Unit 5: Using the Node.js Client API

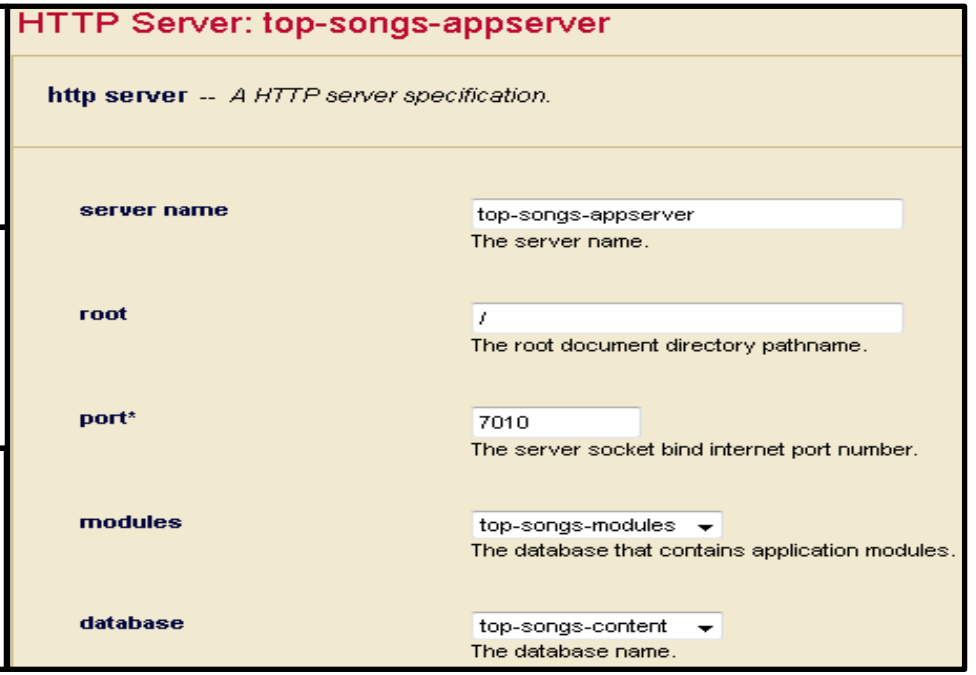
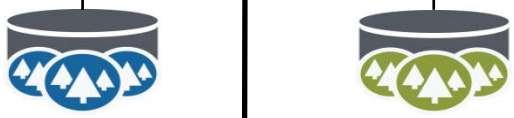
MarkLogic

Learning Objectives

- Install and import the MarkLogic Node.js client API.
- Connect to a database.
- Create database connections using project security configuration.
- Access JSON and XML document data.
- Use callbacks, promises and streams.
- Work with document descriptors.
- Explore Samplestack code.

Getting Started Requirements

- Create a REST API instance and load some sample data (done)
- Create users and assign appropriate roles (done)
- Now its time to write some code!

HTTP App Server Configured for REST • top-songs-appserver • Port = 7010		
Databases (logical) • top-songs-content • top-songs-modules		
Forests (physical) • top-songs-content-1 • top-songs-modules-1		

