Node.js

Ch 7: Mongo DB

hogeschool vives

mongoose

mongoDB

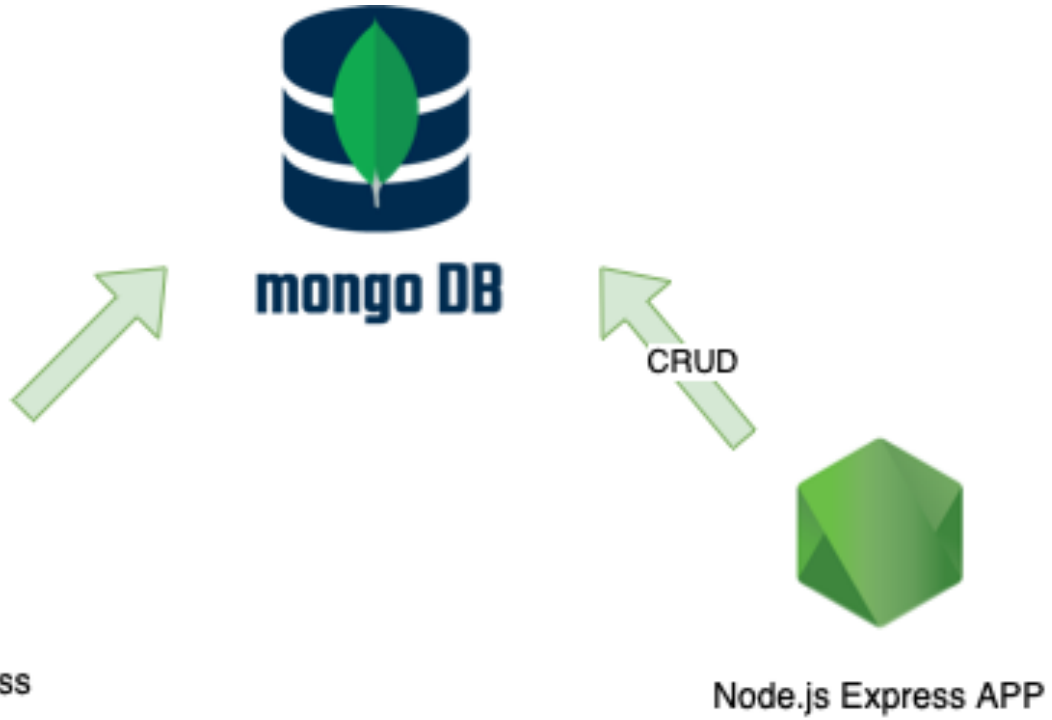
D. HOSTENS & M. DIMA

— Database

- Until now we always kept data in *memory objects*
- This is not ideal, each server restart we lose all new data
- Solution => use a database
- This time we choose MongoDB, but other DB's are also an option
- MongoDB is NOT a relational DB
- No tables, schemas, views, records, etc... (collections)
- We store JSON objects (documents) in MongoDB
- The data is also returned as JSON objects from the DB

— Setup

- 3 parts (2 to be installed):
- MongoDB: de database
- MongoDB Compass: a client application to visualize the DB data. This one is optional and there are alternatives (e.g. CLI).
- The Node applicatio,: communicates directly with the DB (CRUD).



Visualisatie

Collection	
PK	UniqueID
	Row 1
	Row 2
	Row 3

— Install MongoDB Community Edition

- All platforms: https://www.mongodb.com/try/download/community-kubernetes-operator
- Guide Windows: https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-windows/
- Guide Mac: https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/
- Guide Linux: https://www.mongodb.com/docs/manual/administration/install-on-linux/
- Mac option 2: with HomeBrew

```
milan@homebrew~brew tap mongodb/brew
milan@homebrew~brew install mongodb-community@4.4
```

- Mac run service:
- `mongod --config /usr/local/etc/mongod.conf`

— MongoDB Docker & Cloud

- Docker option: https://hub.docker.com/_/mongo

```
(base) ~ ➤ docker run -p 27017:27017 mongo
{"t":{"$date":"2022-05-01T10:49:58.325+00:00"},"s":"I",
"c":"NETWORK",  "id":4915701,
"ctx":"thread1","msg":"Initialized wire
specification"...
```

