CS 572 Modern Web Applications

Najeeb Najeeb, PhD (najeeb@miu.edu)

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

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.

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?

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

```
var 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 app.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

app.js

```
var express= require("express");
var path= require("path");
var 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"));
});
var server= app.listen(app.get("port"), function() {
    var 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 theam available from 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/ (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="test-muted text-center">
```

Index.html

```
            <a href="https://compro.miu.edu" target="_blank"><img src="/images/compro-web-logo-442x112.png" height="60" alt="MIU Compro"></a>
            <br/>
            <small class="text black-50 text-center">&copy; 2020 Maharishi International University. All Rights Reserved.</small>
          </p>
        </div>
    </footer>
    <script src="jquery/jquery-3.5.1.min.js"> </script>
  </body>
```

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)

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

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

```
var express= require("express");
var path= require("path");
var routes= require("./routes");
var 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);
var server= app.listen(app.get("port",
function() {
    var 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

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

```
var express= require("express");
var path= require("path");
var routes= require("./routes");
var 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);
var server= app.listen(app.get("port",
function() {
    var port= server.address().port();
    console.log("Listening to port "+ port);
});
```

Router

Separate routes

Subset routes

REST Test

Add a Chrome REST extension

I picked Boomerang SOAP & 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 thier functionality
  - Controles 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

```
var express= require("express");
var router= express.Router();
var controllerGames=
require("../controllers/games.cont
rollers.js");
router.route("/games").get(contro
llerGames.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

```
var express= require("express");
var path= require("path");
var routes= require("./api/routes");
var 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(__dirnam
e, "public");
app.use("/api", routes);
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/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

```
var 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) {
    var gameId= req.params.gameId;
    var theGame= gameData[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);
   }
   var pageGames= gameData.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 not natively supported by Express. We need a library to parse form body.

Install body-parser

```
npm install --save body-parser
```

app18.js add the followings

```
var bodyParser= require("body-parser");
...
app.use(express.static(path.join(__dirname, "public")));
app.use(bodyParser.urlencoded({extended : false});
app.use("/api", routes);
```

Add new route, api/routes/index.js

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

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.