ADVENTURES IN NODE
Who am I?

- Node.js Eng @ Dynamatik - Node.js & UI/UX Consulting
- Node.js Developer for 2 years
- Android developer since beta days
- Blogs at www.FaisalAbid.com
- Tweets at @FaisalAbid
- 1 on 1 Node.js help every week. @FaisalAbid #NodeAssist
What is Node.js?

- 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. Client & Server
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. i.e Sinatra
- Modules for Node.js make it into a web framework. i.e Express
- Node.js is not Multi-threaded.
- A single thread to rule them all.
Single threaded?!?!?

- Node.js is build around “events”
- The event loop is always running checking for new events.
- Fires callbacks when new events are received.
- “Single threaded = blocking”. Won’t my server hang up?
- Yes, if you have blocking code.
Blocking Pseudocode

1. int registerUser(username,password){
2.     userID = lotsofcomplexcode(username,hash(password))
3.     return userID;
4. }

6. userID = registerUser("faisal","FaisalIsSoCool");
7. output("Your userID is "+userID);
Confused?

- In a multi-threaded/Process 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
- What an awesome platform...
Confused?

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

- By introducing callbacks:
  - Node can move on to other requests
    - Whenever the callback is called, node will process it.
  - You should read non-blocking code as
    - “put function and params in queue
    - fire callback when you reach the end of the queue”
Non-blocking Pseudocode

1. registerUser(username,password,callback){
2.   userID = longcomplexslow(username,Hash(password));
3.   callback(userID);
4. }

6. registerUser("faisal","FaisalIsSoCool",function(userId){
7.   output("Your userID is "+userId);
8. });
9.
Event Loop

- The event loop keeps on running.
- Checking for new events, so it can call them.
- Let's take a look at an example.
Non-blocking Pseudocode

1. var http = require('http');
3. var server = http.createServer(function(request, response){
4.     response.writeHead(200); // HTTP status
5.     response.write("Hello CodeMotion");
6.     response.end();
7. })
9. server.listen(8080);
Event Loop

- The event loop keeps on running.
- Checking for new events, so it can call them.
- Setup events to listen to, and Node.js will do other things till that event comes in.
Events in Action

- 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!
- Let's see some in action
Events

1. `var http = require('http');`
2. 
3. `var server = http.createServer();`
4. 
5. `server.on('request', function(request, response){`
6. `  response.writeHead(200); // HTTP status`
7. `  response.write("Hello OSCON");`
8. `  response.end();`
9. `});`
10. 
11. `server.on('connection', function(socket){`
12. `  console.log("New Connection");`
13. `});`
14. `server.listen(3030);`
15. 
Custom Events in Action

- How would we write custom events?
- We use the awesome EventEmitter class.
- Let's see how.
Events

1. var ee = require('events').EventEmitter;
2. var oscon = new ee();
3. 
4. oscon.on("start", function()
5.   console.log("open the doors");
6. );
7. 
8. oscon.on("end", function(wasItGood)
9.   if(wasItGood)
10.     console.log("YAY");
11.   else
12.     console.log("Bummer");
13. }
14. 
15. );
16. 
17. oscon.emit("start");
18. oscon.emit("end",false);
19. 
Custom Events in Action

- 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.
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”
Using Modules

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

2. var http = require("http") // http is a module thats being imported to this file

4. 
Using Modules

- So where is http anyways?
- Well, that stuff is stored where node.js is installed.
- What Node does is first looks in a node_modules folder
- Then it looks in a node_modules folder in your home folder “~”
- Then it looks where Node is installed. This is where http is.
Using 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

1. exports.somePublicMethod = function(){
3.   :
5.   exports.someOtherPublicMethod = function(){
6.     :
7.   }
9.   function somePrivateMethod(){
10.     :
11.   }
13.   var someRandomPublicMethod = function(){
14.     :
15.   }
17.   exports.randomName = someRandomPublicMethod;
19.
Using 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.
- Let's see how a module looks to get a better idea.
var oscon = require("./oscon");

console.log(oscon);

/*
{ somePublicMethod: [Function],
someOtherPublicMethod: [Function],
randomName: [Function] }
*/
Using 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!
Getting New 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.
- Let's see it in action.
Modules

1. `npm install nodemon`
2. `npm install -g coffee-script`
Getting New Modules

- You will have noticed when you installed a module, it downloaded a bunch of other modules also.