MarkLogic

Installing the API

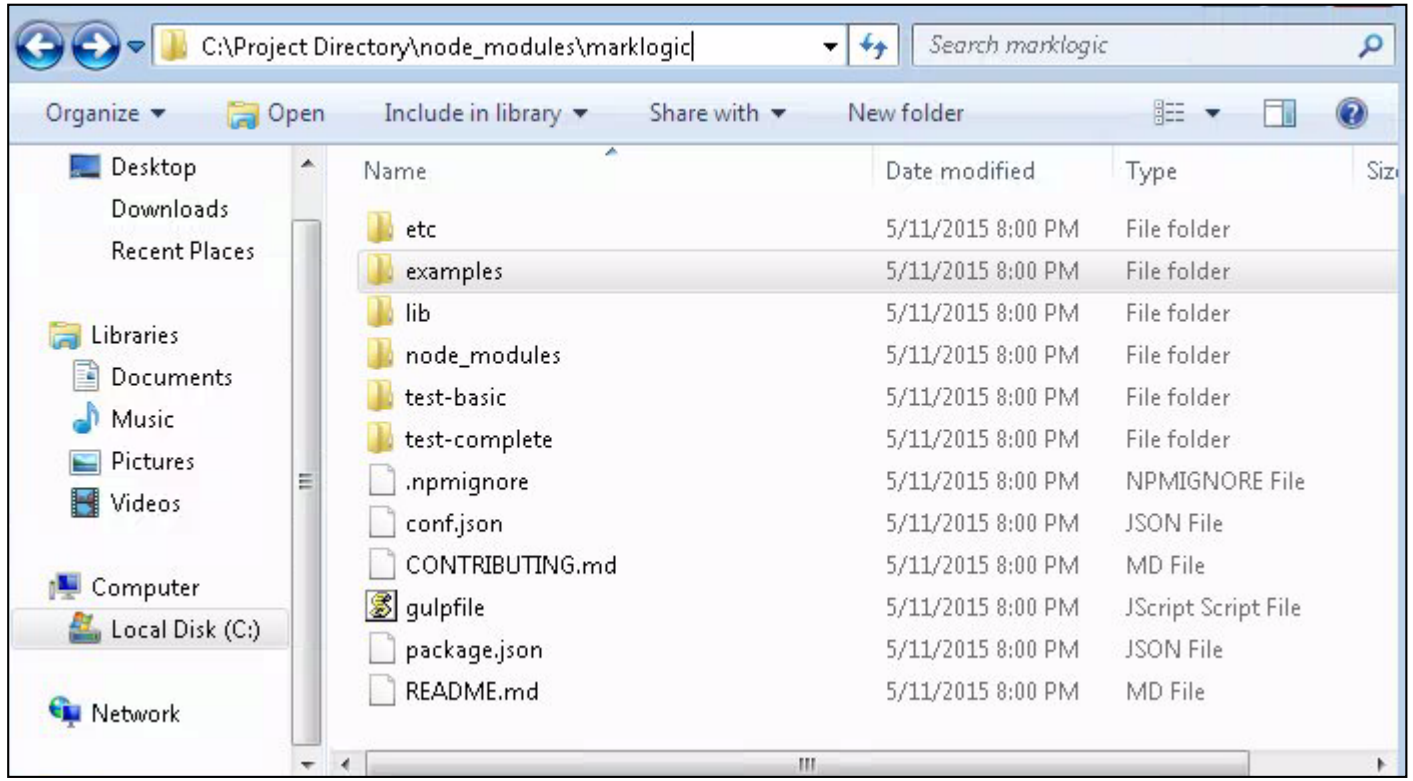
- Github site:
  - https://github.com/marklogic/node-client-api
- Available through NPM
  - https://www.npmjs.com/package/marklogic
- Install using NPM from the command line:

```
Project Directory>npm install marklogic
```

MarkLogic

Installing the API

- After installation you'll see the API in your project folder under **/node_modules/**
  - See the **/marklogic/examples/** directory for additional code samples



MarkLogic

Using the API

- Load the Node.js client API module:

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

MarkLogic

Connect to a Database

- Create a database client connection through your REST instance:

```
var db = marklogic.createDatabaseClient(
  {
    host:"localhost",
    port:7010,
    authType:"digest",
    user:"admin",
    password:"admin"
  }
);
```

MarkLogic

Managing Database Client Connections

- Remember our security configuration!
  - Using Admin account isn't realistic.
- Instead, create a pool of connections:
  - One for each project user.
  - Use appropriate connections in your app based on desired goal.

```
module.exports = {
  restReader: {
    host: 'localhost',
    port: 7010,
    user: 'rest-reader-user',
    password: 'training'
  },
  restWriter: {
    host: 'localhost',
    port: 7010,
    user: 'rest-writer-user',
    password: 'training'
  },
  restAdmin: {
    host: 'localhost',
    port: 7010,
    user: 'rest-admin-user',
    password: 'training'
  }
};
```

MarkLogic

Managing Database Client Connections

- Import and use the connection module.
  - Example assumes connections.js is in the same folder as the module loading it.
  - This creates a database client for each of our project users.

```
var dbConn = require("./connections.js")

var dbRead = marklogic.createDatabaseClient(dbConn.restReader);
var dbWrite = marklogic.createDatabaseClient(dbConn.restWriter);
var dbAdmin = marklogic.createDatabaseClient(dbConn.restAdmin);
```

