

P.PORTO

Introdução ao Backend com NodeJS

Programação em Ambiente Web

Índice

Introdução

Callbacks

NPM

Custom Modules

Módulos comuns em JavaScript

Padrões de Software

Introdução

O que é NodeJS?



Introdução

NodeJS não é um servidor da web

Não funciona como servidores como o Apache por definição

Para construir um servidor HTTP, é preciso codificar um servidor HTTP (com a ajuda de bibliotecas externas)

O NodeJS é apenas uma maneira de executar código JavaScript no computador

Introdução

NodeJS = Runtime Environment + JavaScript Library



Introdução

O NodeJS é um ambiente de execução de JavaScript:

- open-source
- cross-application
- permite criar ferramentas e aplicações no servidor com JavaScript

Team como pontos fortes:

- Boa performance
- Otimiza a transferência de dados e a escalabilidade de aplicações;

Introdução

O que se pode fazer com NodeJS?



Introdução

Aplicações Web

- Aplicações web tradicionais
- Aplicações tem tempo real
- Páginas Web baseadas em eventos

Serviços Web

- ex: REST

Aplicações Desktop

- Linux/Windows/OS X

Introdução

Exemplos:

- NodeJS pode gerar conteúdo de páginas dinâmicas
- NodeJS pode criar, abrir, ler, gravar, excluir e fechar ficheiros no servidor
- NodeJS pode obter dados de formulário
- NodeJS pode adicionar, eliminar, modificar dados em base de dados

Introdução

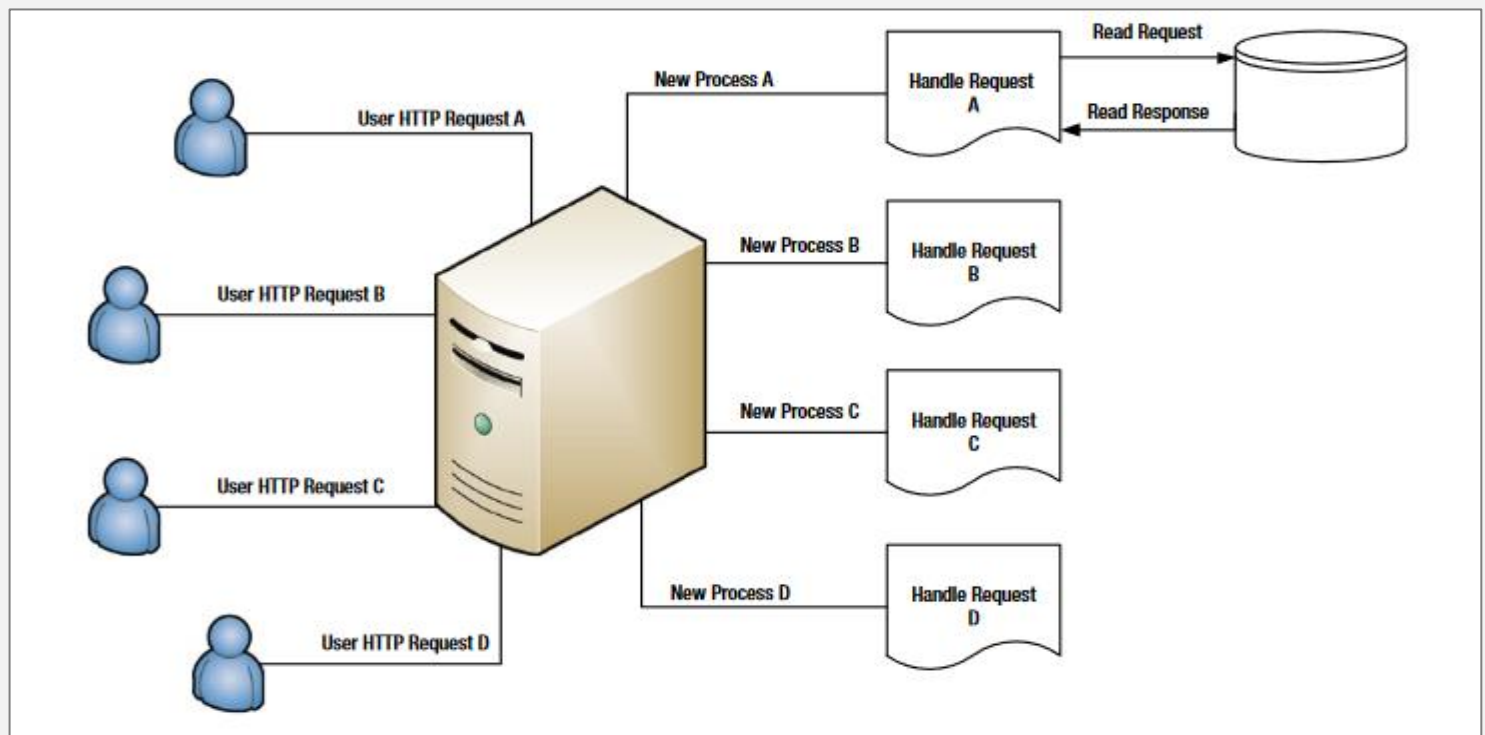
O código-fonte é escrito em JavaScript, o que significa que menos tempo é gasto a lidar com recodificação entre linguagens de programação do browser e do servidor da Web

