

Apostila de Introdução ao Node.JS

Versão 4.0

Profª Mª Denilce Veloso

Sorocaba/SP

Maio/2022

ÍNDICE

11.	O que é NODE.JS.....	6
11.3	Elementos que compõem o Node.js	8
12.	Modelo Tradicional x Modelo Node.js.....	10
13.	Instalação Node.js	11
4.	Instalação do editor Sublime Text.....	13
5.	Exercícios.....	14
5.1	Primeiro exercício.....	14
5.2	Exercício 2	15
5.3	Exercício 3	16
5.4	Exercício 4	17
5.5	Exercício 5	22
6.	Respondendo requisições HTTP	23
7.	Respondendo requisições usando a URL	25
8.	Ferramentas para auxiliar no Desenvolvimento	26
8.1	NPM.....	26
8.2	Instalação do Express	29
9.	Utilizando as novas ferramentas.....	32
9.1	Guardando os exercícios anteriores.....	32
9.2	Utilizando o Express	32
9.3	Utilizando o EJS	35
9.4	Testando o Nodemon.....	39
9.5	Organizando melhor a aplicação.....	41
9.5.1	Módulos e CommonJS.....	45
9.5.2	Modularizando a aplicação	48
10.	Banco de Dados.....	53
10.1	Instalação do SQL Server	54
10.2	Instalação do driver mssql.....	54
10.3	Usando o SQL Server no Prompt.....	55
10.4	Listando dados de uma tabela na página.....	56
10.5	Listando dados de uma tabela na página (através da view)	58
10.6	Alterando a forma de conexão com o banco SQL Server.....	60
11.	Separando a aplicação em Camadas.....	63
11.1	MVC.....	63

11.1.1 Model ou Modelo.....	63
11.1.2 Controller ou Controlador.....	63
11.1.3 View ou Visão	64
11.2 Melhorando a organização das rotas.....	64
11.2.1 Consign	64
11.2.2 Inclusão do Consign no server.js	65
11.2 Restruturação da parte do banco de dados.....	68
12. Criando uma página para recuperar professor pelo ID.....	73
12.1 Criação da rota	73
12.2 Criação da view detalhaprofessor.ejs	74
13. Implementando os Models	77
14. Criação do formulário Inclusão do Professor	81
14.3 Body-Parser	84
14.1.1 Instalação do Body-Parser.....	85
14.1.2 Alteração do server e model para inserir no banco de dados	85
Referências.....	91

Índice de Figuras

Figura 1: Arquitetura do Node.Js	7
Figura 2: Modelo Tradicional x Modelo Node.Js.....	10
Figura 3: Site nodejs.org.....	11
Figura 4: Instalação do arquivo do Node.Js.....	12
Figura 5: Versão do Node.Js	13
Figura 6: Site sublimetext.com.....	13
Figura 7: Instalação do Sublime text	14
Figura 8: Primeiro.js	15
Figura 9: Executando Primeiro.Js	15
Figura 10: Diferença entre código síncrono e assíncrono	16
Figura 11: Execução do Exercício 3	17
Figura 12: Arquivo file.txt.....	18
Figura 13: Execução do QuartoS	18
Figura 14: Execução do QuartoA.....	21
Figura 15: Execução do Exercício quinto.js	23
Figura 16: Execução do exercicio6.js.....	24
Figura 17: Execução do exercicio6.js (chamando o browser)	24
Figura 18: Execução do exercicio7.js.....	25
Figura 19: Execução do exercicio7.js (chamando no browser a opção historia)	26
Figura 20: Versão do NPM.....	27
Figura 21: Execução npm init	27
Figura 22: Arquivo package-json	28
Figura 23: Pasta node_modules	29
Figura 24: Pasta node_modules\ejs	30
Figura 25: Teste nodemon.....	31
Figura 25: Pasta ExerciciosAnteriores	32
Figura 26: exercicio7.js e app.js.....	33
Figura 27: Teste do servidor com express.....	34
Figura 28: Teste local do servidor	34
Figura 29: Teste local.....	35
Figura 30: Teste da opção Cursos.....	35
Figura 31: Pasta views e seção	36
Figura 32: Secao historia	39
Figura 33: Teste do Nodemon.....	39
Figura 34: Teste do site principal	40
Figura 35: Novo Teste do site principal.....	40
Figura 37: Salvando o arquivo no formato UTF-8	43
Figura 36: Nova Pasta views.....	43
Figura 38: Estrutura Módulo	45
Figura 39: Teste chamando o módulo.....	47
Figura 40: Teste chamando a página.....	47
Figura 41: Teste chamando o módulo usando nodemon	48
Figura 42: Pasta config	48

Figura 43: Cópia da pasta views	50
Figura 44: Tela console iniciando servidor	50
Figura 45: Tela console iniciando servidor	50
Figura 46: Pasta routes.....	51
Figura 47: Tela console iniciando servidor	53
Figura 48: Tela página professores	53
Figura 49: Site download do SQL Server	54
Figura 50: Módulos do SQL Server	55
Figura 51: Testando o MySQL no console	55
Figura 52: Testando a página dos professores (listar registros)	58
Figura 53: Testando a página dos professores (usando view dinâmica).....	60
Figura 54: Novo arquivo dbConnection para acesso ao banco.....	61
Figura 55: Estrutura MVC.....	63
Figura 56: node_modules com o Consign	65
Figura 57: Consign incluindo a pasta routes	66
Figura 58: Consign incluindo a pasta routes mesmo sem estarem no app.js.....	67
Figura 59: Página principal depois do Consign.....	67
Figura 60: Página história depois do Consign.....	68
Figura 61: Página professores depois do Consign (sem startar banco de dados).....	68
Figura 61: Recarregando servidor com autoload carregando routes	72
Figura 62: Página principal com autoload carregando routes	72
Figura 63: Página história com autoload carregando routes	73
Figura 64: Rota detalhaprofessor.js	73
Figura 65: View detalhaprofessor.ejs.....	74
Figura 66: Servidor carregando rota detalhaprofessor.js	76
Figura 67: Teste página professores.....	76
Figura 68: Teste página detalhaprofessor.....	77
Figura 69: Criação pasta models	77
Figura 70: Servidor carregando a pasta models.....	80
Figura 71: Página dos professores com servidor carregando a pasta models	81
Figura 72: Página detalhaprofessor com servidor carregando a pasta models	81
Figura 73: Rota adicionar_professor.js.....	81
Figura 74: Arquivo (página) adicionar_professor.ejs	82
Figura 75: Página adicionar_professor.....	84
Figura 76: Retorno do Enviar (Salvar) professor	84
Figura 77: Body-Parser no node_modules	85
Figura 78: Carregando página adicionar_professor	86
Figura 79: Retorno do Salvar de adicionar_professor.....	86
Figura 80: Servidor recarregado	89
Figura 81: Página adicionar_professor	89

Introdução ao Node.Js

11. O que é NODE.JS

Em 2009, o Node.js foi desenvolvido por Ryan Dahl essencialmente como uma forma de rodar (interpretar) programas JavaScript fora do contexto de um browser, foi escrito em C++.

Essa nova abordagem fez nascer uma nova técnica para criação de aplicações web onde com apenas uma linguagem de programação, é possível criar tantos scripts para o front end quanto para o back end.

O Node.js uma plataforma construída sobre o motor JavaScript do Google Chrome (V8) para facilmente construir aplicações de rede rápidas e escaláveis. Usa um modelo de I/O direcionada a evento não bloqueante (figura 1) que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos.

O JavaScript é uma linguagem **interpretada**, o que o coloca em desvantagem quando comparado com linguagens compiladas, pois cada linha de código precisa ser interpretada enquanto o código é executado. O **V8 compila o código para linguagem de máquina**, além de **otimizar drasticamente a execução** usando heurísticas (procedimentos, estratégias), permitindo que a **execução seja feita em cima do código compilado** e não interpretado.

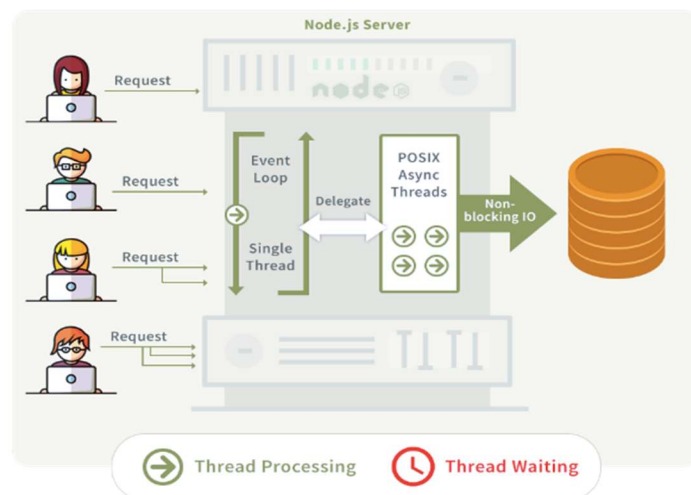
O Node.js é construído com as novas versões do **V8**. Mantendo-se em dia com as últimas atualizações desta **engine**, as novas funcionalidades da **especificação JavaScript ECMA-262** são trazidas para os desenvolvedores Node.js em tempo **hábil**, assim como as melhorias contínuas de performance e estabilidade.¹Todas as funcionalidades em lançamento (shipping), que o V8 considera estáveis, **são ativadas por padrão no Node.js e NÃO** necessitam de nenhum tipo de flag de tempo de execução.

O Node.js é open-source e multiplataforma e permite aos desenvolvedores criarem todo tipo de aplicativos e ferramentas do lado servidor (backend) em JavaScript. Node é usado fora do contexto de um navegador (ou seja, executado diretamente no computador ou no servidor). Como tal, o ambiente omite APIs

¹ <https://nodejs.org/pt-br/docs/es6/>

JavaScript específicas do navegador e adiciona suporte para APIs de sistema operacional mais tradicionais, incluindo bibliotecas de sistemas HTTP e arquivos.²

Figura 1: Arquitetura do Node.js³



Thread → é um pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se autodividir em duas ou mais tarefas (ordem de execução).

Single Thread -> Qual seria a vantagem de limitar a execução da aplicação em somente um *thread*? Linguagens como Java, PHP e Ruby seguem um modelo onde cada nova requisição roda em um *thread* separada do sistema operacional. Esse modelo é eficiente, mas tem um custo de recursos muito alto e nem sempre é necessário todo o recurso computacional aplicado para executar um novo *thread*.

O Node.js foi criado para solucionar esse problema, usar programação assíncrona e recursos compartilhados para tirar maior proveito de um *thread*.

Event Loop → O Node.js é guiado (orientado) por eventos, termo também conhecido como *Event Driven*. Esse conceito já é bastante aplicado em

² <https://nodejs.org/pt-br/about/>

³ Fonte: <https://blog.schoolofnet.com/como-comecar-com-node-js/>

interações com interface de usuário. O JavaScript possui diversas APIs baseadas em eventos para interações com o *DOM* (*onClick*, *onHide*, *onShow*, etc) são muito comuns no mundo front-end com JavaScript. *Event driven* é um fluxo de controle determinado por eventos ou alterações de estado, a maioria das implementações possuem um *core* (central) que escuta todos os eventos e chama seus respectivos *call-backs* (função passada a outra função como argumento) quando eles são lançados (ou têm seu estado alterado). Esse basicamente é o resumo do *Event Loop* do *Node.js*.

Call Stack → A *stack* (pilha) é um conceito bem comum no mundo das linguagens de programação, já ouviu “estouro de pilha”? No *Node.js* e no JavaScript, em geral, esse conceito não se difere muito de outras linguagens. Sempre que uma função é executada, ela entra na *stack*, que executa somente uma coisa por vez, ou seja, o código posterior ao que está rodando precisa esperar a função atual terminar de executar para seguir adiante.

Multi Threading → Na verdade, quem é *single thread* é o V8, o motor do *google* utilizado para rodar o *Node.js*. Para que seja possível executar tarefas assíncronas, o *Node.js* conta com diversas outras APIs – algumas delas providas pelos próprios sistemas operacionais, como é o caso de eventos de disco, *sockets TCP* e *UDP*. Quem toma conta dessa parte de *I/O* assíncrono, de administrar múltiplas *threads* e enviar notificações é a *libuv*.

Posix - A interface do sistema operacional portátil para ambientes de computação é um conjunto de padrões e especificações que definem maneiras de os programas de computador interagirem com um sistema operacional, apesar da base dele ter sido o Unix.

11.3 Elementos que compõem o Node.js

V8: É a engine de código aberto de alto desempenho para JavaScript e WebAssembly do Google, escrito em C ++. É usado no Chrome e no Node.js, entre outros. Ele implementa ECMAScript e WebAssembly e é executado no Windows 7 ou posterior, macOS 10.12+ e sistemas Linux que usam

processadores x64, IA-32, ARM ou MIPS. O V8 pode ser executado de forma autônoma ou pode ser incorporado a qualquer aplicativo C ++.⁴

Libuv: É uma biblioteca de suporte multiplataforma com foco em entrada e saída assíncrona. Ele foi desenvolvido principalmente para uso por Node.js, mas também é usado por Luvit⁵, Julia, pyuv e outros.⁶

HTTP-parser: É um analisador para mensagens HTTP escritas em C. Ele analisa solicitações e respostas. O analisador é projetado para ser usado em aplicativos HTTP de desempenho. Ele não faz chamadas nem alocações, não armazena dados em buffer, pode ser interrompido a qualquer momento. Dependendo da sua arquitetura, ele requer apenas cerca de 40 bytes de dados por fluxo de mensagens (em um servidor web por conexão).⁷

C-ares: É uma biblioteca C para solicitações assíncronas de DNS (incluindo resolução de nomes). Compatibilidade com C89, licenciado pelo MIT, é construído e executado em POSIX, Windows, Netware, Android e muitos outros sistemas operacionais.⁸

OpenSSL: É um kit de ferramentas robusto, de nível comercial e completo para os protocolos Transport Layer Security (TLS) e Secure Sockets Layer (SSL). É também uma biblioteca de criptografia de propósito geral.⁹

zlib: O zlib foi projetado para ser uma biblioteca de compactação de dados sem perdas, gratuita e de uso geral, legalmente desimpedida (ou seja, não coberta por nenhuma patente) para uso em virtualmente qualquer hardware de computador e sistema operacional. O próprio formato de dados zlib é portátil entre plataformas. O método de compactação atualmente usado em zlib essencialmente nunca expande os dados. A área de cobertura da memória do

⁴ <https://v8.dev>

⁵ Plataformas de e-learning

⁶ <http://docs.libuv.org/en/v1.x/>

⁷ <https://github.com/nodejs/http-parser/blob/master/README.md>

⁸ <https://c-ares.haxx.se>

⁹ <https://www.openssl.org>

zlib também é independente dos dados de entrada e pode ser reduzida, se necessário, com algum custo na compactação.¹⁰

12. Modelo Tradicional x Modelo Node.Js

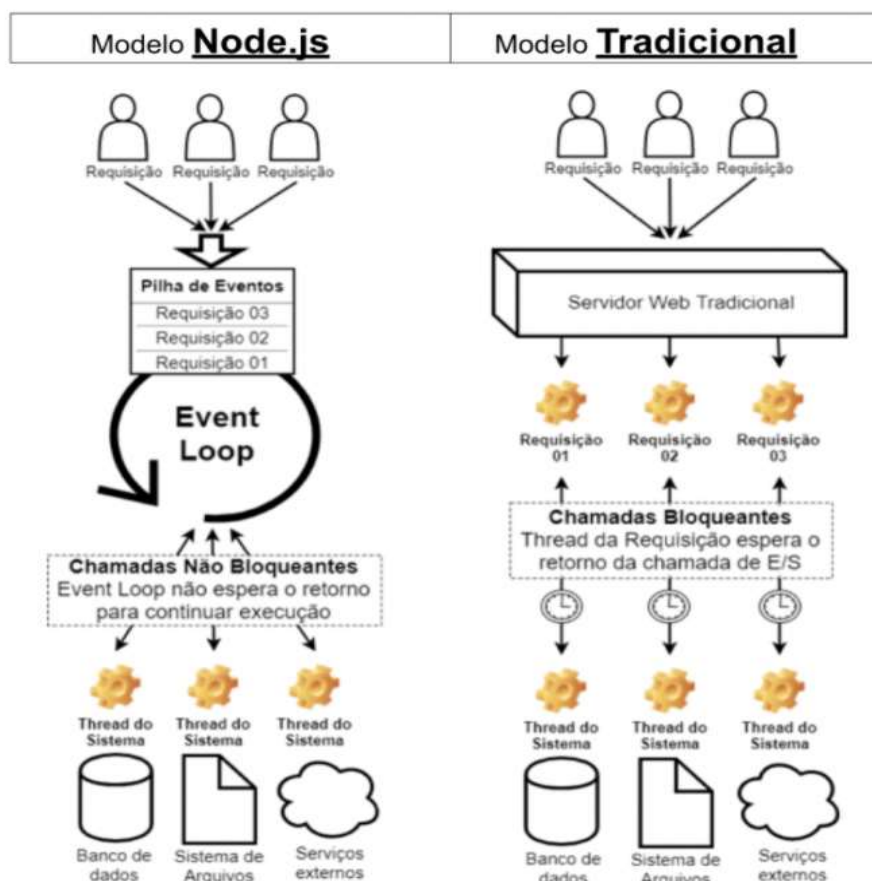
Em um servidor web utilizando linguagens tradicionais, para cada requisição recebida é criada uma nova *thread* para tratá-la. A cada requisição, serão demandados recursos computacionais (memória RAM, por exemplo) para a criação dessa nova *thread*. Uma vez que esses recursos são limitados, as *threads* não serão criadas infinitamente, e quando esse limite for atingido, as novas requisições terão que esperar a liberação desses recursos alocados para serem tratadas.

O Node.Js foi pensado em ser escalável, e se destaca pelo fato de trabalhar de forma assíncrona utilizando todas as APIs nativas com I/O (input e output) não bloqueante afirmando mais uma vez o poder de desempenho da tecnologia. Todo processamento acontece através de um event loop e uma única thread que fica capturando as requisições e juntamente com a sua API **não bloqueante** consegue processar diversas requisições ao mesmo tempo e respondendo com as que acabar em primeiro. Não existe um encadeamento pois nem sempre o primeiro a entrar será o primeiro a sair, tudo dependerá do nível de processamento da requisição.

