

# CS 572 Modern Web Applications

Najeeb Najeeb, PhD ([najeeb@miu.edu](mailto:najeeb@miu.edu))

Copyright © 2021 Maharishi International  
University. All Rights Reserved.  
V1.1.0



# Syllabus

- Course Goal
- Course Schedule
- Grading
- Exam Objection Policy
- Your Personal Goals During any Course

# 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

# Installation

- NodeJS
  - [nodejs.org](https://nodejs.org)
- Mongo DB
  - [mongodb.com](https://mongodb.com)
- IDE
  - [vscode.com](https://vscode.com)





NodeJS

# NodeJS and History

- Install Node from [nodejs.org](https://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](https://nodejs.org)

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

v14.17.1

Check node package manager (npm)

`npm -v`

6.14.13

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

```
> const 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)

```
const username = "Jack";
```

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

Run file

```
node instantHello.js
```

Hello Jack

# Modular Programming

- Best practice is to have building blocks
  - You do not want everything running from a single file (hard to maintain, test).
- 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");  
const 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("Hi");  
}
```

app02.js file

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

Run file

```
node app02.js
```

Hello Jack

Hi

# Module.exports

s

Single function

Multi functions

Return values



Create talk/index.js file

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

app02.js file

```
const talk = require("./talk");
talk.greeting("Jim");
talk.intro();
```

Run file

```
node app02.js
```

Hello Jack

Hello Jim

I'm a node file called index.js

# Module.exports

s

Single function

Multi functions

Return values



Create talk/question.js file

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

app02.js file

```
const question= require("./talk/question");
const 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");  
const 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

```
const fs= require("fs");  
console.log("1: Get a file");  
const 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

```
const 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

```
const fs= require("fs");  
const 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

Blocking

non-Blocking



./computation/\_fibonacci.js file

```
const fib= function(number) {  
  if (number <= 2) {  
    return 1;  
  } else {  
    return fib(number-1) + fib(number-2);  
  } };  
console.log("Fibonacci of 45 is "+ fib(45));
```

Run file

```
node _fibonacci.js
```

Fibonacci of 45 is 1134903170

# Computation

Fibonacci

Blocking

non-Blocking



app07.js file

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

Run file

```
node app07.js
```

Start

Fibonacci of 45 is 1134903170

End

# Computation

Fibonacci

Blocking

non-Blocking



app08.js file

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

Run file

```
node app08.js
```

Start

End

Fibonacci of 45 is 1134903170



# 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"  
}
```

Create app09.js

```
console.log("1: App Started");  
console.log("2: App Ended");
```

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 (send response).
- 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 start script

```
"start": "node app10.js",
```

Add dependency on Express (using npm command line)

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

app10.js file

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

Run the application:

```
npm start
```

The server terminates before we send a request!

# Express

Add

Listen

Application

Variables

Callback



app10.js file

```
const express= require("express");  
const 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

```
const express= require("express");  
const 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

```
const express= require("express");  
const app= express();  
app.set("port", 3000); // In one place  
const server= app.listen(app.get("port"), function(){  
  const 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?

# Routing using Express

- Routing is listening to requests on certain URLs and doing something on the server side then sending a response back.
- Route definition
  - HTTP method
  - Path
  - Function to run when route is matched

# Routing

Define

HTTP Status

Data Response

File Response



app11.js file

```
var express= require("express");  
var app= express();  
app.set("port", 3000);  
app.get("/", function(req, res) {  
  console.log("GET received");  
});  
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>)

Are you getting a response? Is the server getting the request?

# Routing

Define

HTTP Status

Data Response

File Response



app11.js file

```
var express= require("express");  
var app= express();  
app.set("port", 3000);  
app.get("/", function(req, res) {  
  console.log("GET received");  
  res.send("Received your GET request.");  
});  
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>)



# Routing

Define

HTTP Status

Data Response

File Response



app11.js file

```
var express= require("express");
var app= express();
app.set("port", 3000);
app.get("/", function(req, res) {
  console.log("GET received");
  res.status(404).send("Received your GET request.");
});
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>)

# Routing

Define

HTTP Status

Data Response

File Response



app11.js file

```
app.get("/", function(req, res) {  
  console.log("GET received");  
  res.status(404).send("Received your GET request.");  
});  
app.get("/json", function(req, res) {  
  console.log("JSON request received");  
  res.status(200).json({"jsonData": true});  
}
```

Run the application

```
npm start
```

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

# Routing

Define

HTTP Status

Data Response

File Response



app11.js file

