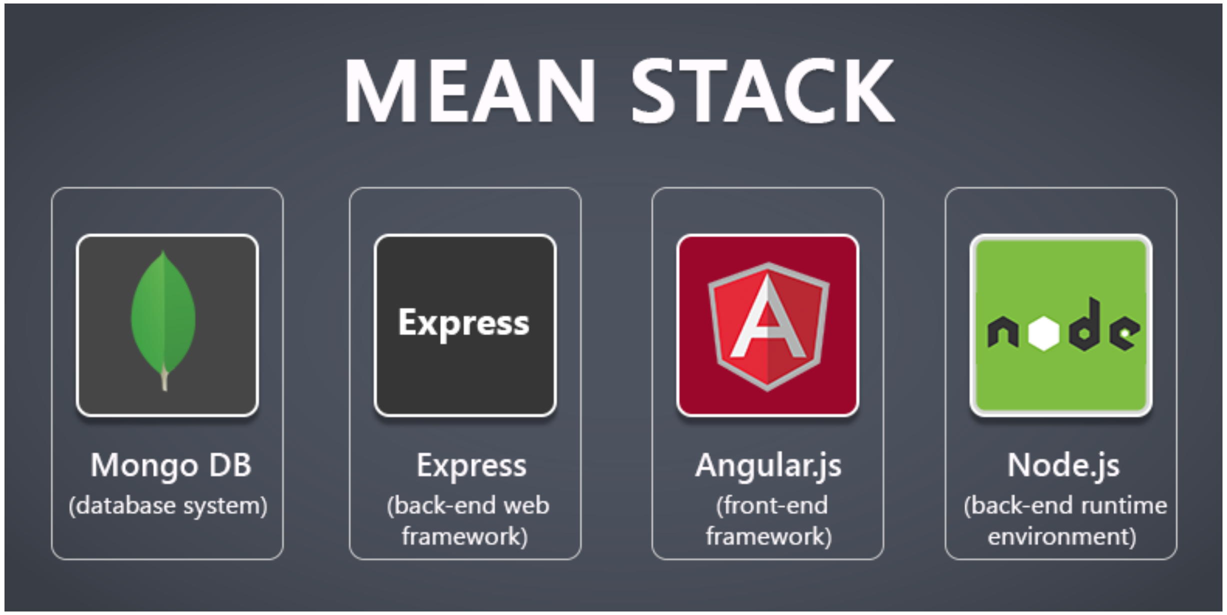
Node.js and the MEAN Stack



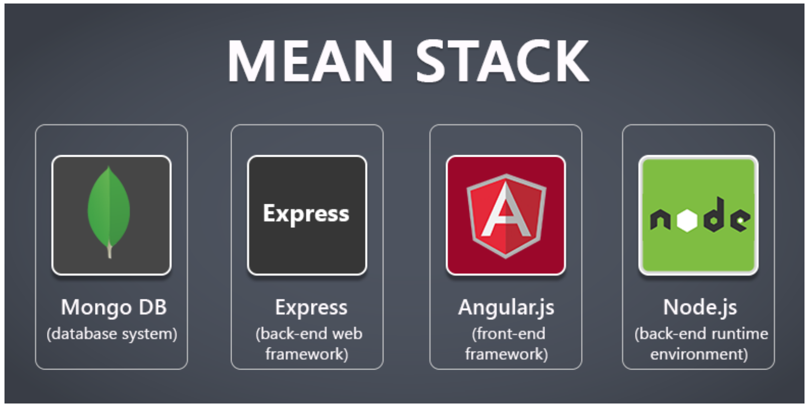
What is NodeJS?

