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

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 mke index.js backwards compatible.
- When require searches (require(name)):
  - Serach 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

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

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

Insall 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?

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.

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.

NoSQL DB

NoSQL Database Types

- Key-value store, ArangoDB
  - Store unique key and value, high scalability for caching (session management)
- Document store, MongoDB
  - Store semi-structured data in document format, no schema insert(mobile applications)
- Wide- column store, Amazon DynamoDB
  - Store in columns not rows, fast (catalogs, recommendation engines)
- Graph databases, Amazon Neptune
  - Store data as nodes and edges, show connections (reservation systems)
- More

Document Store vs Relational DB

RELATIONAL DB

STUNDET_ID	NAME	GPA
1	Jack	3.0
2	Jill	3.3
3	John	2.8

ID	COURSE_NAME	STUDENT_ID
1	Software Engineering	1
2	Web Programming	2
3	Algorithms	2

DOCUMENT STORE


```
[
 { "StudentID" : 1,
   "Name" : "Jack",
   "GPA" : 3.0,
   "Courses" : [
    { "ID" : 1,
      "CourseName" : "Software Engineering" } ] },
 { "StudentID" : 2,
   "Name" : "Jill",
   "GPA" : 3.3 },
   "Courses" : [
    { "ID" : 2,
      "CourseName" : "Web Programming" },
    { "ID" : 3,
      "CourseName" : "Algorithms" } ] },
 { "StudentID" : 3,
   "Name" : "John",
   "GPA" : 2.8 }
]
```

NoSQL DB Design

- What is all the data you wish to output (at once) on a pages.
  - Put that information in one place.
- If on some page you wish to display some of the information from another document.
  - Add what needs to be displayed and include an ID to link to the other document.
- Optimize for the most common operation.
  - Reduce updates for the most common changeable items.
  - Increase speed of displaying most common pages.
- Keep number of Collections (Tables) to a minimum.
- Try to reduce each page to one collection (or minimum number of joined collections)
- Most common operations must run faster (even at the expense of less common operations)

MongoDB

MongoDB Collections

REVIEW.JSON


```
[
{ "ReviewID" : 1,
  "Title" : "Good Game.",
  "Review" : "I enjoyed the game.",
  "Stars" : 4,
  "Game" : {
    "ID" : 1,
    "Name" : "Trains"}
},
{ "ReviewID" : 2,
  "Title" : "Too Long.",
  "Review" : "The game is nice, but it was too long.",
  "Stars" : 3,
  "Games" : {
    "ID" : 2,
    "Name" : "Monopoly"}
}]
```


GAME.JSON


```
[{ "ID" : 1,
   "Name" : "Trains",
   "Price" : 48.82,
   "MinPlayers": 2,
   "MaxPlayers": 4,
   "EstimatedTimeToPlay": 45,
   "ReleaseYear": 2013},
{  "ID" : 2,
   "Name" : "Monopoly",
   "Price" : 29.97,
   "MinPlayers": 2,
   "MaxPlayers": 8,
   "EstimatedTimeToPlay": 180,
   "ReleaseYear": 1933},
{  "ID" : 3,
   "Name" : "Risk",
   "Price" : 20.99,
   "MinPlayers": 2,
   "MaxPlayers": 6,
   "EstimatedTimeToPlay": 120,
   "ReleaseYear": 1959}
]
```

How to Design a Document

- Why not have one Collection and store everything in it?
  - Not good logically and performance.
  - Hard to maintain.
- A review is for a game, so why not only have one Collection of Games.
  - A review can exists by itself.
  - Get all positive reviews, negative, …
  - A Game could also have several reviews.
- Collections may reference each other.
- You do not use a collection to get data from another collection.
  - What you want from another collection embed in your collection.

JSON and BSON

- JSON is what you use in your application.
- JSON is a close representation of what MongoDB stores.
- BSON is Binary-JSON, it is what MongoDB uses.
- BSON not human readable but maintains the flexibility and ease of use of JSON plus the speed of binary format.
- MongoDB accepts JSON and returns JSON (but stores it as BSON).

JSON ID

- MongoDB creates unique ID for a document when created.
- _id property is what MongoDB creates.
- The value is ObjectId("5f9aef68980db44d37c1aaed") unique combination of time (Unix epoch) , machine ID, process ID, and counter.

Install and Work With MongoDB

- Install from MongoDB website (www.mongodb.com/try/download)
- Running MongoDB
  - mongo --version
  - mongo
  - exit (or Ctrl + C)
- Create Database
- Create Collection
- Retrieve Collection

MongoDB
Database
Collection

List all databases on your system

```
show dbs
admin 0.000GB
config 0.000GB
local   0.000GB
```

Select database to work with

```
use local
switched to db local
```

Create new database, make sure it does not exist

```
use newTestDb
switch to newTestDB
```

Note: new database not created until you add a collection to it.

Get the current database being used

```
db (or db.getName();)
newTestDB
```

Delete database

```
db.dropDatabase();
{ "dropped" : "newTestDb", "ok" : 1}
```

MongoDB
Database
Collection

List collections in current database

```
use local
show collections
startup_log
use newTestDB
show collections
```

Create collection

```
db.createCollection("technology")
{ "ok" : 1}
```

Delete collection

```
db.technology.drop()
true
```

CRUS

Crteate

Read

Update

Delete

Add document in current collection

```
db.technology.insert(
... {
... name : "MongoDB",
... role : "Database"
... }
... );
WriteResult({ "nInserted" : 1 })
```

List documents in current collection

```
db.technology.find();
{ "_id" : ObjectId("5f9aef68980db44d37c1aaed"), "name" : "MongoDB",
"role" : "Database" }
db.technology.find().pretty();
```

Insert multiple documents at once

```
db.technology.insert([{name : "Express", role: "Web application server"},
... {name : "Angular", role: "Front-end framework"},
... {name : "Node.js", role: "Platform"}]);
BulkWriteResult({ ... "nInserted" : 3 ... })
```

CRUS

Crteate

Read

Update

Delete

List all documents in current collection

```
db.technology.find();
{ "_id" : ObjectId("5f9aef68980db44d37c1aaed"), "name" : "MongoDB",
"role" : "Database" }
db.technology.find().pretty();
```

List based on document id in current collection

```
db.technology.find({"_id" : ObjectId("5f9aef68980db44d37c1aaed")});
{ "_id" : ObjectId("5f9aef68980db44d37c1aaed"), "name" : "MongoDB",
"role" : "Database" }
```

List based on name in current collection

```
db.technology.find({"name" : Angular")});
{ "_id" : ObjectId("5f9af651980db44d37c1aaef"), "name" : "Angular",
"role" : "Front-end framework" }
```

Sorting, 1 for assending -1 for decending

```
db.technology.find().sort({"name" : 1});
```

Limit returned fields, projection (the second parameter in find).

```
db.technology.find({}, {"name" : true});
db.technology.find({}, {"name" : true, "_id" : false});
```

CRUS

Crteate

Read

Update

Delete

Update a document , finds the documents of interest the updates them. The first parameter is the query, the second is the data to set.

```
db.technology.update( {"name" : "Angular"}, {$set : {"name" : AngularJS"} } );
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Update more than one document at once

```
db.technology.update({},{$set:{"language":JavaScript"}},{multi:true}  );
{ "_id" : ObjectId("5f9aef68980db44d37c1aaed"), "name" : "MongoDB", "role" : "Database", "language" : "JavaScript" }
{ "_id" : ObjectId("5f9af651980db44d37c1aaee"), "name" : "Express", "role" : "Web application server", "language" : "JavaScript" }
{ "_id" : ObjectId("5f9af651980db44d37c1aaef"), "name" : "Angular", "role" : "Front-end framework", "language" : "JavaScript" }
{ "_id" : ObjectId("5f9af651980db44d37c1aaf0"), "name" : "Node.js", "role" : "Platform", "language" : "JavaScript" }
```

CRUS

Crteate

Read

Update

Delete

Delete document from collection, you provide a query object

```
db.technology.remove( { "name" : "Express" })
WriteResult({ "nRemoved" : 1 })
```

```
db.technology.remove( {})
```

This will remove all the documents from the collection :(

Import & Export Data

BSON

Import

Export

Import MongoDB data from BSON file

```
mongorestore --db newTestDB2 --gzip dump/newTestDb
2020-11-01T13:46:25.982-0800    building a list of
collections to restore from dump/newTestDb dir
2020-11-01T13:46:25.987-0800    reading metadata for
newTestDb2.technology from
dump/newTestDb/technology.metadata.json.gz
2020-11-01T13:46:26.058-0800    restoring
newTestDb2.technology from
dump/newTestDb/technology.bson.gz
2020-11-01T13:46:26.076-0800    no indexes to restore
2020-11-01T13:46:26.076-0800    finished restoring
newTestDb2.technology (4 documents)
2020-11-01T13:46:26.076-0800    done
```

BSON

Import

Export

Export MongoDB data as BSON file

```
mongodump --db newTestDB
writting newTestDb.technology to
done dumping newTestDb.technology (4 documents)
/dump/newTestDb/technology.bson
```

Compress the BSON output data

```
mongodump --db newTestDB --gzip
/dump/newTestDb/technology.bson.gz
```

JSON

Export

Import

Export MongoDB data as JSON file (for a collection only)

```
mongoexport --db newTestDB --collection technology
{"_id":{"$oid":"5f9aef68980db44d37c1aaed"},"name":"MongoDB","role":"Databas
e","language":"JavaScript"}

{"_id":{"$oid":"5f9af651980db44d37c1aaee"},"name":"Express","role":"Web
    application server","language":"JavaScript"}

{"_id":{"$oid":"5f9af651980db44d37c1aaef"},"name":"Angular","role":"Front-end
    framework","language":"JavaScript"}

{"_id":{"$oid":"5f9af651980db44d37c1aaf0"},"name":"Node.js","role":"Platform","l
    anguage":"JavaScript"}

2020-11-01T13:54:18.608-0800    exported 4 records
```

Export to file

```
mongoexport --db newTestDB –collection technology --out output/technology.json
exported 4 records
```

Export as an array

```
mongoexport --db newTestDB –collection technology --out output/technology.json
--jsonArray --pretty
exported 4 records
```

JSON

Export

Import

Import MongoDB data from JSON file

```
mongoimport --db newTestDB3 --collection technology --
jsonArray output/technology.json
```

imported 4 documents

Connecting MongoDB to NodeJS

MongoDB & NodeJS

- Installing mongoDB driver in our app.
- Createing reusable connections.
- Defining connection string.
- Accessing connections from controllers.
- Best practices while doing all this.

Connect to DB

Install driver

Connections

Use DB

Install MongoDB native driver

```
npm install mongodb --save
```

```
mongodb@2.1.7 node_modules/mongodb
```

Connect to DB

Install driver

Connections

Use DB

Create file to manage connections,
File api/data/dbconnection.js

```
var MongoClient= requires("mongodb").MongoClient;
var dbName= "meanGamesDb";
var dburl= "mongodb://localhost:27017/"+dbName;
var_connection= null;
var open= function(){
    MongoClient.connect(dburl,{useUnifiedTopology: true}, function(err, client){
        if(err){
            console.log("DB connection failed");
            return;
        }
        _connection= vlirny.db(dbName);
        console.log("DB connection open", _connection);
    });
};
var get= function(){
    return _connection;
};
module.exports= {
    opend : open,
    get : get
};
```

Connect to DB

Install driver

Connections

Use DB

Open the connection as soon as the application starts, app.js

```
require("./api/data/dbconnection.js).open();
```

Run

```
npm start
```

DB connection open ....

Check for error, change the port number in dbconnection.js and run again.

Connect to DB

Install driver

Connections

Use DB

Use the db connection in the controllers.
api/controllers/games.controllers.js

```
var dbConnection= require("../data/dbconnection.js");

… gameGetAll= ..
var db= dbConnection.get();
console.log("db", db);
```

Run on browser (http://localhost:3000/api/games)