```
const path= require("path");  
app.get("/file", function(req, res) {  
  console.log("File request received");  
  res.status(200).sendFile(path.join(__dirname,  
"app11.js"));  
});
```

Run the application

```
npm start
```

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

# MEAN Games

- Create package.json
- Add Express using npm
- Set your start script (we will use app.js as our starting point)
- Create HTML file
- Create app12.js to send the home page back.
- No CSS :( no images :(

# MEAN Games

public/index.html



```
<!DOCTYPE html>
<html>
  <head>
    <title>MEAN Games</title>
  </head>
  <body>
    <h1>MEAN Games
    homepage.</h1>
  </body>
</html>
```

# MEAN Games

## app12.js



```
const express= require("express");
const path= require("path");
const app= express();
app.set("port", 3000);
app.get("/", function(req, res) {
  console.log("GET received.");
  res.status(200).sendFile(path.join(__dirname,
    "public", "index.html"));
});
const server= app.listen(app.get("port"),
  function() {
    const port= server.address().port;
    console.log("Listening to port "+ port);
  });
```

# Express Serving Static Files

- Applications require foundations
  - HTML pages
  - JS libraries
  - CSS files
  - Images
- Easier to deliver static pages through Express directly.

# Static Pages

## Folder

Subset of routes

CSS

JS

IMG



app12.js file, after port definition and before routes we define the static folder (introduce middleware)

```
app.use(express.static(path.join(__dirname, "public")));
```

Run the application

```
npm start
```

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

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



# Static Pages

## Folder

Subset of routes

CSS

JS

IMG



app12.js file, after port definition and before routes we define the static folder (introduce middleware)

```
app.use("/public", express.static(path.join(__dirname, "public")));
```

Run the application

```
npm start
```

Check the browser

(<http://localhost:3000/public/index.html>)

# Static Pages

## Folder

Subset of routes

CSS

JS

IMG



CSS bootstrap theme available from  
[www.bootswatch.com/superhero](http://www.bootswatch.com/superhero) (bootstrap.min.css)

Link CSS file to html file

```
<link href="css/bootstrap.min.css" rel="stylesheet" />
```

Run the application

```
npm start
```

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

**Static Pages**

**Folder**

Subset of routes

CSS

jQuery

IMG

jQuery from [www.jquery.com/download/](http://www.jquery.com/download/) (jquery-3.5.1.min.js)

Reference jquery in the page

```
<script src="jquery/jquery-3.5.1.min.js"/>
```

Run the application

```
npm start
```

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



# Static Pages

## Folder

Subset of routes

CSS

jQuery

IMG

Create images folder, Copy your image into the folder (MIU logo)

Create custom.css

Add image to your page

Run the application

`npm start`

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



# Static Pages

## Folder

Subset of routes

CSS

jQuery

IMG



custom.css

```
html {  
    position: relative;  
    min-height: 100%;  
}  
body {  
    margin-bottom: 90px;  
}  
.padded {  
    padding-top: 30px;  
}
```

custom.css

```
.footer {  
    position: absolute;  
    bottom: 0;  
    width: 100%;  
    height: 105px;  
    background-color:  
    #f5f5f5;  
    padding-top: 5px;  
}
```

Static Pages

Folder

Subset of routes

CSS

JQuery

IMG



index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>MEAN
Games</title>
    <link
href="css/bootstrap.min.css"
rel="stylesheet" />
    <link
href="css/custom.css"
rel="stylesheet" />
  </head>
  <body>
    <h1>MEAN Games
homepage.</h1>
    <footer class="footer">
      <div class="container">
        <p class="text-muted
text-center">
```

Index.html

```
      <a
href="https://compro.miu.edu"
target="_blank"></a>
      <br/>
      <small class="text-
black-50 text-center">&copy;
2021 Maharishi International
University. All Rights Reserved.
</small>
    </p>
  </div>
</footer>
<script
src="jquery/jquery-
3.5.1.min.js"> </script>
  </body>
</html>
```

# Express & Middleware

- What is middleware?
- Create logging function
- When and how to use middleware

# Express & Middleware

- Example: `app.use`
  - Interact with request before response
  - Make the response, or passes it through
- Define a function that will process something in the request, do something, then follow through to the response.
- Order is important, they will run in the order defined.