Figura 2: Modelo Tradicional x Modelo Node.Js¹¹

¹⁰ <https://zlib.net>

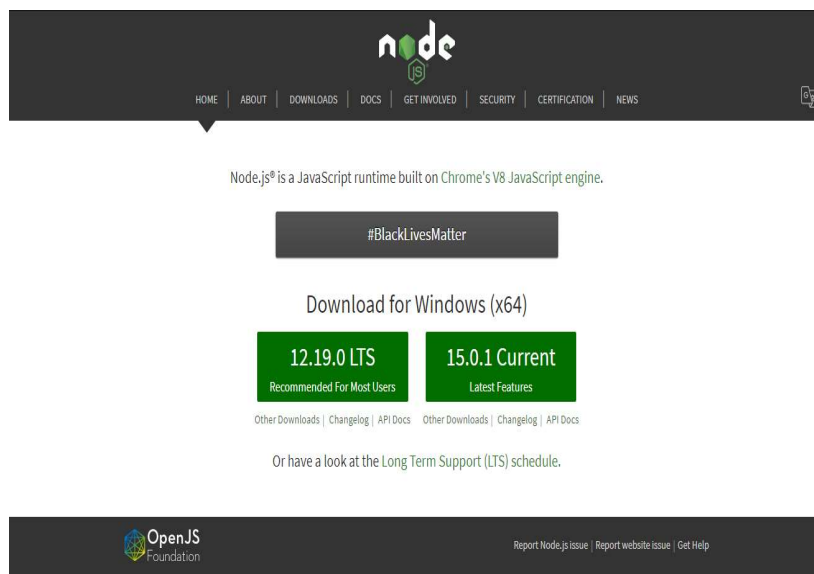
¹¹ <https://www.opus-software.com.br/node-js/>



13. Instalação Node.js

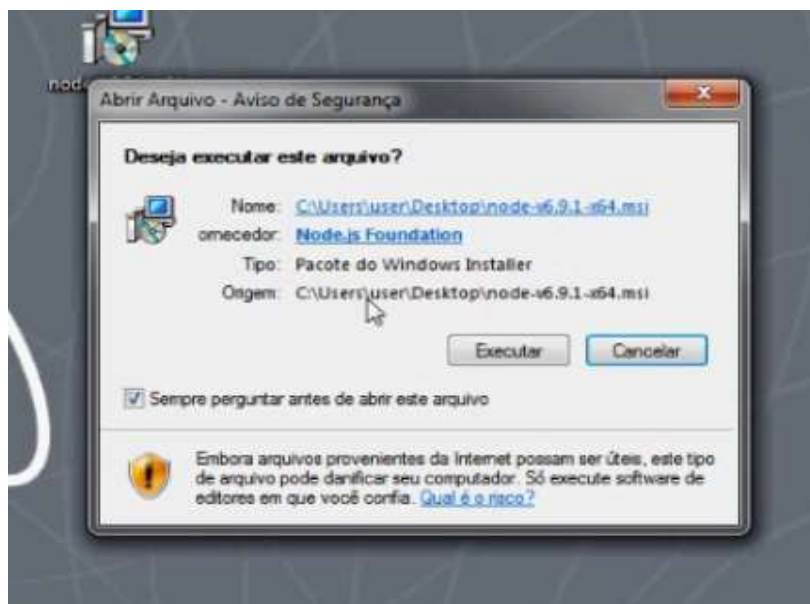
Para efetuar a instalação do Node.js, o primeiro passo é acessar o site referente a tecnologia: nodejs.org

Figura 3: Site nodejs.org



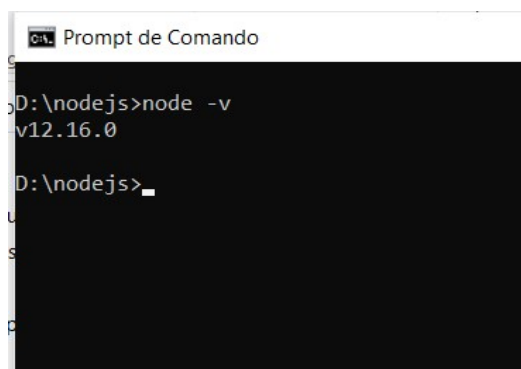
Com o site aberto, verificar a versão compatível com a máquina a ser instalada e selecionar a opção LTS, que como o próprio site indica é recomendável para a maior parte dos usuários por ser mais estável. Após o download, abra o arquivo baixado e execute-o.

Figura 4: Instalação do arquivo do Node.Js



Após a instalação, pode-se confirmar se está tudo ok, basta abrir o prompt de comando (cmd) e digitar o comando: `node -v`.

Figura 5: Versão do Node.Js



```
C:\> Prompt de Comando

D:\nodejs>node -v
v12.16.0

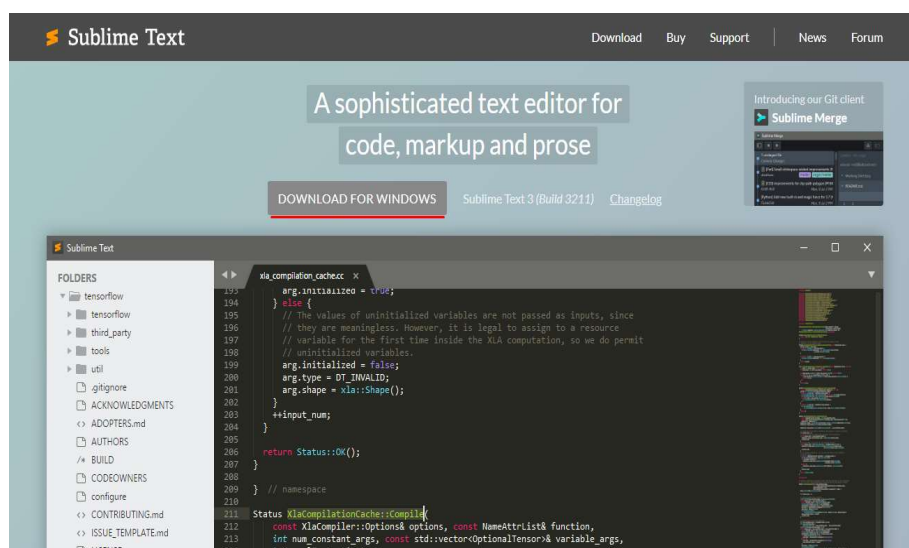
D:\nodejs>
```

4. Instalação do editor Sublime Text

Você poderá utilizar o editor que estiver acostumado ou gostar mais, o Sublime Text está sendo apenas sugerido como uma opção.

O Sublime Text é um editor de texto muito escolhido pelos desenvolvedores por ser leve, simples e com uma boa interface. Para instalar, acessar o site [sublimetext.com](https://www.sublimetext.com), no topo do site se encontra a opção de download, destacado em vermelho na imagem.¹²

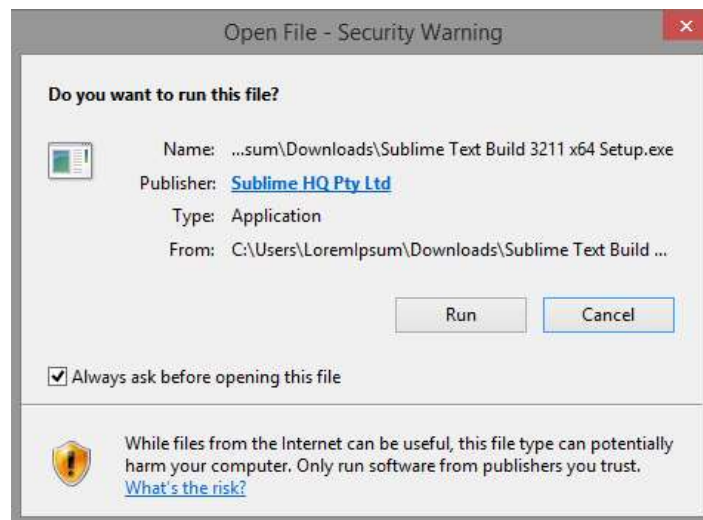
Figura 6: Site [sublimetext.com](https://www.sublimetext.com)



<https://www.sublimetext.com>

Após clicar nesta opção, o instalador será baixado no computador, basta instalar.

Figura 7: Instalação do Sublime text



5. Exercícios

A seguir são mostrados alguns exercícios no Node.Js.

5.1 Primeiro exercício

Antes dos exercícios, criar uma pasta chamada PWEBNode e dentro dela outra pasta chamada exercícios, onde serão gravados os exercícios.

Este Computador > DATA (D:) > PWEBNode

Nome	Data de modificação	Tipo	Tamanho
Exercicios	28/11/2020 17:40	Pasta de arquivos	

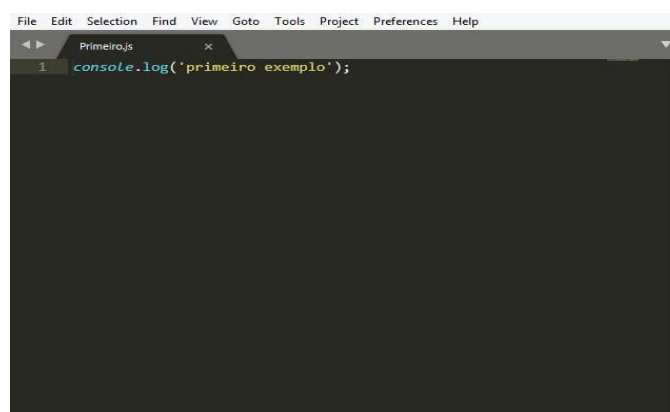
Com o Node.Js e o editor Sublime text instalado, criar o primeiro exercício para mostrar uma mensagem no console.

Abrir o editor Sublime Text 3 e nele escrever o seguinte comando:

```
cd
```

Salvar como Primeiro.js na pasta PWebNode\exercicios.

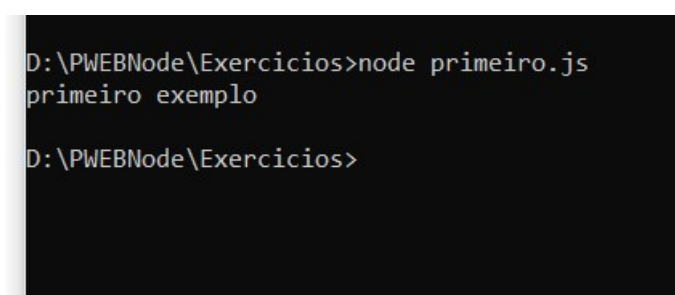
Figura 8: Primeiro.js



Por questões práticas, pelo cmd acesse a pasta PWebNode\exercícios. Para acessar o cmd, pode-se digitar cmd na barra do Explorer ou na barra de pesquisa, nessa última vai precisar de mais comandos para abrir a pasta desejada, como CD.

Quando estiver dentro da pasta exercícios, execute o comando: node Primeiro.js

Figura 9: Executando Primeiro.Js



5.2 Exercício 2

Nesse exercício será testada a diferença entre um código síncrono e outro assíncrono.

Digitar o código para o arquivo segundoS.js (S de síncrono)

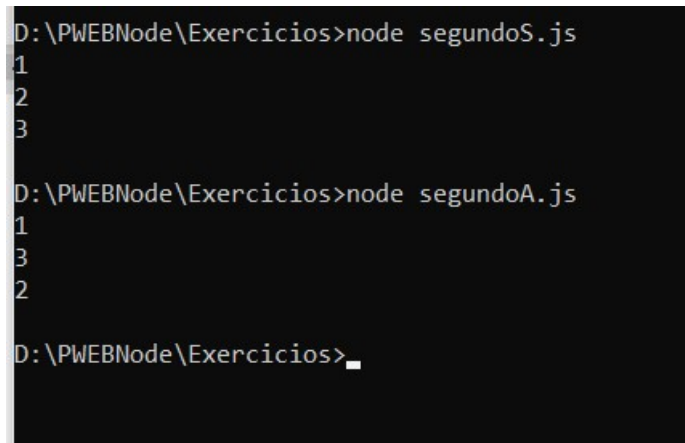
```
console.log('1');  
t();  
console.log('3');  
function t() {  
  console.log('2');  
}
```

Digitar o código para o arquivo segundoA.js (A de assíncrono)

```
console.log('1');  
t();  
console.log('3');  
function t() {  
  setTimeout(function() {  
    console.log('2');  
  }, 10);  
}
```

Testar os dois códigos no prompt.

Figura 10: Diferença entre código síncrono e assíncrono



```
D:\PWEBNode\Exercicios>node segundoS.js  
1  
2  
3  
  
D:\PWEBNode\Exercicios>node segundoA.js  
1  
3  
2  
  
D:\PWEBNode\Exercicios>_
```

5.3 Exercício 3

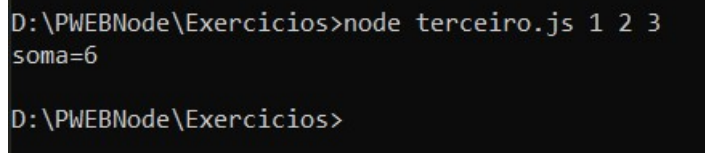
Nesse exercício será mostrado como acessar os argumentos da linha de comando através do objeto global process. O objeto process possui uma propriedade argv, que é um array contendo a linha de comando completa. Por exemplo: process.argv.

Digitar o código para somar os valores recebidos via argumentos (parâmetros) para o arquivo terceiro.js


```
var soma = 0;
for (var i=2; i<=process.argv.length-1; i++)
  soma=soma+Number(process.argv[i]);
console.log("soma="+soma);
```

Testar o arquivo terceiro.js passando parâmetros.

Figura 11: Execução do Exercício 3



```
D:\PWEBNode\Exercicios>node terceiro.js 1 2 3
soma=6

D:\PWEBNode\Exercicios>
```

5.4 Exercício 4

Nesse exercício será mostrado como utilizar o sistema de arquivos (*filesystem*) para ler e imprimir linhas.

Para realizar essa operação no sistema de arquivos (*filesystem*), será necessário incluir o módulo `fs` da *library* principal do Node.js. Para carregar esse tipo de módulo ou qualquer outro módulo "global", use o seguinte código:

```
var fs = require('fs');13
```

O comando `require` permite incorporar outros arquivos ao arquivo corrente, podendo importar bibliotecas, outras páginas etc.

Todos os métodos de sistema de arquivos síncronos (ou bloqueantes) no módulo `fs` terminam com `'Sync'`. Para ler um arquivo, usar `fs.readFileSync('caminho/do/arquivo')`. Esse método irá retornar um objeto `Buffer` contendo o conteúdo completo do arquivo.

Objetos `Buffer` são a maneira do Node.js representar eficientemente arrays arbitrários de dados, sejam eles `ascii`, binários ou quaisquer outros formatos.

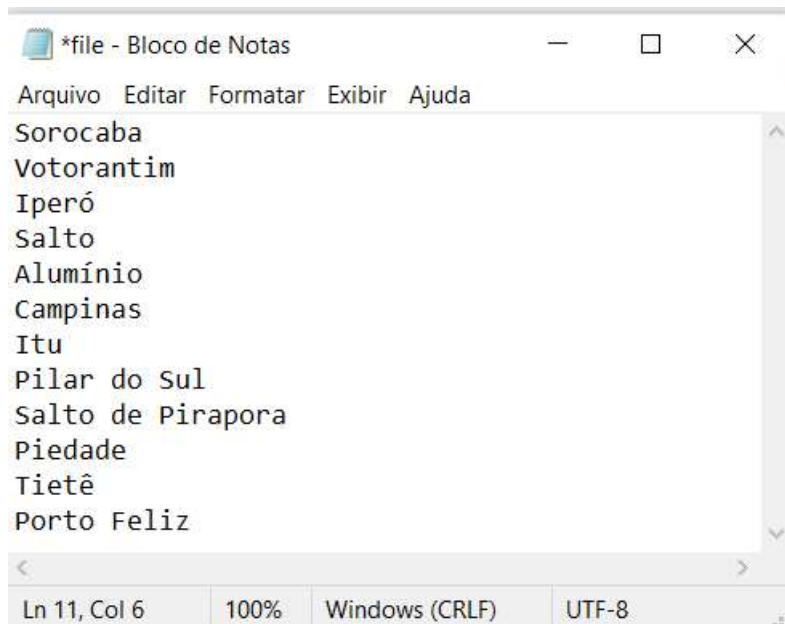
Objetos `Buffer` podem ser convertidos em strings invocando o método `toString()` neles. Por exemplo: `var str = buf.toString()`.¹⁴

¹³ <https://nodejs.org/api/fs.html>

¹⁴ <https://nodejs.org/api/buffer.html>

Criar um arquivo file.txt.

Figura 12: Arquivo file.txt

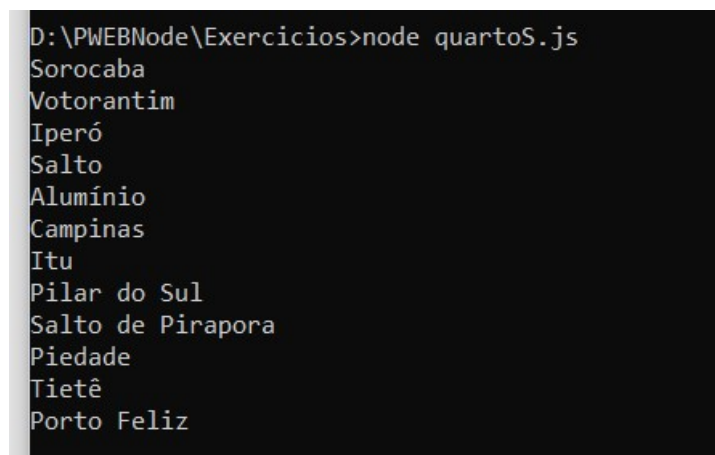


Digitar o código para ler o arquivo file.txt no arquivo quartoS.js

```
const fs = require('fs');  
const data = fs.readFileSync('file.txt');  
// a execução é bloqueada aqui até o arquivo ser lido  
console.log(data.toString());
```

Testar o arquivo quartoS.js

Figura 13: Execução do QuartoS



Ao invés de `fs.readFileSync()` usar `fs.readFile()` e ao invés de usar o valor de retorno desse método, será coletado o valor de uma função de callback¹⁵ que será passada como o segundo argumento.

