

# CS 572 Modern Web Applications

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

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



# 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 the 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

```
const mongoose= require("mongoose");  
const dbURL= "mongodb://localhost:27017/meanGamesDb";  
mongoose.connect(dbURL, { useNewUrlParser: true, useUnifiedTopology: 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/data/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

```
const mongoose= require("mongoose");  
const gameSchema= new mongoose.Schema({  
  title: 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

```
const mongoose= require("mongoose");
const gameSchema= mongoose.Schema({
  title: {
    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

```
const 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
  }
});

const 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

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

const 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

```
const 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)
  }
});

const 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

```
const 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"
    }
  }
});

const 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's 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

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

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

```
const 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

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

```
const Game= mongoose.model("Game");
```

```
module.exports.gamesGetOne= function(req, res) {
```

```
  const 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

```
const 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

```
const mongoose= require("mongoose");  
const Game= mongoose.model("Game");  
module.exports.reviewGetAll= function(req, res) {  
  const 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

```
const mongoose= require("mongoose");  
const Game= mongoose.model("Game");  
module.exports.reviewGetOne= function(req, res){  
  const gameId= req.params.gameId;  
  const reviewId= req.params.reviewId;  
  console.log("GET reviewId "+ reviewId+ " for gameId "+ gameId);  
  Game.findById(gameId).select("reviews").exec(function(err, game) {  
    const 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

```
const runGeoQuery= function(req, res) {
  const lng= parseFloat(req.query.lng);
  const lat= parseFloat(req.query.lat);
  console.log("Geo search lng ", lng, " lat ", lat);
  const query= {
    "publisher.location": {
      $near: {
        $geometry: {
          type: "Point",
          coordinates: [lng, lat]
        },
        $maxDistance: 1000, // or 1000000
        $minDistance: 0
      }
    }
  }
  Game.find(query).exec(function(err, games){
    if(err) {
      console.log("Error ", err);
    }
    console.log("Found games", games);
    res.status(200).json(games);
  });
};

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);
  }
  ...
};
```

Make sure you index the database to use 2dsphere on relevant location

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