香港中文大學
The Chinese University of Hong Kong

CSCI2720 2023-24 Term 1: Building Web Applications

Lab 8: MongoDB (by Mongoose)

Dr Colin Tsang

Outline

- Revisit lab07
- Setting up MongoDB
- Connect via Mongoose
- Mongoose Schema and model
- TASK 1: CRUD – Create
- TASK 2: CRUD – Read
- TASK 3: CRUD – Update
- TASK 4: CRUD - Delete
- Appendix:
  - Setting up MongoDB with Atlas cloud

Revisit lab07

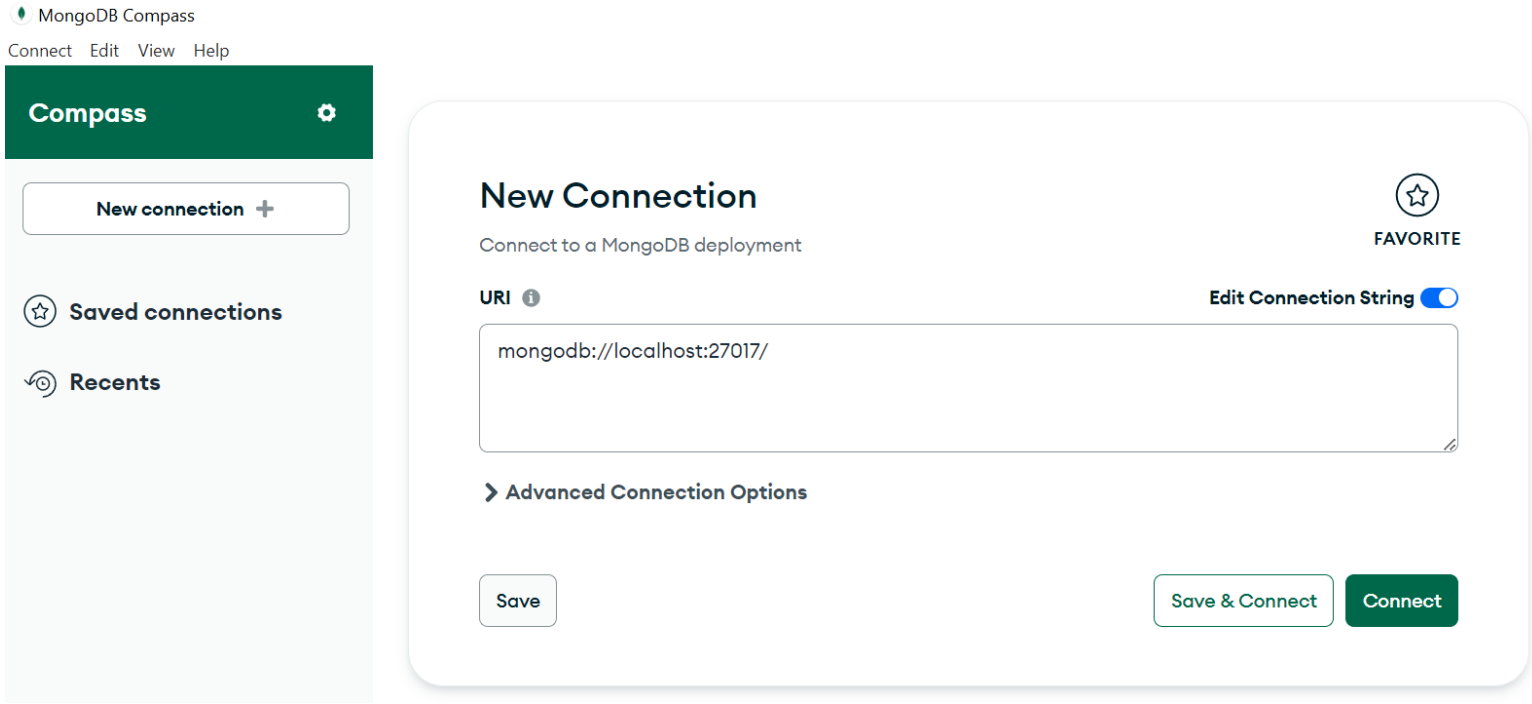
- Download *lab08_server.js* from Blackboard
  - Put this JS file into a new folder
- Follow the instructions from lab07, set up a web server on http://localhost:3000
  - Navigate to your directory: **`npm init`**
    - Accept default answers for all questions with **`Enter`**
  - Install Express: **`npm install express`**
- Also install the mongoose for this lab: **`npm install mongoose`**
- You can install two things together: **`npm install express mongoose`**

