## Overview

Node.js is the back end for the front end instead of Java/C++/Csharp.

In convention languages worker process is blocked by I/O. So even if we are multiple threads, concurrency is impacted if the there are several I/O operations

In Node.js the event loop is non-blocked and even though its single threaded it can achieve higher concurrency.

In Node.js you have to build your stack ground up. For example, you build the webserver from scratch.

REPL (Read, Eval, Print, Loop) environment is bundled with Node.js. Use “node” to invoke the REPL environment.

Node JS goes beyond the standard Request/Response paradigm and allows Web application to have Real time 2-Way connection, by employing push technology over web sockets over port 80. Node.JS does not require no sandbox like Flash/Applets

Node.JS is implements non-blocking event driven I/O. Hence it can handle a large number of simultaneous connections with high throughput. A single thread can manage a large number of connections. It shines in building fast scalable N/W applications

Uses: Chat, API/Object DB, Queued Inputs, Data Streaming, Proxy, Trader Dashboard, App Monitoring

* Data Streaming: Real time video encoding. Processing a file as its being uploaded
* Proxy: Large number of concurrent requests pulling data from different sources

Node.js cannot be used for CPU intensive operations. Since, it only uses a single thread it cannot take advantage of Multi CPU/Core. Hence, it is not the best platform for heavy computation CPU. You don’t want to build a Fibonacci computation server. Also, unhandled exceptions can cause the entire application to abort as there is only a single thread.

Node.JS was not created to solve CPU/CPU scaling problems. It was created to solve the I/O solving problem

Node.js may not be SEO friendly as it is designed for Single Page applications using websockets

With traditional relational DBs, Node.JS does not work as well as ASP.NET or Ruby on Rails

## Modules

Module - is library or object containing many reusable functions.

Module can import other modules.

Node.Js Objects

new XYZ() - creates an object instance.

1. module.exports =

. module.exports = anonymous/named and can export only one.

call using requirevarname

instantiate using required varname

exports.functioname =

call using requirevarname.functioname()

exports = named multiple

module.exports = internalfunction/construnctornamename

privatename.prototype.publicmethod name = function()

object name = function

set properties of object

define more functions in the object (prototype)

module.exports = new Object

call using requirevarname.requiredfunction.

var inside a module is private wherease exports is public..

export a "class", "object" or property.

var configuration ??? -- private,global variable in module but common/shared by all users.

//constructor.. keep it small for performance reasons.

function tkapiconsumer(credential) {

this.credential = credential }

//externalise(prototype) and call connection from here.

this.tkconsumer = Oauth(uri,configuration,credential);

log('connected')

}

tkapiconsumer.prototype.get = function (uri) {

this.tkconsumer.get(uri,config)

}

1 module.exports = tkapiconsumer //-export, named but name is internal, becomes constructor

tkapiconsumer = require(tkapis) //call (external/require\_varname

tkconsumer = new tkapiconsumer(credentials) ---> connects and keeps consumer as private. Need to be passed to other modules

tkconsumer.prototype.connect(

tkconsumer.prototype.get(uri,callback)

tkconsumer - returned as call back?

view,configs,routes

./Config/Config.js

Load based on Dev.Json or Prod.json using App environment variable

Implement APIs in such a way that they

return control immediately and then "callback" with results sent as parameters to the call back function.

## Express

Router: One route for each URI

express app makes use of callback function whose parameters are the request and response object.

res.json

res.render

\_dirname

setTimeout(cb,ms)

setInterval(cb,ms)

define routes in a JSON object

Next ensures that express looks for other routes.

app.use() called always. Move to beginning. Session Management, Body Parser

## Router Middleware

Middleware is a functions with access to Request and Response Objects

A middleware must end the current request-response cycle or call the "next" middleware otherwise the request will be left hanging.

req.parms.id gives value in :id

Define your middleware module and start using it with app.user() --> will get executed for every request response.

req.path

req.parms.id

req.query.q

request - nodejs library to make requests to other sites.

Impement one route for each URI. Use a JSON document to create routes.