```
npm start
db …
```

Opening db is asynchronous. So make sure you get it when you need it. Don't just open it at the start of the file.

Opening db connection is slow. Best to open it once at application start and reuse it.

No need for a global variable for db. Encapsulated in dbconnection.

Working with MongoDB in NodeJS

Query DB

GetAll

Pagination

GetOne

Use the db connection in the controllers.

api/controllers/games.controllers.js

```
… gameGetAll= ..
var collection= db.collection("games");
// var docs= collection.find(); //Sync not good :(
collection.find().toArray(function(err, docs) {
    console.log("Found games", docs);
    res.status(200).json(docs);
}
```

Run on browser (http://localhost:3000/api/games)

```
npm start
```

Found games …

Query DB

GetAll

Pagination

GetOne

Use the db connection in the controllers.
api/controllers/games.controllers.js

```
… gameGetAll= ..
var collection= db.collection("games");
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);
}
collection.find().skip(offset).limit(count).toArray(function(err, docs) {
    console.log("Found games", docs);
    res.status(200).json(docs);
}
```

Run on browser (http://localhost:3000/api/games)

```
npm start
Found games …
```

Query DB

GetAll

Pagination

GetOne

Use the db connection in the controllers.

api/controllers/games.controllers.js

```
var ObjectId= require("mongodb").ObjectId;
… gameGetOne= ..
var db= dbConnection.get();
var collection= db.collection("games");
var gameId= req.params.gameId;
collection.findOne({_id : ObjectId(gameId)}, function(err,
doc) {
    console.log("Found game", doc);
    res.status(200).json(doc);
}
```

Run on browser (http://localhost:3000/api/games)

```
npm start
```

Found games …

Insert DB

Error Checking

InsertOne

Use the db connection in the controllers.

api/controllers/games.controllers.js

```
var ObjectId= require("mongodb").ObjectId;
… gameAddOne= ..
var db= dbConnection.get();
var collection= db.collection("games");
if (req.body && req.body.name && req.body.starts) {
    console.log(req.body);
    res.status(200).json(req.body);
} else {
    console.log("Data missing from POST body");
    res.status(400).json({error : "Required data missing from
POST"});
}
```

Run app.boomerangapi.com/workspace on browser

```
npm start
```

error: "Required data missing from POST" …

Insert DB
Error Checking
InsertOne

Use the db connection in the controllers.

api/controllers/games.controllers.js

```
... gameAddOne= ..
var newGame;
if (req.body && req.body.name && req.body.starts) {
   newGame= req.body;
   newGame.price= parseFloat(req.body.price);
   collection.insertOne(newGame, function(err, response) {
      console.log(response.ops);
      res.status(201).json(response.ops);
   }
} ...
```

Run app.boomerangapi.com/workspace on browser

npm start

Found games ...

MongoDB & NodeJS

- We will not be using mongoDB directly from nodeJS.
- There is a much easier way to work with mongoDB from Node.
- We will use Mongoose.

Main Points

- MongoDB is a document-based NoSQL database. It is ideal for mobile application development.
- We can use mongoDB driver to connect to a MongoDB instance from our node code. You will need a connection, make sure you create only once and use it several times. Also make sure it is available when needed (since it is asynchronous).
- The best practice when working with mongoDB from your node code is to create a connection then have all your DB related code in controllers.

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.

REST API

URL Patterns

PATTERN

- Base URL (www.myapplication.com)
- Actions, depending on the method
- Get all/multiple items
  - GET (/api/items)
- Create a new item
  - POST (/api/items)
- Get single item
  - GET (/api/items/123)
- Update a single item
  - PUT (api/items/123)
- Delete a single item
  - DELETE (api/items/123)

NESTED

- Get all reviews for item (123)
  - GET (/api/items/123/reviews)
- Create a review for item (123)
  - POST (/api/items/123/reviews)
- Get single review (222) for items 123
  - GET (/api/items/123/reviews/222)
- Update a single review
  - PUT (api/items/123/reviews/222)
- Delete a single review
  - DELETE (api/items/123/reviews/222)

Mongoose

Why Mongoose

- Create a controller for each document and define everything you need there.
  - Too much work and could end up repeating a lot of the same stuff.
  - Errors and inconsistencies.
- Better to have one schema (define it once) and use it for all my documents.
- Mongoose comes to the rescue.
  - Helps us focus on building our application and building the API.
  - Abstracts complexity of using native driver.
  - Provides helper methods to work with DB.
  - We can define the structure of our data in the application (schema).

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Install Mongoose

```
npm install --save mongoose
```

```
mongoose@5.10.14 node_modules/mongoose
```

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Create file /api/data/db.js

```
var mongoose= require("mongoose");
var dbURL= "monodb://localhost:27017/meanGamesDb";
mongoose.connect(dbURL, { useNewUrlParser: true, useUnifi
edTopology: true });
mongoose.connection.on("connected", function() {
  console.log("Mongoose connected to "+ dbURL);
});
mongoose.connection.on("disconnected", function() {
  console.log("Mongoose disconnected");
});
mongoose.connection.on("error", function(err) {
  console.log("Mongoose connection error "+ err);
});
```

Update app.js to use mongoose

```
require("./api/dta/db.js");
```

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Create file /api/data/db.js

```
process.on("SIGINT", function() {
    mongoose.connection.close(function() {
    console.log("Mongoose  disconnected by app
termination");
    process.exit(0);
});
});
```

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Create file /api/data/db.js

```
process.on("SIGTERM", function() {
    mongoose.connection.close(function() {
    console.log("Mongoose  disconnected by app
termination");
    process.exit(0);
});
});
```

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Create file /api/data/db.js

```
process.once("SIGUSR2", function() {
    mongoose.connection.close(function() {
    console.log("Mongoose  disconnected by app
termination");
    process.kill(process.pid, "SIGUSR2");
});
});
```

Mongoose Schemas & Models

Mongoose

Add Schema

Data Validation

Compile Model

Separate schema from connection, what gets exported is a model (even though it is all schema)

Modify file /api/data/games-model.js

```
var mongoose= require("mongoose");
var gameSchema= mongoose.Schema({
    name : String,
    price : Number,
    designers : [String],
    players : Number,
    rate: Number
});
```

Mongoose

Add Schema

Data Validation

Compile Model

Mandatory fields for a document
Modify file /api/data/games-model.js

```
var mongoose= require("mongoose");
var gameSchema= mongoose.Schema({
    name : {
        type: String,
        required: true
    }
    price : Number,
    designers : [String],
    players : {
        type: Number,
        min : 1,
        max: 10
    },
    rate: {
        type: Number,
        min: 1,
        max: 5,
        "default": 1
    }

});
```

Mongoose

Add Schema

Data Validation

Compile Model

Mandatory fields for a document

Modify file /api/data/games-model.js

```
mongoose.model("Game", gameSchema, "games");
```

Modify db.js to let it know about our model

```
require("./games-model.js");
```

Schema

Nested Docs

Geo-Location

A review is a sub-document. A review is for a game by a user with some rating and description at a certain date.
Modify file /api/data/games-model.js

```
var reviewSchema= new mongoose.Schema({
name: {
    type: String,
    required: true
},
rating: {
    type: Number,
    min: 0,
    max: 5,
    required: true
},
review: {
    type: String,
    required: true
},
createdOn:{
    type: Date,
    "default": Date.Now
}
});

var gameSchema = …
    reviews: [reviewSchema]
});
```

Schema

Nested Docs

Geo-Location

A game is normally published by a publisher. The publisher is from a certain country, established at a certain date, also famous for a certain game
Modify file /api/data/games-model.js

```
var publisherSchema= new mongoose.Schema({
name: {
    type: String,
    required: true
},
country: {
    type: Number,
    required: true
},
established: {
    type: Date,
    required: false
},
location: {
    address: String
}
});

var gameSchema = …
    publisher: publisherSchema
});
```

Schema

Nested Docs

Geo-Location

The publisher is at a certain location, add that location. This can also apply to the physical location of a shop that can sell the game.
Modify file /api/data/games-model.js

```
var publisherSchema= new mongoose.Schema({
name: {
   type: String,
   required: true
},
country: {
   type: Number,
   required: true
},
established: {
   type: Date,
   required: false
},
location: {
   address: String,
   coordinates: [Number] // Store coordinates in order longitude (E/W), latitude (N/S)
}
});

var gameSchema = …
   publisher: publisherSchema
});
```

Schema

Nested Docs

Geo-Location

To search coordinates we need to index, we will use
Modify file /api/data/games-model.js

```
var publisherSchema= new mongoose.Schema({
name: {
    type: String,
    required: true
},
country: {
    type: Number,
    required: true
},
established: {
    type: Date,
    required: false
},
location: {
    address: String,
    // Store coordinates in order longitude (E/W), latitude (N/S)
    coordinates: {
        type: [Number],
        index: "2dsphere"
    }
}
});

var gameSchema = …
    publisher: publisherSchema
});
```

Geo-Locations

- There are two geo-location index systems
  - 2D index of coordinates on flat surface.
  - 2D index of coordinates on a sphere (we consider earth curvature).
- This is needed to find distance between locations
  - Near my locations.
  - Close to certain location.

Mongoose

GetAll

GetOne

Use Mongoose to get all Games, simpler way of doing things.
Modify file /api/data/games-controller.js
remove all required and use mongoose and model

```
var mongoose= require("mongoose");
var Game= mongoose.model("Game");

module.exports.gamesGetAll= function(req, res) {
    var offset= 0;
    var count= 5;
    if (req.query && req.query.offset) {
        offset= parseInt(req.query.offset, 10);
    }
    if (req.query && req.query.count) {
        offset= parseInt(req.query.count, 10);
    }
    Game.find().exec(function(err, games) {
        console.log("Found games", games.length);
        res.json(games);
    });
};
```

Mongoose

GetAll

GetOne

Use Mongoose to get all Games, simpler way of doing things.
Modify file /api/data/games-controller.js
remove all required and use mongoose and model

```
var mongoose= require("mongoose");
var Game= mongoose.model("Game");

module.exports.gamesGetAll= function(req, res) {
    var offset= 0;
    var count= 5;
    if (req.query && req.query.offset) {
        offset= parseInt(req.query.offset, 10);
    }
    if (req.query && req.query.count) {
        offset= parseInt(req.query.count, 10);
    }
    Game.find().skip(offset).limit(count).exec(function(err, games) {
        console.log("Found games", games.length);
        res.json(games);
    });
};
```

Mongoose

GetAll

GetOne

Use Mongoose to get one Game, simpler way of doing things.

Modify file /api/data/games-controller.js

remove all required and use mongoose and model

```
var mongoose= require("mongoose");
var Game= mongoose.model("Game");

module.exports.gamesGetOne= function(req, res) {
    var gameId= req.params.gameId;
    Game.findById(gameId).exec(function(err, game) {
        res.status(200).json(game);
    });
};
```

Mongoose
Sub-documents
Sub-document

Separate Controllers into logical collection.
Modify file /api/routes/index.js

```
var controllerReviews= require("../controllers/reviews.controller");
router.route("/games/:gameId/reviews")
        .get(ctrlReviews.reviewsGetAll);
router.route("/games/:gameId/reviews/:reviewId")
        .get(ctrlReviews.reviewsGetOne);
```

Add file /api/controllers/reviews-controller.js

```
var mongoose= require("mongoose");
var Game= mongoose.model("Game");
module.exports.reviewGetAll= function(req, res) {
   var gameId= req.params.gameId;
   Game.findById(gameId).select("reviews").exec(function(err, doc) {
      res.status(200).json(doc.reviews);
   });
}
module.exports.reviewGetOne= function(req, res) {
}
```

Mongoose

Sub-documents

Sub-document

Add review id if the database does not have it.

