Unit 12

Introduction to Server Side JavaScript

MarkLogic

Learning Objectives

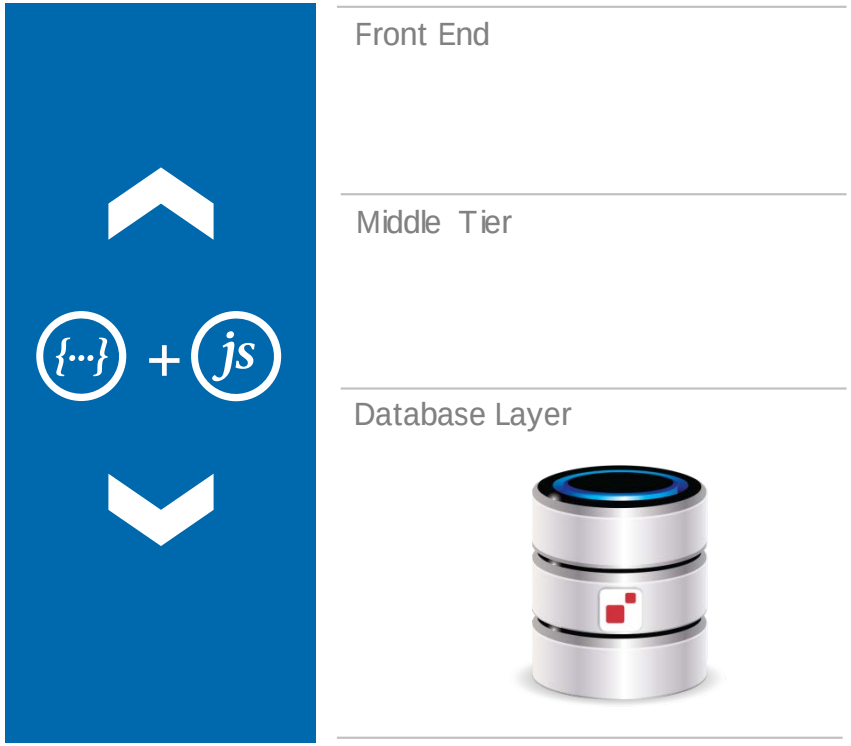
- Describe the implementation of server side JavaScript in MarkLogic.
- Describe the uses of server side JavaScript for the Node.js developer.
- Write code using server side JavaScript.
- Install and invoke a server side JavaScript extension.
- Explore server side JavaScript extensions in Samplestack.

