

# Introdução ao NodeJS

Prof. Claudio Lima

# Introdução ao NojeJS

JavaScript foi criado para ser uma linguagem que funciona apenas dentro dos navegadores das páginas web. JavaScript tem um interpretador JavaScript do Chrome se chama V8.

Todo navegador tem um interpretador JavaScript que permite que código JavaScript seja executado em páginas web.

JavaScript foi inventada como a linguagem para rodar apenas em navegadores.

**NodeJS é um interpretador JavaScript, que roda fora dos navegadores.**

Hoje graça o NodeJS eu consigo rodar código JavaScript independente de um navegador.

# Por que usar NodeJS

## ➤ **Muito leve**

- O Node consegue ser mais leve do que outras tecnologias como o Java, C# e PHP.

## ➤ **Muito rápido**

- O Node é entre as dez linguagens entre as 10 tecnologias de desenvolvimento web mais rápidas que você vai desenvolver

## ➤ **Usa JavaScript**

- Uma vantagem, porque o JavaScript hoje em dia é a linguagem mais utilizada do mercado

## ➤ **Tem um dos maiores ecossistema do mundo**

- Tem muitas bibliotecas disponíveis para NodeJS para o desenvolvimento de tudo o que é tipo de coisa que você imagina para JavaScript.

## ➤ **Está usando fortemente no mercado**

- Grande e pequena empresas. Esta sendo muito utilizado por startups.

Quem usa?



Google Drive



slack

globo  
esporte  
.com

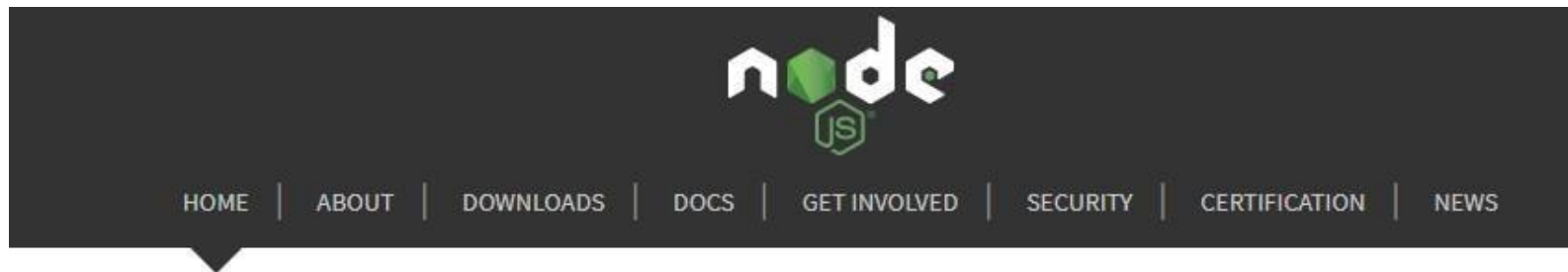
facebook.

twitter



# Como instalar o NodeJS

O primeiro passo é acessar seu site oficial do Node.js: <http://nodejs.org>.



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

## Download for Windows (x64)



**14.17.3 LTS**

Recommended For Most Users

**16.5.0 Current**

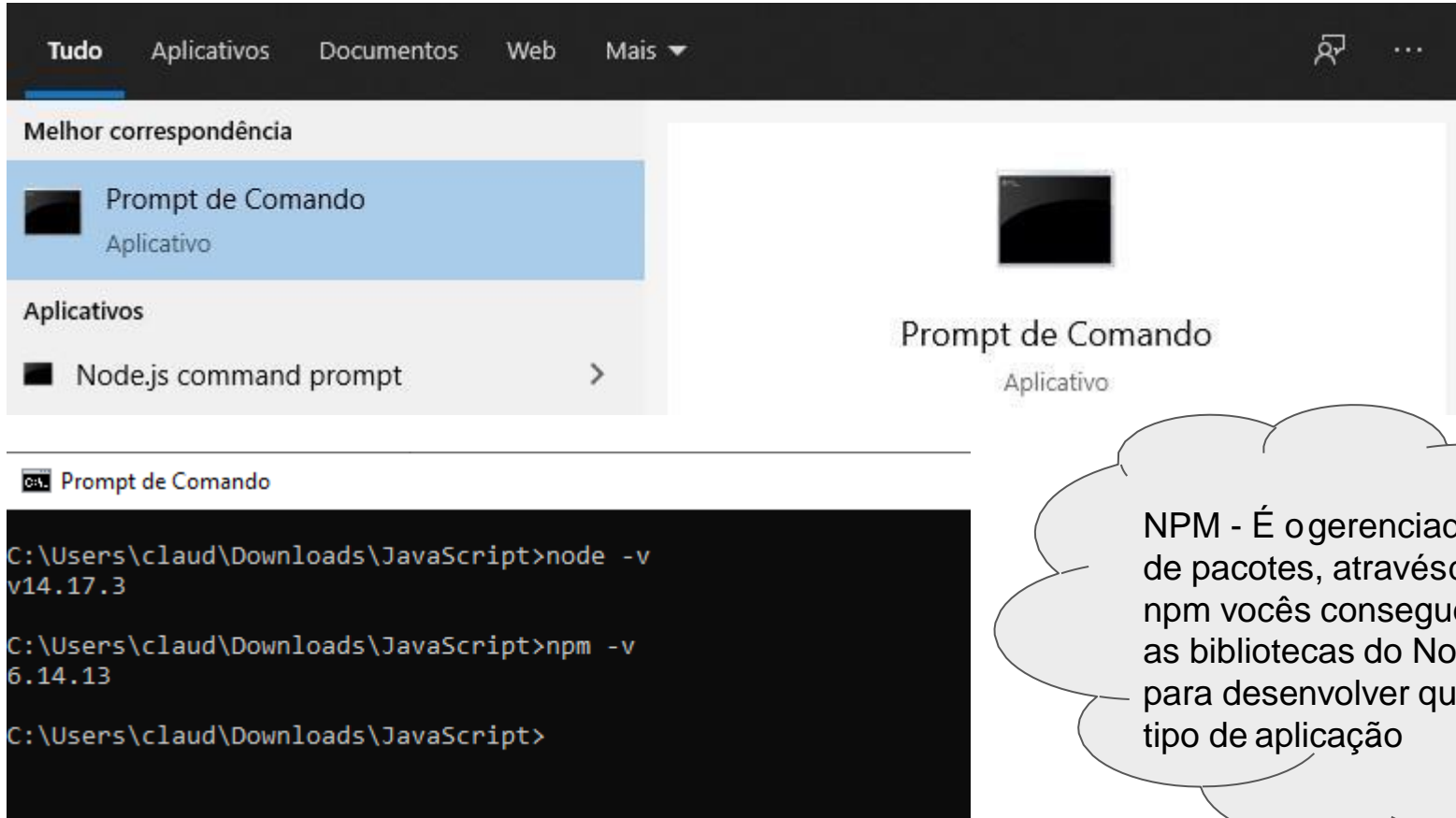
Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#).

# Testar o NodeJS



The image shows a Windows Start menu search interface. The search bar at the top contains the text 'Prompt de Comando'. Below the search bar, the results are categorized under 'Melhor correspondência' (Best match) and 'Aplicativos' (Apps). Under 'Melhor correspondência', there is a result for 'Prompt de Comando' (Application). Under 'Aplicativos', there is a result for 'Node.js command prompt' with a right arrow. To the right of the search results, there is a preview of the 'Prompt de Comando' application. Below the search results, there is a terminal window titled 'C:\> Prompt de Comando'. The terminal shows the following commands and output:

```
C:\Users\claud\Downloads\JavaScript>node -v
v14.17.3

C:\Users\claud\Downloads\JavaScript>npm -v
6.14.13

C:\Users\claud\Downloads\JavaScript>
```

NPM - É o gerenciador de pacotes, através do npm você consegue baixar as bibliotecas do Node para desenvolver qualquer tipo de aplicação

# Principais comandos do NPM

Utilizar o NPM é muito fácil. Suas utilidades vão além de um simples gerenciador de dependência, pois ele permite também que você crie comandos de automatização de tarefas para seus projetos que são declarados, por meio do arquivo package.json. A seguir, veja os principais comandos e seus respectivos significados:

**npm init:** exibe um miniquestionário para auxiliar na criação e descrição do package.json do seu projeto

**npm install nome\_do\_módulo:** instala um módulo no projeto;

**npm install -g nome\_do\_módulo:** instala um módulo global;

**npm install nome\_do\_módulo --save:** instala o módulo e adiciona-o no arquivo package.json, dentro do atributo "dependencies";

**npm list:** lista todos os módulos que foram instalados no projeto;

**npm list -g:** lista todos os módulos globais que foram instalados;

**npm remove nome\_do\_módulo:** desinstala um módulo do projeto;

**npm remove -g nome\_do\_módulo:** desinstala um módulo global;

**npm remove nome\_do\_módulo --save:** desinstala um módulo do projeto, removendo também do atributo "dependencies" do package.json;

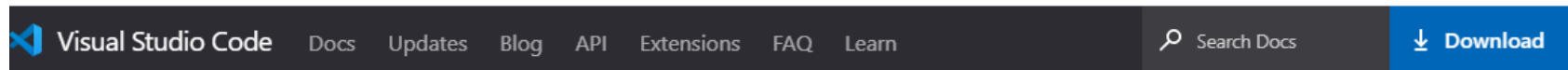
**npm update nome\_do\_módulo:** atualiza a versão de um módulo do projeto;

**npm update -g nome\_do\_módulo:** atualiza a versão de um módulo global;

**npm -v:** exibe a versão atual do NPM

# Recomendação de editor de código

<https://code.visualstudio.com/download>



## Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.



↓ Windows

Windows 7, 8, 10

User Installer [64 bit](#) [32 bit](#) [ARM](#)  
System Installer [64 bit](#) [32 bit](#) [ARM](#)



↓ .deb

Debian, Ubuntu

↓ .rpm

Red Hat, Fedora, SUSE

.deb [64 bit](#) [ARM](#) [ARM 64](#)  
.rpm [64 bit](#) [ARM](#) [ARM 64](#)



↓ Mac

macOS 10.11+

.zip [Universal](#) [Intel Chip](#) [Apple Silicon](#)



# Primeiro código com NodeJs

Criei o meu primeiro arquivo node. Um arquivo Node é simplesmente um arquivo JavaScript.

<> index.html

JS meu\_script.js X

Settings

C: > Users > claud > Downloads > JavaScript > JS meu\_script.js

```
1 console.log('Primeiro projeto Node')
```

# Executando arquivos JavaScript no NodeJS

 Prompt de Comando

```
C:\Users\claud\Downloads\JavaScript>node meu_script.js  
Primeiro projeto Node
```

# O que é HTTP?

É um protocolo de transferência de dados na web. HTTP é muito importante que entenda ele para que consigamos criar aplicações Web mais robustas, mais completas e consigamos melhor entender a estrutura do Node para Web.

