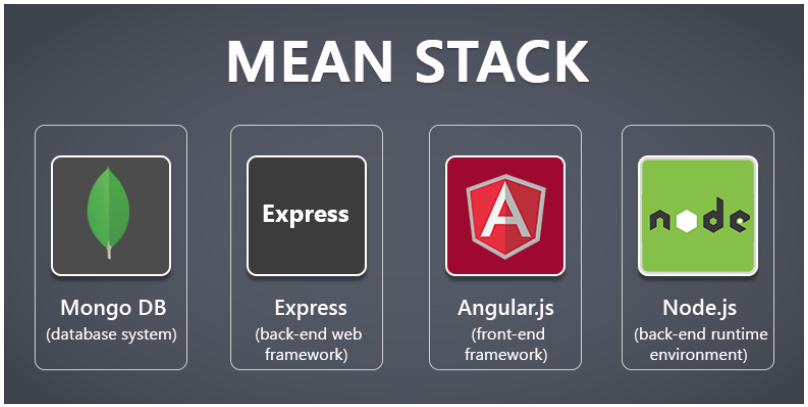
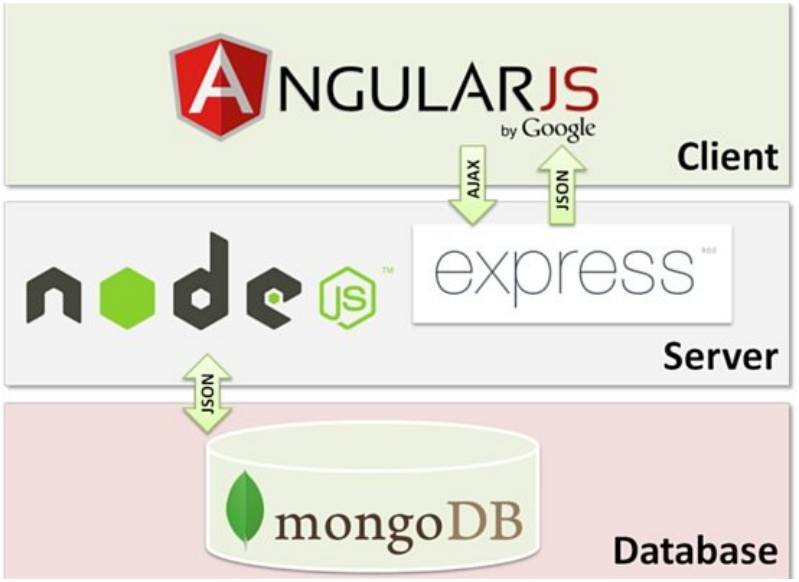
NodeJS

Rajeev Gupta
Trainer & consultant

First what is MEAN

- M = mongoDB
- E = Express
- A = Angular.js
- N=Node.js





What is node.js ?

- Created 2009
- Evented I/O for JavaScript
- Server Side JavaScript
- Runs on Google's V8 JavaScript Engine

Why Use Node.js ?

- Node's goal is to provide an easy way to build scalable network programs.