- Guide Docker: https://www.mongodb.com/docs/manual/tutorial/install-mongodb-community-with-docker/
- MongoDB Atlas (free MongoDB in thee Cloud): Link
- Also see: https://www.mongodb.com/docs/manual/installation/

Install MongoDB Client and connect in Compass

- Go to https://www.mongodb.com/products/compass
- Download and install the right version of Compass
- Start mongodb service
- Connect with DB: mongodb://localhost
- There are already three db's for the internal MongoDB setup: admin, config and local.
- Create a new DB: CREATE COLLECTION

Run mongod in Windows (Obsolete if run as service was checked at install)

1. Create database directory. Create the data directory where MongoDB stores data. MongoDB's default data directory path is the absolute path \data\db on the drive from which you start MongoDB.

From the **Command Interpreter**, create the data directories:

```
cd C:\
md "\data\db"
```

2. Start your MongoDB database. To start MongoDB, run exe.

```
"C:\Program Files\MongoDB\Server\5.0\bin\mongod.exe" --dbpath="c:\data\db"
```

The --dbpath option points to your database directory.

If the MongoDB database server is running correctly, the **Command Interpreter** displays:

```
[initandlisten] waiting for connections
```

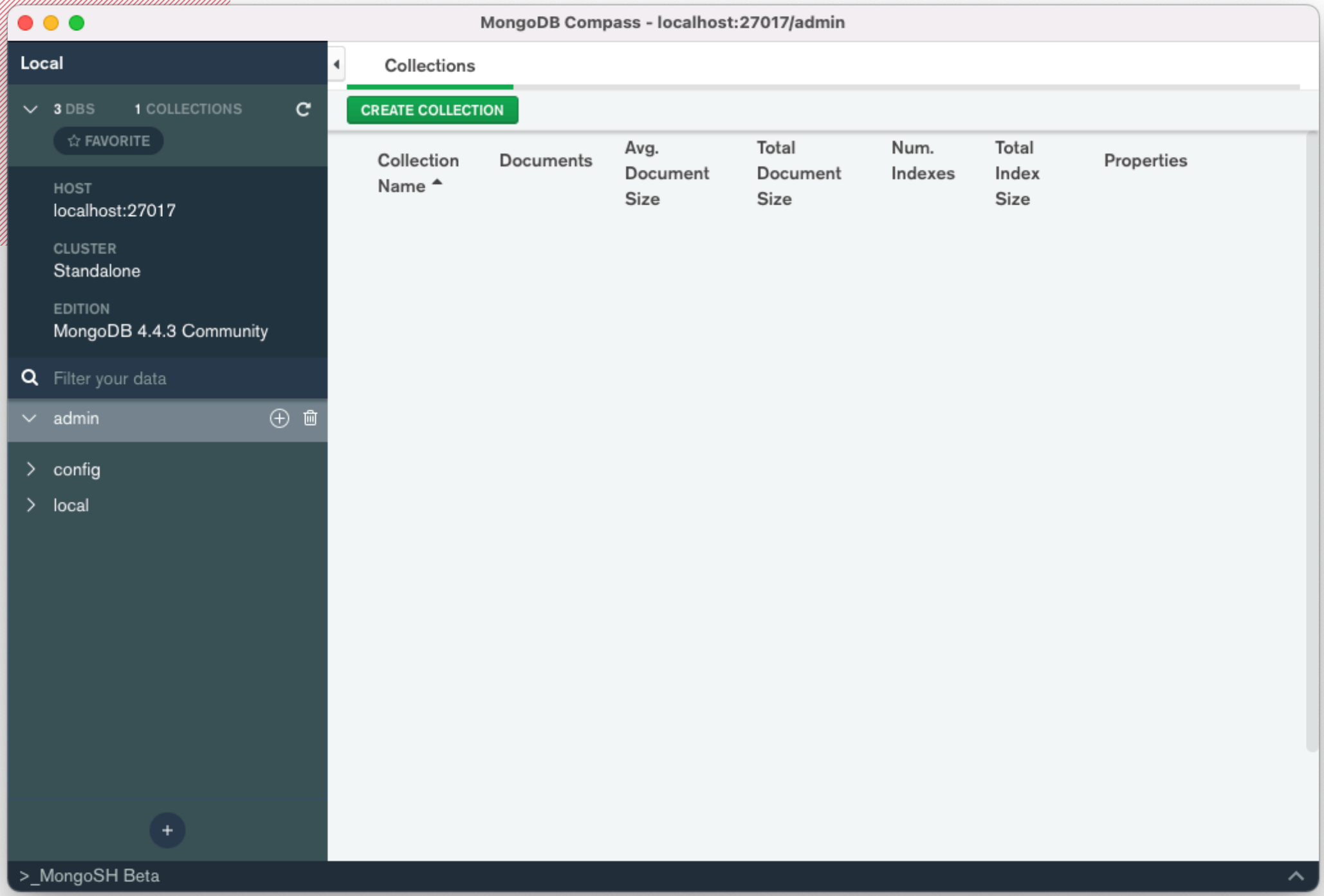
IMPORTANT
Depending on the

Windows Defender Firewall

settings on your Windows host, Windows may display a **Security Alert** dialog box about blocking "some features" of C:\Program Files\MongoDB\Server\5.0\bin\mongod.exe from communicating on networks. To remedy this issue:

- Click **Private Networks, such as my home or work network**.
- Click **Allow access**.

To learn more about security and MongoDB, see the Security Documentation.



— Node app with MongoDB integration

- We create a new project: mongo-demo
- We install mongoose (API for mongoDB) https://mongoosejs.com/

```