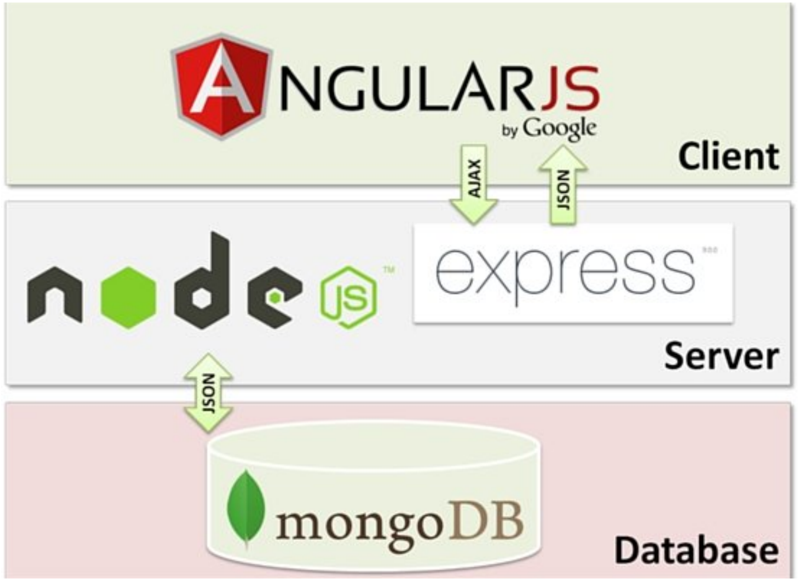
- A JavaScript runtime environment running Google Chromes V8 engine
  - a.k.a. a server-side solution for JS
  - Compiles JS, making it really fast
- Created 2009
- Server Side JavaScript
- Designed for high concurrency
  - Without threads or new processes
- Evented I/O for JavaScript
- Never blocks, not even for I/O

First what is MEAN

- M = mongoDB -- lightly covered in this class
- E = Express -- lightly covered in this class
- A = Angular.js (client side) – will not cover
- N=Node.js -- lightly covered in this class

