MarkLogic UNIVERSITY

Unit 12

Introduction to Server Side JavaScript

Using Server Side JavaScript

Install a Server Side JavaScript Extension

Invoke a Server Side JavaScript Extension from Node.js

Exercise 1: Using Server Side JavaScript

In this exercise you will experiment with server side JavaScript functions using the Star Wars database you created earlier.

1. Open Query Console (**http://localhost:8000/qconsole**).
2. Import the workspace found at **c:\mls-developer-node\Unit12\Using JavaScript.xml**

   Click the "**Example 1**" tab to study the code, which is a simple example of reading a document from the database.
3. In Query Console, set the Content Source to the **star-wars-content** database:

4. Run the "**Example 1**" code in Query Console and study the response.
5. Click on the tab labeled "**Example 2**" and study the code.
6. Run the "**Example 2**" code against the **star-wars-content** database.
7. Click the "**Example 3**" tab. Study the code and then run the code against the **star-wars-content** database.
8. Click the "**Example 4**" tab. Study the code and then run the code against the **star-wars-content** database.
9. Click the "**Example 5**" tab. Study the comments. Try to write the code to satisfy the requirements yourself.
10. If you get stuck, you may see a working example on tab "**Example 6**".

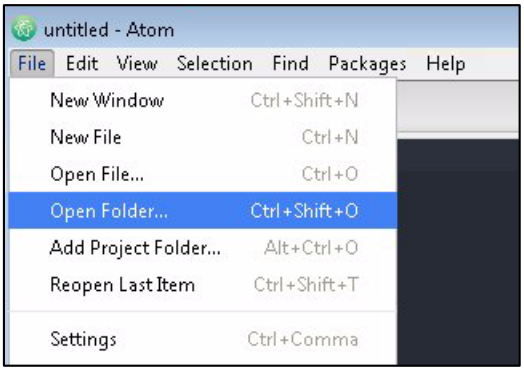
Exercise 2: Install a Server Side JavaScript Extension

In this exercise you will install a server side JavaScript extension to your projects modules database.

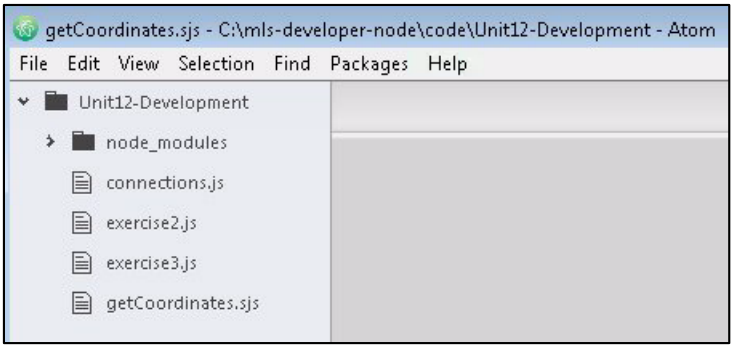
The purpose of this extension is to take some input from the Node.js application that represents a location, and use a web service to return the latitude and longitude of that location back to the Node.js application.

1. From the command line, navigate to the **c:\mls-developer-node\code\Unit12-Development** directory and run **npm install marklogic**.
2. Open the Atom editor from your Desktop:

3. In the Atom editor select File→Open Folder…:



4. Open the **Unit12-Development** folder from c**:\mls-developer-node\code**
5. You should see the following structure in place in your editor:



6. Click on **connections.js** and note that there are now connection details using the MarkLogic Administrator account. This is because the ability to invoke an extension requires the **xdmp-invoke** privilege. We don't need to be an admin to do this -- we could also have assigned this privilege to one of our existing roles:

```
mlAdmin: {
  host: 'localhost',
  port: 7010,
  user: 'admin',
  password: 'admin'
}
```

