

AMIS SIG Introduction Node.js-Hands-on

31st March 2016

In this hands-on, you will get going with Node.js. You will walk through some of the basics – running your first node application (hello world!), working with the file system and other core modules and with some packages downloaded through npm.

Then we step it up a little with some practices around http. You will create a simple static file server as well as some programs that handle more interesting http requests, similar to Java servlets. We look at making http call outs from the Node.js program in addition to listening and responding to incoming calls.

Then we introduce the Express framework – the more or less de facto standard web application framework for Node.js applications that serve UI or API oriented applications. We will look at the implementation of simple REST API.

One example of a rich client web application that can be served from Node.js is the Oracle JET application that you will work on a little. Note that similar applications, for example based on Angular2 or other frameworks, can run from Node.js just as easily.

Somewhat more advanced topics are suggested as bonus practices, looking at new language features such as promises and generators and mechanisms for parallel processing (the latter applied in an artist api that provides enriched artist information by leveraging APIs from Spotify and Echonest). We also discuss the Node Oracle DB Database Driver for Node.js; this driver allows us to access an Oracle Database from a Node.js program, similar to the way JDBC allows us to access a database from Java applications.

You can get access to the sources for the practices from the GitHub repository:

<https://github.com/lucasjellema/sig-nodejs-amis-2016> .

1. Installation of Node.js

Native versions of Node.js are available for Mac OS X, Windows and Linux. Go to <https://nodejs.org/en/download/> and download the installer for your platform.



Downloads

Current version: v4.4.1

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	 Windows Installer node-v4.4.1-x86.msi	 Macintosh Installer node-v4.4.1.pkg	 Source Code node-v4.4.1.tar.gz
Stable Latest Features			
Windows Installer (.msi)	32-bit	64-bit	
Windows Binary (.exe)	32-bit	64-bit	
Mac OS X Installer (.pkg)		64-bit	
Mac OS X Binaries (.tar.gz)		64-bit	
Linux Binaries (.tar.xz)	32-bit	64-bit	
Source Code			node-v4.4.1.tar.gz

Next, run the installer to perform the installation. You can check on the success of the installation by opening a command line window or terminal and typing node -v. This should result in an indication of your current version of Node.js:

```
C:\>node -v  
v4.4.1
```

Alternatively you can also work in a Docker container. The official Docker image for Node.js is on https://hub.docker.com/_/node/. This page also contains instructions on how to build a container with Node.js set up inside, and run it.

A third way to go is to use a prebuilt Virtual Machine image. One source for such images is Bitnami: <https://bitnami.com/stack/nodejs/virtual-machine> (and this article explaining how to load the VM Ware images into Virtual Box:

https://wiki.bitnami.com/Virtual_Appliances_Quick_Start_Guide#How_to_start_your_Bitnami_Virtual_Application_3f).

A fourth option of working with Node.js is in a Cloud environment, such as Google App Engine, Azure, Heroku, AWS and Oracle Application Container Cloud. Given the light weight nature of node.js you will

be able to use the “free tier” resources in most cases. We will not explore these options in this workshop.

2. Basic steps with Node.js

A Node.js application is a JavaScript application (more formally: ECMA Script v6 or ES6) that can be run – outside the browser – on the Node.js platform [which contains the V8 engine for JavaScript].

A node application – the informal way of referring to Node.js applications – is simply run on the command line using the command:

node app.js

where app.js is the presumed name of the (main) application script. The name is yours to decide. Other common names include main.js and of course hello-world.js.

3. Handling HTTP Requests

One of the

4. Express Web Application Framework

A Node.js application is a JavaScript application (more formally: ECMA Script v6 or ES6) that can be run -

5. Run Oracle JET application of Node.js

Rich client web applications do their work in the client – the web browser. The role of the middle tier is reduced for these applications to exposing APIs (REST/JSON) that the rich client application can make use of. The middle tier can also add what is sometimes called mBaaS – Mobile Backend as a Service – to handle various features that mobile apps benefit from (device management, proxying, analytics, caching, push notification, authentication,..). Some of these mechanisms may be useful for ‘regular’ [rich client] web applications to.

