CS 572 Modern Web Applications

Najeeb Najeeb, PhD (najeeb@miu.edu)

V3.1.1

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.
- Angular: Investigate Angular and the architecture of an Angular application. Build a single-page application.
- MEAN application: Learn by example. We will create a MEAN Games application.

Why Mongoose

- Create a controller for each document and define constraints in the controller.
  - 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
Do Less Accomplish More

Wholeness

Mongoose is built on top of MongoDB driver; that is why it provides us with all the benefits of using MongoDB driver. By understanding Mongoose and using it properly we not only gain performance benefits, but also the lines of code we need to write are fewer than what is needed to perform the same task using MongoDB driver alone. When you are in tune with the laws of nature your actions become spontaneously correct. By actions being correct the first time, we do not need to spend a lot of time on an issue, we get an issue addressed properly with fewer actions.

Mongoose
Do Less Accomplish More

1. What is Mongoose and how to set it up?
2. How to use Schema with Mongoose?
3. How to perform CRUD operations in Mongoose?
4. What is GeoSearch, and how to use it?

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

What is Mongoose

- A module built on top of MongoDB driver
- Can perform all functionality performed by MongoDB driver
- Enables our application to have a Schema
- Makes it easy to perform GeoLocation Searches
- Enables us to focus on our application and it will take care of dealing with the Database
  - More application features
  - Create REST API
  - Hardening of API

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Use the last application from Lesson02 (app2)

Install Mongoose

```
npm install mongoose
mongoose@6.7.1 node_modules/mongoose
```

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Create file /api/data/db.js

```
const mongoose= require("mongoose");
mongoose.connect(process.env.DB_URL, { useNewUrlParser: true, useUn
ifiedTopology: true });
mongoose.connection.on("connected", function() {
  console.log("Mongoose connected to "+ process.env.DB_NAME);
});
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/data/db.js");
```

Add environment variables to .env

```
DB_URL= "mongodb://localhost:27017/meanGames"
DB_NAME= meanGames
```

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Create file /api/data/db.js

```
process.on("SIGINT", function() {
    mongoose.connection.close(function() {
    console.log(process.env.SIGINT_MESSAGE);
    process.exit(0);
});
});
```

Add environment variable to .env

```
SIGINT_MESSAGE= "Mongoose disconnected by app
disconnect"
```

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Create file /api/data/db.js

```
process.on("SIGTERM", function() {
    mongoose.connection.close(function() {
    console.log(process.env.SIGTERM_MESSAGE);
    process.exit(0);
});
});
```

Add environment variable to .env

```
SIGTERM_MESSAGE= "Mongoose disconnected by app termination"
```

Mongoose

Install

Connect

Disconnect

Terminate

Restart

Create file /api/data/db.js

```
process.once("SIGUSR2", function() {
    mongoose.connection.close(function() {
    console.log(process.env.SIGUSR2_MESSAGE);
    process.kill(process.pid, "SIGUSR2");
});
});
```

Add environment variable to .env

```
SIGUSR2_MESSAGE= "Mongoose disconnected by app restart"
```

Mongoose

Install

Connect

Disconnect

Terminate

Restart

New nodemon and Windows not sending SIGUSR2

Add to nodemon.json

```
{
   "signal": "SIGHUP",
   "env": {
      "NODE_ENV": "development"
   }
}
```

Run nodemon as

```
nodemon --inspect
```

Update package.json to use this

```
"dev": "nodemon --inspect",
```

To start the application run

```
npm run dev
```

Mongoose
Do Less Accomplish More

1. What is Mongoose and how to set it up?
2. How to use Schema with Mongoose?
3. How to perform CRUD operations in Mongoose?
4. What is GeoSearch, and how to use it?

Mongoose Schemas & Models

Mongoose

Add Schema

Data Validation

Compile Model

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

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

```
const mongoose= require("mongoose");
const gameSchema= mongoose.Schema({
    title: String,
    year: Number,
    rate: Number
    price : Number,
    minPlayers: Number,
    maxPlayers: Number,
    minAge: Number,
    designers : [String]
});
```

Mongoose

Add Schema

Data Validation

Compile Model

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

```
const mongoose= require("mongoose");
const gameSchema= mongoose.Schema({
    title: {
        type: String,
        required: true
    },
    year: Number,
    rate: {
        type: Number,
        min: 1,
        max: 5,
        "default": 1
    },
    price : Number,
    minPlayers: {
        type: Number,
        min : 1,
        max: 10
    },
    maxPlayers: {
        type: Number,
        min : 1,
        max: 10
    },
    minAge: Number,
    designers : [String]
});
```

