Web Programming

Tutorial 10

To begin this tutorial, please create a Node project or use an existing one. When you finish, zip all your source codes (excluding the `node_modules` folder) to submit to this tutorial's submission box. The zip file's name should follow this format: `tclass_sid.zip` where `tclass` is your tutorial class name (e.g. `1C21C, 2C21C`, etc.) and `sid` is your student's ID (e.g. `2101040015`).

Activity 1 – MongoDB command-line

In this exercise, we aim to create the database for our dictionary web application. Start the mongodb command-line then:

- Name all databases that your server has.

- Switch to use database with name: eng-dict (created automatically if db name not exist)

- Name all collections that this db has.

- Add a new word into words collection: {word: 'dog', definition: 'friend'}

- Add a new word into words collection: {word: 'cat, definition: 'boss'}

    o Query all words to check if success inserted.

- Add some more words (at least 5)

- Query for definition of the word: 'dog'

- Update definition of the word 'dog' from 'friend' into 'woof woof'

    o Query all words to check if success inserted.

- Set all words to have definition: 'empty: to-update'

- Delete the word 'dog' from the words collection

    o Query all words to check if success inserted.

- Delete all words from collection words.

    o Query all words to check if success inserted.

- Delete the collection words from database

    o Name all collection that this **db** has to check if success.

Activity 2 – MongoDB Exercises

Try the following test to check how good you are with MongoDB:

MongoDB Exercises (w3schools.com)

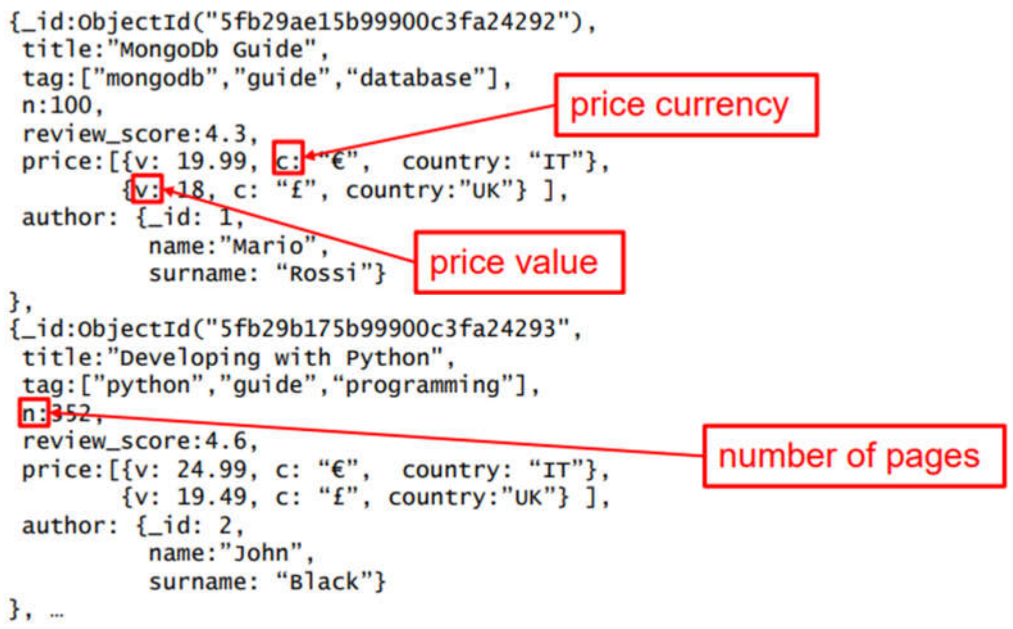
→ **Delivery**: `activity_2.txt` with content following this template:

```
Number of finished exercises: …

Time spent: … (minutes)

Number of exercises which need Googling: …
```

Activity 3 – Query Exercises

In this exercise, you will work with a collection of books and perform various queries based on specific criteria. Please complete all the required tasks and submit your answers in a file named `activity_3.txt`.

