WHO AM I?

- A software engineer and entrepreneur
- On the tablet team at Kobo
- Node.js Developer for ~1 year
- Find me blogging at FaisalAbid.com
- Tweeting @FaisalAbid

A few things I have worked on.

Node.js & **You**

Part 1:
WTF is Node?

WTF is Node?

- A powerful platform to let you run JS on the server side.
- How? Uses Google’s V8 Engine
- V8 is built in C.
- V8 is the fastest JS engine on the planet!
- Great way to build modern web apps in JS on both client
and server side!

What Can I Do in Node?

- Anything you want!
- Chat servers, Analytic servers & Crazy fast backends
- Socket.io library is a wicked way to build real time apps
- Seriously, anything you want.
- Build a social network! LinkedIn, Dropbox all using

Node.js

What Can’t I Do in Node?

- Contradicts previous slide but
  - Node.js is not a web framework.
    - Modules for Node.js make it into a web framework. I.e Express
- Node.js is not Multi-threaded.
  - A single thread to rule em all.

Single threaded?!?!

- Node.js is build around “events”
- The event loop is always running checking for new events and fires callbacks.
- But Faisal, “Single threaded = blocking”. Won’t my server hang up?
- Yes, if you have blocking code. Non-blocking code will make Node.js fly.

Non-Blocking? Blocking? Im so confused

- Don’t worry, I was too.
- You are use to writing blocking code. I.e PHP, CF, any server language thats multi-threaded.

Blocking Pseudo-Code Example

```
int registerUser(username,password){
    userID = registerUser(username,Hash(password));
     return userID;
}

userID = registerUser(“faisal”,”FaisalIsSoCool”); // Don’t tell anyone. this is my bank pass
output(“Your userID is ”+userID);
```

Non-Blocking? Blocking? Im so confused

- In a multi-threaded system, this would run just fine.
- In Node.js this will run just fine.
- **FOR ONE USER AT ONE TIME**
- All other users will have to wait till the previous user is registered, so they can get registered.
- What an awesome platform Your app can support only one user doing something at one time!

Non-Blocking? Blocking? Im so confused

- How would we make it work **properly** in Node.js?
- By writing it as asynchronous code. I.e using Callbacks

Blocking Pseudo-Code Example

```
registerUser(username,password,callback){
    userID = registerUser(username,Hash(password));
    callback(userID);
}

registerUser("faisal","FaisalIsSoCool",function(userID){
      output("Your userID is "+userID);
});
```

Non-Blocking? Blocking? Im so confused

- By introducing callbacks, Node can move on to other requests and whenever the callback is called, node will process is.
- You should read non-blocking code as “put function and params in queue and call callback when you reach the end of the queue.
- Blocking = return. Non-blocking = no return. Only callbacks. *(well we can use other stuff, but we will get into that!)*

Part 2:
Node.js **Event Loop**

Node.js runs on the event loop

- The event loop keeps on running. Checking for new events, so it can call callbacks for those events.
- Lets take a look at an example.

Event Loop Example

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

var server = http.createServer(function(request,response){
    response.writeHead(200); // HTTP status
    response.write("Hello Web Unleashed");
    response.end();
});

server.listen(8080);
```

Event Loop Example

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

var server = http.createServer(function(request,response){
    Callback to use . Contents will be executed when HTTP request event happens
    response.writeHead(200); // HTTP status
    response.write("Hello Web Unleashed");
    response.end();
});

server.listen(8080);
```

Event Loop Example

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

var server = http.createServer(function(request,response){
    response.writeHead(200); // HTTP status
    response.write("Hello Web Unleashed");
    response.end();
});
```

server.listen(8080); Sets up to listen to network events.

Event Loop Example

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

var server = http.createServer(function(request,response){
 Which executes this function when network events happen
     response.writeHead(200); // HTTP status
     response.write("Hello Web Unleashed");
     response.end();
});

server.listen(8080); Sets up to listen to network events.
```

Node.js runs on the event loop

- The event loop keeps on running. Checking for new events, so it can call callbacks for those events.
- Events are processed one at a time. Callbacks are fired for those events.
- This makes Node.js awesome. Setup events to listen to, and Node.js will do other things till that event comes in.

