

Aula 06:

Introdução ao Node.js
e ao Express.js

Node JS

Node.js é um interpretador de código Javascript que roda fora dos navegadores. Através dele podemos usar o Javascript também para desenvolvimento back-end.



Link para download: <https://nodejs.org/en/download>



JS

Vantagens

- Muito leve
- Muito rápido
- Permite usar a mesma linguagem no front e back-end (JS)
- Possui uma vasta quantidade de bibliotecas
- Está sendo utilizado fortemente no mercado



JS

Express JS

O Express.js é um framework que nos ajuda a construir aplicações web com o Node.js. Possui diversas ferramentas deixando o trabalho muito mais fácil e prático.

Express

JS



JS

Node Package Manager (NPM)



O NPM é o gerenciador de pacotes do Node.js, é utilizado para criar projetos Node e baixar diversas bibliotecas.



npm -v :

Verifica a versão do npm

npm init :

Inicia um projeto Node.js

npm install express --save :

Instala o Express.js

JS

Iniciando um projeto Node



Após já ter instalado o Node em seu computador, o primeiro passo para iniciarmos um projeto Node, é criar a pasta do projeto.

Em seguida acesse o console do VS Code (atalho: **ctrl + “**) e navegue até a pasta do projeto (utilize **cd** e **cd..** para navegar entre as pastas).

Uma vez dentro da pasta do projeto utilize o comando **npm init** para iniciar um projeto.

```
Centro Paula Souza\Node.js\Node> npm init
```

JS

Iniciando um projeto Node



Em seguida, basta preencher as informações do projeto ou deixa-las em branco e avançar com a tecla ENTER.

```
Press ^C at any time to quit.  
package name: (node)  
version: (1.0.0)  
description:  
entry point: (index.js)  
test command:  
git repository:  
keywords:  
author:  
license: (ISC)  
About to write to C:\Users\Prof. Diego\OneDrive -  
e\package.json:
```

Uma vez iniciado o projeto um arquivo com o nome **package.json** será criado contendo as informações do projeto.

JS

Instalando o Express.js

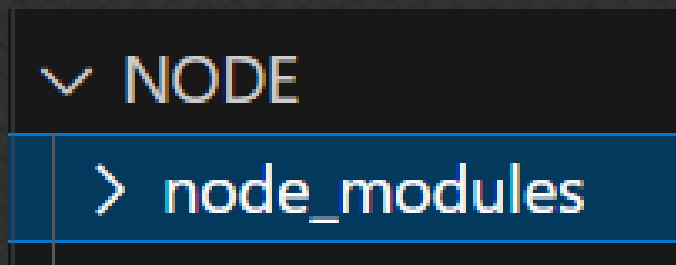
Agora que iniciamos nosso projeto, precisamos instalar o framework Express.js.

Para isso basta utilizar o comando **npm install express --save** no console

```
Prof. Diego\OneDrive - Etec Centro Paula Souza\fatec\node> npm install express --save  
.....] / idealTree:node: sill idealTree buildDeps
```

Após a instalação do Express, uma pasta com o nome **node_modules** será criada.

Nesta pasta estão os **módulos** do framework.



JS

Módulos

Módulos são “pedaços” do seu programa. Consiste em uma forma de dividir o código de um sistema em vários arquivos. Através dos módulos é possível também importar bibliotecas a sua aplicação.

Exemplo:

Exportando: **module.exports** = *connection*

Importando: **require**('*connection*')



JS

Importando o Express.js



Após instalar o Express é necessário importá-lo em nossa aplicação. Para isso, crie o primeiro arquivo com o nome **index.js** na pasta do projeto. Dentro do arquivo **index.js**, declaramos uma constante com o identificador **express** e importamos a biblioteca express como um **módulo**. Após isso, carregamos o express dentro da constante **app**.

```
JS index.js > ...
```

```
1  const express = require("express"); // Importando o express
2  const app = express(); // Iniciando o express
```

JS

Iniciando o servidor

Uma vez que temos o Express carregado dentro da constante `app`, podemos utilizá-la para chamar os seus métodos. Para iniciarmos nosso servidor Node, utilizaremos o método `listen`, informando a porta que será utilizada pelo servidor, no nosso caso utilizaremos a porta `8080`:

```
app.listen(8080, function(err){
  if(err){
    console.log("Ocorreu um erro!")
  } else {
    console.log("Servidor iniciado com sucesso!")
  }
})
```

Nota: Por padrão na web utilizamos a porta 80 para HTTP e 443 para HTTPS.