```
db.games.update(
   {},
   {
      $set : {
         "reviews.0._id": ObjectId()
      }
   },
   {
      multi: true
   }
)
```

Add file /api/controllers/reviews-controller.js

```
var mongoose= require("mongoose");
var Game= mongoose.model("Game");
module.exports.reviewGetOne= function(req, res){
   var gameId= req.params.gameId;
   var reviewId= req.params.reviewId;
   consloe.log("GET reviewId "+ reviewId+ " for gameId "+ gameId);
   Game.findById(gameId).select("reviews").exec(function(err, game){
      var review= game.reviews.id(reviewId);
      res.status(200).json(review);
   });
}
```

Geo-Location Search

Search Routes

- Do we need a new route to search?
- Did we get a subset of games previously?
  - pagination
- We can use the same route; we need to add some filtering (query strings).

Mongoose

Geo-Search

Sub-document

Add query string to the game controller. Modify games-controller.js

```
var runGeoQuery= function(req, res) {
    var lng= parseFloat(req.query.lng);
    var lat= parseFloat(req.query.lat);
    var point= { //GeoJSON Point
        type: "Point",
        coordinates: [lng, lat]
    };
    Game.aggregate([{
        "$geoNear": {"near": point, "spherical": true, "distanceField": "distance", "maxDistance": 750000,
"num": 5}}], function(err, results){
        console.log("Geo results", results);
        console.log("Geo error ", err);
        res.status(200).json(results);
    });
};
module.exports.gamesGetAll= function(req, res) {
    var offset= 0;
    var count= 5;
    if (req.query && req.query.lat && req.query.lng) {
        runGeoQuery(req, res);
        return;
    }
    if (req.query && req.query.offset) {
        offset= parseInt(req.query.offset, 10);
    }
    ...
};
```

API Design & Hardening

API Design Golden Rules

- Always return a response. Never leave a request hanging.
- Return the correct HTTP status code.
- Return contents or a message.

Error Traps

- Missing query string parameters.
- Correct query string parameter types.

API - GetAll

Types Check

Error Check

Limit Check

Add query string type checking to the game controller. Modify games-controller.js

```
module.exports.gamesGetAll= function(req, res) {
   var offset= 0;
   var count= 5;
   if (req.query && req.query.lat && req.query.lng) {
      runGeoQuery(req, res);
      return;
   }
   if (req.query && req.query.offset) {
      offset= parseInt(req.query.offset, 10);
   }
   if (req.query && req.query.count) {
      count= parseInt(req.query.count, 10);
   }
   if (isNaN(offset) || isNaN(count)) {
      res.status(400).json({"message":  "QueryString Offset and Count should be
numbers"});
   return;
   }
  ...
};
```

API – GetAll

Types Check

Error Check

Limit Check

Add mongoose error handling to the game controller. Modify games-controller.js

```
module.exports.gamesGetAll= function(req, res) {
    ...
    if (isNaN(offset) || isNaN(count)) {
        res.status(400).json({"message": "QueryString Offset and Count
should be numbers"});
    return;
    }
   Game.find().skpi(offset).limit(count).exec(function(err, games) {
        if (err) {
            console.log("Error finding games");
            res.status(500).json(err);
        } else {
            console.log("Found games", games.length);
            res.json(games);
        }
    }
};
```

API - GetAll

Types Check

Error Check

Limit Check

Add query string limit checks to the game controller. Modify games-controller.js

```
module.exports.gamesGetAll= function(req, res) {
    var offset= 0;
    var count- 5;
    var maxCount= 10;
    ...
    if (isNaN(offset) || isNaN(count)) {
        res.status(400).json({"message": "QueryString Offset and Count should be numbers"});
    return;
    }
    if (count > maxCount) {
        res.status(400).json({"message": "Cannot exceed count of "+ maxCount});
        return;
    }
    Game.find().skpi(offset).limit(count).exec(function(err, games) {
        if (err) {
            console.log("Error finding games");
            res.status(500).json(err);
        } else {
            console.log("Found games", games.length);
            res.json(games);
        }
    }
};
```

API - GetOne

Error Check

Result Check

Add error checking to single Game finder in controller.
Modify games-controller.js

```
module.exports.gamesGetOne= function(req, res) {
    var gameId= req.params.gameId;
    Game.findById(gameId).exec(function(err, game) {
        if (err) {
            console.log("Error finding game");
            res.status(500).json(err);
        } else {
            res.status(200).json(game);
        }
    });
};
```

API - GetOne

Error Check

Result Check

Add result checking to single Game finder in controller.
Modify games-controller.js

```
module.exports.gamesGetOne= function(req, res) {
  var gameId= req.params.gameId;
  Game.findById(gameId).exec(function(err, game) {
    if (err) {
      console.log("Error finding game");
      res.status(500).json(err);
    } else if(!game) {
      res.status(404).json({"message" : "Game ID not
found"});
    } else {
      res.status(200).json(game);
    }
  });
};
```

API - GetOne

Error Check

Result Check

Refactor controller for easier readability and maintainability. Modify games-controller.js

```
module.exports.gamesGetOne= function(req, res) {
  var gameId= req.params.gameId;
  Game.findById(gameId).exec(function(err, game) {
    var response= {
      status: 200,
      message: game};
    if (err) {
      console.log("Error finding game");
      response.status= 500;
      response.message= err;
    } else if(!game) {
      response.status= 404;
      response.message= {"message" : "Game ID not found"};
    }
    res.status(response.status).json(response.message);
  });
};
```

API - GetOne

Error Check

Result Check

Type Check?

Type check for ID is done by mongoose.

Try it out

On your browser enter:

localhost:3000/api/games/SomeTextNotID

Create Documents

Create Game Publisher

To create a document in DB we need a route for the API, then a controller. Modify api/routes/index.js

```
router.route("/games/")
        .get(controllerGames.gameGetAll)
        .post(controllerGames.gameAddOne);
```

Modify the api/controller/gameController.js

```
module.exports.gamesAddOne= function(req, res) {
   Games.create({title: req.body.title, year: parseInt(req.body.year),
price: parseFloat(req.body.price), designer: req.body.designer,
publisher: [name: "empty", location: []], minPlayers:  parseInt(req.body.minPlayers),
maxPlayers:  parseInt(req.body.maxPlayers),  rate: parseFloat(req.body.rate)},
function(err, game) {
       if (err) {
          console.log("Error creating games");
          res.status(400).json(err);
       } else {
          console.log("Game created", game);
          res.status(201).json(game);
       }
   });
};
```

Create Game Publisher

To create a sub-document in DB we need a route for the API, then a controller. Modify api/routes/index.js

```
router.route("/games/:gameId/publisher")
        .get(controllerPublisher.publisherGet)
        .post(controllerPublisher.publisherAdd);
```

Modify the api/controller/publisherController.js

```
module.exports.publisherGet= function(req, res) {
  var gameId= req.params.gameId;
  console.log("Get gameId ", gameId);
  Game.findById(gameId).select(publisher).exec(function(err, game) {
    var response= {status: 200, message: []};
    if (err) {
      console.log("Error finding game");
      response.status= 500; response.message= err;
    } else if (!game) {
      console.log(Game id not found in database", id);
      response.status= 404; response.message= {"message": "Game ID not
found"+gameId};
    } else {
      response.message= game.publisher? game.publisher : [];
    }
    res.status(response.status).json(response.message);
};
```

Create
Game
Publisher

Modify the api/controller/publisherController.js

```
var _addPublisher= function(req, res, game) {
    game.publisher.name= req.body.name;
    game.publisher.location.coordinates= [parseFloat(req.body.lng),  parseFloat(req.body.lat)];
    game.save(function(err,  updatedGame){
        var  response= {status: 200, message: []};
        if (err) {
            reponse.status= 500;
            response.message= err;
        } else  {
            reponse.status= 201;
            response.message= updatedGame.publisher;
        }
            res.status(response.status).json(response.message);
    });
}

module.exports.publisherAdd= function(req, res) {
    var gameId= req.params.gameId;
    console.log("Get gameId ", gameId);
    Game.findById(gameId).select(publisher).exec(function(err, game) {
        var  response= {status: 200, message: []};
        if (err) {
            console.log("Error  finding game");
            response.status= 500; response.message= err;
        } else if (!game) {
            console.log(Game  id not found in database", id);
            response.status= 404; response.message= {"message":  "Game ID not found"+gameId};
        }
        if (game) {
            _addPublisher(req, res, game);
        } else {
            res.status(response.status).json(response.message);
        }
};
```

Update Documents

Update Game Publisher

To update an existing game we need to create a route and a controller. Update the routes in api/routes/index.js

```
router.route("/games/:gameId")
.get(controllerGames.gamesGetOne)
.put(controllerGames.gamesUpdateOne);
```

Update api/controllers/games-controller.js

```
module.exports.gamesUpdateOne= function(req, res) {
  var gameId= req.params.gameId;
  Game.findById(gameId).select("-reviews -publisher").exec(function(err, game) {
    var response= {status: 204};
    if (err) {
      console.log("Error finding game");
      response.status= 500;
      response.message= err;
    } else if(!game) {
      response.status= 404;
      response.message= {"message" : "Game ID not found"};
    }
    if (response.status !== 204){
      res.status(response.status).json(response.message);
    } else {
      game.title= req.body.title; game.year= parseInt(req.body.year);
game.price= parseFloat(req.body.price); game.designer= req.body.designer;
game.minPlayers= parseInt(req.body.minPlayers); game.maxPlayers= parseInt(req.body.maxPlayers);
game.rate= parseFloat(req.body.rate); game.minAge= parseInt(req.body.minAge);
      game.save(function(err, updatedGame) {
        if(err) {
          response.status= 500;
          response.message= err;
        }
        res.status(response.status).json(response.message);
      });
    }
  });
};
```

Update Game Publisher

To update an existing game we need to create a route and a controller. Update the routes in api/routes/index.js

```
router.route("/games/:gameId/publisher")
.get(controllerPublisher.publisherGet)
.post(controllerPublisher.publisherAdd)
.put(controllerPublisher.publisherUpdate);
```

Update api/controllers/publisher-controller.js

```
var _updatePublisher= function(req, res, game) {
  game.publisher.name= req.body.name;
  game.publisher.location.coordinates= [parseFloat(req.body.lng), parseFloat(req.body.lat)];
  game.save(function(err, updateGame) {
    var response= {status: 204};
    if (err) {
      console.log("Error finding game");      response.status= 500;      response.message= err;
    }
    res.status(response.status).json(response.message);
  });
}
module.exports.publisherUpdate= function(req, res) {
  var gameId= req.params.gameId;
  console.log("PUT gameId ", gameId);
  Game.findById(gameId).select("-reviews").exec(function(err, game) {
    var response= {status: 204};
    if (err) {
      console.log("Error finding game");      response.status= 500;      response.message= err;
    } else if(!game) {
      response.status= 404;      response.message= {"message" : "Game ID not found"};
    }
    if (response.status !== 204){
      res.status(response.status).json(response.message);
    } else {
      _updatePublisher(req, res, game);
    }
  });
};
```

Delete Documents

Delete Game Publisher

To delete an existing game we need to create a route and a controller. Update the routes in api/routes/index.js

```
router.route("/games/:gameId")
.get(controllerGames.gamesGetOne)
.put(controllerGames.gamesUpdateOne)
.delete(controllerGames.gameDeleteOne);
```

Update api/controllers/games-controller.js

```
module.exports.gamesDeleteOne= function(req, res) {
    var gameId= req.params.gameId;
    console.log("DELETE gameId ", gameId);
    Game.findByIdAndRemove(gameId).exec(function(err, deletedGame) {
        var response= {status:  204};
        if (err) {
            console.log("Error finding game");
            response.status= 500;
            response.message= err;
        } else if(!deletedGame) {
            response.status= 404;
            response.message= {"message" : "Game ID not found"};
        }
        res.status(response.status).json(response.message);
    });
};
```

Delete Game Publisher

To delete an existing publisher from a game, we need to create a route and a controller. Update the routes in api/routes/index.js

```
router.route("/games/:gameId/publisher")
.get(controllerPublisher.publisherGet)
.post(controllerPublisher.publisherAdd)
.put(controllerPublisher.publisherUpdate)
.delete(controllerPublisher.publisherDelete);
```

Update api/controllers/publisher-controller.js

