UPT UNIVERSIDADE PORTUCALENSE

CONTEÚDO

1. Contexto
2. Express
3. Principais características
4. Middleware
5. Middleware: utilização
6. Rotas
7. Routing:
    1. Exemplos
        1. ‘Ola mundo’
        2. Duas rotas
        3. Sessão prévia em base de dados
    2. Módulo router
    3. Sequencias de carateres (expressões regulares)
8. Métodos de resposta
9. Ficheiros estáticos
10. Bases de dados
    1. Módulo Sequelize
    2. Sequelize com MySQL
    3. MySQL com Node.js

Contexto

- O **Express**, juntamente com o **Node.js**, dão ao **JavaScript** funcionalidade de *back-end*
    - Permitem usar **JavaScript** para a criação de software *back-end*
    - Permitem construir uma aplicação completamente baseada em **JavaScript**

- Aplicações desenvolvidas do lado do *back-end* com recurso a **JavaScript**
    - A seguir, aplicações publicadas com o **Express**

- **Node.js** não foi criado para desenvolver aplicações
    - **Express** cria uma camada na estrutura interna
    - **Express** publica as funções necessárias para construir a aplicação

Express

- **Framework do lado do servidor e aplicação mobile**
  - Permite criar aplicações de uma página, várias páginas, híbridas, mobile e Web
  - Permite desenvolver funcionalidades de *back-end* para aplicações Web e API
- Usa linguagem **JavaScript**
- **Templating**
  - Contém dois motores de modelos (Jade e EJS) que facilitam o fluxo de dados e permitem usar outros modelos
- Suporta arquitetura *Model-View-Controller* (**MVC**)
- Usa o **Node.js**
- É **multiplataforma** (não é limitado ao sistema operativo)
- **Geradores de código** Express permitem criar rapidamente aplicações complexas

Principais características

- Código minimalista
- *Routing* robusto
  - *Routing*: função de decidir o que fazer quando chega um determinado pedido
- Facilmente integrável com os principais motores de templates
- Trabalha com o conceito de *middleware*
  - *Middleware*: “camada” intermédia capaz de mediar diferentes tecnologias
- Focado em alta performance
- Adota padrões e boas práticas de serviços *Representational State Transfer* (**REST**)
- Permite *content negotiation* (**RESTFul**)
  - *Content negotiation*: mecanismo HTTP que permite aceder a várias versões de um documento a partir do mesmo *Uniform Resource Identifier* (**URI**)

Middleware

- Representado por um conjunto de funções invocadas pela camada de *routing* do Express antes de serem manipuladas
- Colocado entre um pedido inicial e a rota pretendida
- Funções
  - Definidas como *middleware stack* (pilha de *middleware*)
  - Invocadas pela ordem a que são adicionadas (*First In First Out* – **FIFO**)
- Funciona como um filtro dos pedidos efetuados
  - Ao passarem pelo *middleware*, os pedidos podem ser modificados antes de serem entregues ao processo seguinte
- Torna possível criação de aplicações mais fáceis de manter e com menos código

Middleware: utilização

- Usa-se:
  - `app.use()` [chamado para cada método HTTP]
  - `app.METODO()` [METODO = GET, POST, PUT]

Ao definir a variável `app` sem `var` ou definindo-a como `global.app`, passa a ser considerada uma variável global e pode ser acedida em qualquer ficheiro externo (dentro do projeto)

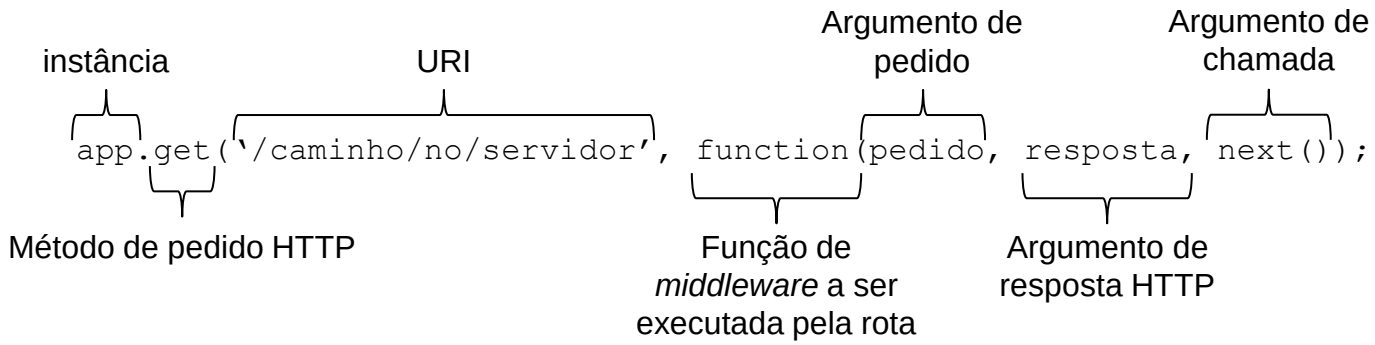
```
// Exemplo:
var app = express();

app.use(function (request, response, next) {
  console.log('Time:', Date.now());
  next();
});
```