One middle tier role is indispensable for now: serving the [static] rich client application’s resources – the HTML, JS, CSS and image files, at least the first time the application is started in a browser. Note that static file serving is a very trivial affair with Node.js. It scales well, serves files well and is easy to configure. It can do what Nginx can do – and more. And this *more* is why Node.js is often used as the web server (and REST API provider) for rich client web applications.

In this section we will quickly create an Oracle JET application – just as an example of a rich client web application - and run it on Node.js. Next, we will add a data bound section to the application and hook it up with a REST API that we inject into the Node.js backend for the application.

- a) Create a new directory part4. Navigate to that directory in the command line [terminal]. Install express and express-generator using npm:

```
npm install express  
npm install express-generator -g
```

- b) Generate a new express application called jet-on-node:

```
express jet-on-node
```

A new application scaffold is generated for *jet-on-node*. This has nothing yet to do with Oracle JET – you can do this for any Express application!

- c) Navigate to directory jet-on-node. Add dependencies to the application using:

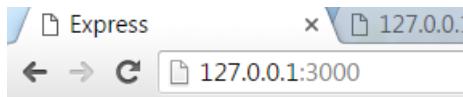
```
npm install
```

This installs a number of node modules. You can check directory part4\jet-on-node\node-modules to get an impression.

- d) At this point we have a runnable Express application. No Oracle JET elements have been added yet; we will do that shortly. Let’s try out the application quickly. From directory part4\jet-on-node, execute:

```
SET DEBUG=jet-on-node:* & npm start
```

Now access the application at <http://127.0.0.1:3000/>.



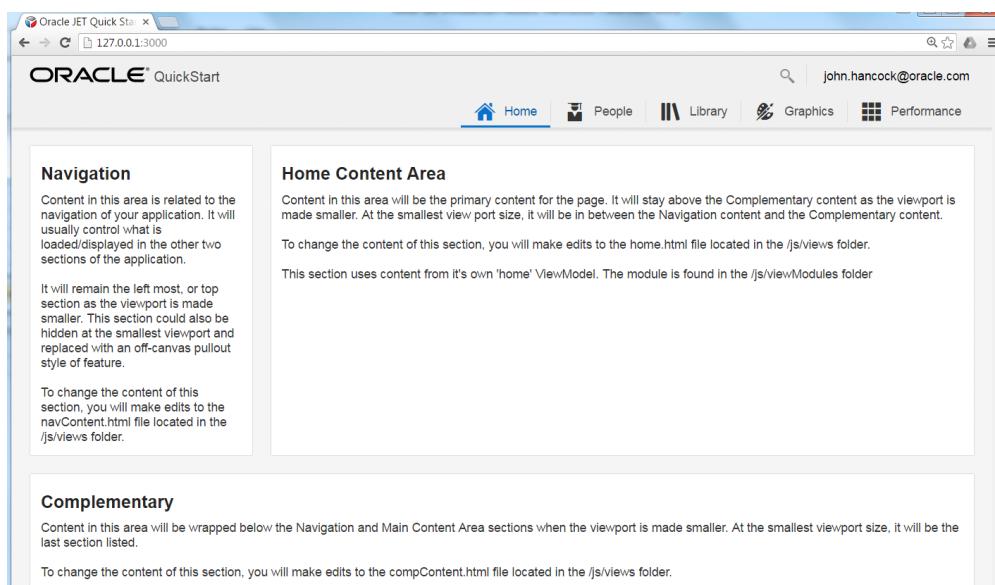
Express

Welcome to Express

- e) We will now add Oracle JET – using the quickstart application. Go to <http://www.oracle.com/technetwork/developer-tools/jet/downloads/index.html> and download [Quickstart: Basic Starter Template with Oracle JavaScript Extension Toolkit Pre-configured](http://download.oracle.com/otn/JET/20/OracleJET_QuickStartBasic.zip) (http://download.oracle.com/otn/JET/20/OracleJET_QuickStartBasic.zip).

Extract all files from this zip-file to directory part4\jet-on-node\public.

