



Parte teórica

Conceito cliente-server (Cliente-Servidor)

O Servidor é uma aplicação com diversos serviços que podem ser requisitados por outras aplicações.

O cliente é uma aplicação capaz de requisitar serviços a uma aplicação servidora. Geralmente uma aplicação cliente e aplicação servidora estão em instaladas em computadores distintos e se comunicam por uma rede. Entretanto, por se tratarem de aplicações, podem perfeitamente coexistir no mesmo computador.

Para que seja possível estabelecer a comunicação a aplicação cliente e aplicação servidor, é necessário estabelecer um padrão (protocolo) de aplicação

Em aplicação web, do lado do cliente estão os browsers. Do lado do servidor, estão os servidores web. Como o apache, tomcat, nodeJs entre outros.

Dessa forma a aplicação cliente (o browser) é capaz de processar a resposta e exibir os dados em tela.

Tecnologias para o desenvolvimento Web

Codificação do Cliente – front-end

Codificação do Servidor – back-end

Banco de dados

Que plataforma / framework iremos utilizar nessa disciplina?

Plataforma: Node.js

Framework: Express

O que é o Node.JS?

É uma plataforma construída sobre o motor JavaScript do Google Chrome para facilmente construir aplicações de rede rápidas e escaláveis.

Node.js usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos.

Programação orientada a Evento

Node utiliza o que é chamado modelo de programação orientada a evento.

Uma conexão é feita – evento! Dado é recebido através da conexão – evento! Dado parou de chegar através da conexão – evento!

As funções de callback que são chamadas quando um evento ocorre podem ser escritas no mesmo lugar onde você captura o evento. Fácil para desenvolver, fácil para manter. Sem frameworks.

Resumindo:

Node.js é uma plataforma para desenvolvimento back-end – com linguagem JavaScript. É um interpretador de códigos JavaScript

JavaScript foi criada para ser executada dentro do próprio navegador. O tradicional é utilizarmos JavaScript para rodar no Front-end. Para o Back-end, utilizava-se PHP.

Do ponto de vista do desenvolvimento Web, o NodeJs implementa recursos capazes de executar protocolos como o HTTP.

Nasce aí uma nova técnica de aplicações Web. Apenas com uma única linguagem podemos agora criar aplicações front-end e back-end

O Node.js traz também um gerenciador de pacotes – npm – com isso a aplicação vai poder contar com milhares de soluções (bibliotecas).

OK. Mas o nome da disciplina é Desenvolvimento baseado em Framework.

Se Node.js é uma plataforma, qual framework iremos utilizar?

Express

O Express é um framework para aplicações web sob a plataforma Node.js.

O Express.js cria abstrações de rotas, middlewares e muitas outras funções para facilitar a criação de aplicações web.

Express é um framework Back-end, ou seja Express está voltado para a criação e obtenção dos dados a partir do seu servidor.

Outros conceitos a serem estudados

- padrão de desenvolvimento MVC
 - formato Json - função de call-back
 - Abstrações de rotas
 - Serviços Rest
 - Protocolo HTTP
-

O que é REST?

“O termo transferência de estado representacional – **REST (Representational State Transfer)** foi introduzido e definido no ano de **2000** através de uma **tese de Ph.D** do cientista **Roy Fielding**, um dos **principais autores** da especificação do protocolo **HTTP**”

“O intuito geral da tese era a **formalização** de um conjunto de **melhores práticas** denominadas **constraints**”

“Essas **constraints** tinham como objetivo determinar a forma na qual **aplicações web** devem ser modeladas.

Constraints

Cliente-Servidor

Constraints / Cliente-Servidor

“A principal característica dessa constraint é **separar as responsabilidades** de diferentes partes de um sistema.”

“Essa divisão pode se dar por exemplo com uma separação entre mecanismos da **interface do usuário** e o **back-end** da aplicação.”

“Isso nos permite a evolução e **escalabilidade** destas responsabilidades **de forma independente.**”

Constraints

Stateless

Constraints / Stateless

“Essa característica propõe que **cada requisição** ao servidor **não deve ter ligação com requisições anteriores ou futuras**, ou seja, cada requisição deve conter todas as informações necessárias para que ela seja tratada com sucesso pelo servidor.”

Constraints

Cache

Constraints / Cache

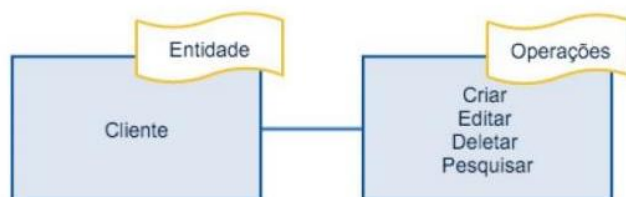
“Para uma melhor performance, um sistema **REST** deve **permitir** que suas respostas sejam passíveis de **cache**.”

Constraints

Interface Uniforme

“Bastante esforço deve ser feito para que o sistema possua uma **interface modelada** seguindo alguns padrões importantes.”

“Devemos pensar em criar uma interface que permita a manipulação desses conceitos. Veja o exemplo:”

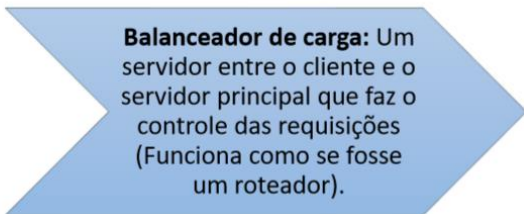


Constraints

Sistema em Camadas

“Com o intuito de permitir a escalabilidade necessária para grandes sistemas distribuídos, um sistema REST deve ter a capacidade de adicionar elementos intermediários e que sejam totalmente transparentes para seus clientes.”

Ex: Balanceador de Carga



Balanceador de carga: Um servidor entre o cliente e o servidor principal que faz o controle das requisições (Funciona como se fosse um roteador).

REST vs RESTful

“Quando estamos discutindo sobre o modelo e sobre as características que nós vimos anteriormente, você deve utilizar o termo **REST**, já no momento em que você estiver falando de uma implementação que usa essas mesmas características, você deve usar **RESTful**”

“**REST** nada mais é que um conjunto de melhores práticas denominadas **constraints**”

“Ou seja, se temos uma API que não segue os **princípios REST**, teremos apenas uma API HTTP”

Material extraído do curso Entendendo e Documentando RESTfull API's do professor Jackson Pires

Referências:

Node JS - <http://nodebr.com/o-que-e-node-js/>

Express - <http://expressjs.com/pt-br/>

Apostila do Prof. Ricardo Augusto Lins do Nascimento (IFMS).

Parte prática

Instalação do NodeJs

Baixar o Node.Js no site:

<https://nodejs.org/en/>

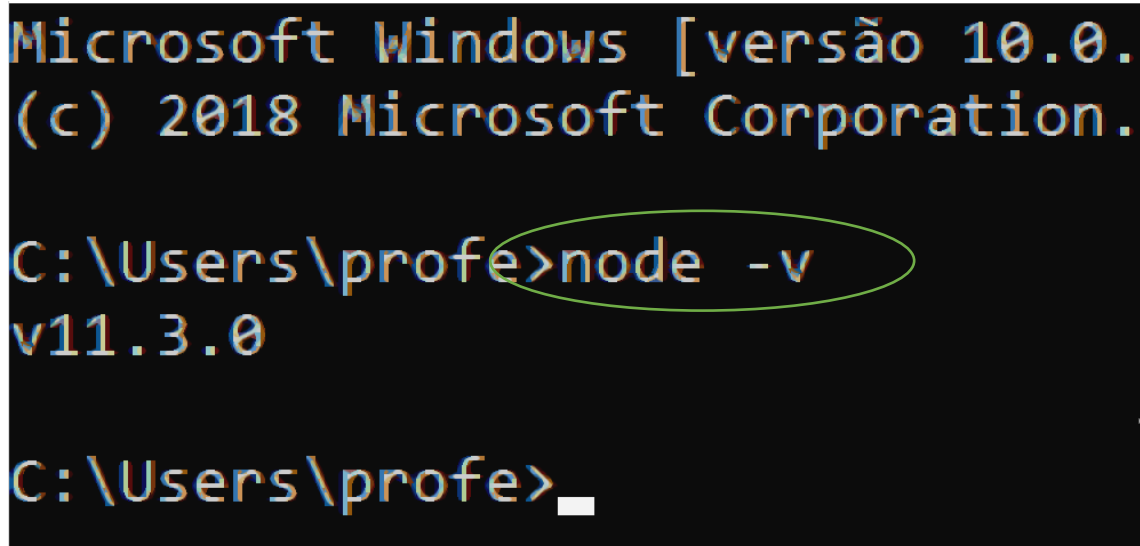
Fazer a instalação padrão, com todos os pacotes

Depois da instalação, testar:

Abrir o prompt de comando:

Digitar:

node -v (vai mostrar a versão do node instalada)

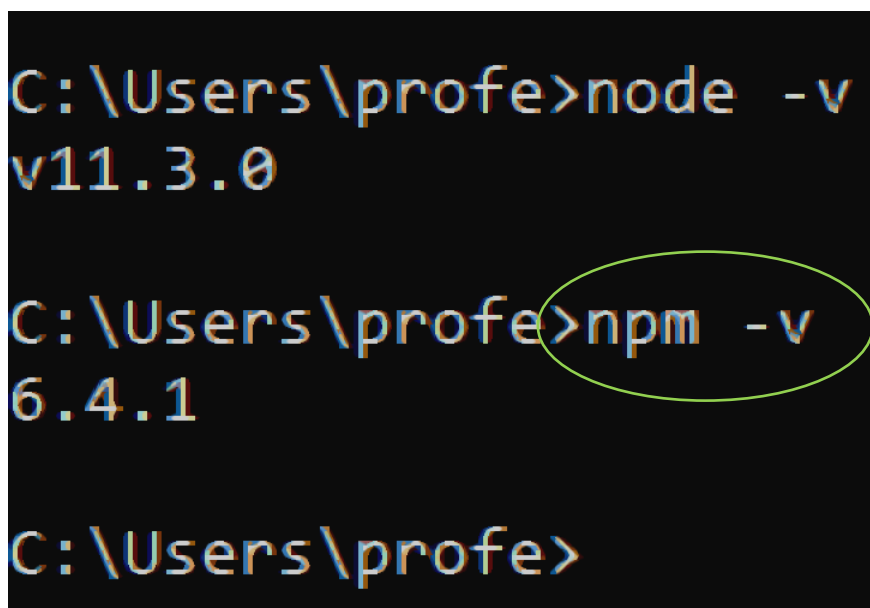


```
Microsoft Windows [versão 10.0.
(c) 2018 Microsoft Corporation.

C:\Users\profe>node -v
v11.3.0

C:\Users\profe>
```

npm -v (vai mostrar a versão do npm instalada)



```
C:\Users\profe>node -v
v11.3.0

C:\Users\profe>npm -v
6.4.1

C:\Users\profe>
```

para habilitar o node, digitar:

node

podemos executar comandos javascript

para fechar, digitar:

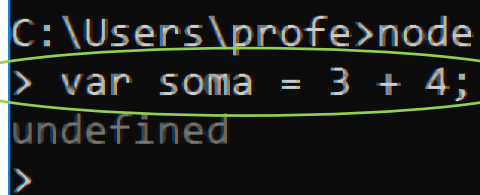
ctrl + c

ou node.exit

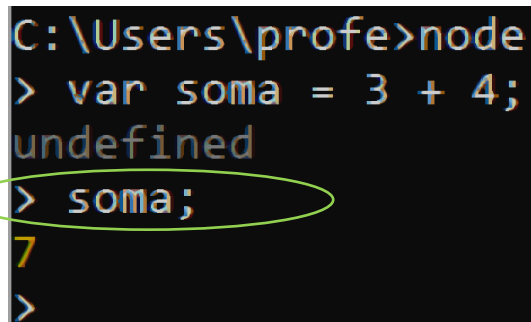
Este comando (comando node) vai fazer entrar no interpretador do Node aqui no terminal. A partir de agora podemos trabalhar com JavaScript no terminal.

Exemplo:

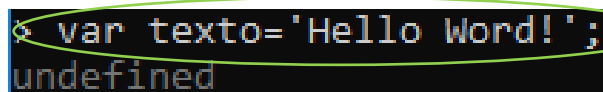
var soma = 3 + 4;



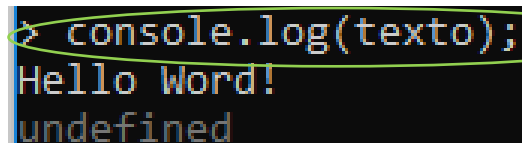
```
C:\Users\profe>node
> var soma = 3 + 4;
undefined
>
```



```
C:\Users\profe>node
> var soma = 3 + 4;
undefined
> soma;
7
>
```



```
> var texto='Hello Word!';
undefined
```



```
> console.log(texto);
Hello Word!
undefined
```

- Podemos criar laços de repetição:

```
> for (var k=0; k<10; k++){
```

No momento em que colocamos a chave e damos enter, ela vai colocar 3 pontinhos (...). Isso significa que estamos dentro do for.

```

Prompt de Comando - node
Microsoft Windows [versão 10.0.16299.547]
(c) 2017 Microsoft Corporation. Todos os direitos reservados.

C:\Users\profe>node
> for (var k=0; k<10; k++){
... _

```

- Podemos criar laços de repetição:

```
> for (var k=0; k<10; k++){
... console.log(k);
```

Para imprimir o valor de k. Ao darmos um enter, podemos perceber que ele tem mais três pontinhos (...), o que significa que ainda estamos implementando dentro do for. Na sequência, podemos colocar as chaves para fechar }

No momento que colocamos a chave para fechar e damos um enter, ele vai executar o for.

```
...
```

```
}
```

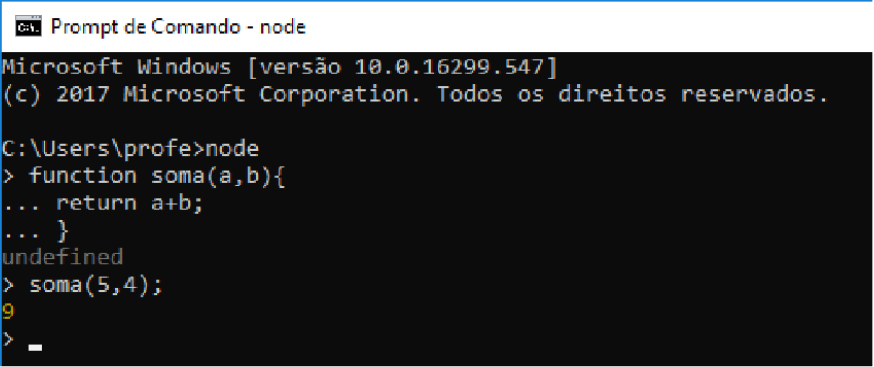
```
> for (k=0; k<10; k++){  
... console.log(k);  
... }  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
undefined  
>
```

