

CS 572 Modern Web Applications

Najeeb Najeeb, PhD (najeeb@miu.edu)

Copyright © 2021 Maharishi International
University. All Rights Reserved.
V1.0.0



JavaScript Full Stack Development



- MongoDB
 - NoSQL database (document store)
 - Stores JSON documents
- Express
 - JavaScript web framework
 - On top of Node
- Angular
 - JavaScript UI framework
 - Single Page Applications
- Node
 - JavaScript server-side platform
 - Single threaded, fast and scalable

Full Stack Development

- Build the front end and back end of a website or web application.
- Front end: Interaction with browser.
- Back end: Interaction with database and server.
- Database driver application.

No Frameworks

- We will start with nothing and build up.
- No opinionated frameworks (you are advised to investigate these in the future)
 - MEAN.io
 - MEANjs
 - Express Generator
 - Yeoman
- Frameworks are good for complex projects and for advanced users not good for learning and understanding for beginners.

Roadmap and Outcomes

- Node.js: write asynchronous (non-blocking) code. Understand node platform to start a project.
- Express: setup express and get requests and send back responses. REST API.
- MongoDB: what NoSQL DB looks like. Full API interacting with DB.
- AngularJS: Investigate AngularJS and architect it. A single page application.
- MEAN application: Learn by example. We will create a MEAN Games application.



Demo MEAN Games



NodeJS

NodeJS and History

- Install Node from nodejs.org.
- Versions jumped from 0.x to 14.x
 - Due to the merge back from io.js to Node.js
 - Some original Node.js developers forked io.js why
 - community-driven development
 - Active release cycles
 - Use of semver for releases.
 - Node.js owned by Joyent had slow development, advisory board

Joyent Advisory Board

- Centralize Node.js to make development and future features faster.
- Board of large companies that use Node.js
- It moved Node.js from mailing lists and GitHub issues and developer's contribution to the power of the "big shots".
- Companies like Walmart, Yahoo, IBM, Microsoft, Joyent, Netflix, and PayPal were controlling things not the developer.
- The advisory board resulted in slower development and feature releases.

SEMVER

- Semantic Versioning
- MAJOR.MINOR.PATCH
- Major: incompatible API changes
- Minor: add backward compatible functionality
- Patch: add backward compatible bug fixes.

NodeJS

Check version

Run Node

Create and run
node file



Install node from nodejs.org

`node -v` (or `node --version`)

v14.13.1

Check node package manager (npm)

`npm -v`

6.14.8

Start node

`node`

Print "Hello World!" from node

`> console.log("Hello World!");`

Hello World!

NodeJS

Check version

Run Node

Create and run
node file



Start node

```
node
```

Print "Hello World!" from node

```
> console.log("Hello World!");
```

Hello World!

Write some JS

```
> var name = "Jack";
```

```
> console.log("Hello " + name);
```

Hello Jack

```
> .exit
```

NodeJS

Check version

Run Node

Create and run
node file

I use vsCode (it has a lot of MEAN plugins)

Create a file (instantHello.js)

```
var name = "Jack";
```

```
console.log("Hello " + name);
```

Run file

```
node hello.js
```

Hello Jack



Modular Programming

- Best practice to have building blocks
 - You do not want everything running from a single file (hard to maintain).
- Separate the main application file from the modules you build.
- Separate loading from invocation.
- Each module exposes some functionality for other modules to use.

Modular Node

Multi files Node
application

Require to load file

Expose functionality
using
`module.exports`

Create app01.js file

```
require("./instantHello");
```

Run file

```
node app01.js
```

Hello Jack



Modular Node

Multi files Node
application

Require to load file

Expose functionality
using
`module.exports`



Create goodbye.js file

```
module.exports = function(){  
  console.log("Goodbye");  
}
```

app01.js file

```
require("./instantHello");  
var goodbye = require("./goodbye");  
goodbye();
```

Run file

```
node app01.js
```

Hello Jack

Goodbye

Exports

- Export more than one function.
- Encapsulation; reducing side effects, improve code maintainability.
- Avoid using .js in require. This will enable changing the structure of your modules in the future. If a file becomes complex, we can put it in a folder by itself as a module and make index.js backwards compatible.
- When require searches (require(name)):
 - Search for name.js, if not found
 - Search for index.js in folder name
- Three ways to export
 - Single function
 - Multi functions
 - Return value

Module.exports

Single function
Multi functions
Return values



Create talk/index.js file

```
module.exports = function(){  
  console.log("Goodbye");  
}
```

app02.js file

```
require("./instantHello");  
var goodbye = require("./talk");  
goodbye();
```