Given the following collection of book



```
{_id:ObjectId("5fb29ae15b99900c3fa24292"),
 title:"MongoDb Guide",
 tag:["mongodb","guide",“database"],
 n:100,
 review_score:4.3,
 price:[{v: 19.99, c: “€",  country: “IT"},
        {v: 18, c: “£", country:"UK"} ],
 author: {_id: 1,
          name:"Mario",
          surname: “Rossi"}
},
{_id:ObjectId("5fb29b175b99900c3fa24293",
 title:"Developing with Python",
 tag:["python","guide",“programming"],
 n:352,
 review_score:4.6,
 price:[{v: 24.99, c: “€",  country: “IT"},
        {v: 19.49, c: “£", country:"UK"} ],
 author: {_id: 2,
          name:"John",
          surname: “Black"}
}, …
```

Requirements:

1. Find all the books with **a number of pages** greater than 250
2. Find all the books **authored** by Mario Rossi
3. Find all the books with a **price** less than 20 € for **Italy** (IT)
4. Increase the **review score** of 0.2 points for all the books with the tag "database"
5. Insert the **tag** "NoSQL" for all the books with **tag** "mongodb"
6. Insert the **publisher** for all the documents **authored** by Mario Rossi with the default value {'name': 'Polito', city:'Turin'}
7. Find the maximum, the minum and the average **price** of all the books with **tag** "database"
8. Compute the number of books authored by Mario Rossi

Activity 4 – Student and Course Management System with MongoDB

In this exercise, you will build a basic "*Student and Course Management System*" using the `MongoDB Node.js driver (mongodb library).` This system will allow you to manage students and their courses, performing basic database operations such as `inserting`, `querying`, `updating`, and `deleting` documents.

Requirements:

1. **Project Structure:**
   - Organize your exercise into 2 files:
     - `dbOperations.js:` Contains functions for database operations (connecting, inserting data, querying, updating, deleting).
     - `index.js:` Contains the main control part where you call functions from `dbOperations.js` to execute the program.
2. **Detailed Tasks:**

   Step 1. Setup and Connect to the Database

   - Create a `Node.js` script that connects to a MongoDB server.
   - The script should print "*`Connected to the database`*" when connected successfully.
   - Handle connection errors and display an error message if something goes wrong.

Step 2. Create Collections and Insert Documents

- In a MongoDB database called school, create two **collections**: `students` and `courses`.
- For students collection, insert the following documents:

```
{ "name": "John", "age": 22, "major": "Math" }

{ "name": "Anna", "age": 20, "major": "Computer Science" }

{ "name": "Mike", "age": 21, "major": "Physics" }
```

- For courses collection, insert the following documents:

```
{ "course_name": "Database Systems", "credit_hours": 4 }

{ "course_name": "Operating Systems", "credit_hours": 3 }

{ "course_name": "Artificial Intelligence", "credit_hours": 4 }
```

Step 3. Query Documents

- Write a function tov `query` and `print all students` from the `students collection`.
- Write another function to query and print all courses from the courses collection.

Step 4. Query a Specific Document

- Query for a student with the name "`Anna`" from the students collection using the `findOne()` method.
- Query for a course with the name "Database Systems" from the courses collection.

Step 5. Update a Document

- Write a function to `update` the major of the student "`John`" from "`Math`" to "`Statistics`".

- Use the upsert option to add a new student named "Tom" with any data if he doesn't already exist in the database.

Step 6. Delete Documents

- Write a function to delete one student (e.g., delete "Mike") from the students collection.
- Write another function to delete all courses from the courses collection.

Step 7. Advanced Querying

- Write a function to query all students older than 20 years from the students collection.
- Write another function to query all students majoring in "*Computer Science*".

Step 8. Sorting and Limiting

- Query all students, sort them by `age (ascending),` and limit the result to only 2 documents.

Note:

- Ensure your project structure is correct, and both dbOperations.js and index.js files are properly implemented.
- Submit your project folder containing both files.

Activity 5 – Student and Course Management System API

1. **Create the API**
   - Create a file named **`server.js`** in act4's project
2. **Connect to MongoDB**
   - In index.js, write a function to connect to your MongoDB database. Use the following connection URL and database name:
     - **URL**: `mongodb://localhost:27017`
     - **Database Name**: `school`

3. **Create API Endpoints**

- **Get All Students**

  a. Create an endpoint at /students to return a list of all students.

  b. The result should look like this:

```
[
    {
        "_id": "student_id_here",
        "name": "Student Name",
        "age": 20,
        "major": "Major"
    },
    ...
]
```

- **Add a Student**
  - Create an endpoint at **`/students`** to add a new student.
  - Accept the following JSON format in the request body:

```
{
    "name": "Student Name",
    "age": 20,
    "major": "Major"
}
```

  - Return a success message like this:

```
{
    "message": "Student added successfully",
    "studentId": "student_id_here"
}
```

- **Add a Course**
  - Create an endpoint at /courses to add a new course.
  - Accept the following JSON format in the request body:

```
{
    "course_name": "Course Name",
    "credit_hours": 3
}
```

Return a success message like this:

```
{
    "message": "Course added successfully",
    "courseId": "course_id_here"
}
```