milan@nodeVb〽cd mongodb-demo
milan@mongodb-demo〽npm i mongoose
```

- We add an index.js
- Connect with MongoDB
- `require mongoose`
- Add connection string

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

- In production: different connection string (in config files and not hardcoded!)

— Connect with the DB

```
mongoose.connect('mongodb://localhost/playground');
```

- If the 'playground' collection does not exist yet it will be created at the first connection with the MongoDB
- The connect method returns a promise
- This means we can consume it with .then and .catch
- index.js

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/playground')
    .then(()=> console.log('Connected with MongoDB'))
    .catch(err => console.error('Can not connect to
the DB ...', err));
```

- terminal:

```
milan@mongodb-demo nodemon index.js
...
Connected with MongoDB
```

— Schemas

- Schemas are used to shape the documents in a MongoDB collection
- Schema is typical for Mongoose (not in MongoDB)
- Collection ≈ Tabel or schema in a relational DB
- Document ≈ Row or record in a relational DB
- A document is a container of key - value pairs
  - `_id:"Milans-MacBook-Pro.local-1616612097741"`
    `hostname:"Milans-MacBook-Pro.local"`
    `startTime:2021-03-24T18:54:57.000+00:00`
    `startTimeLocal:"Wed Mar 24 19:54:57.741"`
    `cmdLine:Object`
  - `pid:53360`
    `buildinfo:Object`

— Schema Types

- Types you can use inside a schema:

- String
- Number
- Date
- Buffer (binaire data storen)
- Boolean
- ObjectID (unieke identifiers)
- Array

— Create Schema

- We create a Course schema

```
const courseSchema = new mongoose.Schema({
    name: String,
    author: String,
    tags: [String],
    date: { type: Date, default: Date.now },
    isPublished: Boolean
})
```

- Of this Schema now we need to create a model
- We need a Course Class. We compile our schema to a model.

