chamada node_modules, dentro dessa pasta há tudo o que é necessário para a sua aplicação rodar.
- Como você instalou o pacote de express, dentro da pasta há todos os arquivos do express.
  - Você não precisa salvar a pasta node_modules para “transportar” o seu projeto, pois é uma pasta que ocupa muito em disco.
  - Tipicamente os programadores preferem não salvar a pasta e usar o comando npm install, que baixa e instala todos os pacotes novamente.



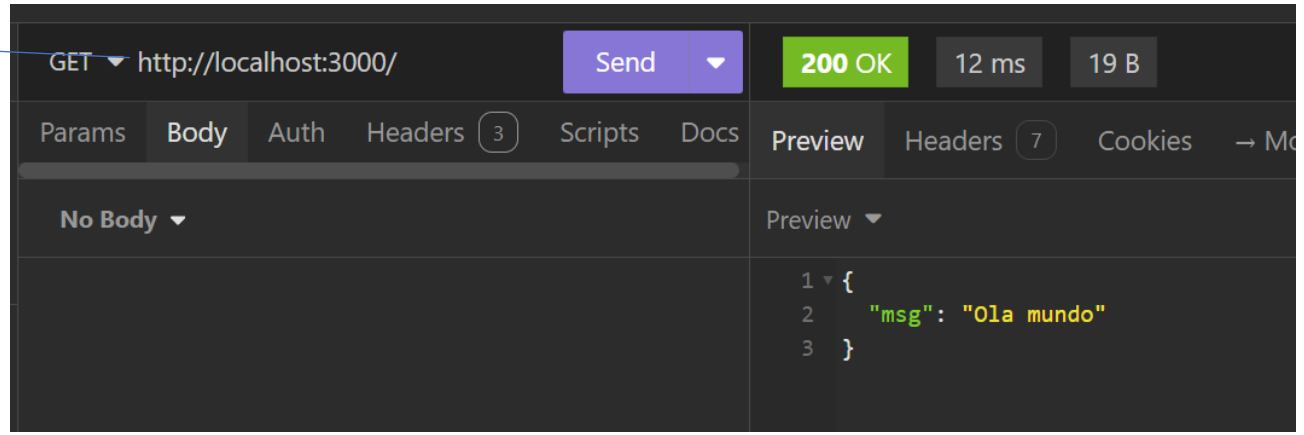
Crie o arquivo app.js

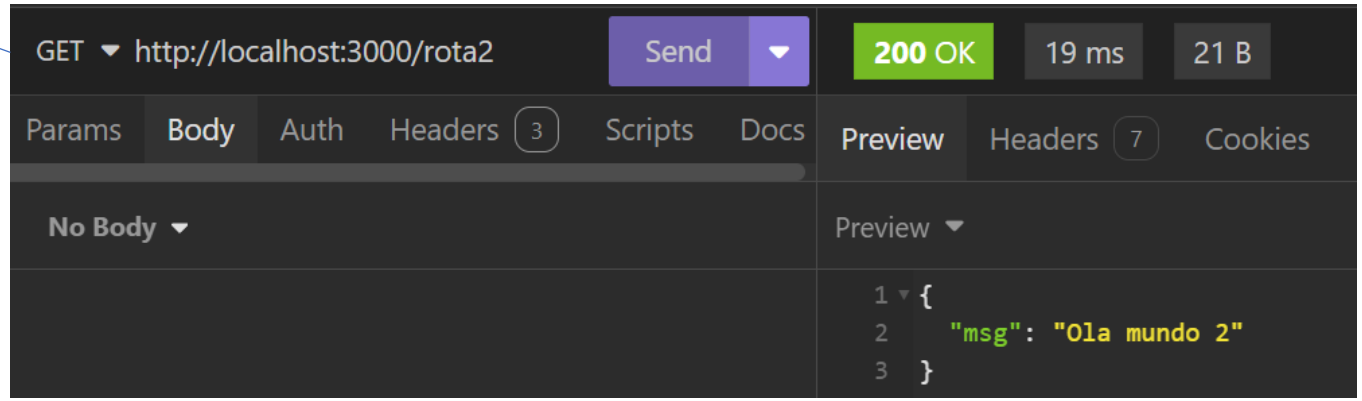
- Esse arquivo será a porta de entrada para a sua aplicação é por meio dele que serão acessados e inicializados os recursos da sua aplicação.
- O nome do arquivo em si é pouco relevante, programadores tipicamente usam index.js, app.js, main.js



