Node.js

Day 3

Alexandre Perrin

July 12, 2017

nomades.ch

Today

- MongoDB
  - Collections and Documents
  - MongoDB Shell
  - CRUD
  - Mongoose
  - Node MongoDB Native
- API design
  - RESTful API
- Let’s code!

MongoDB

NoSQL and MongoDB

NoSQL databases are *non relational* and usually lack *ACID transactions*.

MongoDB is a NoSQL, *document-oriented* database Engine using *JSON-like* documents. It is popular in combination with Node.js/Express.js and AngularJS (aka the ”MEAN” stack).

MongoDB is well documented at docs.mongodb.com.

Collections and Documents

In the MongoDB terminology, a *collection* is a group of *documents* within a single *database*. It is the equivalent of an RDBMS table, although unlike a table a *collection* does not enforce a schema.

A *document* is a record in a MongoDB *collection* and the basic unit of data in MongoDB. Documents are analogous to JSON objects but exist in the database in a more type-rich format known as BSON.

MongoDB Shell

MongoDB has a interactive JavaScript command line shell (based on Mozilla SpiderMonkey).

```
% mongo
MongoDB shell version v3.4.6
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.6
Welcome to the MongoDB shell.
For interactive help, type "help".
> interpreterVersion()
MozJS-38
> console.log("Hello Mongo?")
2017-07-08T14:07:57.981+0200 E QUERY    [thread1] ReferenceError: console is not
     defined : @(shell):1:1
> print("Hello Mongo!")
Hello Mongo!
> [1, 2, 3].map(x => x * x)
[ 1, 4, 9 ]
```

You can write Mongo Shell scripts and execute them:

```
% mongo dbname myscript.js
```

MongoDB Shell

databases and *collections* are automatically created when the first document is inserted:

```
> show dbs
admin  0.000GB
local  0.000GB
> db
test
> show collections
> db.posts.insertOne({title: "Homemade Brownie"})
    {
        "acknowledged" : true,
        "insertedId" : ObjectId("5960ce1537eda72315ee913f")
    }
> show dbs
admin  0.000GB
local  0.000GB
test   0.000GB
> show collections
posts
> db.posts.find().pretty()
{
    "_id" : ObjectId("5960ce1537eda72315ee913f"),
    "title" : "Homemade Brownie"
}
```

Create

Creating one document:

```
db.posts.insertOne({                    # collection
    title: "Bacon Avocado Salad",       # field: value |
    body: "Place bacon in a large...",  # field: value | document
    score: 5,                           # field: value |
})
```

Creating many documents:

```
db.posts.insertMany([
    {title: "Crispy Orange Beef", body: "Lay beef strips out in...", score: 2},
    {title: "Simple BBQ Ribs", body: "Place ribs in a large...", score: 4},
])
```

More at create-operations.

Read

MongoDB support many query operators, like `$lte`, `$eq`, `$neq`, `$in`, etc.

```
db.posts.find(              # collection
    {score: {$gt: 3}},      # query criteria
    {title: 1, score: 1}    # projection
).limit(5)                  # cursor modifier
```

Additionally, it provide more powerful interfaces for queries like an Aggregation Pipeline, the `$where` operator, and Map-Reduce.

More at read-operations.

Update

Updating one document:

```
db.posts.updateOne(                 # collection
    {title: "Simple BBQ Ribs"},     # update filter
    {$set: {score: 2}}              # update action
)


db.posts.findAndModify({
    query:  {title: "Simple BBQ Ribs"},
    update: {$set: {score: 1}},
    "new": true,
})
```

Updating many documents:

```
db.posts.updateMany(
    {title: /avocado/i},
    {$set: {score: 9001}}
)
```

More at update-operations.

Delete

Deleting one document:

```
db.posts.deleteOne(                 # collection
    {title: "Crispy Orange Beef"}  # delete filter
)
```

Deleting many documents:

```
db.posts.deleteMany(
    {score: {$lt: 5}}
)
```

More at delete-operations.

Indexes

Indexes should be used in order to optimize queries, otherwise MongoDB must scan the whole collection:

```
db.posts.createIndex({score: 1}, {unique: false});
```

MongoDB automatically create a unique index on the `_id` field that can not be dropped.

More at indexes and db.collection.createIndex().

Model Data for Atomic Operations

in MongoDB, *all write operations* are atomic on the level of *a single document*.

See Model Data for Atomic Operations.

Mongoose

Mongoose is a MongoDB object modeling framework for Node.js. It provide mechanisms to describe *Schema*, *validations*, limited *versioning*, *relations* (!), and more.

```
1  var mongoose = require('mongoose');
2  mongoose.connect('mongodb://localhost/test');
3
4  var Cat = mongoose.model('Cat', { name: String });
5
6  var kitty = new Cat({ name: 'Zildjian' });
7  kitty.save(function (err) {
8    if (err) {
9      console.log(err);
10   } else {
11     console.log('meow');
12   }
13 });
```

Documentation at mongoosejs.com.

Node MongoDB Native

The official MongoDB Node.js driver the most direct way to talk to MongoDB. It is also the most similar to the MongoDB Shell interface.

Documentation at mongodb.github.io.