Setting up MongoDB

- You can set up a MongoDB server on your computer
  - Download the installer:
    - https://www.mongodb.com/try/download/community
    - Select your platform (MacOS, Windows, or Linux)
    - Install the *Complete* version
    - Install MongoDB as a service *(run service as Network Service user)*
    - Install the *MongoDB compass*

Setting up MongoDB

- After installation, you should see a *MongoDB compass*:



- Click the *connect button* to start

Connect via Mongoose

- The *server URL* is:
  - `mongodb://127.0.0.1:27017/myDatabase`
- The number 27017 is the default value. You can use another number.
- **myDatabase** is the database name decided by you.

Connect via Mongoose

- Try to understand the code in *lab08_server.js*:
- This is a standard way to use Mongoose:

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/myDatabase'); // put your own database link here
```

- Checking the connection to MongoDB:

```
const db = mongoose.connection;
// Upon connection failure
db.on('error', console.error.bind(console, 'Connection error:'));
// Upon opening the database successfully
db.once('open', function () {
  console.log("Connection is open...");

  // your code here

})
```

Connect via Mongoose

- You can try to access http://localhost:3000 now, like what you have done in lab07.

- If the connection is successful:
  - You should be able to see the *Hello World* message in the browser, as we handle all requests with the same message.
  - Also, there should be a *connection is open…* message in the **terminal** or **cmd**.

Mongoose Schema and Model

- In Mongoose, we use *Schema* and *Model* to perform **CRUD** on MongoDB:
  - Create, read, update, and delete
- We will do **CURD** one by one in this lab.
- Now, we want to set up a Mongoose *Schema* and *Model* for events with the following data:
  - Event ID
  - Event location
  - Quota of the event

- *We use fake data in this lab, you should have real-world data in your project*

Mongoose Schema and Model

- Create the *Schema* and *Model* for event ID and location:

```
const EventSchema = mongoose.Schema({
  eventID: {
    type: Number,
    required: [true, "Name is required"],
  },
  location: {
    type: String,
    required: true,
  },
});

const Event = mongoose.model("Event", EventSchema);
```

Mongoose Schema and Model

- For the quota, you may want to add a validation function:

```
quota: {
  type: Number,
  validate: {
    validator: function (value) {
      return value > 0;
    },
    message: () => "Please enter a valid quota",
  },
```

TASK 1: CRUD - Create

- We can create a new *document* on MongoDB very easily using the *Schema* and *Model*:

```
//Creating a new event
 let newEvent = new Event({
   eventID: 123,
   location: "SHB130",
   quota: 9999,
 });
```

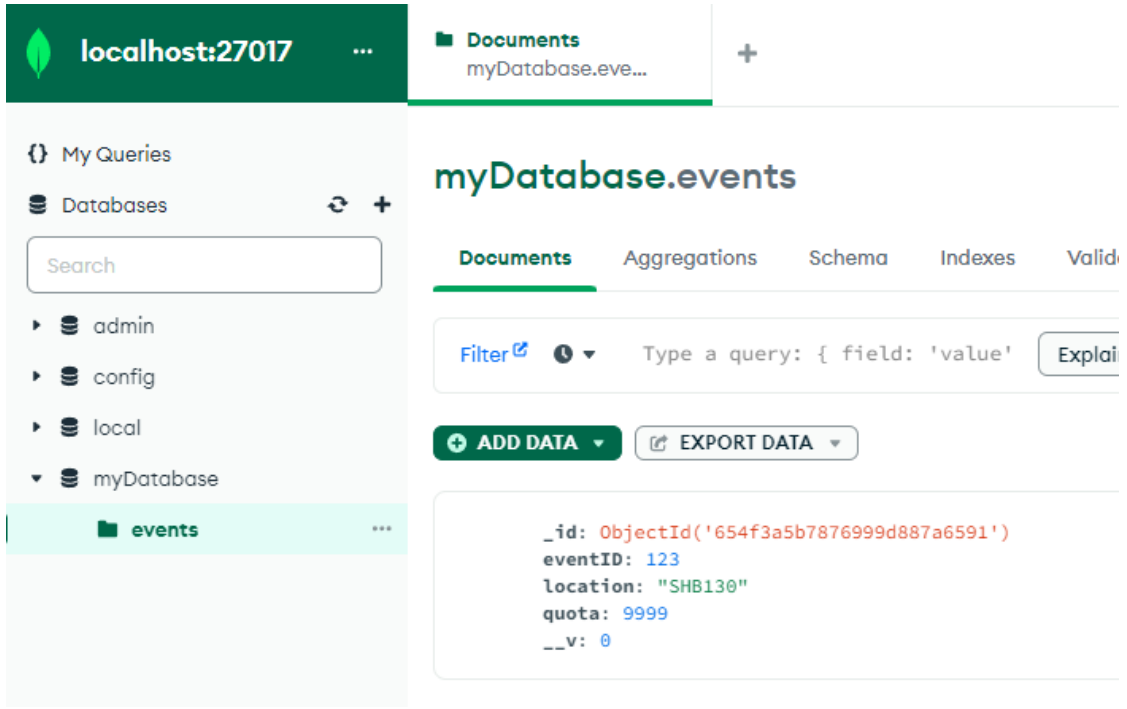
TASK 1: CRUD - Create

- Next, we want to save this newly created *document* to our MongoDB:

```
//Saving this new event to database
newEvent
  .save()
  .then(() => {
    console.log("a new event created successfully");
  })
  .catch((error) => {
    console.log("failed to save new event");
  });
```

TASK 1: CRUD - Create

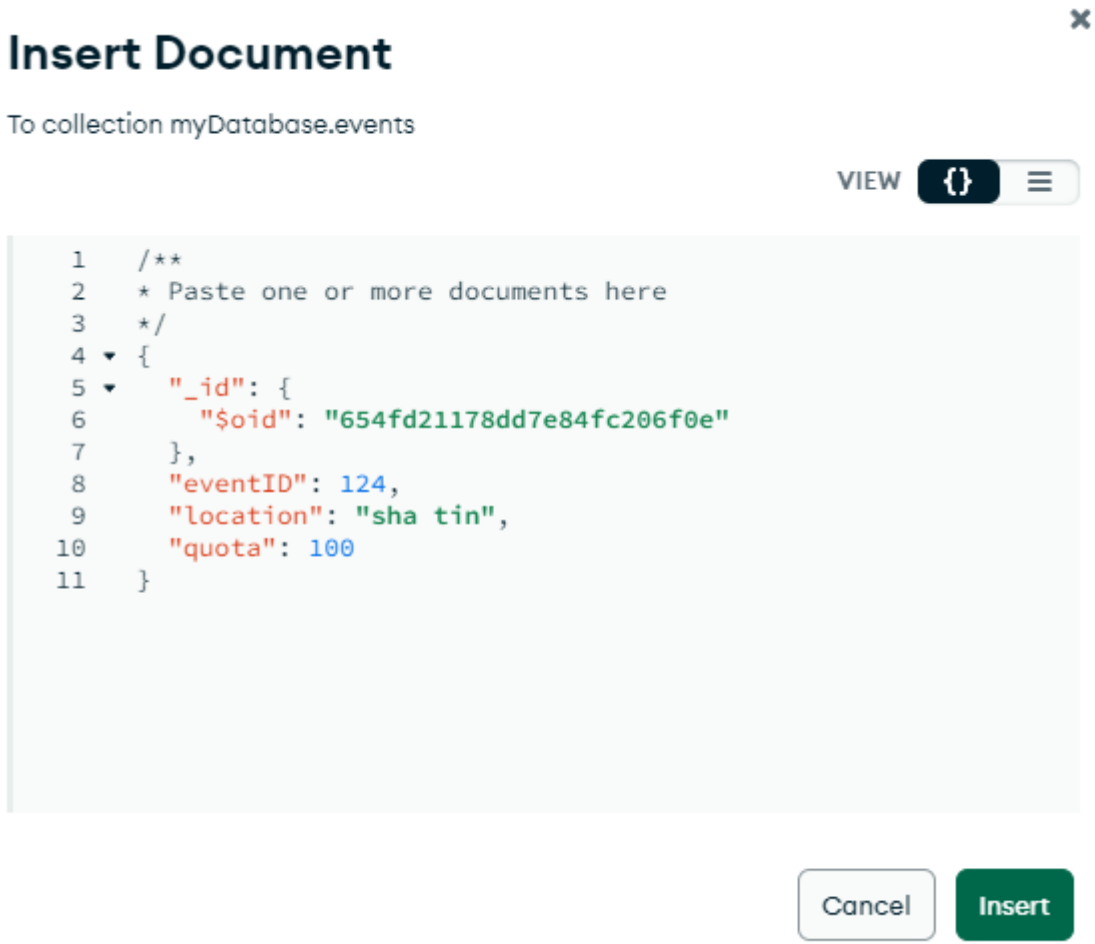
- Run the JS file in your **terminal** or **cmd**.
- You will see the message *"a new event created successfully"*
- Go to your *MongoDB compass*, click *refresh*. You will see the new *document* is added into your database
- The *refresh* button in the *MongoDB compass* may fail. You may need to restart the whole *MongoDB compass* in this case……



TASK 1: CRUD - Create

- You can directly add *document* to MongoDB
  - Click *add data*, use *Insert document*

- Now, you have 2 data in your database.



TASK 2: CRUD - Read

- In Mongoose, you can use the **find()** method to access all the saved data