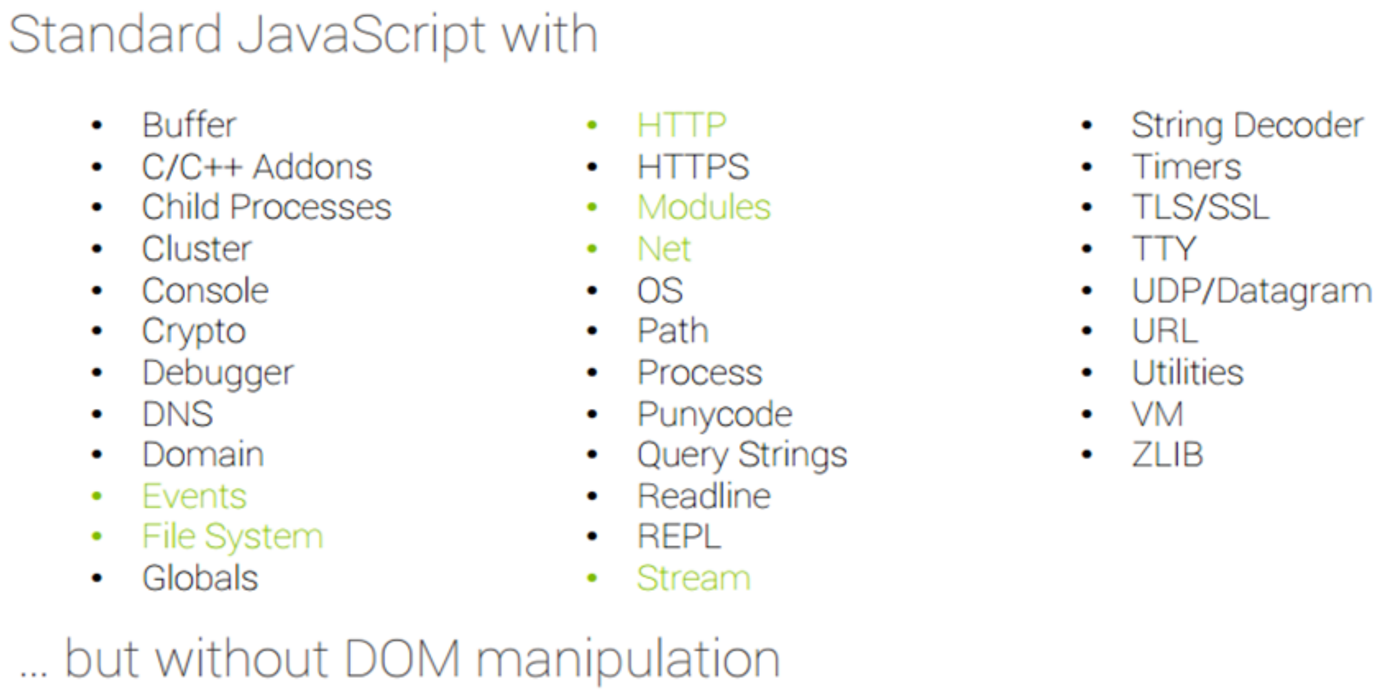
FULL stack solution





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

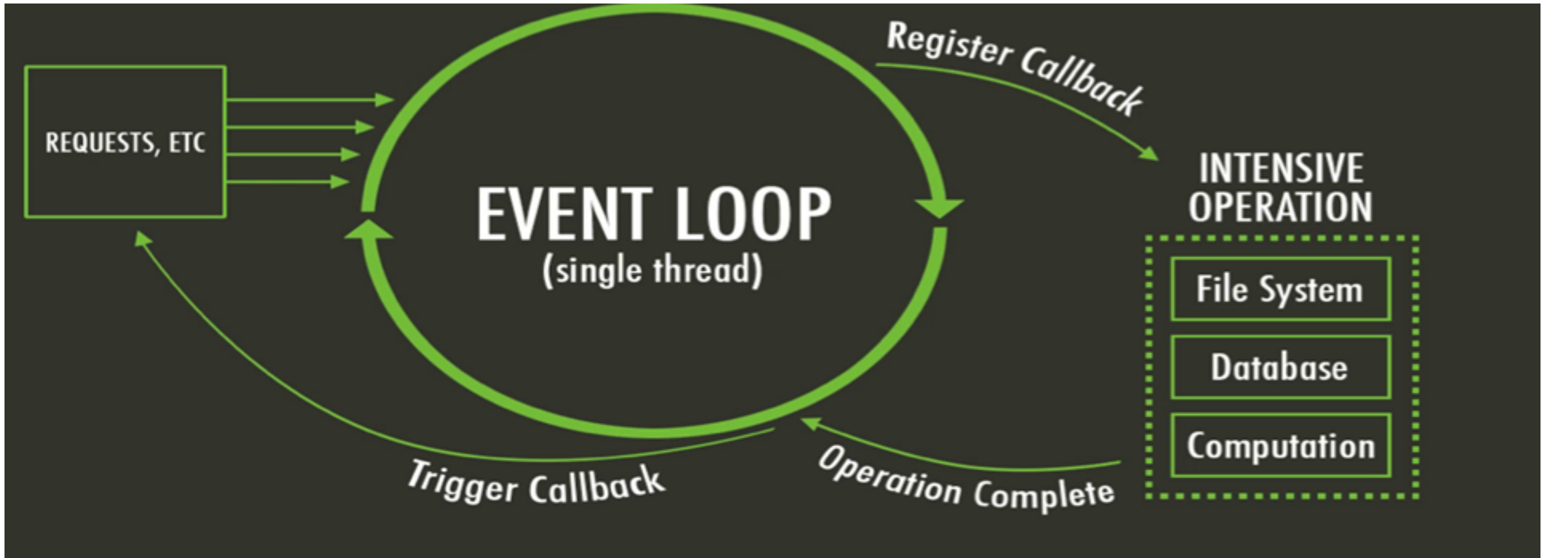
... but without DOM manipulation

What can you do with Node ?

- JS executed by the V8 javascript engine, the same thing that makes Google Chrome so fast.
- Node provides a JavaScript API to access the network and file system.
- Instead of threads Node uses an event loop with a stack which alleviates overhead of context switching

What is unique about Node.js?

1. JavaScript on server-side ensures that communication between client and server will happen in same language – native JSON objects on both sides
2. Servers are normally thread based but Node.JS is "Event" based. Node.JS serves each request in an Evented loop that can handle simultaneous requests.



