Week 2 Workshop

NodeJS Express MongoDB

Agenda

Activity	Time
Get Prepared: Log in to Nucamp Learning Portal • Slack • Screenshare	10 minutes
Check-In	10 minutes
Week Recap	40 minutes
Task 1 & 2	60 minutes
BREAK	15 minutes
Tasks 2 & 3 - Leave time for testing!	90 minutes
Check-Out	15 minutes

Check-In

- How was this week? Any particular challenges or accomplishments?
- Did you understand the Exercises and were you able to complete them?
- You must complete all Exercises before beginning the Workshop Assignment.

Week 2 Recap - Overview

New Concepts This Week

• Express Generator
• MongoDB
• NoSQL vs SQL/relational DBs
• Mongo REPL Shell
	• MongoDB Node Driver
	• Mongo REPL Shell
	• Mongoose ODM
	  • Schemas
	  • Models

Next slides will review these concepts

Express Generator

- Use to scaffold out a starter Express application
- Will automatically install some middleware for you, set up the server, error handling, public folder, etc
- **Discuss:** Ask students to share a few answers from the Challenge Question: Express Generator Dependencies

SQL

- Structured Query Language
- Used to perform CRUD operations on relational databases
- Relational databases are based on a relational model proposed in 1970, SQL is based on relational algebra
- Data represented as tables with rows and columns
- Strict schema, all rows in a table must have matching columns
- RDBMS – Relational Database Management System – examples are Oracle, MS Access, MySQL
- SQL/relational DBs are in widespread use

NoSQL

- Loose classification of databases that are non-relational and do not use SQL
- Developed in modern times to address some shortcomings of SQL/relational DBs
- Four types (at least) - key-value based, graph based, column-faily based, document based

MongoDB

- MongoDB is document based, documents are records kept in JSON-like format (BSON under the hood – Binary JSON)
- Short for 'humongous', designed to handle large amounts of data
- Database -> Collection -> Document -> Document Field
- Roughly think of collections as tables, documents as rows (records), fields as cells
- **Discussion:** Can anyone name an example of a collection? How about an example of a document? How about an example of a document field?

MongoDB

- BSON adds additional data types including ObjectId
- If a document is added to MongoDB with no _id field, a unique _id field is generated using ObjectId API
- ObjectID includes a timestamp for when it was generated that can be decoded from the ObjectID
- Use MongoDB's auto-generated ObjectID for _id field unless you have a reason not to

MongoDB/NoSQL vs SQL

- **Discuss**: What do you (the students) remember from this week's discussion about the advantages of MongoDB vs SQL/relational databases? For both – advantages of MongoDB and advantages of SQL/relational databases.
- *(some answers next slide)*

MongoDB/NoSQL vs SQL

Some advantages of MongoDB:

- Vertical scaling vs Horizontal scaling –
  - SQL databases can typically only scale vertically, by adding more CPU/RAM/disk space/etc to the server that the DB is on.
  - MongoDB (and other NoSQL databases) are designed to be easy to distribute across multiple servers (horizontal scaling); great for cloud computing
- Ease of deployment
  - Using JSON-like documents means it's easy to integrate with JavaScript/Node.js-based server-side applications – thus very popular in web apps

MongoDB/NoSQL vs SQL

Some advantages of SQL:

- Mature, established, well-supported technology –
  - We've been using SQL since the '70s. It's battle-tested and in widespread use, lots of support, lots of tools in its ecosystem, large community if you need help.
- More stable, better data integrity
  - Better for high-transaction uses with frequent updates and records that require high data integrity, such as banking applications.
  - MongoDB would be better for, for example, storing many blog posts that are rarely changed, where data integrity is not crucial.
- Faster at performing complex, analytical queries

MongoDB

- Once you have downloaded and installed the MongoDB software, and started the server, you can type `mongo` from anywhere to access the MongoDB server through a REPL shell
- **Discuss:** What does REPL stand for, and what does it mean?
- While you can manipulate the database, create/read/update/delete records through the Mongo REPL shell, you will need another way to access the database server to use with your Express server application

MongoDB Node.js Driver

- Provides API we can use from within any Node.js application to connect to and communicate with a MongoDB server
- The application then becomes the client to the MongoDB server, sends the database server requests and processes responses
- So our Express application acts as a server to the HTTP client (browser, Postman) but then in turn acts as a client to the MongoDB server

Callback Hell

- Asynchronous methods in the MongoDB Node.js Driver's API take a callback as a final argument
- That callback is then called at the end of the method and used for running the next operation
- This can lead to Callback Hell/the Pyramid of Doom:

```
coll.someMethod1(arg, (err, result) => {
    // do some stuff with err and result
    coll.someMethod2(arg, (err, result) => {
        // do some stuff with err and result
        coll.someMethod3(arg, (err, result) => {
            // do some stuff with err and result
        });
    });
});
```

Dealing with Callback Hell with Promises

- We can get out of Callback Hell in various ways, ES6 JavaScript Promises is a good approach
- MongoDB Node.js Driver async methods have built-in promise support
- **Discuss:** How do you cause a method in the MongoDB Node.js Driver to return its value as a promise? Ask for student answers. (*Answer next slide*)

