Laboratório Nº 2: Desenvolvimento de um Servidor Web em Node.js e Express.js

Programação e Integração de Serviços
2020/2021

Prof. Filipe Mariano

Objetivos

- Criar um Servidor Web recorrendo ao Node JS e Express JS
- Distribuir ficheiros estáticos existentes no servidor aos clientes

1. Introdução

Express é uma *framework* para aplicações web que fornece um conjunto de funcionalidades que propícia um desenvolvimento rápido de aplicações web. É uma camada desenvolvida por cima do Node JS para auxiliar a gerir um servidor e respetivas rotas. As suas principais características são:

- Configuração de um *middleware* para responder a pedidos HTTP de forma assíncrona.
- Definição de um roteamento utilizado para executar diferentes ações baseadas num verbo HTTP e num URL.
- Renderização dinâmica de páginas HTML através da passagem de argumentos recorrendo a templates.

2. Instalação

Para o desenvolvimento do servidor será necessário efetuar a instalação do módulo `express` através do seguinte comando no terminal, gravando a sua dependência no ficheiro `packages.json`:

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

3. Ligação de Outros Módulos ao Express

Para tornar o desenvolvimento mais simples, pode ser instalado o `nodemon`. O nodemon permite reiniciar o servidor sempre que se fizer uma alteração em algum ficheiro, caso contrário seria necessário reiniciar o servidor manualmente após cada alteração.

```
npm install -g nodemon
```

O `-g` significa que o módulo irá ser instalado globalmente para o Node.js e não em particular para esta aplicação.

4. Iniciar Servidor

Após a instalação do módulo `express` é possível iniciar um servidor básico com uma rota na raíz, com o seguinte código:

```
var express = require('express'); //importação do módulo

var app = express(); //inicialização do módulo

//rota na raíz
app.get('/', function(req, res){
   res.send('Hello world!'); //resposta do servidor ao pedido
});

app.listen(8081); //servidor à escuta na porta 8081
```

Para iniciar o servidor terá apenas de chamar o comando `node «nome_do_ficheiro»` ou clicar no ícone de iniciar o *debug*.

5. *Request* e *Response*

Após a colocação de um servidor à escuta, desenrola-se um processo de comunicação entre cliente-servidor através dos pedidos do primeiro e as respetivas respostas do segundo. Para compreender esses pedidos é necessário também perceber uma das principais partes que constituem um pedido HTTP, que é como os clientes (por exemplo através de um *browser*) solicitam uma página a um servidor e como essa página é retornada.

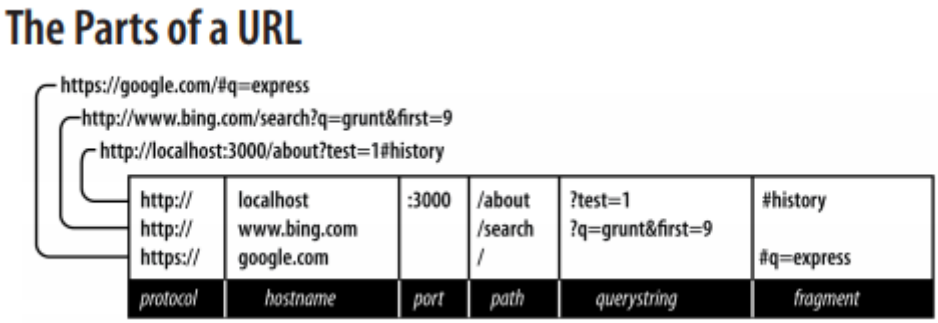

Figura 1: As diversas partes que constituem um URL. Fonte: (Brown, 2014)

De todas as partes que constituem um URL, apenas o *fragment* não é enviado para o servidor, sendo de uso exclusivo do browser para (normalmente) mostrar uma área específica da página retornada.

