**EXPT 7- MONGODB**

**Aim :** Connect a Node.js application to a MongoDB database. Implement CRUD operations to interact with the database using the MongoDB Node.js driver.

**Theory :**

### **1. MongoDB Overview**

MongoDB is a NoSQL database that stores data in a flexible, JSON-like format called BSON. It allows for easy storage, retrieval, and manipulation of data without a fixed schema.

### **2. MongoDB Node.js Driver**

The MongoDB Node.js Driver is the official package that allows Node.js applications to connect and interact with MongoDB databases. It provides methods to perform CRUD operations and manage database connections.

### **3. Connection Setup**

To begin, you must install the MongoDB Node.js driver in your project (usually via npm). Then, you connect to the MongoDB server using a connection URI, which specifies the database address and credentials (if needed).

### **4. CRUD Operations**

* **Create (Insert)**
  + Used to add one or more new documents (records) into a collection (table equivalent).
  + Common methods: insertOne(), insertMany()
* **Read (Find)**
  + Used to query and retrieve data from the database.
  + Common methods: find(), findOne()
* **Update**
  + Used to modify existing documents in a collection.
  + Common methods: updateOne(), updateMany(), replaceOne()
* **Delete**
  + Used to remove documents from a collection.
  + Common methods: deleteOne(), deleteMany()

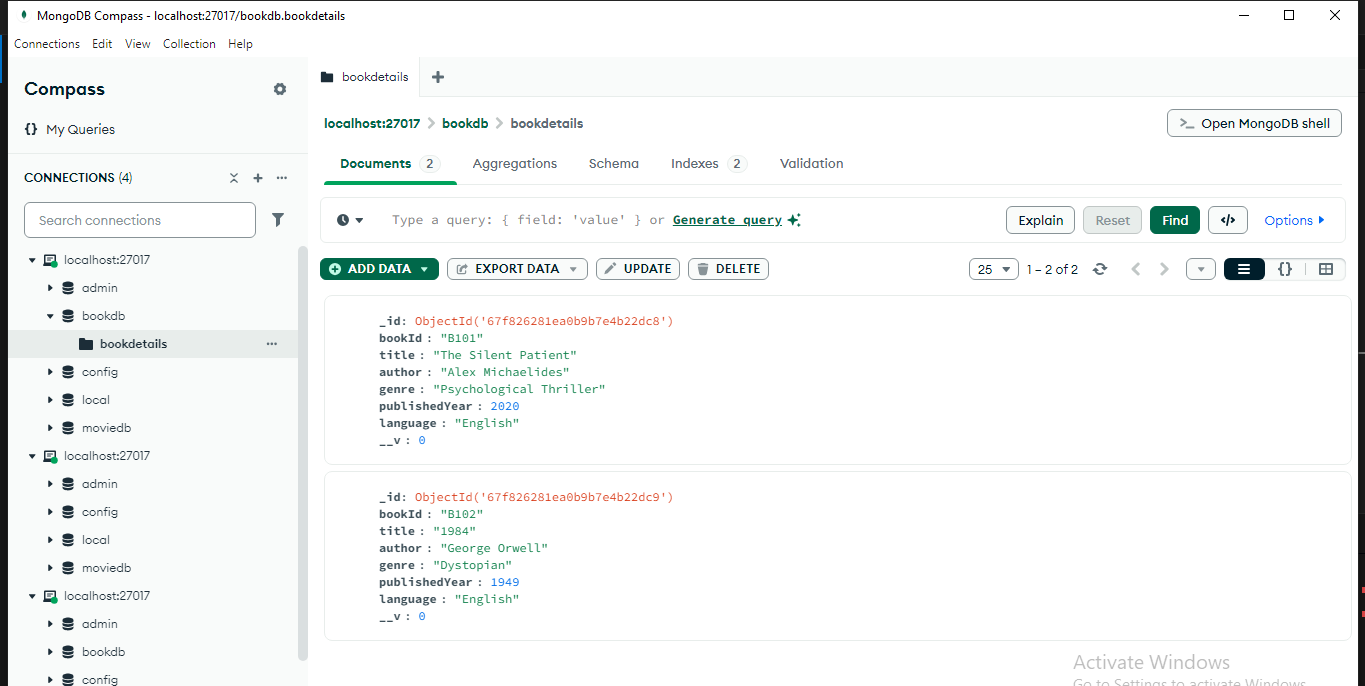
### **5. Error Handling**

Proper error handling is crucial when performing database operations to ensure the application behaves reliably and securely when issues occur, such as connection errors or failed operations.

