Unit 1

MarkLogic in a Three Tiered Architecture

Launch Samplestack

Explore Samplestack

Explore Node.js Client API Resources

Create a Simple Node.js App

Create a Simple Front-End

Exercise 1: Launch Samplestack

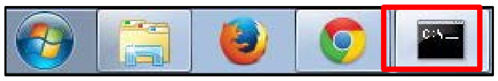
In this exercise we will launch Samplestack.

In order to focus our time on development tasks, Samplestack has already been deployed on your VM. The deployment uses Gulp and Gradle to automate the setup of MarkLogic, load sample data, and setup the middle tier. After this training is complete, if you wish to setup Samplestack on your own machine, you may follow the instructions found here:

https://github.com/marklogic/marklogic-samplestack

Later on in this course we will explore the automated deployment of Samplestack in more detail.

1. Open the command prompt, which has been placed on the Desktop of your VM and on the lower start menu navigation bar:



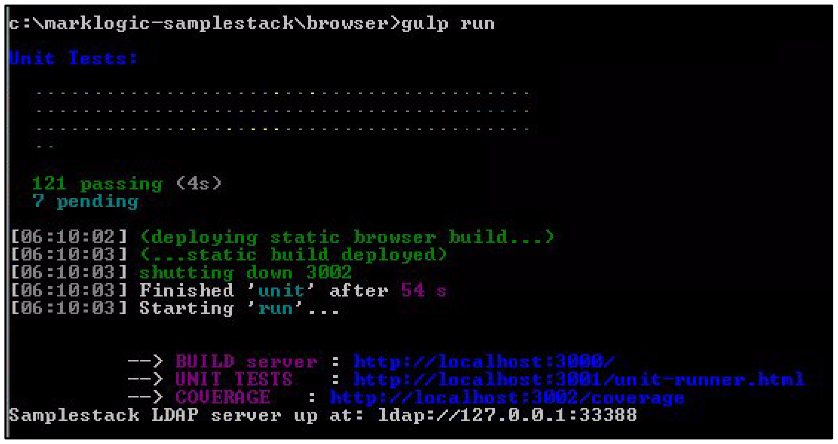
2. From the command prompt, navigate to the **c:\marklogic-samplestack\browser\** directory. That can be accomplished by keying the following command and pressing enter:

```
cd c:\marklogic-samplestack\browser
```

3. Next you will launch Samplestack. From the command line enter:

```
gulp run
```

4. Once the launch is complete, you should see the following response:



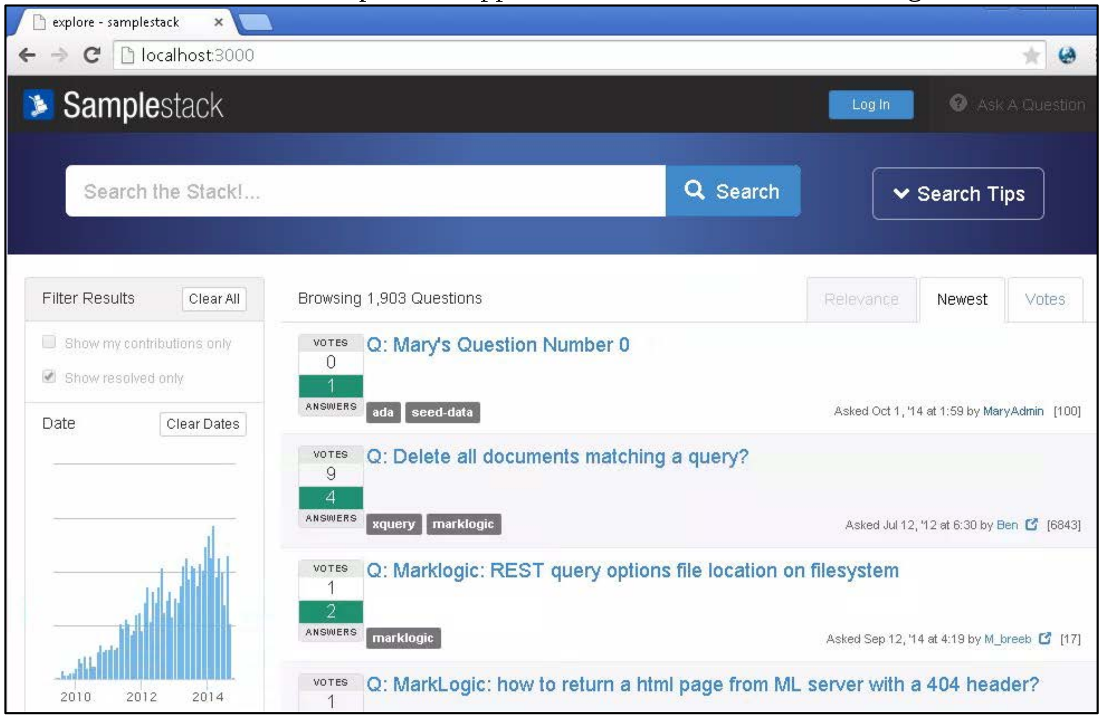
```
c:\marklogic-samplestack\browser>gulp run

Unit Tests:

  ..........................................
  ..........................................
  ..........................................
  ..

  121 passing (4s)
  7 pending

[06:10:02] (deploying static browser build...)
[06:10:03] (...static build deployed)
[06:10:03] shutting down 3002
[06:10:03] Finished 'unit' after 54 s
[06:10:03] Starting 'run'...


          --> BUILD server : http://localhost:3000/
          --> UNIT TESTS   : http://localhost:3001/unit-runner.html
          --> COVERAGE   : http://localhost:3002/coverage
Samplestack LDAP server up at: ldap://127.0.0.1:33388
```

5. To view the application, open your browser and go to **http://localhost:3000/**

Note:
Chrome is the preferred browser due to some its robust developer tools that are quite helpful when testing and debugging JavaScript.

Note:
Please keep the command window open, as closing it would shut down the Samplestack front end.

6. You should now see the Samplestack application as shown in the following screenshot:



Browsing 1,903 Questions

VOTES 0 1 ANSWERS Q: Mary's Question Number 0
ada seed-data
Asked Oct 1, '14 at 1:59 by MaryAdmin [100]

VOTES 9 4 ANSWERS Q: Delete all documents matching a query?
xquery marklogic
Asked Jul 12, '12 at 6:30 by Ben [6843]

VOTES 1 2 ANSWERS Q: Marklogic: REST query options file location on filesystem
marklogic
Asked Sep 12, '14 at 4:19 by M_breeb [17]

VOTES 1 Q: MarkLogic: how to return a html page from ML server with a 404 header?

Exercise 2: Explore Samplestack

This exercise documents the Samplestack demo that your Instructor performed in case you want to experiment with the application on your own or at a later date.

This demo is designed to cover Samplestack functionality from an end user point of view, so that you have a foundational understanding of certain important concepts that we will explore in greater technical detail throughout this training course. These topics include:

1. Samplestack data model
2. Full text search
3. User accounts, security, restricted content
4. Facets
5. Ask, answer, comment
6. Voting
7. Accepted answers and reputation
8. Related tags

Data Model

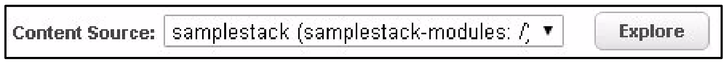
MarkLogic is the database for many different types of data: JSON, XML, RDF triples, full text and binary content. Samplestack data is modeled as JSON making it easy to be consumed by Java applications.

Let's take some time to get familiar with the Samplestack data.

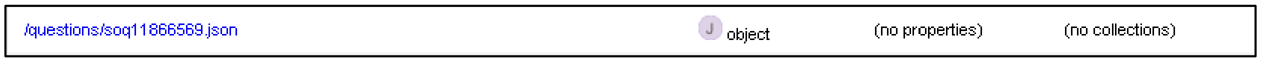
1. Open Query Console (http://localhost:8000/qconsole/)
2. Login with your MarkLogic admin account (Username = admin, Password = admin).