Part 2.1: Using **Events**

Using Events

- So we saw in our example, whenever an HTTP request event is fired, our callback is called.
- Node.js has a LOT of events. Lots and Lots!
- Lets see some in action

Events Example

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

var server = http.createServer();

server.on('request',function(request,response){
      response.writeHead(200); // HTTP status
      response.write("Hello Web Unleashed");
      response.end();
});

server.on('connection',function(socket){
   console.log("New Connection");
});
server.listen(3030);
```

Using Events

- So we saw in our example, whenever an HTTP request event is fired, our callback is called.
- We rewrote our previous example to using events only. Gives us finer control of whats going on with the HTTP server.

Using Events

- How would we write custom events?
- We use the awesome EventEmitter class.
- Lets see how.

Events Example

```
var ee = require('events').EventEmitter;
var unleashed = new ee();


unleashed.on("start",function(){
        console.log("open the doors");
});

unleashed.on("end",function(wasItGood){
        if(wasItGood){
                console.log("YAY");
        }else{
                console.log("Bummer");
        }
});

unleashed.emit('start');
unleashed.emit('end',false);
```

Using Events

- EventEmitter “emits” events.
- Imagine these events being called when a user registers,
or a file is uploaded.
- You can easily call functions when this event happens.
- Prevents nested callback hell also.

Part 3:

Streams

Using Streams

- Node.js loves Streams.
- Streams are an easy way to interact with data, in a streaming manner.
- Imagine uploading a file and saving it piece by piece to disk. No need to preload data in RAM.
- You’ve been using streams already. HTTP response is a stream. Which is why you call response.end() to “close” the stream.

Using Streams

- Lets create a server which streams a file from the server to the client.

Streams

```
var fs = require('fs');
var http = require('http');

var server = http.createServer();

server.on('request',function(request,response){
      fs.createReadStream("/etc/hosts")
            .on('data',function(data){
                  response.write(data+"\n");

      }).on('end',function(){
             response.end();
      });
});

server.listen(3030);
```

Using Streams

- Lets create a server which streams a file from the server to the client.
- So we listen to the HTTP request event. When the request happens, we create a readStream to /etc/hosts and on each chunk of data, we stream it to the response stream.
- Once all the streams are loaded, we close the response stream!

Using Streams

- So wait, Faisal, Why do this?
- Because streaming loads a very small amount of data into ram, rather then loading it all up into memory then sending it to the browser.
- You might notice that the code seems like its a lot for something so simple.
- Lets simplify it!

Streams

```
var fs = require('fs');
var http = require('http');



var server = http.createServer();

server.on('request',function(request,response){
      var file = fs.createReadStream("/etc/hosts");
      file.pipe(response);
});


server.listen(3030);
```

Using Streams

- Woah, what happened here?
- Instead of listening to ‘data’ and ‘close’, we “pipe” one stream to the other stream.
- Very easy and useful method!

Using Streams

- One of the hardest problems on the web is surprisingly uploading files.
- Many many ways exist in CF, PHP etc etc. But Node.js has such an elegant solution.
- Lets take a look!

Streams

```
var fs = require('fs');
var http = require('http');

var server = http.createServer();

server.on('request',function(request,response){
      var file = fs.createWriteStream("file.txt");
      request.pipe(file);

      request.on('end',function(){
        response.write('uploaded!');
        response.end();
      });
});

server.listen(3030);
```

Using Streams

- Woohoo! Thats so cool.
- Pipe just takes in the request and saves it. Holy cow thats cool.
- What if we want to see progress of the upload?

Streams

```
var fs = require('fs');
var http = require('http');

var server = http.createServer();

server.on('request',function(request,response){
        var file = fs.createWriteStream("file.txt");

        request.on('data',function(data){
                file.write(data);
                console.log("Writing \n");
        });

        request.on('end',function(){
                file.end();
                console.log("end");
                response.end();
        });
});

server.listen(3030);
```

Using Streams

- Not using Pipe here, since we want to track the events. Still an awesome solution to show progress. We can get the size of the request and do some % progress too.
- However, one thing to consider is your disk might be slower then the speed of the upload.
- “Backpressure!” this might cause weird glitches and missing chunks in file.
- Pipes() does solve this, but lets see how we would do this without Pipes().

Streams

```
request.on('data',function(data){
 var good = file.write(data);
      if(!good){
            request.pause();
      }
      console.log("Writing \n");
});

file.on('drain',function(){
      request.resume();
})

request.on('end',function(){
      file.end();
      console.log("end");
      response.end();
});
```

Using Streams

- file.write() will return false if it can’t currently write and the buffer is full.
- If its false, well pause the request stream till the file write stream “drains”
- once it drains we can resume away.
- Again, pipe does it behind the scenes, but the core basics are important here!

Part 4:

Modules

Using Modules

- You might have noticed we’ve been using require(“”) a lot.
- Require is basically a way to “import” modules to your application. Modules are basically “classes”.
- They are a module of code that contain functions which have been “exported”
- Exported functions are basically “public”.

Modules

- Phew. Lots of terminology there.
- Don’t worry. Things will get clearer soon.

Modules