MarkLogic

Read Data

- Use the database client to read a specific document from the database:

```
var myURI = "/songs/song2.json";

dbRead.documents.read(myURI).result(
  function(documents){
    documents.forEach(function(document){
      console.log("URI=" + document.uri);
      console.log("DOCUMENT=" + JSON.stringify(document.content));

      // note:  Because of the hyphen in the top-song property we cannot
      // use dot notation to access that property, and instead must use ["top-song"]
      console.log("ARTIST=" + document.content["top-song"].artist);
      console.log("TITLE=" + document.content["top-song"].title);
    });
  },
  function(error){
    console.log(JSON.stringify(error, null, 2));
  }
);
```

MarkLogic

Callbacks

- This example uses a callback.
- We call the result method and pass in a success and error function.
- Asynchronous -- useful if you don't need to synchronize the result with some other action.

```
dbRead.documents.read(myURI).result(
  function(documents){
    documents.forEach(function(document){

      // Do something with your data

    });
  },
  function(error){
    console.log(JSON.stringify(error, null, 2));
  }
);
```

MarkLogic

Promises

- A promise is a way to interact with the results of an asynchronous event. For example:
  - Insert a document and search the database.
  - Ensure the insert has occurred before the search, so that if the new document matches, it would be returned in the results.
- Promises include **then**, **catch** and **finally** methods.

```
dbRead.documents.read(myURI).result().then(
  function(documents){
    documents.forEach(function(document){

      // Do something with your data

    });
  },
  function(error){
    console.log(JSON.stringify(error, null, 2));
  }
);
```

MarkLogic

Streams

- Use when handling large documents or large result sets.
- For both read and write.
- Process results incrementally.

```
dbRead.documents.read(myURI).stream()
  .on("data", function(document){
    // on("data") means a full doc has been received
    // Process the document
  }).on("end", function(){
    // on("end") means all docs have been received
    // Finish up
  }).on("error", function(error){
    // Handle errors
  });
```

MarkLogic

Document Descriptors

- Returned by read operations.
  - It's the response we will work with in our app when we do a read.
- Accepted by write operations
  - Describes the document to write.
- An object that encapsulates document content and metadata as named JavaScript object properties.

```
{
  uri: "myURI.json",
  content:
    {
      book:
      {
        title: "A Tale of Two Cities",
        author: "Charles Dickens"
      }
    },
  collections: ["classics", "fiction", "British"]
}
```

Demo: Database Clients in Samplestack

Labs: Unit 5

Exercise 1: Install the MarkLogic Node.js Client API
Exercise 2: Define Project Database Connections
Exercise 3: Read a JSON Document Using a Callback
Exercise 4: Read an XML Document Using a Callback
Exercise 5: Read Documents Using Promises
Exercise 6: Read Documents Using Streams
DIY: Create a Database Client

Unit Review Question 1:

A project may only have one database client defined.

1. True
2. False

Unit Review Question 1:

A project may only have one database client defined.

1. True
2. **False**

Unit Review Question 2:

You would use a promise if…

1. You desire operations to occur asynchronously.
2. You desire operations to occur synchronously.
3. You need to work with large documents.
4. You don’t really mean what you are about to say.

Unit Review Question 2:

You would use a promise if…

1. You desire operations to occur asynchronously.
2. **You desire operations to occur synchronously.**
3. You need to work with large documents.
4. You don’t really mean what you are about to say.

Unit Review Question 3:

Assume this code returns this document descriptor.

How would you output the title and author of the book?

```
dbRead.documents.read(myURI).result(
  function(documents){
    documents.forEach(function(document){

      // Do something with your data

    });
  },
  function(error){
    console.log(JSON.stringify(error, null, 2));
  }
);
```

```
{
  uri: "myURI.json",
  content:
    {
      book:
      {
        title: "A Tale of Two Cities",
        author: "Charles Dickens"
      }
    },
  collections: ["classics", "fiction", "British"]
}
```

Unit Review Question 3:

Assume this code returns this document descriptor.

How would you output the title and author of the book?

```
dbRead.documents.read(myURI).result(
  function(documents){
    documents.forEach(function(document){
      console.log(document.content.book.title);
      console.log(document.content.book.author);
    });
  },
  function(error){
    console.log(JSON.stringify(error, null, 2));
  }
);
```

```
{
  uri: "myURI.json",
  content:
    {
      book:
      {
        title: "A Tale of Two Cities",
        author: "Charles Dickens"
      }
    },
  collections: ["classics", "fiction", "British"]
}
```