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 doesn’t run in a GUI, but runs on the terminal or as a background process

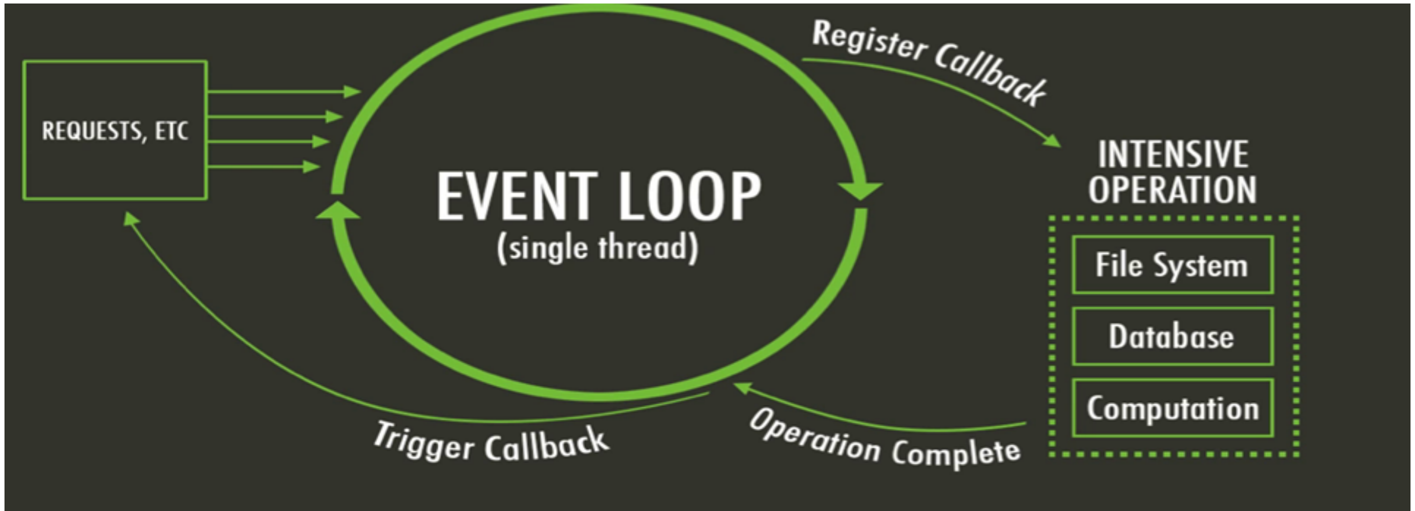
Threads VS Event-driven

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

Node.js uses an event-based model

Normally a webserver waits for server-side IO operations to complete while processing a web client request, thus blocking the next request to be processed.

Node.JS processes each request as events, it 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.