# Primeira aplicação HTTP com Node.JS

Vamos criar nossa primeira aplicação Web que iremos acessar no nosso navegador utilizando JavaScript.

Tudo isso graças a um módulo há uma biblioteca que já vem no Node chamado **HTTP**.

O Node traz várias bibliotecas e uma das biblioteca e a biblioteca HTTP.

# Primeira aplicação HTTP com Node.JS

Importar uma biblioteca chamada `http`, e a partir desta biblioteca podemos criar um servidor, e com isso passar a escutar requisições que são feitas em uma porta específica.

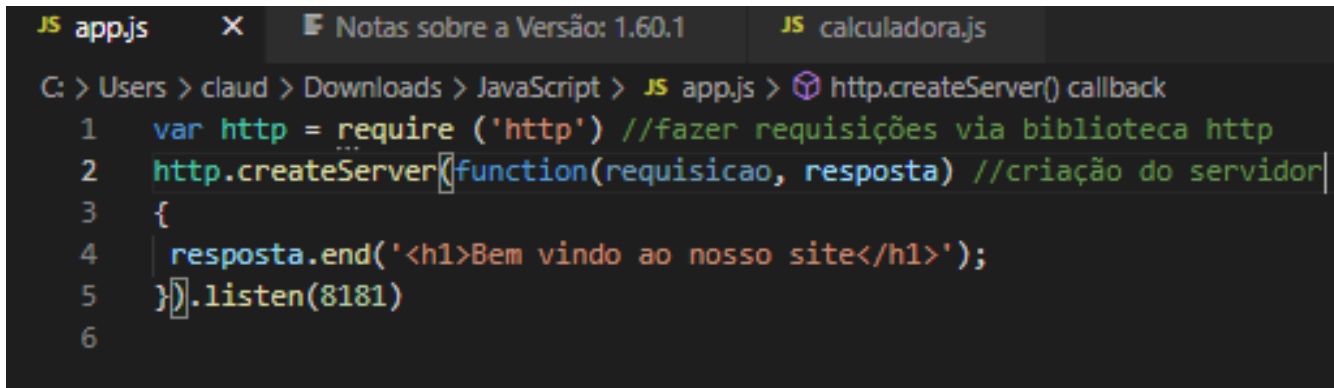
Quando alguém (algun navegador) disparar uma requisição naquela porta, podemos recuperar essa requisição e fornecer uma resposta.

# Primeira aplicação HTTP com Node.JS

**Primeiro passo:** criar uma variável chamada `http` que vai receber o `require` da biblioteca `http`:

```
var http = require('http');
```

Nesse caso estamos incorporando a biblioteca `http`. Em seguida, precisamos criar (subir) um servidor no Node nós vamos utilizar bastante funções como argumentos.



The screenshot shows a code editor with two tabs: `app.js` and `calculadora.js`. The `app.js` tab is active, showing the following code:

```
C: > Users > claud > Downloads > JavaScript > JS app.js > http.createServer() callback
1  var http = require('http') //fazer requisições via biblioteca http
2  http.createServer(function(requisicao, resposta) //criação do servidor
3  {
4    resposta.end('<h1>Bem vindo ao nosso site</h1>');
5  }).listen(8181)
6
```

# Respondendo requisições HTTP com NodeJS



**Bem vindo ao nosso site**

# Respondendo requisições HTTP com NodeJS

**Linha1** – criamos uma variável http recebendo a biblioteca http.

**Linha 2** – criamos a variável server que vai receber o método createServer da variável http, Esse método tem como parâmetro do uma função com dois parâmetros: requisicao e resposta.

A função requisicao vai receber a requisição do cliente (browser)

A função resposta vai fornecer a resposta.

No final, informamos ao servidor qual é a porta que ele está escutando. No caso utilizaremos aqui a porta 8181

O método server.listen(8181) informa que estará escutando a porta 8181.



# O que é o Express.js e NPM

**Express.js** é um framework para desenvolvimento Web Backend com o Node.js.

Framework é basicamente uma **super biblioteca** que ajuda bastante a fazer um determinada tarefa.

Express é uma super biblioteca que nos ajuda fortemente a construir aplicações.

Ele traz várias ferramentas que podem ser utilizadas em nossos códigos para desenvolver aplicações Web robustas e completas.

**NPM** - Gerenciador de pacotes do Node.js, serve para criar projetos Node e para baixar biblioteca de mais diversos tipos de projetos. Sempre que instala o Node o NPM vem junto.

**npm - v** - verificar a versão do npm

# O que é o Express.js e NPM

## npm init - iniciar um projeto Node.js

Precisa iniciar um projeto para ficar mais fácil gerenciar a instalação de biblioteca.  
Porque quando inicia um projeto NPM é criado um arquivo **package.json**. Esse arquivo fala tudo do seu projeto.

```
C: > Users > claud > OneDrive > Área de Trabalho > express > {} package.json > ...
1  {
2    "name": "express",
3    "version": "1.0.0",
4    "description": "Iniciando com express",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC"
11 }
```

```
C:\Users\claud\OneDrive\Área de Trabalho\express>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: (express)
```

```
version: (1.0.0)
```

```
description: Iniciando com express
```

```
entry point: (index.js)
```

```
test command:
```

```
git repository:
```

```
keywords:
```

```
author:
```

```
license: (ISC)
```

```
About to write to C:\Users\claud\OneDrive\Área de Trabalho\express\package.json:
```

```
{
  "name": "express",
  "version": "1.0.0",
  "description": "Iniciando com express",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

```
Is this OK? (yes)
```

```
C:\Users\claud\OneDrive\Área de Trabalho\express>
```

# O que é o Express.js e NPM

Agora que iniciamos um projeto NPM, um projeto Node.js, iremos instalar o express.

No site oficial do express <https://expressjs.com/pt-br/> você pode ver que tem um comando de instalação



# O que é o Express.js e NPM

```
C:\Users\claud\OneDrive\Área de Trabalho\express>npm install express --save
```

Esse --save é o comando de instalação do NPM que vai salvar os arquivos do Express na sua pasta do seu projeto.

# O que é o Express.js e NPM

Vamos rodar - `npm install express --save`

```
C:\Users\claud\OneDrive\Área de Trabalho\express>npm install express --save
npm ERR! code ENOSELF
npm ERR! Refusing to install package with name "express" under a package
npm ERR! also called "express". Did you name your project the same
npm ERR! as the dependency you're installing?
npm ERR!
npm ERR! For more information, see:
npm ERR!   <https://docs.npmjs.com/cli/install#limitations-of-npms-install-algorithm>
npm ERR! A complete log of this run can be found in:
npm ERR!   C:\Users\claud\AppData\Roaming\npm-cache\_logs\2021-09-20T19_14_01_158Z-debug.log
C:\Users\claud\OneDrive\Área de Trabalho\express>
```

Gerou um erro proposital durante a instalação de alguns pacotes de desempenho.

Ele está recusando a instalar o express no nosso projeto porque o nosso projeto já tem um nome dele. Por que o nosso projeto já tem um nome express.

# O que é o Express.js e NPM

```
C: > Users > claud > OneDrive > Área de Trabalho > express > {} package.json > n
1  {
2    "name": "iniciando",
3    "version": "1.0.0",
4    "description": "Iniciando com express",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC"
11 }
```

Mudar o nome

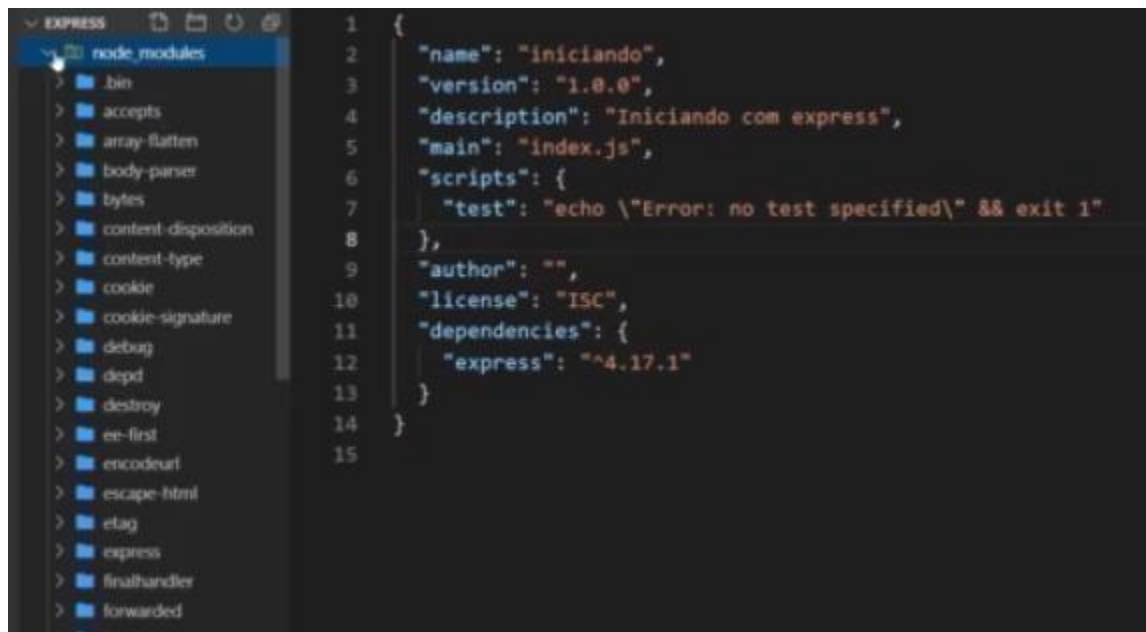
```
C:\Users\claud\OneDrive\Área de Trabalho\express>npm install express --save
```

```
C:\Users\claud\OneDrive\Área de Trabalho\express>npm install express --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN iniciando@1.0.0 No repository field.
```

```
+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 6.118s
found 0 vulnerabilities
```

```
C:\Users\claud\OneDrive\Área de Trabalho\express>
```

# O que é o Express.js e NPM



Podemos dizer que esse package.json é a identidade, ele é o RG do projeto. .Ele fala tudo do seu projeto.

Veja que criou uma pasta, essa pasta é muito comum em projetos Node, aonde fica instalado todas as biblioteca do seu projeto

# Estrutura Inicial

Vamos iniciar um projeto express. Primeira coisa que a gente precisamos fazer para iniciar um projeto express é criar um arquivo inicial chamado index.js



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure under 'SEM TÍTULO (WORKSPACE)':

- express
  - node\_modules
  - index.js (selected)
  - package-lock.json
  - package.json

The main editor area shows the content of index.js with the following code:

```
express > JS index.js > app.listen() callback
1  const express = require("express") // importando o express
2  const app = express() // iniciando o express e passando para a variavel app
3  app.listen(3000, function(err){
4    if (err){
5      console.log ('Ocorreu um erro')
6    }else {
7      console.log ('Servidor iniciado com sucesso')
8    }
9  })
10 }
```

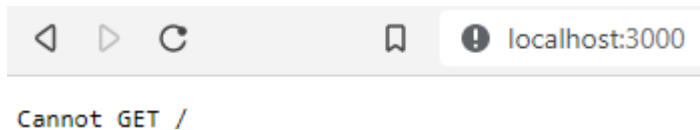
Isso é o suficiente para abrir um servidor express.