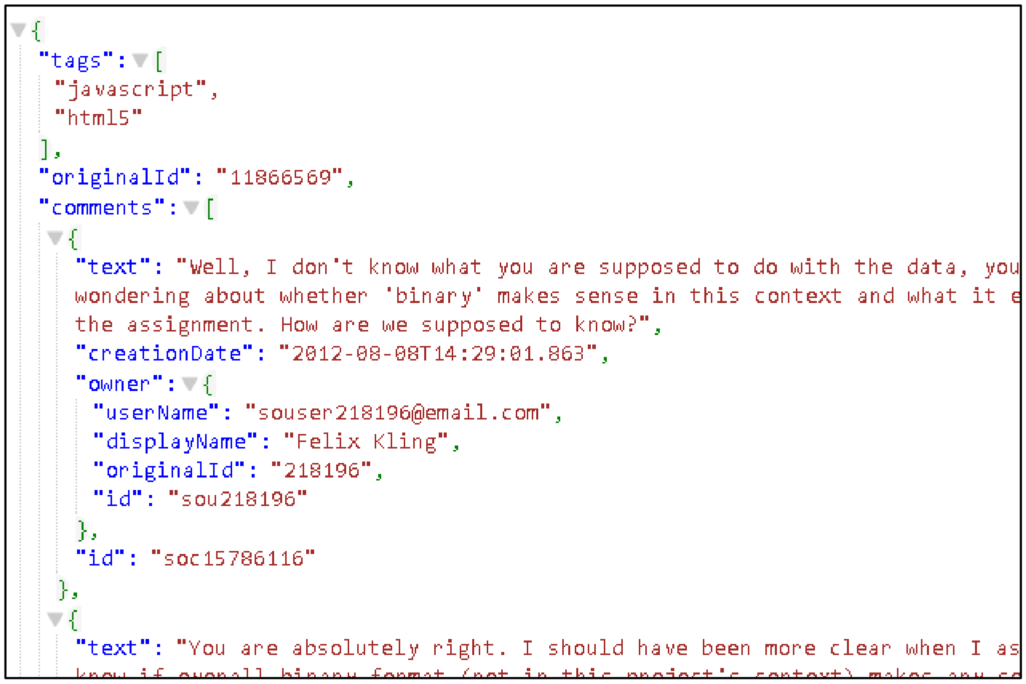
Note: *The MarkLogic Admin account was created during the automated deployment of Samplestack that was already setup for you on your VM.* *The account credentials are admin / admin.*

3. Select the Samplestack database from the Content Source dropdown:


Content Source: samplestack (samplestack-modules: /

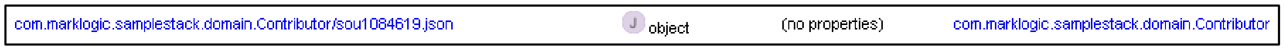
4. Click Explore.
5. Click on a JSON document representing a particular question and study its data model, for example:


/questions/soq11866569.json J object (no properties) (no collections)

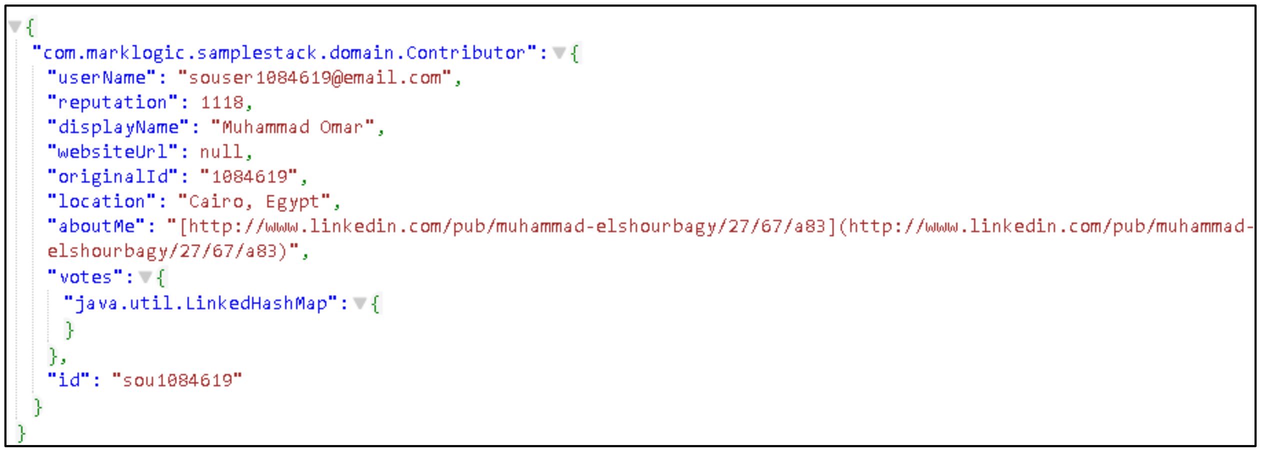


```
{
  "tags": [
    "javascript",
    "html5"
  ],
  "originalId": "11866569",
  "comments": [
    {
      "text": "Well, I don't know what you are supposed to do with the data, you
      wondering about whether 'binary' makes sense in this context and what it e
      the assignment. How are we supposed to know?",
      "creationDate": "2012-08-08T14:29:01.863",
      "owner": {
        "userName": "souser218196@email.com",
        "displayName": "Felix Kling",
        "originalId": "218196",
        "id": "sou218196"
      },
      "id": "soc15786116"
    },
    {
      "text": "You are absolutely right. I should have been more clear when I as
```

6. Click on a POJO that has been persisted to the database:



com.marklogic.samplestack.domain.Contributor/sou1084619.json	J object	(no properties)	com.marklogic.samplestack.domain.Contributor

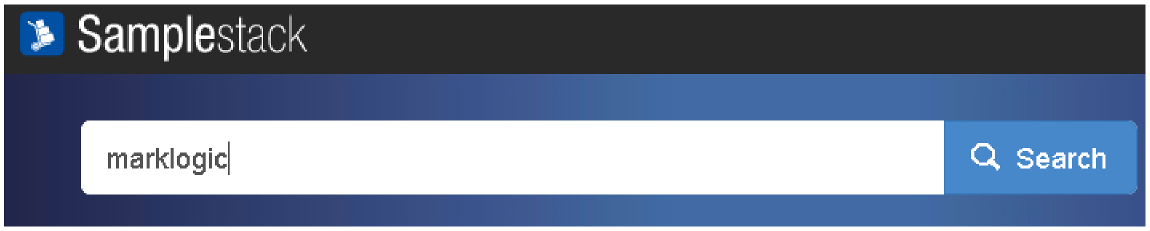


```
{
  "com.marklogic.samplestack.domain.Contributor": {
    "userName": "souser1084619@email.com",
    "reputation": 1118,
    "displayName": "Muhammad Omar",
    "websiteUrl": null,
    "originalId": "1084619",
    "location": "Cairo, Egypt",
    "aboutMe": "[http://www.linkedin.com/pub/muhammad-elshourbagy/27/67/a83](http://www.linkedin.com/pub/muhammad-
    elshourbagy/27/67/a83)",
    "votes": {
      "java.util.LinkedHashMap": {
      }
    },
    "id": "sou1084619"
  }
}
```

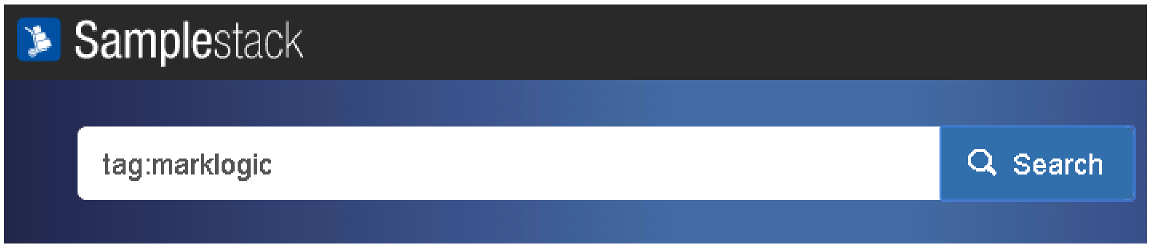
Full Text Search

In this exercise you will explore some of the full text search and facet functionality that has been built into the Samplestack application.

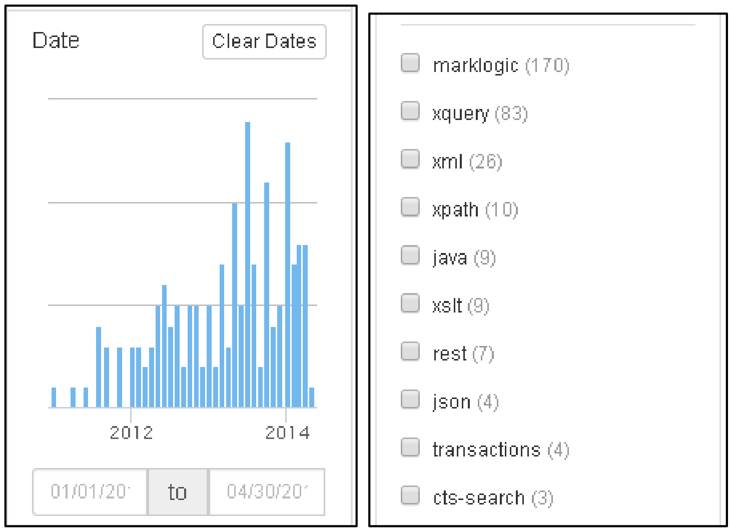
1. Open Samplestack if it's not still open in your browser (http://localhost:3000)
2. Search for "marklogic" and view the results.