Quando se constrói um servidor web em Node JS através da *framework* Express, a maioria do que é feito envolve de início um objeto de `request` e por fim um objeto de `response`. Esses dois objetos representam os primeiros parâmetros que costumam ser observados nas funções de *callback* do servidor, desempenhando um importante papel na medida que encapsulam características relativas ao pedido do cliente e à resposta que o servidor irá dar.

Exemplo de função de callback com o objeto *request* e *response* associado a uma rota definida para o verbo GET na raíz:

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

As principais propriedades do objeto *request* são:

- **req.params**: array que contém os parametros associados às rotas.
- **req.body**: um objeto que contém os parâmetros que são enviados no corpo do pedido. Requer que o corpo do pedido seja analisado e convertido para um formato simples de manipular, através de módulos externos como o *body-parser*.
- **req.headers**: cabeçalhos enviados no pedido do cliente.
- **req.route**: informação acerca da rota.
- **req.cookies**: objeto contendo os valores de *cookies* enviados pelo cliente.
- **req.url, req.protocol, req.host, req.path, req.query**: informação associada ao url do pedido (ver figura 1).
- **req.method**: indica o verbo HTTP do pedido do cliente.

As principais propriedades/métodos do objeto *response* são:

- **res.status(código)**: Atualiza o código HTTP a ser enviado ao cliente. Por omissão o código é 200 (OK).
- **res.redirect([código],url)**: Redireciona o cliente para outra página. Por omissão, o código é 302 (Found).
- **res.send(body)**: Envia uma resposta ao cliente cujo cabeçalho de Content-Type enviado será `text/html`. Caso o argumento body enviado seja um objeto ou array ao invés de uma string, a resposta será enviada no formato JSON. Contudo para o fazer é recomendado utilizar o método `res.json`.
- **res.json(json_string)**: Envia JSON para o cliente.
- **res.sendFile(caminho,[callback])**: este método lê um ficheiro especificado pelo argumento path e envia o seu conteúdo para o cliente.
- **res.render(view, callback)**: Renderiza uma *view* de acordo com o motor de templates que foi configurado no Express.

6. Roteamento

O roteamento refere-se a determinar como o sistema deverá responder a um pedido de um cliente para um determinado caminho, composto por um URL e um verbo HTTP. As rotas são os caminhos que se pretende que sejam utilizados pelos clientes e nos quais o servidor será programado para dar determinada resposta.

No Express para definir as rotas deve-se utilizar o objeto que possui a criação do express, normalmente denominado `app`. Utilizando essa variável só é necessário depois indicar qual o verbo da rota que se pretende definir, o caminho e a função de *callback* quando chegar o pedido do cliente (`app.verbo(caminho,callback)`).

Exemplo de roteamento através do Express para diferentes caminhos e verbos GET e POST:

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

```
// Pedido de GET para a raíz
app.get('/', function (req, res) {
   res.send(`Verbo: ${req.method} URL: ${req.url}`);
});

// Pedido de POST para a raíz
app.post('/', function (req, res) {
   res.send(`Verbo: ${req.method} URL: ${req.url} BODY: ${req.body}`);
});

app.listen(8081, function () {
    console.log("Servidor Iniciado!");
});
```

No âmbito da Unidade Curricular de Programação e Integração de Serviços apenas vamos abordar os verbos mais comuns de serem utilizados (GET e POST), assim como os que correspondem às operações de *Create*, *Read*, *Update* e *Delete* (CRUD), que são por exemplo utilizadas na construção de APIs RESTful que iremos abordar noutras aulas.

6.1. Módulos de Rotas

Como puderam constatar o roteamento desempenha um importante papel na definição daquilo que os clientes podem navegar/aceder. No entanto, definir as rotas de um sistema todas num só ficheiro (tal como foi feito no `index.js`) não é uma boa prática, uma vez que à medida que o sistema vai crescendo torna-se cada vez mais difícil de gerir e manter. Assim, é comum realizar pequenos módulos de rotas de acordo com o contexto em que se inserem, que são agregados ao objeto `Express.Router`.