app.js

```
const express = require('express'); //importa o express
const app = express();              //recupera uma instancia de express
const portaServico = 3000;
//rota: GET: /
app.get('/', (request, response) => {
    const resposta = { msg: 'Ola mundo' }
    response.status(200).send(resposta);

});

//rota: GET: /rota2
app.get('/rota2', (request, response) => {
    const resposta = { msg: 'Ola mundo 2' }
    response.status(200).send(resposta);
});

//inicia a espera por requisições http na porta 3000
app.listen(portaServico);
console.log("Api rodando no endereço: http:localhost:3000/")
```





Rodando a aplicação

- Para rodar a aplicação digite no console:
  - node app.js
- Toda vez que modificar o código é necessário parar o programa atual e reiniciar a aplicação.
  - Para parar a aplicação digite no console:
    - control+c
    - digite novamente: node app.js

Pouco produtivo

- Parar a aplicação e rodar novamente toda vez que ocorrer uma modificação no código é extremamente improdutivo.

nodemon

- O nodemon é uma biblioteca que ajuda no desenvolvimento de sistemas com o Node.js reiniciando automaticamente o servido
- Instale:
  - npm install --save-dev nodemon
- Inicie a aplicação utilizando no nodemoon:
  - npx nodemon app.js
- Sempre que houver uma modificação no código automaticamente o servidor será reiniciado.

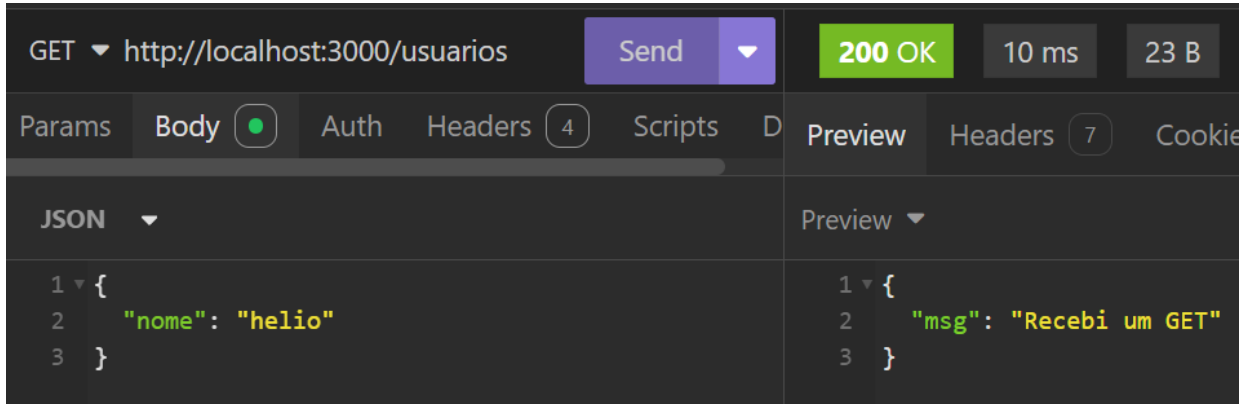
Entendendo o: request, response

```
app.get('/', (request, response) => {
 response.send('Ola mundo');
});
```

- request é um objeto que possui dados da requisição
- response é um objeto que possui dados referente a respota.
- get: verbo http pode ser substituído por: post, put, delete, etc.