JS

Iniciando o servidor



Para testarmos nosso servidor, precisamos agora executar nosso arquivo **index.js** com o comando **node**, através do console. Lembre-se que para isso, devemos estar dentro da pasta do projeto (utilize **cd** e **cd..** para navegar entre as pastas).

```
● PS C:\Users\Prof. Diego\OneDrive - Etec Centro Paula Souza\Node.js> cd node
○ PS C:\Users\Prof. Diego\OneDrive - Etec Centro Paula Souza\Node.js\node> node index.js
Servidor iniciado com sucesso!
```

A mensagem de sucesso deve ser exibida informando o carregamento do servidor.

JS

Automatizando o servidor com Nodemon



Podemos automatizar o processo de reiniciar o servidor com o utilitário Nodemon. Assim ele irá reiniciar o servidor toda vez que detectar alguma alteração no código da aplicação.

O primeiro passo é instalar o Nodemon através do npm. Para isso, dentro da pasta do seu projeto, digite o comando **npm install nodemon -g** no console.

```
\Node.js\node> npm install nodemon -g
```

Feito isso, basta iniciar o servidor com o comando **npx nodemon index.js**. Agora a cada alteração no código, o servidor será reiniciado automaticamente.

```
PS C:\Users\Prof. Diego\OneDrive - Etec Centro Paula Souza\Node.js\node> nodemon index.js
[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Servidor iniciado com sucesso!
```

JS

O que são rotas?



Rotas são os diversos caminhos que uma aplicação pode ter.

São exemplos de rotas:

`meuapp.com/home`

`meuapp.com/perfil`

`meuapp.com/video/14`



JS

Criando rotas no Express.js



Existem várias formas de se criar rotas com Express.js, uma delas é utilizar o método *get*:

```
6  app.get("/", function(req, res){
7    |    res.send("<h1>Bem-vindo ao meu site!</h1>")
8  |  })
9
10 app.get("/perfil", function(req, res){
11    |    res.send("<h1>Perfil do usuário</h1>")
12  |  })
```

req: Requisição

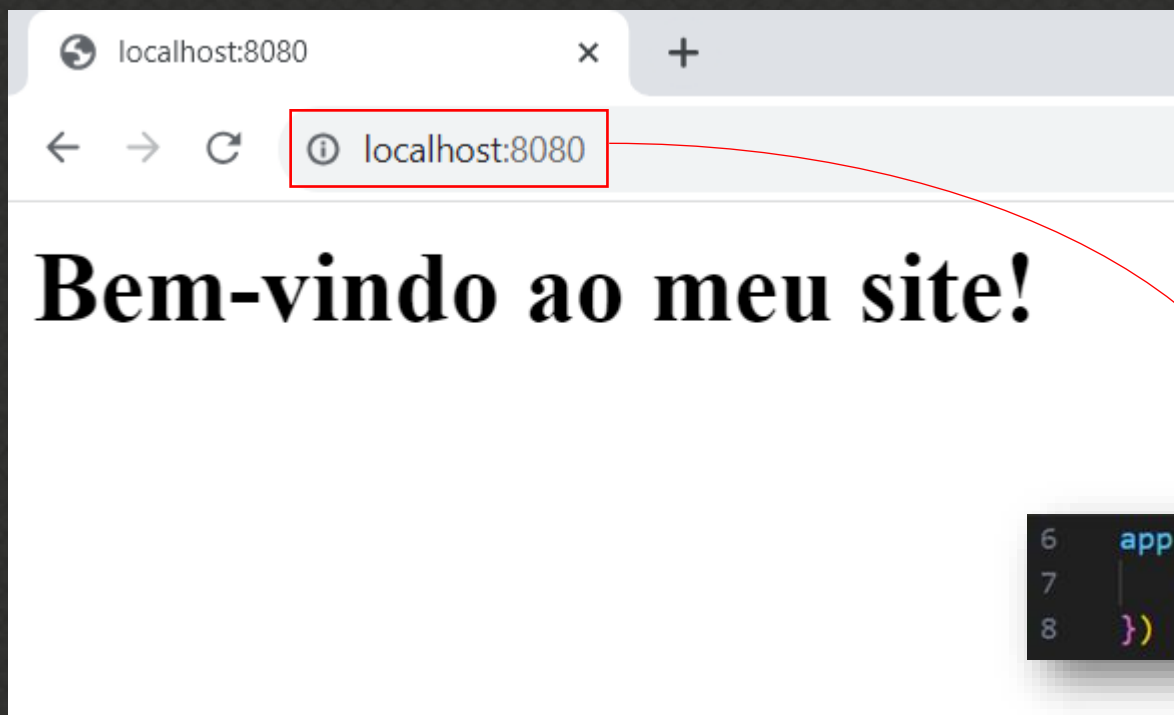
res: Resposta

No exemplo acima, no arquivo **index.js** criamos duas rotas. A rota principal (“/”) e a rota **perfil**. Em seguida, através do objeto **res** utilizamos o método *send*, para enviar uma resposta no momento em que a rota é acessada.