Exemplos de funções callbacks:

Exemplo 1: `callback1.js`

```
const prompt = require('prompt-sync')();

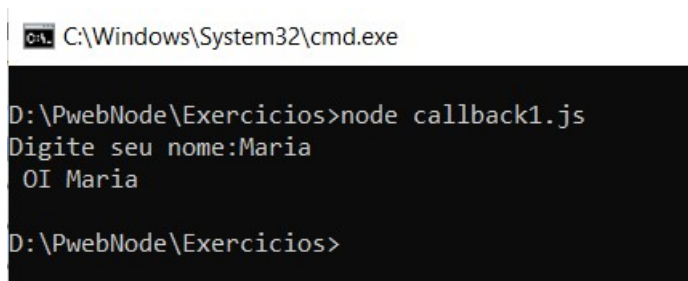
// não esquecer de instalar
// npm install prompt-sync

function saudacao(nome) {
  console.log(' Oi ' + nome);
}

function entradaNome(callback) {
  var nome = prompt('Digite seu nome:');
  callback(nome);
}

entradaNome(saudacao);
```

¹⁵ callback é um tipo de função que só é executada após o processamento de outra função. Na linguagem JavaScript, quando uma função é passada como um argumento de outra, ela é, então, chamada de callback. Isso é importante porque uma característica dessa linguagem é não esperar o término de cada evento para a execução do próximo. Portanto, ela contribui para controlar melhor o fluxo de processamento assíncrono.



```
C:\Windows\System32\cmd.exe

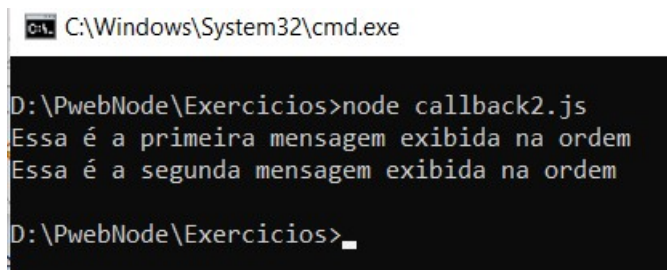
D:\PwebNode\Exercicios>node callback1.js
Digite seu nome:Maria
OI Maria

D:\PwebNode\Exercicios>
```

Exemplo 2: callback2.js

```
function exibeMensagensNaOrdem(mensagem, callback) {
    console.log(mensagem);
    callback();
}

exibeMensagensNaOrdem('Essa é a primeira mensagem exibida na ordem',
function() {
    console.log('Essa é a segunda mensagem exibida na ordem');
});
```



```
C:\Windows\System32\cmd.exe

D:\PwebNode\Exercicios>node callback2.js
Essa é a primeira mensagem exibida na ordem
Essa é a segunda mensagem exibida na ordem

D:\PwebNode\Exercicios>_
```

As tradicionais callbacks do Node.js normalmente têm a assinatura:

```
function callback (err, data) { /* ... */ }
```

Será necessário checar se um erro ocorreu checando se o primeiro argumento é verdadeiro. Se não houver nenhum erro, pode se ter o objeto Buffer como segundo argumento.

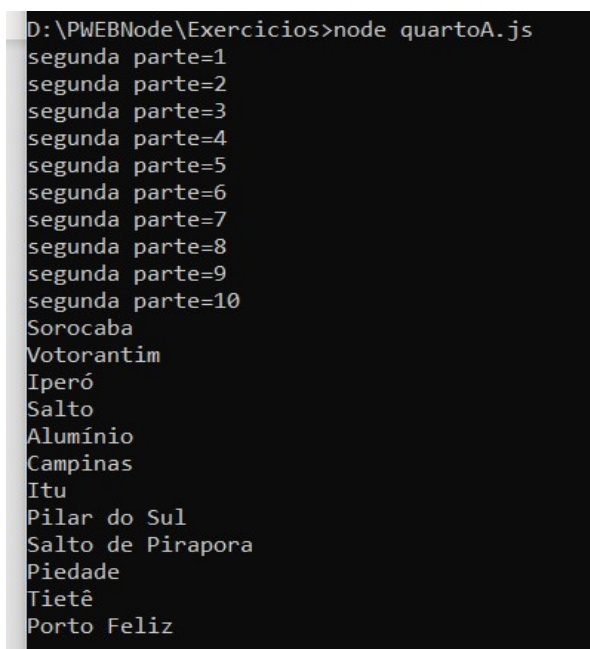
Digitar o código para ler o arquivo file.txt no arquivo quartoA.js

```
const fs = require('fs');
fs.readFile('file.txt', (err, data) => {
  if (err) throw err;
  console.log(data.toString());
});

for (var i=1; i<=10; i++)
  console.log("segunda parte="+i);
```

Testar o arquivo quartoA.js

Figura 14: Execução do QuartoA



```
D:\PWEBNode\Exercicios>node quartoA.js
segunda parte=1
segunda parte=2
segunda parte=3
segunda parte=4
segunda parte=5
segunda parte=6
segunda parte=7
segunda parte=8
segunda parte=9
segunda parte=10
Sorocaba
Votorantim
Iperó
Salto
Alumínio
Campinas
Itu
Pilar do Sul
Salto de Pirapora
Piedade
Tietê
Porto Feliz
```

No exemplo quartoS.js apesar de ser mais enxuto tem um código bloqueando execução de qualquer JavaScript adicional até que todo o arquivo seja lido. Na versão quartoA.js assíncrona, a execução é *single threaded*. Então a concorrência é referente somente à capacidade do event loop de executar funções de callback depois de completar qualquer outro processamento. Qualquer código que pode rodar de maneira concorrente deve permitir que o event loop continue executando enquanto uma operação não-JavaScript, como I/O, está sendo executada.

5.5 Exercício 5

Nesse exercício será mostrado como criados os eventos no lado do servidor. Os eventos do lado do servidor não têm nada a ver com os eventos Javascript que já são conhecidos e utilizados nas aplicações web do lado do cliente. Aqui os eventos são produzidos no servidor e podem ser de diversos tipos dependendo das bibliotecas ou classes que estejam trabalhando. Por exemplo, em um servidor HTTP teria o evento receber uma solicitação. Outro exemplo, em um stream de dados haveria um evento quando se recebe um dado como uma parte do fluxo.

Os eventos se encontram em um módulo independente `events`. Dentro desta biblioteca ou módulo há uma série de utilidades para trabalhar com eventos.

Para carregá-lo:

```
var eventos = require('events');
```

O emissor de eventos, encontra-se na propriedade `EventEmitter`.

```
var EmissorEventos = eventos.EventEmitter;
```

Em Node.js existe um loop de eventos, de modo que quando um evento, o sistema fica escutando no momento que se produz, para executar então uma função. Essa função é conhecida como "callback" ou como "manipulador de eventos" e contém o código que você quer que seja executado no momento que se produza o evento ao que a associamos. Primeiro deve-se que "instanciar" um objeto da classe `EventEmitter`, através variável `EmissorEventos`.

```
var ee = new EmissorEventos();
```

O objeto `EventEmitter` temos vários métodos interessantes para emitir, cadastrar e processar eventos, por exemplo o método `on()`, onde registra-se um evento, e `emit()` onde emite-se de fato o evento, mediante código Javascript.

Usa-se o método `on()` para definir as funções manipuladoras de eventos, ou seu equivalente `addEventListener()`.

O método `Date.now()` retorna o número de milisegundos decorridos desde 1 de janeiro de 1970 00:00:00 UTC.

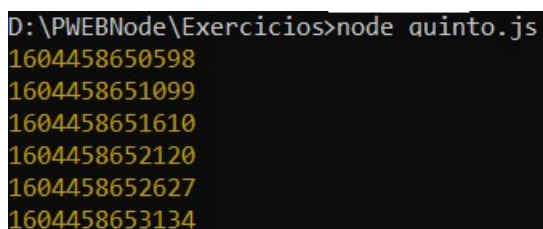
Para aproveitar algumas das características mais interessantes de aplicações NodeJS será usado `setInterval()` para emitir dados a cada intervalo de tempo.

Digitar o código no arquivo quinto.js

```
var eventos = require('events');
var EmissorEventos = eventos.EventEmitter;
var ee = new EmissorEventos();
ee.on('dados', function(fecha){
    console.log(fecha);
});
setInterval(function(){
    ee.emit('dados', Date.now());
}, 500);
```

Testar o arquivo quinto.js

Figura 15: Execução do Exercício quinto.js



```
D:\PWEBNode\Exercicios>node quinto.js
1604458650598
1604458651099
1604458651610
1604458652120
1604458652627
1604458653134
```

6. Respondendo requisições HTTP

Antes a interpretação do código JavaScript ficava a cargo dos navegadores, com o Node é possível trazer essa realização para aplicações desktop, bastando apenas submeter os scripts para que o Node.JS interprete o comando e converta tais comandos em linguagem de máquina, permitindo portanto o controle de recursos do sistema operacional. **Do ponto de vista do desenvolvimento web o Node.js implementa diversos recursos que tornam capaz de responder a diversos tipos de protocolos diferentes, dentre esses protocolos o HTTP.** Isso significa que, apesar do Node.js não ser um servidor HTTP propriamente dito, é possível implementar os recursos necessários para que a aplicação seja capaz de responder a requisições feitas a partir deste protocolo.

É necessário importar uma biblioteca chamada HTTP e a partir dessa biblioteca pode-se criar um servidor e com isso passar a “escutar” requisições que são feitas em uma porta específica.

Para criar (ou subir) um servidor usar o método: `http.createServer`

Esse método vai receber um atributo function com dois parâmetros: uma requisição e uma resposta.

A função vai devolver para o requisitante um código html.

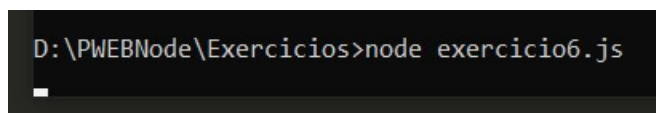
Também é necessário informar para o servidor qual é a porta que ele deve “escutar”.

Digitar o código para criar o servidor no arquivo exercicio6.js.

```
var http = require('http');
var server = http.createServer( function(req,res){
    res.end("<html><body>Site da Fatec Sorocaba</body></html>");
});
server.listen(3000);
```

No prompt executar exercicio6.js e ele vai ficar aguardando (escutando) as requisições.

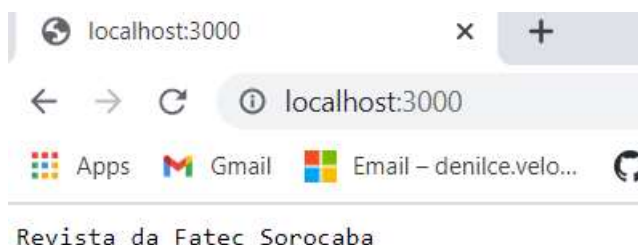
Figura 16: Execução do exercicio6.js



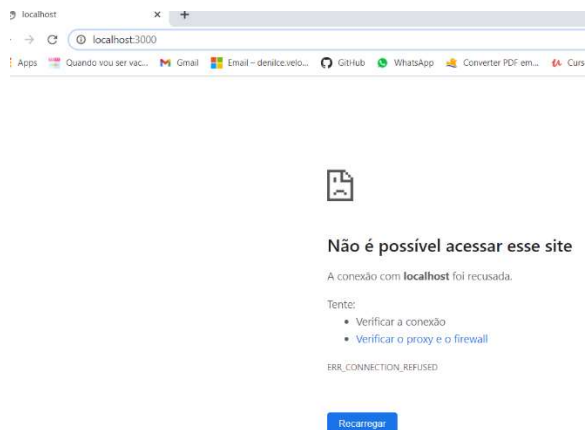
```
D:\PWEBNode\Exercicios>node exercicio6.js
```

Abrir o navegador e digitar localhost:3000, a tela abaixo será apresentada, porque encontrou o servidor.

Figura 17: Execução do exercicio6.js (chamando o browser)



Para “derrubar” o servidor basta dar ctrl+c, e é claro que a chamada no browser vai dar erro.



7. Respondendo requisições usando a URL

As requisições na web são feitas a uma URL (*Uniform Resource Locator*), que nada mais é do que o caminho do site. Um site pode ter muitas URLs.

Digitar o código para criar mais algumas urls no arquivo `exercicio7.js`.

```
var http = require('http');
var server = http.createServer( function(req,res){
  var opcao = req.url;
  if (opcao=='/historia') {
    res.end("<html><body>História da Fatec Sorocaba</body></html>");
  }
  else if (opcao=='/cursos') {
    res.end("<html><body>Cursos</body></html>");
  }
  else if (opcao=='/professores') {
    res.end("<html><body>Professores</body></html>");
  }
  else
  res.end("<html><body>Site da Fatec Sorocaba</body></html>");
});
server.listen(3000);
```

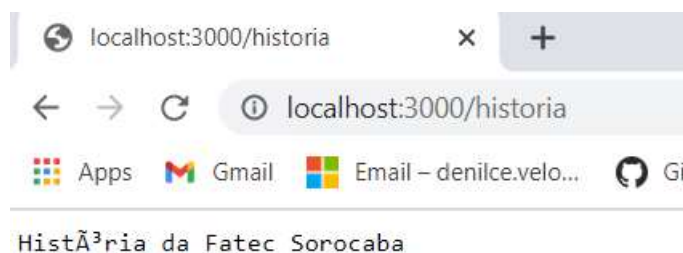
No prompt executar `exercicio7.js` e ele vai ficar aguardando as requisições.

Figura 18: Execução do `exercicio7.js`

```
D:\PWEBNode\Exercicios>node exercicio7.js
```

Abrir o navegador e digitar localhost:3000/historia, a tela abaixo será apresentada, porque encontrou o servidor e a url desejada.

Figura 19: Execução do exercicio7.js (chamando no browser a opção historia)



Repetir para localhost:3000/cursos e localhost:3000/professores.

Observe que toda vez que fizer uma alteração no código será necessário reiniciar o servidor do Node. Mais para frente será utilizada a ferramenta Nodemon para auxiliar nesse problema.

8. Ferramentas para auxiliar no Desenvolvimento

Serão citadas aqui algumas ferramentas que podem auxiliar no desenvolvimento com Node.js.

8.1 NPM

O Npm (*Node Package Manager*) é um serviço (grátis para pacotes públicos) que possibilita aos desenvolvedores criar e compartilhar módulos com outros desenvolvedores, dessa forma facilitando a distribuição de códigos e ferramentas escritas em JavaScript. O lado “bom” do npm é que existem muitos módulos a disposição e o lado “ruim” é que como ele é open-source existem

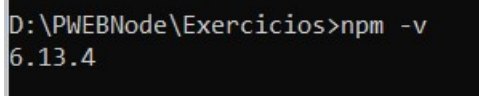
muitos módulos que muitas vezes realizam a mesma tarefa, e aí a necessidade de descobrir o que mais se adapta a necessidade do desenvolvedor. O Npm já foi instalado junto com o Node.js e será utilizado na instalação (inclusão) das demais ferramentas.¹⁶

Para verificar a versão do NPM digitar no prompt:

`npm -v`

A tela apresentada será parecida com essa:

Figura 20: Versão do NPM



```
D:\PWEBNode\Exercicios>npm -v
6.13.4
```

O próximo passo será customizar o npm, que no final vai gerar um arquivo `package.json`¹⁷, para isto digitar no prompt:

`npm init`

Figura 21: Execução npm init

¹⁶ <https://docs.npmjs.com/about-npm>

¹⁷ JSON (*JavaScript Object Notation*) é um modelo para armazenamento e transmissão de informações no formato texto. É bem simples, legível para as pessoas e tem sido bastante utilizado por aplicações Web devido a sua capacidade de estruturar informações de uma forma bem mais compacta do que o formato XML.

```
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.

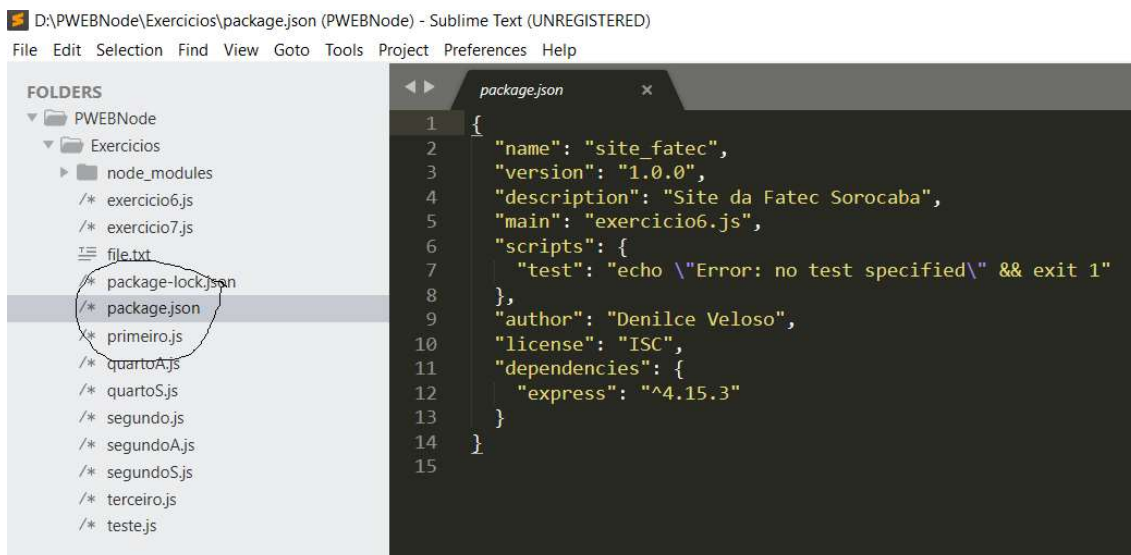
Press ^C at any time to quit.
package name: (exercicios) site_fatec
version: (1.0.0)
description: Site da Fatec Sorocaba
git repository:
keywords:
author: Denilce Veloso
license: (ISC)
About to write to D:\PWEBNode\Exercicios\package.json:

{
  "name": "site_fatec",
  "version": "1.0.0",
  "description": "Site da Fatec Sorocaba",
  "main": "exercicio6.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Denilce Veloso",
  "license": "ISC"
}
```

Se estiver tudo certo, ok.

Ele incluiu o arquivo package-json, importante pois se for instalar a aplicação em algum lugar, ele já sabe quais são as dependências necessárias e suas versões para baixar.

Figura 22: Arquivo package-json



8.2 Instalação do Express

O Express é um popular *framework* web estruturado, escrito em JavaScript que roda sobre o ambiente node. Ele fornece API para controle de rotas e permite adicionar novos processos de requisição por meio de "middleware"¹⁸ em qualquer ponto da "fila" de requisições.¹⁹

Para instalar o Express digitar no prompt:

```
npm install express@4.15.3 --save
```

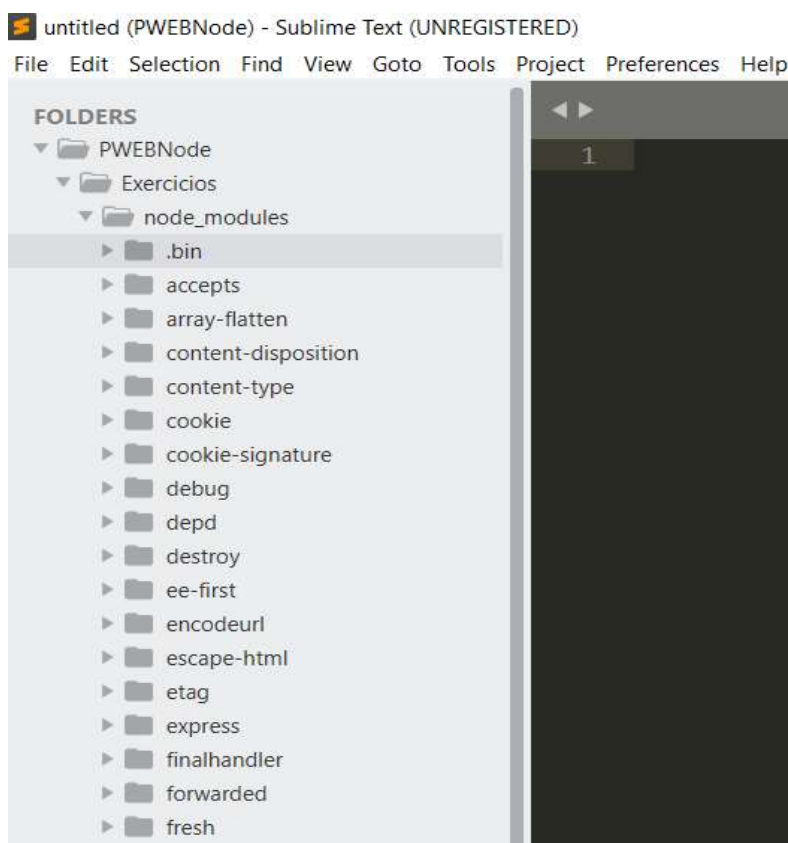
-save -> cria diretório e traz os arquivos de fatos (dependências) para dentro da nossa máquina, não precisa ficar procurando.