3. Now search for "tag:marklogic" and view the results.



4. Now constrain the results even further by using the date range and tag facets.

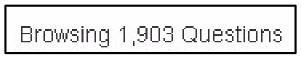


5. Next we will explore the security implementation in Samplestack.

User Accounts, Security and Restricted Content

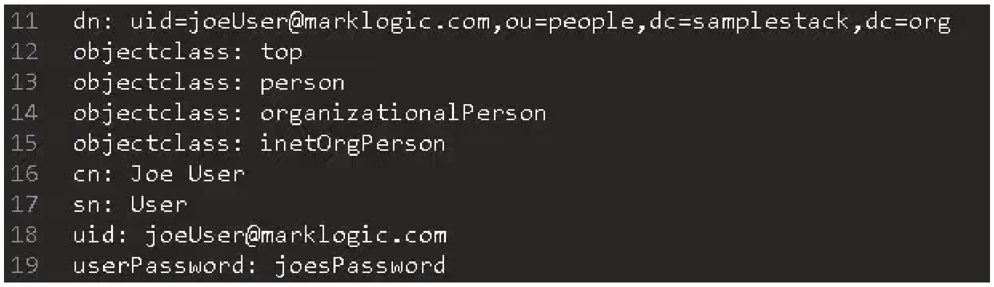
Samplestack is an example of an enterprise application, and therefore its implementation leverages some ancillary technologies common in large enterprises. Specifically, Samplestack shows an example of MarkLogic integrating with LDAP, as well as highlighting internal MarkLogic security concepts like roles, users, document permissions, privileges and authentication.

1. In your browser, go to the Samplestack application. When you go to the application unauthenticated you are by default treated as a user called "samplestack-guest". Notice the total amount of content you are able to see by default:


Browsing 1,903 Questions

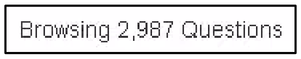
2. Now let's login as a specific user: joe@example.com / joesPassword

    - Note that there is no MarkLogic user called "joe@example.com"
    - This user is defined in LDAP, and the "samplestack-contributor" role is assigned when you login with these credentials.
    - Take a look at **c:\marklogic-samplestack\shared\samplestack-ds.ldif**:

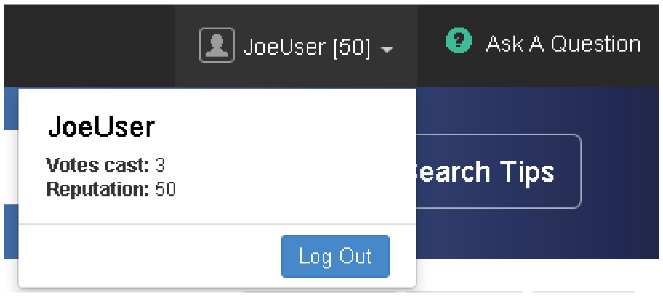


```
dn: uid=joeUser@marklogic.com,ou=people,dc=samplestack,dc=org
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Joe User
sn: User
uid: joeUser@marklogic.com
userPassword: joesPassword
```

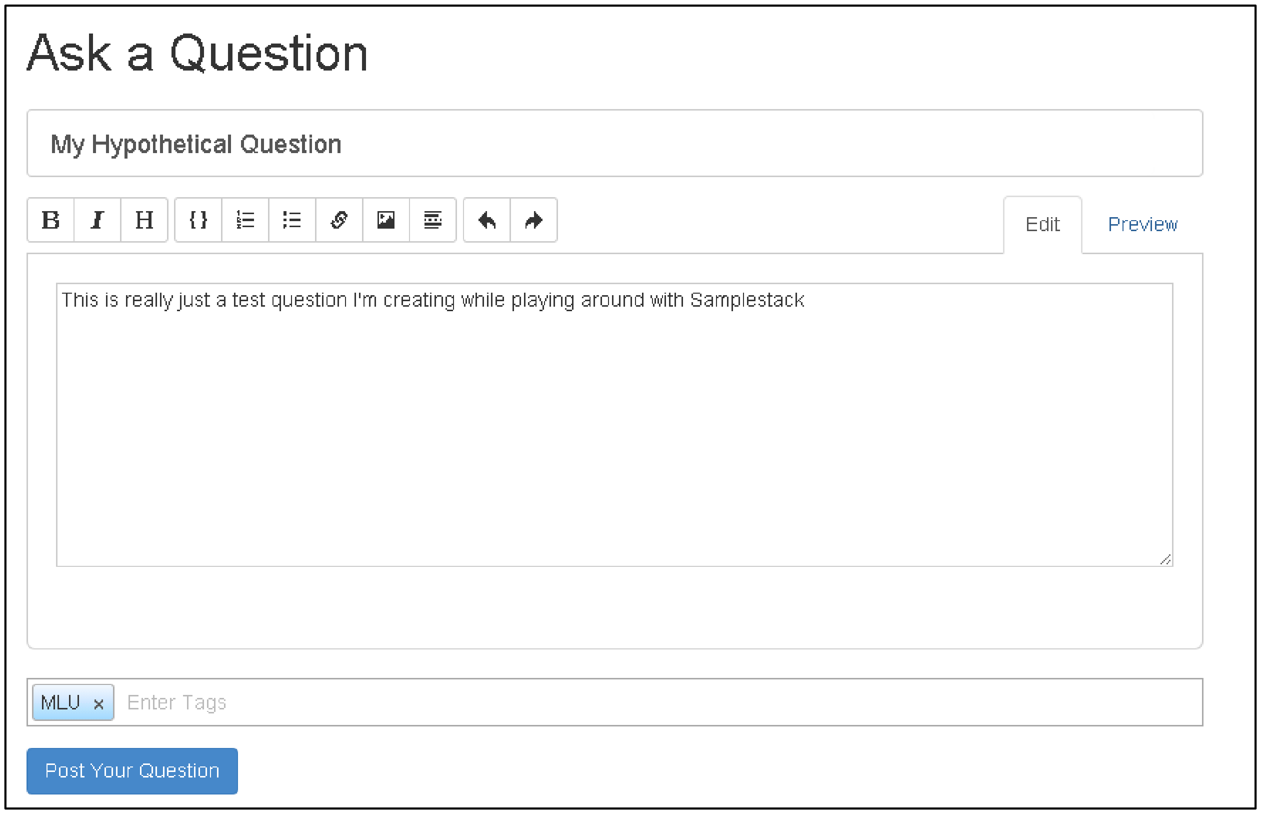
3. Note that since you are now logged in as Joe that you can see more content:


Browsing 2,987 Questions

4. Note also that once you are logged in, you see some statistics about the users contributions and you also have the ability to "Ask a question":



5. Go ahead and post an example question and/or answer a question. Make sure you put something in the question to make it easy to find via a search in query console. For example, try placing an "MLU" tag on your question.

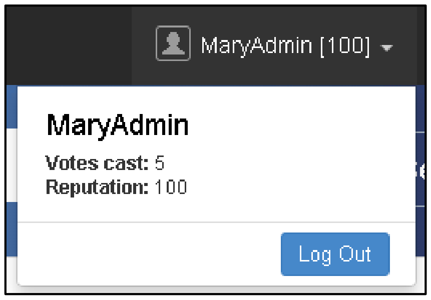


My Hypothetical Question

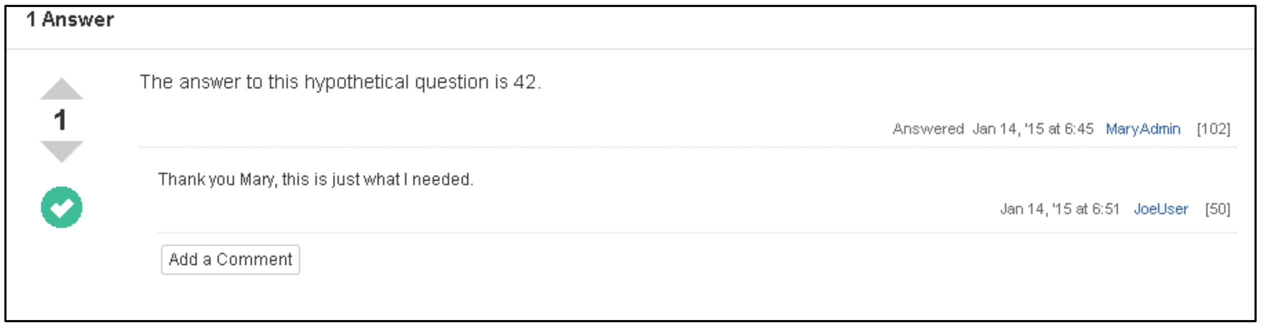
This is really just a test question I'm creating while playing around with Samplestack

MLU ×

6. Now, logout of your session as Joe.
7. Login as Mary (mary@example.com / marysPassword)
8. Note Mary's reputation:



9. Perform a search to find the question that was just created under Joe’s account (it should be the most recently posted question).
10. Add a comment to the question.
11. Add an answer to the question.
12. Logout of Mary’s session.
13. Begin another session as Joe (joe@example.com / joesPassword)
14. Find the question you created and give a positive vote on the answer, and then accept the answer by clicking the green checkmark. Add a comment to the answer if you like.