JS

Acessando as rotas

Para testarmos nossas rotas, basta abrir o navegador e através da barra de endereço digitar o endereço da rota. Lembre-se que com um servidor local, utilizamos primeiramente o localhost, seguido da porta.



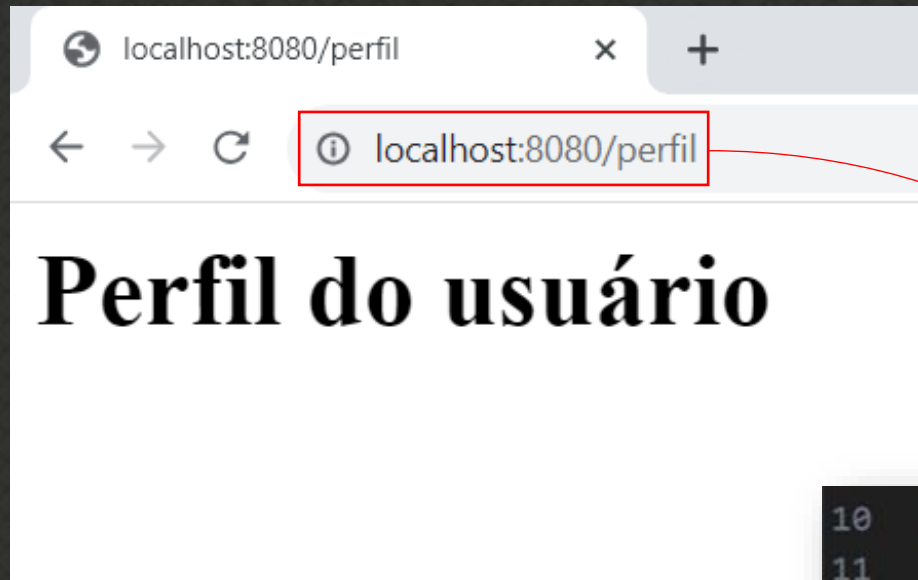
```
6 app.get("/", function(req, res){  
7   res.send("<h1>Bem-vindo ao meu site!</h1>")  
8 })
```



JS

Acessando as rotas

Nossa segunda rota também pode ser acessada conforme a seguir:



```
10 app.get("/perfil", function(req, res){  
11   res.send("<h1>Perfil do usuário</h1>")  
12 })
```



JS

View engine - EJS



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

Basicamente, uma Template Engine serve para facilitar a **criação de páginas HTML** e tornar o envio e exibição de informações para estas páginas um processo mais simples e organizado.



JS

Instalando o EJS



Para instalarmos o EJS, basta utilizarmos o comando **npm install ejs --save** no console.

```
● PS C:\Users\Prof. Diego\OneDrive - Etec Centro Paula Souza\Node.js\node> npm install ejs --save  
  
up to date, audited 75 packages in 799ms  
  
10 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities  
○ PS C:\Users\Prof. Diego\OneDrive - Etec Centro Paula Souza\Node.js\node> █
```

JS

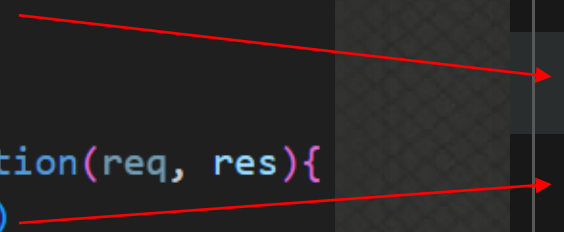
Renderizando páginas com EJS

O primeiro passo para começar a renderizar páginas com EJS, é indicar ao Express através do método **set**, qual view engine iremos utilizar, no caso a **ejs**:

```
4 app.set('view engine', 'ejs')
```

Agora em nossas rotas, já podemos renderizar páginas que possuem o formato de arquivo **.ejs**, através do método **render**. Para isso, essas páginas devem ser criadas dentro de uma pasta com nome **views** dentro da pasta do seu projeto.

```
app.get("/", function(req, res){  
  res.render("index")  
})  
  
app.get("/perfil", function(req, res){  
  res.render("perfil")  
})
```



```
▼ views  
  <> index.ejs  
  <> perfil.ejs
```