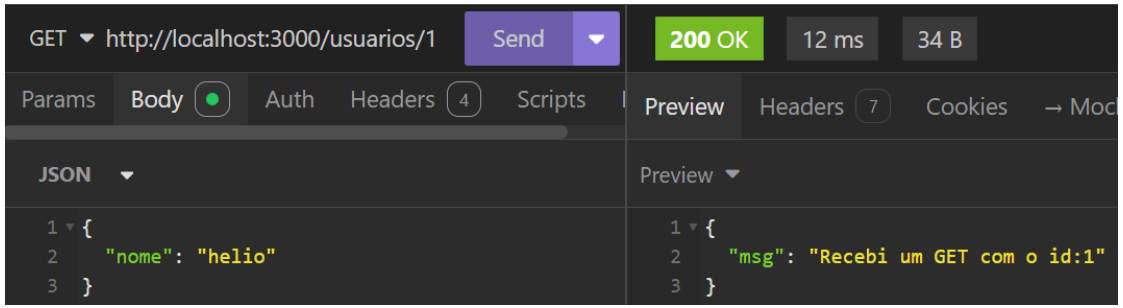
Roteamento completo: 1/6

```
const express = require('express'); // Importa o Express
const app = express();              // Recupera uma instância do Express
const portaServico = 3000;

// Middleware para habilitar o parsing de JSON no corpo das requisições
app.use(express.json());

// Rota: GET /usuarios
app.get('/usuarios', (request, response) => {
    const resposta = { msg: 'Recebi um GET' };
    response.status(200).send(resposta);
});
```



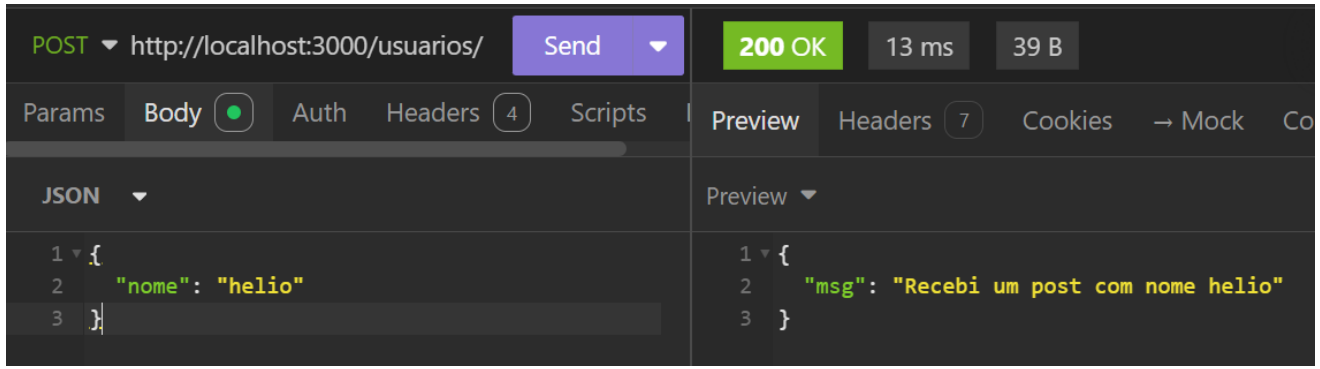
Roteamento completo: 2/6

```
// Rota: GET /usuarios/id
app.get('/usuarios/:id', (request, response) => {
    const id = request.params.id;
    const resposta = { msg: 'Recebi um GET com o id:' + id };
    response.status(200).send(resposta);
});
```