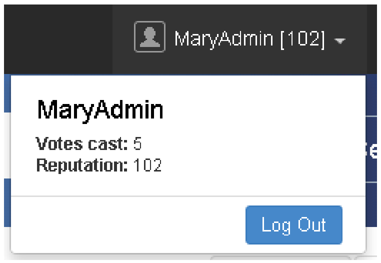
The answer to this hypothetical question is 42.

Answered Jan 14, '15 at 6:45 MaryAdmin [102]

Thank you Mary, this is just what I needed.

Jan 14, '15 at 6:51 JoeUser [50]

15. Logout of your session as Joe.
16. Login one more time as Mary (mary@example.com / marysPassword)
17. Notice that Mary’s reputation has now increased as a result her answer being approved:



18. Logout.

Exercise 3: Explore Node.js Client API Resources

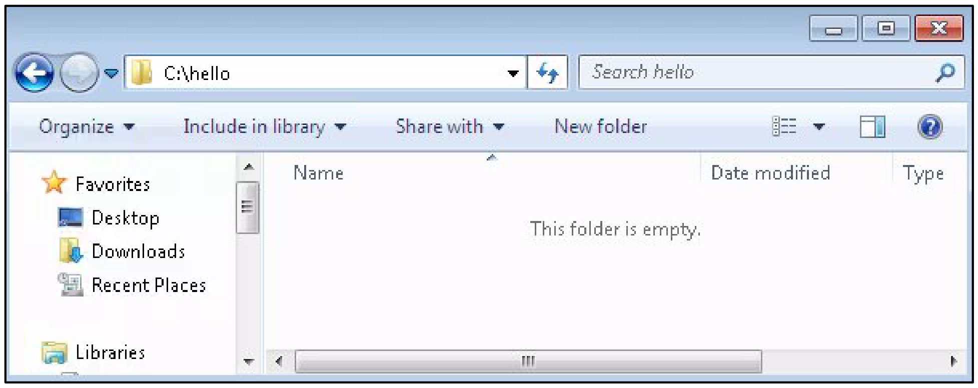
In this exercise you will take a few minutes to familiarize yourself with some of the resources available to help you as a developer. When writing code, as we will be throughout this training course, it's very helpful to have some of these resources available to you for reference.

1. On Demand Training
    - https://mlu.marklogic.com/ondemand/
    - Available for free, these short, self-paced tutorials cover a wide range of MarkLogic Server concepts and features.
2. Developer Community
    - http://developer.marklogic.com/
    - Tutorials, blogs, mailing lists, downloads and more. Join up to get plugged in with other members of the community.
3. Documentation
    - http://docs.marklogic.com/
    - Full product documentation combined with MarkLogic search enables you to find relevant content quickly. Check out the Java Developer guide for starters, and bookmark this site as you'll find it very useful when working with MarkLogic.
4. JS Doc
    - https://www.npmjs.com/package/marklogic
    - NPM documentation of the MarkLogic Node.js Client API.
5. MarkMail
    - http://markmail.org/
    - A searchable email archive (built on MarkLogic) documenting discussions on a variety of different technical forums, including the MarkLogic Developer Community email list.
6. Inside MarkLogic Server
    - http://developer.marklogic.com/inside-marklogic
    - Official technical white paper describing MarkLogic Server.

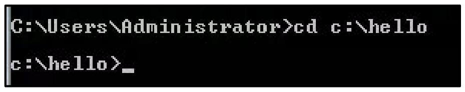
Exercise 4: Create a Simple Node.js App

In this exercise you will create a very simple Node.js app that will read a document from the Samplestack database and output some information from that document.

1. Using Windows Explorer, create a folder for your app off the **c:\** drive called "**hello**". Your result should look like this:

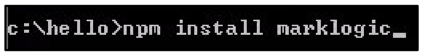


2. Open the command prompt.
3. Enter **cd c:\hello** and press enter to navigate to the new project folder:



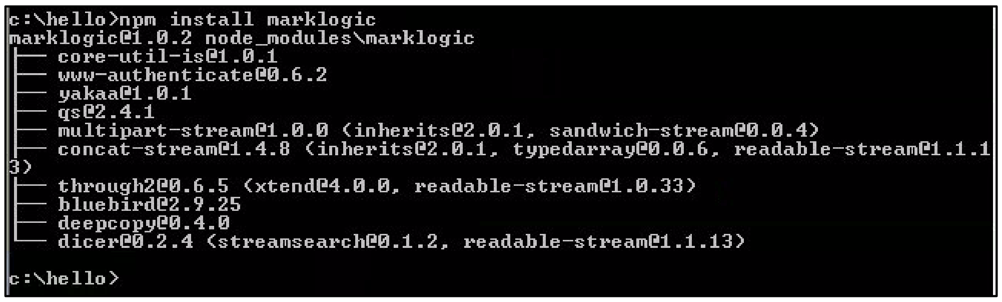
```
C:\Users\Administrator>cd c:\hello

c:\hello>_
```

4. Next, we need to install the MarkLogic Node.js client API.
5. From the **c:\hello** folder at the command line enter **npm install marklogic** and press enter:



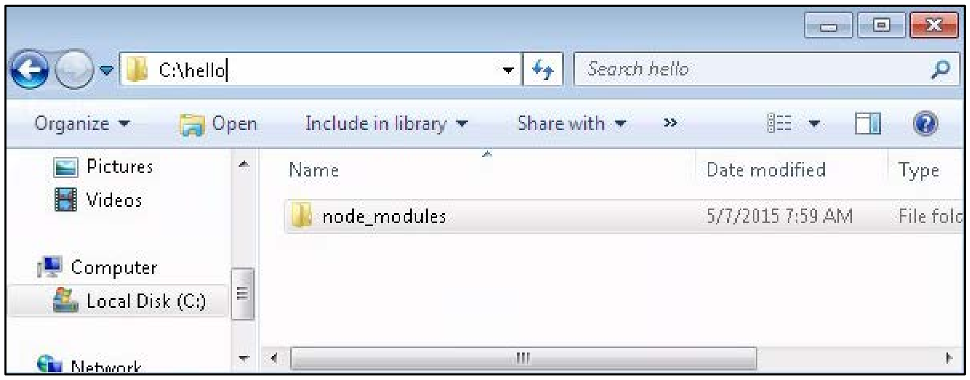
```
c:\hello>npm install marklogic_
```

6. When complete you should see this response:



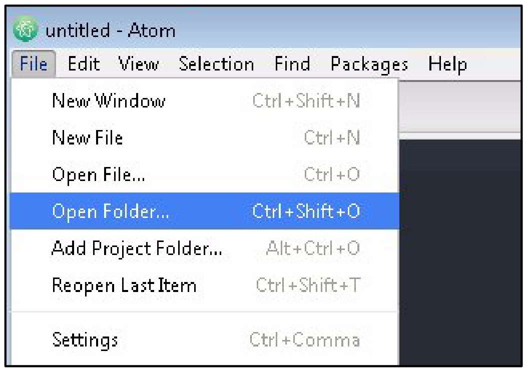
```
c:\hello>npm install marklogic
marklogic@1.0.2 node_modules\marklogic
├── core-util-is@1.0.1
├── www-authenticate@0.6.2
├── yakaa@1.0.1
├── qs@2.4.1
├── multipart-stream@1.0.0 (inherits@2.0.1, sandwich-stream@0.0.4)
├── concat-stream@1.4.8 (inherits@2.0.1, typedarray@0.0.6, readable-stream@1.1.1
3)
├── through2@0.6.5 (xtend@4.0.0, readable-stream@1.0.33)
├── bluebird@2.9.25
├── deepcopy@0.4.0
└── dicer@0.2.4 (streamsearch@0.1.2, readable-stream@1.1.13)

c:\hello>
```

