**O que é o Node.js?**

Neste capítulo, vimos os primeiros passos com o Node.js, sua instalação e primeiras formas de uso. Descreva com suas palavras o que é o Node.js.

**Resposta do instrutor**

O Node.js é uma plataforma, ou um ambiente de execução para códigos javascript escrita em cima da engine de Javascript do Chrome, a V8. Ele é todo baseado em eventos e suas operações são não-bloqueantes, o que lhe confere bastante eficiência e leveza.

**Instalação e primeiros testes com o Node.js.**

Baixe a versão estável mais recente do Node.js a partir do próprio site do projeto:<https://nodejs.org/en/download/>

Execute a instalação do arquivo de acordo com o seu sistema operacional.

Se preferir, você pode instalar via package manager a partir deste outro link:<https://nodejs.org/en/download/package-manager/>

Agora vá no terminal e execute um comando do Node para se certificar que a instalação ocorreu com sucesso. Pode ser o comando que verifica a versão, por exemplo:

node -v

Se tudo ocorrer bem, deverá aparecer no terminal o número da versão. Algo como isso:

v4.2.1

Agora crie um primeiro arquivo para executar um código javascript no ambiente do Node. Crie um arquivo chamado produtos.js e dentro dele escreva um código simples para imprimir algo no console.

console.log('Arquivo de produtos no Node.js');

Agora volte ao terminal, navegue até o diretório onde você criou o arquivo e o execute com o comando do node:

node produtos.js

Por último, crie um arquivo que funcionará como o seu servidor web para exibir produtos. Chame esse arquivo de produtosWeb.js e dentro dele escreva o código para subir um servidor HTTP, enviar um HTML no Response e imprimir uma linha de código no console.

Esse é com você! =D

Acesse agora a url no ip e porta que você utilizou para configurar o server e veja se o html é exibido na tela e a mensagem do console no terminal.

Deu tudo certo com a instalação? Foi possível acessar a URL normalmente através do navegador? A mensagem do console foi exibida corretamente no terminal?

**Resposta do instrutor**

O código para escrever o primeiro server deve ficar algo assim:

var http = require('http');  
var porta = 3000;  
var ip = "localhost";  
  
var server = http.createServer(function(req, res) {  
 console.log("Recebendo request");  
 res.writeHead( 200, {'Content-Type': 'text/html'});  
 res.end('<html><body>Request recebido!</body></html>');  
});  
  
server.listen(porta, ip);  
  
console.log("Server running at http://" + ip + ":" + porta + "/");

Utilizamos a função require() para carregar a lib http, que já faz parte do pacote de módulos default do Node.

Declaramos variáveis para armazenar os valores da porta e do ip onde queremos disponibilizar o nosso server e invocamos a função createServer(), que criar o servidor.

Como parâmetro da createServer(), passamos uma função de callback, onde escrevemos o código que será executado no retorno da função, seguindo a forma de programação assíncrona imposta pelo Node.

Dentro dessa função escrevemos algo no console, invocamos a função writeHead() que escreve algum cabeçalho http para o response da requisição e por fim, mandamos o html na response através da chamada da função res.end().

Após essa chamada, que serviu para criar o server, foi invocada a função listen() passando o ip e a porta, que é a função que de fato deixa o server criado disponível para receber requisições.

Utilizando essas configurações, a url a ser acessada é a [http://localhost:3000](http://localhost:3000/).

**O que é o npm e para que serve?**

Quando precisamos adicionar bibliotecas javascript externas ao Node, utilizamos o npm. Mas afinal o que é o npm e para que ele serve exatamente?

**Resposta do instrutor**

O npm é o gerenciador de pacotes do ecosistema Node. Ele tem a função de gerenciar os projetos e pacotes javascript externos que precisemos utilizar em cada aplicação. Quando o Node é instalado, o npm também já vem por default e com certeza será muito útil em qualquer projeto desenvolvido sobre o Node, visto que é muito comum que se precise de libs externas para facilitar o desenvolvimento de certas funcionalidades. O próprio npm possui diversas bibliotecas que podem ser integradas com o Node. O npm consegue, portanto ajudar na instalação de pacotes, na execução de algumas tarefas dentro do ambiente do Node e como repositório de bibliotecas.