- Para finalizar, façamos mais um exemplo bem básico. Vamos criar uma função utilizando JavaScript

```
> Function soma(a,b){  
... return a +b;  
...}
```

```
Prompt de Comando - node  
Microsoft Windows [versão 10.0.16299.547]  
(c) 2017 Microsoft Corporation. Todos os direitos reservados.  
C:\Users\profe>node  
> function soma(a,b){  
... return a+b;  
... }  
undefined  
>
```

- Já temos a nossa função soma. Agora podemos executá-la.
- > Soma (5,4);



```

Prompt de Comando - node

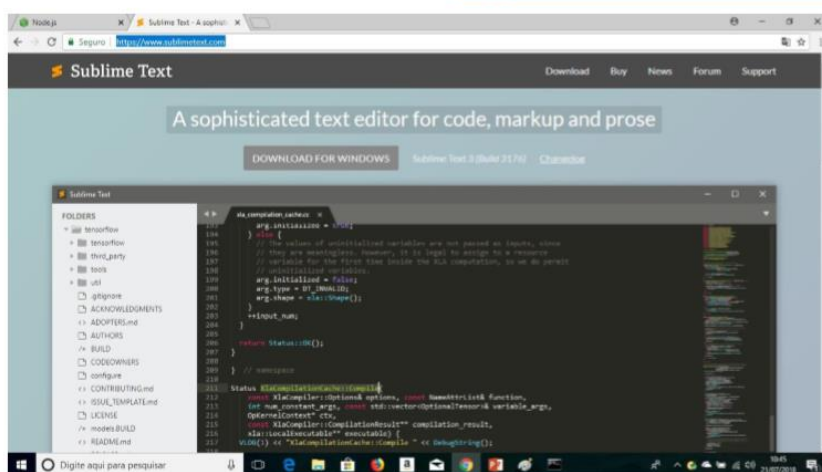
Microsoft Windows [versão 10.0.16299.547]
(c) 2017 Microsoft Corporation. Todos os direitos reservados.

C:\Users\profe>node
> function soma(a,b){
... return a+b;
... }
undefined
> soma(5,4);
9
>

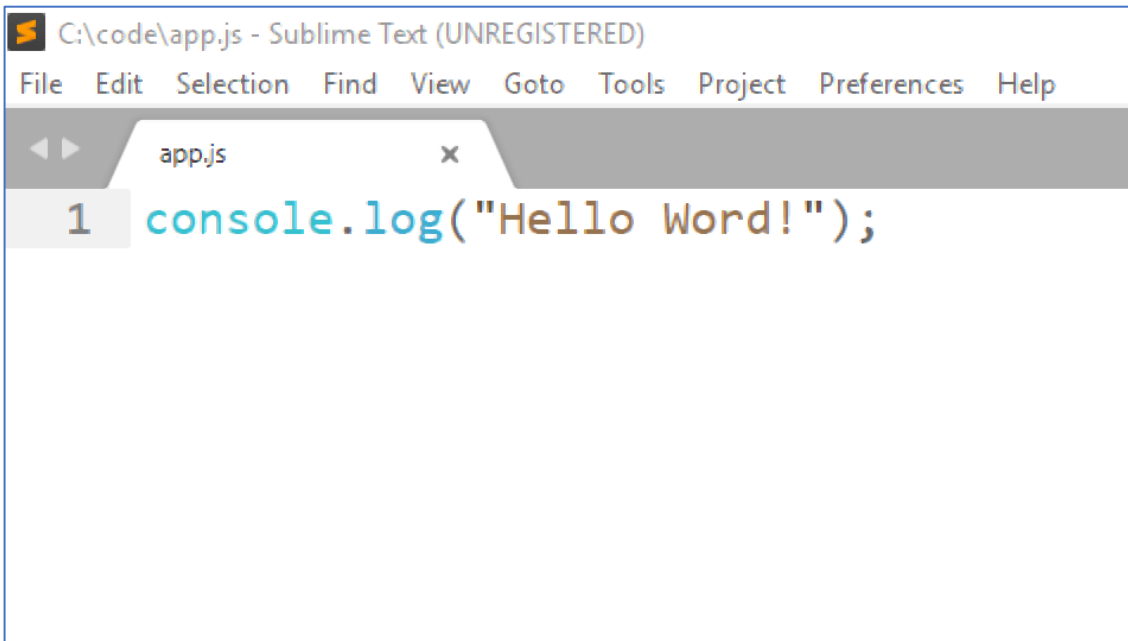
```

Trabalhando com código escrito em arquivo

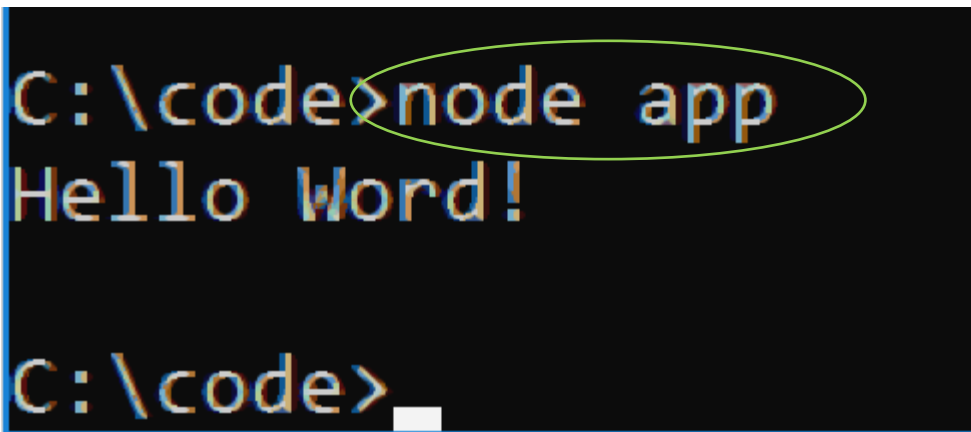
- Precisamos de um editor de texto (sugestão: Sublime text).
- Você pode baixa-lo no site: <https://www.sublimetext.com/>



- Após a instalação do sublime, crie uma pasta no diretório c da máquina.
- Sugestão:
c:\code
- Abre o sublime e crie um arquivo com o nome app.js
- vamos fazer um teste para verificarmos se o Node vai rodar o nosso arquivo app.js.



```
C:\code\app.js - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
app.js
1 console.log("Hello Word!");
```



```
C:\code>node app
Hello Word!
C:\code>
```

Começando pra valer!

Como alguém utilizando um browser conseguiria consumir dados do servidor?

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.

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

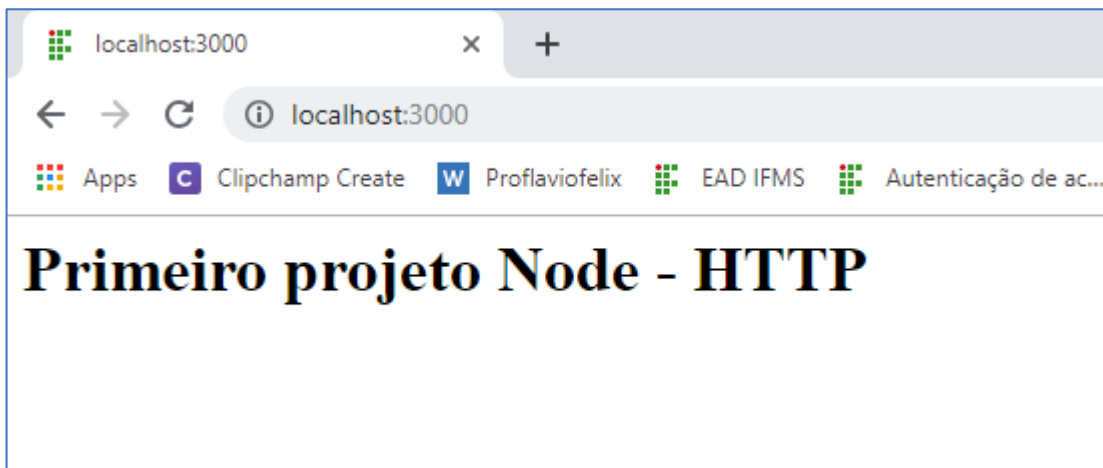
```
var http = require('http');
```

Nesse caso estamos incorporando a biblioteca http

Em seguida, precisamos criar (subir) um servidor

no Node nós vamos utilizar bastante funções como argumentos.

```
1 var http = require('http');
2
3 var server = http.createServer(function(req,res){
4   res.end("<html><body><h1> Primeiro projeto Node - HTTP </h1></body></html>");
5 });
6
7 server.listen(3000);
```



Linha1 – criamos uma variável http recebendo a biblioteca http.

Linha 3 – criamos a variável server que vai receber o método createServer da variável http, Esse método tem como parâmetro do uma função com dois parâmetros: req e res.

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

A função res vai fornecer a resposta.

res.end é o fim da resposta. No caso, estamos devolvendo um código html.

Estamos colocando o código direto aqui, em outras reposta devolveremos arquivos html.

No final, informamos ao servidor qual é a porta que ele está escutando. No caso utilizaremos aqui a porta 3000

O método server.listen(3000) informa que estará escutando a porta 3000.

Vamos entender agora como funcionam as requisições

```
1 var http = require('http');
2
3 var server = http.createServer(function(req, res){
4   var categoria = req.url;
5   if (categoria == '/cursos'){
6     res.end("<html><body><h1> IFMS - Notícias de Cursos </h1></body></html>");
7   } else if (categoria == '/esportes'){
8     res.end("<html><body><h1>IFMS - Notícias de Esportes</h1></body></html>");
9
10  } else if (categoria == '/pesquisa'){
11    res.end("<html><body><h1>IFMS - Notícias de Pesquisa</h1></body></html>");
12  }
13  else {
14    res.end("<html><body><h1>Portal de notícias IFMS </h1></body></html>");
15  }
16 });
17
18 server.listen(3000);
```

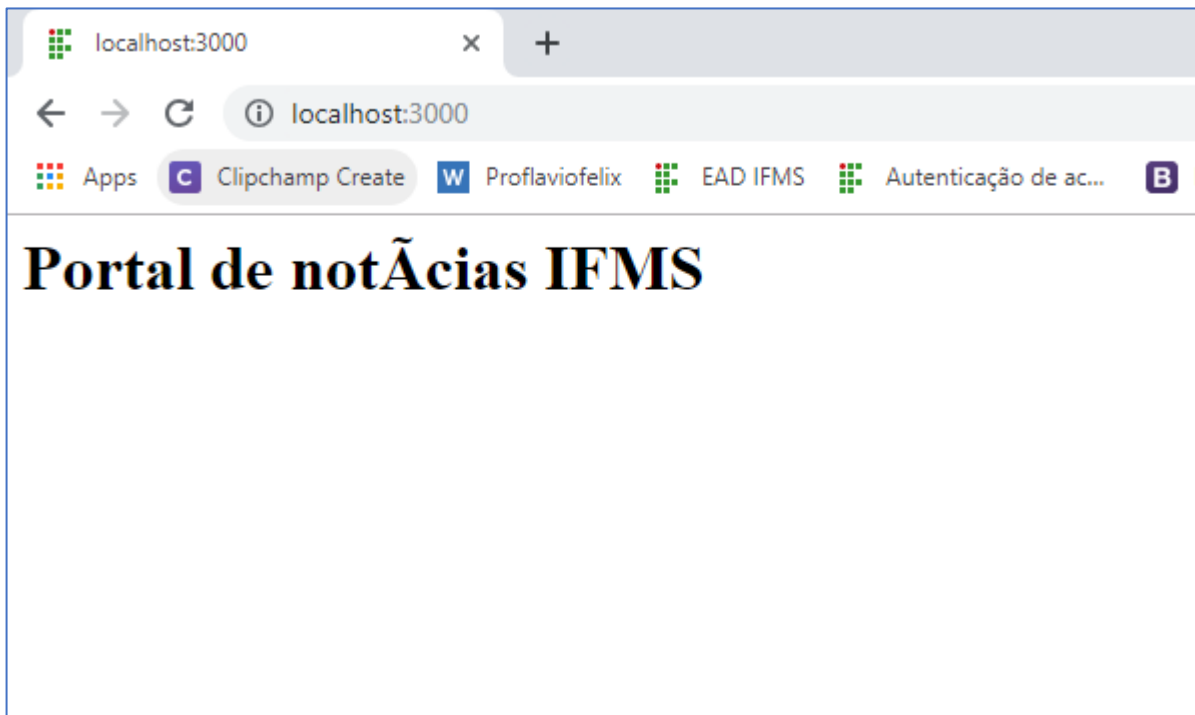
Criamos a variável categoria, que vai receber a url requisitada pelo cliente.

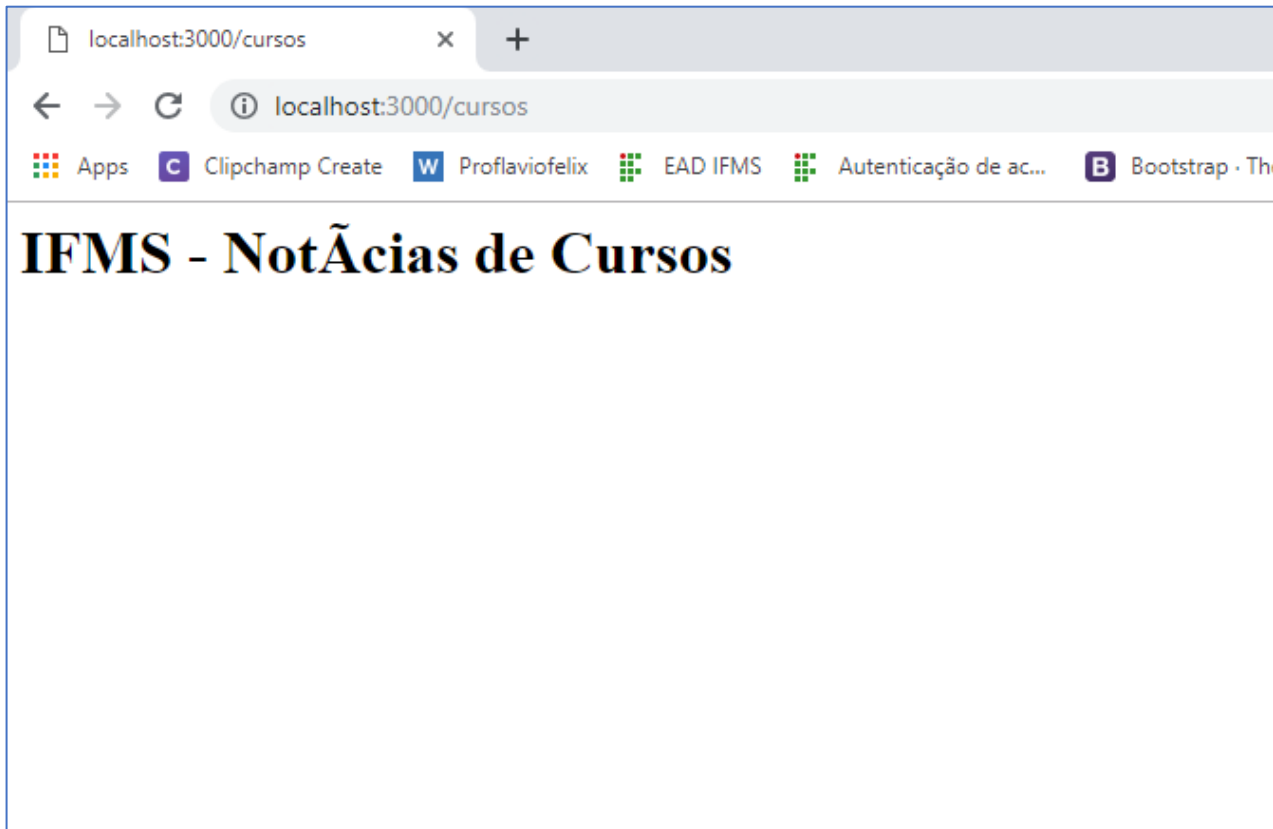
De acordo com a url, a resposta será o código url referente a requisição

Caso o usuário informe uma url inexistente, vai direcionar para o html padrão.