7. Notice that a folder called **node_modules** has been created in your project directory:

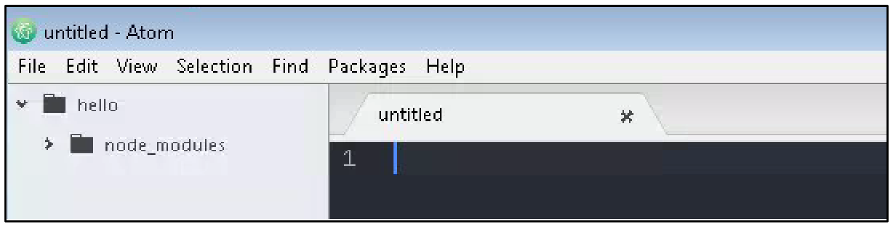


8. Keep your command line window open as we'll need to use that later on.
9. Next, open the Atom editor using the shortcut on your VM desktop:

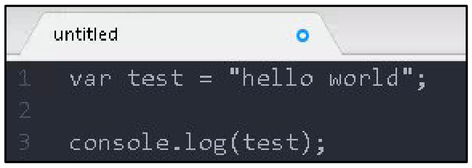
10. In the Atom editor, select File→Open Folder:


untitled - Atom
File Edit View Selection Find Packages Help
New Window Ctrl+Shift+N
New File Ctrl+N
Open File... Ctrl+O
Open Folder... Ctrl+Shift+O
Add Project Folder... Alt+Ctrl+O
Reopen Last Item Ctrl+Shift+T
Settings Ctrl+Comma

11. Open your "**hello**" project folder.
12. You should see the following folder structure in place in your editor:

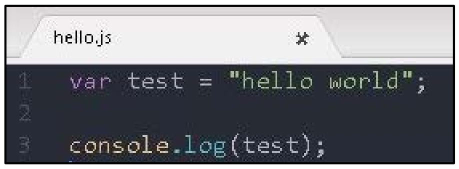


13. Next let's write some simple code and perform a test to make sure our environment is functioning.
14. In the "**untitled**" file that is in your editor, enter the following code:



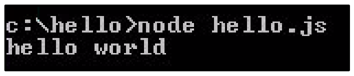
```
var test = "hello world";

console.log(test);
```

15. Save the untitled file (CTRL+S) and name it "**hello.js**":



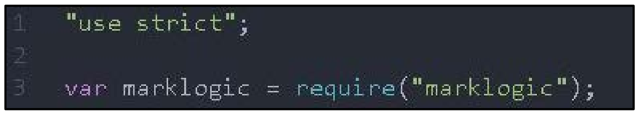
```
var test = "hello world";

console.log(test);
```

16. Back at the command line from the **c:\hello** folder enter **node hello.js** and press enter.
17. You should see the following results:



```
c:\hello>node hello.js
hello world
```

18. As simple as that is, it serves to validate our Node.js environment.
19. Now let's build on this by connecting to a MarkLogic database to read a document. Since Samplestack has already been setup on your VM, we'll use that database for this example.
20. Delete all the code from the hello.js file.
21. The first thing to do is import the MarkLogic Node.js client API. Enter the following code:



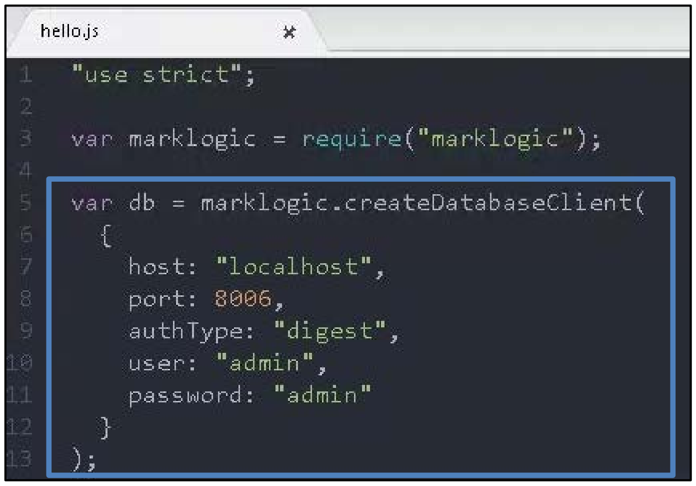
```
"use strict";

var marklogic = require("marklogic");
```

Note:
People learn in different ways. Some find value in writing the code and following along step by step. Others would prefer to study a working example.

If you fall into the latter category, you may copy+paste the final code into your hello.js file from:

c:\mls-developer-node\Unit01\ex4.txt

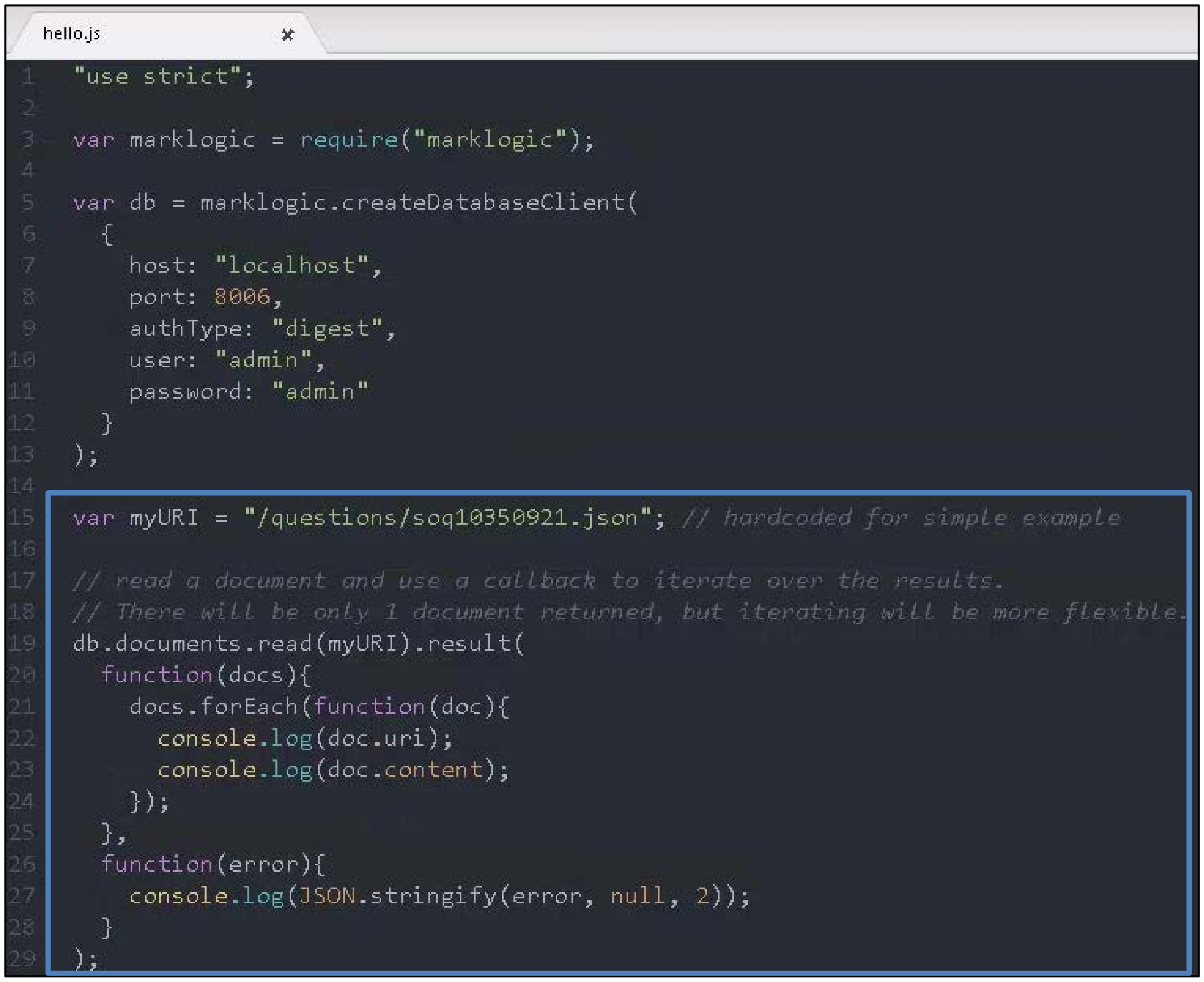
22. Next you will need to use the MarkLogic Node.js client API to connect to a database. This connection happens through a REST instance in MarkLogic (more on that later). Therefore to work with the database, we need to provide connection details to this REST instance, which already exists for the Samplestack application. Add the following code:



hello.js

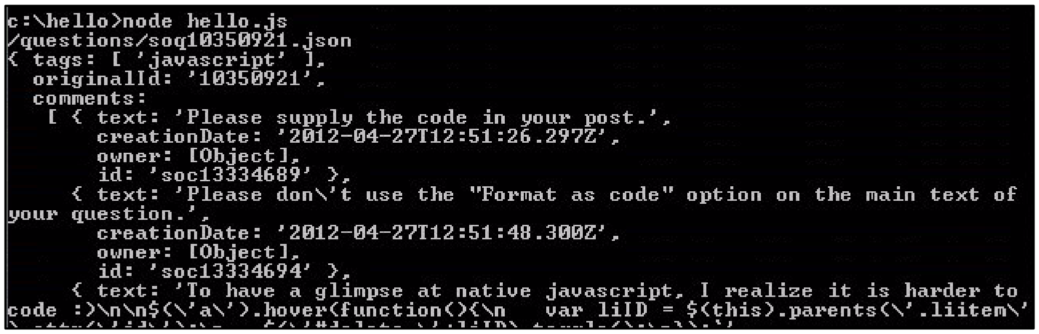
```
"use strict";

var marklogic = require("marklogic");

var db = marklogic.createDatabaseClient(
  {
    host: "localhost",
    port: 8006,
    authType: "digest",
    user: "admin",
    password: "admin"
  }
);
```