- These are dependencies for that module. Just like your app is dependent on module a, module a is dependent on module x,y,z.

- Defining dependencies is a great way to keep the project organized.

- Dependencies are defined using package.json
Modules

```json
{
    "name": "Hello Web",
    "preferGlobal": "true",
    "version": "0.0.1",
    "author": "Faisal Abid <faisal.abid@gmail.com>",
    "description": "Sample app for OSCON",
    "repository": {
        "type": "git",
        "url": "https://github.com/FaisalAbid/OSCON2013.git"
    },
    "dependencies": {
        "express": "~3.0.0rc5",
        "hbs": "~1.0.5",
        "mongoose": "~3.3.1"
    }
}
```
Package.json

- Every app should have a package.json. It's not required, but it's best practice.
- Define your app, Github repo, version etc.
- Define dependencies.
- But why? What's 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.
Maintain Package.json

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

- Thats cool, 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.
Express.js

- So earlier, I told you that Node.js is not a web framework.
- Technically that's 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
- It's sinatra inspired.
- What's cool about Express is that it's very powerful, very performant and most of all very very simple to use.
- Let's take a look.
API in 11 lines

1. var express = require('express');
2. var app = express();

4. app.get('/', function(request, response){
5.   return response.send('Hello World');
6. });

8. app.get('/conference/:name', function(request,response){
9.   return response.send(request.params.name);
10. });
11. app.listen(8080);
Express.js

- Let's see how this example would look without express.
API in 11 lines

1. var http = require('http');
3. var server = http.createServer(function(request, response){
4.   if(request.url == '/'){
5.     response.writeHead(200); // HTTP status
6.     response.write("Hello World");
7.     response.end();
8.   } else if(request.url.indexOf("conference") > -1){
9.     var n = request.url.split('/')[2]
10.    response.writeHead(200); // HTTP status
11.    response.write(n);
12.    response.end();
13.  } else{
14.    // do some other stuff
15.  }
16. });
17. server.listen(8080);
Express.js

- Wow thats messy and hard and error prone.
- Express encapsulates all this in a great API.
- Lets see another example for Express
Express.js

- Lets build a simple twitter feed viewer.
API in 11 lines

```javascript
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);
```
Express.js

- 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
Express.js

- Templates are a great way to separate your views from your controller and model.
- Express has great support for templates for a wide variety of style.
- There is Jade, EJS, Dust (LinkedIn now basically owns Dust), and my favorite Handlebars
- Handlebars offers a sweet and simple syntax to populate your templates.
Express.js Templating

- Templates are a great way to separate your views from your controller and model.
- Express has great support for templates for a wide variety of style.
- There is Jade, EJS, Dust (LinkedIn now basically owns Dust), and my favorite Handlebars
- Handlebars offers a sweet and simple syntax to populate your templates.
Handlebars

1. `<title>Hello</title>`
2. `<h1>{{variableName}}</h1>`
3.
**Express.js Templating**

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

1. var express = require('express');
2. var app = express();
3. var hbs = require('hbs');
4. 
5. app.set('views', __dirname + '/public/views');
6. 
7. app.set('view engine', 'hbs');
8. 
9. app.get('/', function(request, response) {
10.   // instead of send() or sendFile()
11.   // we now do render and layout name
12.   response.render("message", {message: "Hello"});
13. });
14. 
15. app.listen(8080);
Conclusion Why Node?

- Node.js lets you easily build stuff using existing skills
- No need to learn additional languages
- 32 K modules out there. HUGE community support.
- Node is a crazy fast growing platform.
Thank you!

@FaisalAbid
www.faisalabid.com

Need node consulting?
www.dynamatik.com
ADVENTURES IN NODE