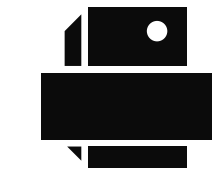
MarkLogic

Server-side JavaScript

JavaScript runtime *inside* MarkLogic using Google's V8

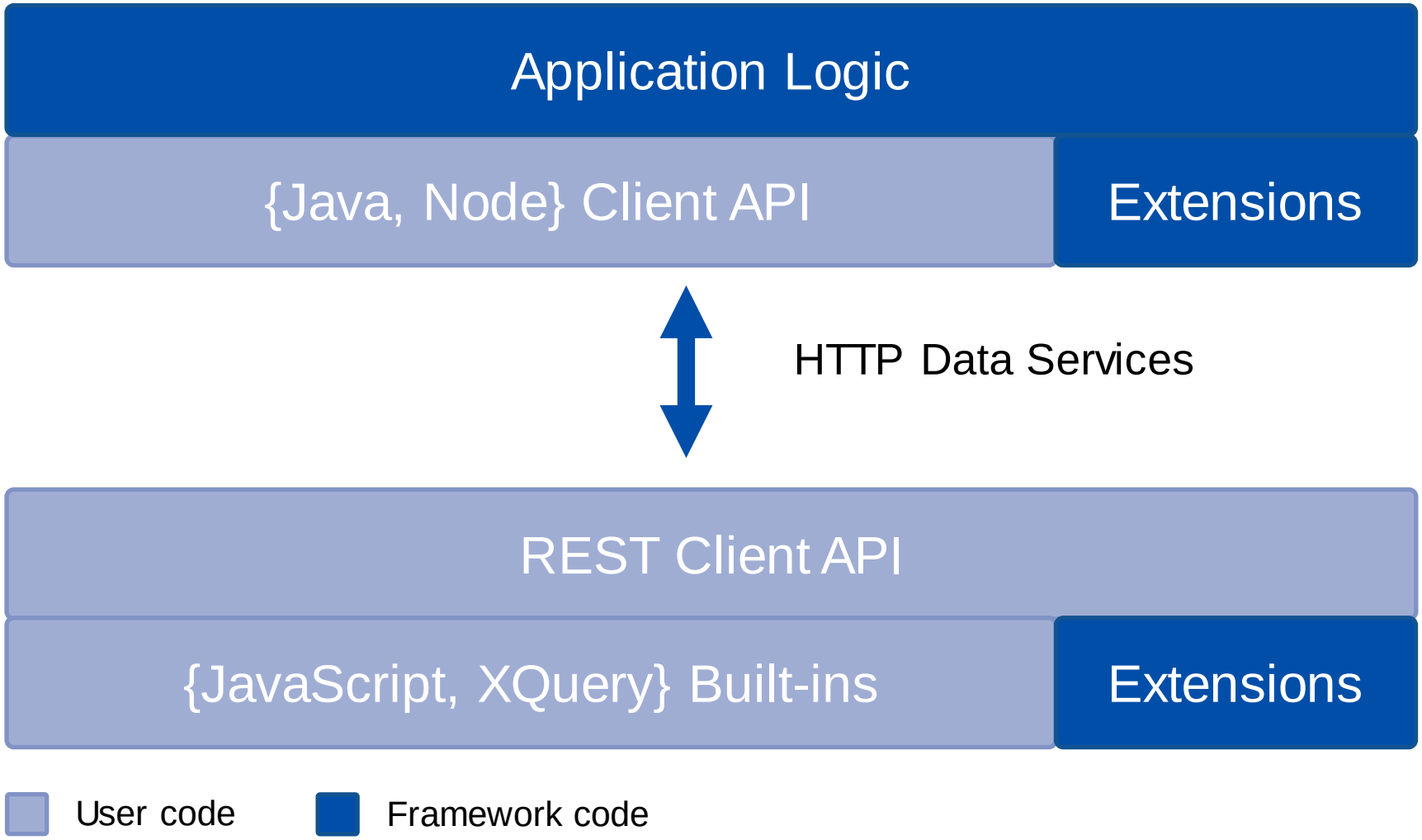
- Run code near the data for unparalleled power, efficiency
- Build applications faster from a growing pool of skills, tools
- Reduce risk with proven performance and reliability
- Decrease brittle ETL and lost fidelity and functionality from JSON data conversions
- Server Side JavaScript is executed at the E-Node level





Application

MarkLogic



MarkLogic

Why Server Side JavaScript?

- Enable developers to write code closest to the data when needed.
- Develop extensions to client APIs for custom functionality.
- Leverage a common scripting language (JavaScript).
- Model and manipulate documents, relationships, and metadata—combining JSON, XML, RDF, text, and binary
- Unified JavaScript interface for all indexes, data formats
  - Text search, semantic inference, aggregates, geospatial, alerting
- An alternative to XQuery / XML for writing server side code / data modeling.

MarkLogic

Functions & Logic

- XQuery functions have a JavaScript equivalent.
- Callbacks, variable declarations, logic structures, etc. are available in JavaScript.
- Server responses in JSON format as opposed to XML.
- Function Examples:

XQuery	JavaScript
fn:doc()	fn.doc()
xdmp:document-load()	xdmp.documentLoad()
cts:word-query()	cts.wordQuery()
cts:element-range-query()	cts.elementRangeQuery()
sem:rdf-insert()	sem.rdfInsert()

MarkLogic

Namespaces and Modules

- Global namespaces:
  - `xdmp`, `cts`, `sem`, etc.
- CommonJS-style modules:
  - `module.exports`, `require()`
- Same path resolution, precedence as XQuery
- Import existing XQuery modules
  - Admin
  - Search
  - Etc.

MarkLogic

Built in Types

- Value
  - toObject();
  - valueOf();
- Value Iterator
  - for (var item of iterator) {…}
  - iterator.toArray()
- Nodes
  - Document
  - ObjectNode
  - XMLNode
  - Etc.