```
var _deletePublisher= function(req, res, game) {
        game.publisher.remove();
        game.save(function(err, game) {
          var response= {status: 204};
          if (err) {
            console.log("Error finding game");          response.status= 500;          response.message= err;
          }
          res.status(response.status).json(response.message);
        });
    }
    module.exports.publisherDelete= function(req, res) {
      var gameId= req.params.gameId;
      console.log("PUT gameId ", gameId);
      Game.findById(gameId).select("-reviews").exec(function(err, game) {
        var response= {status: 204};
        if (err) {
          console.log("Error finding game");          response.status= 500;          response.message= err;
        } else if(!game) {
          response.status= 404;          response.message= {"message" : "Game ID not found"};
        }
        if (response.status !== 204){
          res.status(response.status).json(response.message);
        } else {
          _deletePublisher(req, res, game);
        }
      });
    };
```

Delete Game Publisher

We may have to update the create method to enable it to function properly after deletes. Update the controller in api/routes/publisher-controller.js

```
Module.exports.publisherAdd= function(req, res) {
…
    if(game) {
        if(!(game.publisher)) {
            game.publisher= {name: "empty", location: []};
        }
        _addPublisher(req, res, game);
…
```

Main Points

- Using Mongoose is better than using MongoDB driver directly. Mongoose enables us to focus on building our application by abstracting complexity of using the native driver. Mongoose provides helper methods to speed up development.
- We define the structure of our data using Schemas. Schemas not only define the types of fields in the document but also provide constraints and default values.
- Mongoose makes CRUD operations simpler and easier. Mongoose also enforces non-blocking operations.

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.

Introduction to AngularJS

AngularJS

jsbin.com

Forgiving expr

Concatenation

Data Binding

Scope

Use AngularJS for the first time. Go to www.jsbin.com
Add library "Angular 1.4.0 Stable"

Add directive

```
<html ng-app>


<body>
1 + 2 = {{ 1+2 }}
</body>
```

AngularJS

jsbin.com

Forgiving expr

Concatenation

Data Binding

Scope

Error will be silently failling.

```
<body>
{{ console.log(something.doesNotExist);  }}
</body>
```

AngularJS

jsbin.com

Forgiving expr

Concatenation

Data Binding

Scope

JS operations can be performed, like string concatenation

```
<body>
{{ "Hello "+ "world!" }}
</body>
```

AngularJS

jsbin.com

Forgiving expr

Concatenation

Data Binding

Scope

Data Binding, use ng-model directive and assign a variable. Two-way data binding, any update to the model updates the view and any update to the view updates the model.

```
<body>
<p> Hello {{ user }}! </p>
<input type="text" name="" id="" ng-model="user" />
</body>
```

AngularJS

jsbin.com

Forgiving expr

Concatentation

Data Binding

Scope

Scope of ng-app depends on where it is defined. The scope of your angular application depends on the scope of ng-app.

```
<html>
...
<body>
<p> Hello {{ user }}! </p>
<div ng-app>
</div>
<input type="text" name="" id="" ng-model="user" />
</body>
</html>
```

Built-in Directives

Directives

- ng-app
- ng-model
- ng-init
- ng-click
- ng-if
- ng-hide
- ng-class
- ng-repeat
- ng-options
- ng-cloak

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Directive to initialize variables (string, numbers, bool, array or object) do not assign values to variables using ng-init

```
<div ng-init="name= 'Jack'">
   {{ name }}
</div>
```

{{ }} is a shortcut for the directive ng-bind

```
<div ng-init="name= 'Jack'">
   <p ng-bind="name"></p>
</div>
```

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Directive to execute a method or an expression

```
<div ng-init="number= 0">
   <button ng-click="number= number + 1">+1</button>
   <button ng-click="number= number - 1">-1</button>
   <p> {{ number }} </p>
</div>
```

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Directive to execute a method or an expression

```
Check the checkbox to see the paragraph
<input type="checkbox" ng-model="showParagraph">
<p ng-if="showParagraph"> The paragraph. </p>
```

We can replace with ng-show and ng-hide

```
<p ng-show="showParagraph"> The paragraph. </p>
```

ng-if removes the element from the DOM tree, ng-show applies CSS display none.

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

```
<div ng-init="number = 19">
   <input type=" text" ng-model="guess">
   <p ng-hide= "guess != number">Correct</p>
   <p ng-show= "guess != number">Incorrect</p>
</div>
```

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Modify the CSS class dynamically based on conditions

```
<style>
   .red {border-color: red;}
   .green {border-color: green;}
</style>
</head>
<body>
<div ng-init="number = 19">
   <input type=" text" ng-model="guess" ng-class="{red:
guess != number, green: guess == number}">
</div>
```

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Modify the CSS class dynamically based on conditions

```
<style>
   .red {color: red;}
   .green {color: green;}
</style>
</head>
<body>
<div ng-init="numbers = [0,1,2,3,4,5,6,7,8]">
   <ul>
     <li ng-repeat="number in numbers" ng-class="{red:
$even, green:$odd}">{{ number }}</li>
   </ul>
</div>
```

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Repeat by values or index

```
<style>
    .red {color: red;}
    .green {color: green;}
</style>
</head>
<body>
<div ng-init="names  = ['Jack', 'John', 'Jack']">
    <ul>
      <li ng-repeat="name in names" ng-class="{red: $even,
green:$odd}">{{ name }}</li>
    </ul>
</div>
```

Repeat using index

```
      <li ng-repeat="name in names track by $index" ng-class="{red:
$even, green:$odd}">{{ name }}</li>
```

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Repeat over objects

```
<style>
   .red {color: red;}
   .green {color: green;}
</style>
</head>
<body>
<div ng-init="names = [{fistName: 'Jack', lastName: 'Smith'},
{fistName: 'John', lastName: 'Simson'}]">
   <ul>
     <li ng-repeat="name in names" ng-class="{red: $even,
green:$odd}">{{ name.lastName }}, {{ name.firstName}}</li>
   </ul>
</div>
```

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Do not populate options using ng-repeat, use ng-options

```
</head>
<body>
 <div ng-init="students = [{name: 'Jack', course: 'MPP', gpa: 3.0}, {name: 'John', course: 'MWA', gpa: 2.5}, {name:'Jill', course: 'SWE', gpa: 3.3}, {name: 'Jim', course: 'MWA', gpa: 2.8}]">
   <select name="" id="" ng-model="student" ng-options="student.name for student in students"></select>
    <p>You have selected: {{student:name}} ({{student.gpa}})
</p>
 </div>
```

Grouping

```
<select name="" id="" ng-model="student" ng-options= "student.name group by student.course for student in students"></select>
```

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Performance and user expereince

```
<head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.0
/angular.min.js"></script>
…
</head>
<body>
…
```

This will result in a delay due to downloading of resource.

displaying some {{…}} while the script is being downloaded.

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Better performance not so good user expereince

```
...
<body>
...
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.0/angular.min.js"></script>
</body>
```

This will result displaying some {{...}} while the application is loading, then they will be populated.

Directives

ng-init
ng-click
ng-if
ng-show ng-hide
ng-class
ng-repeat
ng-options
ng-cloak

Better performance and user experience use ng-cloak

```
...
 <style>
  .ng-cloak, [ng-cloak], [ng\:cloak] {
    display: none !important;
  }
 </style>
</head>
<body>
 <div ng-cloak
...
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/
1.4.0/angular.min.js"></script>
</body>
```

Until AngularJS has not finished the bootstrapping process it will not display anything, after it is done it will display.

Built-in Filters

Some Built-in Filters

- Currency
- Number
- String
- Date
- Limit
- Order
- Filter

Filters

Currency
Number
Strings
Date
Limit
OrderBy
Filter

Display money values using currency filter.

```
<div ng-init="total= 123.45">
  <p>{{ total | currency}}</p>
<div>
```

Different currency?

```
  <p>{{ total | currency:"£"}}</p>
  <p>{{ total | currency:"¥"}}</p>
  <p>{{ total | currency:"€"}}</p>
```

Filters

Currency

Number

Strings

Date

Limit

OrderBy

Filter

Display a format for decimal digits to display.

```
<div ng-init="interest= 123.456789">
  <p>{{ interest | number: 4}}</p>
<div>
```

Negative numbers?

```
  <p>{{ -interest | number:4}}</p>
```

Filters

Currency

Number

Strings

Date

Limit

OrderBy

Filter

Display a format for decimal digits to display.

```
<div ng-init="title= 'Maharishi International University'">
  <p>{{ title | uppercase}}</p>
<div>
```

Lower case?

```
  <p>{{ title | lowercase}}</p>
```

Filters

Currency

Number

Strings

Date

Limit

OrderBy

Filter

Display a format for decimal digits to display.

```
<div ng-init="firstDayOfCourse= 1610380800000">
  <p>{{ firstDayOfCourse | date}}</p>
<div>
```

Date and Time?

```
  <p>{{ firstDayOfCourse | date: "short"}}</p>
  <p>{{ firstDayOfCourse | date: "medium"}}</p>
```

Fine ontrol?

```
  <p>{{ firstDayOfCourse | date: "MMM-dd-yyyy
(hh:mm:ss:(sss)"}}</p>
  <p>{{ firstDayOfCourse | date: "MM-dd-
yy (hh:mm:ss:(sss)"}}</p>
```

Filters

Currency

Number

Strings

Date

Limit

OrderBy

Filter

Limit items returned from Array or Object.

```
<div ng-init="numbers= [0,1,2,3,4,5]">
  <p>{{ numbers | limitTo: 4}}</p>
<div>
```

Get last numbers instead of first?

```
  <p>{{ numbers | limitTo: -2}}</p>
```

Filters

Currency

Number

Strings

Date

Limit

OrderBy

Filter

Limit items returned from Array or Object.

```
<div ng-init="students = [{name: 'Jack', course: 'MPP', gpa: 3.0}, {name: 'John', course: 'MWA', gpa: 2.5}, {name:'Jill', course: 'SWE', gpa: 3.3}, {name: 'Jim', course: 'MWA', gpa: 2.8}]">
  <ul>
    <li ng-repeat="student in students | orderBy: 'gpa'">{{student.name}} with gpa {{student.gpa}} taking {{student.course}}.</li>
  </ul>
<div>
```

Reverse order?

```
    <li ng-repeat="student in students | orderBy: '-gpa'">
```

Nested ordering?

```
    <li ng-repeat="student in students | orderBy: ['course', '-gpa']">
```

Filters

Currency

Number

Strings

Date

Limit

OrderBy

Filter

Search through the data set.

```
<div ng-init="students  = [{name: 'Jack', course: 'MPP', gpa: 3.0}, {name: 'John', course: 'MWA', gpa: 2.5}, {name:'Jill', course: 'SWE', gpa: 3.3}, {name: 'Jim', course: 'MWA', gpa: 2.8}]">
  <ul>
    <li ng-repeat="student in students | filter: 'jack'">{{student.name}} with gpa {{student.gpa}} taking {{student.course}}.</li>
  </ul>
<div>
```

Implement a dynamic search instead of a static one?

```
    <input type="text" ng-model="searchText">
    <li ng-repeat="student in students | filter: searchText>
```

Only search courses?

```
    <input type="text" ng-model="searchText.course">
```

Search all fields?

```
    <input type="text" ng-model="searchText.$">
```

Controllers

Controllers

Controller

As Syntax

My First Controller, create an Angular module (myFirstApp)

```
<html ng-app="myFirstApp">
...
<script>
  angular.module("myFirstApp",
[]).controller("MyFirstController", MyFistController);
  function MyFirstController($scope) {
    $scope.name=  "Jack";
  }
</script>
...
<body>
  <div ng-controller="MyFirstController">
    Hello, {{name}}!
  </div>
</body>
```

Controllers

Controller

As Syntax

Use Arrays in controller.

