

Fundamentos



"..é um interpretador **JavaScript** do lado do servidor que altera a noção de como um servidor deveria funcionar. Seu objetivo é possibilitar que um programador crie aplicativos altamente escalável se escreva código que manipule dezenas de milhares de conexões simultâneas.."

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

"..é uma plataforma para desenvolvimento de aplicações server-side baseadas em rede utilizando **JavaScript** e o V8 Engine, ou seja, com **Node.js** podemos criar uma variedade de aplicações Web utilizando apenas código em **JavaScript**.."

Todos os exemplos e respostas dos exercícios estão disponíveis no github (último slide)

- Não Bloqueante (Non-Blocking-Thread)
- Javascript Engine V8 - altamente escalável e de baixo nível, pois você vai programar diretamente com diversos protocolos de rede e internet ou utilizar bibliotecas que acessam recursos do sistema operacional, principalmente recursos de sistemas baseado em Unix
- Single-thread - cada aplicação terá instância de um único processo.
- Orientado a eventos - a mesma lógica de orientação de eventos do Javascript client-sided

Downloads

Latest LTS Version: **v6.11.4** (includes npm 3.10.10)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS

Recommended For Most Users

Current

Latest Features



Windows Installer

node-v6.11.4-x86.msi



Macintosh Installer

node-v6.11.4.pkg



Source Code

node-v6.11.4.tar.gz

- Windows Installer (.msi)
- Windows Binary (.zip)
- macOS Installer (.pkg)
- macOS Binaries (.tar.gz)
- Linux Binaries (x86/x64)
- Linux Binaries (ARM)
- Source Code

32-bit		64-bit	
32-bit		64-bit	
64-bit			
64-bit			
32-bit		64-bit	
ARMv6	ARMv7		ARMv8
node-v6.11.4.tar.gz			

NODE

```
# node -v
```

```
v6.9.4
```

NPM

```
# npm -v
```

```
3.10.10
```

1. Olá: *ola.js*
2. Não Bloqueante: *testeNaoBloqueante.js*
3. Parse arquivo texto: *my-parse.js*

Assim como o *Gems* do *Ruby* ou o *Maven* do *Java*, o Node.js também possui o seu próprio gerenciador de pacotes, ele se chama NPM. Ele se tornou tão popular pela comunidade, que foi a partir da versão 0.6.0 do Node.js que ele se integrou no instalador do Node.js, tornando-se o gerenciador *default*. Isto simplificou a vida dos desenvolvedores na época, pois fez com que diversos projetos se convergissem para esta plataforma. Segue a lista dos principais comandos:

- `npm install nome_do_módulo`: instala um módulo no projeto.
- `npm install nome_do_módulo --save`: instala o módulo no projeto, atualizando o `package.json` na lista de dependências.
- `npm list`: lista todos os módulos do projeto.
- `npm remove nome_do_módulo`: desinstala um módulo do projeto.
- `npm update nome_do_módulo`: atualiza a versão do módulo.
- `npm -v`: exibe a versão atual do npm.

adicionado o parâmetro `-g` as configurações se tornam globais.

Exemplo de uso:

npm install nodemon -g (instala o nodemon)

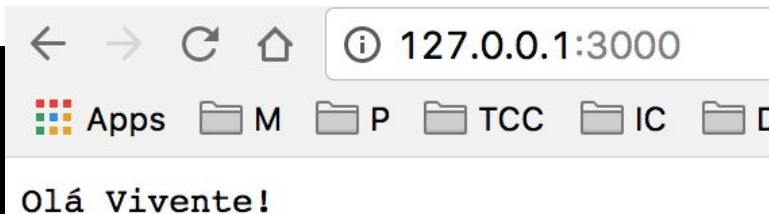
nodemon -v (verifica a versão do nodemon)

O exemplo a seguir sobe um serviço http, simples, na porta 3000, com uma response.

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

http.createServer(function(req,res) {
  res.writeHead(200, { 'Content-Type': 'text/plain; charset=utf-8' });
  res.end('Olá Vivente!');
}).listen(3000);

console.log('Servidor iniciado em localhost:3000. Ctrl+C para encerrar...');
```



"Ola Vivente"

- 1) Crie um resposta, utilizando o Node.js, conforme o exemplo.
- 2) A resposta tem que ser na porta 3002 e localhost.
- 3) imprima na resposta seu nome completo, data de nascimento e e-mail.



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

http.createServer(function(req,res) {
  res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
  res.write('<p>Nome: Cassio Trindade!</p>');
  res.write('<p>Data de Nascimento: 07/maio/1970</p>');
  res.end('<p>E-mail: casssiowt@gmail.com</p>');
}).listen(3002);

console.log('Servidor iniciado em localhost:3000. Ctrl+C para encerrar...');
```

O http é um módulo padrão do node que implementa um servidor http muito simples.

a mini-aplicação roda na porta 3002, em localhost.

o método `createServer` cria e inicializa o servidor http.

o método `.write()` escreve no response.

o método `.end()` finaliza o response.