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");
```

Schema

Nested Doc

Nested Docs

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

```
...
const publisherSchema= new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    country: String,
    established: Number, //Not a date since we only have year
    location: String
});
const gameSchema = mongoose.Schema({
...
    publisher: publisherSchema
});
...
```

Schema

Nested Doc

Nested Docs

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

```
…
const reviewSchema= new mongoose.Schema({
    title: {
        type: String,
        required: true
    },
    rating: {
        type: Number,
        min: 1,
        max: 5,
        required: true
    },
    review: {
        type: String,
        required: true
    },
    postDate: {
        type: Date,
        "default": Date.Now
    },
});
const gameSchema = mongoose.Schema({
…
    reviews: [reviewSchema]
});
…
```

Mongoose

GetAll

GetOne

Use Mongoose to get all Games, simpler way of doing things.
Modify file /api/controllers/games.controller.js

```
const mongoose= require("mongoose");
const Game= mongoose.model(process.env.GAME_MODEL);

const getAll= function(req, res) {
    let offset= 0;
    let 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/controllers/games.controller.js

```
const mongoose= require("mongoose");
const Game= mongoose.model(process.env.GAME_MODEL);

const getAll= function(req, res) {
    let offset= 0;
    let 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

```
const getOne= function(req, res) {
    const gameId= req.params.gameId;
    Game.findById(gameId).exec(function(err, game) {
        res.status(200).json(game);
    });
}
```

Mongoose GET

Sub-document

Sub-documents GetAll

Sub-documents GetOne

Add a route to the sub-document (based on REST rules).
Separate Controllers into logical collections.
Add a Controller for the sub-document.
Modify file /api/routes/index.js

```
…
const publisherController= require("../controllers/publisher.controllers");
…
router.route("/games/:gameId/publisher")
    .get(publisherController.getOne);
…
```

Add file /api/controllers/publisher.controllers.js

```
const mongoose= require("mongoose");
const Game= mongoose.model(process.env.GAME_MODEL);
const getOne= function(req, res){
    console.log("GET One Publisher Controller");
    const gameId= req.params.gameId;
    Game.findById(gameId).select("publisher").exec(function(err, game) {
        console.log("Found publisher ", game.publisher, " for Game ", game);
        res.status(200).json(game.publisher);
    });
}
module.exports= {
    getOne : getOne
}
```

Mongoose GET

Sub-document

Sub-documents GetAll

Sub-documents GetOne

Add a route to the sub-document (based on REST rules).
Separate Controllers into logical collections.
Add a Controller for the sub-document.
Modify file /api/routes/index.js

```
…
const reviewsController= require("../controllers/reviews.controllers");
…
router.route("/games/:gameId/reviews")
   .get(reviewsController.getAll);
…
```

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

```
const mongoose= require("mongoose");
const Game= mongoose.model(process.env.GAME_MODEL);
const getAll= function(req, res) {
   console.log("GET Reviews Controller");
   const gameId= req.params.gameId;
   Game.findById(gameId).select("reviews").exec(function(err, game) {
      console.log("Found reviews ", game.reviews, " for Game ", game);
      res.status(200).json(game.reviews);
   });
}
module.exports= {
   getAll : getAll
}
```

Mongoose GET

Sub-document

Sub-documents

GetAll

Sub-documents

GetOne

Add a route to the sub-document (based on REST rules).
Separate Controllers into logical collections.
Add a Controller for the sub-document.
Modify file /api/routes/index.js

```
…
router.route("/games/:gameId/reviews/:reviewId")
   .get(reviewsController.getOne);
…
```

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

```
...
const getOne= function(req, res) {
   console.log("GET One Publisher Controller");
   const gameId= req.params.gameId;
   const reviewId= req.params.reviewId;
   Game.findById(gameId).select("reviews").exec(function(err, game) {
      console.log("Found review ", game.review.id(reviewId), " for Game ", game);
      res.status(200).json(game.review.id(reviewId));
   });
}
module.exports= {
   getAll: getAll,
   getOne : getOne
}
```

Mongoose
Do Less Accomplish More

1. What is Mongoose and how to set it up?
2. How to use Schema with Mongoose?
3. How to perform CRUD operations in Mongoose?
4. What is GeoSearch, and how to use it?

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

```
...
getAll= function(req, res) {
  console.log("GET Games Controller");
  if (req.query && req.query.lat && req.query.lng) {
    runGeoQuery(req, res);
    return;
  }
  let offset = parseFloat(process.env.DEFAULT_FIND_OFFSET, 10);
  let count = parseFloat(process.env.DEFAULT_FIND_COUNT, 10);
  // console.log("req.query ",req.query);
  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

```
…
getAll= 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.status(200).json(games);
        }
    });
}
…
```

API - GetAll

Types Check

Error Check

Limit Check

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

```
getAll= function(req, res) {
    …
    let offset = parseInt(process.env.DEFAULT_FIND_OFFSET, 10);
    let count = parseInt(process.env.DEFAULT_FIND_COUNT, 10);
    const maxCount= parseInt(process.env.DEFAULT_MAX_FIND_LIMIT, 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) {
…
```

API - GetOne

Error Check

Result Check

Type Check?