## Hogan Template Engine

Express –hogan

app.engine('html', require('hogan-express'));

app.set('view engine', 'html');

app.set('layout', 'layouts/default');

app.set('partials', {header: "includes/header"});

Layouts : Parent Templates

Partials: Include Templates

Lambdas: Custom Filters

he first line will wrap all your further hogan views within this express with this default layout file, the actual view content will fall wherever you put the

{{{ yield }}}

clause in the[layouts/default.html](https://github.com/mgenev/MEAN/blob/master/app/views/layouts/default.html)code. The second line will make the header partial available for inclusion in any view whenever fit with the following syntax:

{{> header}}

That’s almost it, you are at this point able to pass dynamic data from the [Express controller](https://github.com/mgenev/MEAN/blob/master/app/controllers/users.js)to the HTML view like this:

exports.show = function (req, res) {

var user = req.profile

res.render('users/show',layout: ‘mylayout’, {

title: user.name,

user: user

})

}

And render it in the view with the Mustache syntax:

Hello, {{user}}, the title is {{title}}.

Layout 🡪 Page ({{yield}}}

Layout can “yield” whole or sections of ‘page’ {{{yield-tag}}}

Layout can include default or explicit partials (e.g. headers, footers or scripts) – These are reusable include templates

Page can include default of explicit partials {{ >partial }} or locals

Page can have different sections to be yielded in different sections of the Layout

## Resources/Modules

1. Express
2. NPM
3. Passport.js: Helps login via FB, Twitter
4. MongoDB as Service
5. Mongoose
6. Socket.IO
7. Heroku PAAS
8. Graphics, Magick
9. Amazon, CloudFront
10. Sublime
11. Routes
12. Views
13. Config
14. Data
15. Static/Public
16. Knox to talk to S3
17. Formidable
18. Putty/Puttygen
19. Filezilla
20. URLModule
21. PostMan Chrome Plugin: Postman is a Web REST client that allows you to enter and monitor HTTP requests and responses
22. QueryStringModule

Develop an instinct to select the correct Node.js module from NPM/GitHub

github stats - stars, has it been updated recently. -

## Events

* Event Emitter: e.g. HTTPServe emits “req” event. Request is readable stream
* Event Listener e.g req.on(‘data’, cb)
* ‘data’ is the event

## Socket

<http://socket.io/docs/>

|  |  |
| --- | --- |
| Server/App = require('express').createServer() | Client |
| var io = require('socket.io')(app); | /socket.io/Socket.io.js  Jquery (for event listeners) |
| //io = (socket.io).listen(server)  app.listen(port); |  |
| var chatroom = io.of(ns).on(‘connection’,function(socket){ socket.emit(a private message that only connected user will get will get) })  io.emit(received by everyone)  chatroom.emit(a message that everyone in chat will get) | Chatroom = io.connect(host + namespace) e.g /roomlist |
|  | Chatroom.on(‘connect’, function (){  Chatroom.emit(Test Hi)}); |
|  | /\*Create New Room\*/  Socket.emit(‘newroom’) |
| Socket.on(‘newroom’, function (){  Room.push  /\*Broadcast to everyone else except for the socket that starts it \*/  Socket.broadcast.emit(“room update”, JSON.stringify(room)} |  |
|  | Socket.on(room update, JSON.parse(data))  Href = “rooms/number” |
| .of(/messages).on(“connection” , function(socket){  Socket.on(joinroom,function(data) {  **Socket.join**(data.room);  Socket.to(room).emit(‘updateuserlist’,JS.s(userlist));  }  Socket.on(newmessage,function(data) {  Socket.broadcast.**to**(data.room\_number).emit(  messagefeed,JSON.stringify(data);  }  }) | Messages = .connect( /messages) |
| Set interval to updateuserlist | Message.on(‘connect’, function() {  Connection established  Message.emit(joinroom,room numberanddata)  } |
|  | $(document).on(keyup,.newmessage,function(e) {  If(e.which == 13){  Messages.emit(‘newMessage’,datawithmessage)  }  } |
|  | Messages.on(‘messagefeed’, function(data) {  Mgs = JSON.parse(data);  } |
|  | Messages.on(‘updateuserlist’, function(data){  }) |

Namespace has the benefit of multiplexing a single connection. Instead of socket.io using two WebSocket connections, it’ll use one.

## Node.JS at Citi

* DevCloud
* IDE Debugger,Node Inspector, Node Eclipse
* Hermes, RTC,
* Open Source Look Module for monitoring
* NGIX (Load Balance) A,B
* CitiWebApplication toolkit