é um framework que torna a criação de sites normais muito simples. A primeira coisa que você tem que fazer é instalá-lo. Use o node package manager - NPM

```
$ npm install express
```

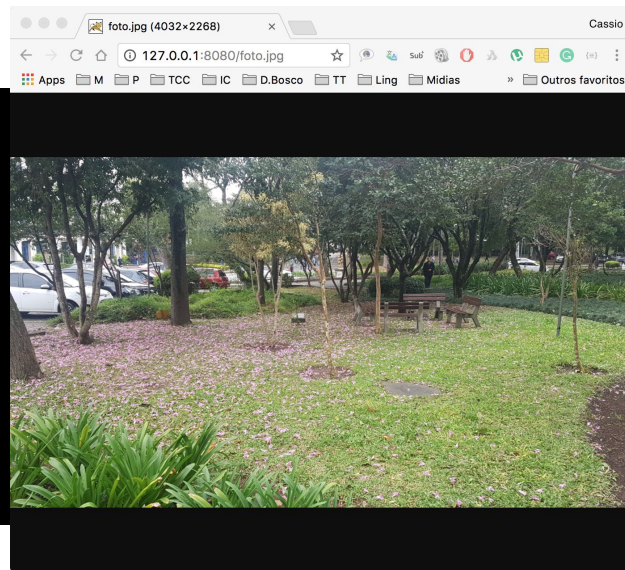
```
var express = require('express');
```

```
app = express();
```

```
app.use(express.static(__dirname + '/public'));
```

```
console.log(__dirname);
```

```
app.listen(8080);
```



Model-View-Controller (ou MVC) é provavelmente uma das arquiteturas mais populares para aplicativos. Tal como acontece com muitas outras coisas interessantes na história do computador, o modelo MVC foi concebido na linguagem Smalltalk como uma solução para o problema de organizar aplicativos com interfaces gráficas de usuário. Foi criado para aplicativos de desktop, mas desde então, a idéia foi adaptada a outros meios, incluindo a web.

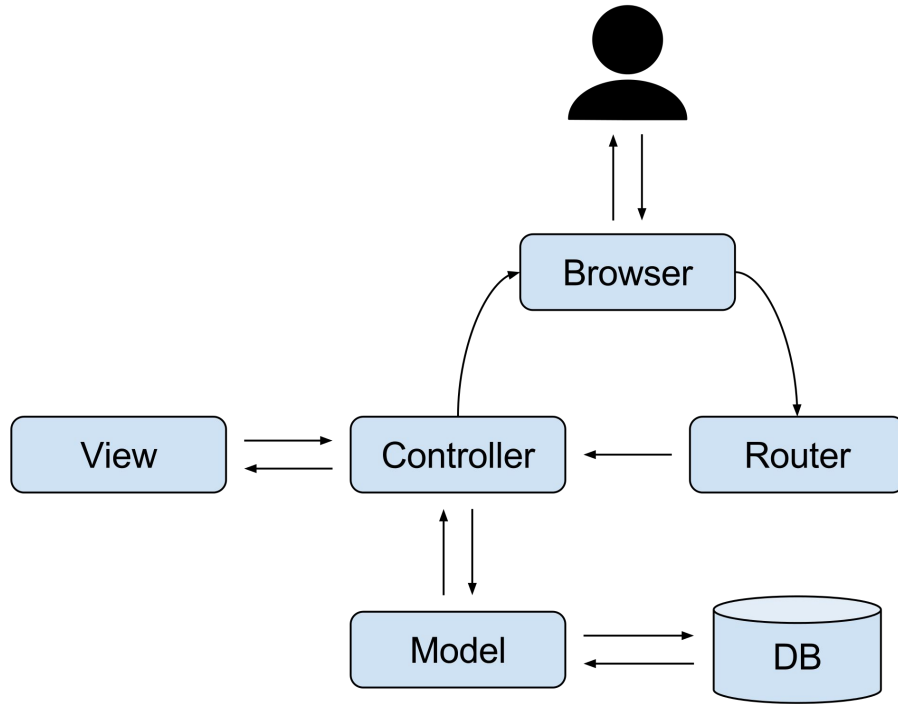
Podemos descrever a arquitetura MVC em palavras simples:

Modelo: A parte do nosso aplicativo que tratará o banco de dados ou qualquer funcionalidade relacionada a dados.