```
C:\Users\claud\OneDrive\Área de Trabalho\express>node index.js
Servidor iniciado com sucesso
```



# Rotas

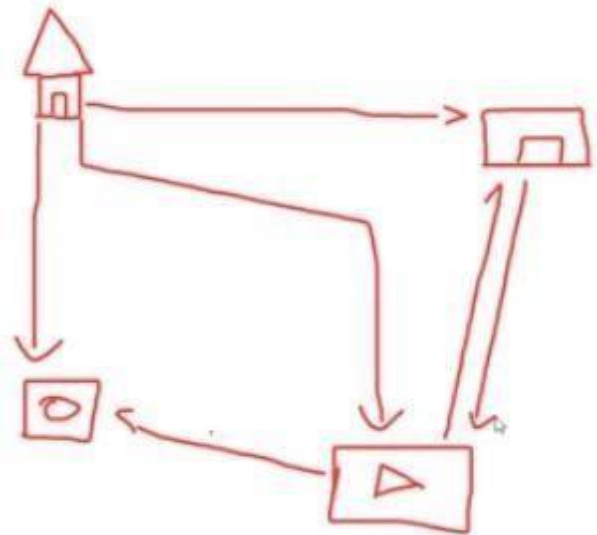
No último slide iniciamos a nossa aplicação express. Quando nós iniciamos a nossa aplicação através do endereço localhost:3000 apareceu um erro



Esse erro acontece porque não existe rota

Para resolver esse erro é necessário usar rotas.

Vamos trazer a rota para a programação



# Rotas

```
JS app.js  {} package.json  JS index.js  X  JS calculadora.js

express > JS index.js > app.get("/") callback
1  const express = require("express") // importando o express
2  const app = express() // iniciando o express e passando para a variavel app
3
4  app.get("/",function(requisicao, resposta){ // forma mais simples de criar um rota
5  |  resposta.send("Bem Vindo ao Express")
6  |  })
7
8  app.listen(3000, function(erro){
9  |  if (erro){
10 |      console.log ('Ocorreu um erro')
11 |  }else {
12 |      console.log ('Servidor iniciado com sucesso')
13 |  }
14
15  })
```

Toda rota que for criar precisa devolver uma resposta. Nesse caso estou devolvendo num formato de texto. Essa resposta pode ser uma página HTML, um arquivo...

# Outras Rotas


JS app.js

{ } package.json

JS index.js

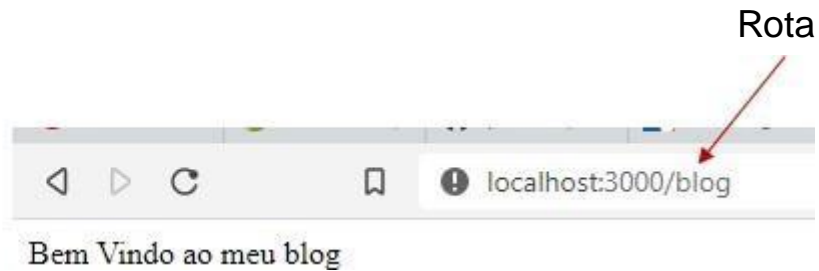
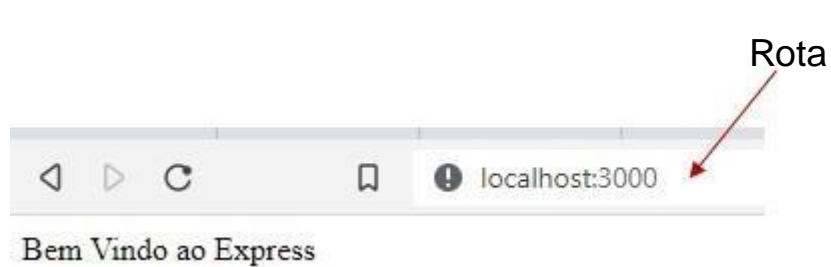
X

JS calculadora.js

express > JS index.js >  app.listen() callback

```
1  const express = require("express") // importando o express
2  const app = express() // iniciando o express e passando para a variavel app
3
4  app.get("/",function(requisicao, resposta){ // forma mais simples de criar um rota
5    resposta.send("Bem Vindo ao Express")
6  })
7
8  app.get("/blog",function(requisicao, resposta){
9    resposta.send("Bem Vindo ao meu blog")
10  })
11
12  app.listen(3000, function(erro){
13    if (erro){
14      console.log ('Ocorreu um erro')
15    }else {
16      console.log ('Servidor iniciado com sucesso')
17    }
18  }
19  })
```

# Outras Rotas



# Como instalar o Nodemon no Node.js para o servidor ser recarregado automaticamente

Toda vez que fazemos uma modificação no nosso projeto nós tínhamos que interromper o nosso servidor e iniciar novamente. Isso em uma aplicação real não acontece. E como fazer isso?

Para resolver esse tipo de problema para a nossa aplicação precisamos trabalhar com um módulo do Node chamado Nodemon. O Nodemon é um módulo responsável por iniciar o seu servidor, reiniciar a sua aplicação sempre que ele detectar alguma modificação. Como instalar:

**npm install nodemon -g** ( -g, significa que você quer instalar globalmente no seu computador)

# Carregando automático

```
C:\Users\claud\OneDrive\Área de Trabalho\express>npm install nodemon -g
C:\Users\claud\AppData\Roaming\npm\nodemon -> C:\Users\claud\AppData\Roaming\npm\node_modules\nodemon\bin\nodemon.js

> nodemon@2.0.12 postinstall C:\Users\claud\AppData\Roaming\npm\node_modules\nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
  > https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.3.2 (node_modules\nodemon\node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ nodemon@2.0.12
added 119 packages from 53 contributors in 16.846s

C:\Users\claud\OneDrive\Área de Trabalho\express>nodemon index.js
[nodemon] 2.0.12
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Servidor iniciado com sucesso
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Servidor iniciado com sucesso
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
```

# Exibindo HTML

Como exibir arquivos html nas suas rotas. Porque até agora estamos apenas exibindo textos.

Vamos primeiro criar uma pasta para deixar os arquivos html



The screenshot shows the Visual Studio Code interface. On the left, the 'EXPLORADOR' (Explorer) sidebar is open, showing a project structure for an 'EXPRESS' application. A folder named 'html' has been created under the project root. Inside the 'html' folder, a file named 'index.html' is selected. The main editor area shows the content of 'index.html'. The breadcrumb navigation at the top of the editor indicates the path: 'html > index.html > html > head > meta'. The code in the editor is a basic HTML5 boilerplate with a title 'Document'.

```
html > index.html > html > head > meta
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8  </head>
9  <body>
10     |
11 </body>
12 </html>
```

# Views HTML

```
JS app.js > ...
1  const express = require("express");
2
3  const app = express();
4
5  app.get("/", function(req, res){
6    |   res.sendFile(__dirname + "/src/index.html");
7  });
8
9  app.get("/sobre-empresa", function(req, res){
10   |   res.sendFile(__dirname + "/src/sobre-empresa.html");
11 });
12
13 app.get("/blog", function(req, res){
14   |   res.send("Pagina do blog");
15 });
16
17 app.get("/contato", function(req, res){
18   |   res.send("Pagina de contato");
19 });
20
21 //localhost:8080
22 app.listen(8080);|
```

A variável `dirname` vai pegar o caminho que está a pasta e vai concatenar com mais o caminho do meu html.



# Utilizando template engine EJS com Node.js

O EJS (Embedded JavaScript Templating) é uma Template Engine que podemos utilizar com Node.js.

Chamamos de template engine, que é um motor de templates que tem a função de desenhar HTML

Existem diversas outras bibliotecas para desenhar HTML para exibir HTML no Node.

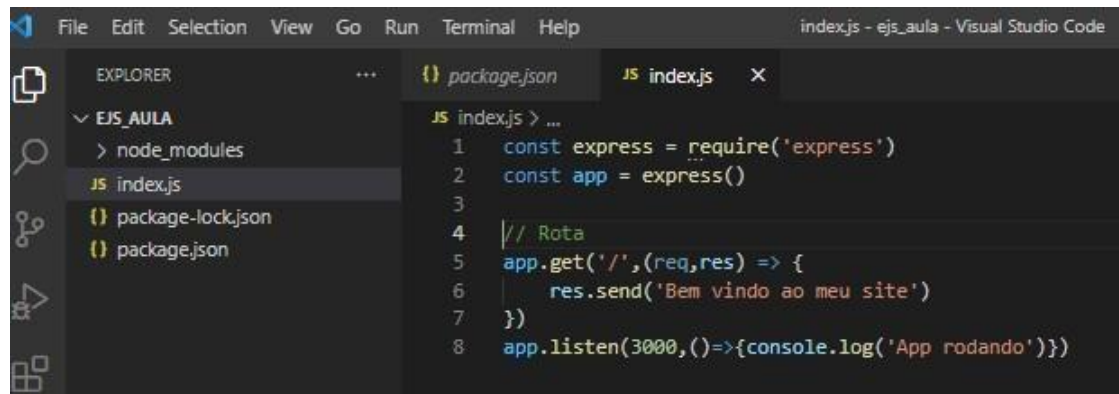
EJS é o melhor, porque é simples e tem todas as funcionalidades que precisamos e permite usar código em JavaScript dentro do HTML

1º Para Iniciar um novo projeto node temos que digitar => **npm init;**

2º Instalar o express => **npm install express --save;**

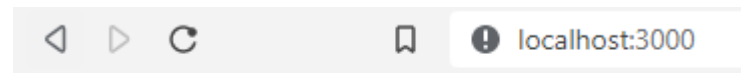
3º Instalar o EJS => **npm install ejs --save**

# Configurando e exibindo HTML com EJS



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor in the center. The file explorer shows a project named 'EJS\_AULA' with a 'node\_modules' directory and two files: 'package-lock.json' and 'package.json'. The code editor shows the 'index.js' file with the following code:

```
index.js - ej_s_aula - Visual Studio Code  
1 const express = require('express')  
2 const app = express()  
3  
4 // Rota  
5 app.get('/', (req, res) => {  
6   res.send('Bem vindo ao meu site')  
7 })  
8 app.listen(3000, () => { console.log('App rodando') })
```



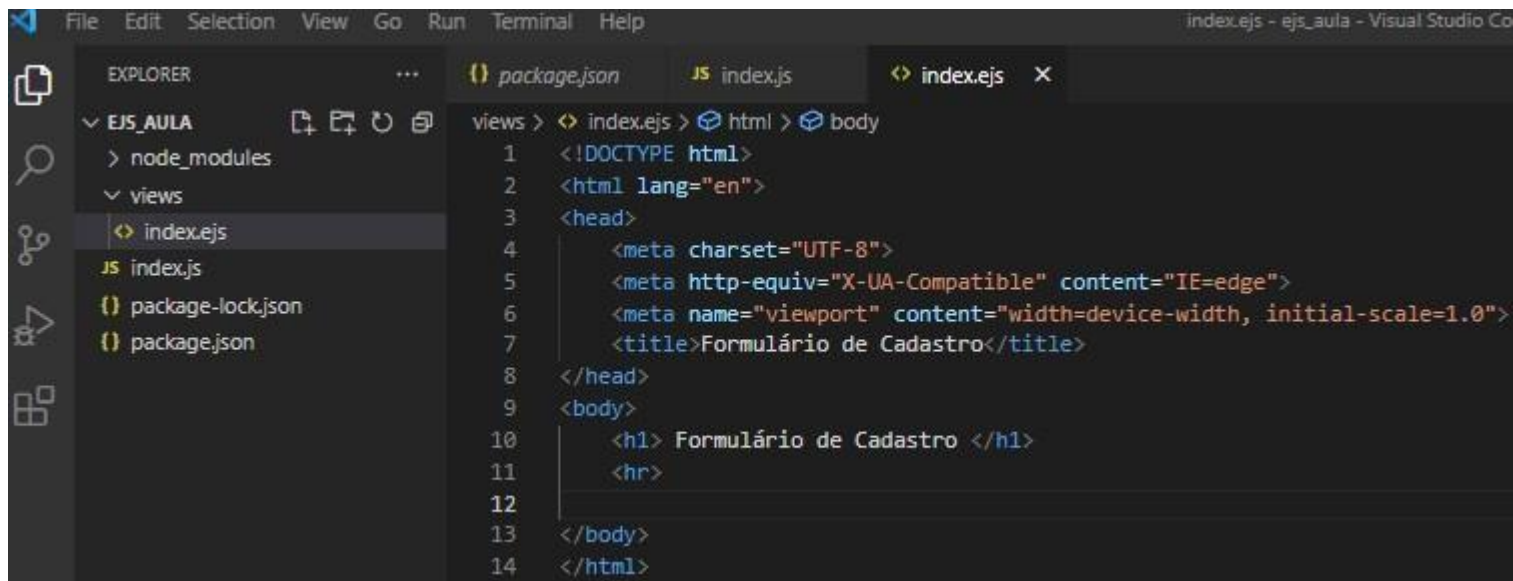
Bem vindo ao meu site

Quero que ele exiba HTML e é por isso que a gente vai aprender a configurar o EJS no express.

# Configurando e exibindo HTML com EJS

1º Criar uma pasta chamado **Views**

2º Criar um arquivo chamado index.ejs



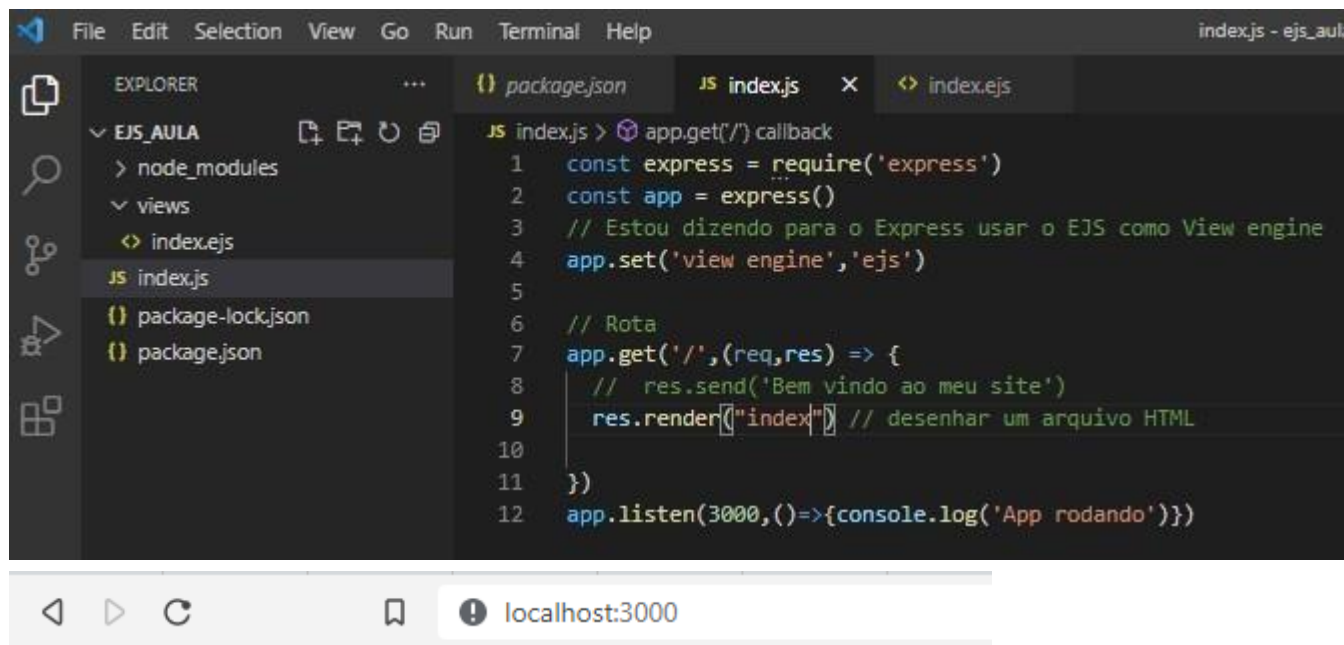
The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left and the Code editor on the right. The Explorer sidebar shows the project structure with a folder named **EJS\_AULA** containing subfolders **node\_modules** and **views**. The **views** folder is expanded, showing the file **index.ejs**. The Code editor displays the content of **index.ejs**, which is an HTML document with a head section containing meta tags for charset, http-equiv, and viewport, and a body section containing a heading and a horizontal line.

```
File Edit Selection View Go Run Terminal Help index.ejs - ejjs_aula - Visual Studio Code

EXPLORER
  EJS_AULA
    node_modules
    views
      index.ejs
  JS index.js
  package-lock.json
  package.json

views > <> index.ejs > html > body
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Formulário de Cadastro</title>
8 </head>
9 <body>
10   <h1> Formulário de Cadastro </h1>
11   <hr>
12
13 </body>
14 </html>
```

# Configurando e exibindo HTML com EJS



The image shows a development environment with VS Code and a web browser. In VS Code, the Explorer sidebar shows a project named 'EJS\_AULA' with files 'node\_modules', 'views', 'index.ejs', 'index.js', 'package-lock.json', and 'package.json'. The 'index.js' file is open in the editor, showing the following code:

```
JS index.js > app.get('/', callback)
1  const express = require('express')
2  const app = express()
3  // Estou dizendo para o Express usar o EJS como View engine
4  app.set('view engine', 'ejs')
5
6  // Rota
7  app.get('/', (req, res) => {
8    // res.send('Bem vindo ao meu site')
9    res.render("index") // desenhar um arquivo HTML
10
11  })
12  app.listen(3000, () => { console.log('App rodando') })
```


Below the code editor, a web browser window is visible, showing the address bar with 'localhost:3000'.

## Formulário de Cadastro

---

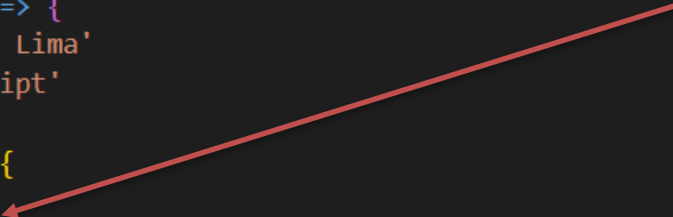
# Exibindo variáveis no HTML com EJS

JS index.js × <> index.ejs

JS index.js >  app.get("/") callback

```
1  const express = require("express");
2  const app = express();
3
4  // Estou dizendo para o Express usar o EJS como View engine
5  app.set('view engine', 'ejs');
6
7  app.get("/", (req, res) => {
8    var nome = 'Claudio Lima'
9    var lang = 'JavaScript'
10
11    res.render("index", {
12      nome: nome,
13      lang: lang,
14      empresa: "Fatec-SJC",
15    });
16  });
17
18
19  app.listen(3000, () => { console.log("App rodando!"); });
```

Passando variáveis para serem exibida no HTML



# Exibindo variáveis no HTML com EJS

```
JS index.js  <> index.ejs  X
views > <> index.ejs > html > body > h3
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <meta http-equiv="X-UA-Compatible" content="ie=edge">
7    <title>Guia perguntas</title>
8  </head>
9  <body>
10   <h3>Lista de Dados</h3>
11
12   <p>Nome:<%= nome %></p>
13   <p>Linguagem favorita:<%= lang %></p>
14   <p>Empresa:<%= empresa %></p>
15
16
17 </body>
18 </html>
```

Exibir valor da variável uma a tag  
<%= %>

# Exemplo usando HTML

```
JS app.js  X  <> formulario.ejs  <> cadastro.ejs
JS app.js > app.post('/') callback > email
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const ejs = require('ejs');
4
5  const app = express();
6  app.set('view engine', 'ejs');
7
8  app.use(bodyParser.urlencoded({ extended: true }));
9
10 app.get('/', (req, res) => {
11   res.render('formulario');
12 });
13
14 app.post('/', (req, res) => {
15   const nome = req.body.nome;
16   const email = req.body.email;
17   const senha = req.body.senha;
18
19   res.render('cadastro', { nome: nome, email: email });
20 });
21
22 app.listen(3000, () => {
23   console.log('Servidor rodando na porta');
24 });
25
```

Arquivo  
App.js

Para esse exemplo deve instalar:> npm install express ejs body-parser

# Exemplo usando HTML

```
JS app.js  formulario.ejs X  cadastro.ejs
views > <> formulario.ejs > html > body > form
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta charset="utf-8">
5    <title>Cadastro de Usuário</title>
6  </head>
7  <body>
8    <h1>Cadastro de Usuário</h1>
9    <form action="/" method="POST">
10     <label for="nome">Nome:</label>
11     <input type="text" id="nome" name="nome" required><br><br>
12
13     <label for="email">E-mail:</label>
14     <input type="email" id="email" name="email" required><br><br>
15
16     <label for="senha">Senha:</label>
17     <input type="password" id="senha" name="senha" required><br><br>
18
19     <input type="submit" value="Cadastrar">
20   </form>
21 </body>
22 </html>
```

Arquivo  
formulario.ejs



# Exemplo usando HTML