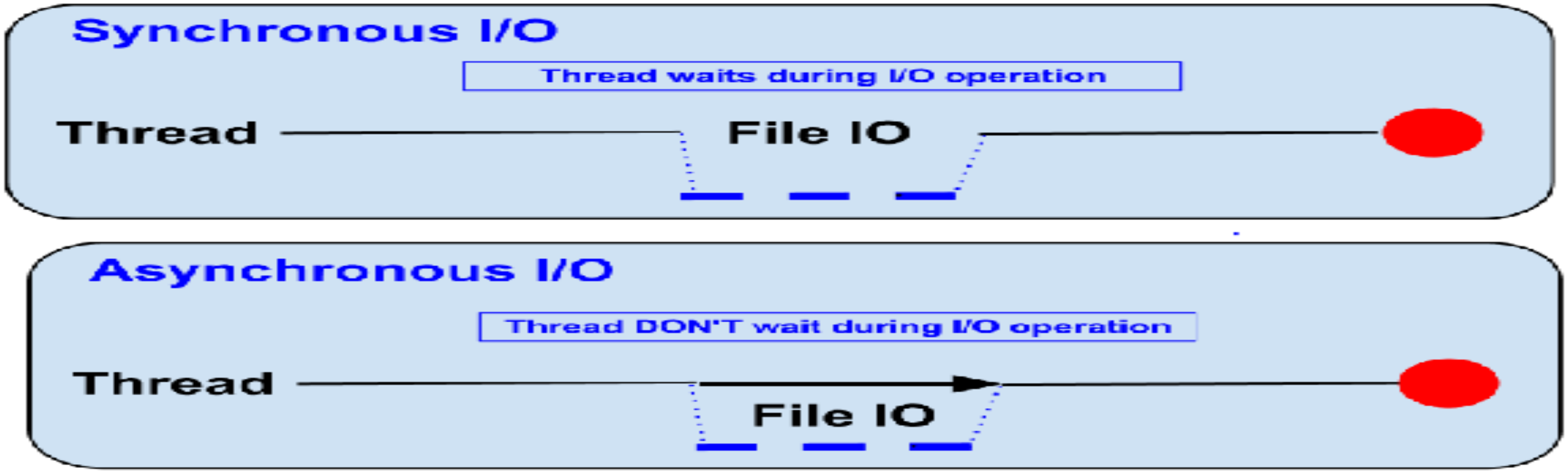


Non-blocking I/O

- Servers do nothing but I/O
    - Scripts waiting on I/O requests degrades performance
- To avoid blocking, Node makes use of the event driven nature of JS by attaching callbacks to I/O requests
- Scripts waiting on I/O waste no space because they get popped off the stack when their non-I/O related code finishes executing

Blocking vs Non-Blocking……

Example :: Read data from file and show data



Synchronous I/O
Thread waits during I/O operation
Thread
File IO
Asynchronous I/O
Thread DON'T wait during I/O operation
Thread
File IO

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);
});
```

Callback

PHP vs Node Example

```
<?php
$result = mysql_query('SELECT * FROM ...');
while($r = mysql_fetch_array($result)){
    // Do something
}

// Wait for query processing to finish...
?>

<script type="text/javascript">
mysql.query('SELECT * FROM ...', function (err, result, fields){
        // Do something
    });

// Don't wait, just continue executing
</script>
```

Node.js VS Apache

1. It's fast
2. It can handle tons of concurrent requests
3. It's written in JavaScript (which means you can use similar code server side and client side)

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

Netflix has over 160 million customers worldwide
Change to Node.js reduced startup time by 70%

Success Stories…..

NETFLIX

Linked in

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 [...] »

PayPal™

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

GROUPON

Walmart

YAHOO!

UBER

node js®

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.json.....

Project information

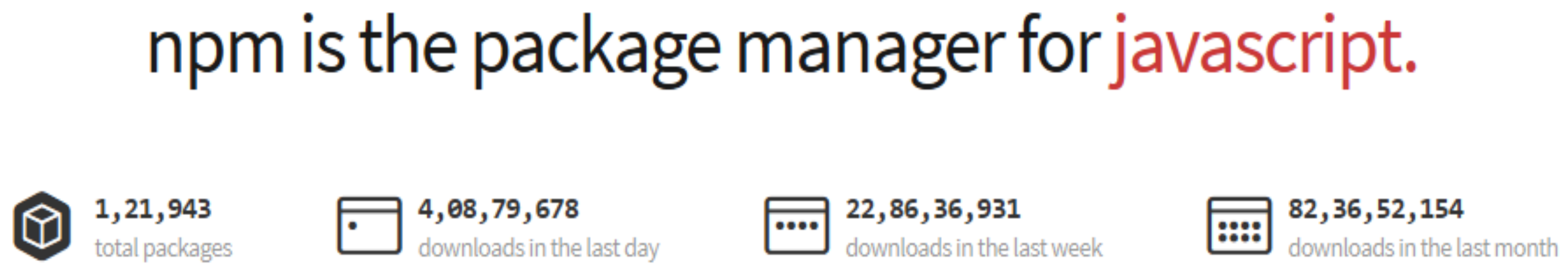
- Name
- Version
- Dependencies
- Licence
- Main file

etc...

```json
{
  "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 = 450k and counting



npm is the package manager for javascript.

1,21,943 total packages	4,08,79,678 downloads in the last day	22,86,36,931 downloads in the last week	82,36,52,154 downloads in the last month

Install a module…..inside your project directory

```
$npm install <module name> --save
```