View: Tudo o que o usuário verá. Basicamente, as páginas que vamos enviar ao cliente.

Controller: a lógica do nosso site e a cola entre modelos e visualizações. Aqui chamamos nossos modelos para obter os dados, então colocamos esses dados em nossos pontos de vista para serem enviados aos usuários.

Nossas aplicações permitem utilizar muitas funcionalidades, e quando temos uma arquitetura sólida definida, facilita a criação destas funcionalidades e suas expansões e manutenções.



1. O *request* sai do **Browser** e entra no **Router**
2. O **Router** encaminha ao **Controller**
3. O **Controller** faz uma solicitação ao **Model**
4. **Model** retorna um *reponse* ao **Controller** (se necessário)
5. O **Controller** (se necessário) executa algumas operações usando dados do **Model**.
6. O **Controller** envia um *response* à **View**
7. A **View** processa o *response* (renderizando) e retorna ao **Controller**
8. O **Controller** encaminha ao **Browser**

- **Servidor** - para ouvir e responder às solicitações HTTP
- **Router** - para enviar a entrada solicita ao controlador correto
- **Controller** - para executar operações e interrogar os dados
- **Model** - para fornecer dados
- **View** - para fornecer a renderização HTML que vamos ver no navegador

MÓDULOS

Node tem um sistema simples de carregamento de módulos, a utilização de módulos permite incluir outros arquivos JavaScript em sua aplicação.

```
export.funcao =
```

Módulos são cruciais para construção de aplicações em Node pois eles permitem incluir bibliotecas externas, como bibliotecas de acesso ao banco de dados, e ajudam a organizar seu código em partes separadas com responsabilidades limitadas. Você deve tentar identificar partes reusáveis do seu código e transformá-las em módulos separados para reduzir a quantidade de código por arquivo e para ficar mais fácil de ler e manter seu código.

Utilizar módulos em Node é simples, você usa a função `require()`, que recebe um argumento: o nome da biblioteca do core ou o caminho do arquivo do módulo que você quer carregar.

```
var PI = Math.PI;

exports.area = function (r) {
  return PI * r * r;
};

exports.circunferencia = function (r) {
  return 2 * PI * r;
};
```

```
var circulo = require('./circulo.js');

console.log( 'Um circulo de raio 4 tem area de '
  + circulo.area(4));
```

Crie um módulo para realizar os cálculos matemáticos de multiplicar, dividir, somar e subtrair.

```
$ node app.js
```

