

MEAN STACK



Mongo DB
(database system)

Express

Express
(back-end web
framework)



Angular.js
(front-end
framework)



Node.js
(back-end runtime
environment)

UNIT 3

GRAPHQL:

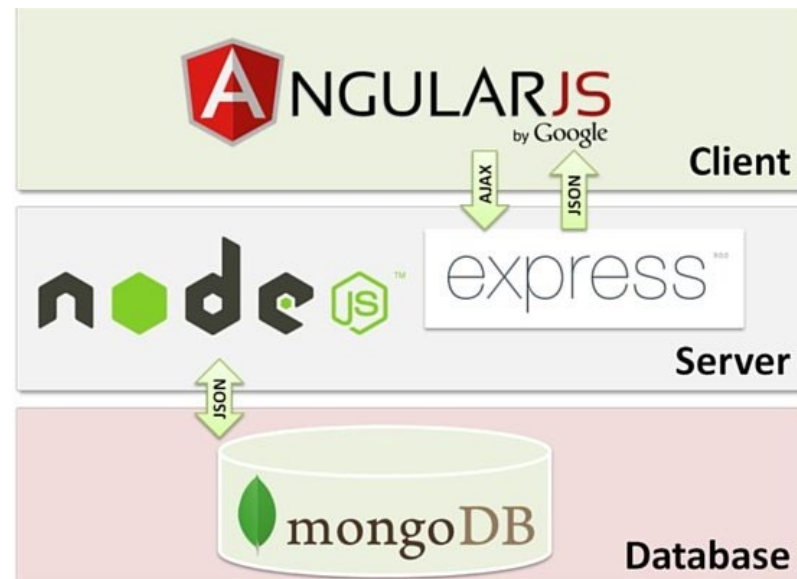
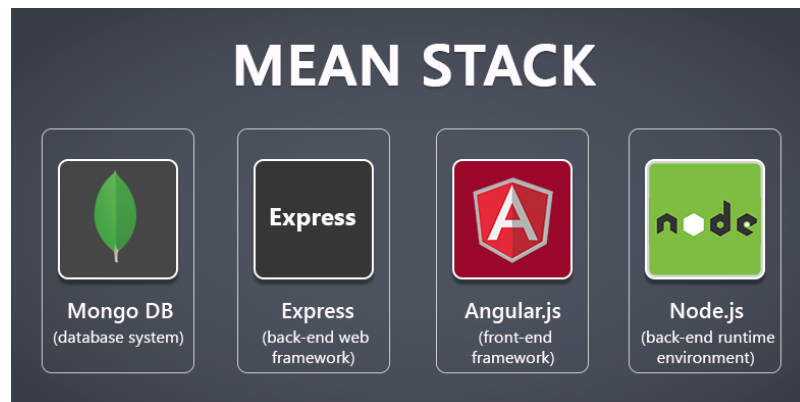
NODEJS

- Introduction
- GraphQL is the better REST
- Core Concepts
- schema definition language
- Queries and mutations
- schemas and types
- GraphQLclient and server
- connecting database via prisma
- graphql tools and ecosystem
- security

First what is MEAN

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

FULL stack solution



What is node.js ?

- Created 2009
- Evented I/O for JavaScript
- Server Side JavaScript
- Runs on Google's V8 JavaScript Engine
- Node.js is a cross-platform environment and library for running JavaScript applications which is used to create networking and server-side applications.
- Node.js = Runtime Environment + JavaScript Library

Why Use Node.js ?