Install a module…..globally

```
$npm install <module name> --g
```

Using a 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

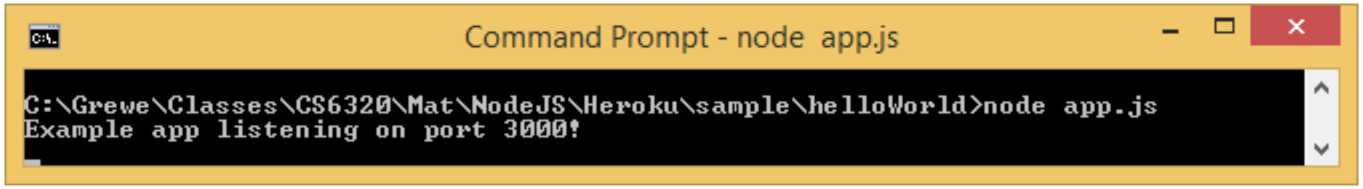
```
{

        "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

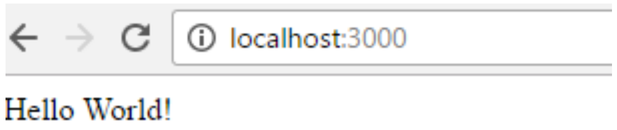
Run the app with the following command:

$ node app.js

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



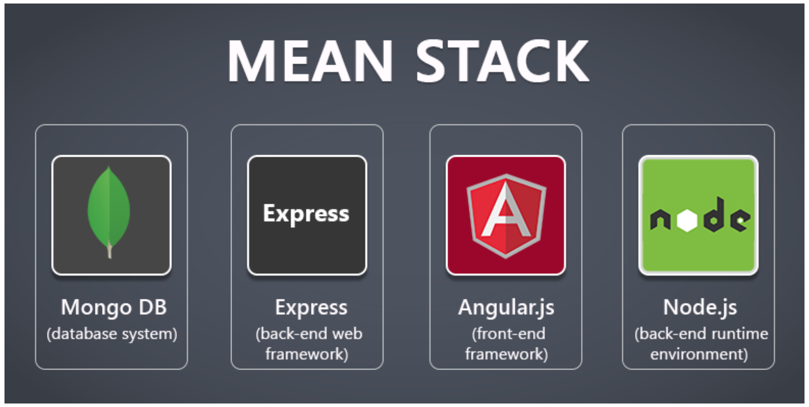
```
C:\Grewe\Classes\CS6320\Mat\NodeJS\Heroku\sample\helloWorld>node app.js
Example app listening on port 3000!
```

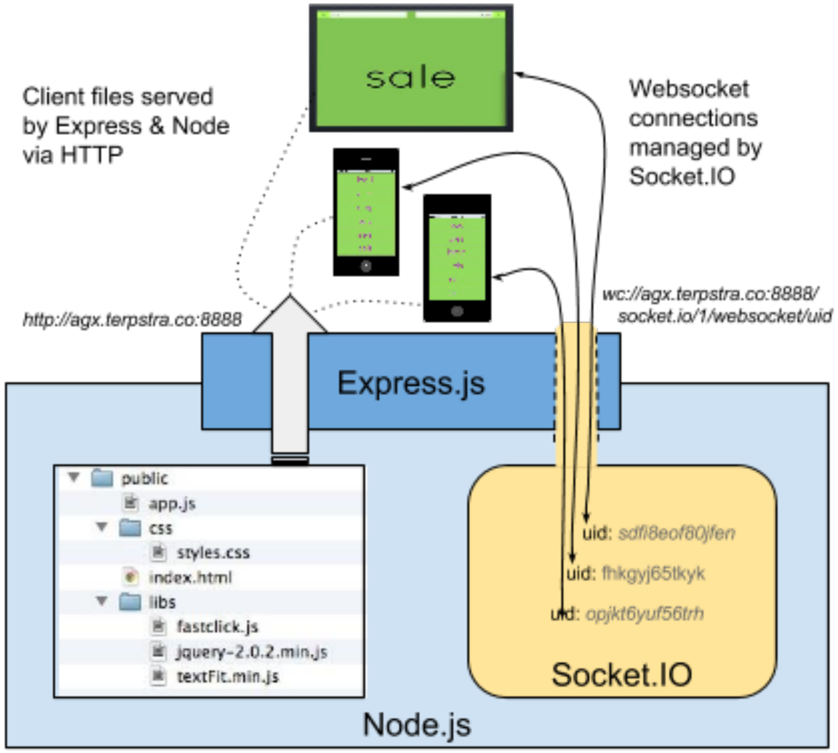


Hello World!

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

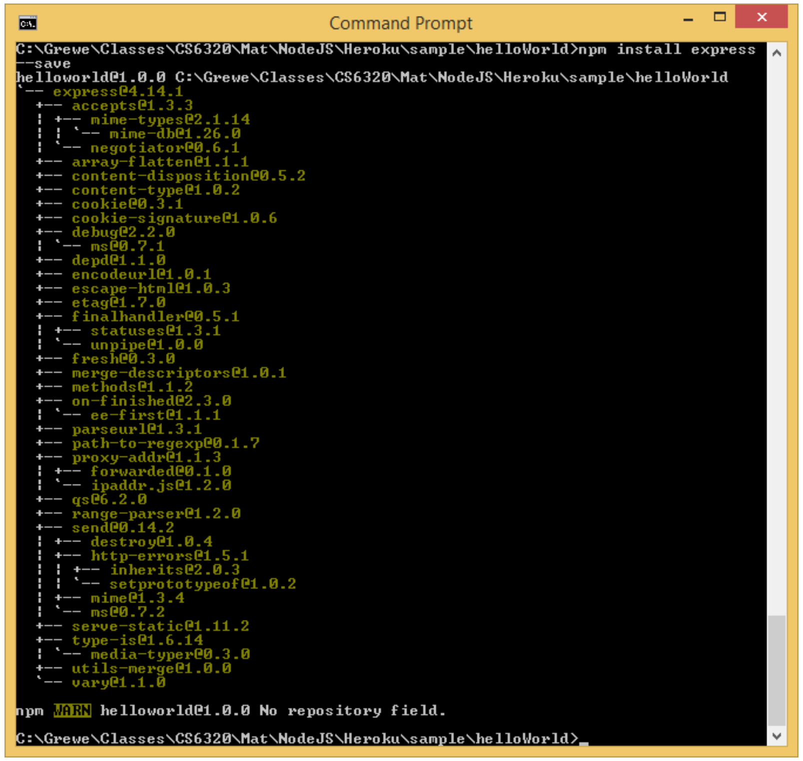
A lot of this you can do in NodeJS but, you may write more code to do it than if you use the framework Express.

There are other alternatives than Express (the E in MEAN) like Sail, Meteor

Install express

- **install Express (if you want it, most will) and any other dependencies needed**
- Now install Express in the myapp directory and save it in the dependencies list. For example:

  >$ npm install express --save

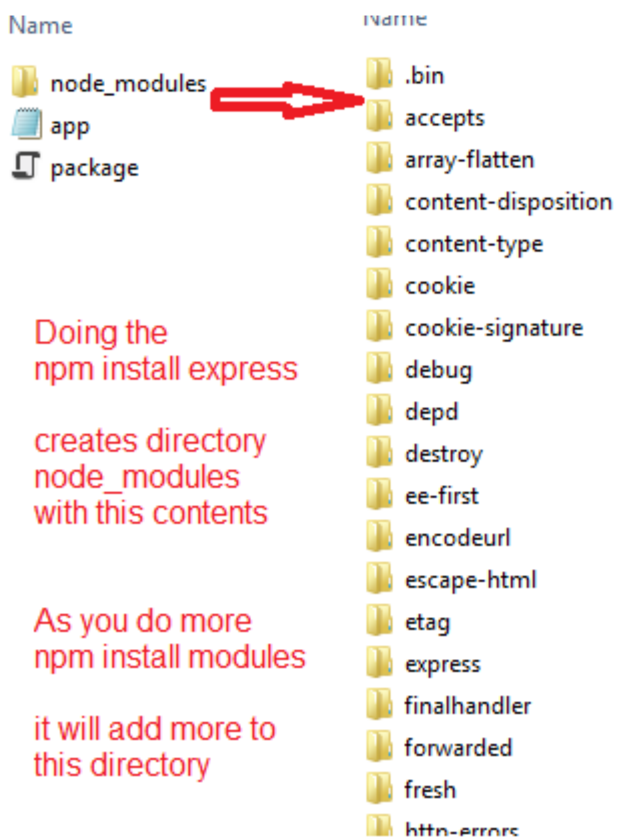