```
// Read all data
Event.find({})
.then((data) => {
  console.log(data);
})
.catch((err) => {
  console.log("failed to read");
});
```

```
C:\Users\user\Desktop\lab8_code> node server.js
Connection is open...
[
  {
    _id: new ObjectId('654f3a5b7876999d887a6591'),
    eventID: 123,
    location: 'SHB130',
    quota: 9999,
    __v: 0
  },
  {
    _id: new ObjectId('654fd21178dd7e84fc206f0e'),
    eventID: 124,
    location: 'sha tin',
    quota: 100
  }
]
```

TASK 2: CRUD - Read

- You can also perform searching with **find()**

```
// Search for quota >= 500
 Event.find({ quota: { $gte: 500 } })
 .then((data) => console.log("the event with quota more than 500:", data))
 .catch((error) => console.log(error));
```

```
C:\Users\user\Desktop\lab8_code> node server.js
Connection is open...
the event with quota more than 500: [
  {
    _id: new ObjectId('654f3a5b7876999d887a6591'),
    eventID: 123,
    location: 'SHB130',
    quota: 9999,
    __v: 0
  }
]
```

TASK 3: CRUD - Update

- In Mongoose, we can use the **findOneAndUpdate()** method to perform updating.
- Suppose we want to update the location if the quota is too large

```
// update the location if quota >= 500
Event.findOneAndUpdate(
    { quota: { $gte:500 } },
    { location:"Large Conference Room"},
    { new: true},
  )
  .then((data) => {console.log('the updated data is:', data)})
  .catch((error) => console.log(error));
```

```
C:\Users\user\Desktop\lab8_code> node server.js
Connection is open...
the updated data is: {
  _id: new ObjectId('654f3a5b7876999d887a6591'),
  eventID: 123,
  location: 'Large Conference Room',
  quota: 9999,
  __v: 0
}
```

TASK 4: CRUD - Delete

- In Mongoose, delete can be done by **findOneAndDelete()**
- The syntax is similar to **findOneAndUpdate()**

```
// delete the event if quota >= 500
Event.findOneAndDelete(
  { quota: { $gte:500 } }
)
.then((data) => {console.log('the deleted data is:', data)})
.catch((error) => console.log(error));
```

```
C:\Users\user\Desktop\lab8_code> node server.js
Connection is open...
the deleted data is: {
  _id: new ObjectId('654f3a5b7876999d887a6591'),
  eventID: 123,
  location: 'Large Conference Room',
  quota: 9999,
  __v: 0
}
```

Further readings

- There are many other ways to perform CRUD on MongoDB. Try to explore more in the project.
- You may want to read:
  - https://www.makeuseof.com/how-to-use-mongoose-in-express-apps/
- MongoDB Sell could be useful:
  - https://www.mongodb.com/docs/mongodb-shell/

Submission

- No submission is needed for labs
- What you have done could be useful for your further exploration or the upcoming assignment/project
- Please keep your own code safely

Appendix:
Setting up MongoDB with Atlas cloud

- You can try the Atlas Cloud instead of installing MongoDB locally
- Follow the instructions here:
  - https://www.mongodb.com/docs/atlas/tutorial/deploy-free-tier-cluster/
  - An account is needed on Atlas
  - To connect to the cloud DB, you can add 0.0.0.0 to the IP access list, which mean ANYWHERE
    - Well, it is not secure…….