Para visualizarmos no console do servidor (não no navegador) para que o administrador saiba qual porta o servidor está escutando, podemos inserir uma mensagem da seguinte forma.


```
1 var http = require('http');
2
3 var server = http.createServer(function(req,res){
4   var categoria = req.url;
5   if (categoria == '/cursos'){
6     res.end("<html><body><h1> IFMS - Notícias de Cursos </h1></body></html>");
7   } else if (categoria == '/esportes'){
8     res.end("<html><body><h1>IFMS - Notícias de Esportes</h1></body></html>");
9
10  } else if (categoria == '/pesquisa'){
11    res.end("<html><body><h1>IFMS - Notícias de Pesquisa</h1></body></html>");
12  }
13  else {
14    res.end("<html><body><h1>Portal de notícias IFMS </h1></body></html>");
15  }
16 });
17
18 server.listen(3000);
19 console.log("Escutando a porta 3000");
```





Quatro recursos que serão utilizados no curso

NPM – Um Gerenciador de pacotes JavaScript.

Express – Framework NodeJs para aplicações WEB – Ele já implementa uma API diversos métodos para controle de rotas, controle de módulos entre requisição e resposta, fornece diversos recursos para agilizar o desenvolvimento.

EJS – é uma linguagem de modelagem para criação de páginas HTML utilizando JavaScript.

Nodemon – Um utilitário que reinicia automaticamente o servidor NodeJS quando houver qualquer alteração em nossos script.

Iniciando o NPM

Como o npm é o gerenciador de pacotes (os outros utilitários serão instalados a partir dele) então é o primeiro utilitário que vamos instalar

Dentro do diretório onde criaremos nosso projeto, vamos iniciar o npm com o seguinte comando:

```
npm init
```

Uma tela de configurações será exibida:

```
C:\code>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 json` 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.

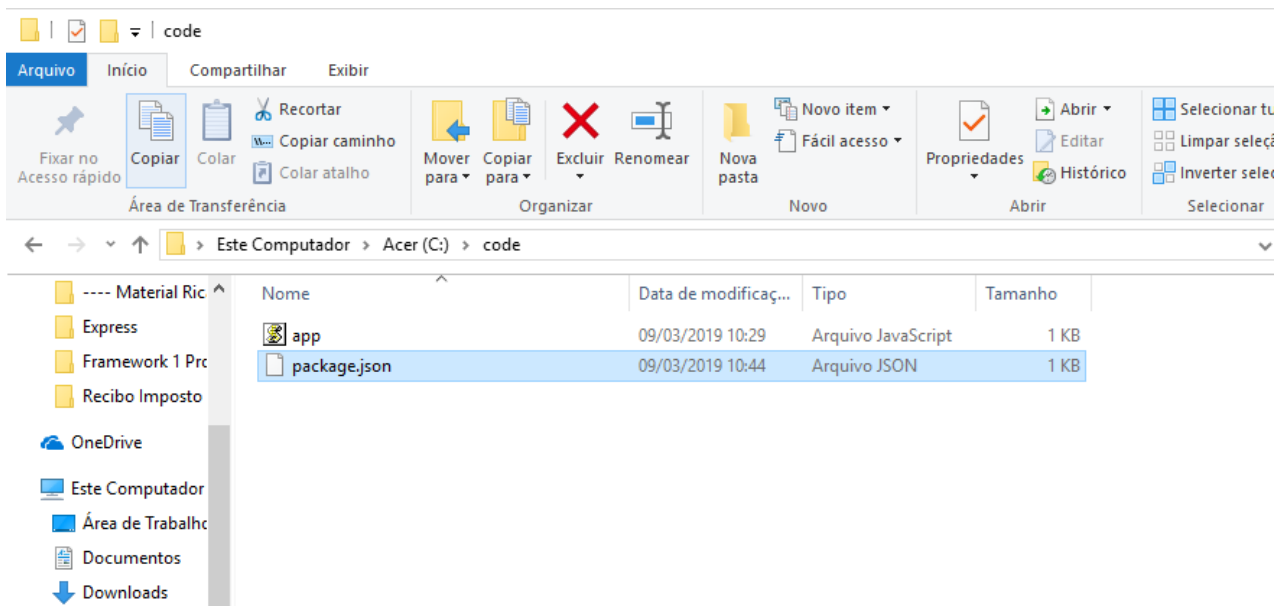
Press ^C at any time to quit.
package name: (code) noticias_ifms
version: (1.0.0)
description: site de noticias do ifms
entry point: (app.js)
test command:
git repository:
keywords:
author: flavio felix medeiros
license: (ISC)
About to write to C:\code\package.json:

{
  "name": "noticias_ifms",
  "version": "1.0.0",
  "description": "site de noticias do ifms",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "flavio felix medeiros",
  "license": "ISC"
}

Is this OK? (yes) yes

C:\code>
```

Após a instalação, observamos que um arquivo package.json foi criado.



Podemos abrir o package.json usando o sublime.

```

1 {
2   "name": "noticias_ifms",
3   "version": "1.0.0",
4   "description": "site de noticias do ifms",
5   "main": "app.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "flavio felix medeiros",
10  "license": "ISC"
11 }
12

```

- formato Json - JSON (JavaScript Object Notation) é um modelo para armazenamento e transmissão de informações no formato texto. Apesar de muito simples, tem sido bastante utilizado por aplicações Web devido a sua capacidade de estruturar informações de uma forma bem mais compacta do que a conseguida pelo modelo XML, tornando mais rápido o parsing dessas informações. Isto explica o fato de o JSON ter sido adotado por empresas como Google e Yahoo, cujas aplicações precisam transmitir grandes volumes de dados.

Referência: <https://www.devmedia.com.br/introducao-ao-formato-json/25275>

Instalação do express

`npm install express --save`

utilizando o `--save`, vai trazer os arquivos de fato para dentro do projeto. O express vai estar inserido como um modulo dentro da aplicação

Após a instalação vamos verificar que foi criada a pasta `node-modules`, com diversos recursos do express

Ao abrir o `package.json` vemos que o express também está informado como dependência, com a versão instalada.

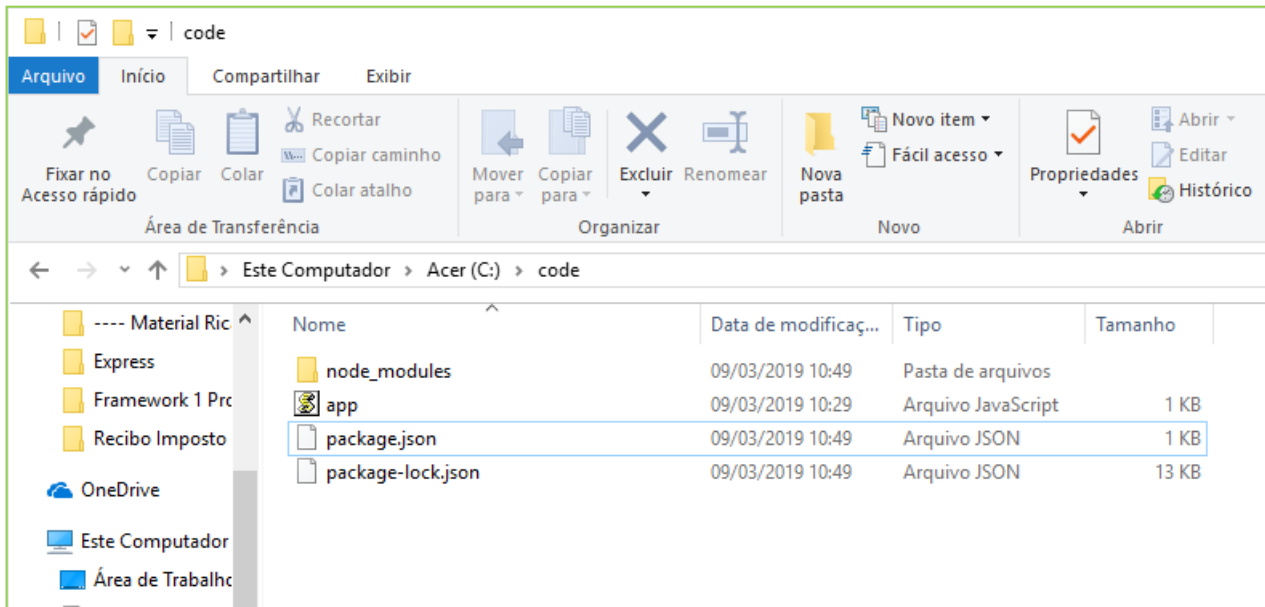
```
{
  "name": "noticias_ifms",
  "version": "1.0.0",
  "description": "site de noticias do ifms",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "flavio felix medeiros",
  "license": "ISC"
}
```

Is this OK? (yes) yes

```
C:\code>npm install express -save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN noticias_ifms@1.0.0 No repository field.

+ express@4.16.4
added 48 packages from 36 contributors and audited 121 packages in 13.053s
found 0 vulnerabilities

C:\code>
```



```

1 {
2   "name": "noticias_ifms",
3   "version": "1.0.0",
4   "description": "site de noticias do ifms",
5   "main": "app.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "flavio felix medeiros",
10  "license": "ISC",
11  "dependencies": {
12    "express": "^4.16.4"
13  }
14 }
15

```

Vamos ajustar a codificação com base no express

(Refactoring do projeto para o formato Express)

Lembrando que:

o Node é uma plataforma que executa códigos JavaScript

Express é um framework (é uma camada que vai acima do node) para fazer uma interface entre os nossos scripts e no node. É necessário que os nossos códigos estejam dentro da estrutura que o express espera.

Quais são essas alterações?

1 – Criar um arquivo chamado app.js

2 - Ao invés de fazer um require do módulo http, vamos fazer um require do express

O Módulo express, retorna uma função (não executa, mas retorna), precisamos fazer a chamada da função:

Com o node puramente fazíamos a chamada do CreateServer.

Com o Express vamos chamar o método listen, (vai ficar escutando uma porta). Precisamos criar uma função de call-back

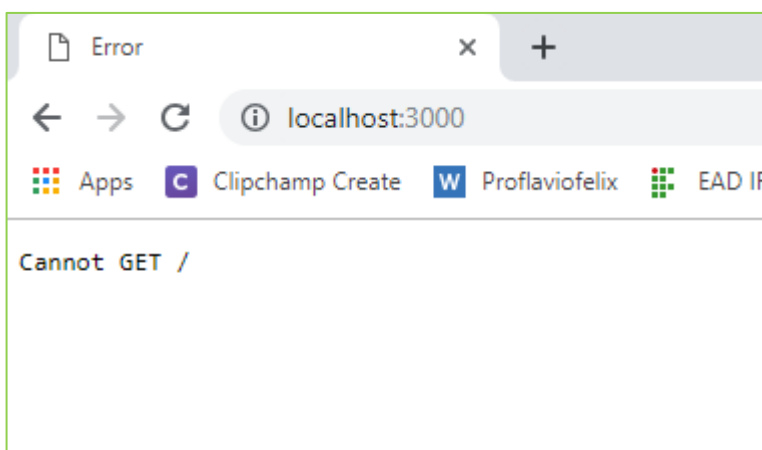
Vamos colocar algo para verificar se o servidor subiu e se está escutando na porta 3000.

```
app2.js x
1 var express = require('express');
2
3 var app = express();
4
5 app.listen(3000, function(){
6   console.log("Servidor rodando com Express");
7 });
```

Ao executar o node app.js, temos a seguinte resposta:

```
C:\code>node app2
Servidor rodando com Express
```

Verificando a aplicação no browser, teremos o seguinte resultado:



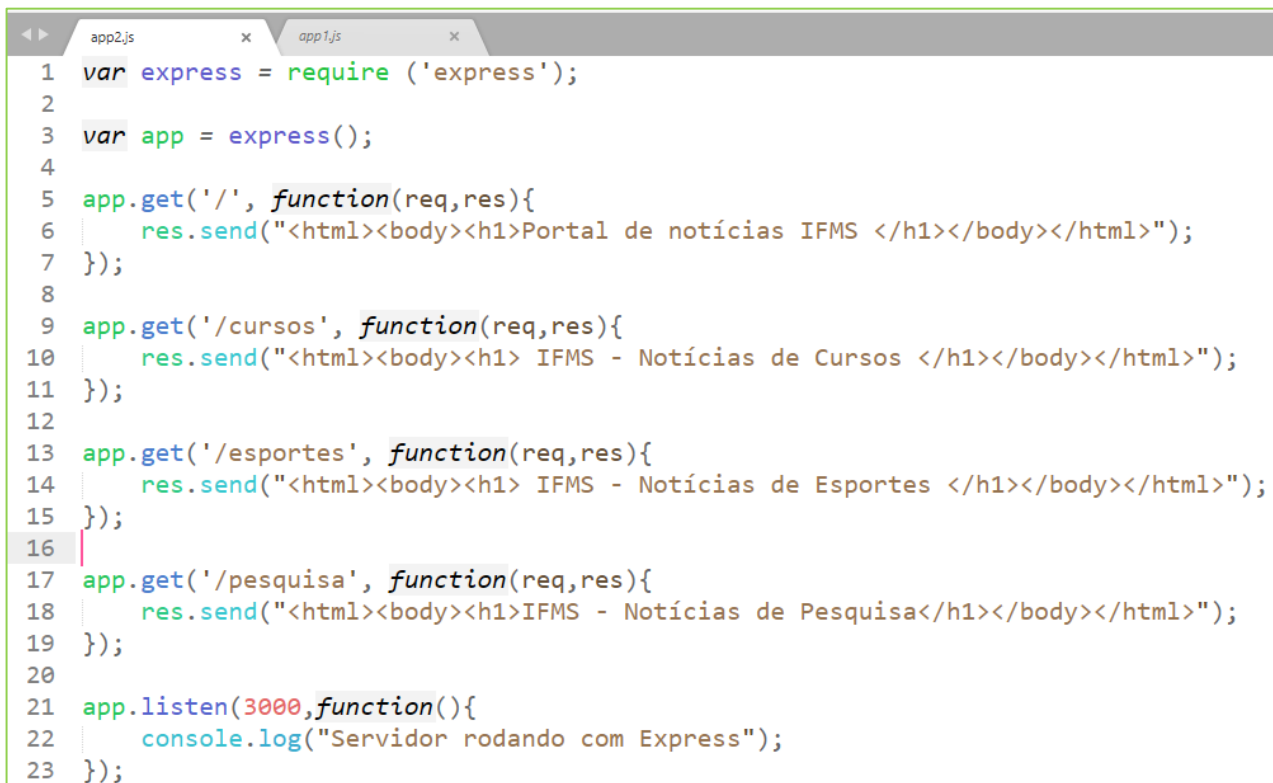
O que significa Cannot Get?