O NodeJS foi desenvolvido para permitir aplicações Web de alto desempenho

Resolve o problema de I/O (aceder a dados em tempos de ciclo CPU) com recurso a callbacks

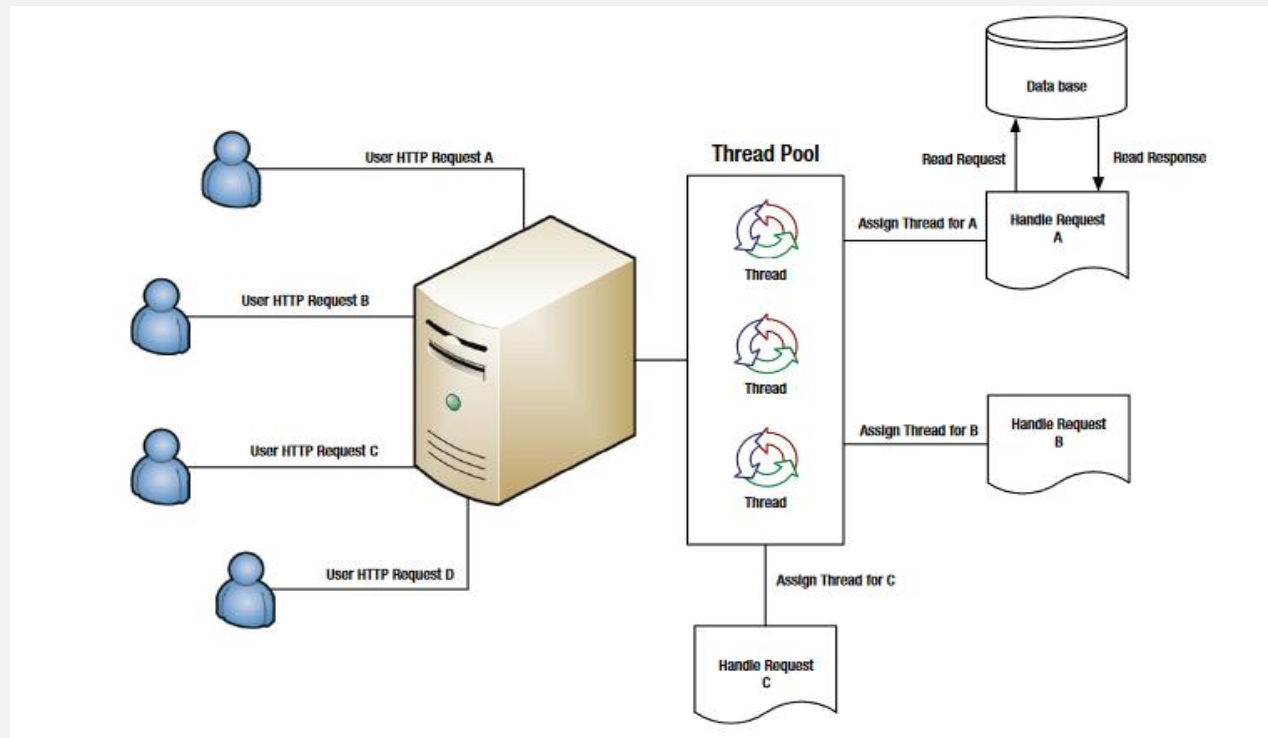
Introdução

Modelo tradicional de servidores Web para lidar com pedidos



Introdução

Modelo tradicional de servidores Web para lidar com pedidos utilizando uma Thread Pool



Introdução

Há um *overhead* sempre que se cria um processo e as *threads* são mais *leves* que os processos

Ao criar processos e threads separados para lidar com pedidos há desperdício de recursos

