MongoDB and Mongoose

Software Engineering - Lab

Marco Robol - marco.robol@unitn.it

Academic year 2024/2025

Contents of today class

- MongoDB and Mongoose

> EasyLib: github.com/unitn-software-engineering/EasyLib

MongoDB - mongodb.com

A distributed, **document-oriented** database that stores data in JSON-like documents, where fields can vary from document to document.

- **Database** - a physical container for collections. Each database gets its own set of files on the file system;
- **Collection** - a group of documents that exists within a single database. Collections do not enforce a schema;
- **Document** model maps to the **objects** in your application code; Typically, all documents in a collection are of similar or related purpose;
- **Queries** and **aggregation** provide powerful ways to access and analyze your data.

> https://www.mongodb.com/en-us/what-is-mongodb

Getting Started

> https://www.mongodb.com/docs/guides/server/introduction/

- Define Your Data Set
- Start Thinking in JSON
- Identify Candidates for Embedded Data and Model Your Data

```
{ "name": "notebook",
  "qty": 50,
  "rating": [ { "score": 8 }, { "score": 9 } ],
  "size": { "height": 11, "width": 8.5, "unit": "in" },
  "status": "A",
  "tags": [ "college-ruled", "perforated"]
}
```

Get MongoDB

- Install MongoDB locally - www.mongodb.com/try/download/community
  - Tutorial https://www.mongodb.com/docs/guides/server/install/
- Use MongoDB as a service - cloud.mongodb.com
- Develop on codesandbox.io or replit.com

MongoDB Shell `mongosh`

- https://www.mongodb.com/docs/mongodb-shell/run-commands/

MongoDB as a service - cloud.mongodb.com

- Register on cloud.mongodb.com
- Create a new project
- Build a Database (Free version)
  - Setup username and password used to connect db
- Go to Network Access -> Add IP adress -> Allow Access from Anywhere
- Go back on 'Datbase' Click and click on 'Connect' to get connection details.

> Replace <password> with the password for the admin user. Replace myFirstDatabase with the name of the database that connections will use by default. Ensure any option params are URL encoded.

```
mongodb+srv://admin:<password>@cluster0.f9mww.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
```

Mongoose mongoosejs.com

> elegant mongodb object modeling for node.js

> Mongoose provides a straight-forward, **schema-based** solution to model your application data. It includes *built-in type casting, validation, query building, business logic hooks* and more, out of the box.

Get Mongoose

```
$ npm install mongoose
```

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/test');

const Cat = mongoose.model('Cat', { name: String });

const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

> https://mongoosejs.com/docs/guide.html

Defining your schema

```
import mongoose from 'mongoose';
const { Schema } = mongoose;

const bookSchema = new Schema({
  title:  String, // String is shorthand for {type: String}
  author: String,
  body:   String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs:  Number
  }
});
```

Ids - By default, Mongoose adds an _id property to your schemas.

```
bookSchema.path('_id'); // ObjectId { ... }
```

Creating a model

To use our schema definition, we need to convert our **bookSchema** into a **Model** we can work with. To do so, we pass it into mongoose.model(modelName, schema):

```
const BookModel = mongoose.model('Book', bookSchema);
```

When you create a new document, a new _id of type ObjectId is created.

```
const doc = new BookModel();
doc._id instanceof mongoose.Types.ObjectId; // true
```

Querying

> https://mongoosejs.com/docs/models.html#querying

Finding documents is easy with Mongoose, which supports the rich query syntax of MongoDB. Documents can be retrieved using a model's **find**, **findById**, **findOne**, or **where** static methods.

```
BookModel.find({ size: 'small' }).where('createdDate').gt(oneYearAgo).exec(callback);
```

Saving

> https://mongoosejs.com/docs/documents.html#updating-using-save

```
const doc = await MyModel.findOne();
doc.name = 'foo';
await doc.save();
```

Subdocuments versus Nested Paths

> https://mongoosejs.com/docs/subdocs.html#subdocuments-versus-nested-paths

```
// Subdocument
const subdocumentSchema = new mongoose.Schema({
  child: new mongoose.Schema({ name: String, age: { type: Number, default: 0 } })
});
const Subdoc = mongoose.model('Subdoc', subdocumentSchema);
// subdoc.child may be undefined

// Nested path
const nestedSchema = new mongoose.Schema({
  child: { name: String, age: { type: Number, default: 0 } }
});
const Nested = mongoose.model('Nested', nestedSchema);
// nested.child will never be undefined
```

Populate

> https://mongoosejs.com/docs/populate.html

```
const personSchema = Schema({
  _id: Schema.Types.ObjectId,
  name: String,
  stories: [{ type: Schema.Types.ObjectId, ref: 'Story' }]
});
const storySchema = Schema({
  author: { type: Schema.Types.ObjectId, ref: 'Person' },
  title: String,
  fans: [{ type: Schema.Types.ObjectId, ref: 'Person' }]
});
const Story = mongoose.model('Story', storySchema);
const Person = mongoose.model('Person', personSchema);

Story.findOne({ title: 'Casino Royale' })
.populate('author').exec(function (err, story) { ... });
```

MongoDB with mongoose in EasyLib

> https://github.com/unitn-software-engineering/EasyLib

How to run: `npm run start_local`

package.json

```
  "scripts": {
    "start": "node index.js",
    "dev": "node -r dotenv/config index.js" }, ...
```

What is *-r dotenv/config* ? ...

dotenv - www.npmjs.com/package/dotenv

```
$ npm install dotenv
```

> Dotenv loads environment variables from a .env file into process.env.

```
require('dotenv').config()
console.log(process.env) // remove this after you've confirmed it working
```

Preload - You can use the --require (-r) command line option to preload dotenv. By doing this, you do not need to require and load dotenv in your application code.

```
$ node -r dotenv/config your_script.js
```

Let's go back on mongoose and EasyLib

- **mongoose models**
  - app/models/
- **express routers**
  - app/

app/models/book.js

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// set up a mongoose model
module.exports = mongoose.model('Book', new Schema({
        title: String
}));
```

app/books.js

```
const Book = require('./models/book');

router.get('', async (req, res) => {
    // https://mongoosejs.com/docs/api.html#model_Model.find
    let books = await Book.find({});
    ...
}
router.get('/:id', async (req, res) => {
    // https://mongoosejs.com/docs/api.html#model_Model.findById
    let book = await Book.findById(req.params.id);
    ...
}
router.post('', async (req, res) => {
    let book = new Book({
            title: req.body.title
    });
    book = await book.save();
    res.location("/api/v1/books/" + book.id).status(201).send();
});
```

index.js

```
const app = require('./app/app.js');
const mongoose = require('mongoose');

app.locals.db = mongoose.connect(process.env.DB_URL,
  {useNewUrlParser: true, useUnifiedTopology: true})
.then ( () => {
    console.log("Connected to Database");
    app.listen(8080, () => { console.log(`Server listening`) });
});
```

Define your own collections and their schema

- Starting from your APIs resources, define collections and their schema.
- You may incorporate some resources into others as subdocuments. For example, booklendings could be nested under book.

```
// https://mongoosejs.com/docs/subdocs.html#adding-subdocs-to-arrays
const Parent = mongoose.model('Parent');
const parent = new Parent();
parent.children.push({ name: 'Liesl' });

// https://mongoosejs.com/docs/subdocs.html#subdoc-parents
const schema = new Schema({
  docArr: [{ name: String }],
  singleNested: new Schema({ name: String })
});
```

- Apply populate() when necessary.

Questions?

marco.robol@unitn.it