```
JS app.js  <> formulario.ejs  <> cadastro.ejs X
views > <> cadastro.ejs > ...
  1  <!DOCTYPE html>
  2  <html>
  3  <head>
  4    <meta charset="utf-8">
  5    <title>Cadastro de Usuário</title>
  6  </head>
  7  <body>
  8    <h1>Cadastro realizado com sucesso!</h1>
  9    <p>Nome: <%= nome %></p>
10    <p>E-mail: <%= email %></p>
11  </body>
12  </html>
13
```

Arquivo  
cadastro.ejs

# Gerador de aplicativos do express

1º Criar uma pasta chamado por exemplo apps

2º Entrar na pasta -> **cd apps**

3º digitar - > **npm install -g -s express express-generator**

4º digitar -> **express --view=ejs projeto**

5º Entrar na pasta projeto -> cd projeto

6º digitar -> **npm install** para instalar todas as dependências

7º digitar -> **npm start**

8º digite no browser localhos:3000

# Utilizando template engine Handlebars com Node.js

Para começar iremos instalar o Handlebars, pois ele ajuda a trabalhar com layout.

<https://handlebarsjs.com/>

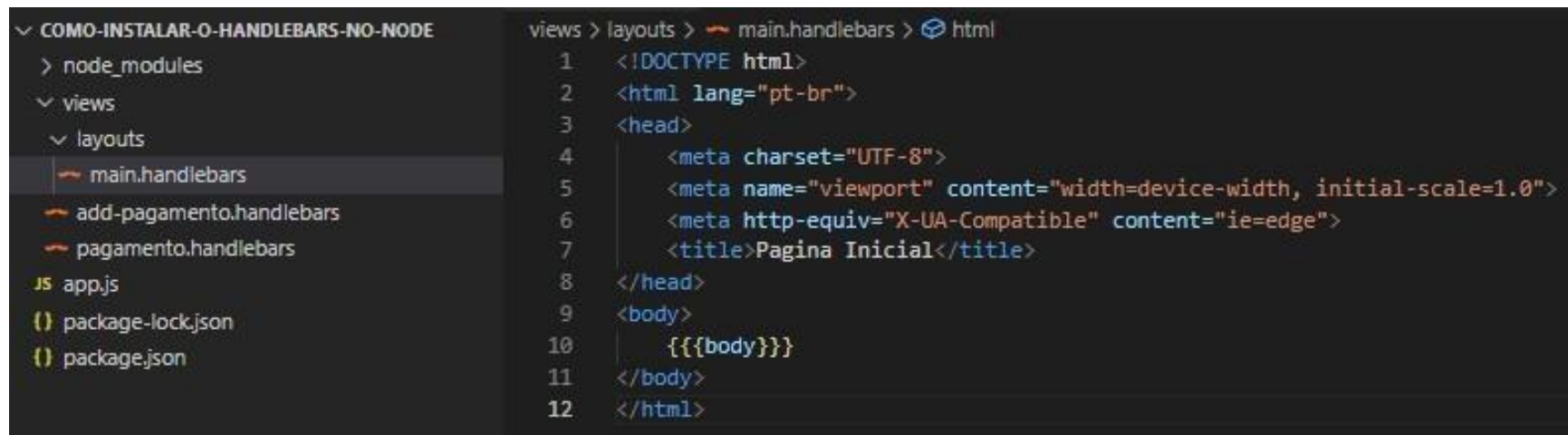
```
npm install --save express-handlebars
```

# Como instalar o Handlebars no Node e criar o layout padrão

```
JS app.js  X  main.handlebars
JS app.js > app.get('/pagamento') callback
1  const express = require("express");
2  const app = express();
3  const handlebars = require("express-handlebars");
4
5  app.engine('handlebars', handlebars({defaultLayout: 'main'}))
6  app.set('view engine', 'handlebars')
7
8  //Rotas
9  app.get('/', function(req, res){
10 |   res.send('Tela principal');
11 | });
12 app.get('/pagamento', function(req, res){
13 |   res.render('pagamento');
14 | });
15
16 app.get('/add-pagamento', function(req, res){
17 |   res.render('add-pagamento');
18 | });
19
20 app.listen(8080);
```

agora vou informar para o express que ele utilize o template Handlebars. Vou criar uma pasta chamado views e dentro dessa pasta criar uma pasta chamado layout e agora sim vou utilizar o Handlebars

# Como instalar o Handlebars no Node e criar o layout padrão

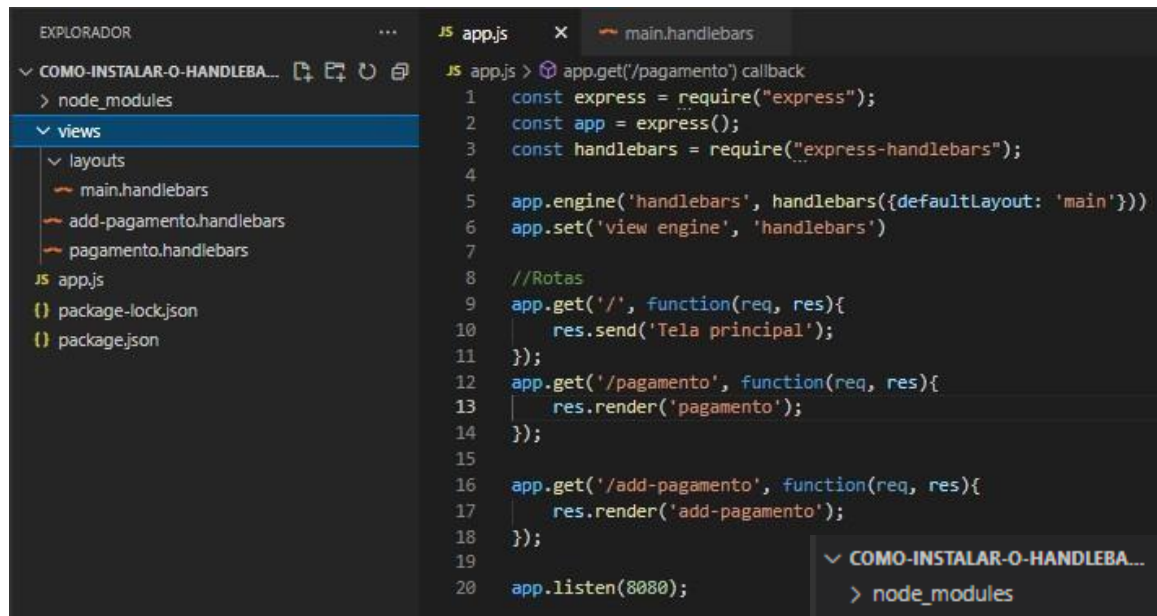


The image shows a code editor with a dark theme. On the left, a file explorer shows a project structure under the heading 'COMO-INSTALAR-O-HANDLEBARS-NO-NODE'. The files listed are 'node\_modules', 'views' (expanded to show 'layouts'), 'main.handlebars' (selected), 'add-pagamento.handlebars', 'pagamento.handlebars', 'app.js', 'package-lock.json', and 'package.json'. On the right, the content of 'main.handlebars' is displayed, showing a basic HTML template with line numbers 1 through 12.

```
views > layouts > main.handlebars > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Pagina Inicial</title>
8  </head>
9  <body>
10     {{{body}}}
11 </body>
12 </html>
```

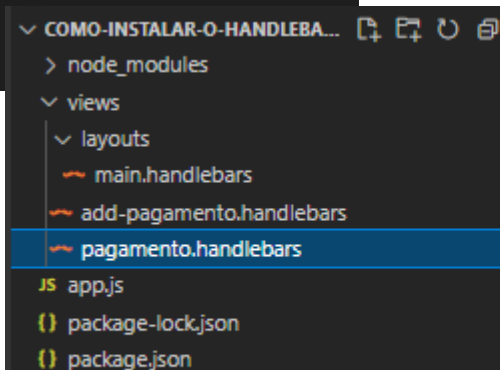
Layout padrão

# Como instalar o Handlebars no Node e criar o layout padrão



The screenshot shows the VS Code interface. On the left, the Explorer sidebar is open, showing a project structure with a 'views' folder containing 'main.handlebars', 'add-pagamento.handlebars', and 'pagamento.handlebars'. The main editor displays the 'app.js' file with the following code:

```
JS app.js > app.get('/pagamento') callback
1  const express = require("express");
2  const app = express();
3  const handlebars = require("express-handlebars");
4
5  app.engine('handlebars', handlebars({defaultLayout: 'main'}))
6  app.set('view engine', 'handlebars')
7
8  //Rotas
9  app.get('/', function(req, res){
10     res.send('Tela principal');
11 });
12 app.get('/pagamento', function(req, res){
13     res.render('pagamento');
14 });
15
16 app.get('/add-pagamento', function(req, res){
17     res.render('add-pagamento');
18 });
19
20 app.listen(8080);
```



This inset shows a closer view of the Explorer sidebar. The 'views' folder is expanded, and 'pagamento.handlebars' is selected. The file list includes 'main.handlebars', 'add-pagamento.handlebars', and 'pagamento.handlebars'.

views > pagamento.handlebars > h1

```
1 <h1>Listar pagamento</h1>
```

# Top 14 Templating Engines for JavaScript 2022 - Colorlib

- <https://colorlib.com/wp/top-templating-engines-for-javascript/>

# Parâmetros

Parâmetros são formas passar valores dinâmicos direto na sua rota. Iremos adicionar mais poder ainda nas rotas trabalhando com parâmetros.

Precisamos definir quais parâmetros quer receber na sua rota

```
JS app.js  {} package.json  JS index.js  X  {} package-lock.json  JS calculadora.js
express > JS index.js > app.get("/blog/:nome") callback
1  const express = require("express") // importando o express
2  const app = express() // iniciando o express e passando para a variavel app
3
4  app.get("/",function(requisicao, resposta){ // forma mais simples de criar um rota
5    resposta.send("Bem Vindo ao Express")
6  })
7
8  app.get("/blog/:nome",function(requisicao, resposta){
9    //requisicao: são dados enviados pelo usuário
10   //resposta: resposta que vai ser enviada para o usuário
11   resposta.send(requisicao.params.nome)
12 })
13
14 app.listen(3000, function(erro){
15   if (erro){
16     console.log ('Ocorreu um erro')
17   }else {
18     console.log ('Servidor iniciado com sucesso')
19   }
20
21 })
```

localhost:3000/blog/claudio  
claudio



# Parâmetros

Passando com variáveis

```
JS app.js  {} package.json  JS index.js  X  {} package-lock.json  JS calculadora.js

express > JS index.js > ...
1  const express = require("express") // importando o express
2  const app = express() // iniciando o express e passando para a variavel app
3
4  app.get("/",function(requisicao, resposta){ // forma mais simples de criar um rota
5      resposta.send("Bem Vindo ao Express")
6  })
7
8  app.get("/blog/:nome",function(requisicao, resposta){
9      //requisicao: são dados enviados pelo usuário
10     //resposta: resposta que vai ser enviada para o usuário
11     var nome=requisicao.params.nome
12     resposta.send("<h1> Olá" + nome + "</h1>")
13 })
14
15 app.listen(3000, function(erro){
16     if (erro){
17         console.log ('Ocorreu um erro')
18     }else {
19         console.log ('Servidor iniciado com sucesso')
20     }
21 })
22 })
```

# Parâmetros