```
const Course = mongoose.model('Course', courseSchema);
```

- mongoose.model takes two arguments: singular of the collection name and the shape of our document

— Create Object

- We create an object based on our new class

```
const course = new Course({
    name: 'Node.js Course',
    author: 'M. Dima',
    tags: ['node','backend'],
    isPublished: true
});
```

- NoSQL DBs have no fixed structure. We can store an array of strings in there (tags)
- This is not possible with relational DBs

— Recap

- 1. We first create a schema (template)
- 2. This schema will be compiled to a model that will provide us with a Class
- 3. After this we can create objects (instances) based on this Class
- 4. That object will automatically be mapped to a MongoDB document

— Save a document (CRUD)

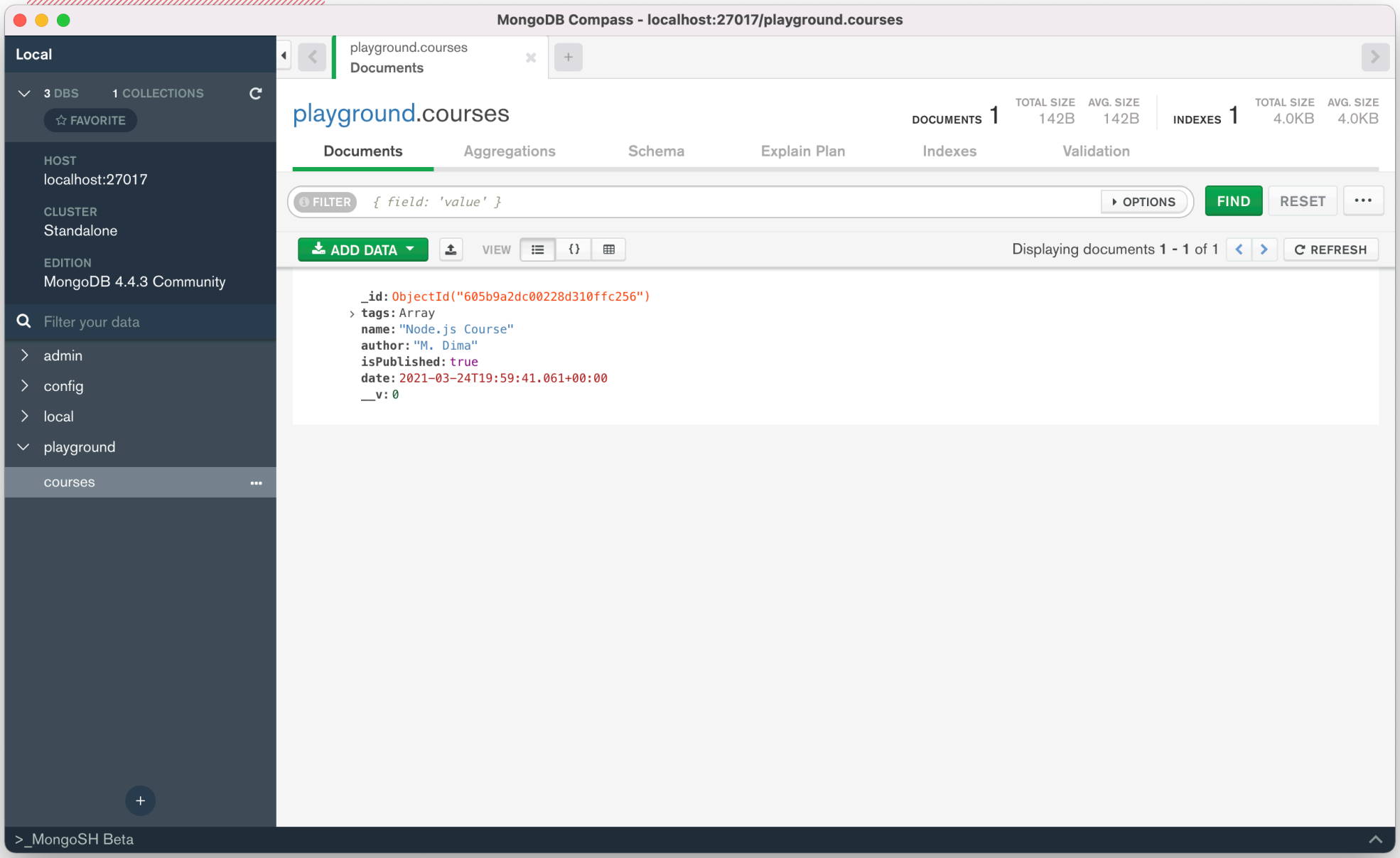
- Our course object comes with a method save()
- Is an asynchronous operation and will take some time
- This method returns a promise
- When we save a document MongoDB will assign it an unique ObjectID
- To be able to *await* our promise we need to put it inside an *async* function scope

Save a document (CRUD)