Refresh the browser. You will now see the Oracle JET quick start application:



- f) Feel free to browse the Oracle JET application a little. The pivotal file – the starting point for the application – is index.html in directory part4\jet-on-node\public. That file is served to the browser when the request comes in to this application at port 3000.

- g) Our next move is to add a data bound component to the JET application – and make it retrieve data from a REST API that we also need to add.

The steps you need to go through:

- copy or move hrm.js from the workshop folder part4-JET to jet-on-node\public\js\viewModels
- copy or move hrm.html from the workshop folder part4-JET to jet-on-node\public\js\views
- copy or move departments.json from the workshop folder part4-JET to jet-on-node
- edit file main.js in jet-on-node\public\js to make it look like main.js in workshop folder part4-JET (see instructions below)
- edit file app.js in jet-on-node to make it look like app.js in workshop folder part4-JET (see instructions below)

ik (C:) ▶ data ▶ sig-nodejs-31march2016 ▶ sig-nodejs-amis-2016 ▶ part4-JET ▶			
Share with ▾	New folder		
Name	Date modified	Type	Size
oraclejetwithnodejs	28-3-2016 17:33	File folder	
app	28-3-2016 17:33	JScript Script File	2 KB
departments.json	28-3-2016 17:35	JSON File	2 KB
hrm	28-3-2016 17:12	Firefox HTML Doc...	1 KB
hrm	28-3-2016 17:34	JScript Script File	2 KB
main	28-3-2016 13:47	JScript Script File	5 KB

In main.js, you need two add two small sections of code, to add a new HRM tab and component. First, add:

```
'hrm': {label: 'HRM'},
```

in the definition of router.configure:

```
require(['ojs/ojcore', 'knockout', 'jquery', 'ojs/ojknockout', 'ojs/ojrouter',
        'ojs/ojmodule', 'ojs/ojofcanvas', 'ojs/ojnavigationlist', 'ojs/ojarraytabledatasource'],
function (oj, ko, $) { // this callback gets executed when all required modules are loaded
    var router = oj.Router.rootInstance;
    router.configure({
        'home': {label: 'Home', isDefault: true},
        'people': {label: 'People'},
        'hrm': {label: 'HRM'},
        'library': {label: 'Library'},
        'graphics': {label: 'Graphics'},
        'performance': {label: 'Performance'}
    });
});
```

then add

```
{name: 'HRM', id: 'hrm',
iconClass: 'demo-education-icon-24 demo-icon-font-24 oj-navigationlist-item-icon'},
```

in the definition of var navData:

```

function RootViewModel() {
  var self = this;
  self.router = router;

  // Shared navigation data and callbacks for nav bar (medium+ screens) and nav list (small screens)
  var navData = [
    {name: 'Home', id: 'home',
     iconClass: 'demo-home-icon-24 demo-icon-font-24 oj-navigationlist-item-icon'},
    {name: 'People', id: 'people',
     iconClass: 'demo-education-icon-24 demo-icon-font-24 oj-navigationlist-item-icon'},
    {name: 'HRM', id: 'hrm',
     iconClass: 'demo-education-icon-24 demo-icon-font-24 oj-navigationlist-item-icon'},
    {name: 'Library', id: 'library',
     iconClass: 'demo-library-icon-24 demo-icon-font-24 oj-navigationlist-item-icon'},
    {name: 'Graphics', id: 'graphics'},
  ];
}

```

Finally, in app.js – the Node.js application itself – we need to add support for the REST API, by adding these two lines, just under var app = express(); :

```

var departments = JSON.parse(require('fs').readFileSync('departments.json', 'utf8'));
app.get('/departments', function (req, res) { //process

  res.send( departments); //using send to stringify and set content-type

});

var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var routes = require('./routes/index');
var users = require('./routes/users');

var app = express();

var departments = JSON.parse(require('fs').readFileSync('departments.json', 'utf8'));
app.get('/departments', function (req, res) { //process
  res.send( departments); //using send to stringify and set content-type
});

// view engine setup
app.set('views', path.join(__dirname, 'views'));

```

h) Restart the node application – again with

SET DEBUG=jet-on-node:* & npm start

and reload the application in the browser. Navigate to the HRM tab. You will now see data in the table component. This is the data loaded from the departments.json file in the root folder of the application. You can verify this by changing the data in the file – and restarting the application (the data is cached when the application starts).