Exemplo de rotas definidas num módulo:

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res){
   res.send('GET raíz');
});
router.post('/', function(req, res){
   res.send('POST raíz');
});
router.get('/teste', function(req, res){
   res.send('GET /teste');
});

//exportar as rotas para serem utilizadas no ficheiro index.js
module.exports = router;
```

```
//ficheiro index.js
var express = require('Express');
```

```
var app = express();

var rotas = require('./routes/rotas.js');

app.use('/rotas', rotas);

app.listen(8081);
```

A chamada da função `app.use` na rota `/rotas` atribui o roteamento definido no módulo ao caminho especificado. Assim, qualquer que seja o pedido para `/rotas`, será tratado pelo roteamento definido no módulo `rotas.js`. A rota `/` e `/teste` nesse módulo são sub-rotas de `/rotas`. Escrevendo esse código no projeto e visitando o http://localhost:8081/rotas/ poderá ver o conteúdo enviado pelo servidor na função de *callback* definida no módulo.

A estruturação das rotas neste modo é muito útil como meio de modularizar o código do servidor desenvolvido. Desta forma, ajuda a escrever um código mais simples de manter/gerir, sendo encarado como uma boa prática definir as rotas relativas a uma entidade num único ficheiro/módulo e incluí-lo posteriormente no ficheiro principal do servidor (`index.js`).

6.2. Rota para Páginas de Erro

Também é possível definir uma rota para uma página de erro que informe o utilizador por ter ocorrido um erro devido ao url ser inválido, por não existir nenhuma rota definida para o pedido que chegou ao servidor. Para tal, é apenas necessário definir a seguinte rota:

```
app.get('*', function(req, res){
   res.send('Erro, URL inválido.'); //apenas chegará a esta rota caso o pedido que
chegue ao servidor não corresponda a nenhuma das rotas definidas
});
```

É de enorme importância que esta rota seja a última a ser colocada, visto que as rotas definidas em primeiro lugar serão as primeiras a ser analisadas.

7. Distribuir Ficheiros Estáticos

Para servir ficheiros estáticos o Express fornece um *middleware* interno que permite definir diretorias como estáticas, útil principalmente para guardar imagens, CSS, ficheiros JavaScript, páginas HTML ou qualquer outro recurso que se pretenda que possa ser acedido pelo cliente. Assim, esses ficheiros já poderão ser carregados pelos browsers nas diversas páginas que o servidor web fornecer. Como exemplo, se os ficheiros estáticos de um website estiverem numa pasta cujo nome é `public` a instrução a ser colocada seria:

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

Criando a pasta `public` e colocando algumas imagens numa diretoria `images` já será possível visualizar através do browser as imagens disponíveis no servidor para o website.

Os caminhos que possuir para páginas HTML estáticas deverá removê-los das rotas, uma vez que não será necessário porque a diretoria foi definida como estática.

Exemplo enviando o ficheiro html:

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

var app = express();

app.get('/', function (req, res) {
    res.sendFile(__dirname + '/public/index.html');
});
```

Exemplo em que não é preciso enviar o ficheiro porque se definiu a pasta como *static*:

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

var app = express();

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

8. Enviar Dados para o Servidor através de Formulários

Uma das formas mais comuns da passagem de dados dos clientes para o servidor é através de formulários, nomeadamente através dos verbos GET e POST (são os únicos verbos permitidos através da etiqueta `form`). De seguida, são apresentados dois exemplos semelhantes de páginas HTML e em que o(s) formulário(s) enviarão dados para o servidor utilizando esses verbos.