Significa que o Express já está tratando e identificou que não existe nenhuma resposta para ser dada ao endereço / (Raiz)

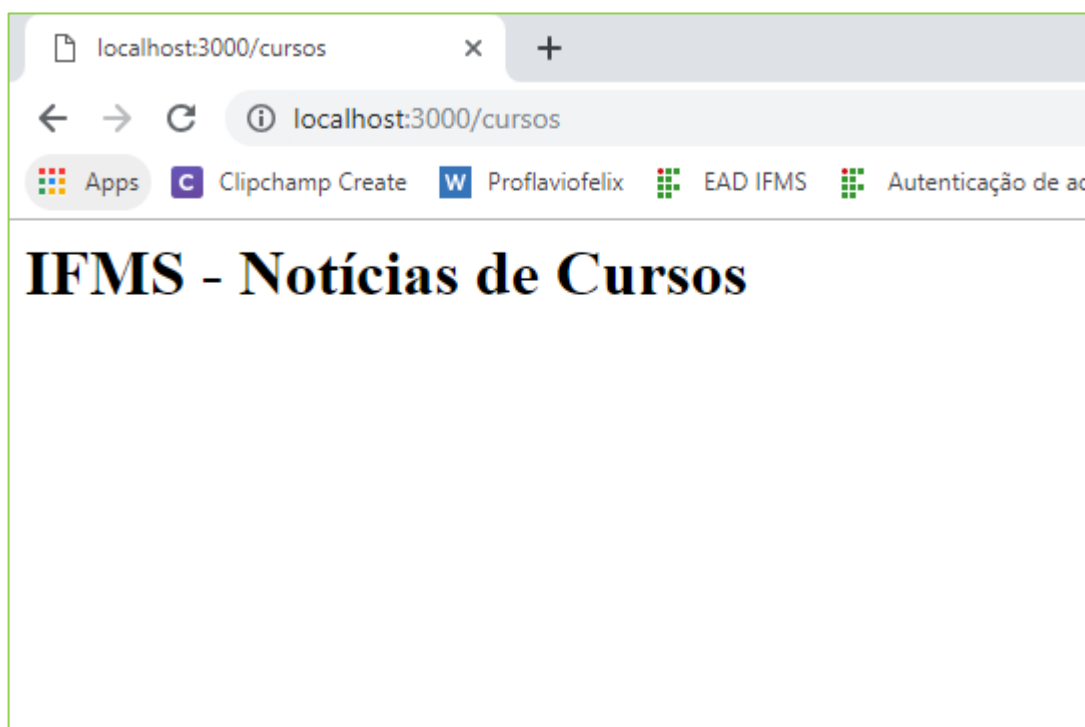
Já vamos ver algumas vantagens do express:

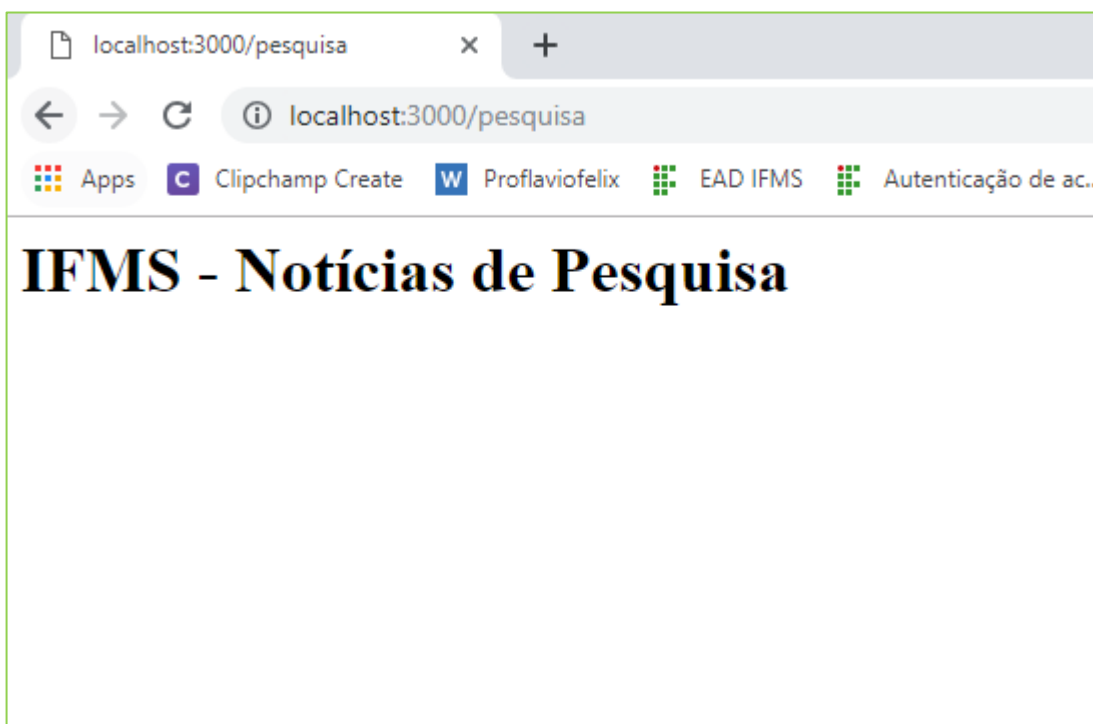
Vamos utilizar a função get do express:

Ao invés de utilizar o método end para retornar, vamos utilizar o metodo send.



```
1 var express = require('express');
2
3 var app = express();
4
5 app.get('/', function(req,res){
6   res.send("<html><body><h1>Portal de notícias IFMS </h1></body></html>");
7 });
8
9 app.get('/cursos', function(req,res){
10   res.send("<html><body><h1> IFMS - Notícias de Cursos </h1></body></html>");
11 });
12
13 app.get('/esportes', function(req,res){
14   res.send("<html><body><h1> IFMS - Notícias de Esportes </h1></body></html>");
15 });
16
17 app.get('/pesquisa', function(req,res){
18   res.send("<html><body><h1>IFMS - Notícias de Pesquisa</h1></body></html>");
19 });
20
21 app.listen(3000,function(){
22   console.log("Servidor rodando com Express");
23 });
```



Instalação do EJS – é um módulo que vai nos permitir escrever páginas html juntamente com javascript.

Comando para instalação do EJS:

npm install ejs -save É um Engine de views que vai permitir a criação de páginas de html dinâmicas.

```

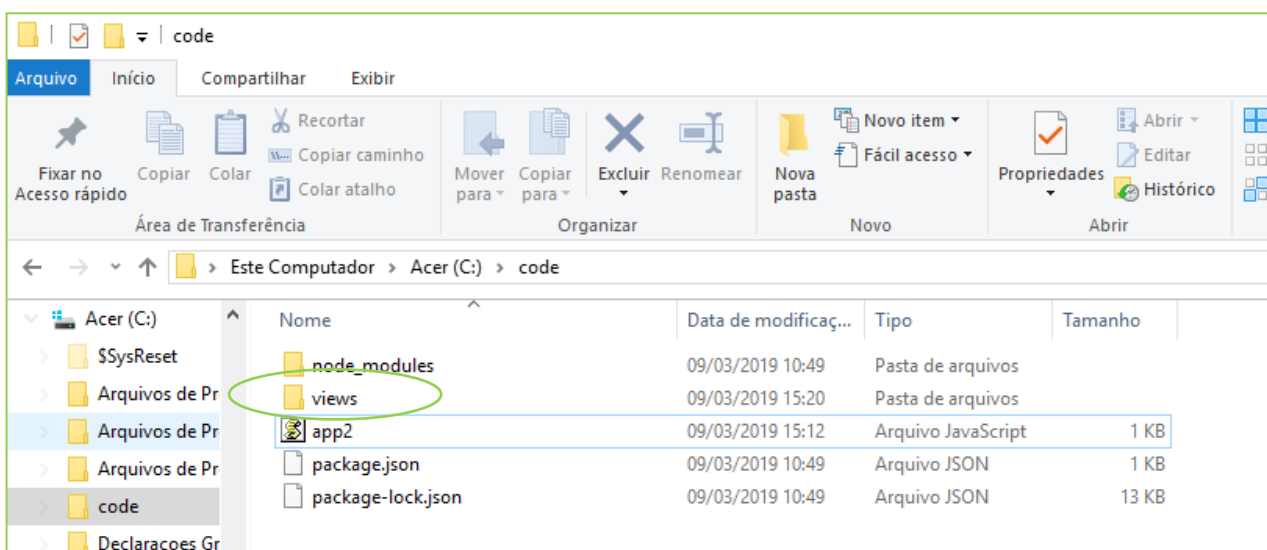
C:\code>node app2
Servidor rodando com Express
^C
C:\code>node app2
Servidor rodando com Express
^C
C:\code>npm install ejs -save

```

Vamos poder criar views separadamente e depois reinderizar o projeto.

Em seguida vamos informar para o Express que o Engine de views do Express passou a ser o módulo EJS

Agora vamos criar uma nova pasta chamada views



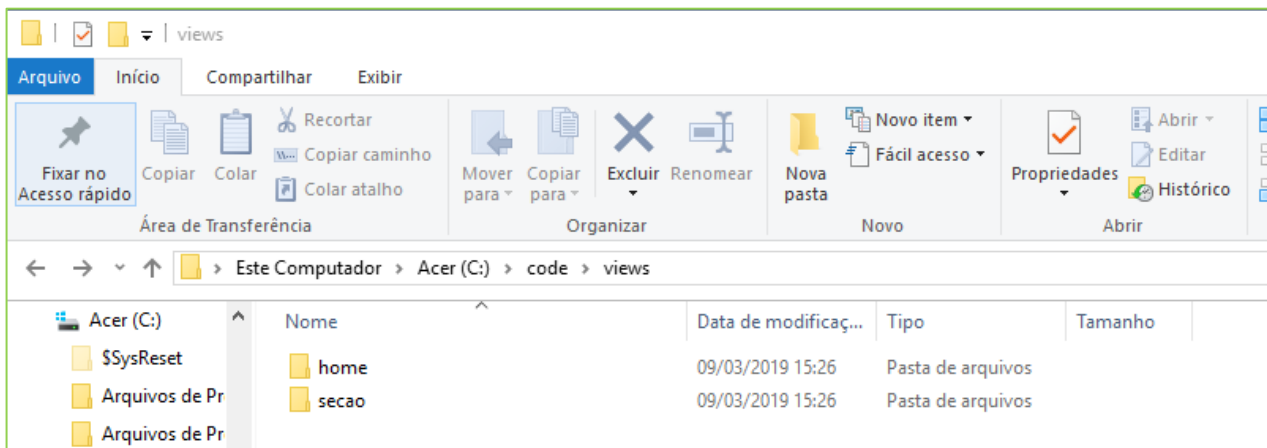
Dentro dos arquivos, que terão extensão ejs, vamos colocar arquivos com javascript e html .

Importante: O EJS vai sempre buscar os arquivos na pasta views. Para alterar o local, teria que alterar da seguinte maneira:

`app.set('views', '<novo_local>')` logo abaixo da instrução `app.set('view engine', 'ejs');` para alterar o local de busca das views.

Com o EJS, ao invés de usar o método End (node) ou Send (Express), vamos utilizar o `render` (EJS), que significa – reinderizar

Para darmos sequência no nosso projeto, vamos criar duas pastas dentro do nosso diretório de views (para organizar melhor nossas telas).