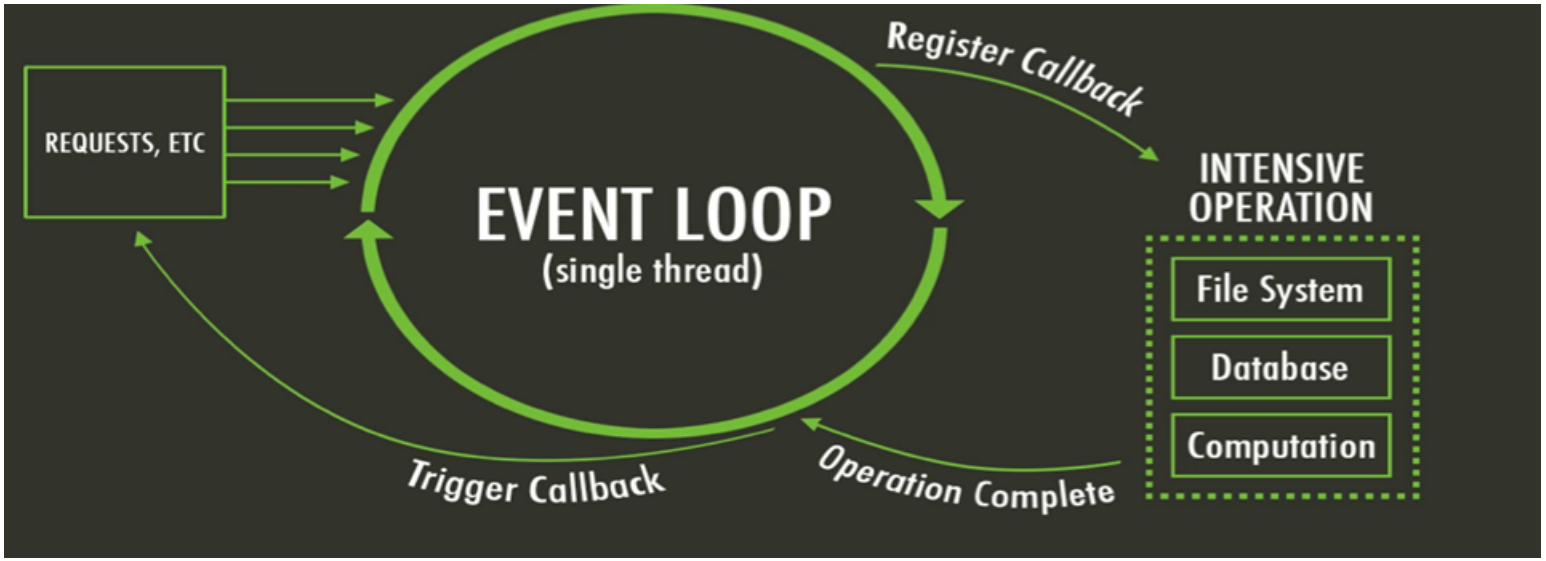
Standard JavaScript with

- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Console
- Crypto
- Debugger
- DNS
- Domain
- Events
- File System
- Globals
- HTTP
- HTTPS
- Modules
- Net
- OS
- Path
- Process
- Punycode
- Query Strings
- Readline
- REPL
- Stream
- String Decoder
- Timers
- TLS/SSL
- TTY
- UDP/Datagram
- URL
- Utilities
- VM
- ZLIB

... but without DOM manipulation

What is unique about Node.js?

1. JavaScript on server-side thus making communication between client and server will happen in same language
2. Servers normally thread based but Node.JS is “Event” based. Node.JS serves each request in a Evented loop that can handle simultaneous requests.



What can you do with Node ?

- It lets you Layered on top of the TCP library is a HTTP and HTTPS client/server.
- The JS executed by the V8 javascript engine (the thing that makes Google Chrome so fast)
- Node provides a JavaScript API to access the network and file system.

What can’t do with Node?

- Node is a platform for writing JavaScript applications outside web browsers. This is not the JavaScript we are familiar with in web browsers. There is no DOM built into Node, nor any other browser capability.