23. Next let's use the MarkLogic Node.js client API to read a specific document from the database and output the document URI and the document data. Enter the following code (without the comments if you wish, or you may copy + paste from **c:\mls-developer-node\Unit07\ex4.txt**):



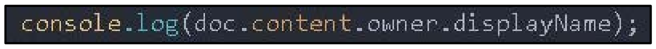
```javascript
"use strict";

var marklogic = require("marklogic");

var db = marklogic.createDatabaseClient(
  {
    host: "localhost",
    port: 8006,
    authType: "digest",
    user: "admin",
    password: "admin"
  }
);

var myURI = "/questions/soq10350921.json"; // hardcoded for simple example

// read a document and use a callback to iterate over the results.
// There will be only 1 document returned, but iterating will be more flexible.
db.documents.read(myURI).result(
  function(docs){
    docs.forEach(function(doc){
      console.log(doc.uri);
      console.log(doc.content);
    });
  },
  function(error){
    console.log(JSON.stringify(error, null, 2));
  }
);
```

24. Save hello.js.
25. Test your code from the command line (**node hello.js**) and view the results:



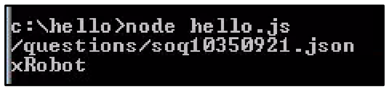
```
c:\hello>node hello.js
/questions/soq10350921.json
{ tags: [ 'javascript' ],
  originalId: '10350921',
  comments:
   [ { text: 'Please supply the code in your post.',
       creationDate: '2012-04-27T12:51:26.297Z',
       owner: [Object],
       id: 'soc13334689' },
     { text: 'Please don\'t use the "Format as code" option on the main text of
your question.',
       creationDate: '2012-04-27T12:51:48.300Z',
       owner: [Object],
       id: 'soc13334694' },
     { text: 'To have a glimpse at native javascript, I realize it is harder to
code :)\n\n$(\'a\').hover(function(){\n    var liID = $(this).parents(\'.liitem\'
```

26. Now let’s say you didn’t want to return the whole document, but just the username of the person who posted the question.
27. To access this specific JSON property, modify the code as follows:



```
console.log(doc.content.owner.displayName);
```

28. Save your changes.
29. Test your code from the command line (**node hello.js**) and view the results:



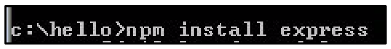
```
c:\hello>node hello.js
/questions/soq10350921.json
xRobot
```

Exercise 5: Create a Simple Front-End

In this exercise you will build a simple front-end for an app using a Jade template. We won't spend much time in this course on front-end development as our main objective is to work with the MarkLogic Node.js client API on the middle-tier and MarkLogic on the database tier.

However, this will give you a simple starter should you be interested to explore the topic further.

1. Install Express. At the command prompt in your project directory enter **npm install express**:



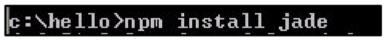
```
c:\hello>npm install express
```

Note:
Express provides simple tooling for HTTP servers ideally suited for single page applications, websites, or HTTP APIs.

Express does not require you to use any specific template engine.

https://www.npmjs.com/package/express

2. Install Jade. At the command prompt in your project directory enter **npm install jade**:

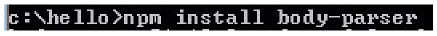


```
c:\hello>npm install jade
```

Note:
Jade is a template engine implemented with JavaScript and designed for Node and browsers. It's the template engine we will be using in this example.

https://www.npmjs.com/package/jade

3. Install body-parser. At the command prompt in your project directory enter **npm install body-parser**:

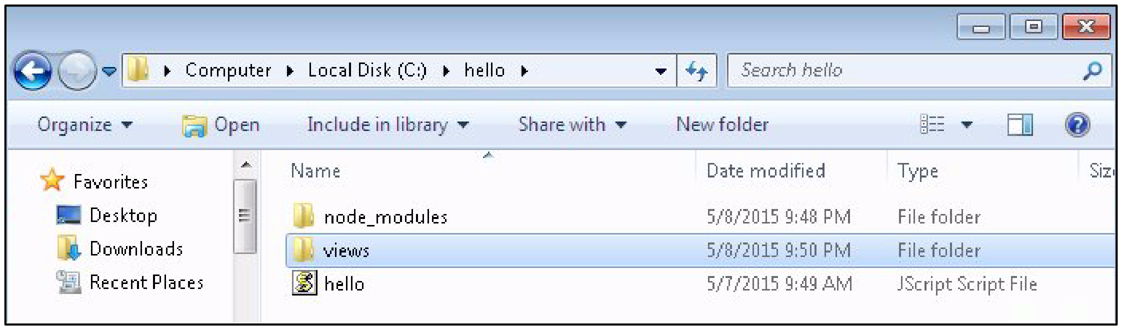


```
c:\hello>npm install body-parser
```

Note:
Body-parser is a package that will help us parse the body of a multi-part post request.

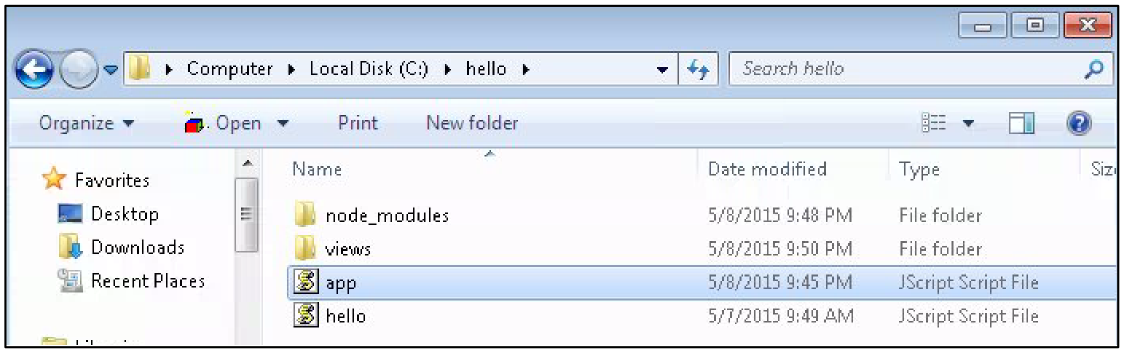
We'll need this in order to get the data entered by a user in an input field on a web form and use that data in our Node.js app.

4. Next let's setup and study the code.
5. Create a folder called **views** in your project directory at **c:\hello** as shown:



Name	Date modified	Type
node_modules	5/8/2015 9:48 PM	File folder
views	5/8/2015 9:50 PM	File folder
hello	5/7/2015 9:49 AM	JScript Script File

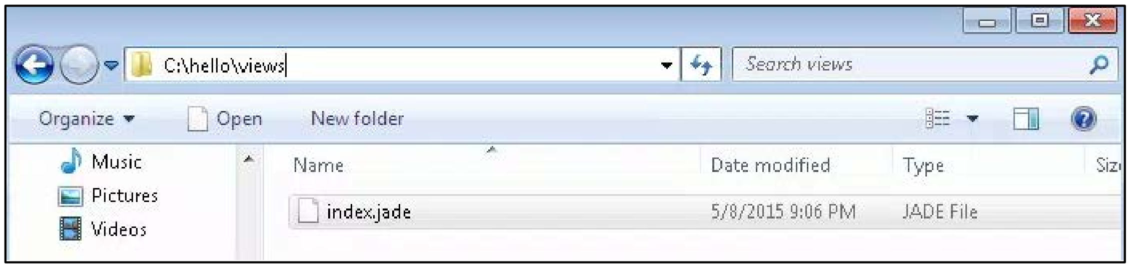
6. Copy **app.js** from **c:\mls-developer-node\Unit01** and paste it into your project directory at **c:\hello**. You should now see the following structure:



Name	Date modified	Type
node_modules	5/8/2015 9:48 PM	File folder
views	5/8/2015 9:50 PM	File folder
app	5/8/2015 9:45 PM	JScript Script File
hello	5/7/2015 9:49 AM	JScript Script File

Note:
app.js is the main module of our Node.js middle tier.

