

Task 1: CRUD Operations with MongoDB and JavaScript

Objective:

Connect to a MongoDB database using JavaScript. Create a database and a collection. Insert documents from a JSON file into the MongoDB collection. Perform CRUD operations (Create, Read, Update, Delete) on the documents.

Prerequisites:

- Basic understanding of JavaScript.
- Node.js installation.
- MongoDB installed and running.

Concepts:

1. Setup and Connect to MongoDB:

- Install MongoDB and start the MongoDB server.
- Create a new project directory and initialize a Node.js project.

Example:

Command Line:

```
mkdir movie-crud
cd movie-crud
npm init -y
npm install mongodb
```

2. Create the Database and Collection, and Insert Documents:

Concept:

- Use the MongoClient from the mongodb package to connect to a MongoDB server.
- The connection URI specifies the address of the MongoDB server.
- useNewUrlParser and useUnifiedTopology are options to ensure compatibility with the latest MongoDB features and avoid deprecation warnings.
- A database is a container for collections.
- A collection is a grouping of MongoDB documents.
- You can create a database and collection implicitly by referring to them in your code.

Example:

JavaScript:

```

const { MongoClient } = require('mongodb');
const fs = require('fs');
const path = require('path');

const uri = 'mongodb://localhost:27017'; // Replace with your MongoDB URI
const client = new MongoClient(uri, { useNewUrlParser: true,
useUnifiedTopology: true });

async function run() {
  try {
    await client.connect();
    console.log("Connected successfully to MongoDB server");

    const database = client.db('movieDB');
    const collection = database.collection('movies');

    // Insert documents
    const movies = JSON.parse(fs.readFileSync(path.join(__dirname,
'movies.json'), 'utf-8'));
    const insertResult = await collection.insertMany(movies);
    console.log(`${insertResult.insertedCount} documents were inserted`);
  } finally {
    await client.close();
  }
}

run().catch(console.dir);

```

MongoDB Compass:

- Open MongoDB Compass.
- Connect to your MongoDB instance.
- Select your database and collection.
- Click on the Insert Document button.
- Copy and paste the JSON documents into the input field and click Insert .

3. CRUD Operations:

Create: Add a New Movie Document:

Concept:

- Use `insertOne` to add a single document to a collection.

Example:

JavaScript:

```

async function addMovie() {
  const newMovie = {
    "title": "New Movie",
    "release_date": "2024-01-01",
    "genre": ["Drama"],
    "vote_average": 8.5
  };
  const result = await collection.insertOne(newMovie);
  console.log(`New movie created with the following id: ${result.insertedId}`);
}

```

MongoDB Compass:

- In MongoDB Compass, go to the `Insert Document` button.
- Enter the new movie document in JSON format and click `Insert`.

Read: Retrieve All Movies:

Concept:

- Use `find` to retrieve documents from a collection.
- Use `toArray` to convert the cursor returned by `find` into an array of documents.

Example:

JavaScript:

```

async function getMovies() {
  const movies = await collection.find({}).toArray();
  console.log(movies);
}

```

MongoDB Compass:

- In MongoDB Compass, go to the `Find` tab.
- Enter an empty filter `{}` to retrieve all documents and click `Find`.

Update: Update the Vote Average of a Specific Movie:

Concept:

- Use `updateOne` to modify a single document in a collection.
- Use a filter to specify which document to update and an update document to specify the changes.

Example:

JavaScript:

```

async function updateMovie() {
  const filter = { "title": "Midnight" };
  const updateDoc = { $set: { "vote_average": 9.0 } };
  const result = await collection.updateOne(filter, updateDoc);
  console.log(`${result.modifiedCount} document(s) was/were updated`);
}

```

MongoDB Compass:

- In MongoDB Compass, go to the `Find` tab.
- Enter the filter `{"title": "Midnight"}` to find the specific document.
- Click on the `Update` button and enter the update document `{$set: {"vote_average": 9.0}}`.
- Click `Update`.

Delete: Remove a Movie Document:

Concept:

- Use `deleteOne` to remove a single document from a collection.
- Use a filter to specify which document to delete.

Example:

JavaScript:

```
async function deleteMovie() {
  const filter = { "title": "New Movie" };
  const result = await collection.deleteOne(filter);
  console.log(`${result.deletedCount} document(s) was/were deleted`);
}
```

MongoDB Compass:

- In MongoDB Compass, go to the `Find` tab.
- Enter the filter `{"title": "New Movie"}` to find the specific document.
- Click on the `Delete` button to remove the document.

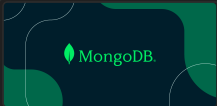
Instructions:


Perform the following tasks:

1. Connect to a MongoDB database and create a `movieDB` database with a `movies` collection.
2. Insert documents from the `movies.json` file provided in the git repository into the `movies` collection.
3. Add a new movie document to the collection using the `addMovie` function.
4. Retrieve all movies from the collection using the `getMovies` function.
5. Update the vote average of a specific movie using the `updateMovie` function.
6. Remove a movie document from the collection using the `deleteMovie` function.



Resources:

- **Documentation:**

-  **MongoDB CRUD Operations**
CRUD operations create, read, update, and delete — .leafygreen-ui-lh88kx{-webkit-...
<https://docs.mongodb.com/manual/crud/>

- 
MongoDB Node Driver
 Welcome to the documentation site for the official MongoDB Node.js driver. — You c...
<https://docs.mongodb.com/drivers/node/>

• Blogs:

- 
Connect to a MongoDB Database Using Node.js | MongoDB Blog
 Node.js and MongoDB is a powerful pairing and in this Quick Start series we show y...
 Author Lauren Schaefer
- 
404
<https://www.mongodb.com/blog/post/performance-tuning-mongodb>
<https://www.mongodb.com/blog/post/performance-tuning-mongodb>

• Videos:

- 
MongoDB Courses and Trainings | MongoDB University
 Discover our MongoDB Database Management courses and begin improving your CV with Mong...
<https://university.mongodb.com/courses/M220JS/about>
- 

GitHub Instructions

Open in Visual Studio Code:

After clicking on the "Open in Visual Studio Code" button from the GitHub Classroom confirmation page, Visual Studio Code (VSCode) will open the repository directly.

If prompted, select "Open" or "Allow" to open the repository in VSCode.

Complete the Task:

In VSCode, open the `index.js` file in the root directory of your repository and write your solution.

Ensure the `package.json` file is present and contains all necessary dependencies. If you need to install additional packages, use:

```
npm i
```

Run and Test Your Code:

Run your code to ensure it works correctly. Use the following command:

```
node index.js
```

Commit Your Changes:

Commit your changes with a meaningful message:

```
git commit -m "Completed task 1"
```

Push Your Changes to Your Forked Repository:

Push your changes to your forked repository:

```
git push origin main
```

Create a Pull Request:

Go to your forked repository on GitHub.

Click on the "Pull Requests" tab.

Click the "New Pull Request" button.

Ensure the base repository is the original template repository and the base branch is `main`.

Ensure the head repository is your forked repository and the compare branch is `main`.

Click "Create Pull Request".

Add a title and description for your pull request and submit it.

Summary of Commands

```
# Fork the repository on GitHub

# Clone the forked repository
git clone https://github.com/your-github-username/repository-name.git
cd repository-name

# Complete the task by editing index.js

# Run your code
node index.js

# Add, commit, and push your changes
git commit -m "Completed task 1"
git push origin main

# Create a pull request on GitHub
```