- *Middleware* processado após receber a ordem enviada pela manipulação da rota. Potenciais problemas:
  - GET executado antes do middleware, POST depois
  - Se a aplicação não atualiza, o código não está a chamar o `next()`
  - `request` e `response` correspondem à mesma instância para todos os *middlewares* e rotas
    - Há dois *middlewares* que modificam as propriedades do objeto de forma diferente
      - Pode causar erros na aplicação
        - É importante estar atento às alterações feitas pelo *middleware*

Rotas

- Função: decidir o que fazer quando um pedido chega
- Estrutura: `app.METODO(CAMINHO, FUNÇÂO)`
  - app: instância do Express
  - METODO: método HTTP (GET, POST, PUT ou DELETE)
  - CAMINHO: caminho no servidor
  - FUNÇÂO: função executada quando a rota é correspondida
    - O next tem de ser explicitamente chamado para o *middleware* saber qual é a operação seguinte

- Roteamento (*routing*): definição de terminais da aplicação (URI) e forma como respondem aos pedidos. Estrutura:



Mais sobre routing: http://expressjs.com/en/guide/routing.html

Routing: exemplo ‘Ola mundo’

```
const express = require('express');
const app = express();
app.get('/', function(request, response){
   response.send('Ola mundo');
});
```

- Exemplo de rota get da raiz (/) que responde com a mensagem “Ola mundo”

Routing: exemplo duas rotas

```
const express = require('express');
const app = express();
app.get('/', function(request, response, next){
   response.send('Vou para a raiz');
});
app.get('/help', function(request, response, next){
   response.send('Vou para a ajuda');
});
```

- O *middleware* chama o next fornecido como parâmetro

- Duas rotas:
  - /: mostra 'Vou para a raiz'
  - /help: mostra 'Vou para a ajuda'

Routing: exemplo sessão prévia em base de dados

```
const express = require('express');
const app = express();
app.use(function(request,response,next){
   db.load(function(err,session){
      if(err){
         return next(err);
      } else if(!session){
         return next(new Error('Sem sessão');
      }
      request.session = session;
      next();
   });
});
app.get('/',function(request,response,next){…});
```

- Pretende-se carregar uma sessão de base de dados antes de processar qualquer pedido
- Com app.use obriga-se a chamar db.load() antes de qualquer pedido
- Chamada ao next é feita à base de dados:
  - Se erro, passa para o next
  - Se sem sessão, cria-se um erro
  - Só depois de verificar tudo é que se chama o next
    - e depois é chamada a rota de '/'

Routing: módulo router

```
//pagina.js
const express = require('express');
const router = express.Router();
router.use(function timeLog(req,res,next){
   console.log('Time: ',Date.now());
   next();
});
router.get('/', function(req,res){
   res.send('Raiz');
});
router.get('/about', function(req,res){
   res.send('Sobre');
});
module.exports=router;
```

```
//servidor.js
const pag = require('./pagina');
app.use('/pagina',pagina);
```

- Simplificar a gestão de rotas
- Cria manipuladores de rota
- Pode ser exportado e usado
- Código:
  - pagina.js
    - cria um router como um módulo
    - carrega uma função de *middleware* (timeLog)
    - define duas rotas ('/' e '/sobre')
    - exporta o router criado
  - servidor.js
    - carrega o ficheiro pagina.js
    - obriga a sua utilização no servidor.js

Routing: sequencias de carateres (expressões regulares)

