

1) Instalação do NodeJs e ExpressJs:

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

<http://expressjs.com/en/starter/installing.html>

Criar um diretório para a aplicação:

```
$ mkdir myapp
```

```
$ cd myapp
```

2) Inicializar o npm

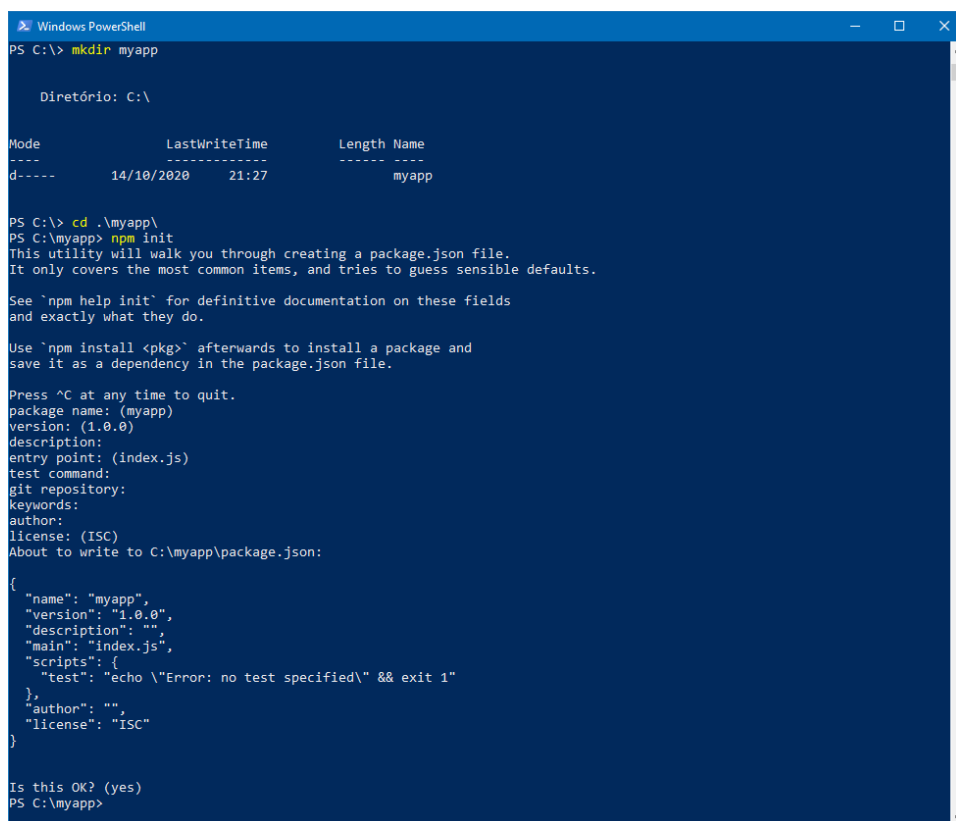
```
$ npm init
```

3) Instalar o ExpressJs

```
$ npm install express --save
```

Servidor HTTP básico:

<http://expressjs.com/en/starter/hello-world.html>



```
Windows PowerShell
PS C:\> mkdir myapp

Diretório: C:\

Mode                LastWriteTime         Length Name
----                -
d-----          14/10/2020   21:27             myapp

PS C:\> cd .\myapp\
PS C:\myapp> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` 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: (myapp)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\myapp\package.json:

{
  "name": "myapp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes)
PS C:\myapp>
```

```
Windows PowerShell
PS C:\myapp> npm install express --save
npm WARN deprecated myapp@1.0.0 No description
npm WARN myapp@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 6.481s
found 0 vulnerabilities

PS C:\myapp> ls

Diretório: C:\myapp

Mode                LastWriteTime         Length Name
----                -
d-----          14/10/2020    21:28             node_modules
-a-----          14/10/2020    21:28    32899 Capturar1.PNG
-a-----          14/10/2020    21:28    14281 package-lock.json
-a-----          14/10/2020    21:28      251 package.json

PS C:\myapp> node
Welcome to Node.js v14.13.1.
Type ".help" for more information.
> const express = require( 'express' )
undefined
> express.
express.__defineGetter__    express.__defineSetter__    express.__lookupGetter__    express.__lookupSetter__
express.__proto__          express.hasOwnProperty      express.isPrototypeOf       express.propertyIsEnumerable
express.toLocaleString

express.apply
express.caller
express.Router
express.cookieParser
express.errorHandler
express.limit
express.name
express.request
express.static
express.urlencoded
express.application
express.cookieSession
express.favicon
express.logger
express.prototype
express.response
express.staticCache
express.vhost
express.bodyParser
express.csrf
express.json
express.methodOverride
express.query
express.responseTime
express.text
>
```