Ele cria uma pasta **node_modules** (com todos os recursos que podem ser utilizados para otimizar o desenvolvimento) dentro do nosso projeto.

Figura 23: Pasta node_modules

¹⁸ O middleware é o software que se encontra entre o sistema operacional e os aplicativos nele executados.

¹⁹ <https://expressjs.com/pt-br/>



8.1 Instalação do EJS

O EJS (*Embedded JavaScript Templating*) é uma engine de visualização, sendo possível transportar dados do back-end para o front-end, utilizando códigos em javascript no html das páginas.

O EJS permite criar HTML junto as instruções JavaScript.²⁰

Para instalar o EJS digitar no prompt:

```
npm install ejs@2.5.6 --save
```

Ele vai ficar disponível no node_modules

Figura 24: Pasta node_modules\ejs

²⁰ <https://medium.com/@pedrompinto/tutorial-node-js-como-usar-o-engine-ejs-12bcc688ebab>



8.2 Instalação do Nodemom

O Nodemom é um utilitário que reinicia automaticamente o servidor NodeJS quando houver qualquer alteração no código. Quando a aplicação for instalada em outro servidor o Nodemom vai ficar nesse servidor.

Uma das vantagens de linguagens interpretadas é só alterar o script e tentar executar sem compilar, o Nodemom vai facilitar isso no caso do Node para não necessitar subir o servidor toda vez que fizer alteração no código.

Como ele é um utilitário e não um módulo do projeto, ele pode ficar na área de desenvolvimento, quanto no servidor onde a aplicação de fato estiver instalada.

Para instalar o Nodemom digitar no prompt:

```
npm install -g nodemon@1.10.2
```

Para iniciar o servidor usando Nodemon basta digitar no prompt:

```
nodemon app.js
```

Por enquanto não temos o arquivo, mas poderia testar por exemplo com o exercicio7.

Figura 25: Teste nodemon

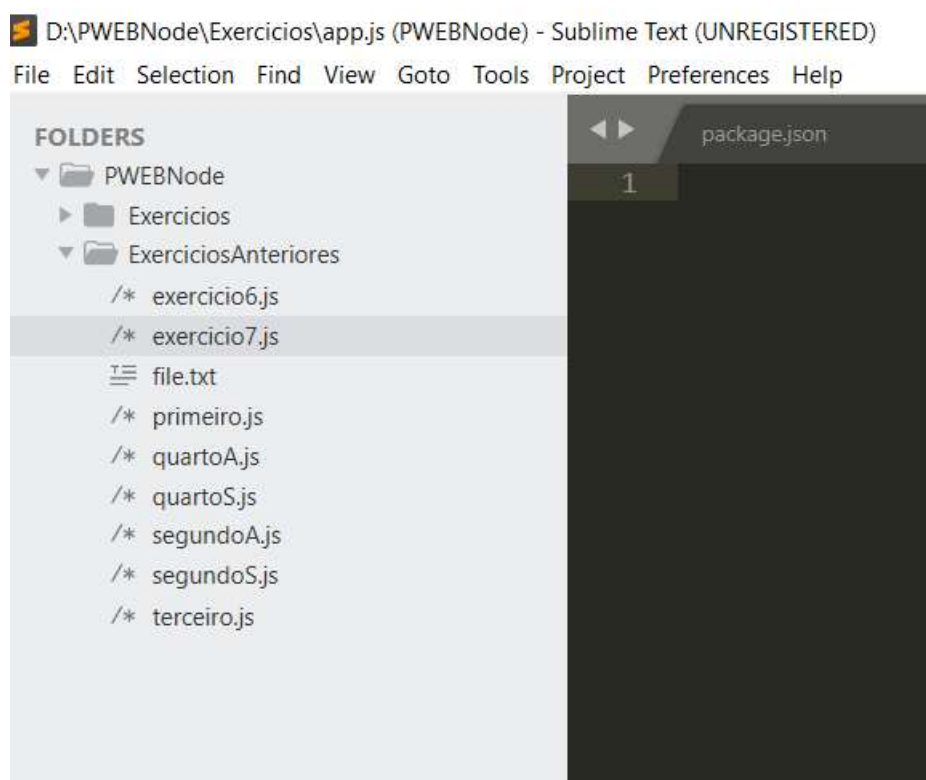
```
D:\PWEBNode\ExerciciosAnteriores>nodemon exercicio7.js
[nodemon] 1.10.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node exercicio7.js`
```

9. Utilizando as novas ferramentas

9.1 Guardando os exercícios anteriores

Com as novas ferramentas instaladas, será desenvolvido um novo exercício. Primeiro passo é criar uma pasta ExerciciosAnteriores para “guardar” os exercícios realizados (**copiar primeiro, segundo, ... exercicio7**) até agora (**somente os exercícios, tomar cuidado para não copiar node_modules, package.json e etc**).

Figura 25: Pasta ExerciciosAnteriores



9.2 Utilizando o Express

Quem executa o código JavaScript é o Node e o *framework* Express é uma camada que fica acima do Node para fazer a interface entre os scripts e o Node,

portanto é necessário que a aplicação seja feita com base na estrutura que o *framework* Express exige.

REVISAR A PARTIR DAQUI

Criar um arquivo `app.js` dentro da pasta `exercícios`, observe `exercício7.js` para fins de comparação.

Figura 26: `exercício7.js` e `app.js`

```
var express = require('express');
var app=express(); // executando o express
app.listen(3000, function(){
    console.log("servidor com express foi carregado");
});
```

```
1 var http = require('http');
2 var server = http.createServer( function(req,res){
3   var opcao = req.url;
4   if (opcao=='/historia') {
5     res.end("<html><body>História da Fatec Sorocaba</body></html>");
6   }
7   else if (opcao=='/cursos') {
8     res.end("<html><body>Cursos</body></html>");
9   }
10  else if (opcao=='/professores') {
11    res.end("<html><body>Professores</body></html>");
12  }
13  else
14    res.end("<html><body>Site da Fatec Sorocaba</body></html>");
15  } );
16  server.listen(3000);
17
```

Mudar o `require` para `express` e mudar também o nome da variável.

O `express` retorna uma função, precisa executar essa função. E o método `listen` fica “escutando” as requisições em uma determinada porta e precisa passar uma função de *callback* como anteriormente. Dentro da função fazer um teste, por

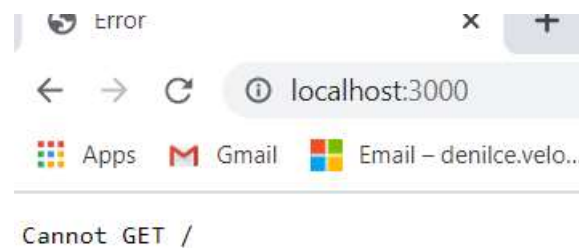
exemplo, com `console.log("servidor com express foi carregado")`, só para ver se o servidor subiu.

Figura 27: Teste do servidor com express

```
D:\PWEBNode\Exercicios>node app.js
servidor com express foi carregado
```

Executar no browser

Figura 28: Teste local do servidor



O Express tratou e identificou que não existe nenhum caminho (página / resposta). Antes precisava tratar a url, agora pode usar a função `get`, incluir a chamada para a página principal (o `/`) e usar o método `send` por estar trabalhando direto com o Express, quando estava trabalhando direto com o node era `end`.

Alterar o arquivo `app.js` para que os caminhos sejam encontrados.

```
var express = require('express');
var app=express(); // executando o express

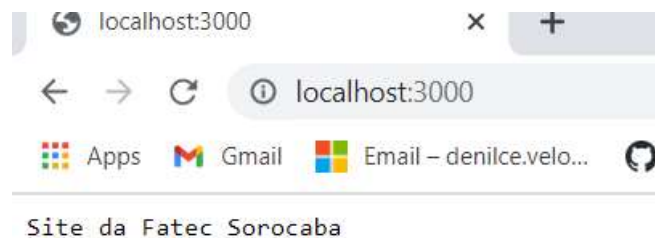
app.get('/', function(req,res){
  res.send("<html><body>Site da Fatec Sorocaba</body></html>");
});

app.get('/historia', function(req,res){
  res.end("<html><body>História da Fatec Sorocaba</body></html>");
});
```

```
app.get('/cursos', function(req,res){  
  res.end("<html><body>Cursos da Fatec Sorocaba</body></html>");  
});  
  
app.get('/professores', function(req,res){  
  res.end("<html><body>Professores da Fatec Sorocaba</body></html>");  
});  
  
app.listen(3000, function(){  
  console.log("servidor com express foi carregado");  
});
```

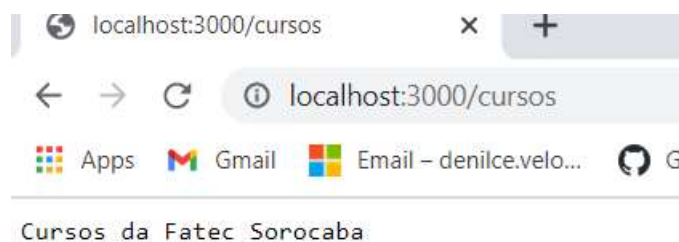
Para o servidor e tentar fazer a requisição novamente.

Figura 29: Teste local



Se tentar chamar a página de Cursos por exemplo.

Figura 30: Teste da opção Cursos



O código fica mais simples utilizando o Express, do que ficar utilizando aqueles ifs como no exemplo exercício7.js, fica mais fácil organizar as rotas.

9.3 Utilizando o EJS

Será utilizando o EJS (linguagem de modelagem), utilizando o Get junto com a função Render do EJS.

Alterar o arquivo app.js, o primeiro passo é informar no app que o engine de view do express passou a ser ejs → `app.set('view engine', 'ejs');`

Agora ao invés do método send, com o ejs pode ser utilizado o render e passar o arquivo que será renderizado, alterar as rotas.

```
var express = require('express');
var app=express(); // executando o express

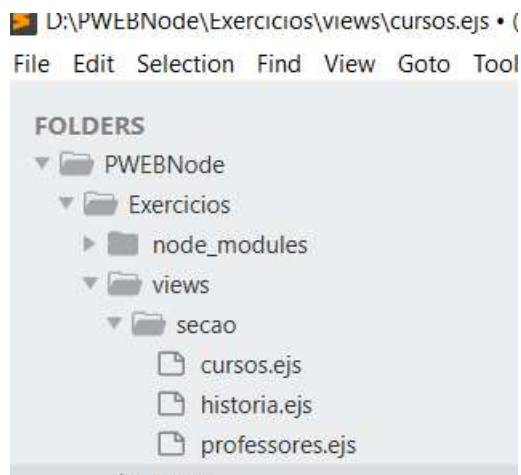
app.set('view engine', 'ejs');

app.get('/', function(req,res){
  res.send("<html><body>Site da Fatec Sorocaba</body></html>");
});

app.get('/historia', function(req,res){
  res.render("secao/historia");
});
app.get('/cursos', function(req,res){
  res.render("secao/cursos");
});
app.get('/professores', function(req,res){
  res.render("secao/professores");
});
app.listen(3000, function(){
  console.log("servidor com express foi carregado");
});
```

O próximo passo arrumar a estrutura da aplicação, criando um new folder chamado views dentro da pasta Exercícios. E dentro dele serão criados os arquivos com extensão ejs. Criar um secao e dentro dela criar os arquivos (opções) historia, cursos e professores.

Figura 31: Pasta views e seção



No arquivo **historia.ejs** digitar o código abaixo:

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <title> História da Fatec Sorocaba</title>
</head>

<body>
  <p> A Faculdade de Tecnologia de Sorocaba foi criada em 20/05/1970 pelo então Governador do Estado de São Paulo, Dr. Roberto Costa de Abreu Sodré. Foi a primeira escola pública de nível superior em Sorocaba.
  </p>
  <br>
  <br>
  <p>
    O primeiro dia letivo na Fatec Sorocaba ocorreu no dia 07/06/1971, nas instalações da atual Etec Rubens de Faria e Souza com 66 alunos que iniciavam seus estudos no então Curso Técnico Superior de Oficinas, atualmente Tecnologia em Fabricação Mecânica.
  </p>
</body>

</html>
```

No arquivo cursos.ejs digitar o código:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title> Cursos Fatec Sorocaba</title>
</head>
<body>
  <ul> Os cursos da Fatec Sorocaba são:
  <li>Análise e Desenvolvimento de Sistemas</li>
  <li>Eletrônica Automotiva</li>
  <li>Fabricação Mecânica</li>
  <li>Gestão da Qualidade</li>
  <li> Logística</li>
  <li>Manufatura Avançada</li>
  <li>Processos Metalúrgicos</li>
  <li> Polímeros</li>
  <li>Projetos Mecânicos</li>
  <li>Sistemas Biomédicos</li>
  <li> Gestão Empresarial - EAD</li>
</ul>
</body>
```

```
</html>
```

No arquivo professores.ejs digitar o código:

```
<!DOCTYPE html>
<html lang="en">

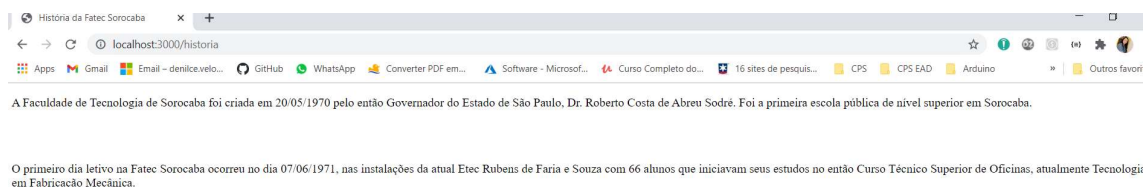
<head>
  <meta charset="UTF-8">
  <title> Professores Fatec Sorocaba</title>
</head>

<body>
  <h1>Está é a lista do corpo docente da Fatec Sorocaba</h1>
</ul>
</body>
</html>
```

Recarregue o servidor com o arquivo app.js.

Através do browser procurar a página historia por exemplo, ficará assim:

Figura 32: Secao historia



9.4 Testando o Nodemon

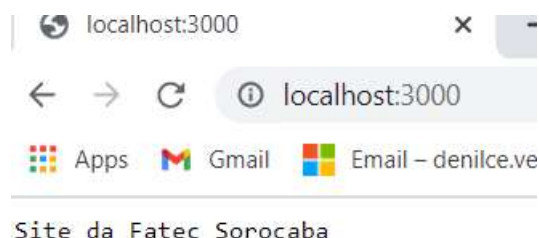
Sair do servidor e executar nodemon app.js

Figura 33: Teste do Nodemon

```
D:\PWEBNode\Exercicios>nodemon app.js
[nodemon] 1.10.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
servidor com express foi carregado
```

Tentar executar a pasta principal

Figura 34: Teste do site principal

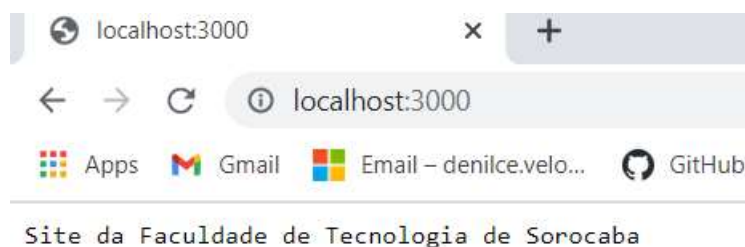


Se for feita qualquer alteração no arquivo app.js, automaticamente ele restarta o servidor considerando a alteração. Supondo que fosse alterada a linha 8 do arquivo app.js

```
res.send("<html><body>Site da Faculdade de Tecnologia de Sorocaba</body></html>");
```

Tentar executar a pasta principal novamente

Figura 35: Novo Teste do site principal



Observar que ele aceitou a mudança e não precisou parar e subir o servidor novamente.

9.5 Organizando melhor a aplicação

Sobre o encoding

Encoding é o mecanismo que define como são apresentados os diversos símbolos e letras de diferentes alfabetos de maneira binária. Não está diretamente ligado ao idioma, só com os símbolos. O mesmo caractere á pode ser representado de diversas maneiras, com 1, 2 ou mais bytes, dependendo do encoding utilizado. E existem diversas maneiras de armazenar esses caracteres em disco/memória. Por exemplo, a palavra olá, convertida usando encoding diferentes:

ISO-8859-1

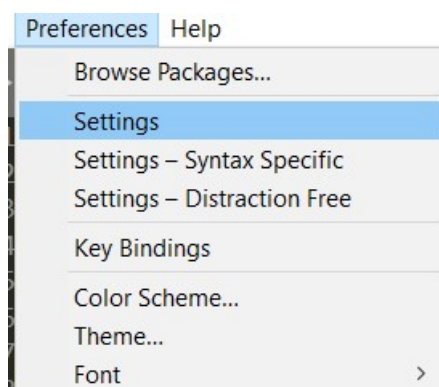
Letras	o	l	á
Hexadecimal	6f	6c	e1
Binário	01101111	01101100	11100001

UTF-8

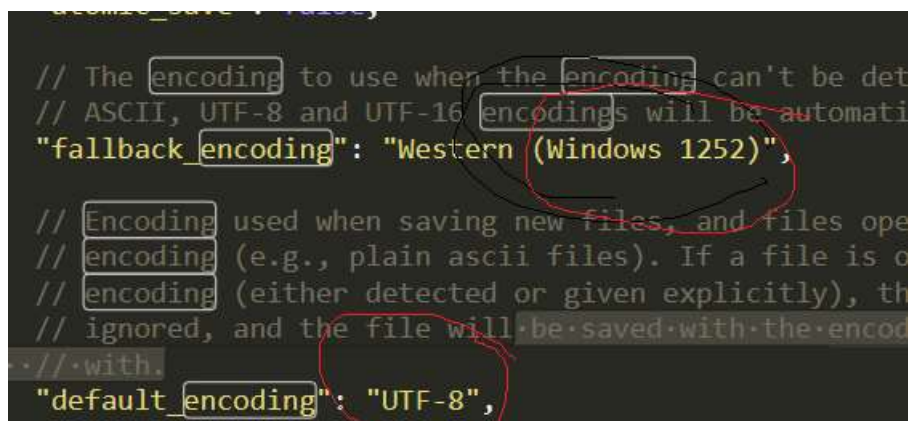
Letras	o	l	á	
Hexadecimal	6f	6c	c3	a1
Binário	01101111	01101100	11000011	10100001

A questão que alguns idiomas existem vários caracteres, como no português tem vários acentos e pode acabar aparecendo de forma errada na tela.