Dentro da pasta **home** vamos criar o arquivo **index.ejs**;

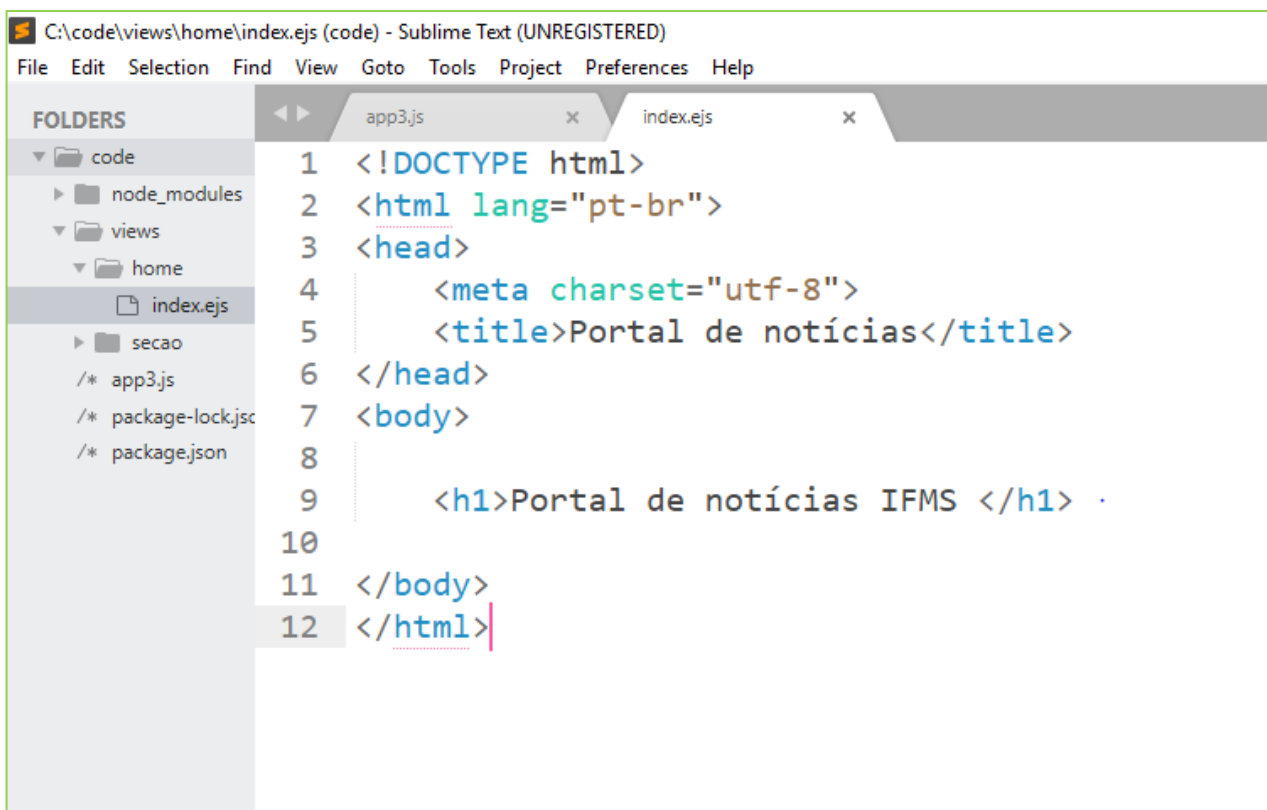
Dentro da pasta **secao** vamos criar os arquivos:

cursos.ejs;

esportes.ejs;

pesquisa.ejs;

Arquivo **index.ejs** que deverá ser criado dentro da pasta **home**:



Arquivo **curso.s.ejs** que deverá ser criado dentro da pasta **secao:**

```

C:\code\views\secao\curso.s.ejs (code) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ code
  └─ node_modules
    └─ views
      └─ home
        └─ index.ejs
      └─ secao
        └─ curso.s.ejs
        /* app3.js
        /* package-lock.js
        /* package.json

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4     <meta charset="utf-8">
5     <title>Portal de notícias</title>
6 </head>
7 <body>
8
9     <h1> IFMS - Notícias de Cursos </h1>
10
11 </body>
12 </html>
  
```

Arquivo **esportes.ejs** que deverá ser criado dentro da pasta **secao:**

```

C:\code\views\secao\esportes.ejs • (code) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ code
  └─ node_modules
    └─ views
      └─ home
        └─ index.ejs
      └─ secao
        └─ curso.s.ejs
        └─ esportes.ejs
        /* app3.js
        /* package-lock.js
        /* package.json

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4     <meta charset="utf-8">
5     <title>Portal de notícias</title>
6 </head>
7 <body>
8
9     <h1> IFMS - Notícias de Esportes </h1>
10
11 </body>
12 </html>
  
```

Arquivo **pesquisa.ejs** que deverá ser criado dentro da pasta **secao**:

The screenshot shows the Sublime Text editor with the file explorer on the left. The file explorer shows a project structure with folders 'code', 'node_modules', and 'views'. Inside 'views', there are folders 'home' and 'secao'. The 'secao' folder contains files 'index.ejs', 'cursos.ejs', 'esportes.ejs', and 'pesquisa.ejs'. The 'pesquisa.ejs' file is selected. The main editor area shows the content of 'pesquisa.ejs' with line numbers 1 through 12. The code is an HTML template using EJS syntax.

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4     <meta charset="utf-8">
5     <title>Portal de notícias</title>
6 </head>
7 <body>
8
9     <h1> IFMS - Notícias de Pesquisa </h1>
10
11 </body>
12 </html>

```

Agora vamos reescrever o código do app para utilizarmos o EJS.

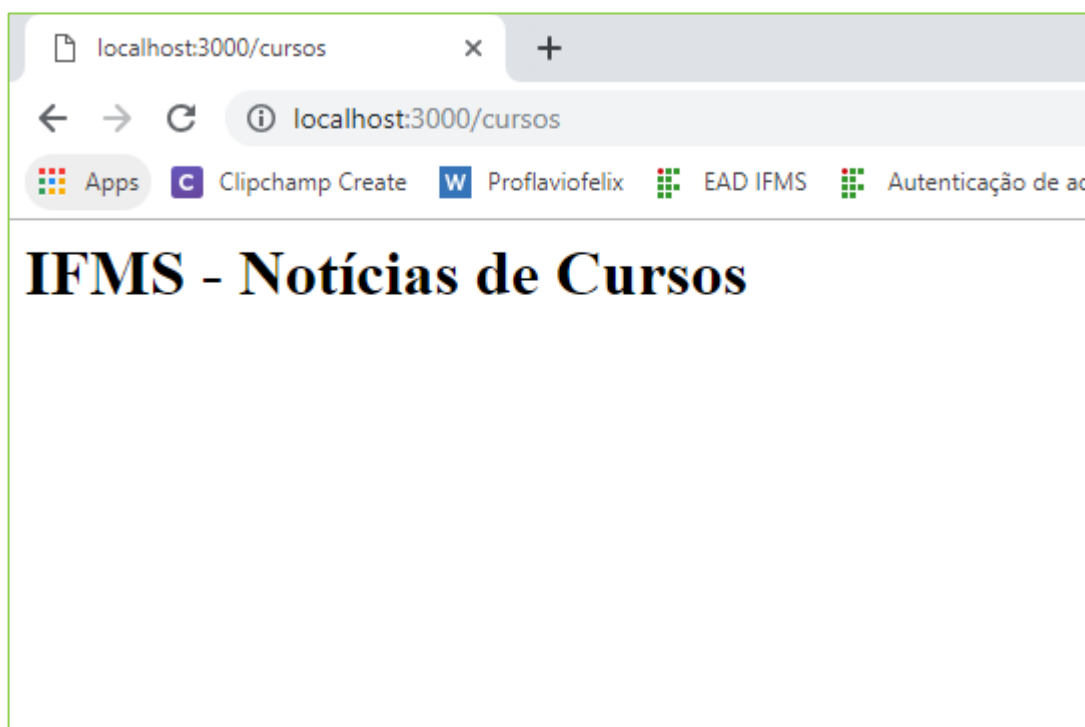
The screenshot shows the Sublime Text editor with the file explorer on the left. The file explorer shows the same project structure as the previous screenshot. The 'app.js' file is selected. The main editor area shows the content of 'app.js' with line numbers 1 through 25. The code is a JavaScript file that sets up an Express application with EJS as the view engine.

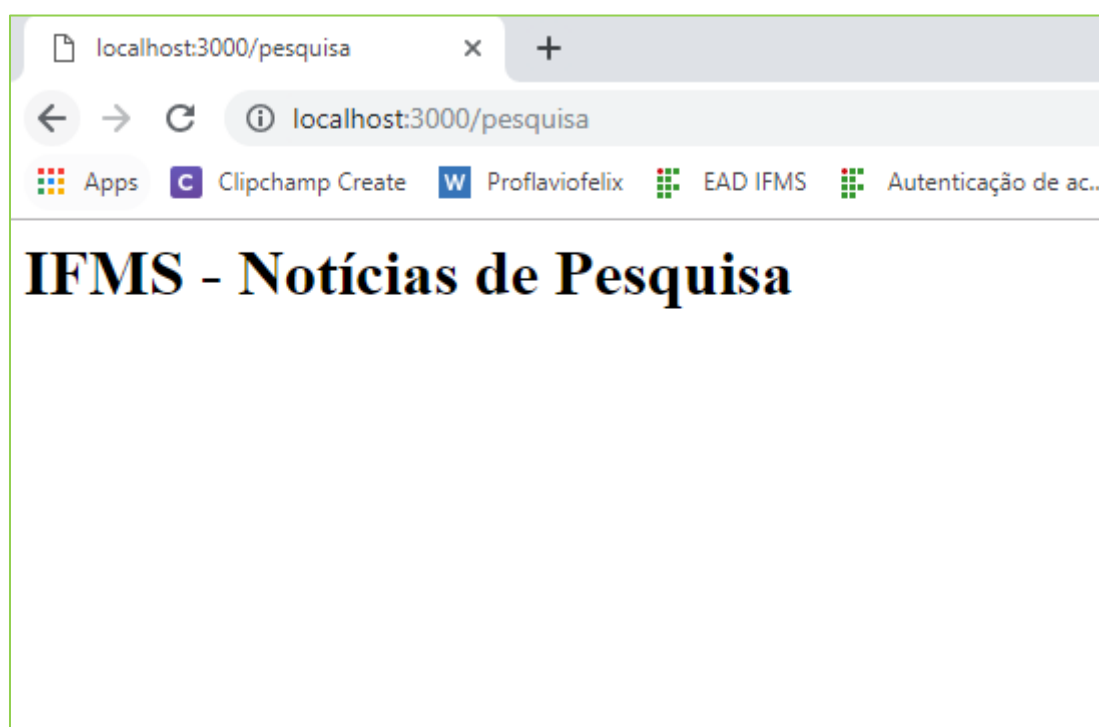
```

1 var express = require('express');
2
3 var app = express();
4
5 app.set('view engine', 'ejs'); //informamos ao Express que o EJS é o engine de views
6
7 app.get('/', function(req,res){
8     res.render("home/index");
9 });
10
11 app.get('/cursos', function(req,res){
12     res.render("secao/cursos");
13 });
14
15 app.get('/esportes', function(req,res){
16     res.render("secao/esportes");
17 });
18
19 app.get('/pesquisa', function(req,res){
20     res.render("secao/pesquisa");
21 });
22
23 app.listen(3000,function(){
24     console.log("Servidor rodando com Express");
25 });

```

Testando no browser:





NODEMON

Para resolver a questão de reiniciar o servidor toda hora, vamos utilizar o nodemon

```
npm install -g nodemon
```

o nodemon não é um modulo do projeto, é um utilitario

```
C:\code>node app3
Servidor rodando com Express
^C
C:\code>npm install -g nodemon
```

agora, para executar projetos vamos utilizar o nodemon da seguinte maneira:

nodemon app

```
C:\code>nodemon app3
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app3.js`
Servidor rodando com Express
```

Trabalhando com módulos

Permitem organizar melhor o código de modo a isolar um conjunto de códigos. Com isso podemos desenvolver um modulo e disponibilizar para outros desenvolvedores.

o Express e o EJS são exemplos de módulos

para incluir um modulo no projeto utilizamos o require (já estamos fazendo isso).

o CommonJS é um formato que define como devem ser construídos os módulos.

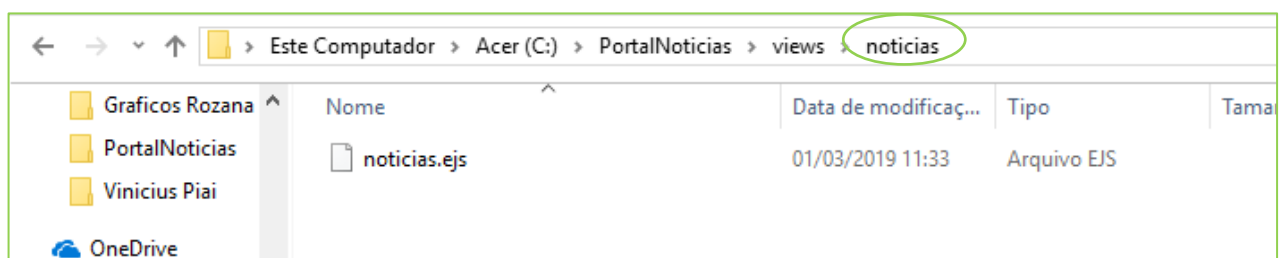
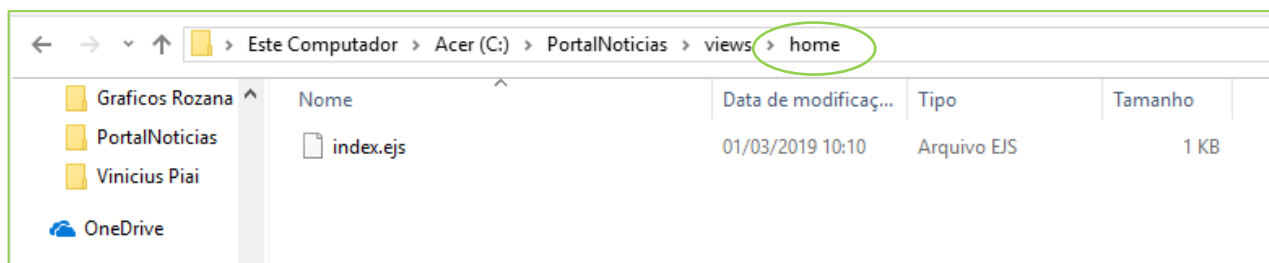
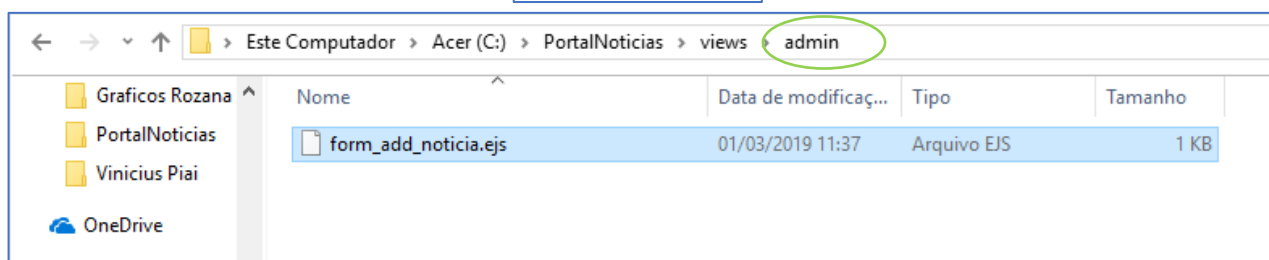
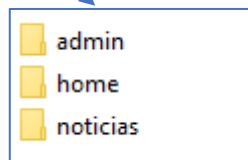
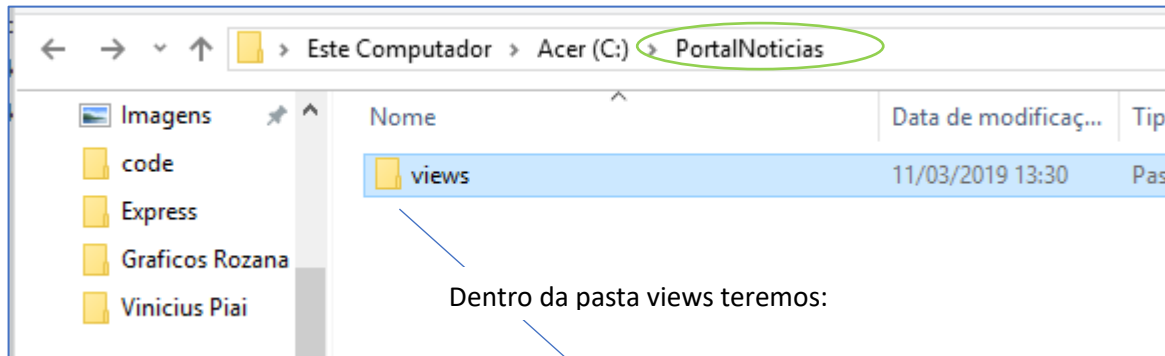
Vamos criar um modulo para entender melhor isso:

Um modulo pode retornar um **objeto**, **uma função**, **uma string**, **um boolean**, depende da necessidade e do objetivo para qual ele foi criado.

Vamos começar a criar nosso portal de notícias

Novo projeto

Criaremos a seguinte estrutura de pastas:



A seguir os códigos HTML de **form_add_noticia.ejs**, **index.ejs** e **noticias.ejs**

```
form_add_noticia.ejs x
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Form_add_noticia</title>
5 </head>
6 <body>
7   <h1> Formulário de adição de notícias </h1>
8
9 </body>
10 </html>
```

```
index.ejs x
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Index</title>
5 </head>
6 <body>
7   <h1>Index</h1>
8
9 </body>
10 </html>
```

```
noticias.ejs x
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Notícias</title>
5 </head>
6 <body>
7   <h1> Notícias </h1>
8
9 </body>
10 </html>
```

Criaremos o nosso arquivo app (dentro do diretório raiz: Portal_Noticias):

```

1 var express = require ('express');
2
3 var app = express();
4
5 app.set('view engine' , 'ejs'); //informamos ao Express que o EJS é
6
7 app.get('/', function(req,res){
8     res.render("home/index");
9 });
10
11 app.get('/formulario_inclusao_noticia', function(req,res){
12     res.render("admin/form_add-noticia");
13 });
14
15 app.get('/noticias', function(req,res){
16     res.render("noticias/noticias");
17 });
18

```

Vamos criar agora um módulo que retorne uma string:

Criaremos um arquivo chamado **mod_teste.js** e salvaremos no diretório raiz (junto com o arquivo app.js).

Esse arquivo retornará uma função com um mensagem.