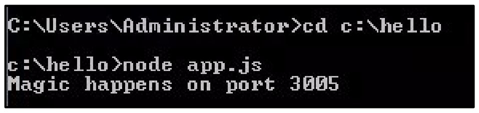
7. Copy **index.jade** from **c:\mls-developer-node\Unit01** and paste it into the newly created **views** folder in your project directory:



Name	Date modified	Type
index.jade	5/8/2015 9:06 PM	JADE File

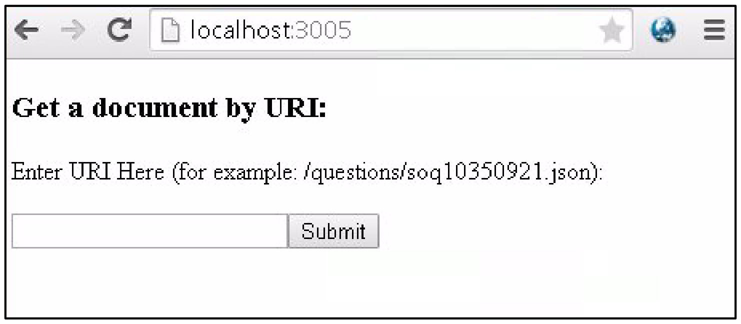
Note:
index.jade is the template for the front-end.

8. Open a command prompt and navigate to **c:\hello**
9. From the command prompt, key **node app.js** and press enter to launch the application:



```
C:\Users\Administrator>cd c:\hello

c:\hello>node app.js
Magic happens on port 3005
```

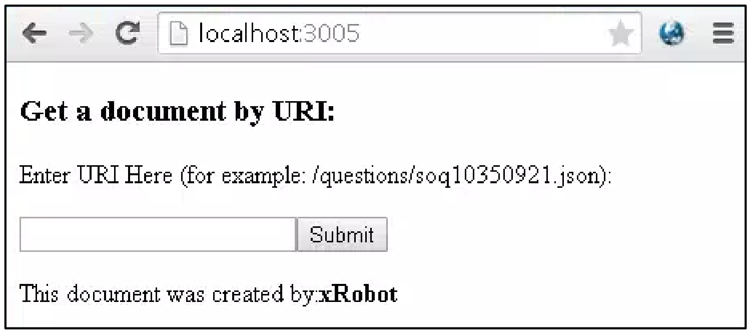
10. Launch a new tab in Chrome and go to **localhost:3005** to view the application:



Get a document by URI:

Enter URI Here (for example: /questions/soq10350921.json):

11. Enter a valid URI in the textbox and click submit to view the response (for an example URI you may copy + paste the one provided on the user interface):

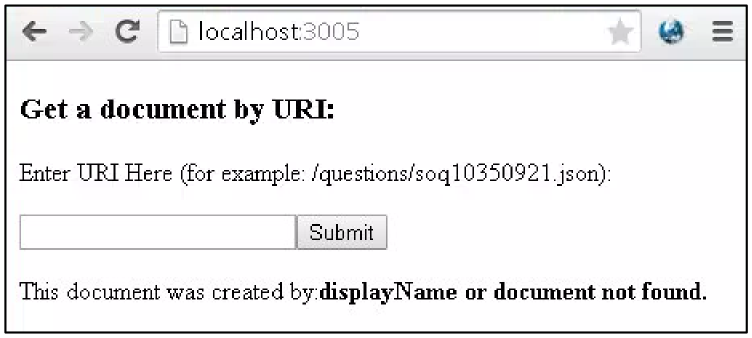


Get a document by URI:

Enter URI Here (for example: /questions/soq10350921.json):

This document was created by:**xRobot**

12. Enter an invalid URI (for example “test”) and click submit to view the response:


localhost:3005

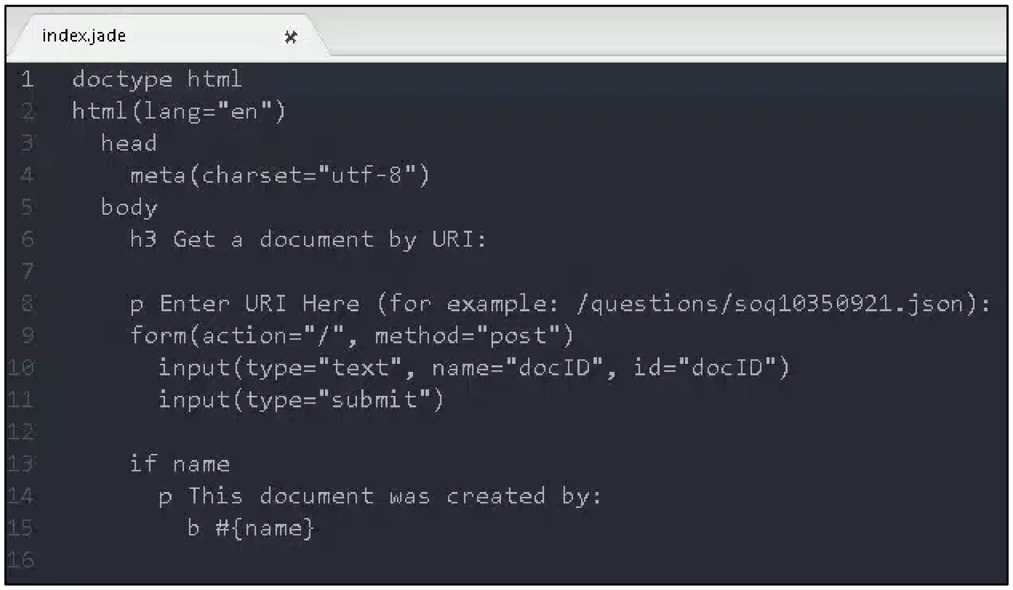
Get a document by URI:

Enter URI Here (for example: /questions/soq10350921.json):

Submit

This document was created by:**displayName or document not found.**

13. Next, let's take a look at the code that enables this functionality.
14. Open **index.jade** from **c:\hello\views\** and study the code:


index.jade

```
doctype html
html(lang="en")
  head
    meta(charset="utf-8")
  body
    h3 Get a document by URI:

    p Enter URI Here (for example: /questions/soq10350921.json):
    form(action="/", method="post")
      input(type="text", name="docID", id="docID")
      input(type="submit")

    if name
      p This document was created by:
        b #{name}

```

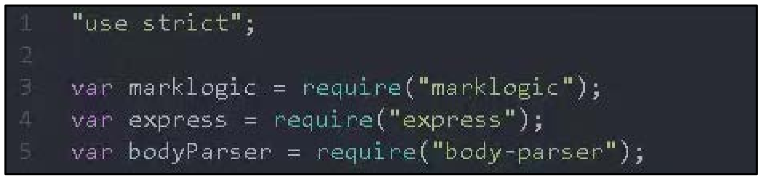
Note:
Jade is one of many alternatives for building a front end. When using Jade for templating, white space and indentation are very important. Indented elements are treated as children of the out dented element. For example, our input fields are children of the outer form element because of their indentation.

If you're interested in front end development with Jade, please see additional documentation at:

http://jade-lang.com/

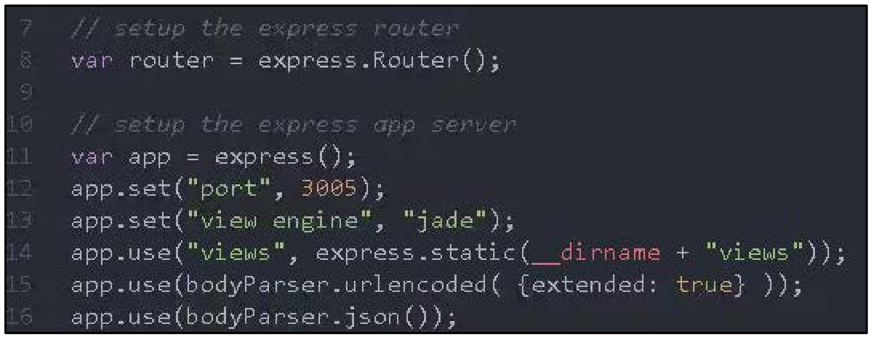
15. Open **app.js** from **c:\hello\** and study the code and the comments.