```
<html ng-app="myFirstApp">
…
<script>
 angular.module("myFirstApp", []).controller("MyFirstController",
MyFistController);
 function MyFirstController($scope) {
   $scope.students= [{name: 'Jack', course: 'MPP', gpa: 3.0}, {name:
'John', course: 'MWA', gpa: 2.5}, {name:'Jill', course: 'SWE', gpa: 3.3},
{name: 'Jim', course: 'MWA', gpa: 2.8}];
 }
</script>
…
<body>
 <div ng-controller="MyFirstController">
   <ul>
     <li ng-repeat="student in students">{{student.name}} with gpa
{{student.gpa}} taking {{student.course}}.</li>
   </ul>
 </div>
</body>
```

Controllers

Controller As Syntax

Use functions in $ scope.

```
<html ng-app="myFirstApp">
…
<script>
 angular.module("myFirstApp", []).controller("MyFirstController", MyFistController);
 function MyFirstController($scope) {
  $scope.number= 0;
  $scope.increment= function(value) {
   $scope.number= $scope.number+value;
  }
  $scope.decrement= function(value) {
   $scope.number= $scope.number-value;
  }
 }
</script>
…
<body>
 <div ng-controller="MyFirstController">
  {{number}}
 <p>
  <button ng-click="increment(5)">+5</button>
  <button ng-click="decrement(5)">-5</button>
 </p>
 </div>
</body>
```

Controllers

Controller As Syntax

More than one controller, using controller as syntax.

```
<html ng-app="myFirstApp">
…
<script>
  angular.module("myFirstApp", []).controller("MyFirstController", MyFistController)
.controller("MySecondController", MySecondController);
  function MyFirstController(){
    var vm= this;
    this.name= "Jack";
  }
  function MySecondController($scope){
    var vm= this;
    this.name= "John";
  }
</script>
…
<body>
  <div ng-controller="MyFirstController as MyJackCtrl">
    <div ng-controller="MySecondController as MyJohnCtrl">
       Hello {{MyJackCtrl.name}}!
       Hello {{MyJohnCtrl.name}}!
    </div>
  </div>
</body>
```

Modules

Modules

HTML Page

Module

Controller

Create file index.html

```
<!DOCTYPE html>
<html ng-app="myProperApp">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width">
<title></title>
</head>
<body>
  <div ng-controller="MyProperController as MyCtrl">
    <p>Hello, {{MyCtrl.name}}!</p>
  </div>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.0/angular.
min.js"></script>
<script src="app.js"></script>
<script src="controller.js"></script>
</body>
</html>
```

Modules
HTML Page
Module
Controller

Create file app.js

```
angular.module("myProperApp", []);
```

Modules
HTML Page
Module
Controller

Create file controller.js

```
angular.module("myProperApp").controller("MyProperCo
ntroller", MyProperController);
function MyProperController() {
  var vm= this;
  vm.name= "Jack";
}
```

Single Page Application (SPA)

SPA

template

templateURL

controller

Bad URL

Modify file index.html

```
...
<body>
  <div ng-view>
  </div>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.0/angular.min.js"
></script>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.0/angular-
route.min.js"></script>
...
</body>
```

Add dependency, modify file app.js

```
angular.module("myTemplateApp", ['ngRoute"]).config(config);
function config($routeProvider){
  $routeProvider.when("/", {
    template: "<h1>This is the home page.</h1>"
  }).when("/about", {
    template: "<h1>This is the about page.</h1>"
  });
}
```

SPA

template

templateURL

controller

Bad URL

Add dependency, modify file app.js

```
angular.module("myTemplateApp",
['ngRoute"]).config(config);
function config($routeProvider) {
  $routeProvider.when("/", {
    templateUrl: "template/main.html"
  }).when("/about", {
    templateUrl: "template/about.html"
  });
}
```

Create template/main.html

```
<h1>This is the home page.</h1>
```

Create template/about.html

```
<h1>This is the about page.</h1>
```

Setup Server to Serve

- Due to security the previous page redirection will not work.
- How to setup a webserver:
- Using Python
  - python -m SimpleHTTPServer 8181
  - python –m http.server 8181
  - python3 –m http.server 8181

SPA

template

templateURL

controller

Bad URL

Add controller, modify file app.js

```
angular.module.("myControllerApp", ['ngRoute']).config(config);
function config($routeProvider) {
  $routeProvider.when("/", {
    templateUrl: "template/main.html",
    controller: "MainController",
    controllerAs: "mainCtrl"
  }).when("/about", {
    templateUrl: "template/about.html",
    controller: "AboutController",
    controllerAs: "aboutCtrl"
  });
}
```

Create controller file mainController.js

```
angular.module("myControllerApp").controller("MainController",  MainController);
function MainController() {
  var vm= this;
  vm.name= "Jack";
}
```

Create controller file aboutController.js

```
angular.module("myControllerApp").controller("AboutController",  AboutController);
function MainController() {
  var vm= this;
  vm.about= "This is my bio";
}
```

SPA

template

templateURL

controller

Bad URL

Add controllers to your HTML file, modify index.html

```
…
<script src="mainController.js></script>
<script src="aboutController.js></script>
…
```

Update main.html, no need to use controller directive

```
<H1> Hello, {{ mainCtrl.name }} </H1>
```

Update about.html

```
<p> {{ aboutCtrl.bio }} </p>
```

SPA

template

templateURL

controller

Bad URL

If a request for a non-exsisting page is made we can handle that using otherwise. Modify app.js

```
…
}).when(…
}).otherwise({
  redirectTo: "/"
});
…
```

We can also handle error with a "Page not Found" 404
In this case we need to add a controller and possibly a template for that.

Services

Services

http

routeParams

Factory

Add service to MainController, modify file mainController.js

```
function MainController($http) {
  var vm= this;
  $http.get("https://official-joke-
api.appspot.com/jokes/ten")  .then(function(response) {
    vm.jokes= response.data;
  });
}
```

Modify template/main.html

```
<ul>
  <li ng-repeat="joke in
mainCtrl.jokes">{{joke.type}}<BR/>
  {{joke.setup}} : {{joke:punchline}}</li>
</ul>
```

Services

http

routeParams

Factory

Read route parameters, modify file app.js

```
…
}).when("joke/:jokeType",{
templateUrl: "template/joke.html",
controller: "JokeController",
controllerAs: "jokeCtrl"});
}
```

Add jokeController.js

```
angular.module("myControllerApp").controller("JokeController",JokeController);
function JokeController($http,$routeParams){
  var vm= this;
  var jokeType= $routeParams.jokeType;
  $http.get("https://official-joke-
api.appspot.com/jokes/"+jokeType+"/random").then(function(response){
    vm.joke= response.data;
  });
}
```

Add template/joke.html

```
{{jokeCntrl.joke.type}}<BR/>
{{jokeCntrl.joke.setup}}: {jokeCntrl.{joke:punchline}}
```

Update index.html

```
<script scr="jokeController.js"></script>
```

Architecture

- Routes
  - app.js
- Create a folder for each part of the application (main, joke, about, …) folder contents
  - Templates file (main.html, joke.html, …)
  - Controller file (main-controller.js, joke-controller.js, …)

Service

http

routeParams

Factory

Create dataFactory/dataFactory.js

```
angular.module("myJokeApp").factory("JokeFactory",   JokeFactory);
function JokeFactory($http) {
  return {
    getTenJokes: getTenJokes,
    getOneJoke: getOneJoke
  };
  function getTenJokes(){
    return $http.get("https://official-joke-api.appspot.com/jokes/ten").then(complete).catch(failed);
  }
  function getOneJoke(jokeType){
    return $http.get("https://official-joke-
api.appspot.com/jokes/"+jokeType+"/random").then(complete).catch(failed);
  }
  function complete(response){
    return response.data;
  }
  function failed(error) {
    return error.statusText;
  }

}
```

Update index.html to read the factory script

```
<script src="dataFactory/dataFactory.js"></script>
```

Service
http
routeParams
Factory

Update controllers to use the Factor, update main-controll.js

```
function MainController(JokeFactory) {
  var vm= this;
  JokeFactory.getTenJokes().then(function(response) {
    vm.jokes= response;
  });
}
```

Update joke-controller.js

```
function JokeController($routeParams, JokeFactory) {
  var vm= this;
  var jokeType= $routeParams.jokeType;
  JokeFactory.getOneJoke(jokeType).then(function(response)
{
    vm.joke= response[0];
  });
}
```

Custom Filters

Filter

Number Filter

String Filter

Add the filter filters/numberPostfix.js

```
angular.module("myJokeApp").filter("order", numberOrder);
function numberOrder() {
  return function(number) {
    if(number && !isNaN(number)) {
      var digit= number%10;
      var suffix= "";
      switch(digit) {
        case 1:
          suffix= "st";
          break;
        case 2:
          suffix= "nd";
          break;
        case 3:
          suffix= "rd";
          break;
        default:
          suffix= "th";
          break;
        }
      return number+suffix;
    }
    return number;
  }
}
```

Update index.html

```
<script src="filters/numberPostfix.js></script>
```

Update main/main.html

```
{{joke.id | order}}
```

Filter

Number Filter

String Filter

Add the filter filters/vowelsRemover.js

```
angular.module("myJokeApp").filter("vowels", vowelRemover);
function vowelRemover() {
  return function(string, vowel) {
    if(string && (vowel=="a" || vowel=="e" || vowel=="i" || vowel=="o" || vowel=="u") {
      var newString= "";
      for(let char of string) {
        c= char.toLocaleLowerCase();
        if (c == vowel) {
          continue;
        }
        newString+= char;
      }
      return newString;
    }
    return string;
  }
}
```

Update index.html

```
<script src="filters/vowelsRemover.js></script>
```

Update main/main.html

```
{{joke.type | vowels:"e"}}
```

Main Points

- AngularJS is the UI part of a MEAN application. It enables building flexible Single Page Applications (SPA).
- AngularJS enforces an MVC architecture. AngularJS enforces proper software engineering practices, separation of concern.
- AngularJS has a set of built-in directive to speed up the development of web applications. At the same time, you may write your own custom directives and filters.

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.

Integrating MEAN

Setup

- Check endpoints working properly using REST browser plugin.
- Create angular-app folder in the application public folder.
- Add public/angular-app/app.js file (empty for now). This is angular app.
- Install AngularJS using npm (or any other way)
  - npm i angular angular-route
- Add the angular files as dependencies to project
  - <script src="node_modules/angular/angular.js"></script>
    <script src="node_modules/angular-route/angular-route.js"></script>
- Include the angular application
  - <script src="angular-app/app.js"></script>
- Enable our node application to reach Angular (add app.use)
  - App.use("/node_modules", express.static(path.join(__dirname, "node_modules")));

MEAN

Title

Get List

Get One

Rating

Get the home page from Angular

Update index.html

```
…
<html ng-app="meanGames">
…
<body>
<div ng-view></div>
…
<script src="angular-app/game-list/game-list-
controller.js"></script>
</body>
}
```

MEAN

Title

Get List

Get One

Rating

Update angular-app/app.js

```
angular.module("meanGames", ["ngRoute"]).config(config);
function config($routeProvider) {
   $routeProvider.when("/", {
      templateURL: "angular-app/game-list/games.html",
      controller:" GamesController",
      controllerAs: "vm"
   });
}
```

Add the controller angular-app/game-list/game-list-controller.js

```
angular.module("meanGames", ["ngRoute"])
.controller("GamesController", GamesController);
function GamesController() {
  var vm= this;
  vm.title= "Mean Games App";
}
```

Add the template angular-app/game-list/gmaes.html

```
<H1>{{vm.title}}</H1>
```

MEAN

Title

Get List

Get One

Rating

Get the list of games from API

Update controller to make the request, public/angular-app/game-list/game-list-controller.js

```
function GamesController($http) {
  var vm= this;
  vm.title= "Mean Games App";
  $http.get("/api/games").then(function(response) {
    vm.games= response.data;
  }
}
```

Update the template angular-app/game-list/games.html