1) Criar um arquivo do código a ser completado:

```
const express = require( 'express' )
const app = express()
const port = 3000
```

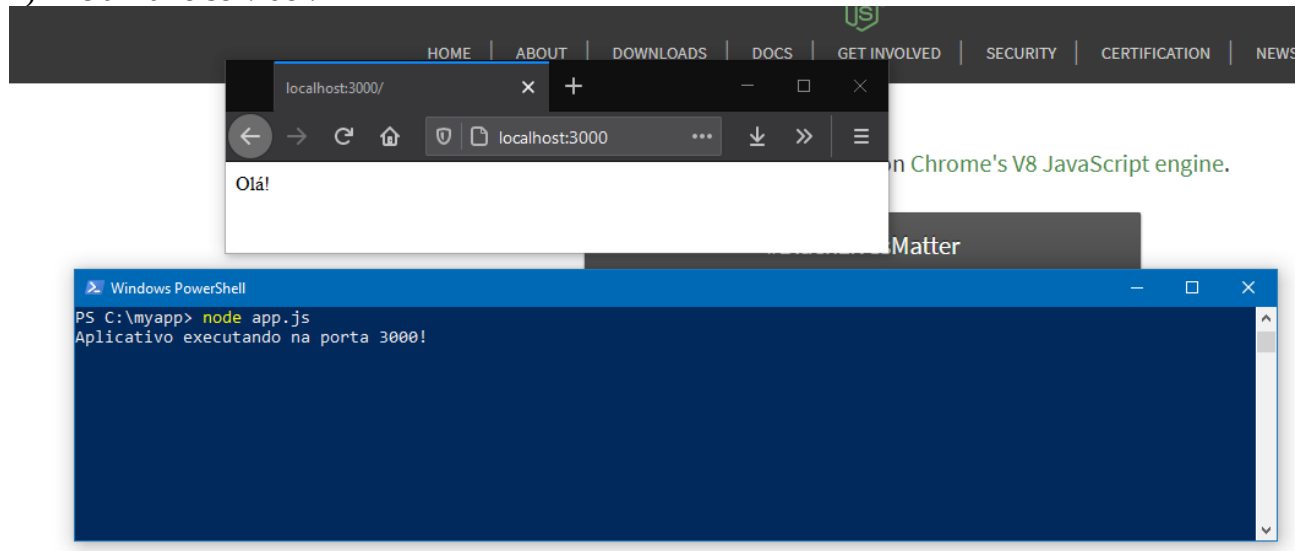
```
app.listen( port, () => {
  console.log( `Aplicativo executando na porta ${port}!\n` );
});
```

Objetos da biblioteca foram definidos como constante já que não devem ser modificadas. E as funções anônimas como *arrow functions* para simplificar a sintaxe.

Esse arquivo foi salvo com o nome *app.js* no diretório base da aplicação (*myapp/app.js*).

O método *listen* cria um servidor na porta especificada. O segundo argumento é um *callback* que será executado quando o servidor for inicializado.

2) Inicializar o servidor.



O servidor pode ser inicializado pela linha de comando:

```
$ node app.js
```

Aplicativo executando na porta 3000!