```
var http = require(“http”) // http is a module thats being imported to this file
```

Using Modules

- So where the hell is http anyways?
- Well my friend, that stuff is stored where node.js is installed.
- What Node does is first looks in a node_modules folder (well learn about it in a few slides)
- Then it looks in a node_modules folder in your home folder “~”
- Then it looks where Node is installed. This is where http is.

Create Modules

- What does require return then? the file?
- Well no, when you require a module. It returns a JS

Object. In require(“http”) it returns the HTTP object.

- This has functions you can call, as well as public variables.
- This is basically a class guys & gals. Don’t sweat it.
- Lets see how a module looks to get a better idea.

Modules

```
exports.somePublicMethod = function(){

};

exports.someOtherPublicMethod = function(){

};

function somePrivateMethod(){

}

var someRandomPublicMethod = function(){

};

exports.randomName = someRandomPublicMethod;
```

Modules

```
var unleashed = require("./unleashed");

console.log(unleashed);


/*

{ somePublicMethod: [Function],
  someOtherPublicMethod: [Function],
  randomName: [Function] }

*/
```

Create Modules

- We declare public methods by putting it in exports.
- If a method is not in exports, then it is private.
- This is a great way to do OOP.
- Modules can require modules and have dependency
hierarchies.

Why Modules?

- Modules make it very easy to add new functionality to your app.
- The core of node is small and efficient. It is how it should be.
- But the public “userland” support is massive and superb.
- You will guaranteed use at least one third party module in a Node.js app.
- Heck to run these samples, I’ve been use nodemon!

How do I get Modules?

- How do you get modules then? Do you download them like jar files?
- Nope! You use a trusty command called “npm”.
- The node package manager.
- npm.org has links to every single Node.js module you want.
- Lets see it in action.

Modules

```
npm install nodemon

npm install -g coffee-script
```

Dependencies

- You will have noticed when you installed a module, it download a bunch of other modules also.
- These are dependecies for that module. Just like your app is dependent on module a, module a is dependent on module x,y,z.
- Defining dependancies is a great way to keep the project organized.
- Dependencies are defined using package.json

Package.json

```
{
  "name": "Hello Web",
  "preferGlobal": "true",
  "version": "0.0.1",
  "author": "Faisal Abid <faisal.abid@gmail.com>",
  "description": "Sample app for web unleashed",
  "repository": {
    "type": "git",
    "url": "https://github.com/FaisalAbid/WebUnleashedNodeWorkshop.git"
  },
  "dependencies": {
    "express": "~3.0.0rc5",
    "hbs": "~1.0.5",
    "mongoose": "~3.3.1",
  }
}
```

Package.json

- Every app **should** have a package.json. Its not required, but Its best practice.
- Define your app, github repo, version etc.
- Define dependencies.
- But why? Whats the point?
- You can easily upload your app, or commit it to github without passing your node_modules file.
- Then when you say checkout your project on your server, you can run npm install and npm will go ahead and download all your packages for you

Package.json

- Thats cool Faisal, but I hate always adding lines to package.json. Is there no easy way??
- Yes there is!

Package.json

```
npm install --save socketio
```

Package.json

- Thats cool Faisal, but I hate always adding lines to package.json. Is there no easy way??
- Yes there is!
- -- save flag automatically **updates** (not create and update) your package.json file
- Remember, it only updates the file. Does not create it. If package.json doesn’t exist, then it will ignore it.

Part 5:

Express.js

What is Express?

- So earlier, I told you that Node.js is not a web framework.
- Technically thats correct, but there are awesome modules for Node that make it into a web framework.
- One of the most popular modules, that you will undoubtedly use is Express.js
- Its sinatra inspired. But I don’t care. I hate ruby.
- Whats cool about Express is that its very poweful, very performant and most of all very very simple to use.
- Lets take a look.

Express.js

```
var express = require('express');
var app = express();

app.get('/', function(request, response){
  response.send('Hello World');
});

app.get('/conference/:name',function(request,response){
       response.send(request.params.name);
});
app.listen(8080);
```

What is Express?

- Very similar to what we were doing before. But with very little code.
- Lets see how this example would look without express.

Express.js