```
<H1>{{vm.title}}</H1>
<ul>
<li ng-repeat="game in vm.games">{{game.title}}</li>
</ul>
```

MEAN

Title

Get List

GetOne

Rating

Date routing to display a game

Update public/angular-app/app.js

```
…
function config($routeProvider, $locationProvier) {
$locationProvier.hashPrefix("");
…
.when("/game/:id", {
templateUrl: "angular-app/game-display/game.html",
controller: "GameController",
controllerAs: "vm"
});
}
```

Add controller to html page public/index.html

```
…
<script src="angular-app/game-data-factory/game-data-factory.js"></script>
<script src="angular-app/game-display/game-display-controller.js"></script>
```

MEAN

Title

Get List

GetOne

Rating

Create the data factory that calls the endpoints, and it used in our app.

Create public/game-data-factory/game-data-factory.js

```
angular.module("meanGames").factory("GameDataFactory", GameDataFactory);
function GameDataFactory($http) {
  return {
    getAllGames: getAllGames,
    getOneGame: getOneGame
  };
  function getAllGames() {
    return $http.get("/api/games").then(complete).catch(failed);
  }
  function getOneGame(id) {
    return $http.get("/api/games/"+id).then(complete).catch(failed);
  }
  function complete(response){
    console.log(response.data);
    return response.data;
  }
  function failed(error) {
    return error.status.statusText;
  }
}
```

Update game-list-controller.js to use the factory

```
function GamesController(GameDataFactory) {
 var vm= this;
 vm.title= "Mean Games App";
 GameDataFactory.getAllGames().then(function(response) {
  vm.games= response;
 });
}
```

MEAN

Title

Get List

GetOne

Rating

Get data about one game, add controller and template

Add controller public/angular-app/game-display/game-display-controller.js

```
angular.module("meanGames").controller("GameController", GameController);
function GameController(GameDataFactory, $routeParams) {
    var vm= this;
    var id= $routeParams.id;
    GameDataFactory.getOneGame(id).then(function(response) {
        vm.game= response;
    });
}
```

Add the template angular-app/game-display/game.html

```
<H1>Information about game: <p>{{vm.game.title}}</p></H1>
<p>
    Price: {{vm.game.price | currency}}<BR/>
    Minimum Players: {{vm.game.minPlayers}}<BR/>
    Maximum Players: {{vm.game.maxPlayers}}<BR/>
    Minimum Age: {{vm.game.minAge}}</BR>
    Publisher: {{vm.game.publisher.name}}
</p>
```

MEAN

Title

Get List

GetOne

Rating

Selecting a game from the list

Update public/angular-app/game-list/games.html

```
…
<li ng-repeat="game in vm.games"><a ng-
href="#/game/{{game._id}}">{{game.title}}</a></li>
}
```

Display Ratings

- What is the best way to display ratings?
- Number :(
- Images :/
- Stars :)
- Custom directive

Custom Directives

MEAN

Title

Get List

GetOne

Rating

Update template public/game-display/game-display-controller.js

…

vm.rating= response.rate;

…

Update template public/game-display/game.html

<H1>Information about game: {{vm.game.title}} - {{vm.rating}} </H1>

…

We would prefer to see stars according to this number

MEAN

Title

Get List

GetOne

Rating

Update template public/game-display/game.html

```
<H1>Information about game: <p>{{vm.game.title}} <game-rating
stars={{vm.rating}}></game-rating> </p></H1>
```

Add to html file index.html

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
...
<script src="angular-app/game-rating/game-rating-directive.js></script>
```

Update controller to send an array instead of a number game-display-controller.js

```
…
    vm.rating= _getStarRating(response.rate);
  });
}
function _getStarRating(stars) {
    return new Array(rate);
}
...
```

MEAN

Title

Get List

GetOne

Rating

Create directive public/angular-app/game-rating/game-rating-directive.js

```
angular.module("meanGames").directive("gameRating".
GameRating);
function GameRating() {
  return {
    restrict: "E",
    templateUrl: "angular-app/game-rating/rating.html",
    bindToController: true,
    controller: "GameController",
    controllerAs: "vm",
    scope: {
      starts: "@"
    }
  }
}
```

Create template public/angular-app/game-rating/rating.html

```
<span ng-repeat="star  in vm.rating track by $index"
class="glyphicon glyphicon-star"></span>
```

MEAN

Title

Get List

GetOne

Rating

Use component instead public/angular-app/game-rating/game-rating-directive.js

```
angular.module("meanGames").component("gameRating"
, {
    bindings: {
      stars: "*"
    },
    templateUrl: "angular-app/game-rating/rating.html",
    controller: "GameController",
    controllerAs: "vm",
});
```

Form Validation

Forms

Field Checking

Pattern Check

Check on submit

Add Game

We will use JSBin for this part

WE can use HTML 5 form validation attributes (required, email number, url) alos AngularJS form validation ng-minlength, ng-maxlength, ng-pattern

```
<form name="myForm">
<input type="text" name="name" required ng-minlength="3"
ng-maxlength="10" ng-model="name"></input>
</form>
<p>{{myForm.$pristine}}</p>
<p>{{myForm.$dirty}}</p>
<p>{{myForm.name.$pristine}}</p>
<p>{{myForm.name.$dirty}}</p>
<p>{{myForm.name.$valid}}</p>
<p>{{myForm.name.$invalid}}</p>
```

Forms

Field Checking

Pattern Check

Check on submit

Add Game

```
<form name="myForm">
<input type="text" name="name" required ng-minlength="3" ng-maxlength="10" ng-model="name"></input>
<span ng-show="myForm.name.$dirty && myForm.name.$invalid">
This feild requires 3-10 characters.
</span>
</form>
```

Forms

Field Checking

Pattern Check

Check on submit

Add Game

```
<form name="myForm">
<input type="text" name="name" required ng-pattern="/^[0-9]{2,3}$/" ng-model="name"></input>
<span ng-show="myForm.name.$dirty && myForm.name.$invalid">
This feild requires 2 or 3 digits.
</span>
</form>
```

Forms

Field Checking

Pattern Check

Check on submit

Add Game

```
<script>
 angular.module("myApp", []).controller("MyController", MyController);
 function MyController() {
  var vm= this;
  vm.message= "hello";
  vm.isSubmitted= false;
  vm.add= function() {
   if (vm.myForm.$valid) {
    console.log("Add to database...");
   } else {
    vm.isSubmitted= true;
   }
  }
 }
</script>
...
 <form name="vm.myForm" ng-submit="vm.add()">
 Please enter age greater than 9: <input type="text" name="name"
required ng-pattern="/^[0-9]{2,3}$/" ng-model="name"></input>
 <span ng-show="vm.myForm.name.$dirty &&
vm.myForm.name.$invalid && vm.isSubmitted">
   This feild requires 2 or 3 digits.
 </span>
 <button type="submit">Add data</button>
</form>
```

Forms

Field Checking

Pattern Check

Check on submit

Add Game

Add Game form to public/angular-app/game-list/games.html

```
<form name="vm.gameForm" ng-submit="vm.addGame()" >
    To add a new game please fill in all the fields below:<BR/>
    Title: <input type="text" name="title" required ng-model="vm.newGameTitle"
style="color:black"/><BR/>
    Price: <input type="text" name="price" required ng-
model="vm.newGamePrice" style="color:black"/><BR/>
    Year of Publication: <input type="text" name="year" required ng-
model="vm.newGameYear" style="color:black"/><BR/>
    Rating: <input type="text" name="rating" required ng-
model="vm.newGameRating" style="color:black"/><BR/>
    Minimum Number of Players: <input type="text" name="minPlayers" required
ng-model="vm.newGameMinPlayers" style="color:black"/>
    <span ng-show="vm.gameForm.minPlayers.$dirty &&
vm.gameForm.minPlayers.$invalid && vm.gameForm.isSubmitted"
style="color:black">Minimum players must be at least 1.</span>
    <BR/>
    Maximum Number of Players: <input type="text" name="maxPlayers" required
ng-model="vm.newGameMaxPlayers" style="color:black"/><BR/>
    Minimum Recommended Player Age: <input type="text" name="minAge"
required ng-model="vm.newGameMinAge" style="color:black"/><BR/>
    Designer name: <input type="text" name="designer" required ng-
model="vm.newGameDesigner" style="color:black"/><BR/>
    <button type="submit" class="btn-success">Add Game</button><BR/>
</form>
```

Forms

Field Checking

Pattern Check

Check on submit

Add Game

Add controller functionality for submitting. Update public/angular-app/game-list/game-list-controller.js

```
function GamesController(GameDataFactory) {
    var vm= this;
    vm.title= "Mean Games App";
    vm.isSubmitted= false;
    GameDataFactory.getAllGames().then(function(response) {
        // console.log(response);
        vm.games=  response;
    });
    vm.addGame= function(){
        var postData= {
            title: vm.newGameTitle,
            price: vm.newGamePrice,
            rate: vm.newGameRating,
            year: vm.newGameYear,
            rating: vm.newGameRating,
            minPlayers: vm.newGameMinPlayers,
            maxPlayers: vm.newGameMaxPlayers,
            minAge: vm.newGameMinAge,
            designers: vm.newGameDesigner,
        };
        if (vm.gameForm.$valid) {
            GameDataFactory.postGame(postData).then(function(response){
                console.log("Game saved");
                //
            }).catch(function(error) {
                console.log(error);
            });
        } else {
            vm.isSubmitted= true;
        }
    };
}
```

Forms

Field Checking

Pattern Check

Check on submit

Add Game

Update the Factory public/angular-app/game-data-factory/game-data-factory.js

```
function GameDataFactory($http) {
    return {
        getAllGames: getAllGames,
        getOneGame: getOneGame,
        postGame: postGame
    };

...
    function postGame(game) {
        return $http.post("/api/games/",
game).then(complete).catch(failed);
    }
```

Forms

Field Checking

Pattern Check

Check on submit

Add Game

Enable JSON processing. Update app05,js

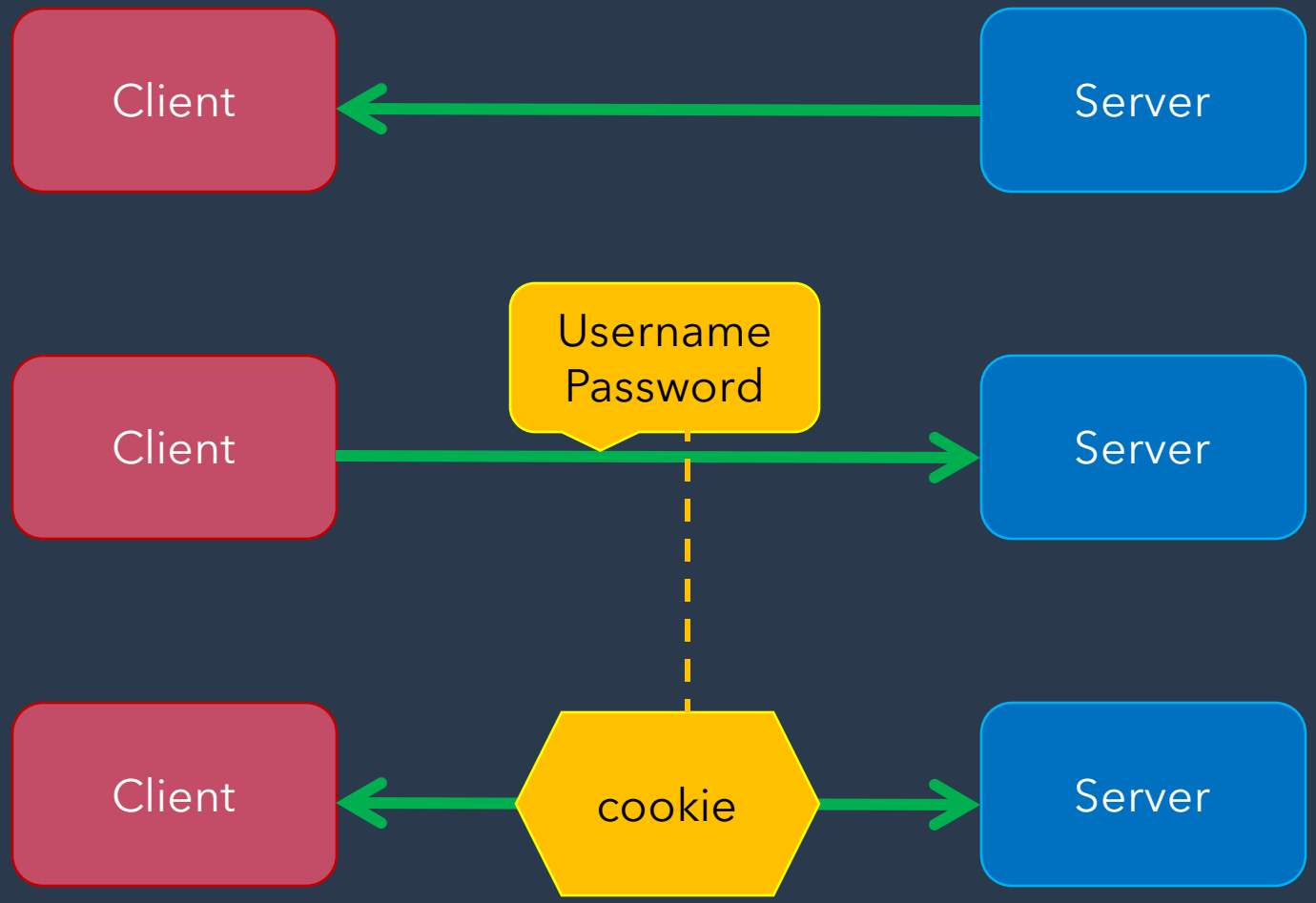
```
...
app.use(bodyParser.urlencoded({extended  : false}));
app.use(bodyParser.json());
...
```