Para resolver isso, em páginas web, temos que nos atentar ao cabeçalho `<meta>`²¹, usado para informar aos navegadores o encoding da página e usar o encoding correto para salvar o arquivo. É mais fácil configurar no editor, em qual o encoding que o arquivo deve ser gravado. No caso do editor Sublime Text 3 ir em Preferences\Settings



Procurar com Ctrl+F → encoding



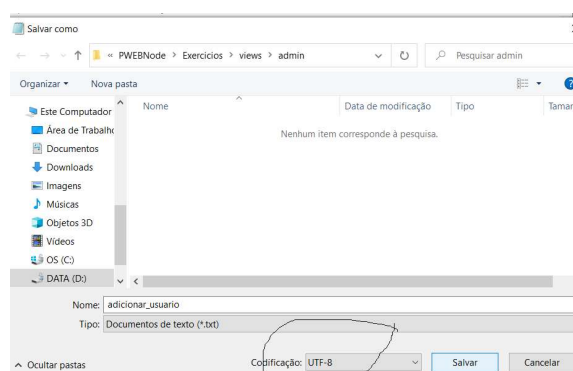
As configurações default (lado esquerdo) podem ser vistas, mas não podem ser modificadas. Para modificar uma configuração default do Sublime Text 3 você precisará aplicar essa modificação nas configurações do usuário (lado direito), para fazer isso basta informar o JSON com os valores atualizados separados por vírgula.

²¹ `<head>`
 `<meta charset="utf-8"/>`
`</head>`

```
// and are overridden in turn by sync  
{  
  "fallback_encoding": "UTF-8"  
}
```

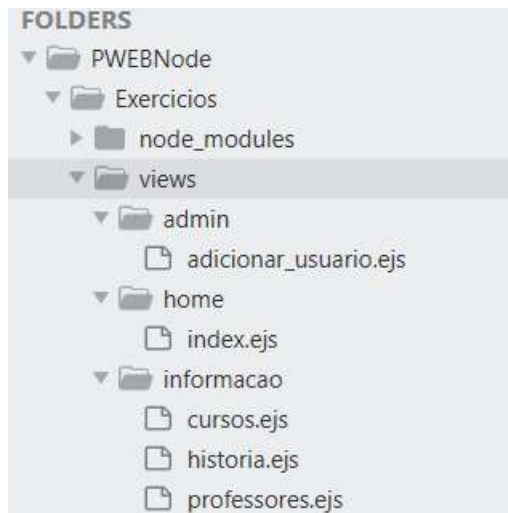
Até no bloco de notas é possível salvar o arquivo no formato utf8. Basta abrir o arquivo no bloco de notas e mudar a opção codificação e salvar novamente o arquivo.

Figura 37: Salvando o arquivo no formato UTF-8



Renomear a pasta views para views anteriores e criar uma pasta views com outras pastas dentro dela (admin, home e informação), e dentro de cada pasta criar alguns arquivos .ejs. Na pasta informacao copiar os arquivos .ejs das views anteriores, na pasta home criar um arquivo index.ejs e na pasta admin criar um arquivo adicionar_usuario.ejs.

Figura 36: Nova Pasta views



Veja sugestão para arquivo adicionar_usuario.ejs

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title> Adicionar usuário </title>
</head>
<body>
  <p> adicionar usuário</p>
</body>
</html>
```

Veja sugestão para arquivo index.ejs

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title> Página Principal </title>
</head>
<body>
  <p> Esta é a página principal </p>
</body>
</html>
```

Alterar o código do arquivo app.js

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

var app=express(); // executando o express

app.set('view engine', 'ejs');

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

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

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

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

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

app.listen(3000, function(){
  console.log(" servidor express carregado");
});
```

Recarregar o servidor e testar os novos caminhos

http://localhost:3000/formulario_adicionar_usuario

<http://localhost:3000/>

<http://localhost:3000/informacao/historia>

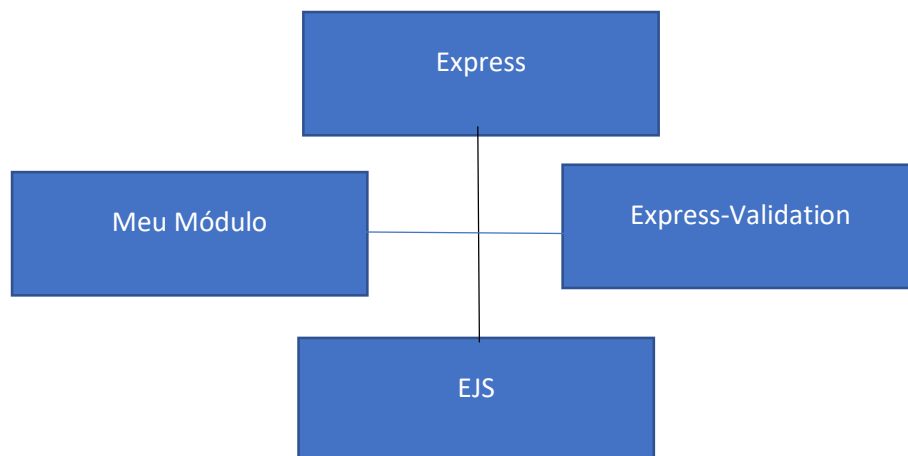
<http://localhost:3000/informacao/professores>

<http://localhost:3000/informacao/cursos>

9.5.1 Módulos e CommonJS

O que são esses módulos e como podem ser construídos?

Figura 38: Estrutura Módulo



No Node.js, um module é uma coleção de funções e objetos do JavaScript que podem ser utilizados por aplicativos externos. Um js pode ser considerado um módulo, caso suas funções e dados sejam feitos para programas externos. Serve para organizar melhor o código e facilitar a manutenção (por exemplo, o que fazemos com as classes). Um módulo pode ser disponibilizado na web para ser utilizado em outros projetos. Exemplos de Módulos: Express e EJS. O módulo pode retornar um objeto, uma função, uma string etc.

Para incluir um módulo, como já visto anteriormente, utiliza-se a função require. "Common JS", diz respeito ao formato de construção dos módulos. É uma API com o objetivo de agrupar as necessidades de diversas aplicações Javascript em uma única API, que funcione em diversos ambientes e interpretadores. Criando o conceito de se módulos que façam essas funções.

Criar um arquivo novo chamado modulo1.js na pasta Exercicios.

```
var texto = "Observe que essa mensagem vem do módulo";
```

Alterar o código do arquivo app.js para utilizar um módulo:

```
var express = require('express');  
var texto = require('./modulo1'); //não precisa colocar o .js ele já entende  
  
var app=express(); // executando o express  
  
app.set('view engine', 'ejs');  
  
app.get('/', function(req,res){  
    res.render("home/index");  
});
```

```
app.get('/formulario_adicionar_usuario', function(req,res){
  res.render("admin/adicionar_usuario");
});

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

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

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

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

Precisa usar exports no modulo1, senão ele não encontrará o texto, pois tentar executar no console não mostrará o texto.

```
var texto = "Observe que essa mensagem vem do módulo";
module.exports = texto;
```

Testar inicialmente chamando o app.js

Figura 39: Teste chamando o módulo

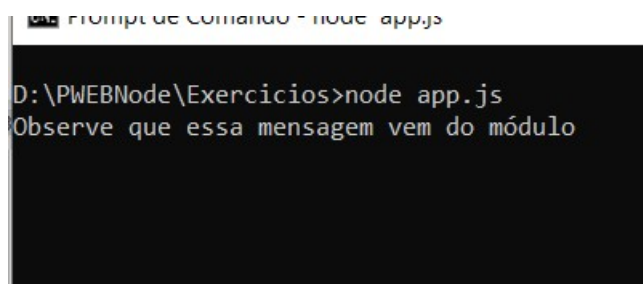


Figura 40: Teste chamando a página



Se o teste for realizado usando o nodemon a tela apresentada será assim:

Figura 41: Teste chamando o módulo usando nodemon

```
D:\PWEBNode\Exercicios>nodemon app.js
[nodemon] 1.10.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Observe que essa mensagem vem do módulo
```

Quando o módulo retornar uma função, precisa executar a função. Embora o módulo possa ter muitos códigos, o que será recebido será o que está no exports. Observe na pasta node_modules por exemplo os arquivos do ejs aparece o exports.

9.5.2 Modularizando a aplicação

Separar código de infraestrutura e regras de negócios traz muitas vantagens como por exemplo facilitar manutenções.

Separando as configurações do servidor

Para separar as configurações do servidor das demais, será necessário criar uma pasta app dentro da pasta exercícios e uma pasta config dentro de app. E dentro da pasta config criar um arquivo (módulo) server.js.

Figura 42: Pasta config



O código do arquivo server.js é o seguinte, observar que é parte do código que estava no arquivo app.js.


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

var app=express(); // executando o express

app.set('view engine', 'ejs'); // o mecanismo de engine a ser usado
app.set('views', './app/views'); // diretório onde os arquivos estão localizados

module.exports = app;
```

O arquivo app.js fica conforme a seguir, não esquecer de chamar o módulo servidor:

```
var app = require('./app/config/server'); // carregando o módulo do servidor

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

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

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

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

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

app.listen(3000, function(){
  console.log("servidor iniciado");
});
```

Antes do teste, fazer uma cópia da pasta views (manter a antiga para consultas) para dentro da pasta app. Deverá ficar assim:

Figura 43: Cópia da pasta views



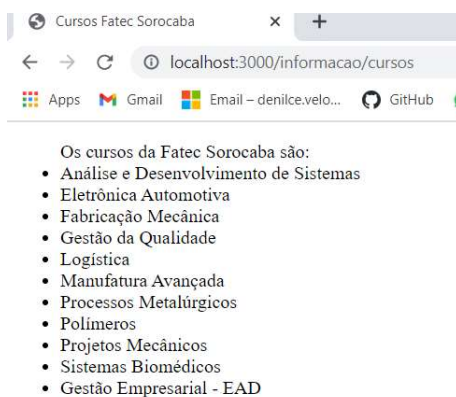
Fazer um teste se inicia o servidor.

Figura 44: Tela console iniciando servidor

```
cmd Prompt de Comando - nodemon app.js
D:\PWEBNode\Exercicios>nodemon app.js
[nodemon] 1.10.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
servidor iniciado
```

Se chamar <http://localhost:3000/> e as outras páginas estiverem funcionando normalmente, poderá testar a página iniciar e a de cursos por exemplo.

Figura 45: Tela console iniciando servidor



Separando as configurações das rotas

Criar uma pasta routes dentro de app. Na pasta routes criar 5 arquivos js, para as rotas que estão em app.js

Figura 46: Pasta routes



Por padrão o EJS procura os arquivos das rotas na pasta raiz, como os arquivos mudaram de pasta, configurar a nova pasta no config/server e incluir os módulos dentro da aplicação. Antes precisa deixar no formato EJS, colocando o exports. Nesse caso precisa exportar como função, porque precisa executar os comandos, não é possível simplesmente atribuir algo. Também precisa definir a variável app como parâmetro porque ele não reconhece a variável app que está dentro de app.js.

Arquivo adicionar_usuario.js

```
module.exports = function(app) {  
  app.get('/admin/adicionar_usuario', function(req,res){  
    res.render("admin/adicionar_usuario");  
  });  
}
```

```
});  
}
```

Arquivo cursos.js

```
module.exports = function(app) {  
  app.get('/informacao/cursos', function(req,res){  
    res.render('informacao/cursos');  
  });  
}
```

Arquivo historia.js

```
module.exports = function(app) {  
  app.get('/informacao/historia', function(req,res){  
    res.render('informacao/historia');  
  });  
}
```

Arquivo home.js

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

Arquivo professores.js

```
module.exports = function(app) {  
  app.get('/informacao/professores', function(req,res){  
    res.render('informacao/professores');  
  });  
}
```

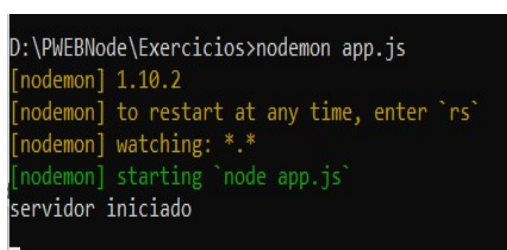
No arquivo app.js deve incluir esses módulos das rotas.

```
var app = require('./app/config/server');  
  
var rotaHome = require('./app/routes/home');  
rotaHome(app);  
  
var rotaAdicionarUsuario = require('./app/routes/adicionar_usuario');  
rotaAdicionarUsuario(app);  
  
var rotaHistoria = require('./app/routes/historia'); // só está definindo  
rotaHistoria(app); // está executando  
  
var rotaCursos = require('./app/routes/cursos'); // só está definindo  
rotaCursos(app); // está executando  
  
var rotaProfessores = require('./app/routes/professores'); // só está definindo  
rotaProfessores(app); // está executando
```

```
/* poderia executar assim também*/  
/*  
var rotaAdicionarUsuario = require('./app/routes/adicionar_usuario')(app);  
  
*/  
  
app.listen(3000, function(){  
    console.log("servidor iniciado");  
});
```

Iniciar o servidor novamente e testar as páginas.

Figura 47: Tela console iniciando servidor



```
D:\PWEBNode\Exercicios>nodemon app.js  
[nodemon] 1.10.2  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching: *.*  
[nodemon] starting `node app.js`  
servidor iniciado
```

Teste da página professores.

Figura 48: Tela página professores



Está é a lista do corpo docente da Fatec Sorocaba

10. Banco de Dados

Nesse tópico será construída uma aplicação em NodeJS que irá se conectar ao banco de dados Microsoft SQL Server e realizar algumas manipulações usando a linguagem DML (*Data Manipulation Language*) do SQL, ou seja, algumas operações de SELECT, INSERT, UPDATE e DELETE.

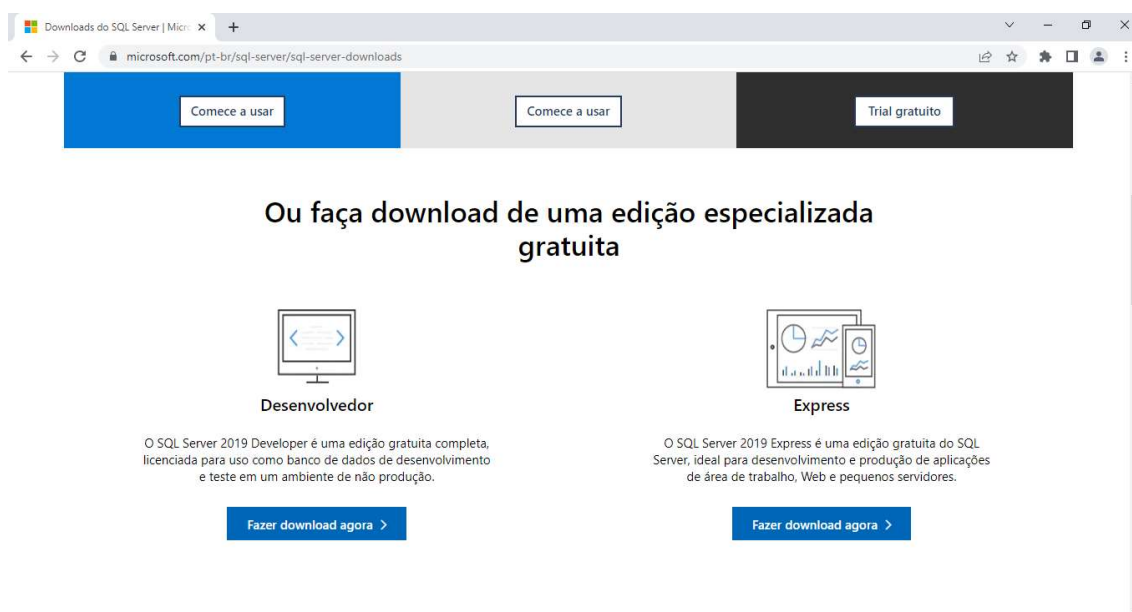
MySQL é um sistema gerenciador de banco de dados, utiliza a linguagem SQL (*Structure Query Language* – Linguagem de Consulta Estruturada), que é a

linguagem mais popular para inserir, acessar e gerenciar o conteúdo armazenado num banco de dados.²²

10.1 Instalação do SQL Server

Para instalar o SQL Server na sua máquina, fazer o download no site abaixo.

Figura 49: Site download do SQL Server



10.2 Instalação do driver mssql

Será necessário baixar um driver do SQL Server para o Node, que irá funcionar como um módulo para a aplicação de maneira mais fácil.

Comandos:

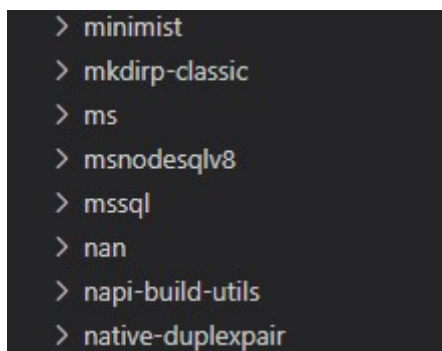
```
npm install mssql --save
```

```
npm install msnodesqlv8 --save
```

²² <https://www.microsoft.com/pt-br/sql-server/sql-server-2019>

Nos módulos deve aparecer o mssql.

Figura 50: Módulos do SQL Server



10.3 Usando o SQL Server no Prompt

Abrir o prompt de comando e digitar o comando:

```
sqlcmd -S NOME_DO_SERVIDOR -U LOGON -P SENHA
```

Após isso, será exibido um contador de linha mostrando que você está conectado ao banco de dados.

Figura 51: Testando o MySQL no console



Se você instalou a ferramenta Microsoft SQL Server Management Studio será mais fácil criar o banco de dados e as tabelas através dessa ferramenta.

Para criar um banco pelo prompt, digitar:

```
create database site_fatec  
go
```

Para acessar o banco criado digitar:

```
use site_fatec;  
go
```

Criar as seguintes tabelas professores e usuario:

```
CREATE TABLE professores  
(  
ID_PROFESSOR INT IDENTITY(1,1) PRIMARY KEY NOT NULL ,  
NOME_PROFESSOR VARCHAR(50) NOT NULL,  
EMAIL_PROFESSOR VARCHAR(100) NOT NULL,  
);  
  
CREATE TABLE usuario  
(  
ID_USUARIO INT IDENTITY(1,1) PRIMARY KEY NOT NULL ,  
NOME_USUARIO VARCHAR(50) NOT NULL,  
EMAIL_USUARIO VARCHAR(100) NOT NULL,  
SENHA_USUARIO VARCHAR(6) NOT NULL,  
);  
go
```

Criar alguns registros na tabela de professores.