- Node can’t run on GUI, but run on terminal

Threads VS Event-driven

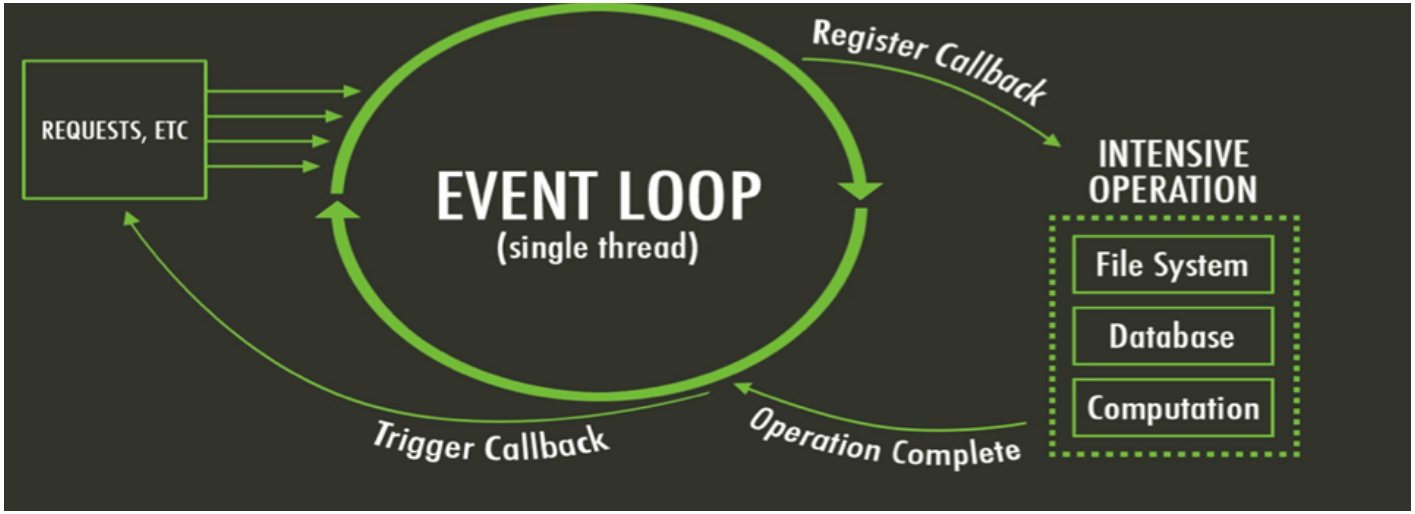
Threads	Asynchronous Event-driven
Lock application / request with listener-workers threads	only one thread, which repeatedly fetches an event
Using incoming-request model	Using queue and then processes it
multithreaded server might block the request which might involve multiple events	manually saves state and then goes on to process the next event
Using context switching	no contention and no context switches
Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock	Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments

Why node.js use event-based?

normal process the webserver while processing the request will have to wait for the IO operations and thus blocking the next request to be processed.

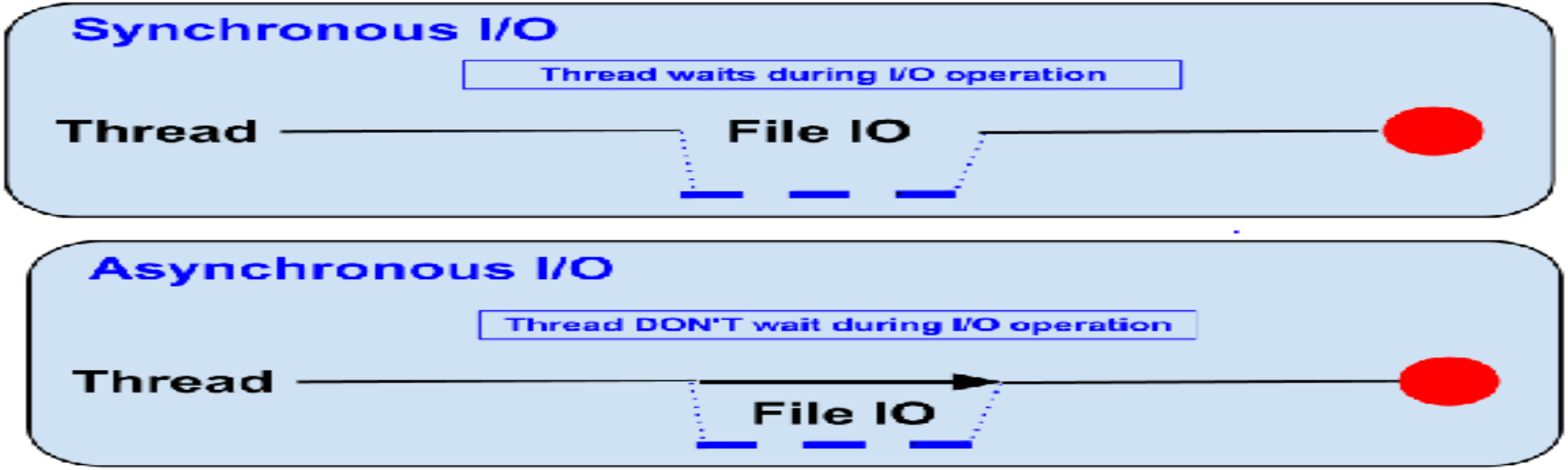
Node.JS process each request as events, doesn't wait (non-blocking) for the IO operation to complete → it can handle other request at the same time.