```
1  const express = require("express") // importando o express
2  const app = express() // iniciando o express e passando para a variavel app
3
4  app.get("/",function(requisicao, resposta){ // forma mais simples de criar um rota
5    | resposta.send("Bem Vindo ao Express")
6    | })
7
8  app.get("/blog/:nome?",function(requisicao, resposta){
9    | //requisicao: são dados enviados pelo usuário
10   | //resposta: resposta que vai ser enviada para o usuário
11   | var nome=requisicao.params.nome
12   | if(nome){
13   |   resposta.send("<h1> Olá " + nome + "</h1>")
14   | }
15   | else{
16   |   resposta.send("Bem vindo ao meu blog!")
17   | }
18   | }) |
19
20  app.listen(3000, function(erro){
21  | if (erro){
22  |   console.log ('Ocorreu um erro')
23  | }else {
24  |   console.log ('Servidor iniciado com sucesso')
25  | }
26
27  })
```

# Banco de dados MySQL

Node.js pode ser usado em aplicativos de banco de dados. Para poder experimentar os exemplos de código, você deve ter o MySQL instalado no seu computador.

# Tutorial de instalação do SGBD MySQL

1º Acessar <https://www.mysql.com/downloads/>

2º Clicar no link [MySQL Community \(GPL\) Downloads »](#)

3º Clicar no link

<https://dev.mysql.com/downloads/mysql/>



General Availability (GA) Releases Archives ⓘ

## MySQL Community Server 8.0.28

Select Operating System:  
Microsoft Windows ▼

Looking for previous GA versions?

**Recommended Download:**

### MySQL Installer for Windows

All MySQL Products. For All Windows Platforms.  
In One Package.

Starting with MySQL 5.6 the MySQL Installer package replaces the standalone MSI packages.

Windows (x86, 32 & 64-bit), MySQL Installer MSI

[Go to Download Page >](#)

Clicar

# Tutorial de instalação do SGBD MySQL

- <https://dev.mysql.com/downloads/file/?id=510039>

## MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

**Login »**

using my Oracle Web account

**Sign Up »**

for an Oracle Web account

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

No thanks, just start my download.

**Para ignorar o  
cadastro**

# MySQL. Installer

Adding Community

## Choosing a Setup Type

Download

Installation

Installation Complete

## Choosing a Setup Type

Please select the Setup Type that suits your use case.

- ☐ **Developer Default**  
Installs all products needed for MySQL development purposes.
- ☒ **Server only**  
Installs only the MySQL Server product.
- ☐ **Client only**  
Installs only the MySQL Client products, without a server.
- ☐ **Full**  
Installs all included MySQL products and features.
- ☐ **Custom**  
Manually select the products that should be installed on the system.

### Setup Type Description

Installs only the MySQL Server. This type should be used where you want to deploy a MySQL Server, but will not be developing MySQL applications.

Next >

Cancel

# MySQL. Installer

## MySQL Server 8.0.28

### Type and Networking

Authentication Method

Accounts and Roles

Windows Service

Apply Configuration

## Type and Networking

### Server Configuration Type

Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.

Config Type: Development Computer

### Connectivity

Use the following controls to select how you would like to connect to this server.

☒ TCP/IP

Port: 3306

X Protocol Port: 33060

☒ Open Windows Firewall ports for network access

☐ Named Pipe

Pipe Name: MYSQL

☐ Shared Memory

Memory Name: MYSQL

### Advanced Configuration

Select the check box below to get additional configuration pages where you can set advanced and logging options for this server instance.

☐ Show Advanced and Logging Options

Next >

Cancel

# MySQL Installer

## MySQL Server 8.0.28

Type and Networking

Authentication Method

Accounts and Roles

Windows Service

Apply Configuration

### Authentication Method

☒ Use Strong Password Encryption for Authentication (RECOMMENDED)

MySQL 8 supports a new authentication based on improved stronger SHA256-based password methods. It is recommended that all new MySQL Server installations use this method going forward.



Attention: This new authentication plugin on the server side requires new versions of connectors and clients which add support for this new 8.0 default authentication (caching\_sha2\_password authentication).

Currently MySQL 8.0 Connectors and community drivers which use libmysqlclient 8.0 support this new method. If clients and applications cannot be updated to support this new authentication method, the MySQL 8.0 Server can be configured to use the legacy MySQL Authentication Method below.

☐ Use Legacy Authentication Method (Retain MySQL 5.x Compatibility)

Using the old MySQL 5.x legacy authentication method should only be considered in the following cases:

- If applications cannot be updated to use MySQL 8 enabled Connectors and drivers.
- For cases where re-compilation of an existing application is not feasible.
- An updated, language specific connector or driver is not yet available.

Security Guidance: When possible, we highly recommend taking needed steps towards upgrading your applications, libraries, and database servers to the new stronger authentication. This new method will significantly improve your security.

&lt; Back

Next &gt;

Cancel



# MySQL<sup>®</sup> Installer

MySQL Server 8.0.28

Type and Networking

Authentication Method

Accounts and Roles

Windows Service

Apply Configuration

## Accounts and Roles

### Root Account Password

Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:

Repeat Password:

Password strength: **Weak**

### MySQL User Accounts

Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL User Name	Host	User Role

Add User

Edit User

Delete

&lt; Back

Next &gt;

Cancel

# MySQL. Installer

MySQL Server 8.0.28

Type and Networking

Authentication Method

Accounts and Roles

Windows Service

Apply Configuration

## Windows Service

☒ Configure MySQL Server as a Windows Service

### Windows Service Details

Please specify a Windows Service name to be used for this MySQL Server instance. A unique name is required for each instance.

Windows Service Name:

☒ Start the MySQL Server at System Startup

### Run Windows Service as ...

The MySQL Server needs to run under a given user account. Based on the security requirements of your system you need to pick one of the options below.

☒ Standard System Account

Recommended for most scenarios.

☐ Custom User

An existing user account can be selected for advanced scenarios.

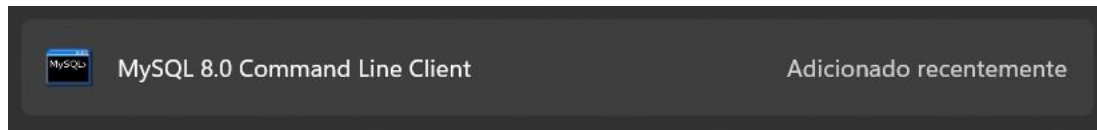
< Back

Next >

Cancel

# Acessando o Mysql

## 01 Opção



02 Opção: Ir no prompt de comando e digitar-> `mysql -h localhost -u root -p`

# Estrutura da tabela no Mysql

---

- 01 – Criar o Banco de dados chamado Crud
- Create database crud;
- Use crud;
- 02 – Criar a tabela

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
nome	varchar(40)	YES		NULL	
idade	int	YES		NULL	
uf	varchar(2)	YES		NULL	

```
CREATE TABLE clientes (  
    id INT NOT NULL AUTO_INCREMENT,  
    nome VARCHAR(255) NOT NULL,  
    idade int NOT NULL,  
    uf VARCHAR(20) NOT NULL,  
    PRIMARY KEY (id)  
);
```

# CRUD Node.js com Mysql

1. Crie uma pasta para o projeto, com o nome que quiser
2. Abra o terminal e dentro dessa pasta rode um **npm init** para inicializar o projeto e criar o package.json

03- Ainda no terminal, mande instalar a dependência mysql2, como abaixo.

**npm** install mysql2 --save

# CRUD Node.js com Mysql – Criando a conexão

Comece criando uma conexão com o banco de dados.

Use o nome de usuário e a senha do seu banco de dados MySQL.

```
JS db1.js > [🔑] con > password
1  var mysql = require('mysql2');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "12345678"
7  });
8
9  con.connect(function(err) {
10    if (err) throw err;
11    console.log("Conectado!");
12  });
```

# CRUD Node.js com Mysql – Criando um banco de dados

Para criar um banco de dados no MySQL, use a instrução "CREATE DATABASE:

```
JS db1.js > [?] con > 🔑 password
1
2   var mysql = require('mysql2');
3
4   var con = mysql.createConnection({
5     host: "localhost",
6     user: "root",
7     password: "12345678"
8   });
9
10  con.connect(function(err) {
11    if (err) throw err;
12    console.log("Conectado!");
13    con.query("CREATE DATABASE crud", function (err, result) {
14      if (err) throw err;
15      console.log("Banco de dados Criado!");
16    });
17  });
18
```

# CRUD Node.js com Mysql – Criando uma tabela


Para criar uma tabela no MySQL, use a instrução "CREATE TABLE:

```
JS db1.js > ...
1  var mysql = require('mysql2');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "12345678",
7    database: "Crud"
8  });
9
10 con.connect(function(err) {
11   if (err) throw err;
12   console.log("Conectado!");
13   var sql = "CREATE TABLE clientes1 (id INT NOT NULL AUTO_INCREMENT,"+
14   || "nome VARCHAR(255) NOT NULL,idade int NOT NULL, uf VARCHAR(20) NOT NULL,PRIMARY KEY (id))"
15   con.query(sql, function (err, result) {
16     if (err) throw err;
17     console.log("Tabela Criada");
18   });
19 });
```



# CRUD Node.js com Mysql – Inserir na Tabela

Para preencher uma tabela no MySQL, use a instrução "INSERT INTO"

```
JS db1.js >  con.connect() callback
1  var mysql = require('mysql2');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "12345678",
7    database: "crud"
8  });
9
10 con.connect(function(err) {
11   if (err) throw err;
12   console.log("Conectado!");
13   var sql = "INSERT INTO clientes (id, nome, idade, uf) VALUES (2, 'Maria',15,'MG')";
14   con.query(sql, function (err, result) {
15     if (err) throw err;
16     console.log("Registro inserido");
17   });
18 });
```

# CRUD Node.js com Mysql – Seleção de uma Tabela

Para selecionar dados de uma tabela no MySQL, use a instrução "SELECT".

```
JS db1.js >  con.connect() callback
1  var mysql = require('mysql2');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "12345678",
7    database: "crud"
8  });
9
10 con.connect(function(err) {
11   if (err) throw err;
12   con.query("SELECT * FROM clientes", function (err, result, fields) {
13     if (err) throw err;
14     console.log(result);
15   });
16 });
```

# CRUD Node.js com Mysql – Selecione com um filtro

Ao selecionar registros de uma tabela, você pode filtrar a seleção usando a instrução "WHERE":

```
JS db1.js > con.connect() callback
1  var mysql = require('mysql2');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "12345678",
7    database: "crud"
8  });
9
10 con.connect(function(err) {
11   if (err) throw err;
12   con.query("SELECT * FROM clientes WHERE nome = 'Maria'", function (err, result) {
13     if (err) throw err;
14     console.log(result);
15   });
16 });
```

# CRUD Node.js com Mysql – Ordenar o resultado

Use a instrução ORDER BY para classificar o resultado em ordem crescente ou decrescente “DESC”.

