UPT UNIVERSIDADE PORTUCALENSE

CONTENT

1. Context
2. Express
3. Main characteristics
4. Middleware
5. Middleware: how to use
6. Routes
7. Routing:
    1. Examples
        1. ‘Ola mundo’
        2. Two routes
        3. Previous database session
    2. Router module
    3. Character sequence (regular expressions)
8. Reply methods
9. Static files
10. Databases
    1. Sequelize module
    2. Sequelize with MySQL
    3. MySQL with Node.js

Context

- **Express**, together with **Node.js**, allow **JavaScript** to be used on the *back-end*
    - Allow using **JavaScript** to create *back-end* software
    - Allow the development of an application completely based on **JavaScript**

- The applications are developed, on the backend with **JavaScript**
    - Then, the applications are published with **Express**

- **Node.js** was not created to develop applications
    - **Express** creates a layer on the internal structure
    - **Express** publishes the functions needed to build the application

Express

- **Server side and mobile app framework**
  - Allows creating mobile and web applications with one or several pages
  - Allows the development of *back-end* functionalities for web applications and API
- Uses **JavaScript**
- **Templating**
  - Includes two module motors (Jade e EJS) that make the data flow easier and allow using other models
- Supports *Model-View-Controller* (**MVC**) architecture
- Uses **Node.js**
- **Multiplatform** (not limited to the operating system)
- Express **code generators** allow the fast creation of complex applications

Main characteristics

- Minimalistic code
- Robust *Routing*
  - *Routing*: decide what to do when a request arrives
- Easily integrable on the top template motors
- Works with the *middleware* concept
  - *Middleware*: intermediate “layer” capable of mediation between different technologies
- Focus on high performance
- Uses *Representational State Transfer* (**REST**) patterns and best practices
- Allows *content negotiation* (**RESTFul**)
  - *Content negotiation*: HTTP mechanism that allows accessing several versions of the same document from the same *Uniform Resource Identifier* (**URI**)

Middleware

- A set of functions called by Express' routing layer
- Put between the initial period and the route needed
- Functions
  - Defined as *middleware stack*
  - Called by the order they are added in (*First In First Out* – **FIFO**)
- Work as a filter of the requests
  - By passing through the *middleware*, the request can be changed before being delivered to the following process
- Applications become easier to maintain, and with fewer code

Middleware: how to use

- Usage:
  - `app.use()` [called for each HTTP method]
  - `app.METHOD()` [METHOD = GET, POST, PUT]

By defining the `app` variable without `var` or defining it as `global.app`, it is now considered a global variable and may be accessed in any external file (inside the project)

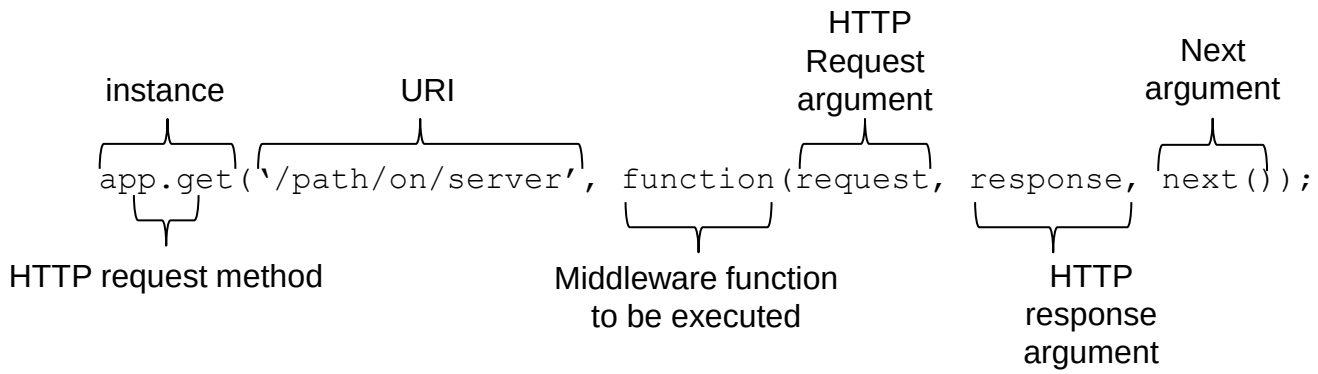
```
// Example:
var app = express();

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

- *Middleware* is processed after receiving the order sent by the routing. Possible problems:
  - GET is executed before the middleware, POST is executed after
  - If the application is not updated, the code isn't invoquing `next()`
  - `request` and `response` are the same instance for all the *middlewares* and routes
    - Two *middlewares* modify the object's properties in different ways
      - Might cause errors on the application
        - Be aware of the changes made by the *middleware*

Routes

- Objective: decide what to do when a request arrives
- Structure: `app.METHOD(PATH, FUNCTION)`
  - app: Express instance
  - METHOD: HTTP method (GET, POST, PUT or DELETE)
  - PATH: path on the server
  - FUNCTION: function to execute when the route is followed
    - next needs to be explicitly invoqued, so that the middleware knows what is the next operation

- *Routing*: definition of URIs and how they respond to requests. Structure:



More information on routing: http://expressjs.com/en/guide/routing.html

Routing: example ‘Ola mundo’

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

- Example of root (/) route that responds with the message “Ola mundo”

Routing: example two routes