Exemplo com GET:

```
<!--Código do Cliente - Página HTML-->
<html>
   <body>
      <form action="/formulario_get" method="GET">
         Nome: <input type = "text" name="nome" required>
         Morada: <input type = "text" name="morada">
         <input type="submit" value="Submit">
      </form>
   </body>
</html>
```

```
//Código do Servidor - Criação de Servidor e Rota
var express = require('express');
var app = express();
```

```
app.use(express.static(__dirname + '/public')); //definição de uma pasta 'public'
como estática

//Rota que será chamada quando se clicar no botão de submit do formulário
app.get('/formulario_get', function (req, res) {
   var response = {
      nome:req.query.nome, //com o GET os parâmetros vêm por querystring
      morada:req.query.morada
   };
   console.log(response);
   res.send(JSON.stringify(response));
});

app.listen(8081);
```

Exemplo com POST:

```
<!--Código do Cliente - Página HTML-->
<html>
   <body>
      <form action="/formulario_post" method="POST">
         Nome: <input type = "text" name="nome" required>
         Morada: <input type = "text" name="morada">
         <input type="submit" value="Submit">
      </form>
   </body>
</html>
```

```
//Código do Servidor - Criação de Servidor e Rota
var express = require('express');
var app = express();

app.use(express.urlencoded()); //Parse URL-encoded body (pedidos de formulários)

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

//Rota que será chamada quando se clicar no botão de submit do formulário
app.post('/formulario_post', function (req, res) {
   var response = {
      nome:req.body.nome, //com o POST os parâmetros vêm no body do pedido
      morada:req.body.morada
   };
   console.log(response);
   res.send(JSON.stringify(response));
});

app.listen(8081);
```

9. Funções de *Middleware*

As funções de *middleware* são funções que têm acesso ao objeto do *request*, ao *response* e à próxima ação de *middleware* a ser executada (através da chamada à função `next()`). Geralmente, são funções utilizadas para modificar os objetos supramencionados para realizarem parsing ao body dos pedidos, adicionar cabeçalhos à resposta do servidor, etc. O exemplo seguinte demonstra uma função de *middleware* que é chamada a cada pedido do cliente:

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

//Função de middleware para registar um log a cada pedido
app.use(function(req, res, next){
   console.log("Novo pedido recebido às: " + Date.now());

   //A seguinte função é muito importante e indica que existe ainda processamento
a ser feito
   next();
});

app.listen(8081);
```

Também pode ser utilizado de forma a restringir a sua chamada apenas a certas rotas. Para tal, deverá indicar a rota ou parte dela:

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

app.use('/rotas', function(req, res, next){
   console.log("Novo pedido a /rotas recebido às: " + Date.now());
   next();
});

app.get('/rotas', function(req, res){
   res.send('teste');
});

app.listen(8081);
```

Um dos aspetos mais importantes acerca de *middleware* em Express é a ordem pela qual são escritas, são executadas e qual a rota correspondente. De seguida, será possível ver um exemplo em que é possível utilizar o *middleware* antes e depois da função de *callback* da rota atuar.

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

//1º middleware
```

```
app.use(function(req, res, next){
   console.log("Start");
   next(); //importante para continuar o processamento
});

//Rota
app.get('/', function(req, res, next){
   res.send("Middle");
   next(); //importante para continuar o processamento
});

app.use('/', function(req, res){
   console.log('End'); //finaliza por isso não precisa do next()
});