```
insert into professores (NOME_PROFESSOR,EMAIL_PROFESSOR) values ('DENILCE','denilce@gmail.com');  
insert into professores values ( 'ANGELICA ','angelica@gmail.com');  
insert into professores values ( 'JEFFERSON','jefferson@gmail.com');  
insert into professores (nome_professor, email_professor) values ('CRISTIANE','cris@gmail.com');  
go
```

10.4 Listando dados de uma tabela na página

A ideia é listar os professores cadastrados na página dos professores.

Alterar o arquivo **professores.js** conforme código abaixo.

Primeiro fazer a conexão com o banco de dados, incluir o módulo do SQL Server.

Quando a página dos professores for chamada irá testar se banco de dados está conectando e listar os registros dos professores. Os parâmetros da “conexão”

devem obedecer a uma estrutura JSON. O `res.send` envia o resultado da consulta. Comentar a linha do `res.render`, pois o que deve ser enviado é a consulta.

```
module.exports = function(app){
  app.get('/informacao/professores', function(req,res){
    const sql = require ('mssql/msnodesqlv8');

    const sqlConfig = {
      user: 'LOGON',
      password: 'SENHA',
      database: 'site_fatec',
      server: 'NOME_DO_SERVIDOR',//Caso o nome tenha uma instância, colocar duas
barras, ex: 'DESKTOP\\SQLSERVER
    }

    async function getProfessores() {
      try {
        const pool = await sql.connect(sqlConfig);

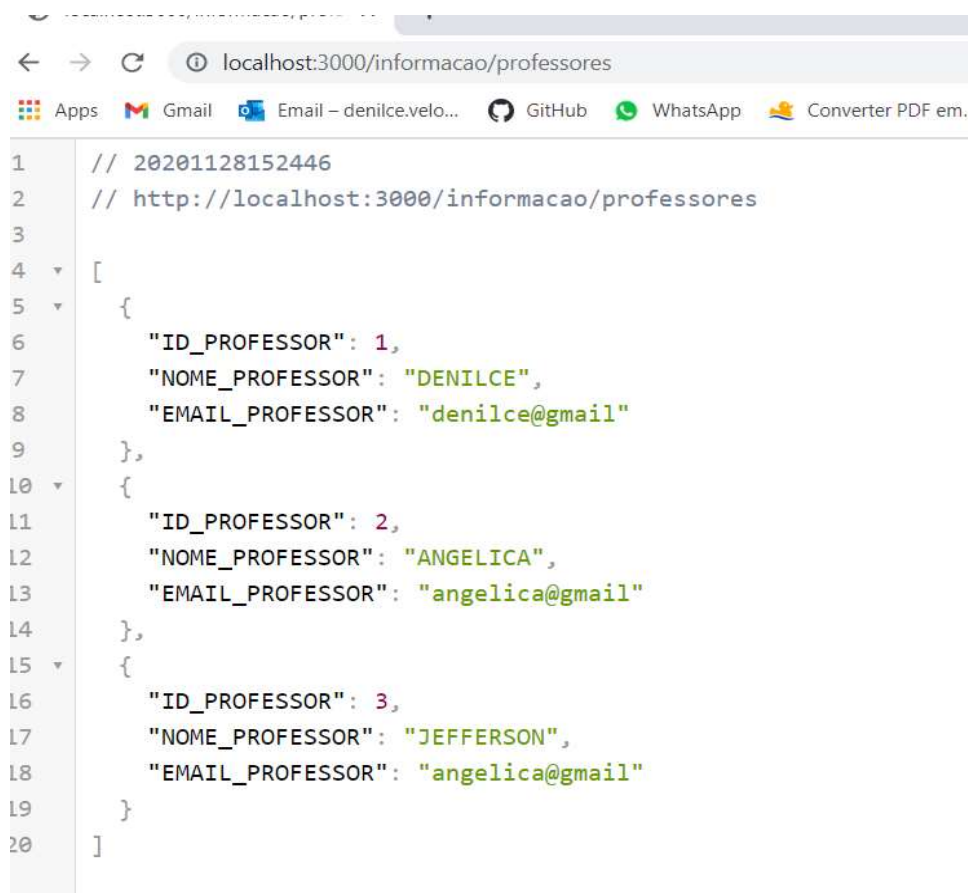
        const results = await pool.request().query('SELECT * from PROFESSORES')

        res.send(results.recordsets)

      } catch (err) {
        console.log(err)
      }
      //res.render('informacao/professores');
    }
    const professores = getProfessores();
  });
}
```

Tentar carregar a página dos professores
<http://localhost:3000/informacao/professores>

Figura 52: Testando a página dos professores (listar registros)



```
1 // 20201128152446
2 // http://localhost:3000/informacao/professores
3
4 [
5   {
6     "ID_PROFESSOR": 1,
7     "NOME_PROFESSOR": "DENILCE",
8     "EMAIL_PROFESSOR": "denilce@gmail"
9   },
10  {
11    "ID_PROFESSOR": 2,
12    "NOME_PROFESSOR": "ANGELICA",
13    "EMAIL_PROFESSOR": "angelica@gmail"
14  },
15  {
16    "ID_PROFESSOR": 3,
17    "NOME_PROFESSOR": "JEFFERSON",
18    "EMAIL_PROFESSOR": "angelica@gmail"
19  }
20 ]
```

Observar que ele retornou os dados no formato JSON.

10.5 Listando dados de uma tabela na página (através da view)

Alterar o arquivo professores.js para que esses dados retornados sejam passados para a View e a ela deve exibi-los. No código substituir o `res.send` por `res.render` e nele colocar também um rótulo “profs” por exemplo e o JSON.

O EJS permite recuperar as informações na view e lá dentro da view escrever também código JavaScript.

Dentro da view professores é possível recuperar as informações de “profs” como se fosse um array, sendo que cada abre e fecha chave representa um índice dentro do JSON.

```
module.exports = function(app){
  app.get('/informacao/professores', function(req,res){
```

```
const sql = require ('mysql/msnodesqlv8');

const sqlConfig = {
  user: 'LOGON',
  password: 'SENHA',
  database: 'site_fatec',
  server: 'NOME_DO_SERVIDOR',
}

async function getProfessores() {
  try {
    const pool = await sql.connect(sqlConfig);

    const results = await pool.request().query('SELECT * from PROFESSORES')

    res.render('informacao/professores',{profs : results.recordset});

  } catch (err) {
    console.log(err)
  }
}

const professores = getProfessores();
});
}
```

Alterar a view **professores.ejs** conforme código abaixo para a impressão. É possível misturar o código HTML com JavaScript, basta usar os códigos **<% e %>**. Tudo que estiver dentro será interpretado.

O sinal de = é necessário quando um valor será impresso.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title> Professores Fatec Sorocaba</title>
</head>

<style>
  table#tabela1 tr td {
    /* Toda a tabela com fundo creme */
    background: #ffc;
  }

  table#tabela1 tr.cabeca td {
    background: #eee;
    /* Linhas com fundo cinza */
  }
</style>

<body>
  <table border="1px" cellpadding="5px" cellspacing="0" ID="tabela1">
```

```
<tr class="cabeca">
  <td>Nome</td>
  <td>E-Mail</td>
</tr>

<% console.log('QUANTIDADE DE REGISTROS→' + profs.length); %>

<% for(var i = 0; i < profs.length; i++){ %>
<tr>
  <td><%= profs[i].NOME_PROFESSOR %></td>
  <td><%= profs[i].EMAIL_PROFESSOR %></td>
</tr>

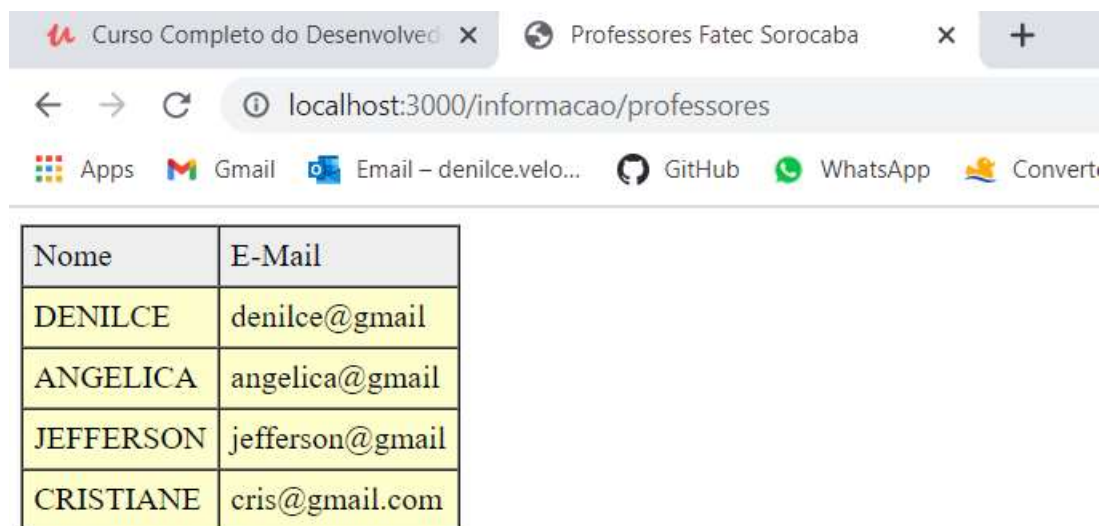
<% } %>

</table>
</body>

</html>
```

Testar a página dos professores, agora retornando os dados para a view.

Figura 53: Testando a página dos professores (usando view dinâmica)



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/informacao/professores'. Below the browser window, a table is displayed with two columns: 'Nome' and 'E-Mail'. The table contains four rows of data.

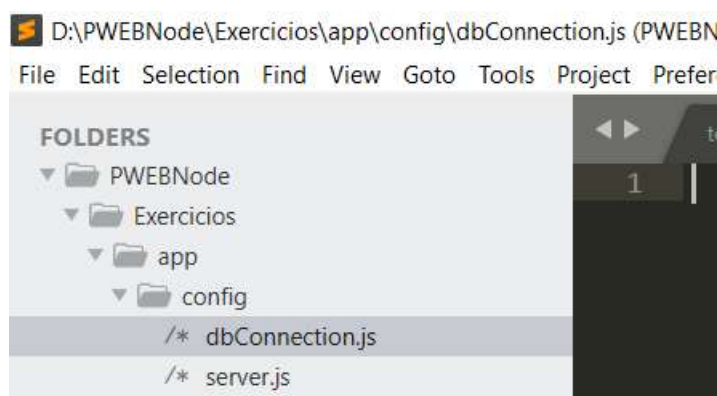
Nome	E-Mail
DENILCE	denilce@gmail
ANGELICA	angelica@gmail
JEFFERSON	jefferson@gmail
CRISTIANE	cris@gmail.com

10.6 Alterando a forma de conexão com o banco SQL Server

Esse exercício pretender modularizar a aplicação de forma a separar views do acesso de banco dados de forma a ficar mais parecido com o MVC (*Model View Control*), a conexão do banco deve ficar separada das rotas.

Criar um arquivo (dbConnection) dentro da pasta config.

Figura 54: Novo arquivo dbConnection para acesso ao banco



Esse arquivo deve exportar o módulo de conexão do banco de dados (exports) e retornar como uma função.

```
var sql = require('mssql/msnodesqlv8');

module.exports = function(){

  const config = {
    user: 'LOGON',
    password: 'SENHA',
    database: 'site_fatec', //your database
    server: 'NOME_DO_SERVIDOR',
    driver: 'msnodesqlv8',
  }
  return sql.connect(config);
}
```

Acertar no arquivo **professores.js** para incluir o módulo de conexão ao banco de dados e executar a conexão através da função.

```
const sql = require('mssql');
```

```
// para acessar o arquivo de config voltar 1 nivel
//D:\PWEBNode\Exercicios\app\config

var dbConnection = require('../config/dbConnection');

module.exports = function(app){
  app.get('/informacao/professores', function(req,res){
    async function getProfessores() {
      try {
        const pool = await dbConnection(); // executando a funcao

        const results = await pool.request().query('SELECT * from PROFESSORES');

        res.render('informacao/professores',{profs : results.recordset});

      } catch (err) {
        console.log(err)
      }
    }

    const professores = getProfessores();
  });
};
```

Testar novamente a página dos professores para ver se está ok.

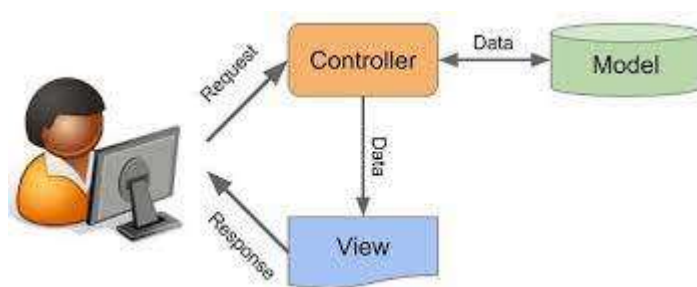
11. Separando a aplicação em Camadas

Vocês devem ter observado, que as separações ou reestruturações que estamos seguindo nos exercícios, apontam para o modelo MVC. A ideia é estruturar a aplicação de forma a aparecer mais com a estrutura MVC, que é um Design Pattern (modelo de projeto).

11.1 MVC

O MVC é uma sigla do termo em inglês Model (modelo), View (visão) e Controller (Controle) que facilita a troca de informações entre a interface do usuário aos dados no banco, fazendo com que as respostas sejam mais rápidas e dinâmicas.

Figura 55: Estrutura MVC



Fonte: <https://medium.com/trainingcenter/mvc-framework-usando-a-arquitetura-sem-c%C3%B3digo-de-terceiros-bf95a744c66d>

11.1.1 Model ou Modelo

Essa classe também é conhecida como *Business Object Model* (objeto modelo de negócio), e é de sua responsabilidade gerenciar e controlar a forma como os dados se comportam por meio das funções, lógica e regras de negócios estabelecidas.

11.1.2 Controller ou Controlador

A camada de controle é responsável por intermediar as requisições enviadas pelo View com as respostas fornecidas pelo Model, processando os dados que o usuário informou e repassando para outras camadas.

11.1.3 View ou Visão

Essa camada é responsável por apresentar as informações de forma visual ao usuário, recursos ligados a aparência como mensagens, botões ou telas. O View está na linha de frente da comunicação com usuário e é responsável transmitir questionamentos ao controller e entregar as respostas obtidas ao usuário.

11.2 Melhorando a organização das rotas

Para auxiliar na organização das rotas, serão utilizadas mais algumas bibliotecas.

11.2.1 Consign

Essa biblioteca é um autoloader de scripts, aqui é sugerida para facilitar o gerenciamento das rotas no Express. Observe que uma biblioteca dessas pode ser muito útil quando se tem muitas telas, pois, quanto mais telas, mais rotas. Sem Consign, as rotas precisariam ser carregadas no app.js uma a uma, usando require.

Com o Consign, no arquivo server.js pode se fazer um require do módulo Consign e através da função include do Consign, incluir todas as rotas que estão dentro de app/routes, ou seja, no momento de carregar a aplicação ele “recupera” todas as rotas. Ele pode fazer também autoloader de views e arquivos de configuração, controllers, etc.

Na documentação do Express, existe também o Router, que permite criar manipuladores de rota da aplicação (<https://expressjs.com/pt-br/guide/routing.html>).

DICA: Se você quiser saber como está o download de um item pelo npm, basta acessar <https://www.npmtrends.com>.

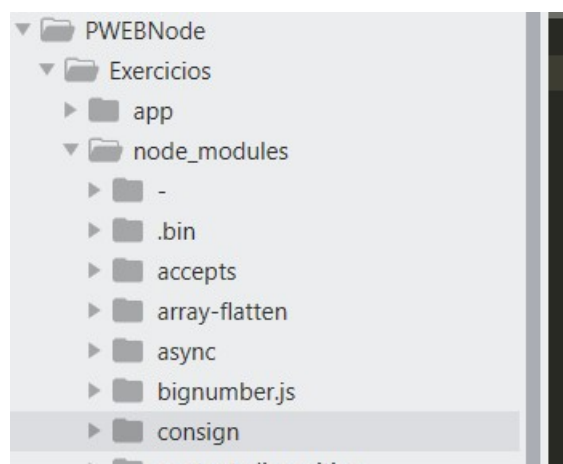
Instalação do Consign

Para instalar o Consign usar:

```
npm install consign@0.1.6 --save
```

Observe na pasta node_modules que ele está aparecendo.

Figura 56: node_modules com o Consign



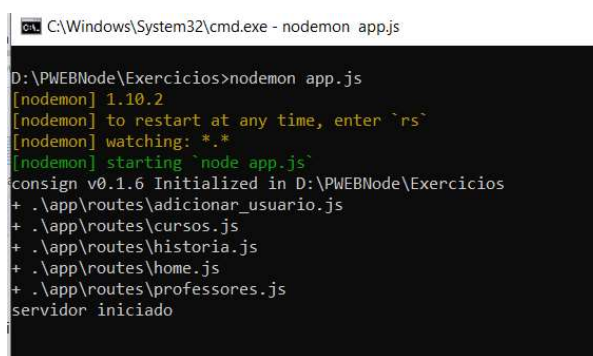
11.2.2 Inclusão do Consign no server.js

Alterar o arquivo **server.js** para incluir o Consign, no futuro se tivermos novas rotas, não será necessário incluir todas as rotas no app.js pois o Consign carregará automaticamente.

```
var express = require('express');  
  
var consign = require('consign');  
  
var app = express();  
  
app.set('view engine', 'ejs');  
  
app.set('views', './app/views');  
  
consign().include('app/routes').into(app); // incluindo a pasta routes como se fosse  
namespaces do c#, pega os módulos e inclui no servidor app  
  
module.exports = app;
```

Executar app.js → nodemon app.js

Figura 57: Consign incluindo a pasta routes



```
C:\Windows\System32\cmd.exe - nodemon app.js  
  
D:\PWEBNode\Exercicios>nodemon app.js  
[nodemon] 1.10.2  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching: *.*  
[nodemon] starting `node app.js`  
consign v0.1.6 Initialized in D:\PWEBNode\Exercicios  
+ .\app\routes\adicionar_usuario.js  
+ .\app\routes\cursos.js  
+ .\app\routes\historia.js  
+ .\app\routes\home.js  
+ .\app\routes\professores.js  
servidor iniciado
```

Alterar o arquivo **app.js**, agora não precisa colocar todas as rotas.

```
var app = require('./app/config/server');  
  
/*var rotaHome = require('./app/routes/home');  
//rotaHome(app);  
  
var rotaAdicionarUsuario = require('./app/routes/adicionar_usuario');  
rotaAdicionarUsuario(app);  
  
var rotaHistoria = require('./app/routes/historia'); // só esta definindo  
rotaHistoria(app); // está executando  
  
var rotaCursos = require('./app/routes/cursos'); // só esta definindo  
rotaCursos(app); // está executando  
  
var rotaProfessores = require('./app/routes/professores'); // só esta definindo
```

```
rotaProfessores(app); // está executando
*/
app.listen(3000, function(){
    console.log("servidor iniciado");
});
```

Executando o arquivo app.js

Figura 58: Consign incluindo a pasta routes mesmo sem estarem no app.js

```
D:\PWEBNode\Exercicios>nodemon app.js
[nodemon] 1.10.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
consign v0.1.6 Initialized in D:\PWEBNode\Exercicios
+ .\app\routes\adicionar_usuario.js
+ .\app\routes\cursos.js
+ .\app\routes\historia.js
+ .\app\routes\home.js
+ .\app\routes\professores.js
servidor iniciado
```

Testar a página principal

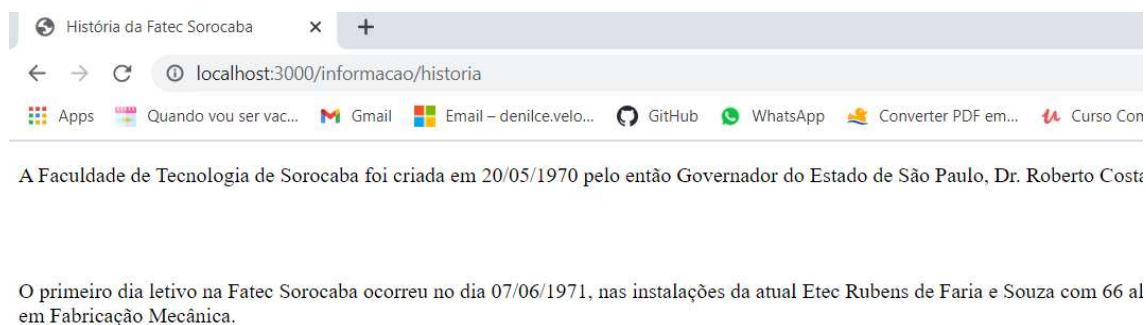
Figura 59: Página principal depois do Consign



Bem vindo a página inicial da Fatec Sorocaba

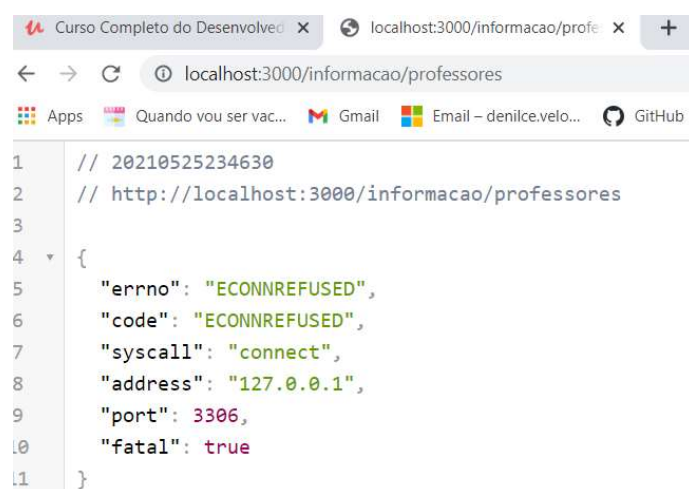
Testar a página história

Figura 60: Página história depois do Consign



Tentar abrir a página dos professores, caso o seu banco MySQL não estiver startado, vai apresentar erro.