Introdução

Em JavaScript existe uma única thread de execução

Se há uma operação bloqueante de longa duração vamos executar outras operações de outras tarefas (ex: callbacks)

Callbacks

Um callback é uma função que irá ser chamada assincronamente quando uma determinada tarefa é realizada

NodeJS faz um grande uso de funções de callback e todas as suas apis fazem uso deles

Isso torna o NodeJS escalável, pois pode processar um número grande de pedidos sem esperar que qualquer função retorne resultados

Callbacks

Exemplo:

- uma função para ler um ficheiro pode começar a ler o ficheiro e retornar o execução ao ambiente de execução
- Quando as operações I/O do ficheiro for concluído, será chamada a função de callback
- Não há bloqueio ou espera por I/O do ficheiro

Callbacks

Qual dos dois exemplos fornecidos em baixo faz uso de callbacks?

```
var fs = require("fs");  
var data = fs.readFileSync('input.txt');  
  
console.log(data.toString());  
console.log("Program Ended");
```

```
var fs = require("fs");  
  
fs.readFile('input.txt', function (err, data) {  
    if (err) return console.error(err);  
    console.log(data.toString());  
});  
  
console.log("Program Ended");
```

Callbacks

Estes dois exemplos explicam o conceito de pedidos bloqueantes e não-bloqueantes

O primeiro exemplo mostra que o programa bloqueia até ler o ficheiro

O segundo exemplo mostra que o programa não espera pela leitura do ficheiro e imprime o "Programa Finalizado" ao mesmo tempo que o programa continua a ler o ficheiro.

Callbacks

Uma tarefa comum para um servidor da Web é abrir um ficheiro no servidor e retornar o conteúdo ao cliente no browser

Servidores tradicionais lidam com um pedido de ficheiro:

- Envia a tarefa para o sistema de ficheiros do computador
- Aguarda enquanto o sistema de ficheiros abre e lê o ficheiro
- Retorna o conteúdo para o cliente
- Pronto para lidar com o próximo pedido

Callbacks

NodeJS processa o pedido de um ficheiro desta forma:

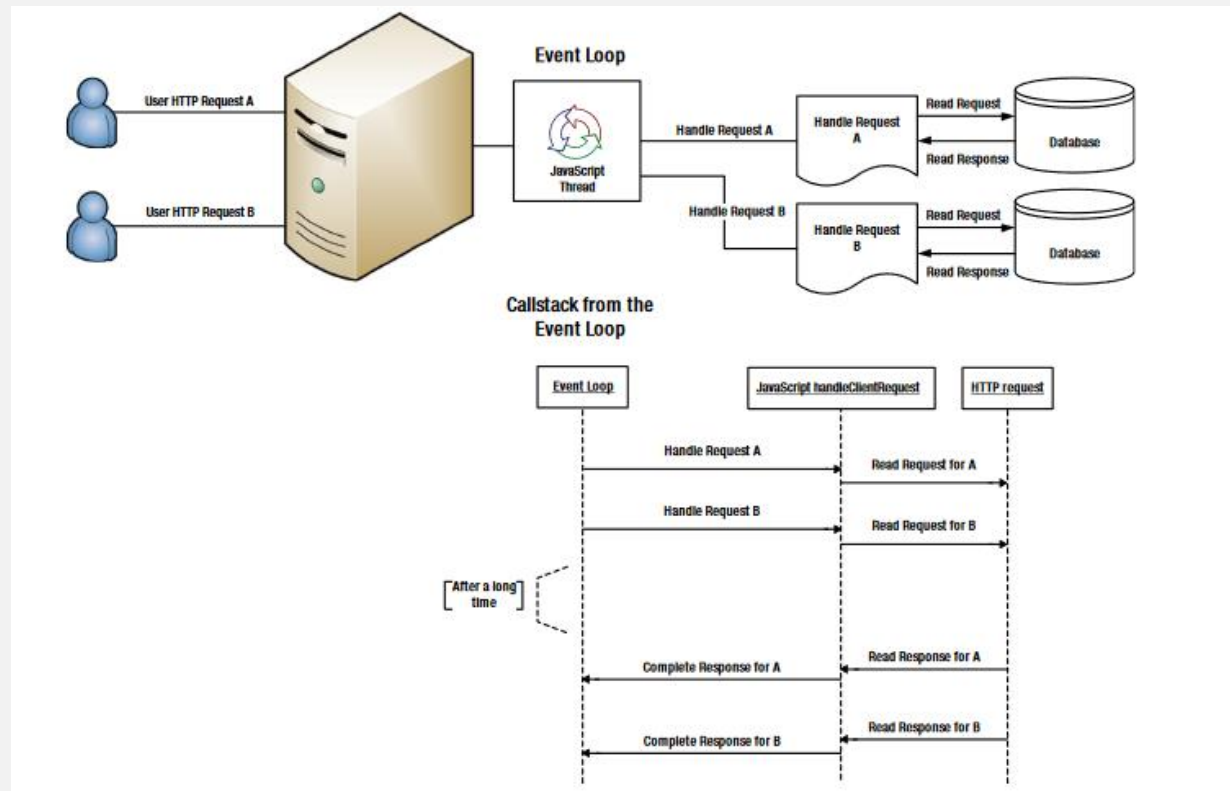
- Envia a tarefa para o sistema de ficheiros do computador.
- Pronto para lidar outros pedidos enquanto espera pelos ficheiros do pedido anterior
- Quando o servidor recebe o ficheiro do sistema de ficheiros, termina o pedido original na primeira oportunidade disponível