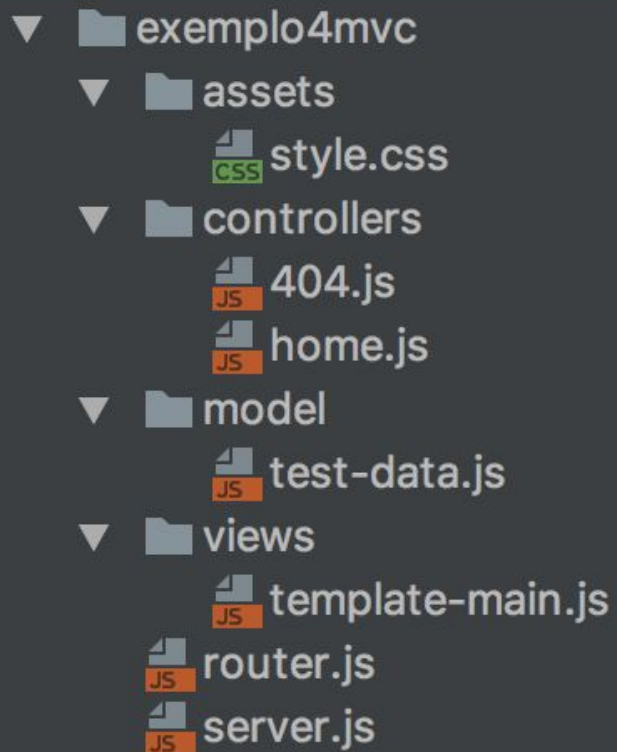
A Soma de $12 + 10 = 22$

A Subtração de $12 - 10 = 2$

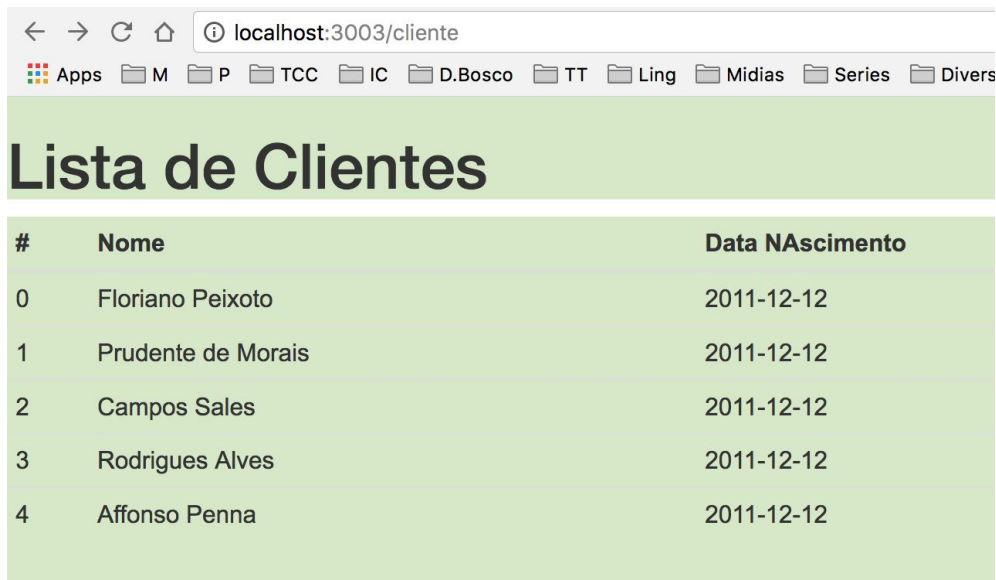
A Multiplicação de $12 * 10 = 120$

A Divisão de $12 / 10 = 1.2$

Resposta: exercicio2.zip



No app de exemplo, adicione mais uma rota para listar os clientes em uma tabela, usando arquivo test-cliente.js como model.



A screenshot of a web browser displaying a list of clients. The browser's address bar shows 'localhost:3003/cliente'. The page has a light green header with the title 'Lista de Clientes'. Below the header is a table with three columns: '#', 'Nome', and 'Data NAscimento'. The table contains five rows of client data.

#	Nome	Data NAscimento
0	Floriano Peixoto	2011-12-12
1	Prudente de Moraes	2011-12-12
2	Campos Sales	2011-12-12
3	Rodrigues Alves	2011-12-12
4	Afonso Penna	2011-12-12

```
var lista = function() {  
  var objJson = {  
    "financeiro": "Clientes VIPs",  
    "count": 5,  
    "clientes": [{  
      "nome": "Floriano Peixoto",  
      "sexo": "Masculino",  
      "dataNascimento": "2011-12-12"  
    }, {  
      "nome": "Prudente de Moraes",  
      "sexo": "Masculino",  
      "dataNascimento": "2011-12-12"  
    }, {  
      "nome": "Campos Sales",  
      "sexo": "Masculino",  
      "dataNascimento": "2011-12-12"  
    }, {  
      "nome": "Rodrigues Alves",  
      "sexo": "Masculino",  
      "dataNascimento": "2011-12-12"  
    }, {  
      "nome": "Afonso Penna",  
      "sexo": "Masculino",  
      "dataNascimento": "2011-12-12"  
    }  
  ]  
};  
return objJson;  
}
```

```
exports.listaClientes = lista();
```

PACKAGE.JSON

\$ npm init

Press ^C at any time to quit.

name: (exemplo4mvc) mvc

version: (1.0.0)

description: Teste MVC

entry point: (router.js)

test command:

git repository:

keywords:

author: Cassio

license: (ISC)

About to write to /Users/cassio.trindade/Google Drive/TargetTrust/TT_Node/arquivos/exemplo4mvcTeste/package.json:

```
{
  "name": "mvc",
  "version": "1.0.0",
  "description": "Teste MVC",
  "main": "router.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "nodemon server.js"
  },
  "author": "Cassio",
  "license": "ISC"
}
```

Executando **npm init** será criado um manifesto JSON válido chamado *package.json* para você e se você executá-lo em um repositório *git* existente, será definido um campo **repositories** dentro do *package.json* automaticamente.

```
$ mkdir my_module
$ cd my_module
$ git init
$ git remote add origin https://github.com/cassiowt/modulo\_mvc.git
$ npm init
$ ...
$ npm install --save
```

```
{
  "name": "mvc",
  "version": "1.0.0",
  "description": "Exemplo MVC no Node.js",
  "main": "router.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/cassiowt/modulo_mvc.git"
  },
  "author": "Cassio Trindade",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/cassiowt/modulo_mvc/issues"
  },
  "homepage": "https://github.com/cassiowt/modulo_mvc#readme",
  "dependencies": {
    "express": "^4.16.1"
  }
}
```

Estrutura web rápida, não organizada e minimalista para o Node.js

.."Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications."..