```

C:\PortalNoticias\mod_teste.js (PortalNoticias) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ PortalNoticias
  └─ node_modules
  └─ views
  /* app.js
  /* mod_teste.js
  /* package-lock.js
  /* package.json

1 module.exports = function(){
2     var msg = "Este módulo contém apenas uma string";
3     return msg;
4 }
5

```

Agora, vamos voltar o arquivo **app.js** e chamaremos o método **listen** para sempre que uma requisição chegar em nossa aplicação, esta retorne o valor da variável **msg**.

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

app.get('/formulario_inclusao_noticia', function(req,res){
  res.render("admin/form_add-noticia");
});

app.get('/noticias', function(req,res){
  res.render("noticias/noticias");
});

app.listen(3000,function(){
  console.log(msg);
});
```

Agora vamos importar o módulo `mod_teste.js` para a nossa aplicação:



```
1 var express = require('express');
2
3 var app = express();
4
5 var msg = require('./mod_teste');
6
7 app.set('view engine', 'ejs'); //informamos ao Express que o EJS é o engine de views
8
9 app.get('/', function(req,res){
10   res.render("home/index");
11 });
12
13 app.get('/formulario_inclusao_noticia', function(req,res){
14   res.render("admin/form_add-noticia");
15 });
16
17 app.get('/noticias', function(req,res){
18   res.render("noticias/noticias");
19 });
20
21
22 app.listen(3000,function(){
23   console.log(msg);
24 });
```


O resultado será:

```
C:\PortalNoticias>nodemon app
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
[Function]
```

O console do Nodemon não vai mostrar o valor, somente identificar que tem uma function.

Quando o modulo retorna uma função é necessário que façamos a execução da função,

A seguir o projeto alterado executando a função:



```
4
5 var msg = require('./mod_teste');
6
7 app.set('view engine', 'ejs'); //informamos ao Express que o EJS é o engine de views
8
9 app.get('/', function(req,res){
10   res.render("home/index");
11 });
12
13 app.get('/formulario_inclusao_noticia', function(req,res){
14   res.render("admin/form_add-noticia");
15 });
16
17 app.get('/noticias', function(req,res){
18   res.render("noticias/noticias");
19 });
20
21
22 app.listen(3000, function(){
23   console.log(msg());
24 });
```

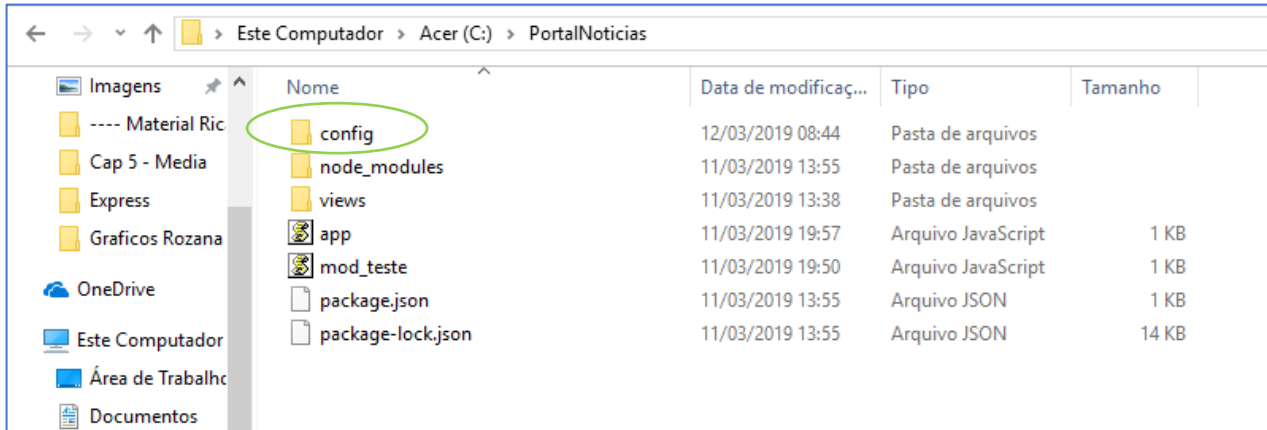
Olhando o console, vemos agora o resultado:

```
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Este módulo contém apenas uma string
```

A partir de agora a configuração do servidor será modularizada

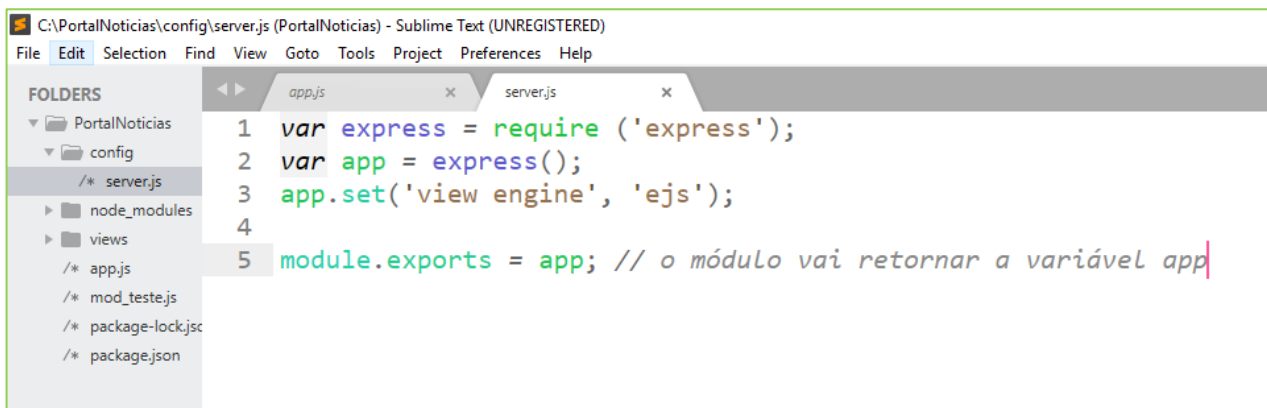
Vamos começar a modularização pela configuração do Server.

1 – criar um diretório **config** e dentro desse diretório criar o arquivo **server.js**



Copiar as primeiras linhas onde tem a configuração do servidor para o novo arquivo

O arquivo server.js ficará assim:



Obs.: Agora já podemos deletar o arquivo **mod_teste.js** (já que o utilizamos apenas para fazer um teste para criação de módulos).

No app.js, iremos importar o módulo server. Ficará assim:

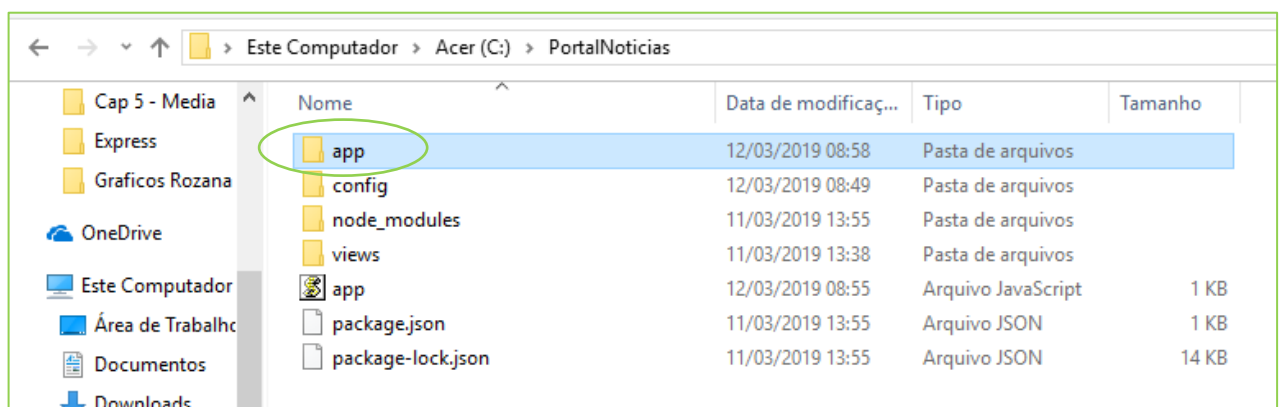
```

1 var app = require('./config/server');
2
3
4
5 app.get('/', function(req,res){
6     res.render("home/index");
7 });
8
9 app.get('/formulario_inclusao_noticia', function(req,res){
10     res.render("admin/form_add-noticia");
11 });
12
13 app.get('/noticias', function(req,res){
14     res.render("noticias/noticias");
15 });
16
17
18 app.listen(3000,function(){
19     console.log(msg());
20 });

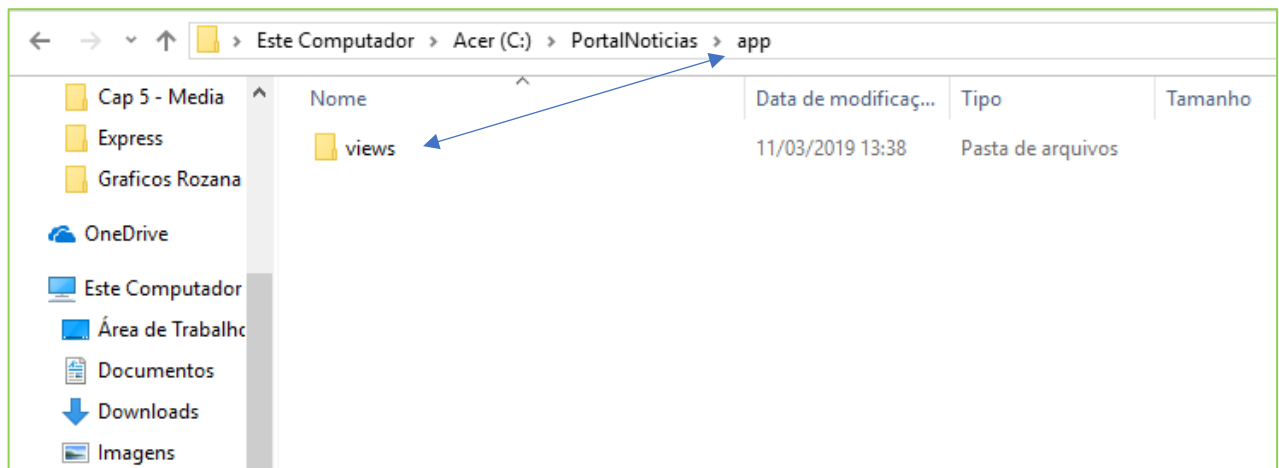
```

Continuando a modularização do projeto, vamos organizar melhor os diretórios e arquivos:

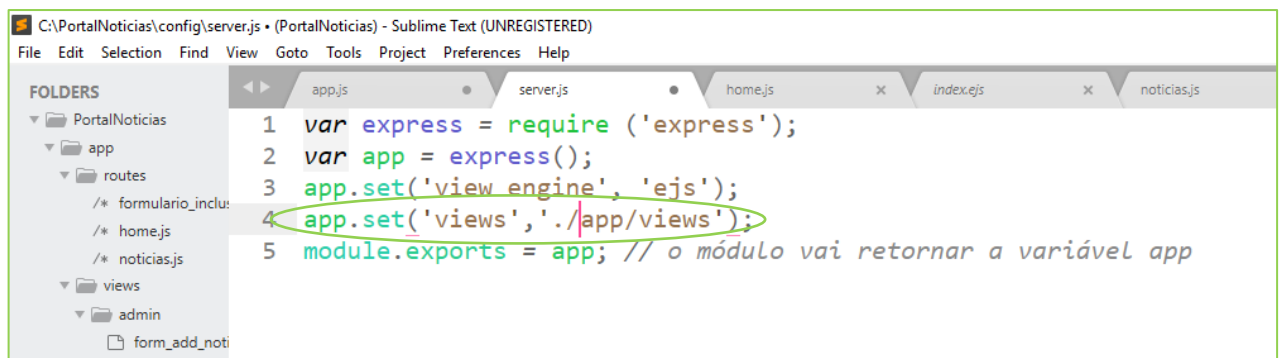
1. Vamos criar uma pasta agora chamada app (no diretório raiz do projeto).



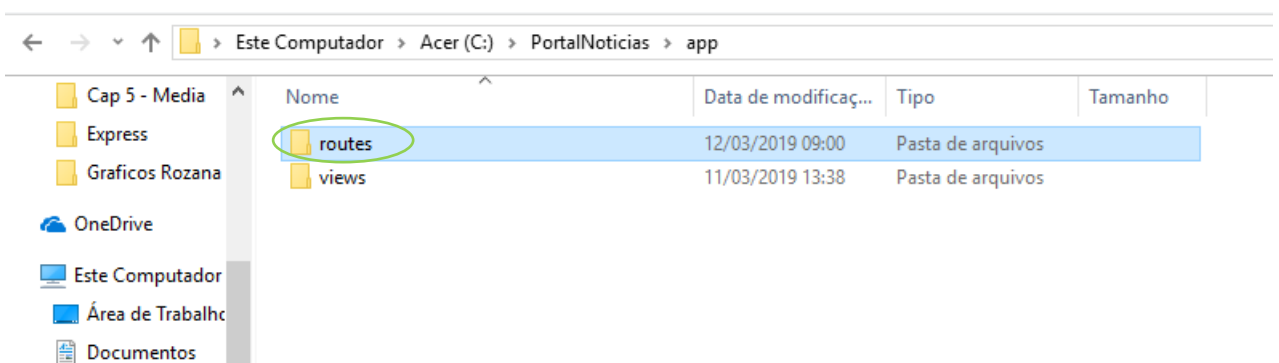
2. Mover a pasta views para o diretório app.