NodeJS:

- elimina a espera síncrona
- executa programação assíncrona, sem bloqueio
- é eficiente no uso de memória

Callbacks

Considere o cenário de uma aplicação que lida com muitos dados e que utiliza um *dataset* de uma base de dados



Callbacks

Em NodeJS todo o trabalho está numa única thread, o que resulta num consumo inferior de recursos (menos thread context switching e CPU load)

O I/O assíncrono é útil para aplicações que lidam com muitos dados

Callbacks

Como podemos iniciar callbacks não bloqueantes?

Através de bibliotecas e APIs disponíveis a partir do runtime JavaScript:

- `setTimeout(...)`
- Operações I/O
- Operações sobre a rede
- ?

Callbacks

Como organizamos o encadementamento de operações baseadas em callback?

- Encadear funções

```
function callback(args){  
  console.log("Do Work");  
  setTimeout(callback2(args), 5000);  
}  
  
function callback2(args){  
  console.log("Do more work");  
}  
  
function workA( args, callback ){  
  // Do work after 5000s  
  setTimeout(callback(args), 5000);  
}  
  
console.log("Program Continues...");
```


Callbacks

Como organizamos o encadramento de operações baseadas em callback?

- Promisses

```
function workA(args) {  
  let p = new Promise(function (resolve, reject) {  
    console.log("Do Work");  
    // If there are no errors  
    resolve(args)  
    // If there may be errors  
    // reject('Error Message')  
  })  
  return p;  
}  
  
function workB(args) {  
  let p = new Promise(function (resolve, reject) {  
    console.log("Do more work");  
    // If there are no errors  
    resolve(args)  
    // If there may be errors  
    // reject('Error Message')  
  })  
  return p;  
}  
  
workA.then(function(result){  
  return workB(result)  
}).then((result) => {  
  console.log("Chained work ends")  
}).catch((error) =>{  
  console.log("An exception was thrown in one of the promises");  
})
```

Callbacks

Como organizamos o encadramento de operações baseadas em callback?

- Await / Async

```
async function workA(args) {  
  console.log("Do Work");  
}  
  
async function workB(args) {  
  console.log("Do More Work!");  
}  
  
workA.then((result) => workB(result))  
  .then((result) => {  
    console.log("Chained work ends")  
  }).catch((error) => {  
    console.log("An exception was thrown in one of the promises");  
  })
```

Callbacks

Como organizamos o encadramento de operações baseadas em callback?

- Await / Async

```
async function workA(args) {  
  console.log("Do Work");  
}  
  
async function workB(args) {  
  console.log("Do More Work!");  
}  
  
async function assincronousExecution(args){  
  let res1 = await workA(args);  
  let res2 = await workB(args);  
  console.log("Chained work ends")  
}  
  
assincronousExecution(args).catch((error) => {  
  console.log("An exception was thrown in one of the promisses");  
})
```

Callbacks

Thread starvation

- Habitualmente durante a duração de uma função invocada através de uma aplicação GUI, mais nenhum outro evento pode ser processado
- Consequentemente, se estivermos a executar uma tarefa de longa duração (como um handler de um click) a GUI estará não responsiva (o que se denomina como “starvation”)

Como o NodeJS é baseado no mesmo ciclo de evento que os programas GUI pode também sofrer de “starvation”

Callbacks

Thread starvation

– Exemplo:

```
console.log('hello project');

function fibonacci(n){
  if ( n < 2 ){
    return 1;
  } else {
    return fibonacci(n-2)+fibonacci(n-1);
  }
}

console.time('timer');
setTimeout( function(){
  console.timeEnd('timer'); // Demora mais do que 1000ms por causa de thread starvation
}, 1000 )

fibonacci(44);
```