Dealing with Callback Hell with Promises

- If you leave out the callback as the final argument, then the method will automatically return its return value as a promise
- You can then use .then and .catch methods to deal with the results asynchronously in an easier-to-read, top-down way
- No longer need to use the error callback convention because promises deal with errors using reject handlers

Mongoose ODM

- MongoDB does not force any structure on documents
- Documents of completely different structure/fields can co-exist in the same collection
- If you want a collection to have a structure, up to you to enforce it
- One popular way to do so is using the Mongoose library

Mongoose ODM

- Mongoose is a wrapper library around the MongoDB Node.js Driver
- Once installed and required, you can use its methods to connect to and access the MongoDB server
- Then you can use Schemas and Models and the Mongoose methods (such as for inserting a new document) will automatically enforce them

Mongoose Schemas

- When creating an instance of a Schema, you provide an argument of an object that defines rules for the fields you want in a document, e.g.:
- Second optional argument for configuration options
- Discuss:
  - What does `timestamps: true` do?
  - What does `unique: true` do?

(*answers next slide*)

```
const campsiteSchema = new Schema({
    name: {
        type: String,
        required: true,
        unique: true
    },
    description: {
        type: String,
        required: true
    }
}, {
    timestamps: true
});
```

Mongoose Schemas

- Answers:
  - timestamps: true as an optional configuration causes Mongoose to automatically generate CreatedAt and UpdatedAt fields when a document is created using that Schema
    - Mongoose will also automatically update the UpdatedAt field when appropriate
  - unique: true set for a field in the Schema will ensure that the field is unique for all documents using that Schema, even if it's not being used as the unique key/_id.
    - Example: If you set this true for a name field, then all names must be unique within the document.

Mongoose Schemas - Subdocuments

- You can add subdocuments to documents in a Schema
- Create the Schema for the subdocument as a normal Schema instance
  - e.g. `const comments = new Schema({ … }, { … });`
- Then use it as a field in another Schema instance:
  - `comments: [commentSchema]`
- The above syntax will add comments as a subdocument array, also possible to add a single subdocument, not as an array, though we did not discuss this during the week:
  - `comment: [commentSchema]`

Mongoose Schemas - Subdocuments

- Subdocuments, like normal documents, have an auto-generated _id field with ObjectId, have their own CreatedAt and UpdatedAt timestamps if timestamps: true option is configured in Schema
- Associated with a parent document
  - You cannot save a subdocument, you can only save it by saving the parent document
- Special id() method to find a subdocument by ID: parentDocument.subDocuments.id(_id)
- Special remove() method to remove a subdocument by ID: parentDocument.subDocuments.id(_id).remove()

```
campsite.comments.id(campsite.comments[i]._id).remove();
```

Mongoose Models

- Schemas are used in creating Models
- Models are created using mongoose.model() method
- Must pass in the upper-cased, singular version of the collection name you want to use (e.g. Campsite for campsites collection) along with a Schema

```
const Campsite = mongoose.model('Campsite', campsiteSchema);
```

- Models are basically classes – you may see them referenced in documentation as "function constructors" or "constructors" or "constructor functions" - this is the de-sugared version of ES6 classes, commonly used in code written pre-ES6

Mongoose Models

- Once a Model is created, you can use many static methods on it
- Static methods are methods that are used on the class itself, rather than on an instance of the class
  - Examples: Campsite.create(arg) will create a new campsite document using the Campsite model, Campsite.find() will find all documents based on the Campsite model
- **Discuss:** In the API documentation for Mongoose, find an example of a *non-static* method of Model. How can you tell that it's not a static method? (*answer next slide*)

Mongoose Models

- **Answer**:
- Any of the methods listed as Model.prototype.*<methodname>* is a non-static method, meant to be used on an instance of Model (an object created using Model.create()) rather than on Model itself.
  - E.g. Model.prototype.save() is used like this: campsite.save(), not Model.save()

Workshop Assignment

- It's time to start the workshop assignment!
- **Leave yourselves time for testing with Postman after Task 3.**
- Break out into groups of 2-3. Sit near your workshop partner(s).
  - Your instructor may assign partners, or have you choose.
- Work closely with each other.
  - 10-minute rule does *not* apply to talking to your partner(s). You should consult each other throughout.
- Follow the workshop instructions very closely.
  - Both the video and written instructions. Pay careful attention to any screenshots in the written instructions.
- Talk to your instructor if any of the instructions are unclear to you.

Check-Out

- Submit to the learning portal one of the following options:
  - Either: a zip file of your entire nucampsiteServer folder with your updated files, *excluding* the node_modules folder,
  - Or: a text file that contains the link to a public online Git repository for the nucampsiteServer folder.
- Wrap up – Retrospective
  - What went well
  - What could improve
  - Action items
- Start Week 3 or work on your Portfolio Project.
- If everyone is done early, then take time to go over the Code Challenges and Challenge Questions from this week – for each one, a student volunteer who has completed the challenge may explain their answer to the class.