**Gerenciando o projeto com npm e aplicando o Express.js.**

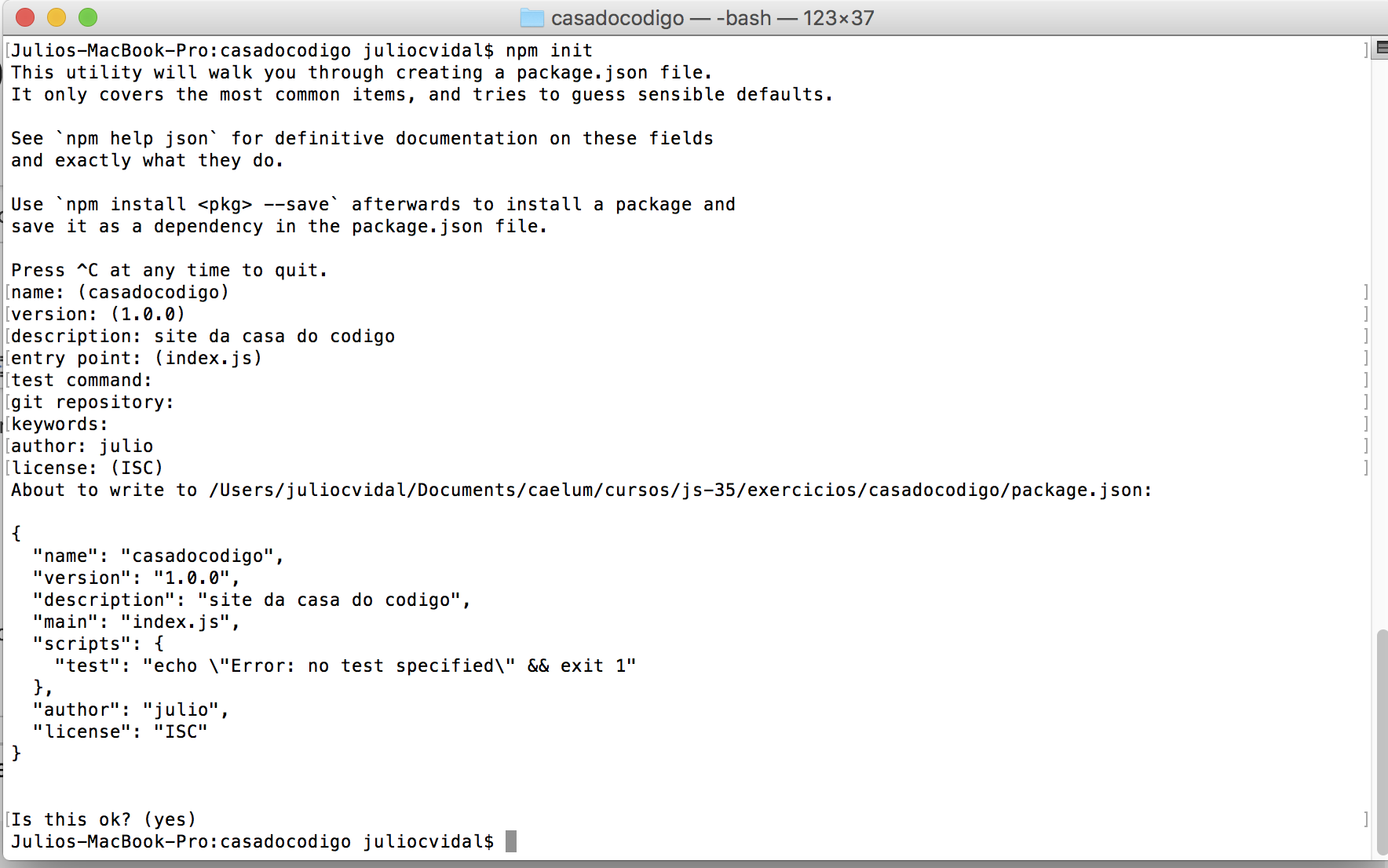
Vá até o terminal e verifique que você tem o npm corretamente instalado rodando o comando que imprime sua versão:

npm -version

Agora que você verificou que está tudo ok, crie um diretório chamado casadocodigo e navegue até ele. Dentro dele execute o comando do npm que inicializa um projeto:

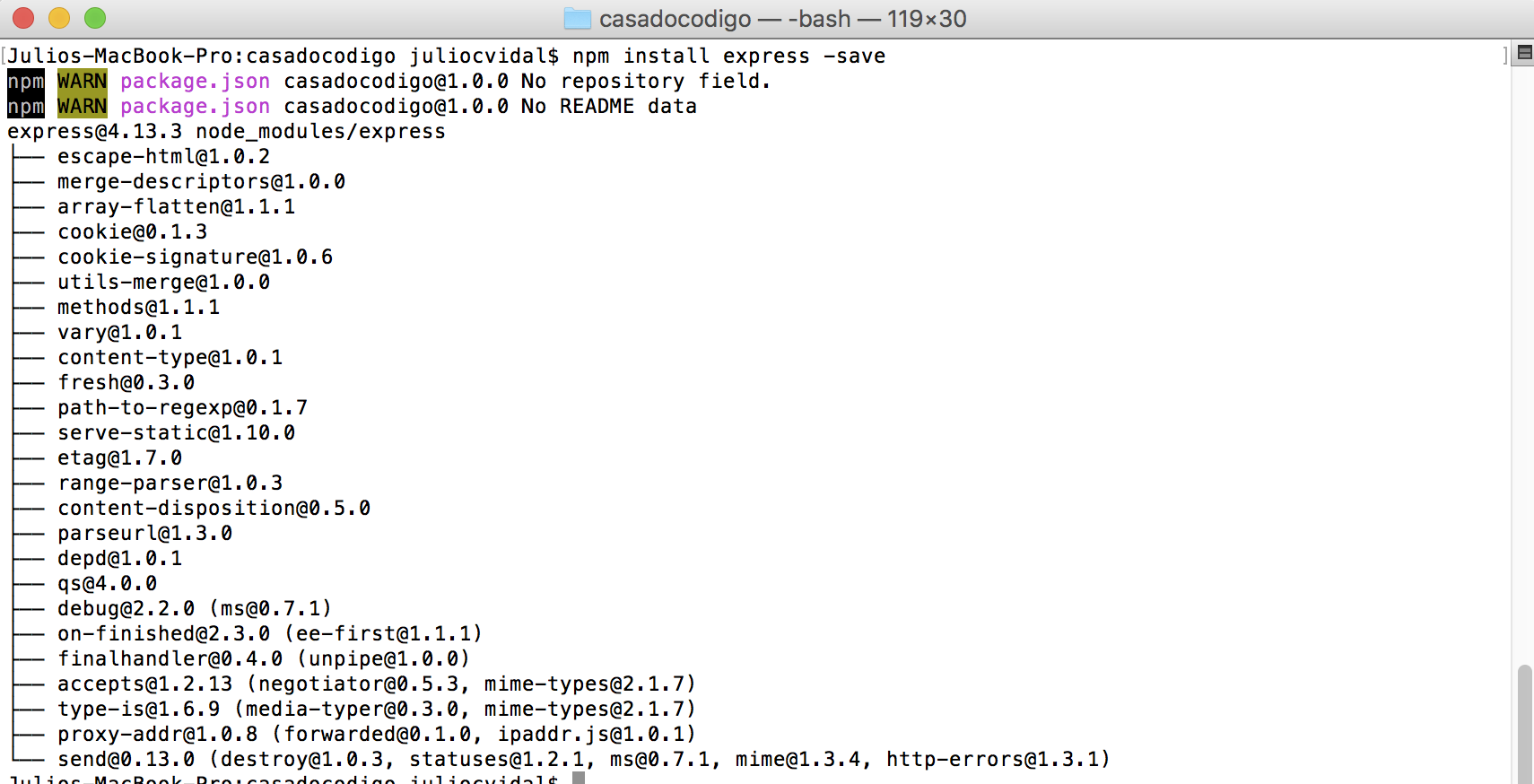
npm init

Não informe nada no nome, utilizando assim o próprio nome da pasta, informe uma descrição e coloque seu nome como autor. Todos os outros itens, você pode ir passando direto ou personalizar as informações da maneira que achar interessante.

A tela fica mais ou menos assim: 

Agora que você já tem seu projeto criado com o npm, utilize-o também para instalar os pacotes necessários para utilizar o express.js:

npm install express --save

Quando a instalação finalizar, o terminal, deve exibir algo assim: 

Agora que você já tem o Express configurado, crie um novo arquivo chamado app.js dentro do diretório casadocodigo. Esse novo arquivo agora fará a função de servidor web, porém utilizando as funcionalidades do Express. Nesse mesmo arquivo, implemente a configuração necessária para que o seu novo server consiga atender requisições quando for solicitado um get na url /produtos.

Para subir este novo servidor, vá até o diretório onde ele foi criado e execute o mesmo comando do node que já vimos anteriormente, só que agora para este novo arquivo:

node app.js

Para verificar que está tudo funcionando, acesse a url /produtos no navegador a partir do ip e porta que você configurou para o seu servidor e verifique o seu html exibido.

Funcionou a criação do projeto e a instalação do Express via npm? Após instalar o Express surgiram novos arquivos e pastas no projeto. O que são esses arquivos? O package.json agora tem uma nova informação. Qual é ela? Para que serve e o que fez com que ela fosse parar lá?