```
var express = require('express');
var app = express();

app.get('/', function(request, response){
  response.send('Hello World');
});

app.get('/conference/:name',function(request,response){
       response.send(request.params.name);
});
app.listen(8080);
```

Without Express.js

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

var server = http.createServer(function(request,response){
    if(request.url == "/"){
        response.writeHead(200); // HTTP status
        response.write("Hello World");
        response.end();
    }else if(request.url.indexOf("conference") > -1){
        var n = request.url.split('/')[2]
        response.writeHead(200); // HTTP status
        response.write(n);
        response.end();

    }else{
        // do some other stuff
    }
});
server.listen(8080);
```

What is Express?

- Wow thats messy and hard and error prone.
- Express encapsulates all this in a great API.
- Lets see another example, this time lets send an html file

instead of just text.

Without Express.js

```
var express = require('express');
var app = express();

app.get('/', function(request, response){
  response.sendfile(__dirname+"/public/index.html");
});


app.listen(8080);
```

What is Express?

- This is cool. Were starting to see how we can build websites using Express and Node.
- Lets go a step further and add some real live data.
- Lets build a simple twitter feed viewer.

Without Express.js

```
var express = require('express');
var app = express();
var request = require('request');
var url = require('url');

app.get('/', function(request, response) {
        response.sendfile(__dirname + "/public/index.html");
});

app.get('/twitter/:username', function(req, response) {
        var username = req.params.username;
        options = {
                protocol: "http:",
                host: 'api.twitter.com',
                pathname: '/1/statuses/user_timeline.json',
                query: {
                        screen_name: username,
                        count: 100
                }
        }
        var twitterUrl = url.format(options);
        request(twitterUrl).pipe(response);
});
app.listen(8080);
```

What is Express?

- Okay, so now I see JSON everywhere?
- Yup, You are seeing the Raw output of the data. Notice we used Pipe to pipe the request stream from twitters API to the response stream.
- Lets see how we can format this properly

Part 5.1:
Express.js **Templates**

Express Templates

- Previously we saw how to display data using response.send().
- Now were going to see how to properly parse that data and make it look pretty, by using templates.
- Templates are a great way to separate your views from your controller and model.
- Express has great support for templates for a wide varity of style.
- Their is Jade, EJS, Dust (LinkedIn now basically owns Dust), and my favorite Handlebars

Express Templates

- Previously we saw how to display data using response.send().
- Now were going to see how to properly parse that data and make it look pretty, by using templates.
- Templates are a great way to separate your views from your controller and model.
- Express has great support for templates for a wide varity of style.
- Their is Jade, EJS, Dust (LinkedIn now basically owns Dust), and my favorite Handlebars

Express Templates

- Handlebars offers a sweet and simple syntax to populate your templates.

Handlebars.js

```
<title>Hello</title>
<h1>{{variableName}}</h2>
```

Express Templates

- VariableName is replaced with your data you pass from Express.
- Theres 3 steps to making Express work with templates.
- 1. npm install the view engine. In our case its hbs
- 2. Tell express you want to use this view engine “hbs”
- 3. Setup the directory where all your views are

Handlebars.js

```
var express = require('express');
var app = express();
var hbs = require('hbs');


// configure expresss. This is an important step.
// set our view engine to hbs. Handlebars.js
// now express will automaticly look for all .hbs files in the directory views
app.set('views', __dirname + '/public/views');

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

app.listen(8080);
```

Express Templates

- But Faisal, how do we render the views?
- Good Question! Instead of using res.sendFile(viewPath), we use res.render(viewName,{viewArgs});
- ViewArgs is optional, but viewName is obviously required.
- Lets take a look

Handlebars.js

```
var express = require('express');
var app = express();
var hbs = require('hbs');


// configure expresss. This is an important step.
// set our view engine to hbs. Handlebars.js
// now express will automaticly look for all .hbs files in the directory views
app.set('views', __dirname + '/public/views');

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

app.get('/', function(request, response) {
        // instead of send() or sendFile()
        // we now do render and layout name
        response.render("message",{message:"Hello"});
});

app.listen(8080);
```

Express Templates

- So lets go back to our twitter app and see how to fix it!

Express Templates

- So lets go back to our twitter app and see how to fix it!
- Handlebars makes it easy to loop through our data.
- Imagine putting a sleek UI on top of this, or any other
app. Using Express and Handlebars is very powerful.

Thank you!

@FaisalAbid

FaisalAbid.com