```
C:\Grewe\Classes\CS6320\Mat\NodeJS\Heroku\sample\helloWorld>npm install express
--save
helloworld@1.0.0 C:\Grewe\Classes\CS6320\Mat\NodeJS\Heroku\sample\helloWorld
`-- express@4.14.1
  +-- accepts@1.3.3
  | +-- mime-types@2.1.14
  | | `-- mime-db@1.26.0
  | `-- negotiator@0.6.1
  +-- array-flatten@1.1.1
  +-- content-disposition@0.5.2
  +-- content-type@1.0.2
  +-- cookie@0.3.1
  +-- cookie-signature@1.0.6
  +-- debug@2.2.0
  | `-- ms@0.7.1
  +-- depd@1.1.0
  +-- encodeurl@1.0.1
  +-- escape-html@1.0.3
  +-- etag@1.7.0
  +-- finalhandler@0.5.1
  | +-- statuses@1.3.1
  | `-- unpipe@1.0.0
  +-- fresh@0.3.0
  +-- merge-descriptors@1.0.1
  +-- methods@1.1.2
  +-- on-finished@2.3.0
  | `-- ee-first@1.1.1
  +-- parseurl@1.3.1
  +-- path-to-regexp@0.1.7
  +-- proxy-addr@1.1.3
  | +-- forwarded@0.1.0
  | `-- ipaddr.js@1.2.0
  +-- qs@6.2.0
  +-- range-parser@1.2.0
  +-- send@0.14.2
  | +-- destroy@1.0.4
  | +-- http-errors@1.5.1
  | | +-- inherits@2.0.3
  | | `-- setprototypeof@1.0.2
  | +-- mime@1.3.4
  | `-- ms@0.7.2
  +-- serve-static@1.11.2
  +-- type-is@1.6.14
  | `-- media-typer@0.3.0
  +-- utils-merge@1.0.0
  `-- vary@1.1.0

