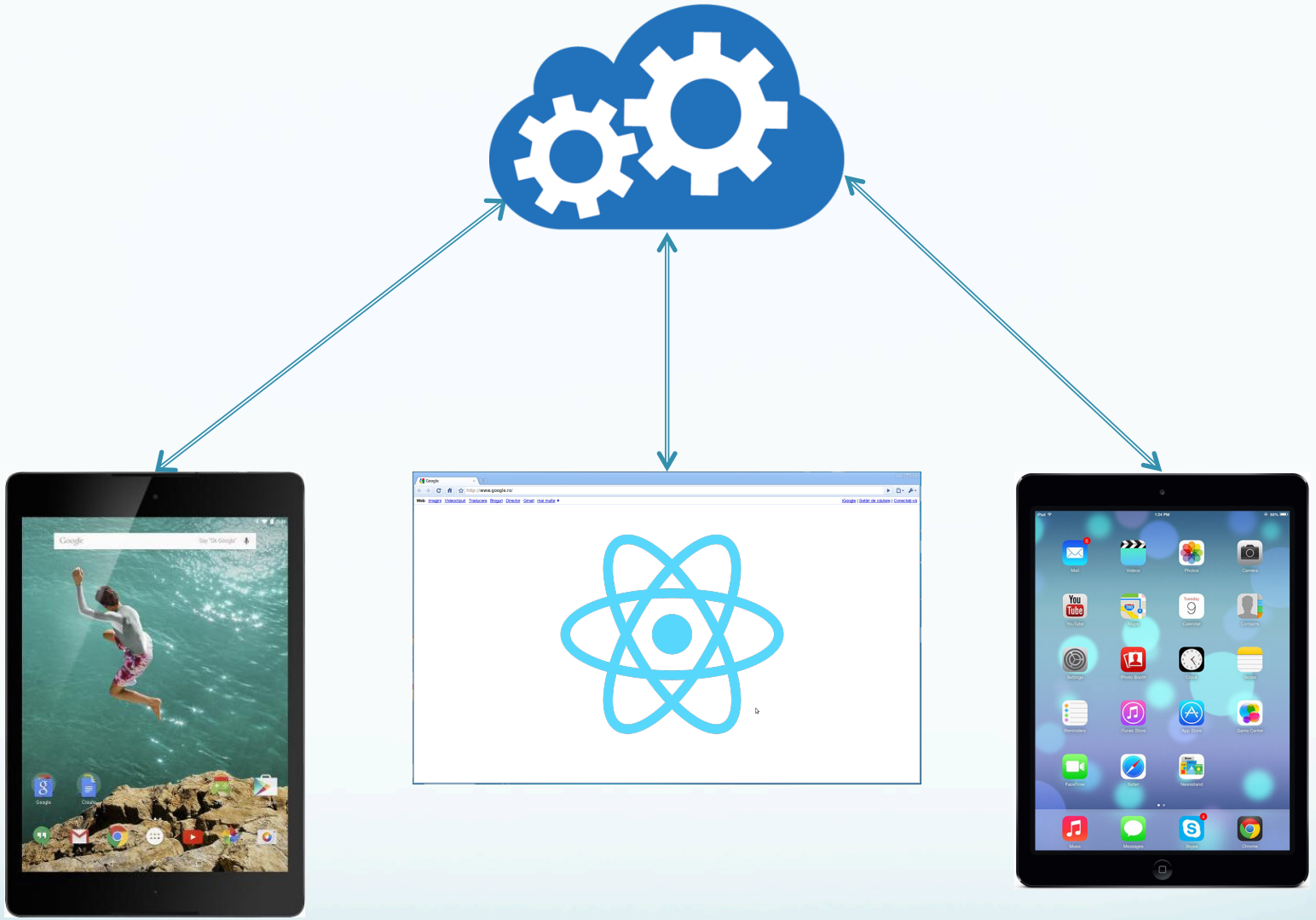
Building APIs in NodeJS

John Carmichael
Director of Operations – WeSpire
john@wespire.com
@JohnnyC115

The What and Why of API

- API – Application Program Interface
- Used to expose the functionality of an application to external actors
- Can be used to allow client applications to access an independent server side application (ex. Twitter's web app, mobile app and Periscope all access the same backend API)
- Can be used to allow partner applications to access (ex. Tweetdeck and Meerkat access Twitter's backend API)