When the IO operation of first request is completed it will call-back the server to complete the request.



Blocking vs Non-Blocking.....

Example :: Read data from file and show



node

Blocking.....

- Read data from file
- Show data
- Do other tasks

```
var data = fs.readFileSync( “test.txt” );
console.log( data );
console.log( “Do other tasks” );
```

Non-Blocking.....

- Read data from file

  When read data completed, show data

- Do other tasks


```
fs.readFile( “test.txt”, function( err, data ) {
      console.log(data);
});
```

Node.js VS Apache

1. It's faster
2. It can handle tons of concurrent requests

Platform	Number of request per second
PHP (via Apache)	3187,27
Static (via Apache)	2966,51
Node.js	5569,30

Success Stories.....

Rails to Node
« Servers were cut to 3 from 30 »
« Running up to 20x faster in some scenarios »
« Frontend and backend mobile teams could be combined [...] »

Java to Node
« Built almost twice as fast with fewer people »
« Double the requests per second »
« 35% decrease in the average response time »

Supports HTTP Method.....

- GET
- POST
- PUT
- DELETE

When to use it ?

- Chat/Messaging
- Real-time Applications
- Intelligent Proxies
- High Concurrency Applications
- Communication Hubs
- Coordinators

Node.js for....

- Web application
- Websocket server
- Ad server
- Proxy server
- Streaming server
- Fast file upload client
- Any Real-time data apps
- Anything with high I/O

File package.jso

Project information

- Name
- Version
- Dependencies
- Licence
- Main file

Etc...

```
{
  "name": "node-js-getting-started",
  "version": "0.2.5",
  "description": "A sample Node.js app using Express 4",
  "engines": {
    "node": "5.9.1"
  },
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "body-parser": "^1.16.1",
    "cookie-parser": "^1.4.3",
    "cool-ascii-faces": "1.3.4",
    "ejs": "2.4.1",
    "express": "^4.13.3",
    "express-session": "^1.15.1",
    "mongodb": "^2.2.24",
    "multer": "^1.3.0",
    "pg": "4.x",
    "pug": "^2.0.0-beta11"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/heroku/node-js-getting-started"
  },
  "keywords": [
    "node",
    "heroku",
    "express"
  ],
  "license": "MIT"
}
```

Node.js Modules..... MANY

- https://npmjs.org/
- # of modules = 1,21,943

Install a module.....inside your project directory

$npm install <module name>

Using module..... Inside your javascript code

- var http = require(‘http’);
- var fs = require(‘fs’);
- var express = require(‘express’);

Hello World Example

STEP 1: create directory and call npm install and follow instructions

>mkdir myapp

>cd myapp

- Use the npm init command to create a package.json file for your application. For more information, see Specifics of npm's package.json handling.

> $ npm init

- prompts you for a number of things, such as the name and version of your application. For now, you can simply hit RETURN to accept the defaults.

Hello World example

- Create file index.js with the following code:

```
http.createServer(function (request, response) {
    // Send the HTTP header
    // HTTP Status: 200 : OK
    // Content Type: text/plain
     response.writeHead(200, {'Content-Type':
  'text/plain'});
     // Send the response body as "Hello World“
     response.end('Hello World\n'); }).listen(8081);

// Console will print the message
 console.log('Server running at http://127.0.0.1:8081/');
```

Hello World example –package.json – describes application

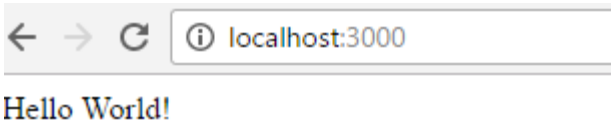
```
{

        "name": "helloworld",
        "version": "1.0.0",
        "description": "simple hello world app",
        "main": "index.js",
        "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
        },
        "author": "L. Grewe",
        "license": "ISC",
        "dependencies": {
        "express": "^4.14.1"
        }

}
```