# Middleware

log requests

Order

Subsets



app13.js file, middleware (explicit)

```
app.use(function(req, res, next) {  
  console.log(req.method, req.url);  
  next();  
});
```

Run the application

```
npm start
```

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

GET /

GET /css/bootstrap.min.css

GET /css/custom.css

GET /jquery/jquery-3.5.1.min.js

GET /images/xompro-web-logo-442x112.png

# Middleware

Log requests

Order

Subsets



app13.js file, middleware (explicit)

```
app.use("/public",  
express.static(path.join(__dirname, "public")));
```

```
app.use(function(req, res, next) {  
  console.log(req.method, req.url);  
  next();  
});
```

Run the application

```
npm start
```

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

# Middleware

Log requests

Order

Subsets



app13.js file, middleware for only paths starting with "css"

```
app.use("/css", function(req, res, next) {  
  console.log(req.method, req.url);  
  next();  
});
```

Run the application

```
npm start
```

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

GET /bootstrap.min.css

GET /custom.css

# Express Router

- Separation of concerns
- Instantiating the router
- Applying router to subset of routes
- Testing routes using REST plugins

# Express Router

- Keep app.js clean and clear
  - Easy to read and understand
  - Easy to maintain and debug
- Don't put too much code of different types in one single file.
- Move different code to different places and keep them separate.

# Router

Separate routes

Subset routes

REST Test



app13.js file, this is what we have (everything in one place)

```
const express= require("express");
const app= express();
const path= require("path");
app.set("port", 3000);
app.use(function(req, res, next) {
  console.log(req.method, req.url);
  next();
});
app.use(express.static(path.join(__dirname, "public")));
app.get("/json", function(req, res) {
  console.log("JSON request received");
  res.status(200).json({"jsonData": true});
}
app.get("/file", function(req, res) {
  console.log("File request received");
  res.status(200).sendFile(path.join(__dirname, "app13.js"));
});
var server= app.listen(app.get("port", function() {
  const port= server.address().port();
  console.log("Listening to port "+ port);
}));
```

# Router

Separate routes

Subset routes

REST Test



Create routes folder, and inside it index.js

```
const express= require("express");
const router= express.Router();
router.route("/json").get(function(req, res) {
  console.log("JSON request received");
  res.status(200).json({"jsonData": true});
}).post(function(req, res) {
  console.log("POST json route request received");
  res.status(200).json({"jsonData": true});
});
module.exports = router;
```

app14.js file, this is what we have (everything in one place)

```
const express= require("express");
const path= require("path");
const routes= require("./routes");
const app= express();
app.set("port", 3000);
app.use(function(req, res, next) {
  console.log(req.method, req.url);
  next();
});
app.use(express.static(path.join(__dirname, "public")));
app.use("/", routes);
const server= app.listen(app.get("port"), function() {
  const port= server.address().port();
  console.log("Listening to port "+ port);
});
```

# Router

Separate routes

Subset routes

REST Test



Create routes folder, and inside it index.js

```
const express= require("express");
const router= express.Router();
router.route("/json").get(function(req, res) {
  console.log("JSON request received");
  res.status(200).json({"jsonData": true});
}).post(function(req, res) {
  console.log("POST json route request received");
  res.status(200).json({"jsonData": true});
});
module.exports = router;
```

app14.js file, this is what we have (everything in one place)

```
const express= require("express");
const path= require("path");
const routes= require("./routes");
const app= express();
app.set("port", 3000);
app.use(function(req, res, next) {
  console.log(req.method, req.url);
  next();
});
app.use(express.static(path.join(__dirname, "public")));
app.use("/api", routes);
const server= app.listen(app.get("port"), function() {
  const port= server.address().port();
  console.log("Listening to port "+ port);
});
```



# Router

Separate routes

Subset routes

REST Test

Add a Chrome REST Client extension

I picked Advanced REST client

Make GET request from browser (<http://localhost:3000/>)

Make GET request from REST Client

Make POST request from REST Client



# Express Controller

- Separation of Concerns
- Creating API (REST API)
- What are controllers and their functionality
  - Controls what happens when a route is visited.
  - Separate logic from routing from UI code.
- Map controllers to routes.

# Controller Setup Static Data



Create api folder, move routes folder inside it.

index.js file

```
const express=
require("express");
const router= express.Router();
const controllerGames=
require("../controllers/games.controllers.js");
router.route("/games").get(controllerGames.gamesGetAll);
module.exports = router;
```

Create controllers folder in api, with file games.controllers.js

```
module.exports.gamesGetAll=
function(req, res) {
  console.log("JSON request received");
  res.status(200).json({"jsonData": true});
};
```

app15.js file

```
const express= require("express");
const path= require("path");
const routes= require("./api/routes");
const app= express();
app.set("port", 3000);
app.use(function(req, res, next) {
  console.log(req.method, req.url);
  next();
});
app.use(express.static(path.join(__dirname, "public")));
app.use("/api", routes);
const server= app.listen(app.get("port"), function() {
  const port= server.address().port();
  console.log("Listening to port "+ port);
});
```