Authentication

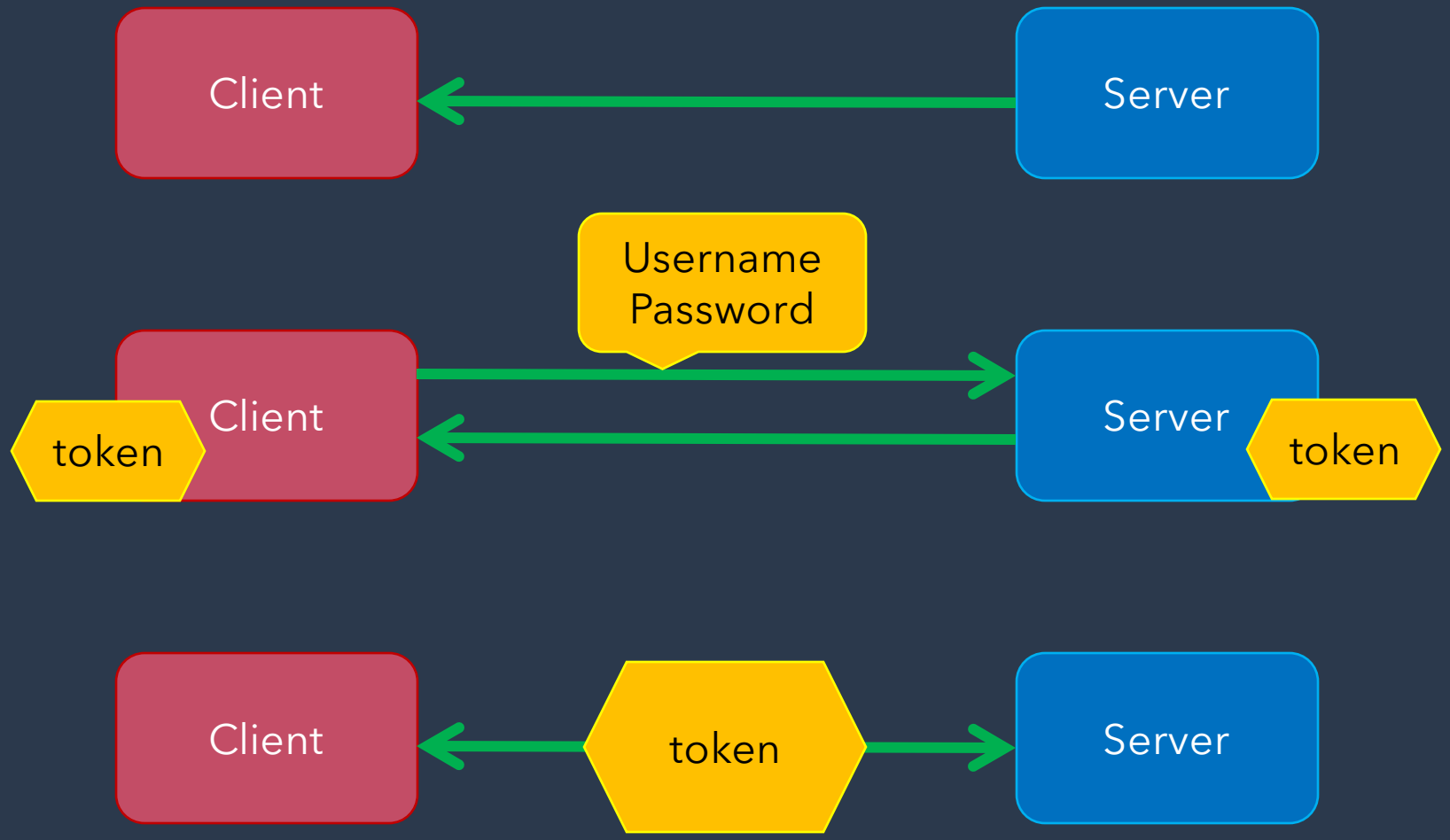
What is Authentication and Authorization

- Are you who you say you are?
- Do you have the authority (privilige) to access this?
- Authentic: Original.
- Authroized: Allowed to do this.

Classic Server Based Authentication



Token Based Authentication



JWT

- JSON Web Token
- "Header", "Payload", "Signature"
- Payload:
  - Any data, username, roles, ...
- Authentication, and encryption methods.

Authentication

Users DB

Encryption

Token Authentication

Before we can authenticate, we need to have a credentials DB. A DB of users with a first names, usernames, and passwords.

Create api/data/users-model.js

```
var mongoose= require("mongoose");
var userSchema= new mongoose.Schema({
username: {
type: String,
unique: true,
required: true
},
name: {type: String},
password: {type: String, required: true}
});
mongoose.model("User",  userSchema);
```

Authentication

Users DB

Encryption

Token Authentication

Make sure we bring in the Schema and module for the application. Update users/api/data/db.js

```
…
require("./users.model");
```

Add the authentication routes to api/routes/index.js

```
…
var controllerUsers=
require("../controllers/users.controller.js");
…
router.route("/users/register")
        .post(controllerUsers.register);
router.route("/users/login").post(controllerUsers.login);
module.exports= router;
```

Authentication

Users DB

Encryption

Token

Authentication

Add a new controller. Create api/controllers/users.controller.js

```
var mongoose= require("mongoose");
var User= mongoose.model("User");
module.exports.register= function(req, res) {
  console.log("Registering user");
  var username= req.body.username;
  var name= req.body.name || null;
  var password= req.body.password;
  User.create({username: username, name: name, password: password},
function(err, user) {
    if (err) { console.log(err); res.status(400).json(err);}
    else {console.log("user created", user); res.status(200).json(user);}
  });
};
module.exports.login= function(req, res) {
  console.log("Logging in user");
  var username= req.body.username;
  var password= req.body.password;
  User.findOne({username: username}).exec(function(err, user) {
    if (err) { console.log(err); res.status(400).json(err);}
    if (user) { console.log("user found", user); res.status(200).json(user);}
    else {console.log("user not found", user); res.status(400).json("Unauthorized");}
  });
};
```

Authentication

Users DB

Encryption

Token Authentication

Install the encryption package bcrypt-nodejs

```
npm I bcrypt-nodejs
```

Modify controller to use encryption. Update api/controllers/users-controller.js

```
var mongoose= require("mongoose");
var User= mongoose.model("User");
var bcrypt= require("bcrypt-nodejs");
module.exports.register= function(req, res){
  console.log("Registering user");
  var username= req.body.username;
  var name= req.body.name || null;
  var password= bcrypt.hashSync(req.body.password, bcrypt.genSaltSync(10));
  User.create({username: username, name: name, password: password}, function(err, user){
    if (err) { console.log(err); res.status(400).json(err);}
    else {console.log("user created", user); res.status(200).json(user);}
  });
};
module.exports.login= function(req, res) {
  console.log("Logging in user");
  var username= req.body.username;
  var password= req.body.password;
  User.findOne({username: username}).exec(function(err, user) {
    if (err) { console.log(err); res.status(400).json(err);}
    if (user) {
      if (bcrypt.compareSync(password, user.password)) {
        console.log("user found", user); res.status(200).json(user);
      } else { res.status(401).json("Unauthorized");}
    } else {console.log("user not found", user); res.status(400).json("Unauthorized");}});
};
```

Authentication

Users DB

Encryption

Token

Authentication

Install the token generation package

```
npm i jsonwebtoken
```

Use the token generator in the controller. Update api/controllers/users-controller.js

```
…
var jwt= require("jsonwebtoken);
…
module.exports.login= function(req, res) {
  console.log("Logging in user");
  var username= req.body.username;
  var password= req.body.password;
  User.findOne({username:  username}).exec(function(err, user) {
    if (err) { console.log(err); res.status(400).json(err);}
    if (user) {
      if (bcrypt.compareSync(password, user.password)) {
        console.log("user found", user);
        var token= jwt.sign({username:  user.username}, "cs572", {expiresIn: 3600});
        res.status(200).json({success:  true, token: token});
      } else { res.status(401).json("Unauthorized");}
    } else {console.log("user not found", user);
res.status(400).json("Unauthorized");}});
};
```

Authentication

Users DB

Encryption

Token

Authentication

Create a middleware function to check the existance of a token, and the token is valid. If successful it will call the next middelware function.

Update api/controllers/users-controller.js

```
...
module.exports.authenticate= function(req, res, next) {
 var headerExists= req.headers.authorization;
 if (headerExists) {
  var token= req.headers.authorization.split(" ")[1];
  jwt.verify(token, "cs572", function(err, decoded){
   if (err) { console.log(err); res.status(401).json("Unauthorized");
   } else {
    req.user= decoded.username;
    next();
   }});
  } else {res.status(403).json("No token provided");}
};
```

Validate JWT Tokens

- Use jwt.io to learn about JWT and validate tokens.
- Paste your token
  - You see the infomration in the token.
  - The signiture is not validated, because it is not in the token.
  - Type your signiture and the token signiture is validated.

Registration UI

UI

Registration

Login

Authentication

Login code

Token Management

Create a registration route, controller, and template.

Update public/angular-app/app.js

```
…
.when("/register",  {
  templateUrl: "angular-app/register/register.html",
  controller: "RegisterController",
  controllerAs: "vm"
});
```

Create registration form public/angular-app/register/register.html

```
<H1>Register</H1>
<DIV ng-if="vm.message" class="alert alert-success" role="alert">
  <P>{{ vm.message}}</P>
</DIV>
<DIV ng-if="vm.err" class="alert alert-danger" role="alert">
  {{vm.err}}
</DIV>
<FORM ng-hide="vm.message" name="register" ng-submit="vm.register()">
  <DIV class="form-group">
    <label for="username">Username</label>
    <input type="username" class="form-control" id="username" placeholder="Username" ng-model="vm.username"
autocapitalize="none"/>
  </DIV>
  <DIV class="form-group">
    <label for="password">Password</label>
    <input type="password" class="form-control" id="password" placeholder="Password" ng-model="vm.password" autocapitalize="none"/>
  </DIV>
  <DIV class="form-group">
    <label for="password-repeat">Repeat  Password</label>
    <input type="password" class="form-control" id="password-repeat" placeholder="Password" ng-model="vm.passwordRepeat"
autocapitalize="none"/>
  </DIV>
<BUTTON type="submit" class="btn btn-success">Register</BUTTON>
</FORM>
```

UI

Registration

Login

Authentication

Login code

Token Management

Create a registration route, controller, and template.