Run your hello world application

Run the app with the following command:

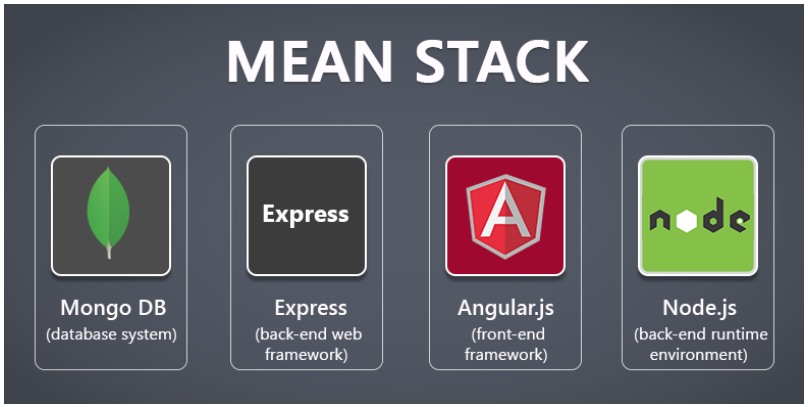
$ node app.js

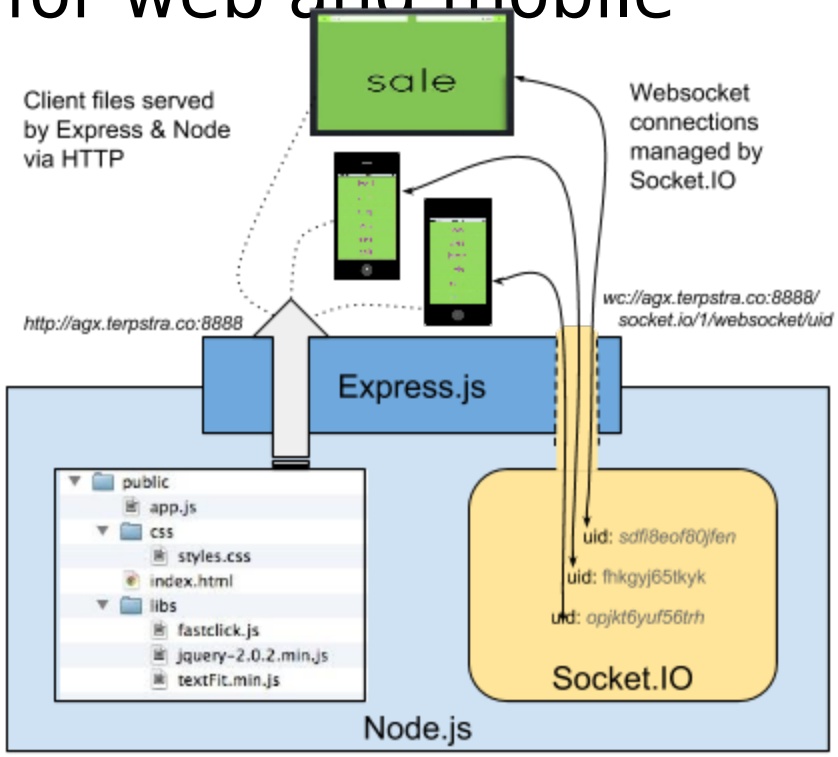
Then, load http://localhost:3000/ in a browser to see the output.



Express

- **minimal and flexible Node.js web application framework** that provides a robust set of features for web and mobile applications.





