REST Introduction

This is an introductory document on how to program RESTful web services with node.js.

Table of Contents

1. Requirements

Node.js is a multiplatform, server-side JavaScript programming language. You can install it to your own computer through www.nodejs.org or by creating an account to a web-based developer environment, for example at www.c9.io

When in doubt, ALWAYS READ THE DOCUMENTATION!

Documentation relevant to this document:

- Node.js documentation: http://nodejs.org/documentation/
- Express.js documentation: http://expressjs.com/4x/api.html

2. Setting up the environment

Node.js uses the powerful **npm** tool to install 3rd party addons. For this example, we are using yeoman generator to create an expressjs-based starting point (more on express-generator here: https://github.com/petecoop/generator-express)

First, start by installing the yeoman generator, by typing into the command line

>npm install -g yo

>npm install -g generator-express

Once done, run the following command

>yo express

Answer to all the questions to create an application you want. However, within this project, we are using the following settings:

- Version to install: **basic**
- View engine to use: **EJS**
- css preprocessor: **None**
- Build tool to use: **Gulp** (this one really doesn't matter here)
- overwrite package.json: **Y**

Once done, run the installer for npm

>npm install

To test that everything works, start the server with node

>node bin/www

Now the server should be running in localhost:3000 (or something else, for example https://projectname-username.c9.io/ if using Cloud9)

3. Programming

Okay, we have finally installed everything and the project is ready for modifications :-D Lets go!

3.1 GET test page

Start by creating a test.js file to /routes/ folder

test.js

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

/* GET home page. */
router.get('/', function(req, res) {
  res.send("Test page");
});

module.exports = router;
```

Then add the newly created route to app.js

app.js (only showing edits, the definition goes to top and the route usage after other routes)

(row 10) var testRoute = require('./routes/test');

(row 30) app.use('/test', testRoute);

Now we can run the program, and navigate to <url>/test/ and see our page in use!

>node bin/www

3.2 POST test page

Lets create a POST page that does something!

3.2.1 Adding POST route to routes/test.js

Start by editing the test.js

test.js (only showing edits)

```
/* POST test page */

router.post('/', function(req, res) {
   // Check that there is a body
        if(!req.body) {
        //No body found, send an error to the client
        res.status(500).send({error : "No body found!"});
        // And return, so nothing beyong this gets executed
        return;
    }

  // POST messages SHOULD ALWAYS contain JSON in their body
  var json = req.body;

  // Just echo the results back to client
  res.send(json);

});
```

Job done!

All we need to do now, is to create a way to actually POST something to the server!

3.2.2 Create a new view at views/test.ejs

Create a new view file to views/ and name it as test.ejs (we are using EJS as the rendering engine)

views/test.ejs

```
<% include header %>

<form action="/test/" method="POST">
First name:<br>
<input type="text" name="firstname" value="Mickey">

<br>

Last name:<br>
<input type="text" name="lastname" value="Mouse">
<br><br>
<input type="submit" value="Submit">
</form>

<% include footer %>
```

Now we have a proper view page.

Now we only need to serve this POST page somehow.

3.2.3 Add method for rendering the page for showing the POST form

Lets edit the routes/test.js just a little bit more

test.js (only showing edits)

```
/* GET Test Create Page */

router.get('/create/', function(req, res) {
  // render the test page, and pass Test POST page as the title
  res.render('test', { title: 'Test POST Page' });
});
```

Now we have added a new route, called /test/create/ that shows our view.

3.2.4 Testing the page

Now we can finally test our newly created thing

Navigate to

<url>/test/create

Add some data and press submit.

This will do a POST and get results from the server-side

4. Some Exercises

Here are some exercises for testing creation of web services with REST

4.1 Using node.js :params to Find prices of cars

This exercise teaches to use req.params, with an example of finding prices of cars by using a GET request.

4.1.1 Short explanation on req.params

In node, you can read the messages sent by the client with *req.params* command. This means, for example:

```
router.get('/test/:userId'/, function(req, res) {
        // You can read anything stated as :<paramName> with req.params.<paramName>
        var userId = req.params.userId
}
```

So, making a GET request at www.myservice.com/test/jack

This should make userId in the application to be jack.

4.1.2 The actual task

Create (or extend this one to contain) a REST service that returns a price of a car, depending on the asked car type.

Basically, the client should be able to get the price of the car with query /cars/:type

e.g. www.myservice.com/cars/mercedes
should result in server replying something like “The price of a Mercedes Benz is 70 000 €”

You can use the following JSON to parse for car-price pairs

```
cars = [
{
        name : “mercedes”,
        price : 70000
},
{
        name : “bmw”,
        price : 55000
},
{
        name : “audi”,
        price : 38000
},
{
        name : “volvo”,
        price  : 42000
},
{
        name : “fiat”,
        price : 18000
},
{
        name : “lada”,
        price : 5000
}
]
```

4.2 Getting familiar with all CRUD parts

Github contains an example project on how to build a little bit bigger REST service with Node.js. You should familiarize yourself with this program code, as it will probably help you a lot on writing the final assignments.

Example code @ GithHub: https://github.com/Japskua/lut-rest-lecture