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

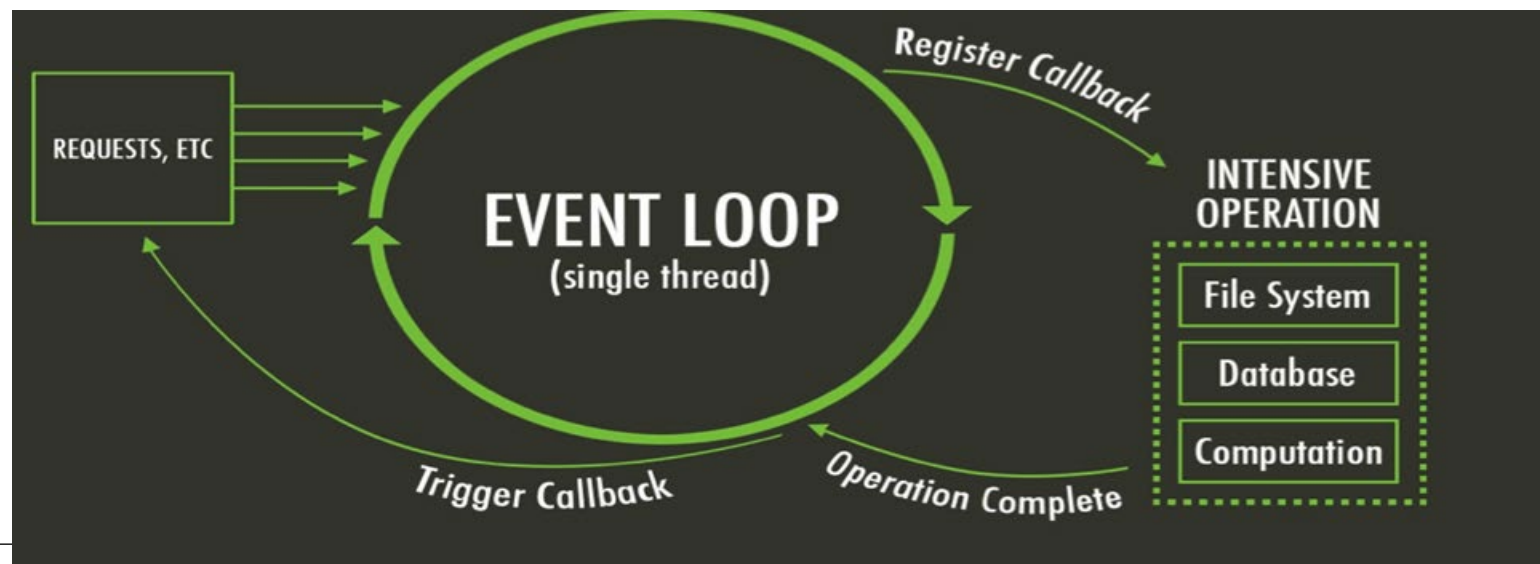
Standard JavaScript with

- 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

Features of Node js

- **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
- **I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.
- **Single threaded:** Node.js follows a single threaded model with event looping.
- **Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
- **No buffering:** Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
- **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
- **License:** Node.js is released under the MIT license.