Figura 61: Página professores depois do Consign (sem startar banco de dados)



11.2 Restruturação da parte do banco de dados

No projeto, por enquanto apenas uma página acessa o banco de dados, a dos professores (professores.ejs), mas supondo muitas páginas acessassem dados, em todas elas teria o carregamento da conexão no arquivo das rotas?

```
var dbConnection = require('../config/dbConnection');

module.exports = function(app){
  app.get('/informacao/professores', function(req,res){
    async function getProfessores() {
      try {
        const pool = await dbConnection(); // executando a funcao

        const results = await pool.request().query('SELECT * from PROFESSORES');

        res.render('informacao/professores',{profs : results.recordset});

      } catch (err) {
        console.log(err)
      }
    }

    const professores = getProfessores();
  });
};
```

O ideal é levar essa conexão para a parte do autoload (carregado inicialmente). Vamos acertar para ficar mais fácil a manutenção.

Primeiro acertar o arquivo **server.js** incluindo a pasta config também.

```
var express = require('express');
var consign = require('consign');

var app = express();

app.set('view engine', 'ejs');
app.set('views', './app/views');

// especificado qual arquivo ele deve executar porque dentro do config tem o server
// ele iria ficar executando o servidor toda hora
// precisa da extensao senao ele pensa que é um subdiretorio

consign({cwd:'app'}) // para incluir a pasta app
  .include('routes')
  .then('config/dbConnection.js')
  .into(app);

module.exports = app;
```

Depois comentar a linha da conexão no arquivo **professores.js**, e acertar de forma que será executada a conexão somente quando a rota for requisitada.

```
const sql = require ('mssql');

//var dbConnection = require('../config/dbConnection');

module.exports = function(app){
  app.get('/informacao/professores', function(req,res){
    async function getProfessores() {
      try {
        var connection = app.config.dbConnection;

        const pool = await connection(); // executando a funcao

        const results = await pool.request().query('SELECT * from PROFESSORES');

        res.render('informacao/professores',{profs : results.recordset});

      } catch (err) {
        console.log(err)
      }
    }

    const professores = getProfessores();
  });
};
```

Observe que no arquivo dbConnection.js anterior, estava sendo feita a conexão, o que faz que **com toda vez que carregar a aplicação ele fará a conexão**, isso não é ideal.

```
var sql = require ('mssql/msnodesqlv8');

// retornando a conexão realizada
module.exports = function(){

  const config = {
    user: 'LOGON',
    password: 'SENHA',
    database: 'site_fatec', //your database
    server: 'NOME_DO_SERVIDOR',
    driver: 'msnodesqlv8',
  }
  return sql.connect(config);
}

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

Para resolver esse problema será utilizado wrap (“embrulhar”), e esse export **não retornará mais um método mais uma variável**, alterar o arquivo **dbConnection.js**

```
var sql = require ('mssql/msnodesqlv8');

var connSQLServer = function(){
    console.log('Conexao com o banco de dados estabelecida!');

    const config = {
        user: 'LOGON',
        password: 'SENHA',
        database: 'site_fatec', //your database
        server: 'NOME_DO_SERVIDOR',
        driver: 'msnodesqlv8',
    }

    return sql.connect(config);
}

// exportando a função e quando chamar a página ele conecta
module.exports = function(){
    console.log('O autoloader carregou o módulo de conexão com o bd');
    return connSQLServer;
}
```

Recarregar o servidor com app.js

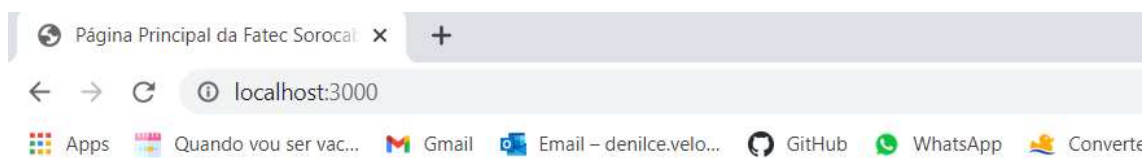
Figura 61: Recarregando servidor com autoload carregando routes

```
C:\Windows\System32\cmd.exe - nodemon app.js

D:\PWEBNode\Exercicios>
D:\PWEBNode\Exercicios>nodemon app.js
[nodemon] 1.10.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
consign v0.1.6 Initialized in app
+ .\routes\adicionar_usuario.js
+ .\routes\cursos.js
+ .\routes\historia.js
+ .\routes\home.js
+ .\routes\professores.js
+ .\config\dbConnection.js
O autoload carregou o módulo de conexão com o bd
servidor iniciado
```

Carregar a página principal.


Figura 62: Página principal com autoload carregando routes



Bem vindo a página inicial da Fatec Sorocaba

Carregar a página dos professores.

Figura 63: Página história com autoloader carregando routes



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/informacao/professores'. The page content is a table with two columns: 'Nome' and 'E-Mail'. The table contains seven rows of data, including names like DENILCE, ANGELICA, JEFFERSON, CRISTIANE, CESAR, Levi, and DENILCe, along with their corresponding email addresses.

Nome	E-Mail
DENILCE	denilce@gmail
ANGELICA	angelica@gmail
JEFFERSON	jefferson@gmail
CRISTIANE	cris@gmail.com
CESAR	cesar@gmail.com
Levi	levi@gmail.com
DENILCe	denilce@gmail.com

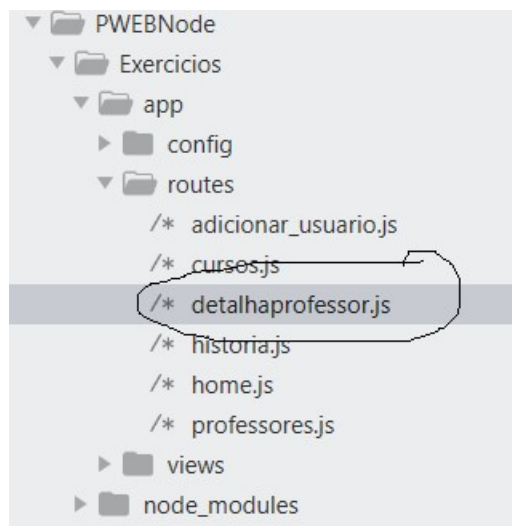
12. Criando uma página para recuperar professor pelo ID

Nesse passo, será criada uma página para mostrar um professor pelo ID.

12.1 Criação da rota

Primeiro criar uma rota **detalhaprofessor.js**, observar que não é necessário incluir essa rota no carregamento, pois ele fará isso no autoloader da aplicação.

Figura 64: Rota detalhaprofessor.js



```
const sql = require ('mssql');

module.exports = function(app){
  app.get('/informacao/professores/detalhaprofessor', function(req,res){

    async function getProfessoresID() {
      try {
        var connection = app.config.dbConnection;

        const pool = await connection();

        const results = await pool.request().query('SELECT * FROM professores WHERE
id_professor = 1')

        res.render('informacao/professores/detalhaprofessor',{profs : results.recordset});

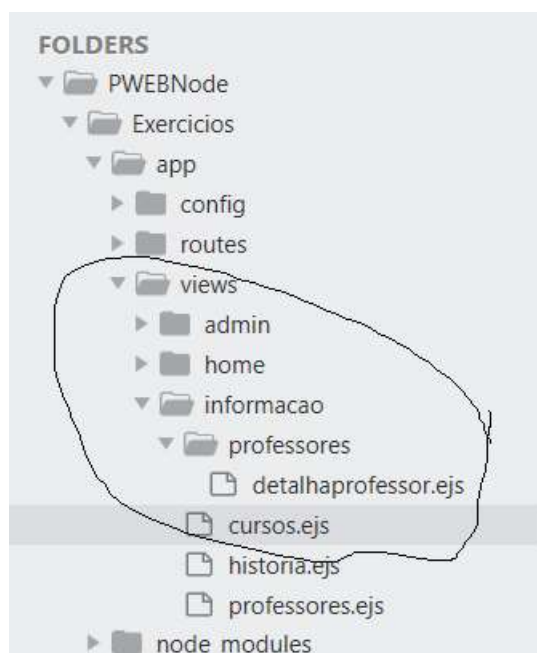
      } catch (err) {
        console.log(err)
      }
    }

    const professoresID = getProfessoresID();
  });
}
```

12.2 Criação da view detalhaprofessor.ejs

Criar a view detalhaprofessor.ejs dentro da pasta informacao\professores, observe que ele listará apenas um registro.

Figura 65: View detalhaprofessor.ejs



```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title> Detalhe Professor Fatec Sorocaba</title>
</head>

<body>
  <table border="1px" cellpadding="5px" cellspacing="0" ID="tabela1">

    <tr class="cabeca">
      <td>ID</td>
      <td>Nome</td>
      <td>E-Mail</td>
    </tr>
    <tr>
      <td><%= profs[0].ID_PROFESSOR %></td>
      <td><%= profs[0].NOME_PROFESSOR %></td>
      <td><%= profs[0].EMAIL_PROFESSOR %></td>
    </tr>

  </table>
</body>

</html>
```

Recarregar o servidor, observar que ele vai carregar a rota do detalhaprofessor.

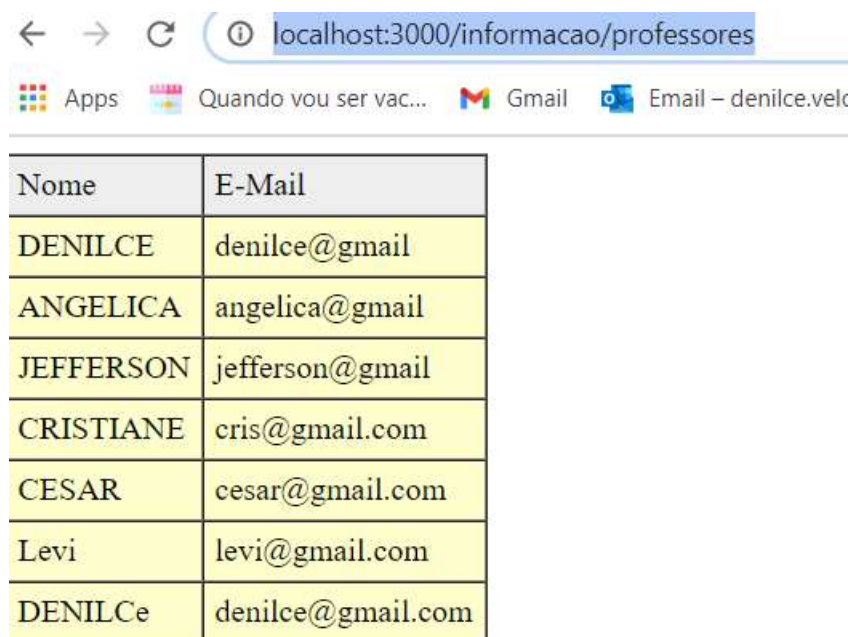
Figura 66: Servidor carregando rota detalhaprofessor.js

```
C:\Windows\System32\cmd.exe - nodemon app.js

D:\PWEBNode\Exercicios>nodemon app.js
[nodemon] 1.10.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
consign v0.1.6 Initialized in app
+ .\routes\adicionar_usuario.js
+ .\routes\cursos.js
+ .\routes\detalhaprofessor.js
+ .\routes\historia.js
+ .\routes\home.js
+ .\routes\professores.js
+ .\config\dbConnection.js
0 autoload carregou o módulo de conexão com o bd
servidor iniciado
```

Testar a página informacao/professores

Figura 67: Teste página professores



Nome	E-Mail
DENILCE	denilce@gmail
ANGELICA	angelica@gmail
JEFFERSON	jefferson@gmail
CRISTIANE	cris@gmail.com
CESAR	cesar@gmail.com
Levi	levi@gmail.com
DENILCe	denilce@gmail.com

Testar a página informacao/professores/detalheprofessor

Figura 68: Teste página detalheprofessor



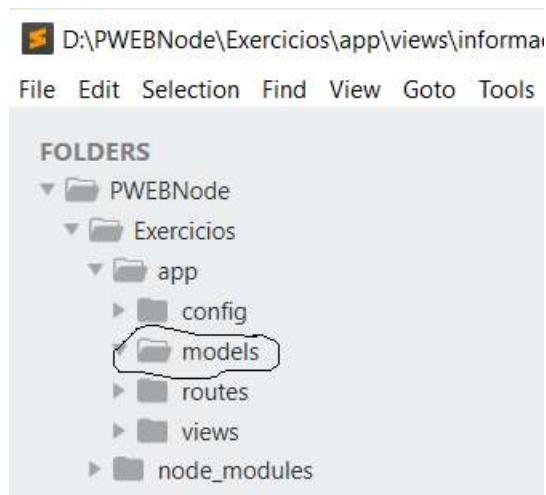
13. Implementando os Models

Continuando o objetivo de estruturar a aplicação como MVC, o próximo passo é implementar os Models, a parte relacionada ao banco de dados, é importante porque vamos separar a parte da lógica de dados da aplicação. As regras de negócio vão para dentro de Models.

Observe, por exemplo, que na rota de professores e detalha professor, está sempre conectando ao banco e listando os dados de acordo com o desejado, pode ser que os comandos de recuperação dos dados sejam bem parecidos, mas estão sendo repetidos, podemos melhorar essa parte.

Criar um diretório/pasta dentro de app.

Figura 69: Criação pasta models



Criar um arquivo **professorModel.js** dentro da pasta models, ele representar a entidade no banco de dados. Criar uma função dentro do contexto do módulo e usar o recurso do **this** para retornar os dados da função.

```
module.exports = function(){  
  
  this.getProfessores = function(connection, callback){  
    connection.query('select * from professores', callback);  
  }  
  
  this.getProfessor = function(connection, callback){  
    connection.query('select * from professores WHERE id_professor=1', callback);  
  }  
  
  return this;  
  
}
```

Alterar o arquivo **server.js** para incluir o Models no autoload.

```
var express = require('express');  
var consign = require('consign');  
  
var app = express();  
  
app.set('view engine', 'ejs');  
app.set('views', './app/views');
```

```
// especificado qual arquivo ele deve executar porque dentro do config tem o server
// ele iria ficar executando o server toda hora
// precisa da extensão, senao ele pensa que é um subdiretorio

consign({cwd:'app'}) // para incluir a pasta app
.include('routes')
.then('config/dbConnection.js')
.then('models')
.into(app);

module.exports = app;
```

Alterar a rota **professor.js** para carregar do Models

```
const sql = require ('mssql');

module.exports = function(app){
  app.get('/informacao/professores', function(req,res){
    async function getProf() {
      try {
        var connection = app.config.dbConnection;
        const pool = await connection();

        var professoresModel = app.models.professorModel;// variável que recupera a
        função exporta

        //executar a função
        // tem passar a conexao e o callback
        professoresModel.getProfessores(pool, function(error, results){
          res.render('informacao/professores', { profs : results.recordset });
        });
      } catch (err) {
        console.log(err)
      }
    }
    const professores = getProf();
  });
}
```

Alterar a rota **detalhaprofessor.js** para carregar do Models, para não ficar precisando fazer require de todos os módulos, porque em uma aplicação normalmente tem várias tabelas (model).