- Example:
  - “doc.json”

```
{
  “title”: “Hello World”,
  “author”: “MarkLogic University”
}
```

```
cts.doc(“doc.json”) instanceof Object; //TRUE

cts.doc(“doc.json”) instanceof Node; //TRUE

cts.doc(“doc.json”) instanceof XMLNode; //FALSE
```

MarkLogic

Nodes vs. Objects

- Nodes: What's in the database
  - Immutable (just like XQuery)
  - JSON (object, array, number, …)
  - XML (element, attribute, …)
  - Binary
  - Full Text
  - RDF
- Objects: What's in your code
  - Mutable: `obj.fullName = "Charles Dickens"`

MarkLogic

Nodes vs. Objects

```
fn.collection()  // ValueIterator
  .next()        // Iterate
  .value         // Document node
  .root          // ObjectNode (not required)
  .toObject();   // Returns plain-old object
```

MarkLogic

Example: A Basic Search

- Simple example of server side JavaScript in MarkLogic 8.

```
// Run in Query Console, or as a module in a *.sjs file

var mySearch = cts.orQuery([cts.wordQuery("cat"), cts.wordQuery("dog")]);

var searchResult = fn.subsequence(cts.search(mySearch), 1, 2);

searchResult;
```

MarkLogic

Example: Updates

```
declareUpdate(); // JavaScript updates must begin with this line!!

for(var item of fn.collection("accounts")) {
  var obj = item.toObject();
  obj.balance = obj.balance * 1.05;
  var collections = xdmp.documentGetCollections(item.nodeUri);
  xdmp.documentInsert(item.nodeUri, obj,
    xdmp.defaultPermissions(), collections);
}
```

MarkLogic

Invoking a SJS Module from Node.js

- Create a SJS module.
- Load SJS module into your project modules database.
  - Note: Requires **rest-admin** role:

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

MarkLogic

Invoking a SJS Module from Node.js

- Invoking a module requires a user with a role assigned the **xdmp-invoke** privilege.

```
{
  "message": "invoke code on server: cannot process response with 400 status",
  "statusCode": 400,
  "body": {
    "errorResponse": {
      "statusCode": 400,
      "status": "Bad Request",
      "messageCode": "SEC-PRIV",
      "message": "SEC-PRIV: Need privilege: http://marklogic.com/xdmp/privileges
/xdbc-invoke",
      "messageDetail": {
        "messageTitle": "Need privilege"
      }
    }
  }
}
```

MarkLogic

Invoking a SJS Module from Node.js

- Pass variables into the SJS module from your Node.js application.

```
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));
});
```

Demo: Server Side JavaScript

Labs: Unit 12

Exercise 1: Using Server Side JavaScript
Exercise 2: Install a Server Side JavaScript Extension
Exercise 3: Invoke a Server Side JavaScript Extension from Node.js

Unit Review Question 1:

A function that returns a value iterator enables you to:

1. Iterate over the results with a for loop.
2. Use the items .next() method.
3. Use the iterator objects .value property.
4. All of the above.

Unit Review Question 1:

A function that returns a value iterator enables you to:

1. Iterate over the results with a for loop.
2. Use the items .next() method.
3. Use the iterator objects .value property.
4. **All of the above.**

Unit Review Question 2:

To update a document in server side JavaScript code you must make it mutable using:

1. .toMutable()
2. .toValue()
3. .toObject()
4. .toPOJO()

Unit Review Question 2:

To update a document in server side JavaScript code you must make it mutable using:

1. .toMutable()
2. .toValue()
3. **.toObject()**
4. .toPOJO()

Unit Review Question 3:

Which is most correct:
In order to invoke a module from Node.js you must:

1. Connect as a user with the MarkLogic Admin role.
2. Connect as a user with a role that has the xdmp-invoke privilege.
3. Connect as a user with the rest-admin role.
4. Connect as a user with the rest-writer role.

Unit Review Question 3:

Which is most correct:
In order to invoke a module from Node.js you must:

1. Connect as a user with the MarkLogic Admin role.
2. **Connect as a user with a role that has the xdmp-invoke privilege.**
3. Connect as a user with the rest-admin role.
4. Connect as a user with the rest-writer role.