Threads VS Event-driven

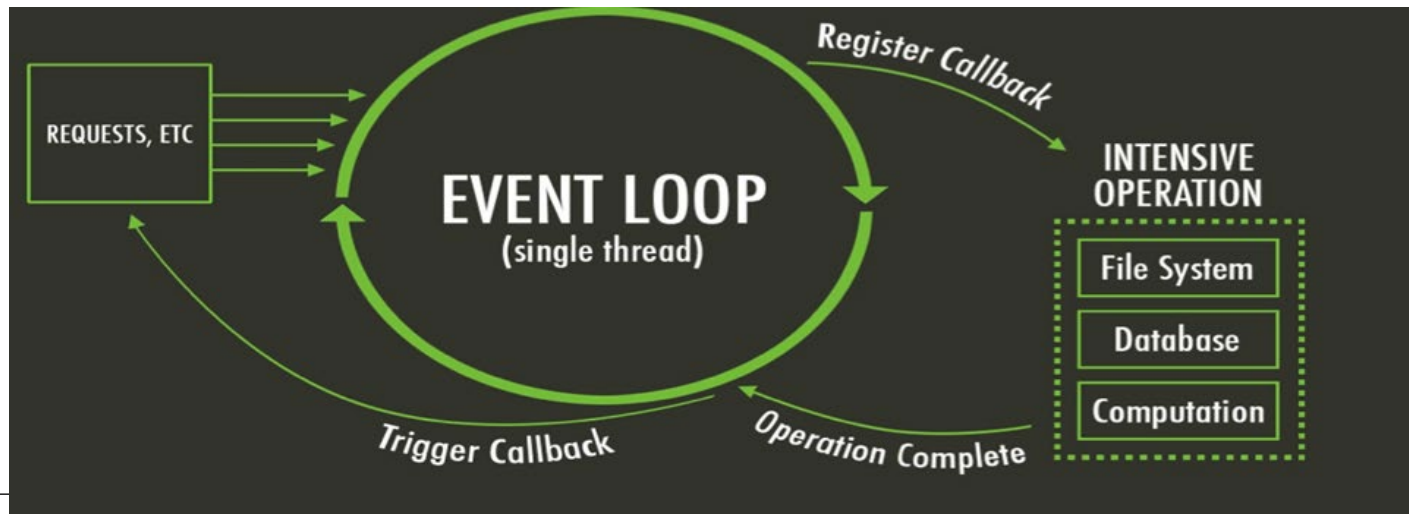
Threads	Asynchronous Event-driven
Lock application / request with listener-workers threads	only one thread, which repeatedly fetches an event
Using incoming-request model	Using queue and then processes it
multithreaded server might block the request which might involve multiple events	manually saves 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 data

Synchronous I/O

Thread waits during I/O operation



Asynchronous I/O

Thread DON'T wait during I/O operation



Node.js web-based Example

- A node.js web application contains the following three parts:
- **Import required modules:** The "require" directive is used to load a Node.js module.
- **Create server:** You have to establish a server which will listen to client's request similar to Apache HTTP Server.
- **Read request and return response:** Server created in the second step will read HTTP request made by client which can be a browser or console and return the response.
- Ex:-web node js

Node.js Console

- The Node.js console module provides a simple debugging console similar to JavaScript console mechanism provided by web browsers.
- There are 3 console methods that are used to write any node.js stream:
- `console.log()`
- `console.error()`
- `console.warn()`
- `console.log('Hello aliens.....');`
- `console.error(new Error('Hell! This is a wrong method.'));`
- `const name = 'John';`
- `console.warn(`Don't mess with me ${name}! Don't mess with me!`);`

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



Node.js Package Manager npm

- Node Package Manager provides two main functionalities:
- It provides online repositories for node.js packages/modules which are searchable on search.npmjs.org
- It also provides command line utility to install Node.js packages, do version management and dependency management of Node.js packages

Librararies installation

- *npm install express* *local*
- *npm install express -g* *global*

File package.json....

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

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>
```



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 "HelloWorld"  
    response.end('HelloWorld\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

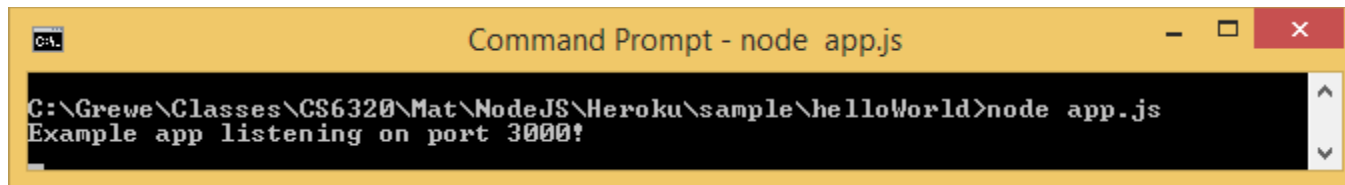
```
{  
  
  "name": "helloworld",  
  "version": "1.0.0",  
  "description": "simple hello world app",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1 "  
  },  
  "author": "L. Grewe",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.14.1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