SOAP vs REST

- SOAP – Simple Object Access Protocol
  - Utilizes own protocol
  - Focuses on exposing app logic (not data) via named operations (ex. getUser(User);)
- REST – Representational State Transfer
  - Utilizes existing well known HTTP
  - Focused on CRUD (create/read/update/delete) operations on data

HTTP Methods

- GET – Requests a representation of the specified resource (data retrieval)
- POST – Requests that the server accept the entire enclosed in the request as a new subordinate of the web resource identified by the URI (ex. a message for a bulletin board or an item to add to a database).
- HEAD – Asks for the response identical to the one that would correspond to a GET request, but without the body.
- PUT – Requests that the enclosed entity be stored under the supplied URI.
- DELETE – Deletes the specified resource.
- TRACE – Echoes back the received request so that a client can see what changes have been made by intermediate servers.
- OPTIONS – Returns the HTTP methods that the server supports for the specified URL.
- CONNECT – Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication through an unencrypted HTTP proxy.
- PATCH – Applies partial modifications to a resource.

HTTP Methods

- **GET – Requests a representation of the specified resource (data retrieval)**
- **POST – Requests that the server accept the entire enclosed in the request as a new subordinate of the web resource identified by the URI (ex. a message for a bulletin board or an item to add to a database).**
- HEAD – Asks for the response identical to the one that would correspond to a GET request, but without the body.
- PATCH – Applies partial modifications to a resource.
- **PUT – Requests that the enclosed entity be stored under the supplied URI.**
- **DELETE – Deletes the specified resource.**
- TRACE – Echoes back the received request so that a client can see what changes have been made by intermediate servers.
- OPTIONS – Returns the HTTP methods that the server supports for the specified URL.
- CONNECT – Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication through an unencrypted HTTP proxy.

HTTP Status Codes

- Informational – 1xx
- Successful – 2xx
- Redirection – 3xx
- Client Error – 4xx
- Server Error – 5xx

HTTP Status Codes

- Successful – 2xx
  - 200 OK – The request succeeded
  - 201 Created – The request has been fulfilled and resulted in a new resource being created

HTTP Status Codes

- Redirection – 3xx
  - 301 Moved Permanently – The requested resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs.
  - 302 Found – The requested resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client SHOULD continue to use the Request-URI for future requests.

HTTP Status Codes

- Client Error – 4xx
  - 400 Bad Request – The request code not be understood by the server due to malformed syntax.
  - 401 Unauthorized – The request requires user authentication.
  - 403 Forbidden – The server understood the request but is refusing to fulfill it. The request should not be repeated.
  - 404 Not Found – The server has not found anything matching the Request-URI.
  - 405 Method Not Allowed – The method specified in the Request-Line is not allowed for the resource identified by the Request-URI. The response MUST include an Allow header containing a list of valid methods for the requested resource.
  - 415 Unsupported Media Type – The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

HTTP Status Codes

- Server Error – 5xx
  - 500 Internal Server Error – The server encountered an unexpected condition which prevented it from fulfilling the request.
  - 501 Not Implemented – The server does not support the functionality required to fulfill the request.
  - 502 Bad Gateway – The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.
  - 503 Service Unavailable – The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
  - 504 Gateway Timeout – The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (e.g. HTTP, FTP, LDAP) or some other auxiliary server (e.g. DNS) it needed to access in attempting to complete the request.

How Do HTTP?