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

```
...
getOne= function(req, res) {
    console.log("GET One Game Controller");
    const gameId = req.params.gameId;
    Game.findById(gameId).exec(function (err, game) {
        if (err) {
            console.log("Error finding game");
            res.status(500).json(err);
        } else {
            console.log("Found game", game);
            res.status(200).json(game);
        }
    });
};
...
```

API - GetOne

Error Check

Result Check

Type Check?

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

```
...
getOne= function(req, res) {
    console.log("GET One Game Controller");
    const 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) {
            console.log("Game id not found");
            res.status(404).json({"message" : "Game ID not found"});
        } else {
            console.log("Found game", game);
            res.status(200).json(game);
        }
    });
}
...
```

Reduce Termination Points

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

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

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(gamesController.getAll)
        .post(gamesController.addOne);
…
```

Modify the api/controller/gameController.

```
…
const addOne = function (req, res) {
    console.log("Game AddOne request");
    const newGame= {
        title: req.body.title, year: req.body.year, rate: req.body.rate, price: req.body.price,
        minPlayers: req.body.minPlayers, maxPlayers: req.body.maxPlayers,
        publisher: {name: "NoName"}, reviews: [], minAge: req.body.minAge,
        designers: [req.body.designers]
    };
    Game.create(newGame, function(err, game){
        const response= { status: 201, message: game };
        if (err) {
            console.log("Error creating game");
            response.status= 500;
            response.message= err;
        }
        res.status(response.status).json(response.message);
    });
}
…
```

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(publisherController.getOne)
        .post(publisherController.addOne);
...
```

Modify the api/controller/publisher.controller.js

```
...
const addOne= function(req, res) {
  console.log("Add One Publisher Controller");
  const gameId= req.params.gameId;
  Game.findById(gameId).select("publisher").exec(function(err, game) {
    console.log("Found game ", game);
    const response= { status: 200, message: game };
    if (err) {
      console.log("Error finding game");
      response.status= 500;
      response.message= err;
    } else if (!game) {
      console.log("Error finding game");
      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);
    }
  });
}
...
```

Create Game Publisher

Modify the api/controller/publisherController.js

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

Test the insert using REST API

name: The Happy Puzzle Company country: UK established: 1993 lat: 51.6385646333886 lng: -0.3065625168940575

Cleanup

remove mongodb driver code from app09.js

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

remove mongodb driver code from games.controller.js

```
const dbConneciton= require("../data/dbconnection");

const ObjectId = require("mongodb").ObjectId;
```

Delete the mongoDB driver code file
api\data\dbconnection.js

Remove the dependency from package.json

As an extra cleanup you may delete the node_modules folder and package-lock.json and then npm install

Update Documents

Update Game Publisher

To update an existing game document, create a route and a controller. Update the routes in api/routes/index.js

```
...
router.route("/games/:gameId")
    .get(gameController.getOne)
    .put(gameController.fullUpdateOne)
    .patch(gameController.partialUpdateOne);
...
```

Update api/controllers/games.controller.js

```
...
const _updateOne = function (req, res, updateGameCallback){
    console.log("Update One Game Controller");
    const gameId = req.params.gameId;
    Game.findById(gameId).exec(function (err, game) {
        const response = { status: 204, message: game };
        if (err) {
            console.log("Error finding game");
            response.status = 500;
            response.message = err;
        } else if (!game) {
            console.log("Game id not found");
            response.status = 404;
            response.message = { "message": "Game ID not found" };
        }
        if (response.status !== 204){
            res.status(response.status).json(response.message);
        } else {
            updateGameCallback(req, res, game, response);
        }
    });
}
...
```

