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.