Com o servidor em execução, pode-se acessar o endereço <http://localhost:3000/> em qualquer navegador para testar as suas funcionalidades.

O servidor pode ser fechado com as teclas Ctrl+C no terminal.

3) Roteamento/Respostas as requisições HTTP

<http://expressjs.com/en/starter/basic-routing.html>

Os métodos de app são utilizados para responder mensagens HTTP. O nome do método é o nome do método HTTP. O primeiro argumento é a URL. O segundo é uma função que recebe objetos de requisição e resposta respectivamente, e deve processar uma resposta com os métodos do objeto resposta.

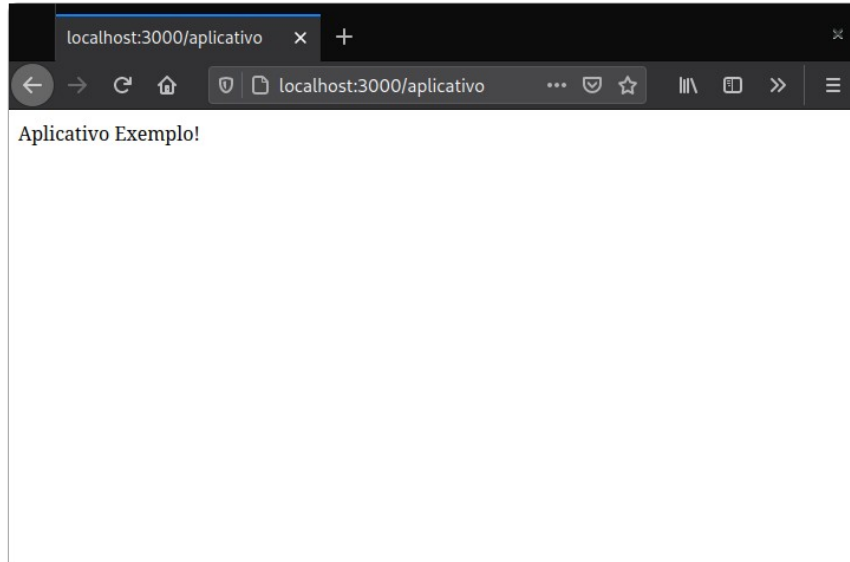
Sintaxe:

```
app.METHOD(PATH, HANDLER)
```

4) Métodos GET

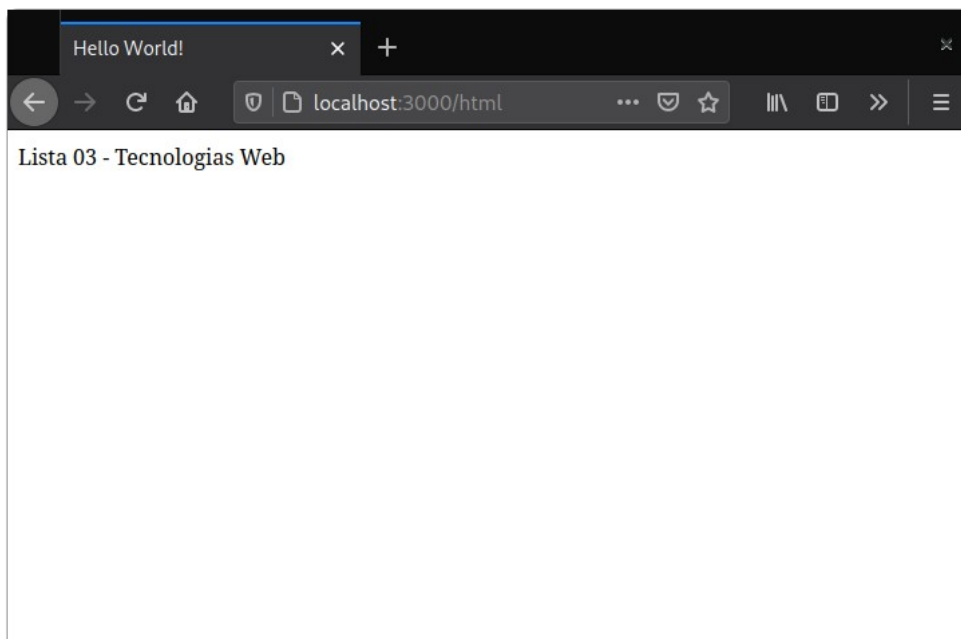
```
app.get( "/aplicativo", ( req, res ) => {  
    res.send( "Aplicativo Exemplo!\n" );  
});
```