Callbacks

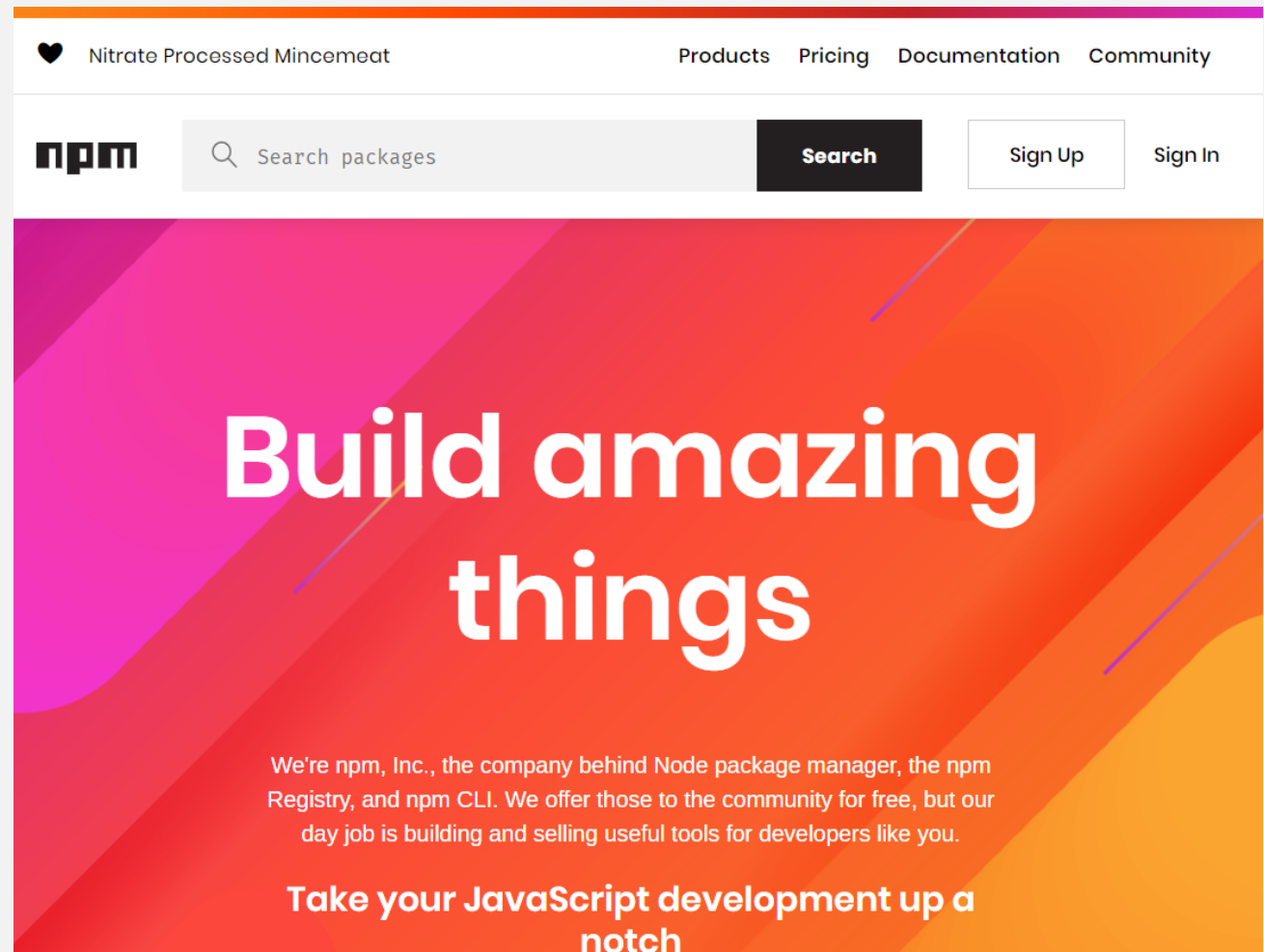
Thread starvation

— Exemplo:

- Temos um NodeJS event-loop (setTimeout) e uma função que mantém a thread do JavaScript ocupada (fibonacci)
- Antes do tempo finalizar, executamos uma função que consume muito tempo de CPU ou seja “reserva” o CPU e a thread do JavaScript
- Enquanto a função “reserva” a thread JavaScript, o event-loop não pode invocar mais nada e o “time-out” é atrasado