- Request

```
GET /student?name=Andrew%20Borstein&cohort=Spring2015 HTTP/1.1
Host: siber.com
```

```
POST /student HTTP/1.1
Host: siber.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 53

name=Andrew%20Borstein&cohort=Spring2015
```

How Do HTTP?

- Response

```
HTTP/1.1 200 OK
Date: Mon, 3 Aug 2015 19:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Fri, 31 Jul 2015 12:17:09 GMT
ETag: "3f80f-1b6-3e1cb03b"
Content-Type: text/html; charset=UTF-8
Content-Length: 110
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>HTTP - Response</title>
</head>
<body>
  Oooooo look at all this HTML!!!!
</body>
</html>
```

cURLing Me Softly

- cURL is a tool for client-side URL transfers which supports multiple protocols including HTTP

```
curl http://jsonip.com
```

- Common Options
  - X – Sets custom request method
  - d – Passes data in the request body
  - H – Sets a header value
  - i – Include the HTTP Headers in Output

And Then There Was Node

- JavaScript was initially only available in browsers
  - Native Objects: String, Array, Date, etc.
  - Host (Browser) Objects: Window, Document, XMLHttpRequest, etc.
- Google Chrome's V8 Engine
- NodeJS created by Ryan Dahl
- New Host Objects: http, fs, url, os, etc.
- Atwood's Law: Anything that can be written in JavaScript eventually will be.

```
1  var http = require('http');
2
3  var server = http.createServer();
4
5  server.on('request', function(request, response) {
6    var body = "";
7
8    if(request.method == 'POST') {
9      request.on('data', function(data) {
10       body += data;
11     });
12
13     request.on('end', function() {
14       var json = JSON.parse(body);
15
16       response.writeHead(200);
17       response.write("<h1>Hello " + json['name'] + "!</h1>");
18       response.end();
19     });
20   }
21   response.writeHead(200);
22   response.write("<p>Hello Joe!</p>");
23   response.end();
24 }).listen(1337);
25
26 console.log('Listening on port 1337...');
```

A Better Way

- ExpressJS – A fast minimalist web framework for NodeJS

```
1  var express = require('express');
2  var bodyParser = require('body-parser');
3
4  var app = express();
5  var parser = bodyParser.json();
6
7  app.post('/', parser, function(request, response) {
8    response.send("<h1>Hello " + request.body.name + "!</h1>");
9  });
10
11 app.listen(1337);
12 console.log("Listening on port 1337...");
```

Don’t Go It Alone

- Middleware – any number of functions that are invoked by the Express routing layer before your final request handler

