Task 2: Indexes in MongoDB

Objective:

Understand the concept and importance of indexes in MongoDB. Create and use indexes to optimize queries.

Prerequisites:

- Basic understanding of JavaScript and MongoDB.
- Node.js installation.
- MongoDB installed and running.
- A MongoDB collection with sample movie data.

Concepts:

1. Introduction to Indexes:

- Indexes are special data structures that store a small portion of the collection's data set in an easy-to-traverse form.
- The index stores the value of a specific field or set of fields, ordered by the value of the field.
- Indexes enhance query performance by providing efficient access to documents.
- However, indexes also require additional space and can slow down write operations.

2. Creating and Using Indexes:

Create an Index:

Create an index on the title field to speed up queries that filter or sort by this field.

Example:

JavaScript:

```
const { MongoClient } = require('mongodb');

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

  try {
    await client.connect();
    const database = client.db('movieDB');
    const collection = database.collection('movies');

    // Create an index on the title field
    const indexName = await collection.createIndex({ title: 1 });
    console.log(`Index created: ${indexName}`);
  } finally {
    await client.close();
  }
}
```



```
createIndex().catch(console.dir);
```

MongoDB Compass:

- Open MongoDB Compass.
- Connect to your MongoDB instance.
- Select your database and collection.
- Click on the Indexes tab.
- Click on Create Index.
- Add the field for the index:

```
{ "title": 1 }
```

• Click on Create Index button.

Using the Index:

• Once an index is created, MongoDB can use it to optimize query performance.

Example:

JavaScript:

```
const { MongoClient } = require('mongodb');

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

try {
  await client.connect();
  const database = client.db('movieDB');
  const collection = database.collection('movies');

  // Query using the index on the title field
  const movies = await collection.find({ title: "Inception" }).toArray();
  console.log('Movies:', movies);
} finally {
  await client.close();
}

findMovies().catch(console.dir);
```

MongoDB Compass:

- In MongoDB Compass, go to the Find tab.
- Enter the following query:



```
{ "title": "Inception" }
```

• Click on the Find button to execute the query.

3. Deleting an Index:

• Indexes can be deleted to free up storage space and reduce maintenance overhead on write operations.

Example:

JavaScript:

```
const { MongoClient } = require('mongodb');

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

  try {
    await client.connect();
    const database = client.db('movieDB');
    const collection = database.collection('movies');

    // Delete the index on the title field
    const result = await collection.dropIndex('title_1');
    console.log('Index deleted:', result);
  } finally {
    await client.close();
  }
}

deleteIndex().catch(console.dir);
```

MongoDB Compass:

- In MongoDB Compass, go to the Indexes tab.
- Find the index on the title field.
- Click on the Delete button next to the index.

4. Creating Additional Indexes:

- o Create an index on the genre field to optimize queries filtering or sorting by genre.
- Write a query that retrieves all movies in the "Sci-Fi" genre, utilizing the index.

Example:

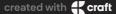
JavaScript:



```
const { MongoClient } = require('mongodb');
async function createGenreIndex() {
  const uri = 'mongodb://localhost:27017'; // Replace with your MongoDB URI
  const client = new MongoClient(uri, { useNewUrlParser: true,
useUnifiedTopology: true });
  try {
   await client.connect();
   const database = client.db('movieDB');
    const collection = database.collection('movies');
   // Create an index on the genre field
   const indexName = await collection.createIndex({ genre: 1 });
   console.log(`Genre index created: ${indexName}`);
   await client.close();
async function findSciFiMovies() {
 const uri = 'mongodb://localhost:27017'; // Replace with your MongoDB URI
  const client = new MongoClient(uri, { useNewUrlParser: true,
useUnifiedTopology: true });
   await client.connect();
   const database = client.db('movieDB');
   const collection = database.collection('movies');
   const movies = await collection.find({ genre: "Sci-Fi" }).toArray();
   console.log('Sci-Fi Movies:', movies);
  } finally {
    await client.close();
createGenreIndex().catch(console.dir);
findSciFiMovies().catch(console.dir);
```

MongoDB Compass:

- Open MongoDB Compass.
- Connect to your MongoDB instance.
- Select your database and collection.
- Click on the Indexes tab.
- Click on Create Index.
- Add the field for the index:



```
{ "genre": 1 }
```

- Click on Create Index button.
- In MongoDB Compass, go to the Find tab.
- Enter the following query:

```
{ "genre": "Sci-Fi" }
```

• Click on the Find button to execute the query.

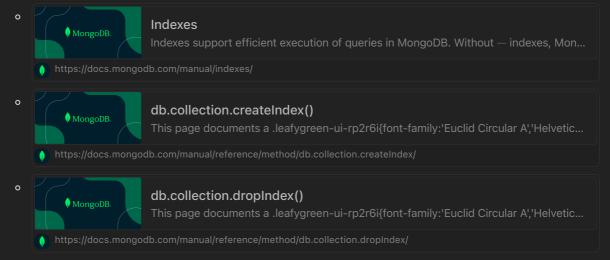
Instructions:

Perform the following tasks:

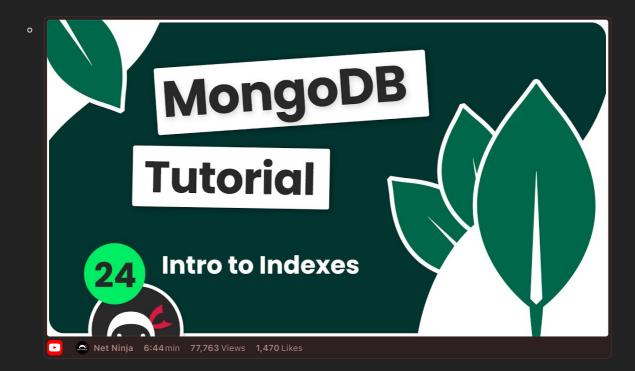
- 1. Create an index on the title field to speed up queries that filter or sort by this field.
- 2. Use the index to perform a query that retrieves all movies with the title "Inception".
- 3. Delete the index on the title field.
- 4. Create an index on the genre field to optimize queries filtering or sorting by genre.
- 5. Write a query that retrieves all movies in the "Sci-Fi" genre, utilizing the index.

Resources:

• Documentation:



• Videos:



GitHub Instructions

- 1. 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.

2. 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 install

3. Run and Test Your Code:

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

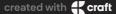
node index.js

4. Commit Your Changes:

• Commit your changes with a meaningful message:

git commit -m "Completed task 8"

5. Push Your Changes to Your Forked Repository:



• Push your changes to your forked repository:

```
git push origin main
```

6. 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

# commit, and push your changes
git commit -m "Completed task 2"
git push origin main

# Create a pull request on GitHub
```