7. Click on **exercise2.js** and study the comments and code.

8. Note that we are reading a SJS file from the file system and then loading the file to the database. When **writing extensions**, you must be a user with the **rest-admin** role. The extensions will be written to the modules database of the REST instance defined in your database client:

```
"use strict";

var marklogic = require("marklogic");
var dbConn = require("./connections.js");
var fs = require("fs");

var dbAdmin = marklogic.createDatabaseClient(dbConn.restAdmin);

var path = "c:/mls-developer-node/code/Unit12-Development/";
var file = "getCoordinates.sjs";

var sjsModule = fs.readFile(path + file, "utf8", function (err, data) {
  if (err) {
    return console.log(err);
  }

  dbAdmin.config.extlibs.write({
    path: "getCoordinates.sjs",
    contentType: "application/vnd.marklogic-javascript",
    source: data
  }).result().then(function(response) {
    console.log("Installed module: " + response.path);
  });
});
```

9. Click on **getCoordinates.sjs** and study the comments and code.

10. Note the use of the MarkLogic HTTP functions which make it easy to work with external web services:

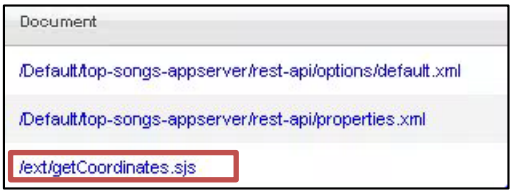
```
// This module takes an input and  build a request to the Google Maps API.
// It returns a response that will include latitude and longitude data.

var request = "http://maps.googleapis.com/maps/api/geocode/json?address=" + input;

var response = xdmp.httpGet(request);

response;
```

11. Now let's test the code.
12. At the command line, go to the **c:\mls-developer-node\Unit12-Development** directory.
13. Enter **node exercise2.js** and press enter.
14. You should see the following response:

```
c:\mls-developer-node\code\Unit12-Development>node exercise2.js
Installed module: getCoordinates.sjs
```

15. Now let's look at the modules database.
16. In your browser, open Query Console (http://localhost:8000/qconsole).
17. From the Content Source, select the **top-songs-modules** database and click Explore:

18. Note that you now have your SJS extension along with the default configurations that were created when you built your rest instance:



19. Click on the extension to view the code:

```
// This module takes an input and  build a request to the Google Maps API.
// It returns a response that will include latitude and longitude data.

var request = "http://maps.googleapis.com/maps/api/geocode/json?address=" + input;

var response = xdmp.httpGet(request);

response;
```

Exercise 3: Invoke a Server Side JavaScript Extension from Node.js

In this exercise you will invoke an extension from a Node.js application and work with the response.

1. In your editor, within the **Unit12-Development** project, select **exercise3.js**.
2. Study the comments and code.
3. Note that we are creating a database client using the MarkLogic Admin credentials, which will give us necessary privileges to invoke a module. Remember that you don't need to be a MarkLogic admin to invoke, but you do need to be a user with a role that has the **xdmp-invoke** privilege.
4. Also note that the SJS module that we are invoking is performing an HTTP GET request. This means you would also need the **xdmp-http-get** privilege. As an admin, you have all roles and privileges.
5. Note that when the module is invoked, we are passing it a variable called **input** that contains the **location** information defined in the application code:

```
"use strict";

var marklogic = require("marklogic");
var dbConn = require("./connections.js");

var mlAdmin = marklogic.createDatabaseClient(dbConn.mlAdmin);

var location = "San Francisco California";

mlAdmin.invoke({
  path: "/ext/getCoordinates.sjs",
  variables: { input: location }
}).result(function(response) {
  console.log(JSON.stringify(response[1], null, 2));
}, function(error) {
  console.log(JSON.stringify(error, null, 2));
});
```

6. Now let's test the code.
7. At the command line, go to the **c:\mls-developer-node\Unit12-Development** directory.
8. Enter **node exercise3.js** and press enter.
9. You should see a response that contains all the geospatial data about the location coming back from the Google Maps API: