Node.js and RESTful API

by Matt Parker
for ACS 560

All demo codes available at github.com/mcparker76

I. What is Node.js (Node)?

A. Developed by Ryan Dahl in 2009.
B. JavaScript-based framework/platform built on Google Chrome's JavaScript V8 Engine.
C. Open Source and cross-platform.
D. Run server-side code using JavaScript.
E. All APIs in the Node library are asynchronous and nonblocking.
1. A Node server never waits for an API to return data.
2. Uses callback functions.

II. Installing and Using Node

A. Available for Windows, Mac, Linus platforms at nodejs.org.
B. To run Node applications from the command line: node *filename.*

III. Node Modules

A. Rich library of various JavaScript modules exist for simplifying development.
B. Modules are added to your installation using Node Package Manager (npm).
C. A require directive is used to load a module. An instance of the module can be assigned to a variable.
var http = require("http");
D. HTTP (http) for handling HTTP requests and responses.
E. File System (fs) for File I/O.
F. Express Framework to set up middlewares for responding to HTTP requests.
G. Body-Parser (body-parser) parses json requests.

IV. Creating a Server

A. Use HTTP module
B. Call createServer method to create a server instance.
C. Bind to port using listen method.
D. Pass createServer anonymous function with response and request parameters.

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

http.createServer(function (request, response) {

    //...
    response.writeHead(200, {'Content-Type': 'text/plain'});

    // Send the response body as "Hello ACS560!"
    response.end('Hello ACS560!\n');
}).listen(8080);

console.log('Server running at http://127.0.0.1:8080/');
```

V. More on Callbacks

A. Asynchronous equivalent for a function.

B. Called at the completion of a given task.

C. All APIs of Node are written to support callbacks.

D. Allows Node to process high number of requests without waiting for any function to return its result.

D. Text File Example

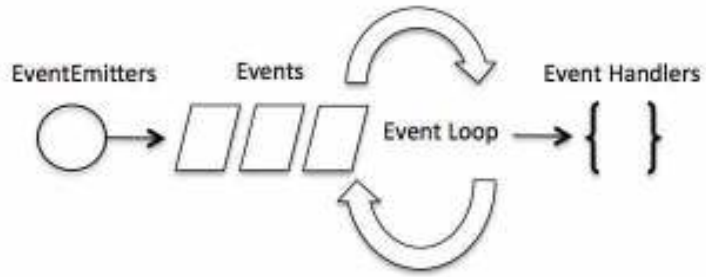
1. A function to read a file may start reading file and return control to execution environment so that the next instruction can be executed.
2. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter.
3. Consequently, there is no blocking (or wait) for file I/O.

VI. Node Event Loop

A. Node is a single threaded application.

B. Supports concurrency through events and callbacks (asynchronous functions).

C. Observer pattern is used. Node thread keeps an event loop; when a task is completed, the corresponding event signals the event listener function to get executed.



Source: HTTP://www.tutorialspoint.com/nodejs/nodejs_event_loop.htm

D. For the following example, readFile passes err and data to callback function after file read operation is complete.

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

fs.readFile('input.txt', function (err, data) {
    if (err) return console.error(err);
    console.log(data.toString());
});

console.log("Program Ended");
```

VII. REST architecture

A. REST stands for REpresentational State Transfer.

B. Web standards based architecture and uses HTTP protocol.

C. Resources are accessed by a common interfrace using HTTP standard methods

D. Various representations can be used for resources including XML and JSON.

E. Commonly used HTTP methods include GET, PUT, DELETE, POST

1. GET: provide read-only access
2. PUT: create new resource
3. DELETE: remove resource
4. POST: Update existing resource

F. A web service based on REST architecture is known as a RESTful web service.

VIII. Implementing REST in Node

A. Use Express module to route HTTP requests and render HTML views

var app = require("express");

B. app.get(path, callback);

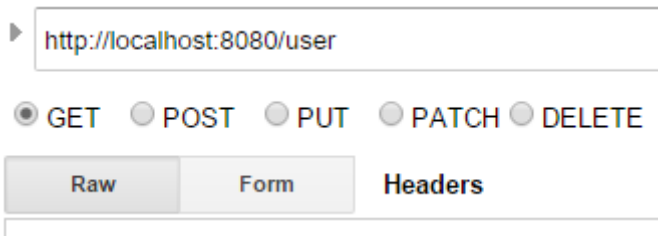
Routes HTTP GET requests to path with specified callback function(s).

C. app.put( )

D. app.post( )

E. app.delete( )

F. To use these methods, the appropriate http request must be made from the client. The Advanced Rest Client extension for Chrome can be used for testing these methods.



RESTful API for Creating, Reading, Updating, Deleting User Information

```
var express = require('express');
var app = express();
var fs = require("fs");
var bodyParser = require('body-parser');

app.use(bodyParser());

app.delete('/user/:id', function (req, res) {
    fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
        var users = JSON.parse( data );
        delete users["user" + req.params.id];
        console.log( users );
        res.end( JSON.stringify(users));
    });
});

app.get('/user', function (req, res) {
    fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
        console.log( data );
        res.end( data );
    });
});

app.put('/user', function (req, res) {
    fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
        var users = JSON.parse( data );
        var aUser = req.body;
        console.log( aUser );
        users["user4"] = aUser;
        res.end( JSON.stringify(users));
    });
});
app.post('/user/:id', function (req, res) {
    fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
        var users = JSON.parse( data );
        users["user" + req.params.id] = req.body;
        console.log(users);
        res.end( JSON.stringify(users));
    });
});

app.get('/user/:id', function (req, res) {
    // First read existing users.
    fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
        var users = JSON.parse( data );
        var user = users["user" + req.params.id];
        console.log( user );
        res.end( JSON.stringify(user));
    });
});

var server = app.listen(8080, function () {
    var host = server.address().address;
    var port = server.address().port;
    console.log("Example app listening at http://%s:%s", host, port)
});
```