O método GET pode ser acessado digitando-se <http://localhost:3000/aplicativo> no navegador. O método send do objeto envia a resposta, que será renderizada no navegador:



```
app.get( "/html", ( req, res ) => {  
    res.sendFile( __dirname + '/hello.html' );  
});
```

Semelhante à URL /aplicativo, mas responde com um arquivo estático hello.html. O nome do arquivo precisa ser concatenado com __dirname porque o método sendFile requer o caminho absoluto do arquivo.



Arquivo hello.html:

```
<html>
  <head>
    <meta charset="UTF-8">
    <title>Hello World!</title>
  </head>
  <body>
    <p>Lista 03 - Tecnologias Web</p>
  </body>
</html>
```

5) Métodos diferentes de GET

```
app.post( "/imagens", (req, res) => {
  res.send( "Imagem 1 - Imagem 2 - Imagem 3.\n" );
});
```

O método GET é usado quando se requisita um endereço em um computador. Para acessar métodos diferentes, deve se usar outros meios:

Utilitário de linha de comando curl (<https://curl.haxx.se/>):

```
$ curl -i -X POST http://localhost:3000/imagens
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 32
ETag: W/"20-UKlnybXxBB0uZMRm9sCXJ090mPc"
Date: Wed, 14 Oct 2020 22:05:19 GMT
Connection: keep-alive
```

Imagem 1 - Imagem 2 - Imagem 3.

```
$ curl -X DELETE http://localhost:3000/clientes/500
Cliente número 500 removido com sucesso.
```

```
$ curl -X DELETE http://localhost:3000/clientes/10
Cliente número 10 removido com sucesso.
```

Também existem plugins para navegadores que fazem essa função:

HTTP request maker: <https://addons.mozilla.org/en-US/firefox/addon/http-request-maker/>

HTTP request maker - G x +

https://addo

HTTP request Maker v

Request Response

Target Site:

http://localhost:3000/imagens

Method:

POST

Request Headers:

Content-Type:application/x-www-form-urlencoded
User-Agent:Mozilla
Accept:*/*

Body Data:

Load Submit

HTTP request Maker v

Request Response

Status Code 200

Response Headers:

connection: keep-alive
content-length: 32
content-type: text/html; charset=utf-8
date: Wed, 14 Oct 2020 22:16:15 GMT
etag: W/"20-UKlNybXxBB0uZMRm9sCXj09OmPc"
x-powered-by: Express

Response Data:

Imagem 1 - Imagem 2 - Imagem 3.

Advanced REST Client: <https://chrome.google.com/webstore/detail/advanced-rest-client/>

Request

Method Request URL
DELETE http://localhost:3000/clientes/6666 SEND

Parameters

Headers Body Variables

<> Toggle source mode + Insert headers set

Header name Header value X

ADD HEADER

✓ Headers are valid Headers size: bytes

200 OK 4.68 ms DETAILS

<> Clientes

Cliente número 6666 removido com sucesso.

Selected environment: Default

Existem muitas outras formas.

6) URLs com parâmetros

<https://expressjs.com/en/guide/routing.html> (em “Route parameters”)

```
app.delete( "/clientes/:num/", (req, res) => {  
    res.send( `Cliente número ${req.params.num} removido com  
sucesso.\n` );  
});
```

Parâmetros da URL podem ser extraídos colocando-se dois pontos nos campos desejados e acessados no campo `params` do objeto de requisição.