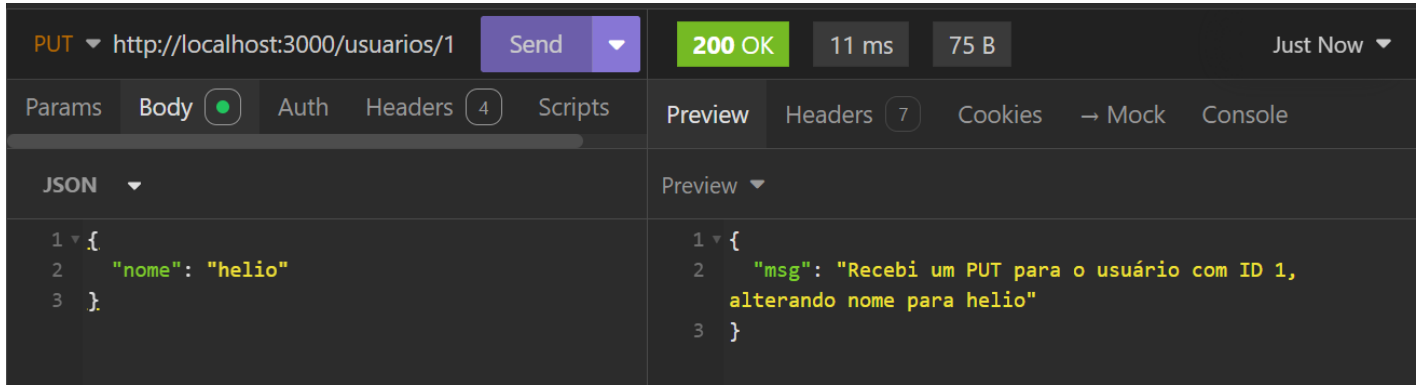
Roteamento completo: 3/6

```
// Rota: POST /usuarios
app.post('/usuarios', (request, response) => {
    const nome = request.body.nome; // Exemplo de dado enviado no corpo da
    const resposta = { msg: `Recebi um post com nome ${nome}` };
    response.status(200).send(resposta);
});
```



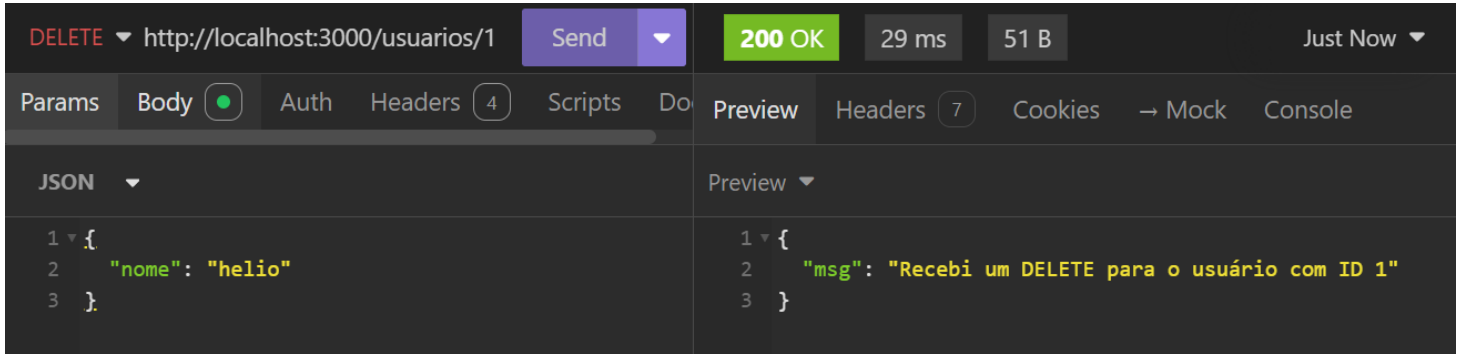
Roteamento completo: 4/6

```
// Rota: PUT /usuarios/:id
app.put('/usuarios/:id', (request, response) => {
    const id = request.params.id;
    const nome = request.body.nome; // Exemplo de dado enviado no corpo da requisição
    const resposta = { msg: `Recebi um PUT para o usuário com ID ${id}, alterando nome para ${nome}` };
    response.status(200).send(resposta);
});
```



Roteamento completo: 5/6

```
// Rota: DELETE /usuarios/:id
app.delete('/usuarios/:id', (request, response) => {
    const id = request.params.id;
    const resposta = { msg: `Recebi um DELETE para o usuário com ID ${id}` };
    response.status(200).send(resposta);
});
```



Roteamento completo: 6/6

```
// Inicia a espera por requisições HTTP
app.listen(portaServico, () => {
    console.log(`API rodando no endereço: http://localhost:${portaServico}/`);
});
```