Run file

```
node app02.js
```

Hello Jack
Goodbye

Module.exports

s

Single function

Multi functions

Return values



Create talk/index.js file

```
var filename = "index.js";  
var hello = function(name) {  
  console.log("Hello " + name);  
}  
var intro = function() {  
  console.log("I'm a node file called " + filename);  
}  
module.exports = {  
  greeting : hello,  
  intro : intro  
}
```

app02.js file

```
var goodbye = require("./talk");  
talk.greeting();  
talk.intro();
```

Run file

```
node app02.js
```

Hello Jack
I'm a node file called index.js

Module.exports

s

Single function

Multi functions

Return values



Create talk/question.js file

```
var answer = "This is a good question.";
module.exports.ask = function(question) {
  console.log(question);
  return answer;
}
```

app02.js file

```
var question= require("./talk/question");
var answer = question.ask("What is the meaning of life?");
console.log(answer);
```

Run file

```
node app02.js
```

What is the meaning of life?
That is a good question.

Single Threaded Node

- Node is single threaded.
 - One process to deal with all requests from all visitors.
- Node.js is designed to address I/O scalability (not computational scalability).
- I/O: reading files and working with DB.
- No user should wait for another users DB access.
- What if a user requests a computationally intense operation? (compute Fibonacci)
- Timers enable asynchronous code to run in separate threads. This enables scalable I/O operations. Perform file reading without everything else having to wait.

Async

setTimeout

readFileSync

readFileAsync

Named callback



app03.js file, setTimeout creates asynchronous code

```
console.log("1: Start app");  
var laterWork = setTimeout( function() {  
    console.log("2: In setTimeout");  
}, 3000);  
console.log("3: End app");
```

Run file

```
node app03.js
```

1: Start app

3: End app

2: In the setTimeout

Async

setTimeout

readFileSync

readFileAsync

Named callback



app04.js file

```
var fs= require("fs");  
console.log("1: Get a file");  
var file= fs.readFileSync("shortFile.txt");  
console.log("2: Got the file");  
console.log("3: App continues...");
```

Run file

```
node app04.js
```

```
1: Get a file  
2: Got the file  
3: App continues...
```

Async

setTimeout

readFileSync

readFileAsync

Named callback



app05.js file

```
var fs= require("fs");  
console.log("Going to get a file");  
fs.readFile("shortFile.txt", function(err, file) {  
    console.log("Got the file");  
});  
console.log("App continues...");
```

Run file

```
node app05.js
```

Going to get a file

App continues...

Got the file

Async

setTimeout

readFileSync

readFileAsync

Named callback



app06.js file

```
var fs= require("fs");  
var onFileLoad= function(err, file) {  
    console.log("Got the file");  
}  
console.log("Going to get a file");  
fs.readFile("shortFile.txt", onFileLoad);  
console.log("App continues...");
```

Run file

```
node app06.js
```

Going to get a file

App continues...

Got the file

Async

setTimeout

readFileSync

readFileAsync

Named callback



app06.js file

```
var fs= require("fs");  
var onFileLoad= function(err, file) {  
    console.log("Got the file");  
}  
console.log("Going to get a file");  
fs.readFile("shortFile.txt", onFileLoad);  
console.log("App continues...");
```

Run file

```
node app06.js
```

Going to get a file

App continues...

Got the file

Benefits of Named Callbacks

- Readability
- Testability
- Maintainability

Intense Computations

- Avoid delays in a single threaded application server.
- If someone performs a task that takes too long to finish, it should not delay everyone else on a webserver.
- Computation is not I/O operations. Computations need a process to perform the operation.
- Spawn a child process to perform the computation. This will consume resources, but it will not block the main server.

Computation

Fibonacci

Blocker

non-Blocker



_fibonacci.js file

```
var fib= function(number) {  
  if (number <= 2) {  
    return 1;  
  } else {  
    return fib(number-1) + fib(number-2);  
  } };  
console.log("Fibonacci of 42 is "+ fib(42));
```

Run file

```
node _fibonacci.js
```

Fibonacci of 52 is 267914296

Computation

Fibonacci

Blocker

non-Blocker



app07.js file

```
console.log("1: Start");  
require("../computation/_fibonacci");  
console.log("2: End");
```

Run file

```
node app07.js
```

Start

Fibonacci of 52 is 267914296

End

Computation

Fibonacci

Blocking

non-Blocking



app08.js file

```
var child_process= require("child_process");  
console.log("1: Start");  
var newProcess= child_process.spawn("node",  
["computation/_fibonacci.js"], {stdio : "inherit"});  
console.log("2: End");
```

Run file

```
node app08.js
```

Start

End

Fibonacci of 52 is 267914296

Node Package Management (npm)

- Define and manage dependencies using npm.
- Using packages enables code reuse, and not writing things from scratch.
- Move code around and use latest versions of dependencies.