<https://expressjs.com/en/4x/api.html>

Verbos REST (*REST é uma maneira simples de organizar interações em sistemas independentes.*)
HTTP (*é um protocolo que permite enviar e receber informações na web*)

GET, POST, PUT e DELETE

Exemplo de rotas com 'HTTP'

Instalar o Postman *

```
http.createServer(function(req, res){
  let url = req.url
  , method = req.method

  switch(url){
    case "/api/cawt/read":
      if(method === "GET"){
        res.writeHead(200, { 'Content-Type': 'application/json; charset=utf-8' });
        res.write(JSON.stringify(SUCCESS));
      }else{
        res.writeHead(405, { 'Content-Type': 'application/json; charset=utf-8' });
        res.write(JSON.stringify(ERROR));
      }
      break;
    default:
      res.writeHead(404, { 'Content-Type': 'application/json; charset=utf-8' });
      ERROR.message = "Not Found";
      ERROR.code = 404
      res.write(JSON.stringify(ERROR));
      break;
  }

  res.end();
})
```

Crie um código, em node.js, para implementar os verbos REST, que entregue um JSON de sucesso, ou erro, nas outras 4 rotas padrões (POST, GET, PUT, DELETE).

Os endpoints deverão ser:

1. *POST: \api\target\cadastra*
2. *GET: \api\target\lista*
3. *PUT: \api\target\altera*
4. *DELETE: \api\target\exclui*

json

testar com o Postman*

```
SUCCESS = {  
  version: 1.0  
  , name: 'TargetTrust'  
  , created_at: date  
}  
  
, ERROR = {  
  message: '404 Recurso não Encontrado'  
}
```


Exemplo de rotas com 'EXPRESS'

```
var express = require('express');
var bodyParser = require('body-parser');

var port = 3005
var ser = '127.0.0.1'

app = express();
app.use(bodyParser());
app.use(express.static(__dirname + '/public'));

app.get('/', function(req, res) {
  res.send('Ola Vivente');
});
.....
```

server.js e index.html

```
.....
app.post('/', function (req, res) {
  res.send('POST request para E-MAil');
});

app.post('/list', function (req, res) {
  console.log(req.body);
  var pEmail = req.body.email;
  res.end('GET request para list: ' + pEmail);
});

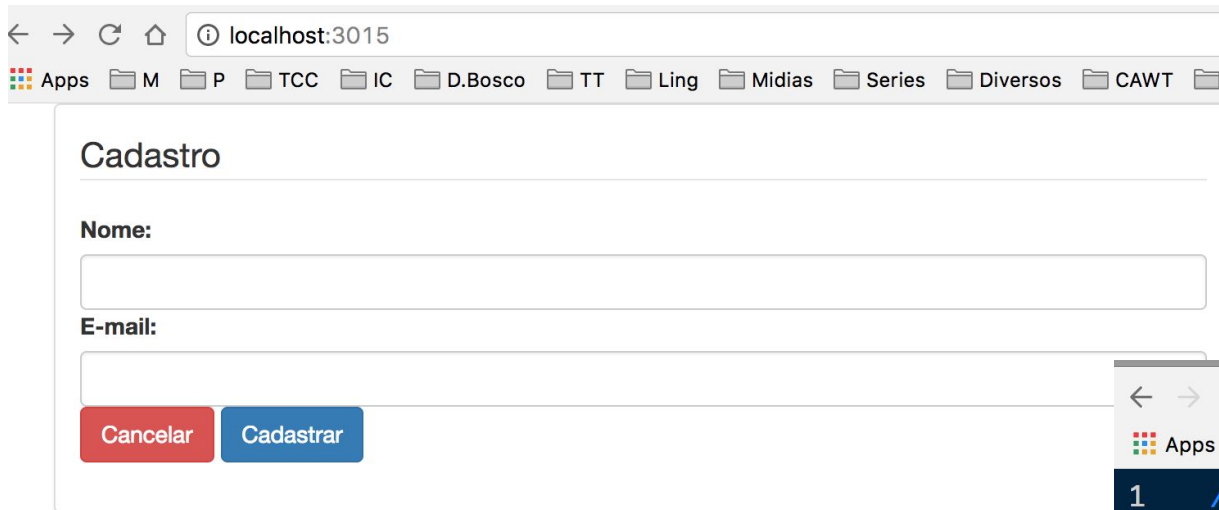
app.get('/ab*cd', function(req, res) {
  res.send('ab*cd');
});

console.log(" Executando em " + ser + ":" + port);
app.listen(port);
```