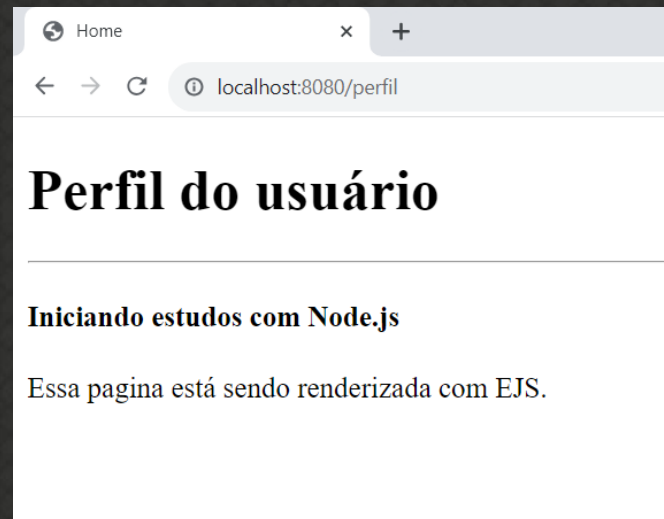
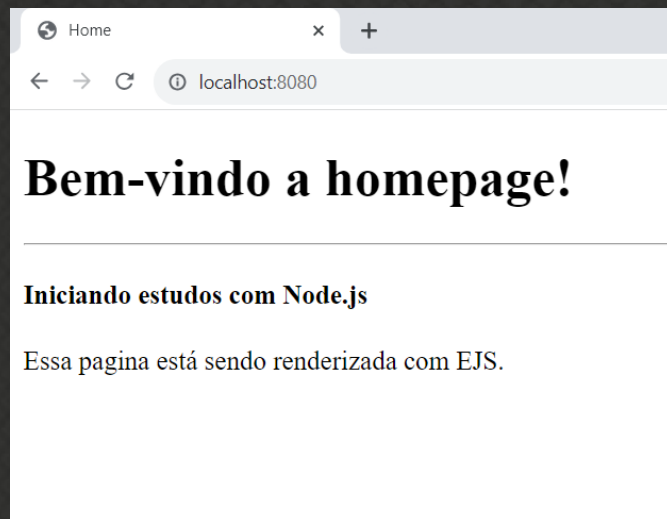


JS

Renderizando páginas com EJS

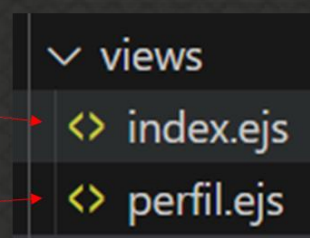


Agora, ao acessarmos nossas rotas o EJS irá renderizar as páginas que indicamos.



```
app.get("/", function(req, res){
  res.render("index")
})

app.get("/perfil", function(req, res){
  res.render("perfil")
})
```



JS

Concluindo...



Ao final nosso arquivo **index.js** terá a seguinte estrutura:

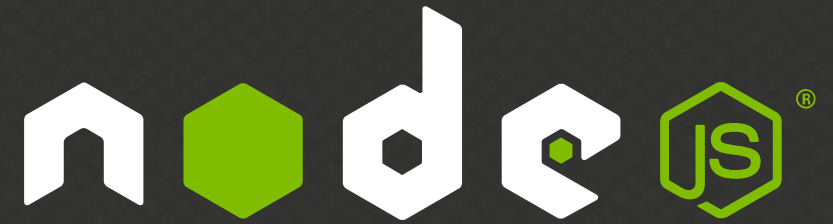
```
Node.js > NODE > JS index.js > ...
1  const express = require("express") //Importando o Express
2  const app = express() // Iniciando o Express
3
4  app.set('view engine', 'ejs')
5
6  app.get("/", function(req, res){
7    res.render("index")
8  })
9
10 app.get("/perfil", function(req, res){
11   res.render("perfil")
12 })
13
14 app.listen(8080, function(erro){
15   if(erro){
16     console.log("Ocorreu um erro!")
17   } else {
18     console.log("Servidor iniciado com sucesso!")
19   }
20 })
```

1 - Nele iniciamos um servidor Node na porta 8080.

2 - Criamos uma rota principal e uma rota com o nome perfil.

3 - Nessas rotas, renderizamos duas páginas index.ejs e perfil.ejs.

JS



Aula 06:

Introdução ao Node.js
e ao Express.js