Run your hello world application

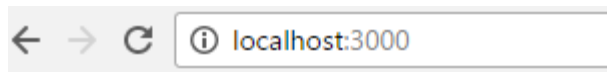
Run the app with the following command:

```
$ node app.js
```

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



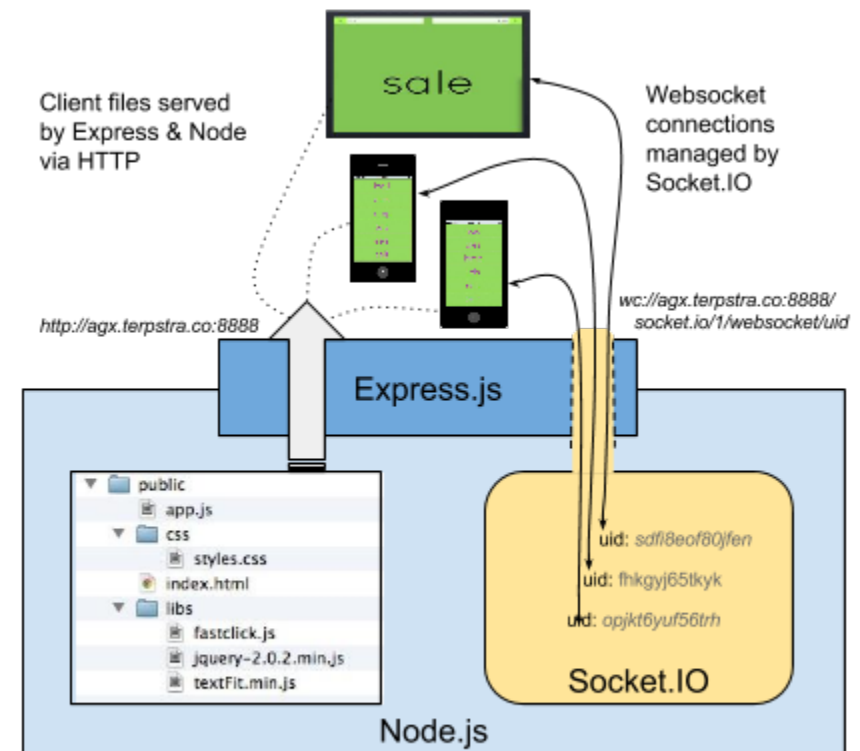
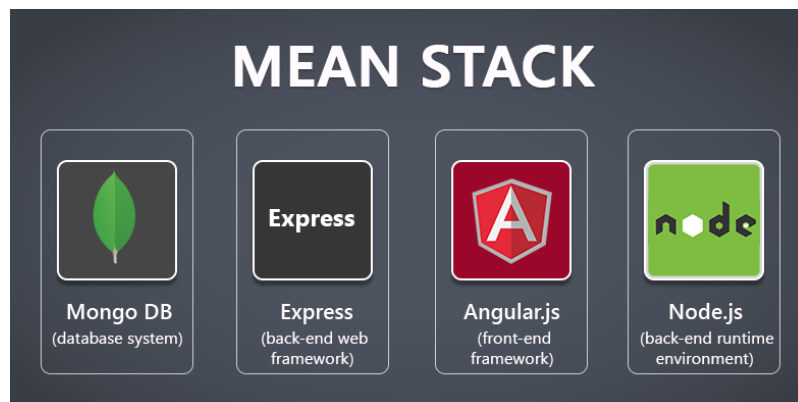
```
CA: Command Prompt - node app.js
C:\Grewe\Classes\CS6320\Mat\NodeJS\Heroku\sample\helloWorld>node app.js
Example app listening on port 3000!
```