```
JS db1.js > con.connect() callback
1  var mysql = require('mysql2');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "12345678",
7    database: "crud"
8  });
9
10 con.connect(function(err) {
11   if (err) throw err;
12   con.query("SELECT * FROM clientes ORDER BY nome", function (err, result) {
13     if (err) throw err;
14     console.log(result);
15   });
16 });
```

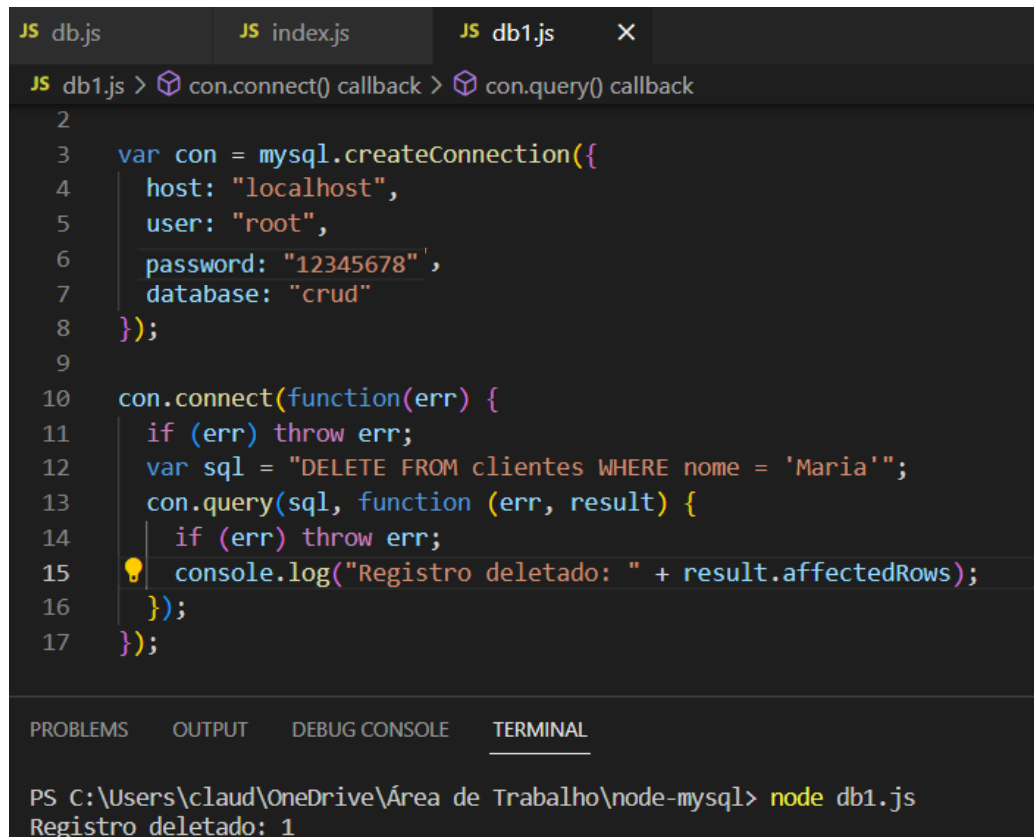
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\claud\OneDrive\Área de Trabalho\node-mysql> node db1.js
```

```
[
  TextRow { id: 1, nome: 'Claudio', idade: 45, uf: 'SP' },
  TextRow { id: 2, nome: 'Maria', idade: 15, uf: 'MG' }
]
```

# CRUD Node.js com Mysql – Apagar registro

Você pode excluir registros de uma tabela existente usando a instrução "DELETE FROM":



```
JS db1.js JS index.js JS db1.js X
JS db1.js > con.connect() callback > con.query() callback
2
3 var con = mysql.createConnection({
4   host: "localhost",
5   user: "root",
6   password: "12345678",
7   database: "crud"
8 });
9
10 con.connect(function(err) {
11   if (err) throw err;
12   var sql = "DELETE FROM clientes WHERE nome = 'Maria'";
13   con.query(sql, function (err, result) {
14     if (err) throw err;
15     console.log("Registro deletado: " + result.affectedRows);
16   });
17 });

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\claud\OneDrive\Área de Trabalho\node-mysql> node db1.js
Registro deletado: 1
```

# CRUD Node.js com Mysql – Excluir tabela

Você pode excluir uma tabela existente usando a instrução "DROP TABLE":

```
JS db1.js  X
JS db1.js > con.connect() callback > con.query() callback
2
3   var con = mysql.createConnection({
4     host: "localhost",
5     user: "root",
6     password: "12345678",
7     database: "crud"
8   });
9
10  con.connect(function(err) {
11    if (err) throw err;
12    var sql = "DROP TABLE clientes";
13    con.query(sql, function (err, result) {
14      if (err) throw err;
15      console.log("Tabela deletada");
16    });
17  });
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```
PS C:\Users\claud\OneDrive\Área de Trabalho\node-mysql> node db1.js
Tabela deletada
```

# CRUD Node.js com Mysql – Atualizar tabela



Você pode atualizar os registros existentes em uma tabela usando a instrução "UPDATE":

```
JS db1.js  X
JS db1.js > [con] con > password
2
3   var con = mysql.createConnection({
4     host: "localhost",
5     user: "root",
6     password: "12345678",
7     database: "crud"
8   });
9
10  con.connect(function(err) {
11    if (err) throw err;
12    var sql = "UPDATE clientes SET nome = 'Luiz Claudio' WHERE nome = 'Claudio'";
13    con.query(sql, function (err, result) {
14      if (err) throw err;
15      console.log(result.affectedRows + " Registro(s) alterado");
16    });
17  });
```

# CRUD Node.js com Mysql – Limite o resultado

Você pode limitar o número de registros retornados da consulta, usando a instrução

"LIMIT":

```
JS db1.js >  con.connect() callback >  sql
2
3   var con = mysql.createConnection({
4     host: "localhost",
5     user: "root",
6     password: "12345678",
7     database: "crud"
8   });
9
10  con.connect(function(err) {
11    if (err) throw err;
12    var sql = "SELECT * FROM clientes LIMIT 5";
13    con.query(sql, function (err, result) {
14      if (err) throw err;
15      console.log(result);
16    });
17  });
```



# CRUD Node.js com Mysql – Juntar duas ou mais tabela

Você pode combinar linhas de duas ou mais tabelas, com base em uma coluna relacionada entre elas, usando uma instrução JOIN.

```
CREATE TABLE clientes (  
  id INT NOT NULL AUTO_INCREMENT,  
  nome VARCHAR(255) NOT NULL,  
  idade int NOT NULL,  
  uf VARCHAR(20) NOT NULL,  
  PRIMARY KEY (id)  
);
```

```
CREATE TABLE pedidos (  
  id INT PRIMARY KEY,  
  cliente_id INT,  
  produto VARCHAR(100),  
  quantidade INT,  
  valor_unitario DECIMAL(10, 2),  
  FOREIGN KEY (cliente_id) REFERENCES clientes(id)  
);
```

# CRUD Node.js com Mysql – Juntar duas ou mais tabela

Você pode combinar linhas de duas ou mais tabelas, com base em uma coluna relacionada entre elas, usando uma instrução JOIN.

```
JS db1.js  X
JS db1.js > con.connect() callback > con.query() callback
1  var mysql = require('mysql2');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "12345678",
7    database: "crud"
8  });
9
10 con.connect(function(err) {
11   if (err) throw err;
12   var sql = "SELECT c.nome,p.produto, p.quantidade, p.valor_unitario FROM clientes c INNER JOIN pedidos p ON c.id = p.cliente_id";
13   con.query(sql, function (err, result) {
14     if (err) throw err;
15     console.log(result);
16   });
17 });
```



**Exemplo de um CRUD simples  
com Front e Back**

# CRUD Node.js com Mysql

## Estrutura da Tabela usuários

```
CREATE TABLE usuarios (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  nome VARCHAR(50) NOT NULL,  
  email VARCHAR(100) NOT NULL  
);
```

# CRUD Node.js com Mysql

## **Passo 1: Configurar o ambiente e instalar as dependências**

Crie uma pasta para o seu projeto e execute o seguinte comando na linha de comando para criar um arquivo package.json e instalar as dependências necessárias:

**npm init**

**npm install express mysql2 body-parser ejs**

# CRUD Node.js com Mysql

## Passo 2: Criar a estrutura do projeto

Crie os seguintes arquivos na pasta do seu projeto:

**app.js:** O arquivo principal do aplicativo.

**views/:** Uma pasta para armazenar os arquivos de visualização.

**views/index.ejs:** A visualização para listar registros.

**views/create.ejs:** A visualização para criar registros.

**views/edit.ejs:** A visualização para editar registros.

# CRUD Node.js com Mysql

**Passo 3: App.js, views/index.ejs, views/create.ejs, views/edit.ejs**

[https://drive.google.com/drive/folders/1Bs7YfStXPCzryZU7O2iVEuaQEAbXS\\_AG?usp=sharing](https://drive.google.com/drive/folders/1Bs7YfStXPCzryZU7O2iVEuaQEAbXS_AG?usp=sharing)



**CRUD Trabalhando com função**



# CRUD Node.js com Mysql – Criando a conexão

Crie um **arquivo db.js** para criarmos a conexão com nossos banco de dados, usando o pacote mysql2 que acabamos de instalar.

JS db.js >  connect

```
1  //db.js
2  const mysql = require("mysql2/promise");
3  async function connect(){
4      if(global.connection && global.connection.state !== 'disconnected')
5          return global.connection;
6
7  const connection = await mysql.createConnection({
8      host      : 'localhost',
9      port      : 3306,
10     user       : 'root',
11     password   : '12345678',
12     database   : 'crud'
13 });
14
15 console.log('Conectou no MySQL!');
16 global.connection = connection;
17 return global.connection;
18 }
```

palavra reservada **await** antes do `createConnection`. Isso porque o `createConnection` é assíncrono, o que me obriga a usar `callback`, `promise` ou `async/await` para garantir que a instrução a seguir só aconteça depois da conexão já estabelecida.