app.listen(8081);
```

10. Exercícios

Agora que foram dados a conhecer/relembrar os diferentes aspetos existentes na criação de um servidor web e de como o cliente usufrui dos recursos disponibilizados pelo mesmo, está na altura de realizar o vosso próprio servidor.

1. Criar um projeto no VSCode `lab2` fazer as seguintes tarefas:
a. Executar o comando `npm init` para criação do ficheiro de configuração.
b. Criar o ficheiro `index.js` e escrever o código base para criação de um servidor em Node JS utilizando o módulo `express` (ver secção 4.), testando no browser se a frase que aparece no endereço http://localhost:8081/ é `Hello World!`.
c. Criar uma pasta `public` e dentro dessa diretoria deverão ser criadas outras três pastas: `images`, `css` e `js` (ver secção 7. acerca dos ficheiros estáticos).
d. Criar uma página `index.html` dentro da pasta public que será a vossa página inicial deste laboratório. Escrever algum html para que a página contenha:

- Menu com 3 hiperligações: Home (`index.html`), Formulário GET (`form-get-aluno.html`) e Formulário POST (`form-post-aluno.html`).
- Colocar associado um ficheiro css `default.css`, na diretoria `public/css` com pelo menos 1 regra. Deverá alterar a rota existente no servidor para a raís do website, para agora servir esta página criada.
- Colocar os seus dados (nº de aluno e nome) e o nome desta disciplina "Programação e Integração de Serviços".

2. Criar a página de `Formulário GET`, dentro da pasta public, cujo ficheiro será `form-get-aluno.html`.
a. Essa página deverá ter um formulário com 4 campos que são:

- `número` (obrigatório) e aceita apenas números
- `nome` (obrigatório)
- `morada` (não obrigatório)

b. O atributo `action` do formulário deverá encaminhar para a rota `/processar-form-get-aluno` que será uma rota configurada no servidor para um pedido de `GET`. Essa rota deverá processar os dados recebidos do cliente e escrever na consola (deverá recorrer à *querystring* para obter os dados):

Exemplo de código:

```
app.get('/processar-form-get-aluno', function(req, res) {

   var nome = req.query.nome;

   //Obter os restantes valores submetidos no formulário
});
```

Output na consola:

```
Número: 123456789
Nome: Filipe Mariano
Morada: A minha morada
```

Nota: Se não for enviada nenhuma morada deverá aparecer na consola como `Morada: Não definida`.

3. Criar a página de `Formulário POST`, dentro da pasta public, cujo ficheiro será `form-post-aluno.html`.
a. Essa página deverá ter um formulário igual ao do exercício anterior.

b. O atributo `action` do formulário deverá encaminhar para a rota `/processar-form-post-aluno` que será uma rota configurada no servidor para um pedido de `POST`. Essa rota deverá processar os dados recebidos do cliente e escrever na consola (deverá recorrer ao *body* do pedido para obter os dados):

Exemplo de código:

```
//colocar a linha da rota: app.{verbo}(rota,callback)

   var nome = req.body.nome; //aqui acedo aos dados do body

   //Obter os restantes valores submetidos no formulário
```

Nota: Para que o exemplo de código anterior funcione, será necessário colocar o servidor a conseguir fazer o *parse* de pedidos de formulário (*urlenconded*).

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

var app = express();

app.use(express.urlencoded()); //Parse URL-encoded body (pedidos de formulários)
```

Output na consola:

```
Número: 123456789
Nome: Filipe Mariano
Morada: A minha morada
```

Nota: Se não for enviada nenhuma morada deverá aparecer na consola como `Morada: Não definida`.

4. Criar uma página de erro, chamada `erro.html` e que tenha uma imagem indicativa de que a página não existe, que será mostrada quando o cliente pedir um caminho que não existe uma rota definida (ver secção 6.2.). Esta rota deverá ser a última rota a ser colocada antes do a chamada ao *listen* para iniciar o servidor.

5. Alterar as rotas definidas para processar os formulários de modo a ficarem separadas num módulo chamado `rotas-forms.js`, numa nova pasta `routes`. Inclua esse módulo de rotas no ficheiro de servidor principal `index.js` (ver secção 6.1.).

6. Crie uma função de *middleware* através do módulo Express que sirva de `contador` para todas as rotas que tenham sido pedidas pelo cliente (ver secção 9.):
a. Deverá guardar o `url` e o `número de pedidos` feito a cada rota num array `contador` declarado globalmente.
b. A cada nova rota pedida, deverá escrever na consola qual a rota que foi pedida e todas as rotas já visitadas e respetivas contagens (fazer console.log do array contador).

7. Crie uma função de *middleware* através do módulo Express que sirva para guardar num ficheiro `requests.txt` a `data`, o `ip` e o `url` dos pedidos que chegam à vossa aplicação. Integre essa função de *middleware* no módulo de rotas `api.js`.