**Resposta do instrutor**

O arquivo app.js deve ficar parecido com isso:

var express = require('express');  
var app = express();  
  
app.get('/produtos', function(req, res){  
 res.send("<html><body>Listagem de Produtos </body></html>");  
});  
  
app.listen(3000, function(){  
 console.log("Servidor rodando");  
});

Se você utilizou os mesmo parâmetros que o código acima, a url a ser requisitada é a seguinte:http://localhost:3000/produtos

Agora que o projeto foi criado utilizando o comando init, o próprio npm se encarrega de gerar toda a estrutura de pastas necessária para que o projeto fique configurado de acordo com as convenções do node e pronto para receber integrações de diversas outras libs javascript.

O arquivo package.json funciona como um descritor do projeto. Ele guarda meta-informações referentes à documentação e também informações de configuração, como as dependências que o projeto possui para que possa ser executado.

A primeira dependência que inserimos no projeto foi da lib express.js, que é uma lib que tem a função de abstrair todo o funcionamento do módulo http do Node, facilitando assim o desenvolvimento de uma aplicação web nesta plataforma.

**Instalação e uso do EJS.**

Instale o EJS utilizando também o npm:

npm install ejs --save

Caso prefira, você pode baixar o EJS direto do site do projeto e seguir o passo-a-passo para a instalação:<http://www.embeddedjs.com/>

No seu arquivo app.js, mude o corpo da função que trata a requisição de produtos, para que ela agora passe retornar um res.render() com o nome do arquivo a ser exibido. Ao invés do res.send() com o html direto no parâmetro, como acontecia antes.

Para que o Express reconheça a engine ejs, você precisa adicioná-la como uma view engine da sua app. Faça isso no arquivo app.js através do método set() da variável que contem o objeto Express carregado.

O EJS por default busca os arquivos a serem exibidos dentro de um diretório chamado 'views', portanto crie esse diretório na raíz do seu projeto e dentro dele crie um outro diretório chamado 'produtos', com um arquivo 'lista.ejs' dentro dele. para que assim fique compatível com a url que foi utilizada no render().

Por fim crie o arquivo 'lista.ejs' com o HTML a ser exibido.

Salve todos os arquivos editados, navegue até o diretório onde está o app.js e rode-o utilizando o comando já conhecido do node:

node app.js

Acesse agora a url no navegador e veja a tabela sendo exibida conforme você escreveu no arquivo 'lista.ejs'.

Funcionou a exibição da tabela? O que você achou de utilizar o EJS para separar o HTML do código javascript? Compartilhe seu código aqui ao final da resolução.

**Resposta do instrutor**

O arquivo final do app.js deve ficar algo parecido com o seguinte:

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

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

app.get('/produtos', function(req, res){ res.render('produtos/lista'); });

app.listen(3000, function(){ console.log("Servidor rodando"); });

```

O caminho completo para a view da lista de produtos fica assim views/produtos/lista.ejs

E o código desse arquivo deve ficar algo parecido com o seguinte:

<html> <body> <table> <tr> <td>Id</td> <td>Nome</td> </tr> <tr> <td>1</td> <td>Node.js e Express</td> </tr> </table> </body> </html>

**Instalando e utilizando o nodemon.**

Para conseguir ter uma espécie de deploy automático no projeto, instale o **nodemom** utilizando também o npm.

sudo npm install -g nodemon

Agora para subir a aplicação, utilize o comando do nodemon:

nodemon app

Para verificar o reload automático, acesse a url da lista de produtos e deixe-a aberta no navegador.

Altere o arquivo lista.ejs adicionando uma borda na tabela:

<table border="1px">

Atualize a página no navegador e veja a tabela ser exibida agora com borda sem que você tenha precisado fazer o reload manualmente do app.js

Teste fazer uma alteração no arquivo app.js, como criar uma nova função para tratar um novo request, salve e veja o que acontece no terminal, agora que você está rodando a app com o nodemon.

O que você achou de utilizar o nodemon? E afinal o que é o nodemon? Explique com suas palavras o que você entendeu dele. Dessa vez ao utilizar o npm para instalar o nodemon foi passado um parâmetro -g. O que esse parâmetro faz? Por que motivo utilizamos essa abordagem?

**Resposta do instrutor**

O **nodemon** é uma ferramenta que fica observando o código do projeto e já faz o reload automático sempre que detecta alguma alteração. Assim você não precisa ficar reiniciando a aplicação a cada nova mudança no código.

Quando ele é instalado, é mais interessante que seja feita uma instalação global para que assim ele fique disponível para todos os projetos que você criar utilizando Node. Por esse motivo, você precisa ter permissão de super usuário na máquina para que ele consiga escrever diretamente na pasta de instalação do node. Esse é o motivo de passarmos também o sudo no início do comando.