Como o ejs utiliza o diretório views como padrão, devemos configurar o novo caminho pois a pasta views foi movida para a pasta app.



3. Criar o diretório routes (dentro do diretório app).

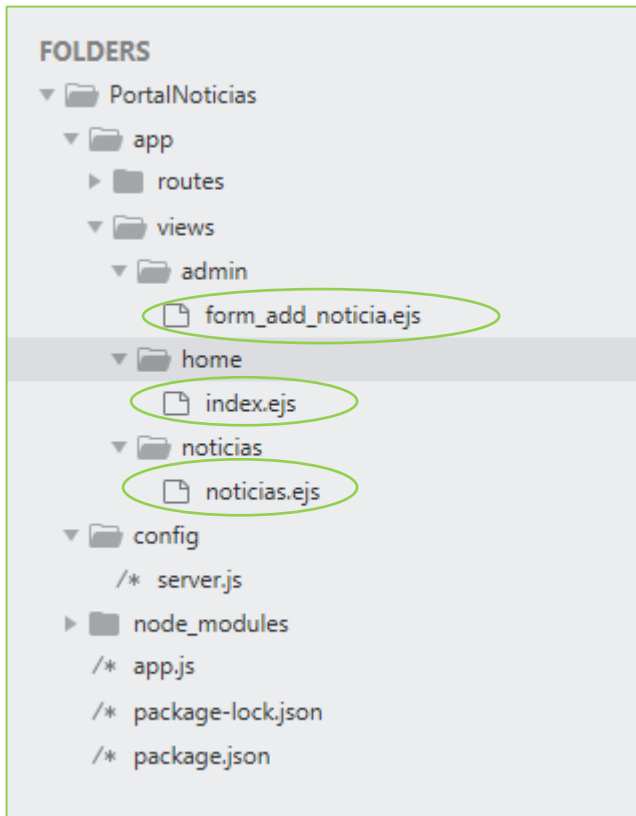


4. Criar um arquivo de rota para cada rota do projeto, com a extensão js (dentro do diretório de routes).

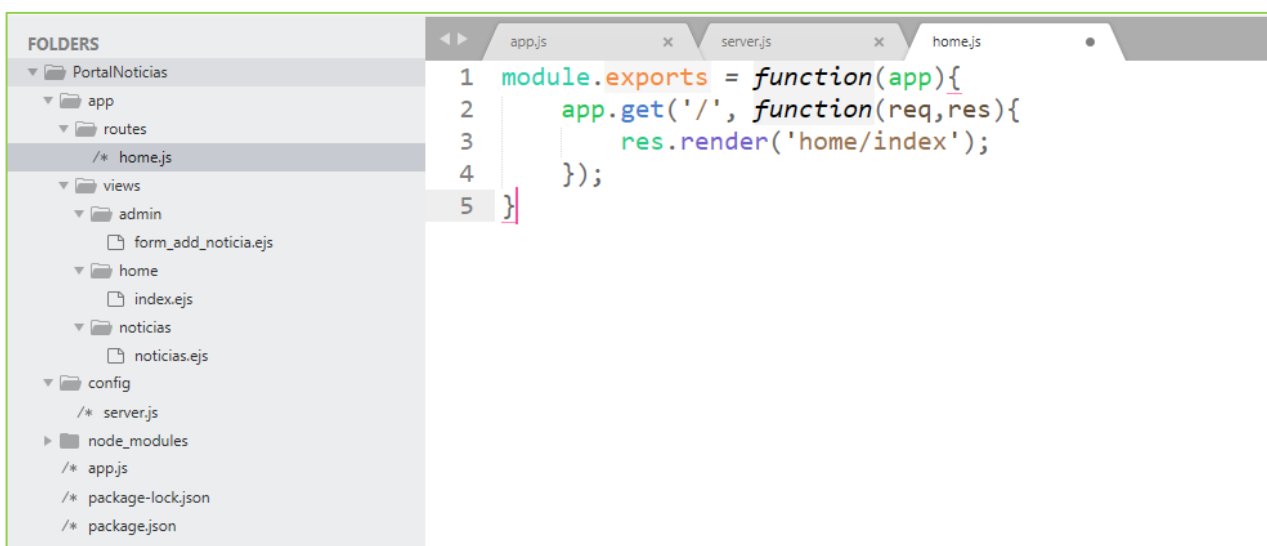
Lembrando:

Uma rota para cada página (view).

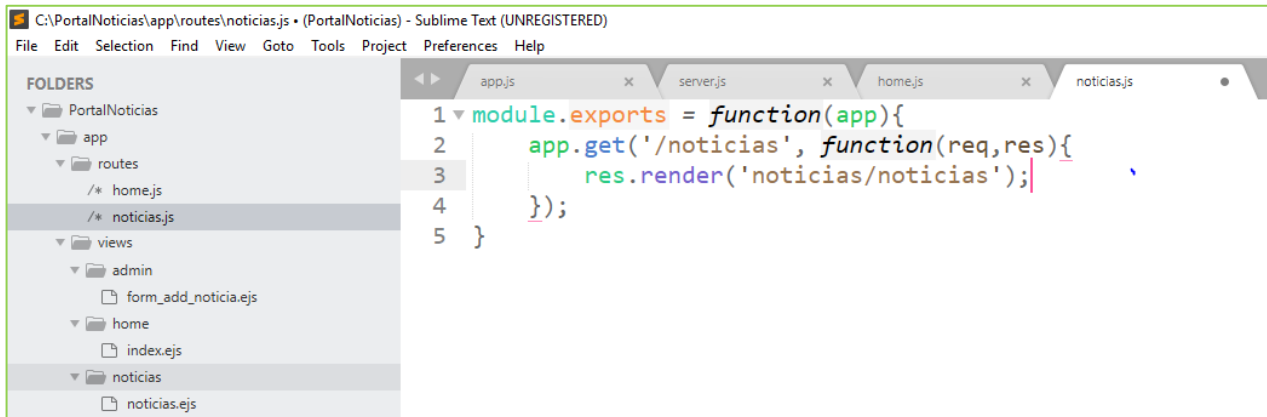
Temos 3 páginas (views):



Dentro do diretório **routes** vamos criar uma rota para a página **index.ejs**. Essa rota se chamará **home.js**



Dentro do diretório **routes** vamos criar uma rota para a página **noticias.ejs**. Essa rota se chamará **noticias.js**



The screenshot shows the Sublime Text editor with the file `C:\PortalNoticias\app\routes\noticias.js` open. The left sidebar displays the project structure, with the `routes` folder expanded, showing `home.js` and `noticias.js`. The main editor window shows the following code:

```
1 module.exports = function(app){
2   app.get('/noticias', function(req,res){
3     res.render('noticias/noticias');
4   });
5 }
```

Dentro do diretório **routes** vamos criar uma rota para a página **form_add_noticia.ejs**. Essa rota se chamará **formulario_inclusao_noticia.js**



The screenshot shows the Sublime Text editor with the file `C:\PortalNoticias\app\routes\formulario_inclusao_noticia.js` open. The left sidebar displays the project structure, with the `routes` folder expanded, showing `formulario_inclusao_noticia.js`, `home.js`, and `noticias.js`. The main editor window shows the following code:

```
1 module.exports=function(app){
2   app.get('/formulario_inclusao_noticia', function(req,res){
3     res.render('admin/form_add_noticia');
4   });
5 }
```

O arquivo `app` precisa ser alterado para importar os módulos, passando a variável `app` como parâmetro

```

1 var app = require('./config/server');
2
3 var rotaNoticias = require('./app/routes/noticias');
4 rotaNoticias(app);
5
6 var rotaHome = require('./app/routes/home');
7 rotaHome(app);
8
9 var rotaFormInclusaoNoticia = require('./app/routes/formulario_inclusao_noticia');
10 rotaFormInclusaoNoticia(app);
11
12
13
14
15 app.listen(3000, function(){
16     console.log('Servidor ON');
17 });

```

Para otimizar o código, a chamada da função pode ser feita na mesma linha do require:

```

1 var app = require('./config/server');
2
3 var rotaNoticias = require('./app/routes/noticias')(app);
4
5 var rotaHome = require('./app/routes/home')(app);
6
7 var rotaFormInclusaoNoticia = require('./app/routes/formulario_inclusao_noticia')(app);
8
9
10
11
12 app.listen(3000, function(){
13     console.log('Servidor ON');
14 });

```

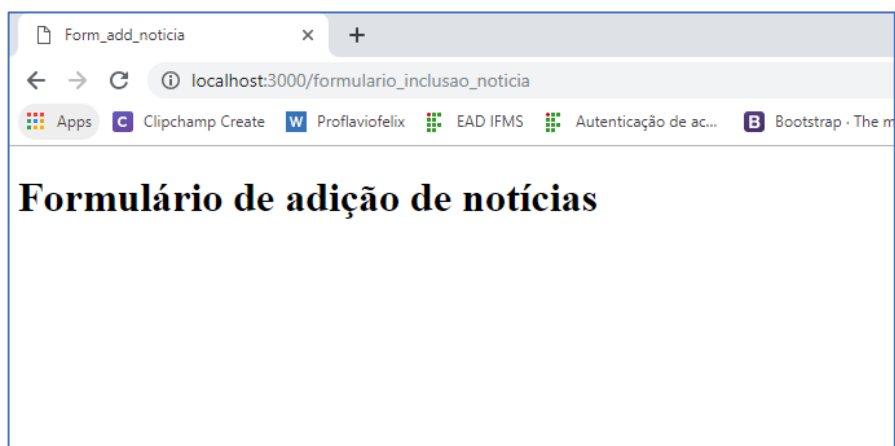
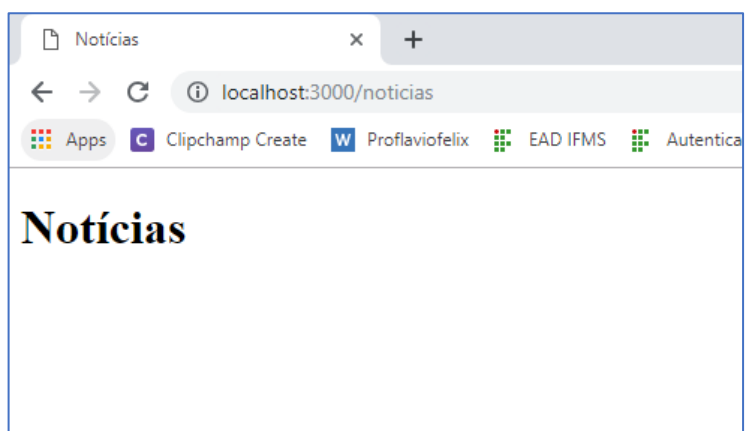
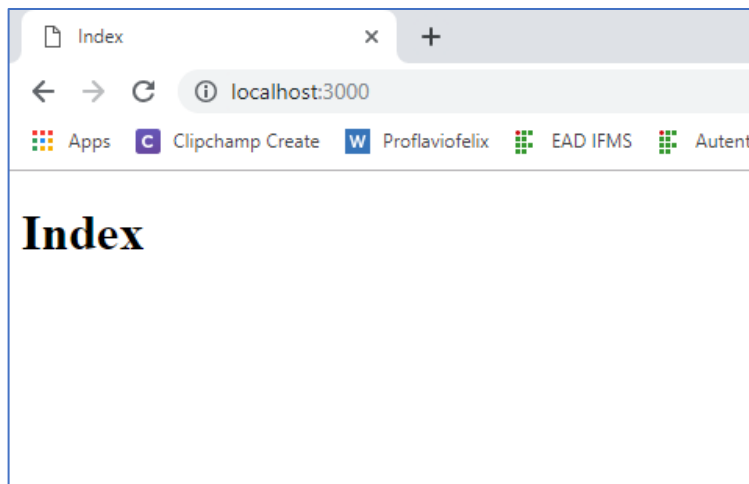
O arquivo app.js está guardando somente informações estruturais. As regras de negócio estão sendo isoladas nos módulos.