Node MongoDB Native

```
"use strict";

const MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017/test', function (err, db) {
    if (err)
        return console.error(err.message);
    const collection = db.collection('cats');
    collection.findOne({name: 'Zildjian'}, (err, kitty) => {
        if (err)
            return console.error(err.message);
        console.dir(kitty, {colors: true});
        collection.findAndModify(
        /* query  */ {_id: kitty._id, __v: kitty.__v},
        /* sort   */ [],
        /* update */ {
            $set: {name: 'Azrael'},
            $inc: {__v: 1} },
        /* options  */ {'new': true},
        /* callback */ function (err, result) {
            if (err)
                return console.error(err.message);
            console.dir(result.value, {colors: true});
            db.close();
        });
    });
});
```

API design

API design

Designing an API is the most critical (and also probably the hardest) part of a project.

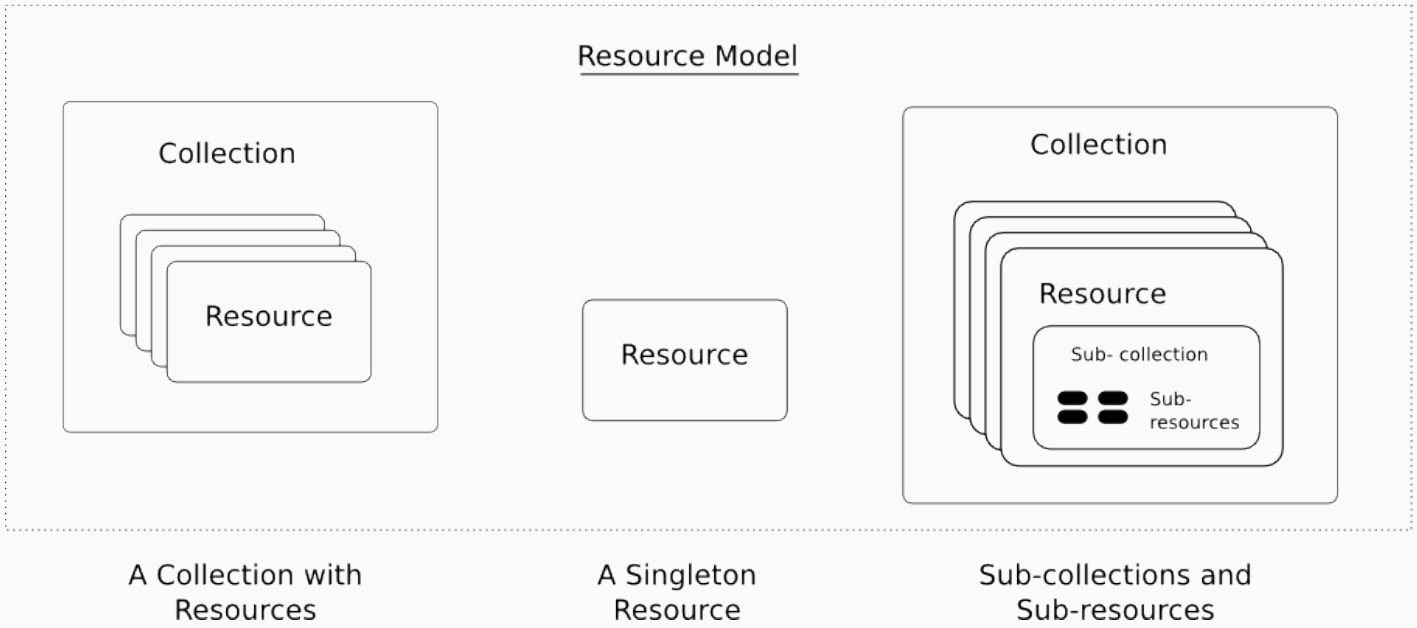
There is a ton of resource on the web to help, for example the five golden rules for designing a great Web API. Experience as both user and designer is key. Always keep in mind the Robustness principle.

Be liberal in what you accept, and conservative in what you send.

RESTful API

REST is an architectural style for API widely adopted in the world wide web. In other words, it is a set of constraints (rules) for API designers.

The fundamental concept in any RESTful API is the *resource*.



CRUD and HTTP verbs

- Create: `POST`
- Read: `GET`
- Update: `PUT and PATCH`
- Delete: `DELETE`

Note that `GET` is a ”Safe method”.

While all HTTP verbs are supported using AJAX, browsers are only able to perform `GET` and `POST` requests from a `<form>`. Thus, it is common to allow method overriding, see for example method-override for Express.js

RESTful API

We'll focus on building a RESTful API using HTTP and JSON based on Heroku's HTTP API Design Guide. Another great, comprehensive resource is RESTful API design.

Let’s code!

Yet Another Blog Engine

1. Replace the static hard-coded posts with MongoDB
2. **Ensure that your API is RESTful and well documented in your own project**.
3. Add a state-changing action, for example publish & unpublish (optional). Rational at RESTful API Design.

```
app.post('/api/posts/:id/actions/publish', (req, res) => ...);
app.post('/api/posts/:id/actions/unpublish', (req, res) => ...);
```

Questions?

Read on later

You Are Not Google by Ozan Onay.