The screenshot shows a web browser window titled 'Oracle JET Quick Start' with the URL '127.0.0.1:3000/?root=hrm'. The page has a header with the 'ORACLE' logo and navigation links for Home, People, HRM (which is selected), Library, Graphics, and Performance. A search bar and user info 'john.hancock@oracle.com' are also present. The main content area is divided into two sections: 'Navigation' on the left and 'HRM Content' on the right.

Navigation

Content in this area is related to the navigation of your application. It will usually control what is loaded/displayed in the other two sections of the application. It will remain the left most, or top section as the viewport is made smaller. This section could also be hidden at the smallest viewport and replaced with an off-canvas pullout style of feature. To change the content of this section, you will make edits to the navContent.html file located in the /js/views folder.

HRM Content

The HRM Details

Department Id	Department Name	Location
10	Administration	Zoetermeer
20	Marketing	Zoetermeer
30	Purchasing	Zoetermeer
40	Human Capital	Zoetermeer
50	Shipping	Zoetermeer
60	IT	Zoetermeer
70	Public Relations	Zoetermeer
80	Sales	Zoetermeer
90	Executive	Zoetermeer

- i) Instead of using the local REST API, you can also link up the application to a remote API – also implemented using Node.js (on Oracle Application Container Cloud) and running live against a DBaaS instance. The URL is <https://data-api-lucasjellema.apaas.em2.oraclecloud.com/departments>. You can look in the file jet-on-node\public\js\viewModels\hrm.js and swap the local url for this remote endpoint. The data will be slightly different and the waiting will be somewhat longer. So apart from ‘just because we can’ there is not much reason for actually making this change.

6. Complex REST API for retrieving rich Artist information

The REST API that we discussed in the previous section was quite straightforward. Information retrieved from a static local file, doing a little manipulation. It was a fine start. In this section, we make it a little bit more interesting. We will work on an API that returns a JSON document in return to an HTTP GET request that specifies the name of a particular artist. This JSON document contains details on the artist – such as a genre label, a biography, a list of the most recently released albums with their details and more. To gather this information, the node application that exposes this API has to go out and fetch data from external services such as the Spotify API (<https://api.spotify.com/v1>) and the Echonest API (<http://developer.echonest.com/api/v4>).

The situation can be described like this:



We will build up this API in a number of steps. You will find the resources in folder part5-artist-api.

- Open the command line window in folder part5-artist-api. Before the application can be started, you need to use npm to install a few packages: express, request, body-parser and async.
- Run the artist-enricher-api-1.js program. It listens to HTTP requests of the form <http://127.0.0.1:5100/artists/get?artist=artistName> (such as <http://127.0.0.1:5100/artists/get?artist=b52s> or <http://127.0.0.1:5100/artists/get?artist=u2>). Try out such a call from your browser or using CURL.
- Inspect the code in artist-enricher-api-1.js. The familiar Express style configuration of a web server to handle GET requests can be found. The configuration of a callback function for the URL path /artists/* that hands off the work to function handleArtists() while passing the value of the artist query parameter. And that function itself. Check how the Spotify API is invoked using the request() function and how the response is processed.
- Stop artist-enricher-api-1.js and run artist-enricher-api-2.js. Make the same HTTP GET call as before. You will now see – after a somewhat longer waiting time – additional information in the response: the artist's biography is included. This information comes from the echonest API.

Inspect the code in artist-enricher-api-2.js to learn about this second API call.

One thing you could notice is that the code slightly becomes harder to read – additional indentation for each consecutive outbound call and callback function to handle the response. Also: the second call (to echonest) does not make use of the outcome from the first call. Yet these two calls are made sequentially, meaning that we spend waiting up to twice as long as should necessary.