16. This first part of the code is importing the MarkLogic, Express, and body-parser packages that were installed for the project:



```
"use strict";

var marklogic = require("marklogic");
var express = require("express");
var bodyParser = require("body-parser");
```

17. The next part of the code is setting up the Express routing and the Express application server:



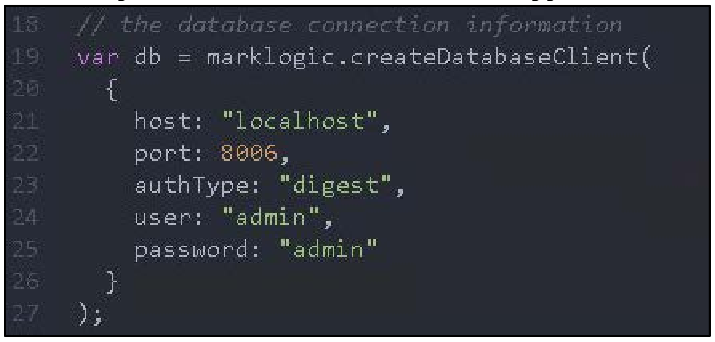
```
// setup the express router
var router = express.Router();

// setup the express app server
var app = express();
app.set("port", 3005);
app.set("view engine", "jade");
app.use("views", express.static(__dirname + "views"));
app.use(bodyParser.urlencoded( {extended: true} ));
app.use(bodyParser.json());
```

Note:
Routing refers to setting up of different endpoints for the application and configuring how they will respond to client requests. For this simple example, we will have a route setup for get and post requests in order to handle them appropriately.

The Express app server is also being configured in this code, where it specifies the port, templating engine, and location of front end templates. For more information on Express please see:

http://expressjs.com/

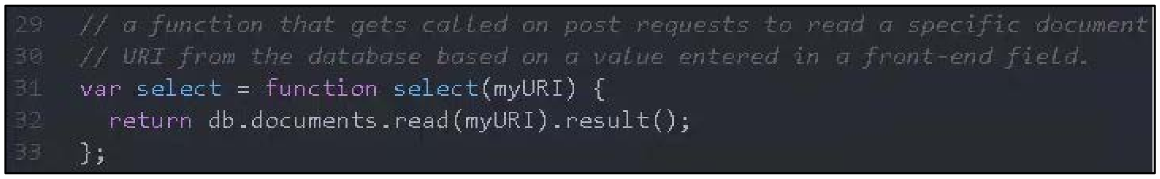
18. The next part of the code defines how the app will connect to the MarkLogic database:



```
// the database connection information
var db = marklogic.createDatabaseClient(
  {
    host: "localhost",
    port: 8006,
    authType: "digest",
    user: "admin",
    password: "admin"
  }
);
```

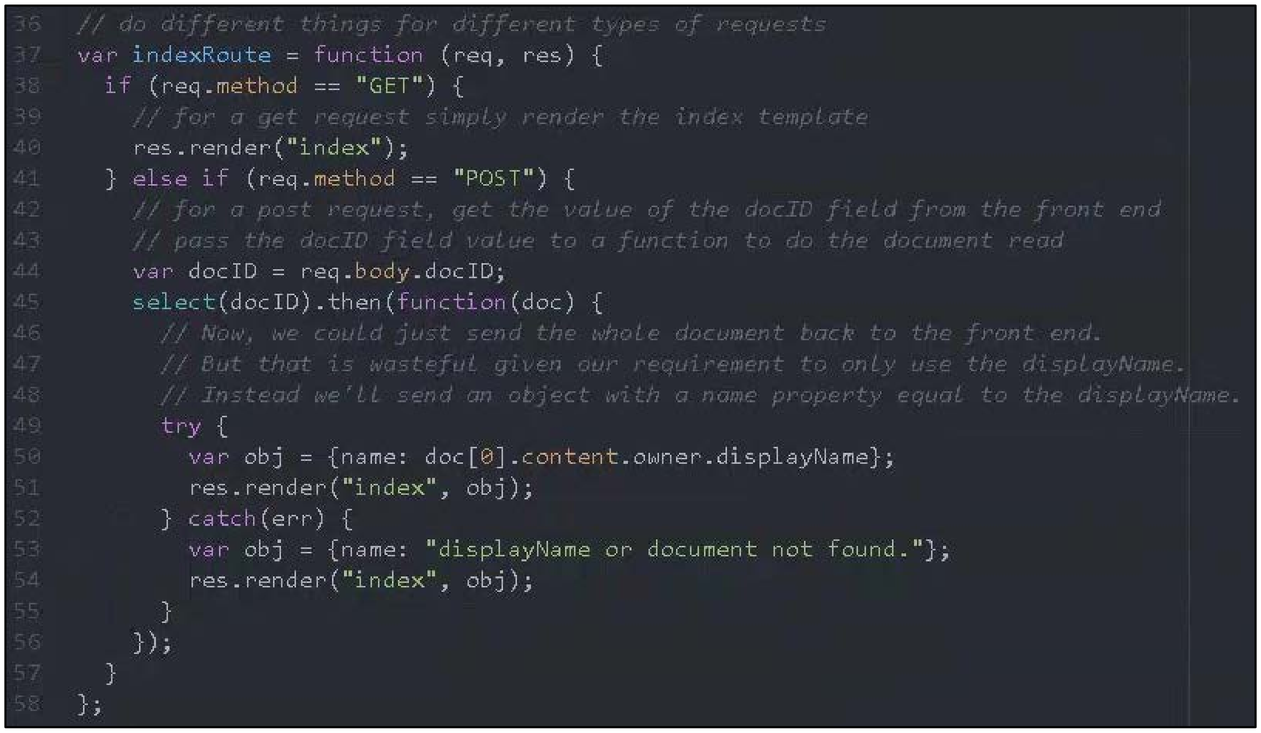
Note:
In this example you are using the Samplestack database that was already setup. Over the course of this training we will also learn to create our own databases and REST instances for the projects we will be building from the ground up.

19. In the next part of the code you see a function that will get a document from the database based on a specific URI that is being sent to the application from the form on the front-end. The routing will dictate that this function will only be used for POST requests, as we only want to go and get a document after a user has clicked the submit button on our form:



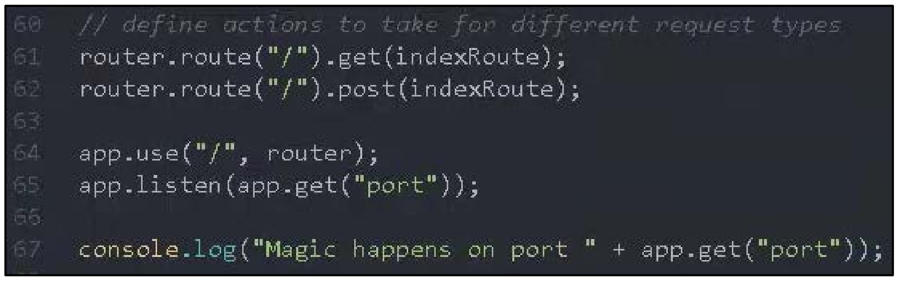
```
// a function that gets called on post requests to read a specific document
// URI from the database based on a value entered in a front-end field.
var select = function select(myURI) {
  return db.documents.read(myURI).result();
};
```

20. The next part of the code is the heart of what we are doing in this example. It defines what to do for both GET and POST requests. If it's a **GET** request, the app should just **display the index page**. If it's a **POST** request, the app should get the value of the **docID** field on the form and pass it to the **select()** function that does the read of the document from the database. A promise (**.then**) is used to process the result of the document read and send the appropriate data back to the front-end:



```
// do different things for different types of requests
var indexRoute = function (req, res) {
  if (req.method == "GET") {
    // for a get request simply render the index template
    res.render("index");
  } else if (req.method == "POST") {
    // for a post request, get the value of the docID field from the front end
    // pass the docID field value to a function to do the document read
    var docID = req.body.docID;
    select(docID).then(function(doc) {
      // Now, we could just send the whole document back to the front end.
      // But that is wasteful given our requirement to only use the displayName.
      // Instead we'll send an object with a name property equal to the displayName.
      try {
        var obj = {name: doc[0].content.owner.displayName};
        res.render("index", obj);
      } catch(err) {
        var obj = {name: "displayName or document not found."};
        res.render("index", obj);
      }
    });
  }
};
```

21. The final bit of code sets up the routes and tells the app server to listen on the port that was defined earlier in the code:



```
// define actions to take for different request types
router.route("/").get(indexRoute);
router.route("/").post(indexRoute);

app.use("/", router);
app.listen(app.get("port"));

console.log("Magic happens on port " + app.get("port"));
```

Note:
As we progress through this training, we will get to go deeper into both MarkLogic as a product and the MarkLogic Node.js client API.

This example is meant to be simple but also to show you an example early on of how all the components of a 3 tiered architecture can come together.

The focus of this training is on learning the MarkLogic Node.js API, so we will spend most of our time developing code on the middle tier using Node.js.