Criar as mesmas rotas feitas com 'HTTP' com o 'EXPRESS'

1. *POST: \api\target\cadastra*
2. *GET: \api\target\lista*
3. *PUT: \api\target\altera*
4. *DELETE: \api\target\exclui*

Criar um formulário index.html para usar rotas do 'EXPRESS'



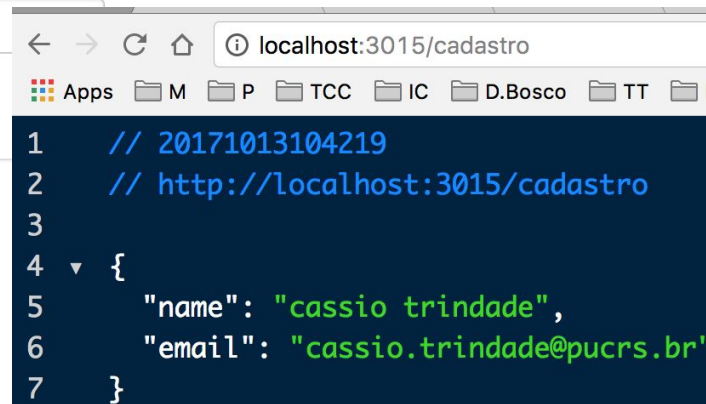
Cadastro

Nome:

E-mail:

Cancelar Cadastrar

Após clicar em cadastrar apresentar o json com os dados no browser.



```
1 // 20171013104219
2 // http://localhost:3015/cadastro
3
4 {
5   "name": "cassio trindade",
6   "email": "cassio.trindade@pucrs.br"
7 }
```

EJS - (Embedded JavaScript templates)















O módulo EJS possui diversas funcionalidades que permitem programar conteúdo dinâmico em código html. Os recursos servem para renderizar conteúdo dinâmico e minimizar repetições de código. Isolamos os conteúdos em outras views, conhecidas como *partials* (*template*), possíveis códigos que serão reutilizados com maior frequência. Dentro do diretório views, criamos as páginas da aplicação.

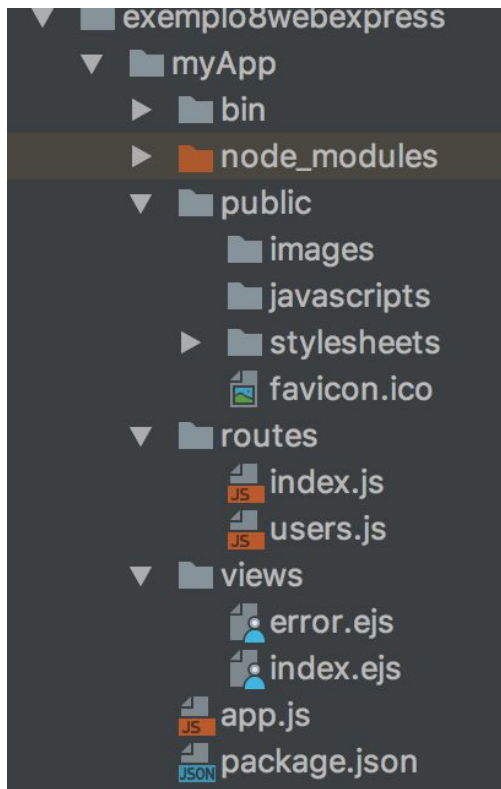
Exemplo de renderização com arquivos .ejs





- Inclua um novo item no menu "Lista Clientes";
- Adicione o arquivo cliente.js ao projeto;
- Adicione uma nova rota "/lista";
- Crie uma nova página "lista.ejs", que será renderizada pela nova rota.

Listagem Cliente VIP			
Nome	Sexo	Data Nascimento	Ações
Floriano Peixoto	Masculino	2011-12-12	  
Prudente de Moraes	Masculino	2011-12-12	  
Campos Sales	Masculino	2011-12-12	  
Rodrigues Alves	Masculino	2011-12-12	  
Afonso Penna	Masculino	2011-12-12	  



Vamos gerar uma aplicação genérica via 'EXPRESS'

```
$ npm install express -g
```

```
$ express --view=ejs web
```

Para ver os exemplos, clone o repositório do Express e instale as dependências

```
$ git clone git://github.com/expressjs/express.git --depth 1
```

```
$ cd express
```

```
$ npm install
```

Escolha o exemplo, e na sequência, execute o:

(<https://github.com/expressjs/express/tree/master/examples>)

```
$ node examples/mvc
```

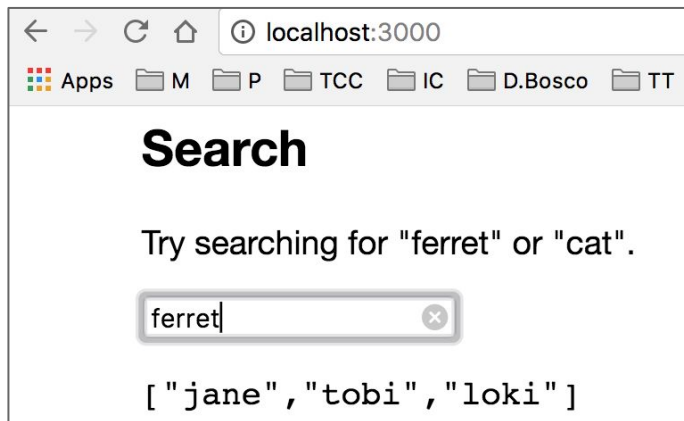
```
$ node examples/error-pages
```

Instale e baixe o express a partir do git em uma pasta chamada "exercicio08".

Vamos executar o exemplo "search".

Para isso há a necessidade de instalarmos o **Redis***, <https://redis.io/download>.

Verifique o arquivo index.js para as dicas de inicialização do **Redis**, e do módulo.

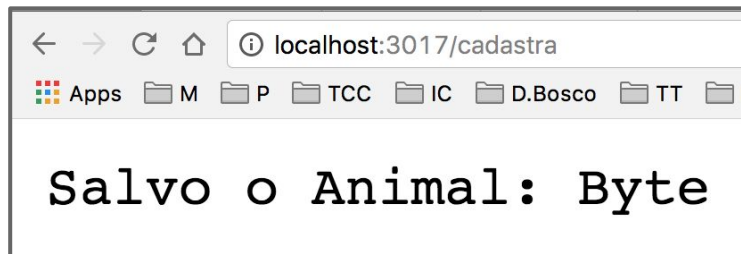
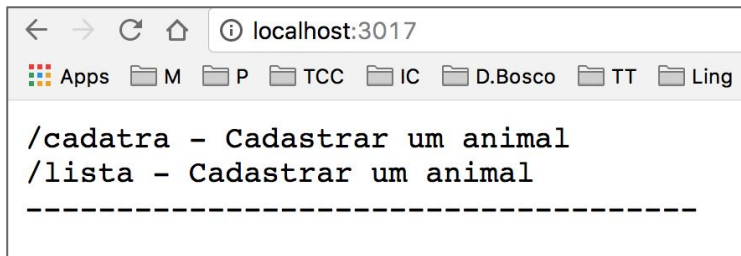


*Redis é um Banco de Dados NÃO Relacional, que suporta várias estruturas de dados. É muito rápido, trabalha com o conceito, chave:valor.