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

/acd, /abcd

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

/abcd, /abbcd, /abbbcd, …

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

/abcd, /abacd, /abxcd, /abQUALQUERcd, /ab453cd, …

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

/abe, /abcde

Métodos de resposta

- Os métodos do objeto de resposta (assumir: res) enviam a resposta ao cliente e finalizam o ciclo pedido-resposta
  - Se nenhum for chamado, o pedido é deixado em suspenso

Método	Descrição
`res.download()`	Solicita que seja efetuado o download de um ficheiro
`res.end()`	Termina o processo de resposta
`res.json()`	Envia uma resposta JSON
`res.jsonp()`	Envia uma resposta JSON com suporte a JSONP
`res.redirect()`	Redireciona um pedido
`res.render()`	Faz *render* um modelo de visualização
`res.send()`	Envia uma resposta (vários tipos)
`res.sendFile()`	Envia um ficheiro
`res.sendStatus()`	Configura o código do estado de resposta e envia a sua representação numa sequência de carateres

Ficheiros estáticos

- O pré-processador de HTML do Express (Jade) não permite a utilização das tags básicas HTML com < e >
- Para evitar a aprendizagem de uma nova linguagem, pode utilizar-se ficheiros estáticos
- Basta indicar o nome da pasta onde se encontram os ficheiros estáticos

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

Vincula o *middleware* à aplicação usando a variável global `__dirname` que contem o caminho para a pasta

```
app.use('/public',express.static('assets');
```

Permite entregar ficheiros a uma pasta chamada assets a partir da rota /public

```
app.use('/template',
     global.express.static('views/template'));
```

A rota /template consegue aceder aos ficheiros em views/template

```
app.get('/template/index', function(req,res){
   app.use(express.static('views');
   res.sendFile(__dirname+'/views/template' +
     'index.html');
```

Cria uma rota /template/index e todos os ficheiros acedidos nessa rota usam a pasta views.

Módulo Sequelize

- **Sequelize:** *framework* de mapeamento objeto-relacional (*Object-relational mapping* – **ORM**) preparada para o Node.js
- Suporta PostgreSQL, MySQL, MariaDB, SQLite, MSSQL
- Permite realizar todas operações de SQL.
- Passos:
    1. Instalar os módulos `sequelize` e `mysql`
    2. Configurar uma ligação
    3. Criar uma instância para a BD
- Mais informação: https://sequelize.org/v5/

Sequelize com MySQL

- Configuração da ligação

```
const sequelize  = new Sequelize('nomeDB','utilizador','pass', {
   host: 'localhost',
   dialect: 'mysql',
   pool : {
      max: 5,
      min: 0,
      idle: 1000
},});
```

ou

```
const sequelize = new sequelize('mysql://utilizador:pass@host/nomeBD');
```

Pessoa

- cod
- nome
- ativo

- Utilização

```
Pessoa.findOne().then(function(p){
   console.log(p.get('nome'));
});
```

Permite encontrar um resultado na tabela e devolve o seu nome

```
Pessoa: findAll({where: {cod: 2}});
```

```
SELECT * FROM Pessoa WHERE cod=2
```

```
Pessoa: findAll({where: {cod: 2, ativo: 1}});
```

```
SELECT * FROM Pessoa WHERE cod=2 and ativo=1
```

```
Pessoa: destroy({where: {cod: 1}});
```

```
DELETE FROM Pessoa WHERE cod=1
```

```
Pessoa: update({nome: 'Manuel'}, {where: {cod: 2}});
```

```
UPDATE Pessoa SET nome= 'Manuel' WHERE cod=2
```

MySQL com Node.js

- Com recurso a objetos JSON

- Passos:
  1. Instalar o módulo `mysql`
  2. Configurar uma ligação (criação de ficheiro próprio)
  3. Chamar a ligação à BD no script

```
// (Passo 2) connect.js
const mysql = require('mysql');
module.exports = {
   con: mysql.createConnection({
      host: 'localhost',
      user: 'utilizador',
      password: 'pass',
      database: 'nomeBD'
   });
}
```

```
// (Passo 3) script.js
const ligacao = require('./connect');
ligacao.con.query("SELECT * FROM Pessoa", function(err,rows,fields){
   if(!err){
      for(x=0; x < rows.length; x++){
         console.log("Pessoa -> Cod: " + rows[x].cod + ", Nome: " + rows[x].nome);
      }
   }
});
```

Pessoa

- cod
- nome
- ativo

Mais sobre MySQL com Node.js: https://www.w3schools.com/nodejs/nodejs_mysql.asp

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.