```
async function createCourse() {
  const course = new Course({
    name: "Node.js Course",
    author: "M. Dima",
    tags: ["node", "backend"],
    isPublished: true,
  });
  const result = await course.save();
  console.log(result);
}
```

- terminal

```
...
Connected with MongoDB
{
  tags: [ 'node', 'backend' ],
  _id: 605b9a2dc00228d310ffc256,
  name: 'Node.js Course',
  author: 'M. Dima',
  isPublished: true,
  date: 2021-03-24T19:59:41.061Z,
  __v: 0
}
```



Figuur 1: screenshot van MongoDB Compass applicatie waar de DB playground aangemaakt werd met collectie courses en een document voor onze Node.JS cursus

— index.js

```
const mongoose = require("mongoose");
mongoose
  .connect("mongodb://localhost/playground")
  .then(() => console.log("Verbonden met de MongoDB"))
  .catch((err) => console.error("Kan niet verbinden met DB ...",
err));
const courseSchema = new mongoose.Schema({
  name: String,
  author: String,
  tags: [String],
  date: { type: Date, default: Date.now },
  isPublished: Boolean,
});
const Course = mongoose.model("Course", courseSchema);
async function createCourse() {
  const course = new Course({
    name: "Node.js Course",
    author: "M. Dima",
    tags: ["node", "backend"],
    isPublished: true,
  });
  const result = await course.save();
  console.log(result);
}
createCourse();
```

— Get Documents (CRUD)

```
async function getCourses(){
    const courses = await Course.find();
    console.log(courses);
}
getCourses();
```

- terminal

```
...
Connected with MongoDB
[
  {
    tags: [ 'node', 'backend' ],
    _id: 605b9a2dc00228d310ffc256,
    name: 'Node.js Course',
    author: 'M. Dima',
    isPublished: true,
    date: 2021-03-24T19:59:41.061Z,
    __v: 0
  },
  {
    tags: [ 'angular', 'frontend' ],
    _id: 605b9bed4dfe11d32a7548f1,
    name: 'Angular Course',
    author: 'M. Dima',
    isPublished: true,
    date: 2021-03-24T20:07:09.606Z,
    __v: 0
  }
]
```

— Get one document (CRUD)

- Filter all M. Dima's courses and published

```
.find({author: 'M. Dima', isPublished: true})
```

- we can customize our Query even more
- limit: limit the number of results
- sorting: *property* + 1 = ascending, -1 = descending
- Select: show only selected properties

```
async function getCourses(){
    const courses = await Course
    .find({author: 'M. Dima', isPublished: true})
    .limit(10)
    .sort({name: 1})
    .select({name: 1, tags: 1});
    console.log(courses);
}
getCourses();
```

- We run our program:

```
milan@mongodb-demo~node index.js
...
Connected with MongoDB
[
  {
    tags: [ 'node', 'frontend' ],
    _id: 605b9bed4dfe11d32a7548f1,
    name: 'Angular Course'
  },
  {
    tags: [ 'node', 'backend' ],
    _id: 605b9a2dc00228d310ffc256,
    name: 'Node.js Course'
  }
```

— Complex filtering

- Comparison Operators

  - eq = equal
  - ne = not equal
  - gt = greater than
  - gte = greater than or equal
  - lt = less than
  - lte = less than or equal
  - in
  - nin = not in

— Some examples

- Suppose our courses have a price property
- all courses with a price of 10€

```
const courses = await Course
.find({ price: 10 }) ...
```

- all courses with a price of at least 10€

```
.find({ price: { $gte: 10 } })
```

- all courses with a price between 10€ and 20€

```
.find({ price: { $gte: 10, $lte: 20 } })
```

- all courses with a price of 10€, 15€ or 20€

```
.find({ price: { $in: [10, 15, 20] } })
```

— Logical Query operators

- Get all of M. Dima's courses AND/OR published = true
- OR

```
    .find()
    .or([{ author: 'M. Dima'}, { isPublished:
true }])
```

- AND

```
    .find()
    .and([{ author: 'M. Dima'}, { isPublished:
true }])
```

— Regular Expressions (Regex)

- Also see https://regexone.com/ and
- https://www.rexegg.com/regex-quickstart.html
- All courses where the author starts with 'M.'