Testando:

```

[nodemon] restarting due to changes...
[nodemon] starting `node app`
Servidor ON

```



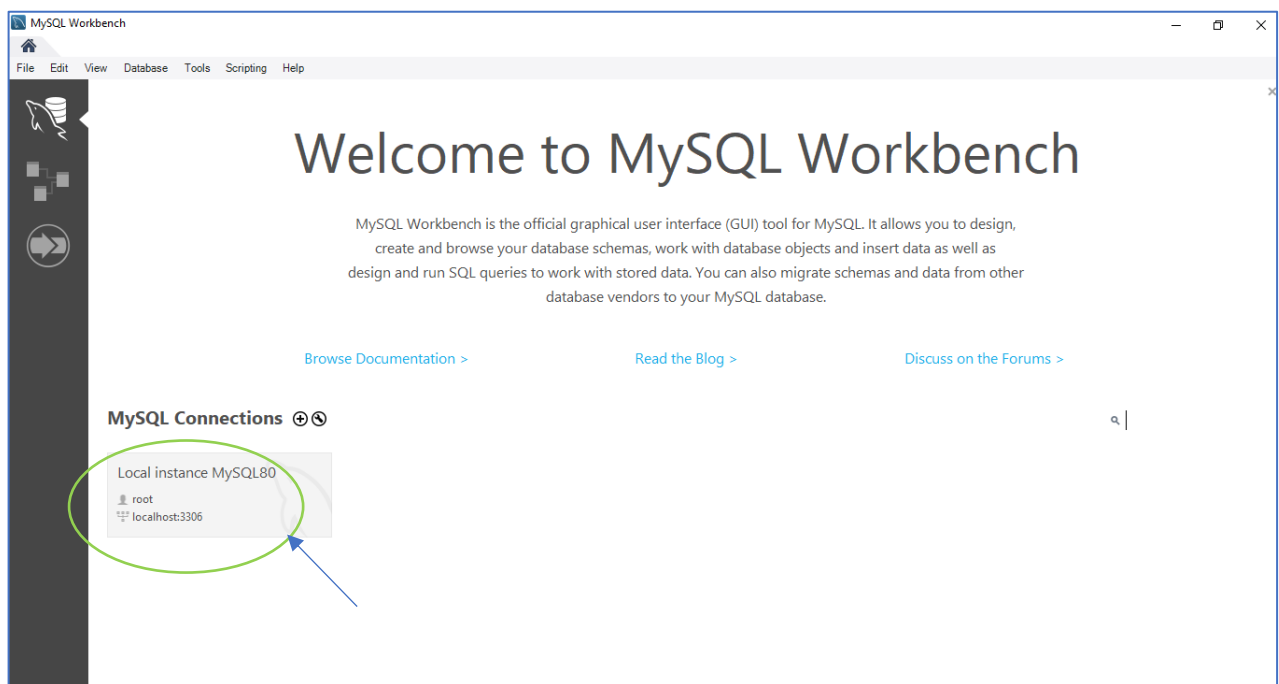
TRABALHANDO COM BANCO DE DADOS – MYSQL

Instalar o módulo mysql no projeto utilizando o NPM

```
npm install mysql --save
```

```
C:\Users\profe>cd C:\PortalNoticias  
  
C:\PortalNoticias>npm install mysql -save  
npm WARN portalnoticias@1.0.0 No description  
npm WARN portalnoticias@1.0.0 No repository field.  
  
+ mysql@2.16.0  
added 9 packages from 14 contributors and audited 135 packages in 80.229s  
found 0 vulnerabilities  
  
C:\PortalNoticias>
```

Abrir o MySql Workbench para criar o banco de dados



A sequencia de instruções sql que usaremos será:

Para criar o banco de dados:

```
create database portal_noticias;
```

Para abrir o banco de dados e poder dentro dele criar tabelas:

`use portal_noticias;`

Para criar a tabela:

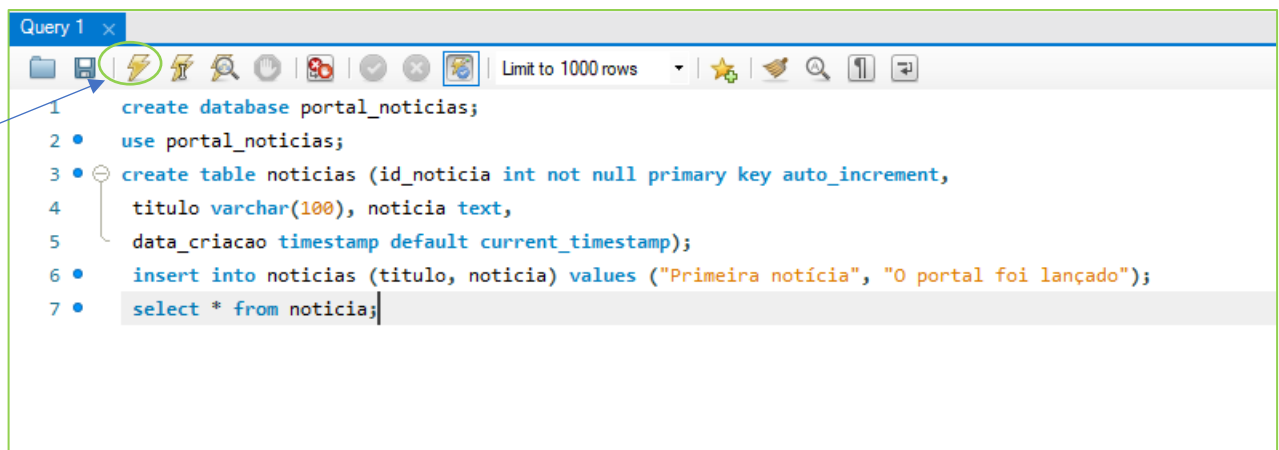
`create table noticias (id_noticia int not null primary key auto_increment,
titulo varchar(100), noticia text, data_criacao timestamp default
current_timestamp);`

Para inserir uma notícia na tabela:

`insert into noticias (titulo, noticia) values ("Primeira notícia", "O portal foi lançado");`

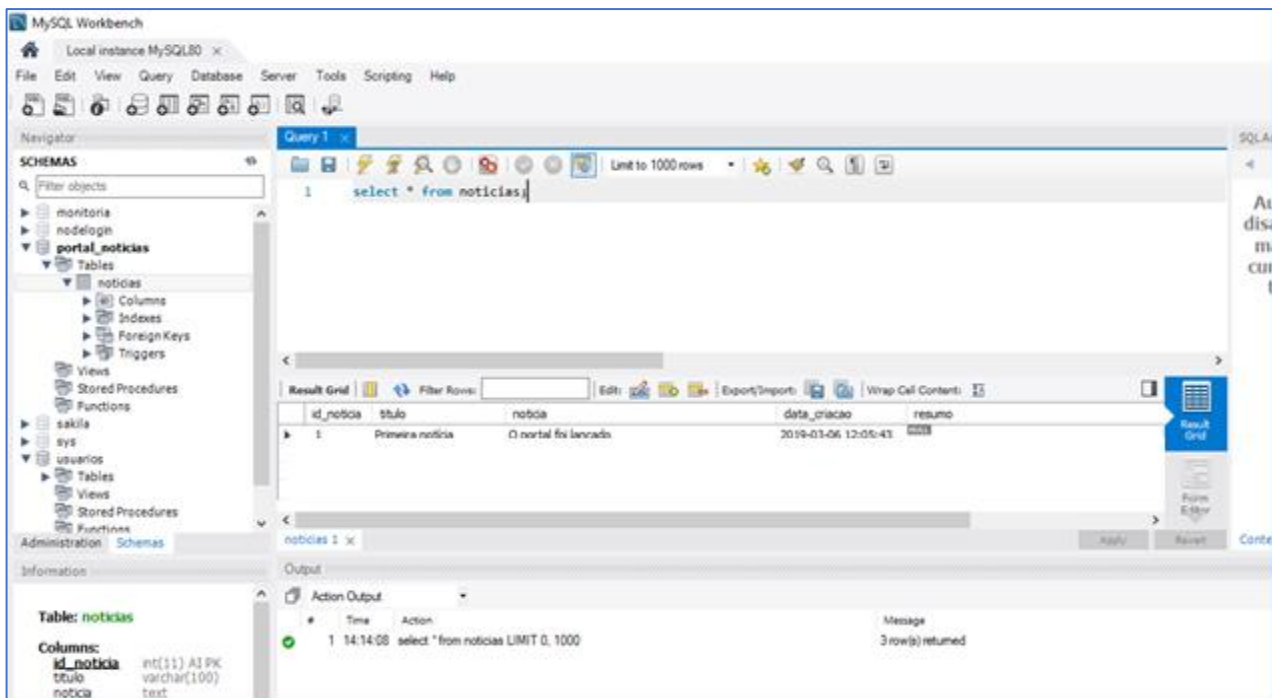
Para ver os registros da tabela:

`select * from noticia`



```
Query 1 x
1 create database portal_noticias;
2 use portal_noticias;
3 create table noticias (id_noticia int not null primary key auto_increment,
4 titulo varchar(100), noticia text,
5 data_criacao timestamp default current_timestamp);
6 insert into noticias (titulo, noticia) values ("Primeira notícia", "O portal foi lançado");
7 select * from noticia;
```

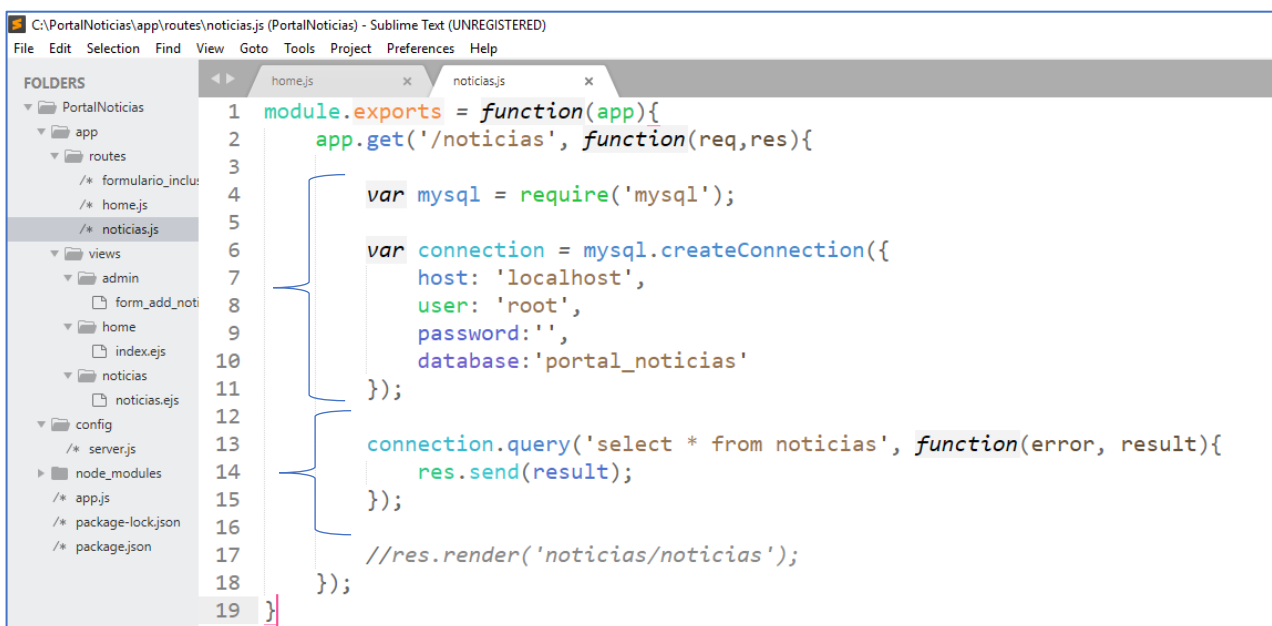
Ao clicar na execução do código, o resultado será:



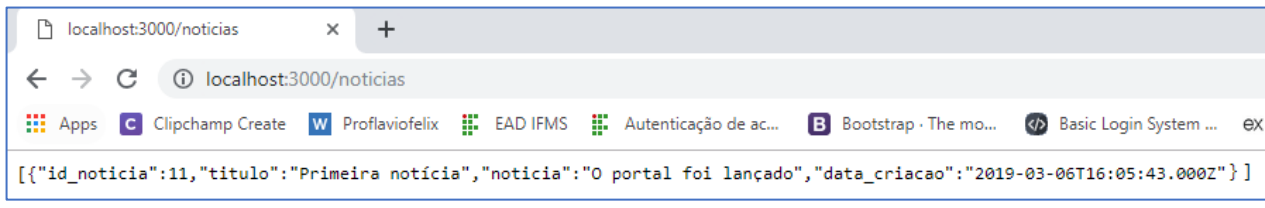
Voltando ao projeto, alterando o arquivo de rotas de noticias:

1 - Vamos fazer a conexão com o banco de dados

2 - Selecionar a tabela noticias e mostrar o resultado, que são a lista de noticias, que será mostrada no formato Json.



Podemos ver o resultado abrindo o navegador, as notícias serão exibidas.



Até aqui somente recuperamos os resultados do banco e mostramos para quem fez a requisição. O interessante agora é passar os resultados para a view e a view mostrar as informações.

O que será alterado:

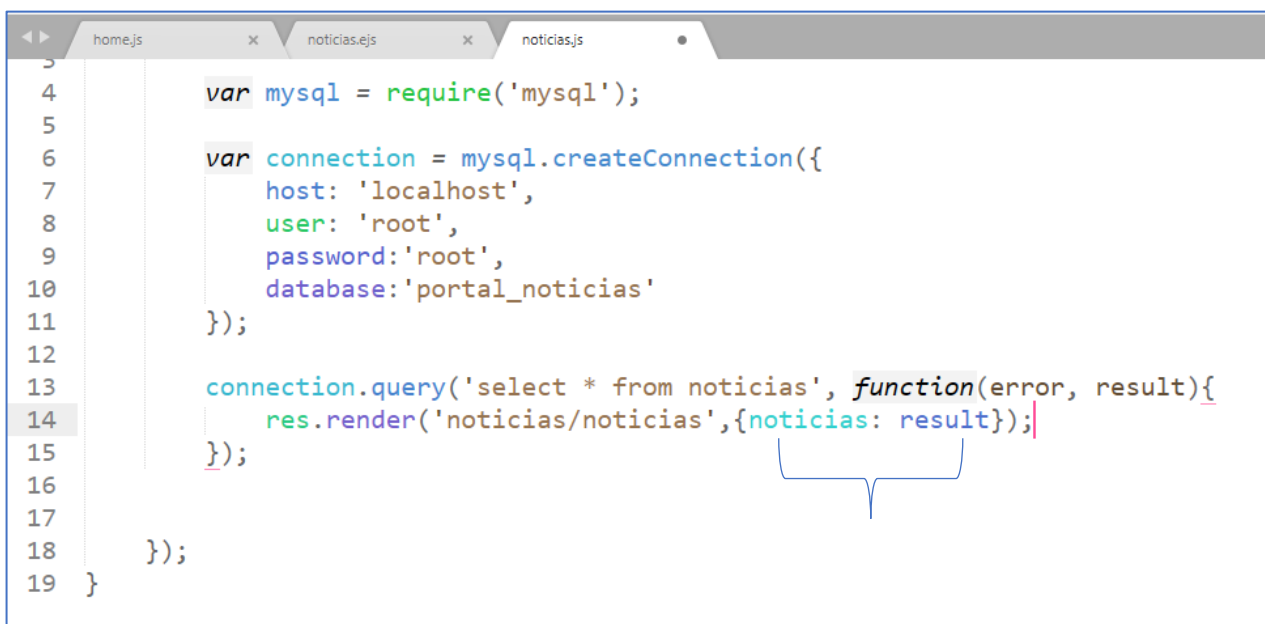
O EJS já suporta o seguinte comportamento: Ao invés de dar um `res.send`, iremos dar um `res.render` (reindenzar).

O EJS vai recuperar o resultado na view e na view vamos escrever códigos JavaScript.

Cada view vai ser processada, cada código Javascript vai ser interpretado e será formado um HTML completo

Resultado:

Noticias.js (arquivo de rotas)



noticias.ejs (Arquivo de view)

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4     <meta charset="utf-8">
5     <title>Notícias</title>
6 </head>
7 <body>
8     <h1> Notícias </h1>
9     <table>
10         <tr>
11             <td>ID</td>
12             <td>Título</td>
13             <td>Notícia</td>
14         </tr>
15         <% for (var i=0; i<noticias.length; i++) { %>
16         <tr>
17             <td><%=noticias[i].id_noticia %></td>
18             <td><%=noticias[i].titulo %></td>
19             <td><%=noticias[i].noticia %></td>
20         </tr>
21         <% } %>
22     </table>
23
24 </body>
25 </html>

```

O resultado no navegador é:

Notícias

ID	Título	Notícia
1	Primeira notícia	O portal foi lançado