NPM

Npm é um
gestor de
packages
para
JavaScript



NPM

Contém :

- Página de internet - um lugar onde você pode navegar por pacotes, ler os documentos e encontrar informações gerais sobre o npm
- Registro npm - é uma base de dados que contém as informações e o código das packages
- Cliente npm - é uma ferramenta instalada em uma máquina do desenvolvedor que permite publicar packages, instalar pacotes e atualizar packages

NPM

Para inicializar o npm no projeto é necessário correr o comando:

- npm init

Após a inicialização é criado um ficheiro packages.json com informação sobre as dependências e características do projeto

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

NPM

Devemo criar o ficheiro *index.js* para iniciar o nosso script

```
JS index.js
1 console.log('hello project');
2
```

Alterar o ficheiro *package.json* para automatizar o início de um projeto nodeJS

```
{
  "name": "paw_demo",
  "version": "1.0.0",
  "description": "paw sample",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

NPM

Para correr o projeto e executar o projeto devemos apenas usar o comando:

- npm start

```
fab@laptop-work: /mnt/c/Users/Faand/PersonalProjects/web/PAW/Aula4$ npm start  
  
> paw_demo@1.0.0 start /mnt/c/Users/Faand/PersonalProjects/web/PAW/Aula4  
> node index.js  
  
hello project
```

Este comando é equivalente a:

- node index.js

NPM

Para instalar bibliotecas javascript é necessário apenas correr o comando:

- `npm install "package" --save`

As bibliotecas são instaladas dentro da pasta
`node_modules`

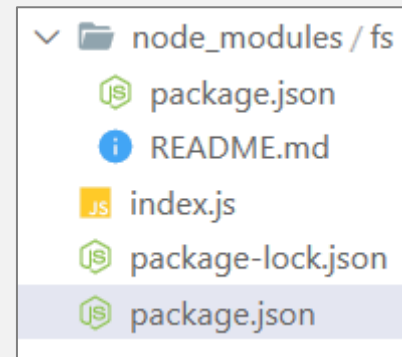
NPM

Exemplo do uso do npm para instalar o package *fs*

– npm install --save fs

```
{
  "name": "paw_demo",
  "version": "1.0.0",
  "description": "paw sample",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "fs": "0.0.1-security"
  }
}
```

package.json



pasta do projeto

NPM

O que é um módulo no NodeJS?

- Para simplificação podemos considerar módulos o mesmo que bibliotecas de JavaScript.
- Representa um conjunto de funções que queremos incluir na aplicação
- Existe um conjunto de módulos integrados que podemos usar sem qualquer instalação adicional.
- Podemos consultar a lista de módulos em:
 - <https://NodeJS.org/api/modules.html>

Incluir Módulos

- Para incluir um módulo, é necessário usar a função `require()` com o nome do módulo:
 - ex: `var http = require('http');`

NPM

O que é um módulo no NodeJS?

- Para simplificação podemos considerar módulos o mesmo que bibliotecas de JavaScript.
- Representa um conjunto de funções que queremos incluir na aplicação
- Existe um conjunto de módulos integrados que podemos usar sem qualquer instalação adicional.
- Podemos consultar a lista de módulos em:
 - <https://NodeJS.org/api/modules.html>

NPM

Incluir Módulos

- Para incluir um módulo, é necessário usar a função `require()` com o nome do módulo:
 - ex: `var http = require(http);`
- Podemos usar as funções exportadas do módulo a partir da variável criada

Nota: Antes de usar um módulo devemos garantir que está instalado via *npm install*

Custom Modules

É possível também criar os nossos próprios módulos via CommonJS

Exemplo

- Criar um módulo que retorna a data e hora atuais
 - firstMod.js:

```
exports.myDateTime = function () {  
  return Date();  
};
```

Utilizar a exports palavra-chave para disponibilizar propriedades e métodos fora do ficheiro do módulo

Guarde o código acima em um ficheiro chamado "myfirstmodule.js"

Custom Modules

Podemos agora incluir o módulo criado usando também o método `require()`

```
var dt = require('./firstMod');  
console.log(dt.myDateTime)
```

Podemos desta forma estruturar as nossas aplicações NodeJS em vários módulos interligados entre si.

Módulo fs

O módulo de sistema de ficheiros de NodeJS permite que trabalhar com o sistema de ficheiros do computador.

Mais uma vez, para incluir o módulo do sistema de ficheiros, usaremos o método `require()` :

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

Módulo fs

O uso mais comum para o módulo do sistema de ficheiros é:

- Ler ficheiros
- Criar ficheiros
- Eliminar ficheiros
- Editar ficheiros

Módulo fs

Podemos usar o módulo http e fs para servir páginas html no nosso servidor

A páginas serão representadas por ficheiros html normais que enviados aos clientes

Considere-se o seguinte ficheiro html:

```
<html>  
<body>  
<h1>PAW</h1>  
<p>Resumos aula 4</p>  
</body>  
</html> |
```

Módulo fs

Considere agora um ficheiro “server.JavaScript” que irá servir os pedidos HTML

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    res.end();
  });
}).listen(8080);
```

A página é lida da pasta do servidor e encaminhada a clientes do servidor

Módulo fs

O módulo de sistema de ficheiros possui métodos para criar novos ficheiros:

- `fs.appendFile()`
- `fs.open()`
- `fs.writeFile()`

Mais opções estão disponíveis neste módulo e para seu tratamento deve ser consultada a documentação oficial

Módulo URL

O módulo URL divide um endereço da Web em partes legíveis.

Para incluir o módulo de URL, usamos também o método `require()` :

- `var url = require('url');`

Analisar um endereço com o método `url.parse()`, irá retornar um objeto URL com cada parte do endereço como propriedades.

Módulo URL

Exemplo

- Dividir um endereço web em partes legíveis:

```
var url = require('url');  
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';  
var q = url.parse(adr, true);  
  
console.log(q.host); //returns 'localhost:8080'  
console.log(q.pathname); //returns '/default.htm'  
console.log(q.search); //returns '?year=2017&month=february'  
  
var qdata = q.query; //returns an object: { year: 2017, month: 'february' }  
console.log(qdata.month); //returns 'february'
```

Módulo URL

Agora sabemos como analisar a string de consulta, aprendemos como fazer Node.js se comportar como um servidor de ficheiros

Vamos combinar os dois, e servir o ficheiro pedido pelo cliente

Módulo Express

Permite a criação de um servidor em NodeJS

Tem uma arquitetura simples e agnóstica do ponto de vista de padrões de software

Permite filtrar pedidos ao servidor por método (GET, POST, ..) e por rota

Módulo Express

Hello World em Express:

- Inicie o projeto com o npm
- Escreva no ficheiro index.js o exemplo fornecido
- Inicie o projeto com *npm start* ou *node index.js*

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

const app = express();

app.get('/', function(req, res){
  res.send('Hello World');
});

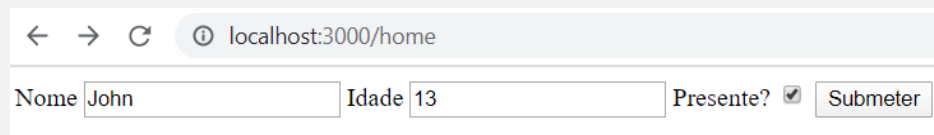
app.listen(3000, () => {
  console.log('Servidor a correr em: http://localhost:3000');
});
```

Módulo Express

Capturar dados de um form

- método GET

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <form method="GET" action="/resposta">
      <label>Nome</label>
      <input type="text" name="nome" id="name">
      <label>Idade</label>
      <input type="number" name="idade" id="age">
      <label>Presente?</label>
      <input type="checkbox" name="presente" id="present">
      <input type="submit" >
    </form>
  </body>
</html>
```



A screenshot of a web browser window. The address bar shows 'localhost:3000/home'. The page displays a form with three input fields: 'Nome' (containing 'John'), 'Idade' (containing '13'), and 'Presente?' (with a checked checkbox). A 'Submeter' button is located to the right of the 'Presente?' field.

É enviado um pedido ao servidor na seguinte forma

- `http://localhost:3000/resposta?nome=John&idade=13&presente=on`

Módulo Express

Capturar dados de um form

- método GET

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <form method="GET" action="/resposta">
      <label>Nome</label>
      <input type="text" name="nome" id="name">
      <label>Idade</label>
      <input type="number" name="idade" id="age">
      <label>Presente?</label>
      <input type="checkbox" name="presente" id="present">
      <input type="submit" >
    </form>
  </body>
</html>
```

