MongoDB and Mongoose

Software Engineering, Tutorial

Antonio Bucchiarone - bucchiarone@fbk.eu

Academic year 2022/2023

Contents of today class

- JSON
- NodeJS and NPM
- MongoDB and Mongoose

> Material: https://github.com/antbucc/IS-22_23

MongoDB - mongodb.com

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

- MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time
- The document model maps to the objects in your application code, making data easy to work with
- Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze your data
- MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use
- MongoDB is free to use.

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 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 'Database' 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.

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.

Express

> Express is a minimal and flexible Node.js **web application framework** that provides a robust set of features for web and mobile applications.

> With a myriad of **HTTP utility methods** and **middleware** at your disposal, creating a **robust API** is quick and easy.

> Express provides a thin layer of fundamental **web application** features, without obscuring Node.js features.

- Install Express and Mongoose by executing the following command on a terminal.

  `npm install express mongoose --save`

- Create a new file app.js and add the following code:

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
    res.send('Hello World!')
})

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

- This app starts a server and listens on port 3000 for connections. The app responds with “Hello World!” for requests to the root URL (/) or route.

> To create a connection to MongoDB Atlas, follow the next steps.

- Open your Cluster tab in MongoDb Atlas and click CONNECT.
- Select **Connect your application** and choose **Node.js** for the driver.
- **Copy** the connection string.

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

> With the connection at hand, write the following to connect the DB

```
const mongoAtlasUri =
    "mongodb+srv://antbucc:test1234@cluster0.szazbxv.mongodb.net/test?retryWrites=true&w=majority";

try {
    // Connect to the MongoDB cluster
    mongoose.connect(
        mongoAtlasUri,
        { useNewUrlParser: true, useUnifiedTopology: true },
        () => console.log(" Mongoose is connected"),
    );
} catch (e) {
    console.log("could not connect");
}

const dbConnection = mongoose.connection;
dbConnection.on("error", (err) => console.log(`Connection error ${err}`));
dbConnection.once("open", () => console.log("Connected to DB!"));
```

Creating the Data Schema for your application

> Create another file ***models.js*** and add the following code.

- We create a schema **UserSchema** using the ***mongoose.Schema()*** method.

```
const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  age: {
    type: Number,
    default: 0,
  },
});
```

Creating a model

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

```
const User = mongoose.model("User", UserSchema);

module.exports = User;
```

Add/Save a User

```
const user = new User({ name: 'Antonio Bucchiarone', age: 32 });
user.save(function (err) {
    if (err) return handleError(err);
    // saved!
    console.log("user saved");
});
```

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

inserting large batches of documents

```
userModel.insertMany(
    [
        { name: 'mario rossi', age: 40 },
        { name: 'paolo neri', age: 25 }],
    function (err) {
        console.log("array of users added");
    });
```

Deleting

- Models have static `deleteOne()` and `deleteMany()` functions for removing all documents matching the given filter.

```
userModel.deleteOne({ name: 'Antonio Bucchiarone' }, function (err) {
    if (err) return handleError(err);
    // deleted at most one tank document
    console.log("element deleted");
});
```

```
userModel.deleteMany({ name: 'Antonio Bucchiarone' }, function (err) {
    if (err) return handleError(err);
    // deleted at most one tank document
    console.log("elements deleted");
});
```

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.

```
// Find one user whose `name` is 'Antonio Bucchiarone', otherwise `null`
userModel.findOne({ name: 'Antonio Bucchiarone' }).exec();

// using callback
userModel.findOne({ name: 'Antonio Bucchiarone' }, function (err, user) {
    console.log("age: "+user.age);
});
```

Model.find()

> In Mongoose, the **Model.find()** function is the primary tool for querying the database.
> The first parameter to Model.find() is a ***filter*** object.
> MongoDB will search for **all documents that match the filter**.
> If you pass an **empty filter**, MongoDB will return **all documents**.

Example

- Suppose you have a **Character model** that contains 5 characters from Star Trek: The Next Generation.

```
const Character = mongoose.model('Character', mongoose.Schema({
 name: String,
 age: Number,
 rank: String
}));

Character.create([
 { name: 'Jean-Luc Picard', age: 59, rank: 'Captain' },
 { name: 'William Riker', age: 29, rank: 'Commander' },
 { name: 'Deanna Troi', age: 28, rank: 'Lieutenant Commander' },
 { name: 'Geordi La Forge', age: 29, rank: 'Lieutenant' },
 { name: 'Worf', age: 24, rank: 'Lieutenant' }
]);
```

```
Character.find({}).sort('age').
   exec(function (err, docs) {
       console.log(docs);
   });
```

```
Character.find({ rank: 'Lieutenant' }).exec(function (err, docs) {
    console.log("documenti trovati: " + docs.length);
});
```

- You can also query **by age**. For example, the below query will find all characters whose age is 29.

```
const docs = Character.find({ age: 29 });
console.log("elementi trovati: "+docs);
```

Suggested link

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

Questions?

bucchiarone@fbk.eu