Update Game Publisher

Full game update. Update api/controllers/games.controller.js

```
...
const fullUpdateOne = function (req, res) {
  console.log("Full Update One Game Controller");
  gameUpdate = function (req, res, game, response) {
    game.title = req.body.title; game.year = req.body.year; game.rate = req.body.rate;
    game.price = req.body.price; game.minPlayers = req.body.minPlayers;
    game.maxPlayers = req.body.maxPlayers; game.minAge = req.body.minAge;
    game.designers = req.body.designers;
    if (req.body.name) {
      console.log("Name passed");
      game.publisher = { name: req.body.name };
    } else {
      console.log("No Name passed");
      game.publisher = { name: "NoName" };
    }
    game.reviews = [];
    game.save(function (err, updatedGame) {
      if (err) {
        response.status = 500;
        response.message = err;
      }
      res.status(response.status).json(response.message);
    });
  }
  _updateOne(req, res, gameUpdate);
}
...
```

Update Game Publisher

Full game update. Update api/controllers/games.controller.js

```
...
const partialUpdateOne = function (req, res) {
    console.log("Full Update One Game Controller");
    gameUpdate = function (req, res, game, response) {
        if (req.body.title) { game.title = req.body.title; }
        if (req.body.year) { game.year = req.body.year; }
        if (req.body.rate) { game.rate = req.body.rate; }
        if (req.body.price) { game.price = req.body.price; }
        if (req.body.minPlayers) { game.minPlayers = req.body.minPlayers; }
        if (req.body.maxPlayers) { game.maxPlayers = req.body.maxPlayers; }
        if (req.body.minAge) { game.minAge = req.body.minAge; }
        if (req.body.designers) { game.designers = req.body.designers; }
        if (req.body.publisher) { game.publisher = req.body.publisher; }
        if (req.body.reviews) { game.reviews = req.body.reviews; }
        game.save(function (err, updatedGame){
            if (err) {
                response.status = 500;
                response.message = err;
            }
            res.status(response.status).json(response.message);
        });
    }
    _updateOne(req, res, gameUpdate);
}
module.exports = {
...
    fullUpdateOne: fullUpdateOne,
    partialUpdateOne: partialUpdateOne
};
```

Update Game Publisher

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

```
...
router.route("/games/:gameId/publisher")
        .get(publisherController.getOne)
        .post(publisherController.addOne)
        .put(publisherController.fullUpdateOne)
        .patch(publisherController.partialUpdateOne);
...
```

Update api/controllers/publisher.controller.js

```
...
const _updateOne= function(req, res, publisherUpdateCallback){
    console.log("Update One Publisher Controller");
    const gameId= req.params.gameId;
    Game.findById(gameId).select("publisher").exec(function(err, game) {
        console.log("Found publisher ", game.publisher, " for Game ", game);
        const response= { status: 204, message: game };
        if (err) {
            console.log("Error  Finding game");
            response.status= 500;
            response.message= err;
        } else if (!game) {
            console.log("Game with given ID not found");
            response.status= 404;
            response.message= {message: "Game ID not found"};
        }
        if (response.status !== 204) {
            res.status(response.status).json(response.message);
        }
        publisherUpdateCallback(req, res, game);
    });
}}
...
```

Update
Game
Publisher

A full update of a publisher. Update the file api/controllers/publisher.controller.js

```
...
const _fullPublisherUpdate= function(req, res, game) {
   game.publisher.name= req.body.name;
   game.publisher.country= req.body.country;
   game.publisher.established= req.body.established;
   game.publisher.location.coordinates= [parseFloat(req.body.lng),
parseFloat(req.body.lat)];
   game.save(function(err, updatedGame) {
      const response= {
         status: 204,
         message: updatedGame.publisher
      };
      if (err) {
         response.status= 500;
         response.message= err;
      }
      res.status(response.status).json(response.message);
   });
}
...
```

Update
Game
Publisher

A partial update of a publisher. Update api/controllers/publisher.controller.js

```
...
const _partialPublisherUpdate= function(req, res, game) {
    if (req.body.name) {
        game.publisher.name= req.body.name;
    }
    if (req.body.country) {
        game.publisher.country= req.body.country;
    }
    if (req.body.established) {
        game.publisher.established=  req.body.established;
    }
    if (req.body.lng && req.body.lat) {
        game.publisher.location.coordinates= [parseFloat(req.body.lng),
parseFloat(req.body.lat)];
    }
    game.save(function(err, updatedGame) {
        const response= { status: 204, message: updatedGame.publisher };
        if (err) {
            response.status= 500;
            response.message= err;
        }
        res.status(response.status).json(response.message);
    });
}
...
```

Update Game Publisher

Map the routes to the controller functions. Update api/controllers/publisher.controller.js

```
...
const fullUpdateOne= function(req, res) {
    console.log("Full Update One", req.body);
    _updateOne(req, res, _fullPublisherUpdate);
}
const partialUpdateOne= function(req, res) {
    console.log("Partial Update One", req.body);
    _updateOne(req, res, _partialPublisherUpdate);
}
module.exports= {
    getOne : getOne,
    addOne : addOne,
    fullUpdateOne: fullUpdateOne,
    partialUpdateOne: partialUpdateOne
}
...
```