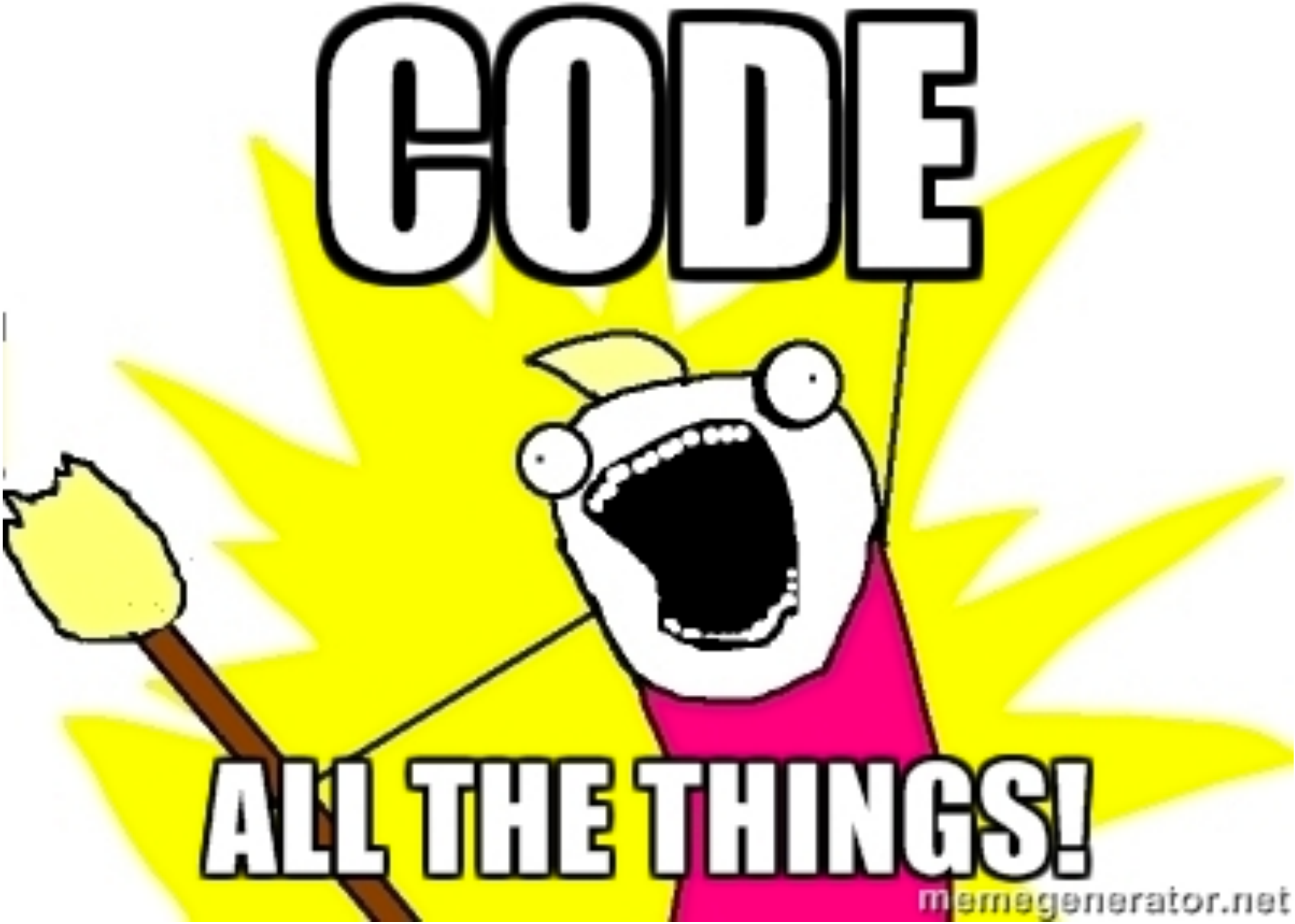
```
1  var express = require('express');
2  var bodyParser = require('body-parser');
3
4  var app = express();
5  var parser = bodyParser.json();
6
7  app.post('/', parser, function(request, response) {
8    response.send("<h1>Hello " + request.body.name + "!</h1>");
9  });
10
11 app.listen(1337);
12 console.log("Listening on port 1337...");
```

We Don't Need No Stinking Packages

- NPM – Node Package Manager
  - Comes standard with Node. Allows you to easily install extra packages

```
npm install --save express
```

```
{
  "name": "node_api_class",
  "version": "1.0.0",
  "description": "Project used in teaching the 'Building APIs in NodeJS' class
  "private": false,
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/jlcarmic/node_api_class.git"
  },
  "author": "John Carmichael",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/jlcarmic/node_api_class/issues"
  },
  "homepage": "https://github.com/jlcarmic/node_api_class",
  "dependencies": {
    "express": "^4.12.0"
  }
}
```


CODE
ALL THE THINGS!
memegenerator.net

Review

- APIs allow us to expose application logic to external actors.
- REST leverages pre-defined HTTP to create data driven APIs.
- NodeJS gives us JavaScript outside of the browser which allows for server-side coding as we just did.
- Express sits on top of Node and gives us faster/ cleaner access to HTTP requests and responses.

Review

- Middleware allows us to preprocess the request before our function runs.
- There are many packages available to extend Node's functionality such as Sequelize for interacting with SQL databases.
- NPM allows us to easily manage all the packages we need for our project.