```
const sql = require ('mssql');

module.exports = function(app){
```

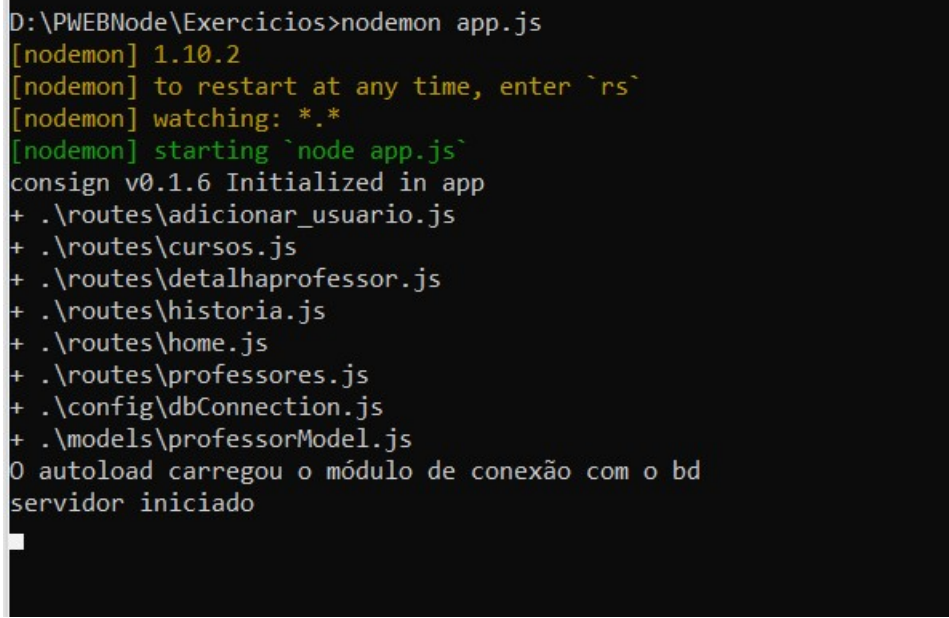
```
app.get('/informacao/professores/detalhaprofessor', function(req,res){
  async function getProfessoresID() {
    try {
      var connection = app.config.dbConnection;
      const pool = await connection();

      var professoresModel = app.models.professorModel;

      professoresModel.getProfessor(pool, function(error, results){
        res.render('informacao/professores/detalhaprofessor', { profs : results.recordset });
      });
    } catch (err) {
      console.log(err)
    }
  }
  const professoresID = getProfessoresID();
});
}
```

Carregar novamente o servidor para verificar se está carregando os models.

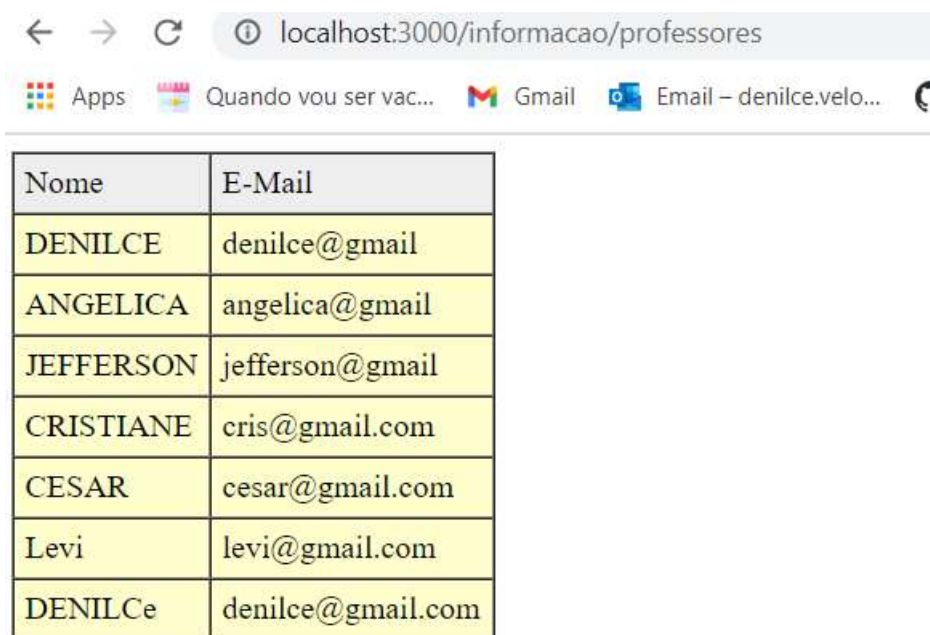
Figura 70: Servidor carregando a pasta models



```
D:\PWEBNode\Exercicios>nodemon app.js
[nodemon] 1.10.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
consign v0.1.6 Initialized in app
+ .\routes\adicionar_usuario.js
+ .\routes\cursos.js
+ .\routes\detalhaprofessor.js
+ .\routes\historia.js
+ .\routes\home.js
+ .\routes\professores.js
+ .\config\dbConnection.js
+ .\models\professorModel.js
O autoload carregou o módulo de conexão com o bd
servidor iniciado
```

Testar a página dos professores

Figura 71: Página dos professores com servidor carregando a pasta models



Nome	E-Mail
DENILCE	denilce@gmail
ANGELICA	angelica@gmail
JEFFERSON	jefferson@gmail
CRISTIANE	cris@gmail.com
CESAR	cesar@gmail.com
Levi	levi@gmail.com
DENILCe	denilce@gmail.com

Testar a página detalha professor

Figura 72: Página detalhaprofessor com servidor carregando a pasta models



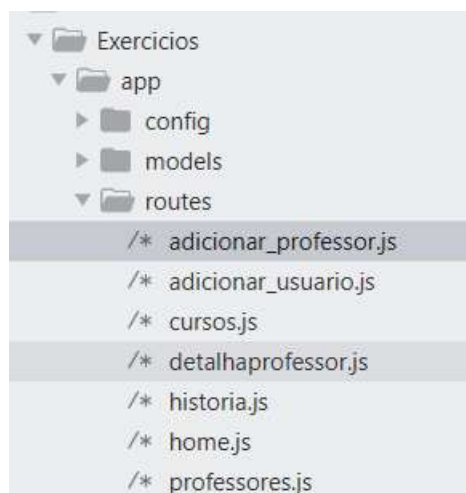
ID	Nome	E-Mail
1	DENILCE	denilce@gmail

14. Criação do formulário Inclusão do Professor

Aqui será criado um formulário para a inclusão de um professor.

Na pasta app\routes incluir arquivo (rota) **adicionar_professor.js**

Figura 73: Rota adicionar_professor.js



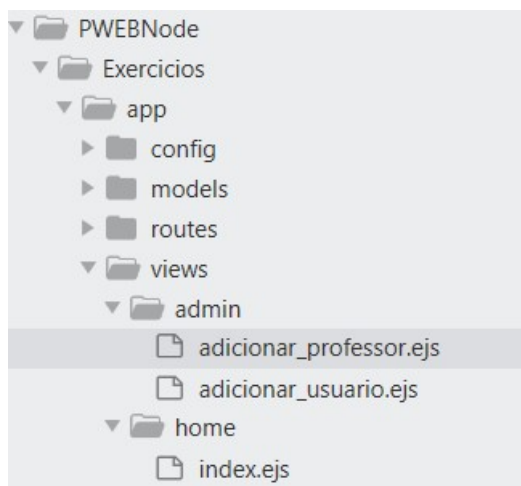
Alterar o **adicionar_professor.js**. Observe que foi colocado um post para quando o formulário for submetido.

```
module.exports = function(application){
  application.get('/admin/adicionar_professor', function(req,res){
    res.render('admin/adicionar_professor');
  });

  application.post('/professor/salvar', function(req,res){
    res.send("Salvo!!!!");
  });
}
```

Na pasta view\admin incluir arquivo (página) adicionar_professor.ejs

Figura 74: Arquivo (página) adicionar_professor.ejs



Alterar arquivo **adicionar_professor.ejs**

```
<!DOCTYPE html>
  <html lang="pt-br">
    <head>
      <meta charset="utf-8"/>
      <title>Cadastro de Professores</title>
    </head>
    <body>

      <h1>Adicionar Professor</h1>

      <form action="/professor/salvar" method="post">
        <label>Nome</label>
        <input type="text" id="nome" name="nome_professor"
placeholder="Nome do Professor" />
        <br/>
        <br/>
        <label>E-mail</label>
        <input type="email" id="email" name="email_professor" placeholder="E-mail do
Professor" />

        <br/>
        <br/>

        <input type="submit" value="Enviar" />
      </form>

    </body>
  </html>
```

Teste a página adicionar_professor e tente salvar.

Figura 75: Página adicionar_professor



← → ↻ ⓘ localhost:3000/admin/adicionar_professor

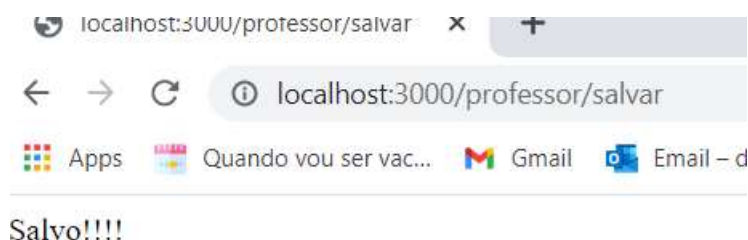
Apps Quando vou ser vac... Gmail Email – denilce.velo...

Adicionar Professor

Nome

E-mail

Figura 76: Retorno do Enviar (Salvar) professor



Observe que ainda não está incluindo no dado no banco de dados, para fazer a efetiva inclusão no banco de dados serão necessários mais alguns ajustes.

14.3 Body-Parser

É um módulo que extrai a parte do corpo inteiro de um fluxo de solicitação de entrada e o expõe em req.body. Ele analisa os dados codificados JSON, buffer, string e URL enviados usando a solicitação HTTP POST. É possível fazer isso sem utilizar esse módulo, mas usá-lo poupará trabalho.

O middleware era parte do Express.js anterior, mas agora você precisa instalá-lo separadamente.

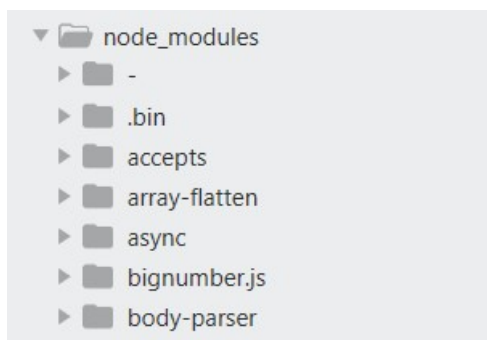
14.1.1 Instalação do Body-Parser

Para instalar o Body-Parser digitar o comando:

```
npm install body-parser@1.17.2 --save
```

Verifique se ele está aparecendo no node_modules.

Figura 77: Body-Parser no node_modules



14.1.2 Alteração do server e model para inserir no banco de dados

Alterar o arquivo **server.js**. Primeiro incluir o módulo do Body-Parser. Como o Body-Parser é um middleware (software que se encontra entre o sistema operacional e os aplicativos nele executados) e vai atuar nos objetos de requisição e resposta, então ele precisa ficar antes do carregamento do Consign.

```
var express = require('express');  
var consign = require('consign');  
var bodyParser = require('body-parser');  
  
var app = express();  
  
app.set('view engine', 'ejs');  
app.set('views', './app/views');  
  
// para ele entender o formato da URL
```

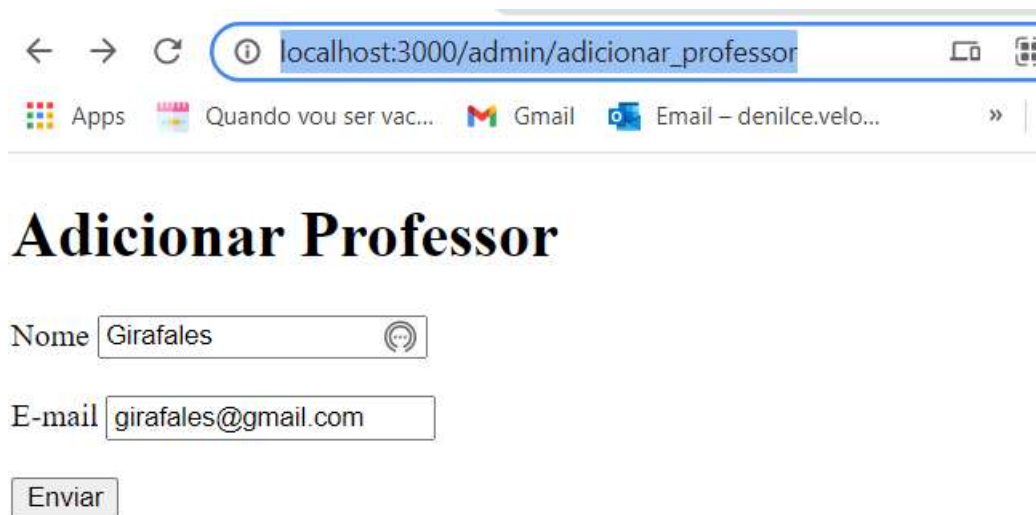
```
app.use(bodyParser.urlencoded({extended: true}));
```

```
consign({cwd:'app'}) // para incluir a pasta app
.include('routes')
.then('config/dbConnection.js')
.then('models')
.into(app);

module.exports = app;
```

Execute a página adicionar_professor e tentar salvar (enviar) os dados.

Figura 78: Carregando página adicionar_professor



← → ↻ ⓘ localhost:3000/admin/adicionar_professor 📱 ☰

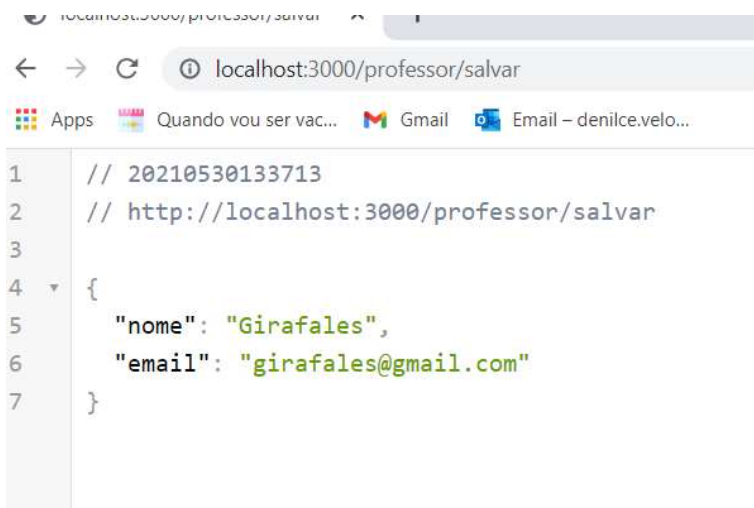
Apps Quando vou ser vac... Gmail Email – denilce.velo... »

Adicionar Professor

Nome

E-mail

Figura 79: Retorno do Salvar de adicionar_professor



```
1 // 20210530133713
2 // http://localhost:3000/professor/salvar
3
4 {
5   "nome": "Girafales",
6   "email": "girafales@gmail.com"
7 }
```

Embora ele esteja retornando o JSON dos dados, agora são precisos alguns ajustes para enviar ao banco. O próximo passo é alterar a rota `adicionar_professor.js` e usar “`req.body`”, requisição que está sendo recuperada e inserir um registro no banco de dados.

Alterar o arquivo `adicionar_professor.js`

```
module.exports = function(application){
  application.get('/admin/adicionar_professor', function(req,res){
    res.render('admin/adicionar_professor');
  });
  application.post('/professor/salvar', function(req,res){

    async function getAdcProfessor(){
      try {
        var professor = req.body;

        var connection = application.config.dbConnection;
        const pool = await connection();

        var professoresModel = application.models.professorModel;

        // //usando uma funcao de callback e informar quem devemos salvar, no caso
        professor

        professoresModel.salvarProfessor(professor,pool, (error, results)=>{
          // após inserir redireciona o navegador para outra página
          // se der erro na inclusao criar um erro 500 --> nao sabe o que significa

          if(error){
            console.log('Erro ao inserir no banco:' + error);
            res.status(500).send(error);
          } else {
            console.log('professor criado!!!');
            res.redirect('/informacao/professores');
          }
        });
      } catch (error) {
        console.log(error);
      }
    }

    getAdcProfessor();
  });
}
```

```
    }  
    });  
  } catch (error) {  
    console.log(error);  
  }  
}  
const adcProfessor = getAdcProfessor();  
});  
}
```

Implementar a função **salvarProfessor** no Model **professorModel.js**:

```
const sql = require ('mssql');  
  
module.exports = function(){  
  
  this.getProfessores = function(connection, callback){  
    connection.query('select * from professores', callback);  
  }  
  
  this.getProfessor = function(connection, callback){  
    connection.query('select * from professores WHERE id_professor=1', callback);  
  }  
  
  this.salvarProfessor = function(professor, connection, callback){  
    connection.query("INSERT INTO dbo.professores  
(NOME_PROFESSOR,EMAIL_PROFESSOR) VALUES ('"+ professor.nome_professor  
+"','"+ professor.email_professor+"')", callback);  
  }  
  
  return this;  
  
}
```


Recarregar o servidor

Figura 80: Servidor recarregado

```
C:\Windows\System32\cmd.exe - node app.js

D:\PWEBNode\Exercicios>node app.js
consign v0.1.6 Initialized in app
+ .\routes\adicionar_professor.js
+ .\routes\adicionar_usuario.js
+ .\routes\cursos.js
+ .\routes\detalhaprofessor.js
+ .\routes\historia.js
+ .\routes\home.js
+ .\routes\professores.js
+ .\config\dbConnection.js
+ .\models\professorModel.js
O autoload carregou o módulo de conexão com o bd
servidor iniciado
```

Testar a página adicionar_professor e salvar (enviar) os dados.

Figura 81: Página adicionar_professor



← → ↻ ⓘ localhost:3000/admin/adicionar_professor

Apps Quando vou ser vac... Gmail Email – denilce.vel

Adicionar Professor

Nome

E-mail

Deve aparecer um resultado assim, pois após gravar ele está chamando a página professores.



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/informacao/profe'. Below the browser window, a table is displayed with two columns: 'Nome' and 'E-Mail'. The table contains ten rows of data, including names like DENILCE, ANGELICA, JEFFERSON, CRISTIANE, CESAR, Levi, DENILCe, NOVO PROFESSOR, SEILA, and Pardal, each with a corresponding email address.

Nome	E-Mail
DENILCE	denilce@gmail
ANGELICA	angelica@gmail
JEFFERSON	jefferson@gmail
CRISTIANE	cris@gmail.com
CESAR	cesar@gmail.com
Levi	levi@gmail.com
DENILCe	denilce@gmail.com
NOVO PROFESSOR	NOVO@TESTE.COM
SEILA	SEILA@GMAIL.COM
Pardal	pardal@gmail.com

Desafio: Chegou até aqui? Parabéns, então agora tenta fazer o cadastro do usuário.

Passos: Criar o models para o usuário `usuarioModel.js`, a rota `adicionar_usuario.js` e a `view adicionar_usuario.ejs`. Lembre-se que as pastas models, routes já estão no autoload.

➔ Subir no GitHub tudo que fizemos hoje como Atividade16.

Referências

BodyParser. O que o body-parser faz com o express? Disponível em: <https://www.tienxame.com/pt/node.js/o-que-o-body-parser-faz-com-o-express/825575205/> Acesso: 30.Mai.2021.

CriarWeb. <http://www.criarweb.com/artigos/eventos-nodjs.html>. Acesso: 04.Jun.2020.

Moraes. William Bruno. Construindo Aplicações com NodeJS. Novatec Editora; 1ª edição (22 setembro 2015).

Mozilla developer: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introdução. Acesso em: 15.Fev.2020.

MVC. O que é padrão MVC? Entenda arquitetura de softwares! Disponível em: <https://www.lewagon.com/pt-BR/blog/o-que-e-padrao-mvc> . Acesso em: 25.Mai.2020.

Udemy Curso de Node. www.udemy.com. Acesso em: 07.Dez.2019.

UFRGS Cliente/Servidor:
http://www.penta.ufrgs.br/redes296/cliente_ser/tutorial.htm. Acesso em:
15.Fev.2020.