# CRUD Node.js com Mysql – Criando a conexão

Para testar este código db.js, crie uma chamada a esta conexão ao fim do módulo e importe-o no **index.js**, para dispará-la.

```
//index.js
```

```
const db = require("./db")
```

# CRUD Node.js com Mysql – Select

---

- Fizemos a conexão, e agora como vamos utilizá-la para fazer um CRUD no MySQL.
- Mesmo módulo **db.js**, cria outra function, desta vez para **select**.

```
JS db.js > [?] <unknown>
1  //db.js
2  const mysql = require("mysql2/promise");
3  async function connect(){
4      if(global.connection && global.connection.state !== 'disconnected')
5          return global.connection;
6
7  const connection = await mysql.createConnection({
8      host      : 'localhost',
9      port      : 3306,
10     user       : 'root',
11     password   : '12345678',
12     database   : 'crud'
13 });
14
15 console.log('Conectou no MySQL!');
16 global.connection = connection;
17 return global.connection;
18 }
19
20 async function selectCustomers(){
21     const conn = await connect();
22     const [rows] = await conn.query('SELECT * FROM clientes;');
23     return rows;
24 }
25 module.exports = {selectCustomers}
```

# CRUD Node.js com Mysql – Select

Exportamos a função **selectCustomers**, pois queremos usar ela no nosso index.js.

```
//index.js
(async () => {
  const db = require("./db");
  console.log('Começou!');

  console.log('SELECT * FROM CLIENTES');
  const clientes = await db.selectCustomers();
  console.log(clientes);
})();
```

# CRUD Node.js com Mysql – Insert

---

- Inserindo clientes no MySQL
- Volte ao **db.js** e insira uma nova função

```
JS db.js > ...
1 //db.js
2 const mysql = require("mysql2/promise");
3 async function connect(){
4     if(global.connection && global.connection.state !== 'disconnected')
5         return global.connection;
6 }
7 const connection = await mysql.createConnection({
8     host      : 'localhost',
9     port      : 3306,
10    user       : 'root',
11    password   : '12345678',
12    database   : 'crud'
13 });
14
15 console.log('Conectou no MySQL!');
16 global.connection = connection;
17 return global.connection;
18 }
19
20 async function selectCustomers(){
21     const conn = await connect();
22     const [rows] = await conn.query('SELECT * FROM clientes;');
23     return rows;
24 }
25
26 async function insertCustomer(customer){
27     const conn = await connect();
28     const sql = 'INSERT INTO clientes(nome,idade,uf) VALUES (?, ?, ?)';
29     const values = [customer.nome, customer.idade, customer.uf];
30     return await conn.query(sql, values);
31 }
32
33 module.exports = {selectCustomers, insertCustomer}
```

# CRUD Node.js com Mysql – Insert

---

- Exportamos a função **InsertCustomer**, pois queremos usar ela no nosso index.js.

```
JS index.js > ...
1  //index.js
2  (async () => {
3      const db = require("../db");
4      console.log('Começou!');
5
6      console.log('INSERT INTO CLIENTES');
7      const result = await db.insertCustomer({nome:"Arthur", idade: 18, uf: "SP"});
8      console.log(result);
9
10     console.log('SELECT * FROM CLIENTES');
11     const clientes = await db.selectCustomers();
12     console.log(clientes);
13
14
15 })();
16
```

# CRUD Node.js com Mysql – Update e Delete

---

- Inserindo clientes no MySQL
- Volte ao **db.js** e insira uma nova função

```
JS db.js > ...
13 });
14
15 console.log('Conectou no MySQL!');
16 global.connection = connection;
17 return global.connection;
18 }
19
20 async function selectCustomers(){
21     const conn = await connect();
22     const [rows] = await conn.query('SELECT * FROM clientes;');
23     return rows;
24 }
25
26 async function insertCustomer(customer){
27     const conn = await connect();
28     const sql = 'INSERT INTO clientes(nome,idade,uf) VALUES (?,?,?)';
29     const values = [customer.nome, customer.idade, customer.uf];
30     return await conn.query(sql, values);
31 }
32
33 async function updateCustomer(id, customer){
34     const conn = await connect();
35     const sql = 'UPDATE clientes SET nome=?, idade=?, uf=? WHERE id=?';
36     const values = [customer.nome, customer.idade, customer.uf, id];
37     return await conn.query(sql, values);
38 }
39
40 async function deleteCustomer(id){
41     const conn = await connect();
42     const sql = 'DELETE FROM clientes where id=?';
43     return await conn.query(sql, [id]);
44 }
45
46 module.exports = {selectCustomers, insertCustomer, updateCustomer, deleteCustomer}
```

# CRUD Node.js com Mysql – Update e Delete

---

- Exportamos as funções, pois queremos usar ela no nosso index.js.

```
JS index.js > ...
1 //index.js
2 (async () => {
3     const db = require("./db");
4     console.log('Começou!');
5
6     console.log('INSERT INTO CLIENTES');
7     const result = await db.insertCustomer({nome:"Arthur", idade: 18, uf: "SP"});
8     console.log(result);
9
10    console.log('SELECT * FROM CLIENTES');
11    const clientes = await db.selectCustomers();
12    console.log(clientes);
13
14    console.log('UPDATE CLIENTES');
15    const result2 = await db.updateCustomer(6, {nome: "Zé José", idade: 19, uf: "SP"});
16    console.log(result2);
17
18    console.log('DELETE FROM CLIENTES');
19    const result3 = await db.deleteCustomer(7);
20    console.log(result3);
21
22 })();
```





**CRUD Trabalhando com função  
passando parâmetro.**

# Função para incluir os dados de um cliente

JS Crud.js > inserirCliente > connection.query() callback

```
1  const mysql = require('mysql2');
2
3  const connection = mysql.createConnection({
4    host: 'localhost',
5    user: 'root',
6    password: 'Cilmara0',
7    database: 'crud',
8  });
9
10 // Função para incluir os dados de um cliente
11 function inserirCliente(nome, email, telefone) {
12   const query = `INSERT INTO clientes (nome, email, telefone) VALUES ('${nome}', '${email}', '${telefone}')`;
13
14   connection.query(query, (error, result) => {
15     if (error) {
16       console.error(error);
17     } else {
18       console.log(`Cliente inserido com sucesso!`);
19     }
20   });
21 }
```

## Função para alterar os dados de um cliente

```
// Função para alterar os dados de um cliente
function alterarCliente(id, nome, email, telefone) {
  const query = `UPDATE clientes SET nome = ?, email = ?, telefone = ? WHERE id = ?`;

  connection.query(query, [nome, email, telefone, id], (error, results, fields) => {
    if (error) {
      console.error(error);
    } else {
      console.log(`Alteração do cliente com Sucesso`);
    }
  });
}
```

## Função para deletar um cliente

```
// Função para deletar um cliente
function deletarCliente(id) {
  const query = `DELETE FROM clientes WHERE id = ?`;
  connection.query(query, [id], (error, results, fields) => {
    if (error) {
      console.error(error);
      callback(error, null);
    } else {
      console.log(`Cliente excluido com Sucesso`);
    }
  });
}
```

## Função para consultar os dados de um cliente

```
// Função para consultar os dados de um cliente
function consultarCliente(id) {
  const query = `SELECT * FROM clientes WHERE id = ?`;

  connection.query(query, [id], (error, results, fields) => {
    if (error) {
      console.error(error);
    } else {
      console.log(results[0]);
    }
  });
}

module.exports = {inserirCliente, alterarCliente,deletarCliente,consultarCliente}
```

## Mostrar resultado das funções

JS dados.js > ...

```
1  const {inserirCliente,alterarCliente,deletarCliente,consultarCliente } = require('./Crud');
2  //inserirCliente('Claudio', 'claudioelima@gmail.com', '00981424744')
3  //alterarCliente(1, 'Arthur','arthur.ribeiro@gmail','1297729439')
4  //deletarCliente(1)
5  consultarCliente(2)
6
```

# CRUD com Node.js, Sequelize e MySQL

O Sequelize é um ORM (Mapeamento Objeto Relacional) para Node.js que oferece uma solução baseada em promessas para trabalhar com bancos de dados relacionais, como o MySQL, PostgreSQL, SQLite, Microsoft SQL Server e outros. Ele fornece uma maneira fácil e intuitiva de definir modelos, mapear tabelas de banco de dados para esses modelos e executar consultas no banco de dados usando esses modelos.

Com o Sequelize, você pode definir seus modelos usando classes JavaScript simples, que são mapeados automaticamente para tabelas de banco de dados. Ele também fornece recursos úteis, como validação de entrada, relacionamentos entre tabelas, transações e muitos outros recursos para trabalhar com bancos de dados relacionais de maneira eficiente e fácil.

# CRUD com Node.js, Sequelize e MySQL

Para conectar o Sequelize ao MySQL, você precisa instalar o pacote do Sequelize e do driver do MySQL. Você pode fazer isso executando os seguintes comandos no terminal:

- 1) Criar uma pasta para o seu projeto;
- 2) Iniciar o projeto => **npm init**;
- 3) **npm install --save sequelize mysql2**



# CRUD com Sequelize e MySQL – Criar conexão

```
JS db.js > [?] sequelize
1  const Sequelize = require('sequelize');
2
3  const sequelize = new Sequelize('crud', 'root', '12345678', {
4    host: 'localhost',
5    dialect: 'mysql'
6  });
7
8  /*
9  sequelize
10   .authenticate()
11   .then(() => {
12     console.log('Conexão bem sucedida')
13   })
14   .catch( err => {
15     console.log('Erro ao conectar:', err)
16   })
17  */
18  module.exports = sequelize;
```

# CRUD com Sequelize e MySQL – Criando Banco

```
JS produto.js > [?] <unknown>
1  const Sequelize = require('sequelize');
2  const database = require('./db');
3
4  const Produto = database.define('produto', {
5    id: {
6      type: Sequelize.INTEGER,
7      autoIncrement: true,
8      allowNull: false,
9      primaryKey: true
10   },
11   nome: {
12     type: Sequelize.STRING,
13     allowNull: false
14   },
15   preco: {
16     type: Sequelize.DOUBLE
17   },
18   descricao: Sequelize.STRING
19 })
20
21 try {
22   Produto.sync({ force: true });
23   console.log('Tabela criada com sucesso!');
24 } catch (error) {
25   console.error('Erro ao criar tabela:', error);
26 }
27
28 module.exports = Produto;
```


# CRUD com Sequelize e MySQL – Create

```
JS index.js > ...
1  //index.js
2  (async () => {
3    const database = require('./db');
4    const Produto = require('./produto');
5
6    const resultado = await database.sync();
7
8    // Operação de criação (Create)
9    const resultadoCreate = await Produto.create({
10
11      nome: 'mouse',
12      preco: 10,
13      descricao: 'Um mouse USB',
14    })
15  })
16
17  const resultadoCreat1 = await Produto.create({
18    nome: 'teclado',
19    preco: 20,
20    descricao: 'Teclado mecanico'
21  })
22  })();
```


# CRUD com Sequelize e MySQL – Read

```
JS index.js > <function>
1  //index.js
2  (async () => {
3    const database = require('./db');
4    const Produto = require('./produto');
5
6    const resultado = await database.sync();
7
8  // Operação de ler (Read)
9  //const produtos = await Produto.findAll() Operação de leitura (Read) - Mostrar todos os produtos
10 const produtos = await Produto.findAll()
11 console.log(produtos);
12 })();
```

# CRUD com Sequelize e MySQL – Update

```
JS index.js >  <function>
1  //index.js
2  (async () => {
3      const database = require('./db');
4      const Produto = require('./produto');
5
6      const resultado = await database.sync();
7
8      // Operação de alteração (Update)
9      const produto = await Produto.findByPk(1);
10
11     produto.nome = "Mouse Top";
12
13     const resultadoSave = await produto.save();
14     console.log(resultadoSave);
15 })();
```

# CRUD com Sequelize e MySQL – Delete

```
JS index.js >  <function>
1  //index.js
2  ✓ (async () => {
3      const database = require('./db');
4      const Produto = require('./produto');
5
6      const resultado = await database.sync();
7
8      // Operação de deletar (delete)
9      const produto = await Produto.findByPk(1);
10     produto.destroy();
11     })();
```