Using npm

- Creating package.json can be done with `npm init`
- Follow the steps npm gives you.
- Entry point: this is the file that will contain the application starting point (the file to run).
 - We use (app.js)
- This creates package.json having all the information you provided.
- Use it to add dependencies, installing packages, development vs testing dependencies, run scripts.
- Ignoring dependencies when uploading to git.

npm

Create

Add

Development

Install

Scripts



How to create package.json file

```
npm init
```

```
package name: (app09)
```

```
version: (1.0.0)
```

```
description: This is my first npm project
```

```
entry point: (index.js) app09.js
```

```
test command:
```

```
git repository:
```

```
keywords: mean
```

```
author: Najeeb Najeeb
```

```
license: (ISC)
```

```
Is this OK? (yes)
```

```
npm create package.json
```

```
package.json
```

npm

Create

Add

Development

Install

Scripts



Add dependency on Express (using npm command line)

```
npm install express --save
```

```
+ express@4.17.1
```

npm added express to package.json

```
ls
```

```
node_modules
```

```
"license": "ISC",
```

```
"dependencies": {
```

```
  "express": "^4.17.1"
```

```
}
```

^x.y.z: use x major and the latest minor and patch.

npm

Create

Add

Development

Install

Scripts



Add dependency on Express (using npm command line)

```
npm install mocha -save-dev  
+ express@4.17.1
```

npm added express to package.json

...

```
"license": "ISC",  
"dependencies": {  
  "express": "^4.17.1"  
},  
"devDependencies": {  
  "mocha": "^8.2.0"  
}
```

^x.y.z: use x major and the latest minor and patch.

npm

Create

Add

Development

Install

Scripts



Dependencies are not uploaded to git

Dependencies should be installed after fetching code from git

```
npm install
```

Install only production dependencies (on production server)

```
npm install --production
```

Create readme.md

```
"This repo contains the MEAN stack application that is built in  
CS572 Modern Web Applications course."
```

Ignore node_modules when pushing to git.

Create .gitignore file and fill it with

```
node_modules
```

npm

Create

Add

Development

Install

Scripts



Start script; shortcut to start your application.

```
"scripts": {  
  "start": "node app09.js",  
  "test": "echo \"Error: no test specified\" && exit 1"  
}
```

To start the application:

`npm start`

> app09@1.0.0 start

/home/cs572/CS572/Lessons/Lesson1/app09

> node app09.js

1- App Started

2- App Ended

What is Express

- Web framework for MEAN stack.
- Listen to incoming requests and respond.
- Deliver static html files.
- Compile and deliver html.
- Return JSON.

Express Application

- Add dependency on Express.
- Require Express.
- Listen to requests (port) at URLs.
- Return HTTP status codes.
- Response HTML or JSON.

Express

Add

Listen

Application

Variables

Callback



Create package.json

```
npm init
```

Add dependency on Express (using npm command line)

```
npm install express -save
```

app10.js file

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

Run the application:

```
npm start
```

The server terminates before we send a request!

Express

Add

Listen

Application

Variables

Callback



app10.js file

```
var express= require("express");  
var app= express();  
app.listen(3000); // Hardcoded more than one place :(  
console.log("Listening to port 3000"); // Another place :(
```

Run the application

```
npm start
```

Check the browser (<http://localhost:3000>)

Nothing interesting, but we do have a server.

Express

Add Listen Application Variables Callback



app10.js file

```
var express= require("express");  
var app= express();  
app.set("port", 3000); // In one place  
app.listen(app.get("port");  
console.log("Listening to port "+ app.get("port");
```

Run the application

```
npm start
```

Check the browser (<http://localhost:3000>)

Same results but better software engineering, right?

Express

Add Listen Application Variables Callback



app10.js file

```
var express= require("express");  
var app= express();  
app.set("port", 3000); // In one place  
var server= app.listen(app.get("port"), function() {  
    var port= server.address().port;  
    console.log("Listening to port "+ port);  
});
```

Run the application

`npm start`

Check the browser (<http://localhost:3000>)

Is this really a callback?