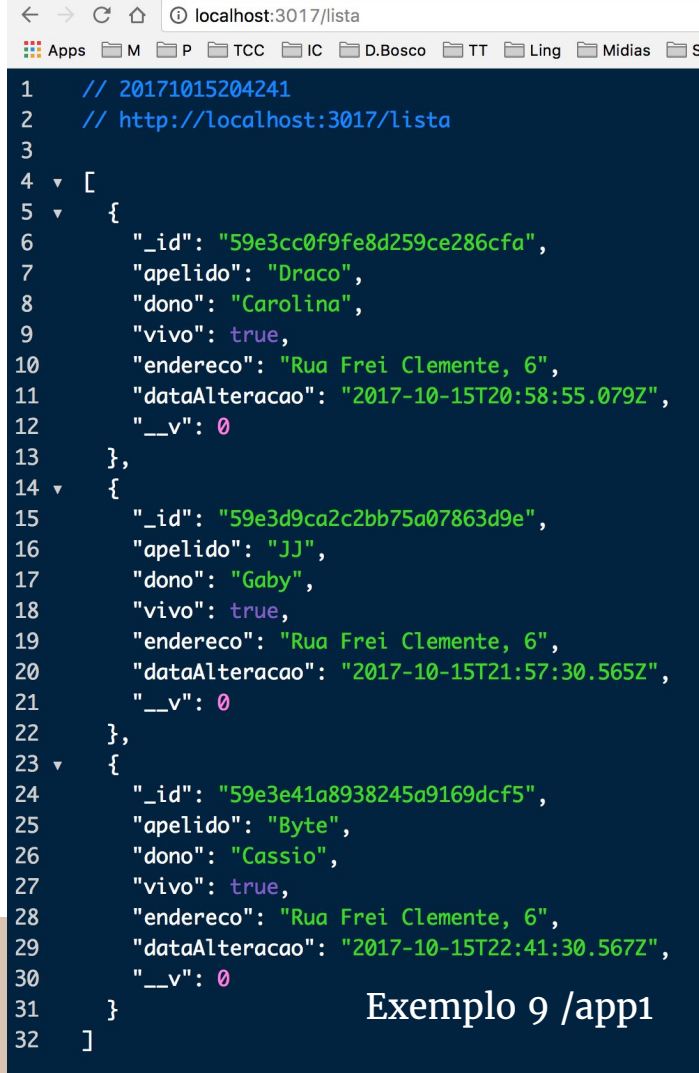
NODE com MONGODB

O MongoDB é um banco de dados NÃO relacional, que usa o conceito de chave:valor. Tem vários tipos de estrutura de dados inclusive arrays e objetos.

Para utilizar o MongoDB com o Node.js utilizamos o módulo "mongoose"



Node com MongoDB



Exemplo 9 /app1

Implementar novas rotas em animal para alterar e deletar.

/alterar - Alterar um animal

/deletar - Deletar um animal

Criar para Dono as rotas:

/cadastrar - Cadastrar um dono;

/listar - Cadastrar um dono;

/alterar - Alterar um dono;

/deletar - Deletar um dono;

`Animal.findByIdAndRemove`

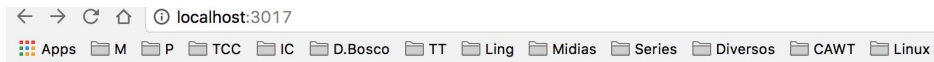
`Animal.findByIdAndUpdate`

Estrutura de dados do Dono

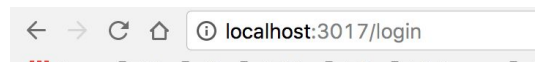
nome:	string
endereço:	string
telefone:	string
ativo:	boolean
credito:	number

MySQL é um banco de dados relacional. Para usarmos com o Node.js utilizamos o módulo 'mysql'

Em nosso exemplo, criamos uma tela de login que verifica se existe o usuário no banco



© Copyright 2017 CAWT - V.0.1.1-Beta



```
1 // 20171016204426
2 // http://localhost:3017/login
3
4 {
5   "Data": "Logado com sucesso"
6 }
```



```
1 // 20171016204536
2 // http://localhost:3017/login
3
4 {
5   "Data": "Email ou password invalido."
6 }
```

EXERCÍCIO 10

Criar uma rota no exemplo do node.js com MySQL para inserir um novo usuário.

Estrutura de dados da Tabela Usuarios:

ID - Inteiro

NOME - String

SENHA - String

NODE + TESTES

Exemplo simples de teste em node.js utilizando os módulos:

Axios - Um cliente HTTP baseado em Promise para o navegador e node.js

Mocha - Uma estrutura de teste popular do Node.js.

Chai - Uma biblioteca de asserção BDD / TDD para Node.js

Nock - Uma biblioteca de mocking e expectativas HTTP para Node.js

```
$ npm init
```

```
$ npm install --save axios
```

```
$ npm install --save-dev mocha chainock
```

```
$ mkdir test
```

```
"scripts": {  
  "test": "node_modules/.bin/mocha"  
},
```

```
$ npm test
```



- ▶ exemplo1
- ▶ exemplo2express
- ▶ exemplo3modulos
- ▶ exemplo4mvc
- ▶ exemplo5moduloGit
- ▶ exemplo6express
- ▶ exemplo7rotas
- ▶ exemplo8webexpress
- ▶ exemplo9mongodeb
- ▶ exemplo10mysql
- ▶ exemplo11testes
- ▶ Exercicio01
- ▶ Exercicio02
- ▶ Exercicio03
- ▶ Exercicio04
- ▶ Exercicio05
- ▶ Exercicio06
- ▶ Exercicio07
- ▶ Exercicio08
- ▶ Exercicio09

Exemplos e respostas dos exercícios estão disponíveis no github