Run the application

npm start

Check the browser

(<http://localhost:3000/api/games>)

GET api/games

json GET request

# Controller

## Setup

### Static Data



Create data folder inside api, create json data file.

games-data.js file

games.controllers.js

```
const gamesData= require("../data/games-data.json");
module.exports.gamesGetAll= function(req, res) {
  console.log("GET all games");
  res.status(200).json(gamesData);
}
```

Run the application

```
npm start
```

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

GET api/games

GET all games

# URL parameters in Express

- What are URL parameters?
  - How can you get information about one game?
    - You need to know the game of interest (user input).
  - Get user input through the URL (localhost:3000/api/games/2021).
    - Create a route for each id? :(
    - Parametrize it :)
- How to define URL parameters in routes.
  - `.route("/games/:gameId")`
- Use URL parameters in controllers.

URL

parameter

Router

Controller

api/routes/index.js add

```
router.route("/games/:gameId").get(controllerGames.games  
GetOne);
```



URL  
parameter  
Router  
Controller



api/controllers/games.controllers.js add

```
module.exports.gamesGetOne= function(req, res) {  
  const gameId= req.params.gameId;  
  const theGame= gamesData[gameId];  
  console.log("GET game with gameId ", gameId);  
  res.status(200).json(theGame);  
}
```

Run the application

npm start

Check the browser (<http://localhost:3000/api/games/3>)

GET api/games/3

GET game with gameId 3

# Other Ways to get Input

- How to pass data from client to server?
  - URL parameter (Express native support)
  - Query string (GET method, Express native support)
  - Form body (POST method, Express no native support)
- Getting queryString data in Express controllers.
- Middleware for parsing forms.
- Getting form data in Express controllers.



# Client Data

## Query string

## Form data



Get certain number of games, for pagination, start from an offset and get a certain number of games

Browser (<http://localhost:3000/api/games?offset=3&count=2>)

Games.controller.js

```
module.exports.gamesGetAll= function(req, res) {  
  console.log("GET the games");  
  console.log(req.query);  
  var offset= 0;  
  var count= 5;  
  if (req.query && req.query.offset) {  
    offset= parseInt(req.query.offset, 10);  
  }  
  if (req.query && req.query.count) {  
    count= parseInt(req.query.count, 10);  
  }  
  const pageGames= gamesData.slice(offset, offset+count);  
  res.status(200).json(pageGames);  
}
```

Run the application

`npm start`

Check the browser (<http://localhost:3000/api/games?offset=3&count=2>)

GET api/games/3

GET game with gameId 3

# Client Data

## Query string

## Form data



Form body parsing is available on Express. But not setup by default. We need to set it up.

app18.js add the followings

```
...  
app.use(express.static(path.join(__dirname, "public")));  
app.use(express.urlencoded({extended : false}));  
app.use(express.json({extended : false}));  
app.use("/api", routes);
```

Add new route, api/routes/index.js

```
router.route("/games/new").post(controllerGames.gamesAddOne);
```

Add the controller, api/controllers/gamesController.js

```
module.exports.gamesAddOne= function(req, res) {  
  console.log("POST new game");  
  console.log(req.body);  
  res.status(200).json(req.body);  
}
```

Use boomerangapi (<http://localhost:3000/api/games/new>)

# Nodemon

- Development tool, not for production system.
- Improve development experience and provide information.
- Install Nodemon globally (not related to an application).
- Use Nodemon.
- Configure Nodemon.

# Nodemon

Install

Run

Configure

Code and tests without having to always stop and start application.

Install nodemon

```
sudo npm install -g nodemon
```



# Nodemon

Install

Run

Configure

Run nodemon, run the start command in package.json

`nodemon`

Change something in app19.js and see how nodemon restarts the application.



# Nodemon

Install

Run

Configure



Nodemon monitors everything, including out static files. But we want them served as is. Configure nodemon to ignore changes made in public directory.

Create nodemon.json

```
{  
  "ignore" : ["public/*"],  
  "verbose" : true  
}
```

Change something in public folder and see how nodemon doesn't restarts the application.

Shows the file that triggered the change.

# Main Points



- NodeJS is a single threaded Java Script platform. NodeJS enables the use of JavaScript for full stack development.
- Express is a JavaScript web framework that enables the development of request-response-based applications.
- Separation of concerns is achieved in Express using routers and controllers. This enables the development of more complex application. Routers and controllers enable easier understanding and debugging of applications.