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

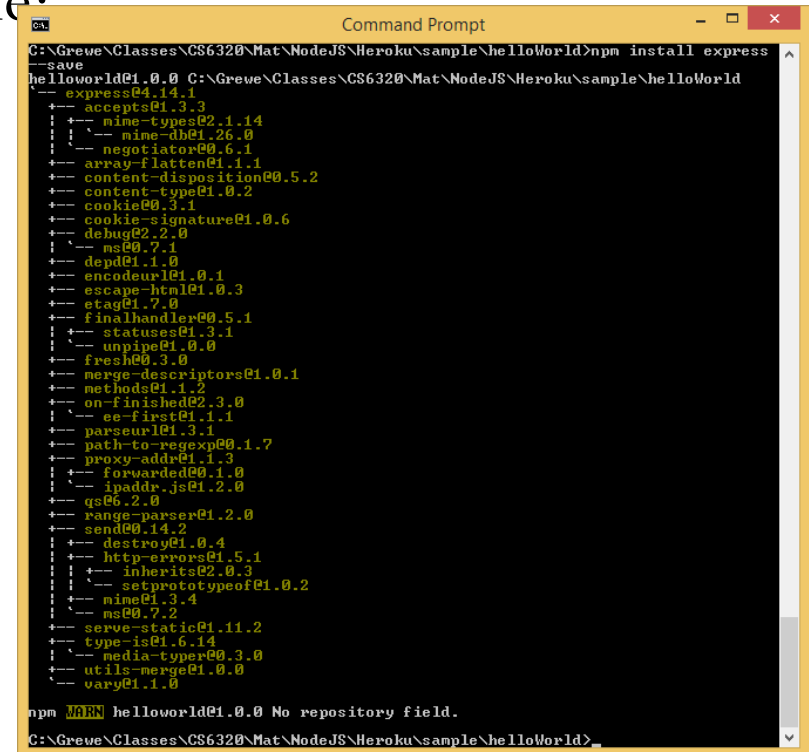
A lot of this you can do in NodeJS but, you may write more code to do it than if you use the framework Express.

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:

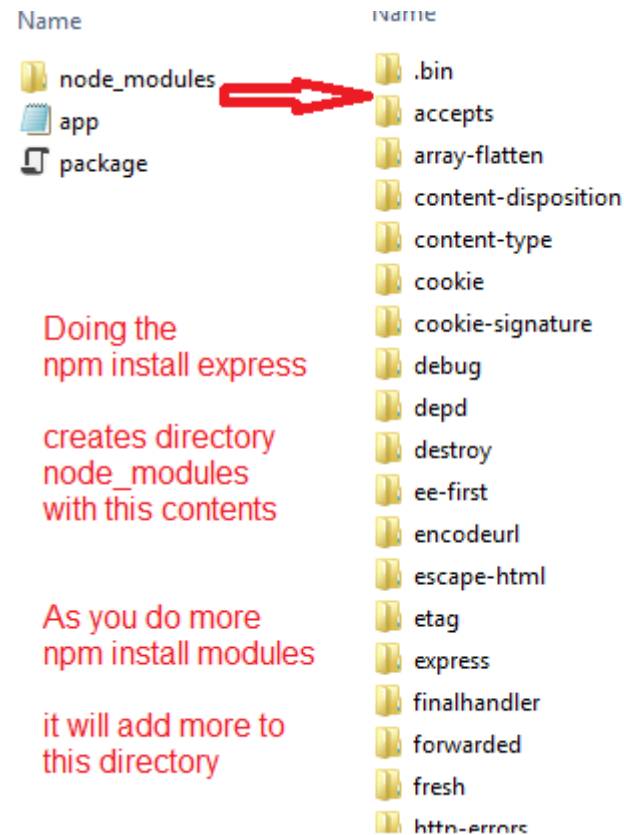
> \$ npm install express --save



```
Command Prompt
C:\Grewe\Classes\CS6320\Mat\NodeJS\Heroku\sample\helloWorld>npm install express --save
hellorworld@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 hellorworld@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.

ALTERNATIVE to install - 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')
```

This says requires module express

```
var app = express()
```

Calls function express to initialize object app

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

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

```
app.listen(3000, function () {  
    console.log('Example app listening on port 3000!')  
})
```

Sets up the HTTP server for listening port 3000