Express gives ease of functionality

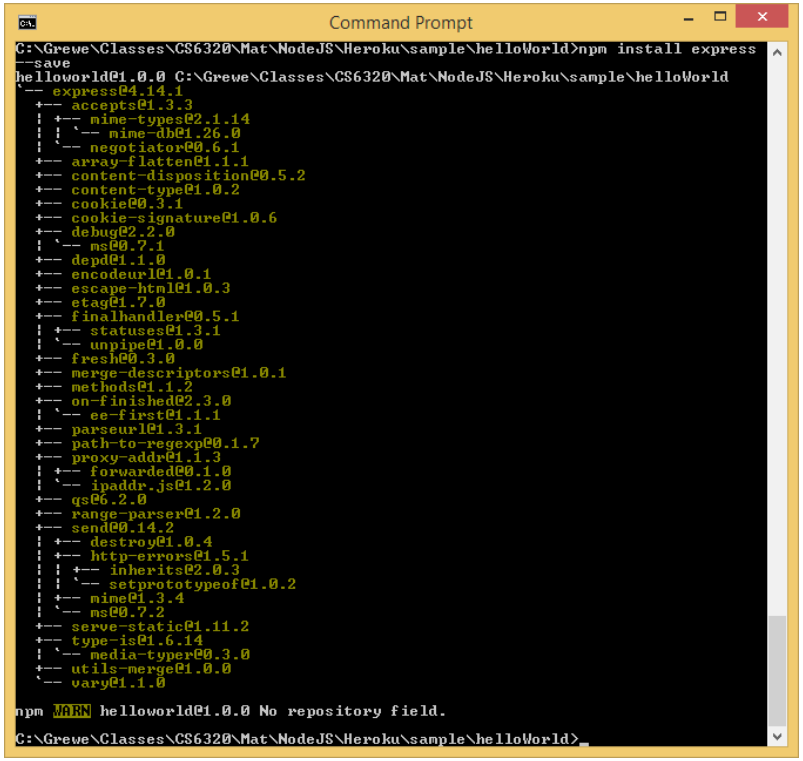
- Routing
- Delivery of Static Files
- “Middleware” – some ease in development (functionality)
- Form Processing
- Simple forms of Authentication
- Cookies and Session Manipulation

A lot of this you can do in NodeJS but, you may write more code to do it than if you use the framework Express.

Install express

- **install Express**
- Now install Express in the myapp directory and save it in the dependencies list. For example:

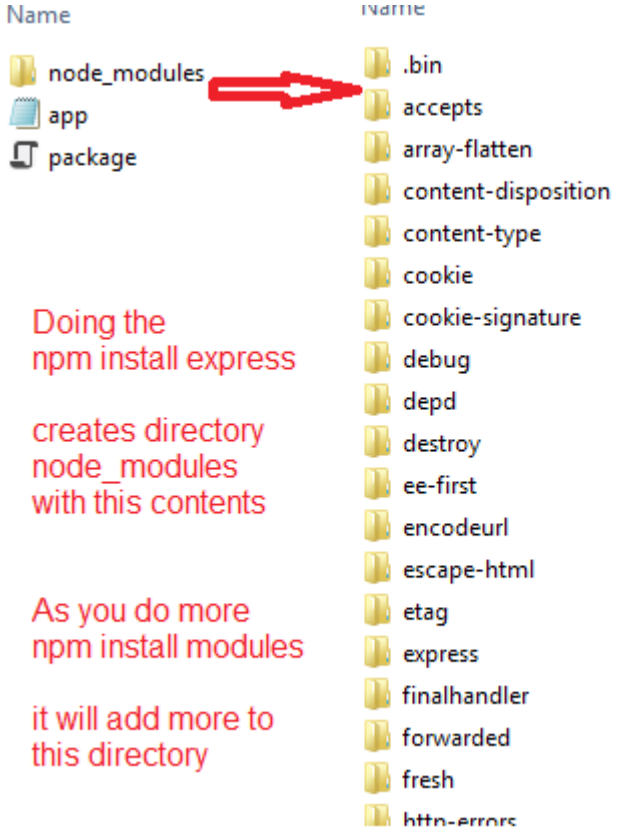
  >$ npm install express --save



Express install

- Will add files to the node_modules directory

- If this is the first module you have installed for current application IT will create the node_modules directory first.



Express – hello world code

- index.js have the code

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

var app = express()

app.get('/', function (req, res) {
    res.send('Hello World!')
})
app.listen(3000, function () {
        console.log('Example app listening on port 3000!')
})
```

This says requires module express

Calls function express to initialize object app

App object has various methods like get
that responds to HTTP get request.
This code will be call the function specified when
a GET for the URI / is invoked

Sets up the HTTP server for listening port 3000