Create register controller public/angular-app/register/register-controller.js

```
angular.module("meanGames").controller("RegisterController",  RegisterController);
function RegisterController() {
  var vm= this;
  vm.register= function($http){
    var user= {username: vm.username, pasword: vm.password};
    if (!vm.username || !vm.password) { vm.err=  "Please add a username and password.";}
    else {
      if (vm.password !== vm.passwordRepeate){
        vm.err=  "Please make sure the passwords match.";
      } else {
        $http.post("/api/users/register", user).then(function(result) {
          console.log(result);
          vm.message= "Successful registration, please login.";
          vm.err=  "";
        }).catch(function(err) {console.log(err);});
      }
    }
  }
};
```

Update public/index.html to include the controller script

```
…
<script src="angular-app/register/register-controller.js"></script>
…
```

UI

Registration

Login

Authentication

Login code

Token Management

Some application changes to support login.
Add welcome page as home page.
Create a welcome template. Create public/angular-app/welcome/welcome.html

```
<H1>Welcome to MEAN Games</H1>
```

Update public/index.html

```
…
  <games-Navigation></games-Navigation>
  <DIV class="container">
    <div ng-view></div>
  </DIV>
…
  <script src="angular-app/navigation-directive/navigation-directive.js"></script>
  <script src="angular-app/login/login-controller.js"></script>
…
```

Add style to our container. Update public/css/custom.css

```
…
.container {
  padding: 60px 0 0 0;
}
```

Update public/angular-app/app.js

```
…
.when("/",  {
  templateUrl:  "angular-app/welcome/welcome.html"
})
.when("/games",  {
  templateUrl:  "angular-app/game-list/games.html",
  controller: GamesController,
  controllerAs: "vm"
})
…
.otherwise({redirectTo:   "/"});
}
```

UI

Registration

Login

Authentication

Login code

Token Management

Add a navigation bar using a directive. Create public/angular-app/navigation-directive/navigation-directive.html

```
<nav ng-controller="LoginController as vm" class="navbar navbar-expand-lg navbar-fixed-top bg-dark" ng-
init="vm.init()">
 <div class="container-fluid">
  <div class="navbar-header">
   <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-
expanded="false" aria-controls="navbar">
    <span class="sr-only">Toggle navigation</span>
    <span class="icon-bar"></span>
    <span class="icon-bar"></span>
    <span class="icon-bar"></span>
   </button>
   <a class="navbar-brand" href="#/">MeanGames</a>
  </div>
  <div id="navbar" class="collapse navbar-collapse">
   <ul class="nav navbar-nav">
    <li ng-class="vm.isActiveTab('')" class="active"><a href="#/">Home</a></li>
    <li ng-class="vm.isActiveTab('games')"><a href="#/games">Games</a></li>
    <li ng-class="vm.isActiveTab('register')"><a href="#/register">Register</a></li>
   </ul>
   <div class="col-xs-6">
    <form ng-if="!vm.isLoggedIn()" class="navbar-form navbar-right" role="login" ng-submit="vm.login()">
     <div class="form-group" ng-class="{'has-error': vm.err}">
      <input type="text" class="form-control input-sm" placeholder="Username" ng-model="vm.username"
autocapitalize="none">
     </div>
     <div class="form-group" ng-class="{'has-error': vm.err}">
      <input type="password" class="form-control input-sm" placeholder="Password" ng-
model="vm.password" autocapitalize="none">
     </div>
     <button type="submit" class="btn btn-sm btn-success">Login</button>
    </form>
   </div>
   <ul ng-if="vm.isLoggedIn()" class="new navbar-nav navbar-right">
    <li><p class="navbar-text">Welcome {{vm.loggedinUser}}</p></li>
    <li><button class="navbar-btn btn-sm btn-info" ng-click="vm.logout()">Logout</button></li>
   </ul>
  </div>
 </div>
</nav>
```

UI

Registration

Login

Authentication

Login code

Token Management

Create public/angular-app/navigation-directive/navigation-directive.js

```
angular.module("meanGames").directive("gamesNavigation",
GamesNavigation);
function GamesNavigation() {
  return {
    restrict: "E",
    templateUrl: "angular-app/navigation-directive/navigation-
directive.html"
  };
}
```

Call the directive in public/angular-app/login/login-controller.js

```
angular.module("meanGames").controller("LoginController",
LoginController);
function LoginController() {
  var vm= this;
}
```

UI

Registration

Login

Authentication

Login code

Token

Management

Create public/angular-app/authentication/auth-interceptor.js

```
angular.module("meanGames").factory("AuthInterceptor",  AuthInterceptor);
function AuthInterceptor($location, $q, $window, AuthFactory) {
  return { request: request, response: response, responseError:  responseError}
  function request(config) {
    config.headers= config.headers || {};
    if ($window.sessionStorage.token) {
      config.headers.Authorization= "Bearer "+ $window.sessionStorage.token;
    }
    return config;
  }
  function response(response){
    if (response.status === 200 && $window.sessionStorage.token && !AuthFactory.isLoggedIn)
{ AuthFactory.isLoggedIn= true;}
    if (response.status === 401) { AuthFactory.isLoggedIn= false;}
    return response || $q.when(response);
  }
  function responseError(rejection) {
    if (rejection.status ===  401 || rejection.status === 403) {
      delete $window.sessionStorage.token;
      AuthFactory.isLoggedIn= false;
      $location.path("/");
    }
    return $q.reject(rejection);
  }
}
```

Create public/angular-app/authentication/auth-factory.js

```
angular.module("meanGames").factory("AuthFactory",  AuthFactory);
function AuthFactory() {
  return {auth: auth};
  var auth= {ifLoggedId: false};
}
```

Add factories to public/index.html

```
<script src="angular-app/authentication/auth-factory.js"></script>
<script src="angular-app/authentication/auth-interceptor.js"></script>
```

UI

Registration

Login

Authentication

Login code

Token

Management

User authentication in controller, update public/angular-app/login/login-controller.js

```
function LoginController($http, $location, $window, AuthFactory) {
  var vm= this;
  vm.isLoggedIn= function() {
    if (AuthFactory.isLoggedIn) { return true}
    else {return false;}
  };
  vm.login= function() {
  }
  vm.logout= function() {
  }
  vm.isActiveTab= function(url) {
    var currentPath= $location.path().split("/")[1];
    return (url === currentPath ? "active" : "");
  }
}
```

UI

Registration

Login UI

Authentication

Login code

Token Management

Write Login code in controller, update public/angular-app/login/login-controller.js

```
....
  vm.login= function() {
    if (vm.username  && vm.password) {
      var user= {
        username:  vm.username,
        password: vm.password
      };
      $http.post("/api/users/login", user).then(function(response) {
        console.log(response);
      }).catch(function(err) {
        console.log(err);
      });
    }
  }
...
```

UI

Registration

Login UI

Authentication

Login code

Token Management

Add token management on client side. Update public/angular-app/login/login-controller.js

```
....
 vm.login= function() {
  if (vm.username && vm.password) {
   var user= {
    username: vm.username,
    password: vm.password
   };
   $http.post("/api/users/login", user).then(function(response) {
    if (response.data.success) {
     $window.sessionStorage.token= response.data.token;
     AuthFactory.isLoggedIn= true;
    }
   }).catch(function(er) {
    console.log(err);
   });
  }
 }
 vm.logout= function() {
  AuthFactory.isLoggedIn= false;
  delete $window.sessionStorage.token;
  $location.path("/");
 }
...
```

UI

Registration

Login UI

Authentication

Login code

Token Management

Dispaly menu items based on login status. Update public/angular-app/app.js

```
angular.module("meanGames", ["ngRoute"]).config(config).run(run);
function config($httpProvider, $routeProvider, $locationProvider) {
 $httpProvider.interceptors.push("AuthInterceptor");
 ....
 .when("/", {
  templateUrl: "angular-app/welcome/welcome.html",
  access: {restricted: false}
 })
 .when("/games", {
  ...
  access: {restricted: false}
 })
 .when("/game/:id", {
  ...
  access: {restricted: false}
 })
 .when("/register", {
  ...
  access: {restricted: false}
 })
 .when("/profile", {
  templateUrl: "angular-app/profile/profile.html",
  controllerAs: "vm",
  access: {restricted: true}
 })
...
}
function run ($rootScope, $location, $window, AuthFactory) {
 $rootScope.$on("$routeChangeStart", function(event, nextRoute, currentRoute) {
  if (nextRoute.access !== undefined && nextRoute.access.restricted && !$window.sessionStorage.token &&
!AuthFactory.isLoggedIn) {
    event.preventDefault(); // Do not go to that path
    $location.path("/");      // Instead go to the root
  }
 });
}
```

UI

Registration

Login UI

Authentication

Login code

Token Management

Update the navigation directive to display the menu according to the new managemnet rules. Update public/angular-app/navigation-directive/navigation-directive.html

```
...
      <li ng-class="vm.isActiveTab('register')" ng-if="!vm.isLoggedIn()"><a href="#/register">Register</a></li>
      <li ng-class="vm.isActiveTab('profile')" ng-if="vm.isLoggedIn()"><a href="#/profile">Profile</a></li>
...
```

Add the profile template, create public/angular-app/profile/profile.html

```
<H1>This is the profile Page.</H1>
```

UI

Registration
Login UI
Authentication
Login code
Token Management

Access JSON Web Token attributes from Angular.

Install package angular-jwt

```
npm i angular-jwt
```

Update LoginController to use angular-jwt to access token attributes. Update public/angular-app/login/login-controller.js

```
angular.module("meanGames").controller("LoginController", LoginController);
function LoginController($http, $location, $window, AuthFactory, jwtHelper) {
...
 vm.login = function() {
 ...
      AuthFactory.isLoggedIn= true;
      var token= $window.sessionStorage.token;
      var decodedToken= jwtHelper.decodeToken(token);
      vm.loggedInUser= decodedToken.username;
...
```

Add dependency on angular-jwt in /public/app.js

```
angular.module("meanGames", ["ngRoute", "angular-jwt"]).config(config).run(run);
...
```

Add angular-jwt to public/index.html

```
...
<script src="node_modules/angular-jwt/dist/angular-jwt.js"></script>
...
```

Authentication Based Usage

- UI
  - Only display pages when users are authenticated.
- BL
  - Only accept api calls from authenticated users.

Authentication

UI

BL

UI where only logged in users can add games.
Modify the public/angular-app/game-list/game-list-controller.js to support this

```
function GamesController($route, GameDataFactory, AuthFactory) {
  …
  vm.isLoggedIn= function() {
    if (AuthFactory.isLoggedIn) {return true;}
    else {return false;}
  };
  …
```

Update the template to use the funciton when displaying game-list.html

```
…
<div ng-if="vm.isLoggedIn()">
  <form name="vm.gameForm"  ng-submit="vm.addGame()"  >
  …
</div>
…
```

Authentication

UI

BL

BL where only logged in users can add games. Only accept a new game to be added if a user is authenticated. Update api/route/index.js

```
...
router.route("/games")
.get(controllerGames.gameGetAll)
.post(controllerUsers.authenticate,
controllerGames.gameAddOne);
...
```

Main Points

- MEAN is the ultimate separation of concerns (SoC). Not only do we have each responsibility separated in code, but each one is handled by a different framework.
- Token authentication integrates perfectly with HTTP and supports JSON which makes it perfect for MEAN applications. Both HTTP and JWT are stateless.
- We can use JWT to prevent the UI from displaying operations the user may not be allowed to perform. At the same time, we use JWT to authorize API calls.