- e) Package node-async (<https://github.com/caolan/async> and installable with npm install async) is a utility module which provides straight-forward, powerful functions for working with asynchronous JavaScript. It can be used in node.js applications as well as in client side JavaScript running in a web browser. It makes it quite easy for example to perform multiple asynchronous activities in parallel and work with the combined result from all these activities.

We will make use of async to perform the two outbound REST API calls in parallel rather than sequentially. Check out the documentation for `async.parallel` at <https://github.com/caolan/async#parallel> and check the code in artist-enricher-api-3.js to see how this is applied in our Artist API.

Run artist-enricher-api-3.js and make one or more API calls from your browser, CURL, Postman, SoapUI or whatever your favorite tool is. Check if the overall response time is shorter than with artist-enricher-api-2.js and the sequential calls.

- f) At this point, we do not get any details for the albums released by our artist. Spotify offers an API that provides a list of albums – maximum of 50 per call. For each album, we get the name and an image URL for the cover image. We do not get a release date.

Run artist-enricher-api-4.js to see the first iteration of album details in the response. Then inspect the code – the extension to the initial call to Spotify that was added. Notice how `async.waterfall` is used to organize the sequential calls to the Spotify API. The functionality is not spectacular – control of the program flow and readability of the code increases notably. Note how `callback()` is used to indicate the completion of *unit* in `async waterfall` as it is in `async parallel`. Also note how `callback(null, artist.spotifyId)` is used to specify that no error occurred (the first parameter is null) and also to pass a result from this unit to the next: `artist.spotifyId` is passed in as the first input argument to the next unit in the waterfall. (through the shared context variable `artist` this same value is accessible using `artist.spotifyId`).

- g) Spotify offers yet another API that allows us to retrieve details for albums – primarily the release date. In artist-enricher-api-5.js, you will find the waterfall extended with a third unit. This third unit calls the albums details API on Spotify. This can be done for up to 15 albums at one time.

Because the second call to Spotify in the waterfall results in a list of up to 50 albums, we need multiple calls to the album details API. Of course these calls should not be made sequentially. In the source code for artist-enricher-api-5.js you will see how a third async mechanism is used: `async.forEachOf`. The `forEachOf` function is handed an array – in this case an array of arrays. For each element in the array, `forEachOf` will execute the function. When all function calls for the elements in the array have signaled their completion – through the `callback()` call – the completion function in `forEachOf` is executed to do any final processing of the joint results. In this example, all that needs to be done is another call to `callback()` to tell `async.waterfall` that this unit is complete.

Run `artist-enricher-api-5.js`, make one or more calls and verify that the release date is now added for each album.

7. Node OracleDB Database Driver

Node applications may want to have access to databases – similar to access to the file system, external APIs over HTTP and other resources. Just like Java applications use a JDBC driver to access a relational database, Node.js applications can leverage a database driver for the database they want to access. There is no standard for database drivers or for interacting with a relational database. There is no JDBC-like API. Working with different databases is done in different ways.

Oracle has created an open source project – available on [GitHub](#) – called node-oracledb. This project provides a Node.js database driver to the Oracle Database. It leverages Oracle Database client libraries – the thick OCI API – that need to be installed on the server running the node application. Through node-oracledb, the interaction from a node application with an Oracle Database becomes fairly easy and straightforward – similar to JDBC but simpler. And asynchronous of course – using callbacks to handle the responses returned from the database. Performing queries, DDL and DML as well as PL/SQL calls is all supported pretty well as are native Oracle data types.

8. Advanced Topics

In this section, we will look at some advanced topics. These include:

- use of new language features: Promises and Generators
- working with XML and SOAP

Resources

Official documentation for Node.js: <https://nodejs.org/en/docs/>

Several useful how-to articles on Node.js: <https://docs.nodejitsu.com/>

Article that explains how to take a running Express web application and Dockerize it:

<https://nodejs.org/en/docs/guides/nodejs-docker-webapp/> .

GitHub Repository for Oracle JET : <https://github.com/oracle/oraclejet>

GitHub Repository for Node OracleDB (database driver): <https://github.com/oracle/node-oracledb>