```
const express = require('express');
const url = require('url');
const fs = require('fs');

const app = express();

app.get('/home', function(req, res){

  fs.readFile('./index.html', function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  })
});


app.get('/resposta', function(req,res){
  let parsed_url = url.parse(req.url, true);
  console.log(req.url);
  console.log(parsed_url.query.nome);
  console.log(parsed_url.query.idade);
  console.log(parsed_url.query.presente);
  res.send(<h1> ${parsed_url.query.nome} foi prcessado! </h1>`);
});
```

Módulo Express

Capturar dados de um form

- método POST

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <form method="POST" action="/resposta">
      <label>Nome</label>
      <input type="text" name="nome" id="name">
      <label>Idade</label>
      <input type="number" name="idade" id="age">
      <label>Presente?</label>
      <input type="checkbox" name="presente" id="present">
      <input type="submit" >
    </form>
  </body>
</html>
```



← → ↻ ⓘ localhost:3000/home

Nome Idade Presente? ☒

É enviado um pedido ao servidor na seguinte forma

- http://localhost:3000/resposta

Os dados são passado no body do pedido HTTP!

Módulo Express

Capturar dados de um form

- método POST

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <form method="POST" action="/resposta">
      <label>Nome</label>
      <input type="text" name="nome" id="name">
      <label>Idade</label>
      <input type="number" name="idade" id="age">
      <label>Presente?</label>
      <input type="checkbox" name="presente" id="present">
      <input type="submit" >
    </form>
  </body>
</html>
```

```
const express = require('express');
const url = require('url');
const fs = require('fs');
const querystring = require('querystring');

const app = express();

app.get('/home', function(req, res){
  fs.readFile('./index.html', function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
});

app.post('/resposta', function(req,res){
  var data = "";
  req.on('data', function(chunk){ data += chunk})
  req.on('end', function(){
    req.rawBody = data;
    req.jsonBody = querystring.parse(data);
    console.log(req.rawBody);
    console.log(req.jsonBody);
    res.send(`<h1> ${req.jsonBody.nome} foi processado! </h1>`);
  })
});

app.listen(3000, () => {
  console.log('Servidor a correr em: http://localhost:3000');
});
```


Módulo Express

Servidor de ficheiros

- Serve ficheiros html
- Usa o módulo express para filtrar pedidos ao servidor
- Quais os problemas?
- Sugestões para melhorar?

```
const express = require('express');
const url = require('url');
const fs = require('fs');

const app = express();

app.get('/*', function(req, res){
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  console.log("Foi pedido: " + filename);
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    //res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  })
});
app.listen(3000);
```

Módulo Express

Na verdade iremos
melhorar a nosso servidor
nas próximas aulas

Iremos usar o módulo
express:

- como uma framework
- implementar padrões de software
- metodologias de desenvolvimento web

```
const express = require('express');
const url = require('url');
const fs = require('fs');

const app = express();

app.get('/*', function(req, res){
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  console.log("Foi pedido: " + filename);
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    //res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  })
});
app.listen(3000);
```

Padrões de Software

Padrões de design representam um conjunto de regras de boas práticas usadas por programadores de software em linguagens de programação orientados a objetos

Os padrões de design são soluções para problemas gerais e comuns no desenvolvimento de software

Padrões de Software

Os padrões de software tem 2 principais vantagens:

- Oferecem uma solução padrão para todos os programadores
- São considerados as melhores práticas

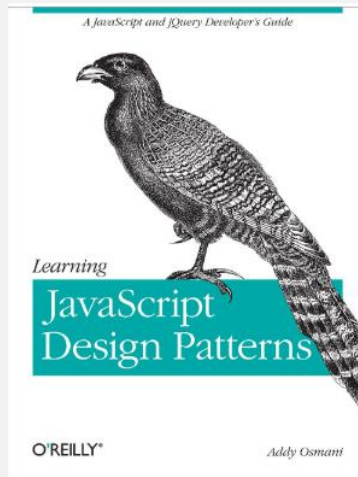
Padrões de Software

Em termos de padrões de software, estes podem ser classificados em categorias como:

- Padrões de criação
- Padrões Estruturais
- Padrões de Comportamento
- ...

Padrões de Software

Uma boa referência de padrões em Javascript pode ser consultada no seguinte livro:



Fonte:

<https://addyosmani.com/resources/essentialjsdesignpatterns/book/index.html>

É um bom ponto de partida para entender a construção de módulos e a sua implementação em plataformas como NodeJS

Referências

NPM website

- <https://www.npmjs.com/package/npm>

NodeJS

- <https://nodejs.org/dist/latest-v12.x/docs/api/>
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs
- <https://expressjs.com/en/starter/hello-world.html>

Padrões de software

- <https://addyosmani.com/resources/essentialJavaScriptdesignpatterns/book/>

P.PORTO

Introdução ao Backend com NodeJS

Programação em Ambiente Web