```
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');
});
```

- *Middleware* calls the next given as parameter

- Two routes:
  - /: shows 'Vou para a raiz'
  - /help: shows 'Vou para a ajuda'

Routing: example previous database session

```
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){…});
```

- We want to load a database session before processin any request
- With app.use we make it mandatory to call db.load() before any request
- Call to next is made to the database:
  - If error, goes to next
  - If no session, creates an error
  - Next is called only after all validations
    - And then, the route '/' is called

Routing: router module

```
//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);
```

- Simplify the route management
- Creates route handler
- Can be exported and used
- Code:
  - pagina.js
    - Creates a router as a module
    - Loads a middleware function (timeLog)
    - Defines two routes ('/' e '/sobre')
    - Exports the router created
  - servidor.js
    - Loads the file pagina.js
    - Forces its usage on servidor.js

Routing: character sequences (regular expressions)

```
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

Reply methods

- The response (res) object‘s methods send the response to the client and end the request-response cycle
  - If none is called, the request is left suspended

Método	Descrição
`res.download()`	Requests to download a file
`res.end()`	Ends the response process
`res.json()`	Sends a JSON response
`res.jsonp()`	Sends a JSON response with support for JSONP
`res.redirect()`	Redirects a request
`res.render()`	*Renders* a view model
`res.send()`	Sends a response (several types)
`res.sendFile()`	Sends a file
`res.sendStatus()`	Configures the response’s status code and sends its representation in a character sequence

Static files

- Express' HTML pre-processor (Jade) does not allow HTML basic tags (wit < and >)
- To avoid learning a new language, we can use static files
- Just refer to the name of the folder that contains static files

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

Links the *middleware* to the application by using the global variable `__dirname` containing the path to the folder

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

Allows delivering files to a folder called assets from the /public route

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

The route /template can access the files in views/template

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

Creates the /template/index route and all the files accessed by that route use the folder views.

Sequelize module

- **Sequelize:** *Object-relational mapping* (**ORM**) framework ready for Node.js
- Supports PostgreSQL, MySQL, MariaDB, SQLite, MSSQL
- Allows all SQL operations.
- Steps:
    1. Install the following modules `sequelize` and `mysql`
    2. Configure a connection
    3. Create an instance for the database
- More information: https://sequelize.org/v5/

Sequelize with MySQL

- Configuring the connection:

```
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

- Usage:

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

Allows finding one result on the database and returns its '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

- Using JSON objects

- Steps:
  1. Install module `mysql`
  2. Configure a connection (creation of a dedicated file)
  3. Call the database connection on the script

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

```
// (Step 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

More about MySQL with Node.js: https://www.w3schools.com/nodejs/nodejs_mysql.asp

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.