npm WARN helloworld@1.0.0 No repository field.

C:\Grewe\Classes\CS6320\Mat\NodeJS\Heroku\sample\helloWorld>_
```

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.



Name

node_modules
app
package

Doing the
npm install express

creates directory
node_modules
with this contents

As you do more
npm install modules

it will add more to
this directory

Name

.bin
accepts
array-flatten
content-disposition
content-type
cookie
cookie-signature
debug
depd
destroy
ee-first
encodeurl
escape-html
etag
express
finalhandler
forwarded
fresh
http-errors

ALTERNATIVE express-generator

npm install express-generator -g express helloapp

```
create : helloapp
create : helloapp/package.json
create : helloapp/app.js
create : helloapp/public
create : helloapp/public/images
create : helloapp/routes
create : helloapp/routes/index.js
create : helloapp/routes/users.js
create : helloapp/public/stylesheets
create : helloapp/public/stylesheets/style.css
create : helloapp/views create : helloapp/views/index.jade
create : helloapp/views/layout.jade
create : helloapp/views/error.jade
create : helloapp/bin
create : helloapp/bin/www

install dependencies:
$ cd helloapp && npm install

run the app:
$ DEBUG=helloapp:* npm start

create : helloapp/public/javascripts
```

Express – hello world code

- index.js have the code

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

 var app = express()

 app.get('/', function (req,
       res.send('Hello World!')
 })

app.listen(3000, function () {
          console.log('Example app listenin
 })
```

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

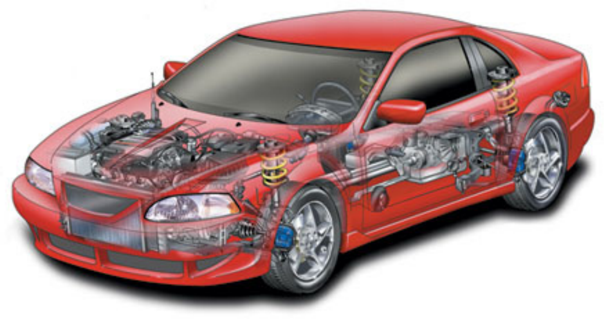
NEXT – Todo Application

- We will go cover a simple Todo application that uses Node.js, Express, EJS, a templating engine that works with Express, and Mongo DB.

Once you understand this, you will know the basics of MEAN (without the A) and a start towards using NodeJS for web systems.

HOWEVER…..you might also want to look at Meteor – less callbacks, more subscription model than using MEAN

The MEAN stack


Meteor.js

References

- http://nodejs.org/
- http://npmjs.com/
- http://www.w3schools.com/nodejs
- http://ajaxian.com/archives/google-chrome-chromium-and-v8