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(gamesController.getOne)
      .put(gamesController.fullUpdateOne)
      .patch(gamesController.paritalUpdateOne)
      .delete(gamesController.deleteOne);
```

Update api/controllers/games.controller.js

```
...
const deleteOne = function (req, res) {
   const gameId = req.params.gameId;
   Game.findByIdAndDelete(gameId).exec(function (err, deletedGame) {
      const response = { status: 204,          message: deletedGame };
      if (err) {
         console.log("Error finding game");
         response.status = 500;
         response.message = err;
      } else if (!deletedGame) {
         console.log("Game id not found");
         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

```
...
const _deletePublisher= function (req, res, game) {
   game.publisher = { name: "NoName" };
   game.save(function(err, updatedGame) {
      const response= {
         status: 204,
         message: []
      };
      if (err) {
         response.status= 500;
         response.message= err;
      } else {
         response.status= 201;
         response.message= updatedGame.publisher;
      }
      res.status(response.status).json(response.message);
   });
}
...
```

Mongoose
Do Less Accomplish More

1. What is Mongoose and how to set it up?
2. How to use Schema with Mongoose?
3. How to perform CRUD operations in Mongoose?
4. What is GeoSearch, and how to use it?

Geo-Location Search

GEO Search

Schema

getAll

find

aggregate

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

```
...
const publisherSchema= new mongoose.Schema({
   name: {
      type: String,
      required: true
   },
   country: String,
   established: Number,
   location: {
      address: String,
      coordinates: [Number]
});
const gameSchema = mongoose.Schema({
...
   publisher: publisherSchema
});
...
```

GEO Search

Schema

getAll

find

aggregate

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

```
...
const publisherSchema= new mongoose.Schema({
...
    location: {
        // Store coordinates in order longitude (E/W), latitude (N/S)
        coordinates: {
            type: [Number],
            index: "2dsphere"
        }
    }
});
const gameSchema = mongoose.Schema({
...
    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's curvature).
- This is needed to find distance between locations
  - Near my locations.
  - Close to certain location.

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).

GEO Search

Schema

getAll

find

aggregate

First make sure your database can perform GEO Search (2dshpere index)

```
db.games.createIndex({"publisher.location.coordinates": "2dsphere"})
```

Use the same getAll and check if it is geo search or not

```
...
const getAll = function (req, res) {
    console.log("GET Games Controller");
    if (req.query && req.query.lat && req.query.lng) {
        _runGeoQuery(req, res);
        return;
    }
...
```

GEO Search

Schema

getAll

find

aggregate

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

```
const _runGeoQuery  = function (req, res) {
   const lng = parseFloat(req.query.lng);
   const lat = parseFloat(req.query.lat);
   //Geo JSON Point
   const point = { type: "Point", coordinates: [ lng, lat ] };
   const query = {
      "publisher.location.coordinates": {
         $near: {
            $geometry: point,
            $maxDistance: parseFloat(process.env.GEO_SEARCH_MAX_DIST, 10),
            $minDistance: parseFloat(process.env.GEO_SEARCH_MIN_DIST, 10)
         }
      }
   };
   Game.find(query).limit(parseFloat(process.env.DEFAULT_FIND_COUNT, 10)).exec(function
(err, games) {
      if (err) {
         console.log("Geo error ", err);
         res.status(200).json(err);
      }
      else {
         console.log("Geo results", games);
         res.status(200).json(games);
      }
   });
}
```

GEO Search

Schema

getAll

find

aggregate

Use aggregate instead of find to the game controller. Modify games.controller.js

```
const _runGeoQuery = function (req, res) {
  const lng = parseFloat(req.query.lng);
  const lat = parseFloat(req.query.lat);
  //Geo JSON Point
  const point = { type: "Point", coordinates: { lat, lng } };
  Game.aggregate([{
    "$geoNear": {
      "near": point,
      "spherical": true,
      "distanceField": "distance",
      "maxDistance": parseFloat(process.env.GEO_SEARCH_MAX_DIST, 10),
      "minDistance": parseFloat(process.env.GEO_SEARCH_MIN_DIST, 10)
    }
  }, {
    "$limit": parseFloat(process.env.DEFAULT_FIND_COUNT, 10)
  }], function (err, games) {
    if (err) {
      console.log("Geo error ", err);
      res.status(500).json(err);
    } else {
      console.log("Geo results", games);
      res.status(200).json(games);
    }
  });
}
```

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.