```
.find({author: /^M./})
```

- Ends in 'Dima'

```
.find({author: /Dima$/})
```

- Same but case insensitive

```
.find({author: /Dima$/i})
```

- All courses where the name of the author contains "Dima" (start, end or middle)

```
.find({author: /.*Dima.*/})
.find({author: /.*Dima.*/i}) //case insensitive
```

Count Documents

- `count()` instead of `select()` returns the number of documents and not the documents

```
async function getCourses(){
    const courses = await Course
   .find({author: 'M. Dima', isPublished: true})
    .limit(10)
    .sort({name: 1})
    .count();
    console.log(courses);
}
```

- terminal

```
milan@mongodb-demo node index.js
...
Connected with MongoDB
2
```

— Pagination

- we use the `skip()` method in combination with the `limit()` method to setup pagination

```
async function getCourses(){
    const pageNumber = 2;
    const pageSize = 10;
    // real world: /api/courses?pageNumber=2&pageSize=10
    const courses = await Course
    .find({author: 'M. Dima', isPublished: true})
    .skip((pageNumber-1)* pageSize)
    .limit(pageSize)
    .sort({name: 1})
    .select({name: 1, tags: 1});
    console.log(courses);
}
```

Good practice run() function

```
const Course = mongoose.model('Course', courseSchema);

async function getCourses() {
  return await Course
  .find({ isPublished: true, tags: 'backend' })
  .sort({ name: 1 })
  .select({ name: 1, author: 1 });
}

async function run() {
  const courses = await getCourses();
  console.log(courses);
}

run();
```

— Update Documents - Query 1st (CRUD)

```
async function updateCourse(id){
const course = await Course.findById(id);
if(!course) return;
/*
course.isPublished = true;
course.author = 'Another author'; */
course.set({
    isPublished: true,
    author: 'Another author'
});
const result = await course.save();
console.log(result);
}
updateCourse('605b9bed4dfe11d32a7548f1');
```

— Update Documents - Update 1st (CRUD)

- MongoDB update operators: https://docs.mongodb.com/manual/reference/operator/update/

```
async function updateCourse2(id) {
  const result = await Course.update(
    { _id: id },
    {
      $set: {
        author: "M. Dima",
        isPublished: false
      }
    }
  );
  console.log(result);
}
updateCourse2("605b9bed4dfe11d32a7548f1");
```

- Terminal:

```
{ n: 1, nModified: 1, ok: 1 }
```

- If we want to return the document:

```
async function updateCourse2(id) {
  const course = await Course.findByIdAndUpdate(id, {
      $set: {
        author: "Jack",
        isPublished: true,
      }
  });
  console.log(course);
}
updateCourse2("605b9bed4dfe11d32a7548f1");
```

- returns the original document
- If you want to return the updated document:

```
  const course = await Course.findByIdAndUpdate(id, {
      $set: {
        author: "Jack",
        isPublished: true
      }
  }, { new:true });
```

— Remove one document (CRUD)

```
async function removeCourse(id) {
    const result = await Course.deleteOne({ _id: id });
    console.log(result);
  }
removeCourse("605b9bed4dfe11d32a7548f1");
```

- terminal

```
Connected with MongoDB
{ n: 1, ok: 1, deletedCount: 1 }
```

- Remove several documents:

```
async function removeCourse(id) {
    const result = await
Course.deleteMany({ isPublished: true });
    console.log(result);
  }
removeCourse("605b9bed4dfe11d32a7548f1");
//findByAndRemove returns the deleted document
```

— Lab

- Class Code: https://github.com/MilanVives/NodeLes7

➡ Lab:

- Assignment link: see Toledo