### **6. Disconnection**

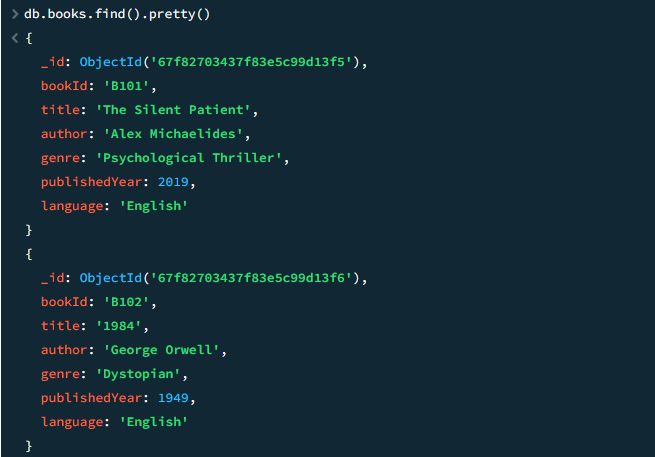
After completing operations, it's good practice to close the connection to the database to free up resources.

1.Initializing mongodb database and collection



2.Performing CRUD operation Using mongodb shell.

Finding all books



Adding multiple books.



**Mongodb driver in vscode :**

**actor.js (model )**

const mongoose = require('mongoose');

const bookSchema = new mongoose.Schema({

bookId: { type: String, unique: true, required: true },

title: { type: String, required: true },

author: { type: String, required: true },

genre: { type: String, required: true },

publishedYear: { type: Number, required: true },

language: { type: String, required: true }

});

const Book = mongoose.model('BookDetails', bookSchema);

module.exports = Book;

**server.js**

const mongoose = require('mongoose');

const Book = require('./model/book');

const uri = "mongodb://localhost:27017/bookdb";

async function performCRUD() {

try {

await mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true });

console.log("✅ Connected to MongoDB");

// INSERT

await Book.insertMany([

{

bookId: "B101",

title: "The Silent Patient",

author: "Alex Michaelides",

genre: "Psychological Thriller",

publishedYear: 2019,

language: "English"

},

{

bookId: "B102",

title: "1984",

author: "George Orwell",

genre: "Dystopian",

publishedYear: 1949,

language: "English"

},

{

bookId: "B103",

title: "Wings of Fire",

author: "A. P. J. Abdul Kalam",

genre: "Autobiography",

publishedYear: 1999,

language: "English"

}

]);

// READ

const allBooks = await Book.find({});

console.log("📚 All Books:\n", allBooks);

// UPDATE

await Book.updateOne(

{ bookId: "B101" },

{ $set: { publishedYear: 2020 } }

);

const updated = await Book.findOne({ bookId: "B101" });

console.log("🔄 Updated Book:\n", updated);

// DELETE

const deleted = await Book.deleteOne({ bookId: "B103" });

console.log("🗑️ Deleted Book:\n", deleted);

// FINAL LIST

const remaining = await Book.find({});

console.log("📃 Remaining Books:\n", remaining);

} catch (error) {

console.error("❌ MongoDB Error:", error);

} finally {

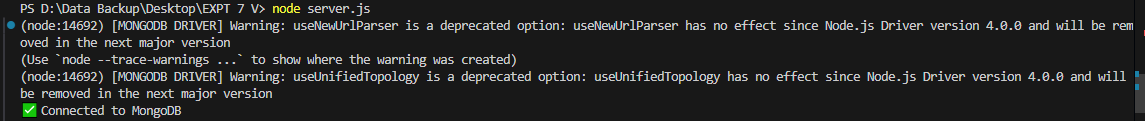
await mongoose.disconnect();

console.log("🔌 Disconnected from MongoDB");

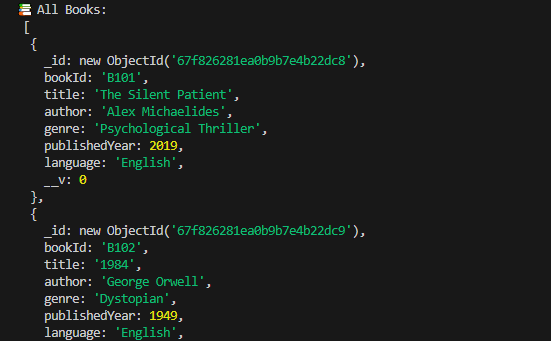
}

}

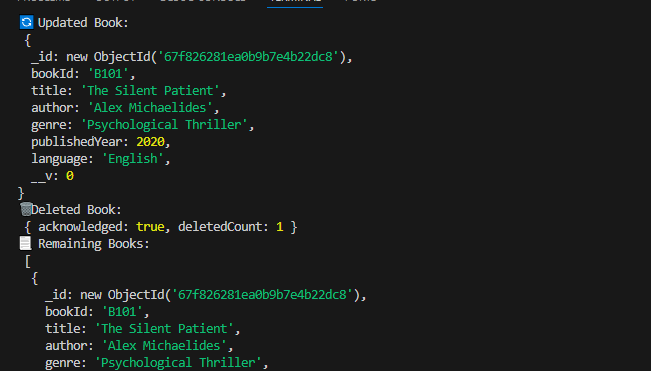
performCRUD();



1.Inserting many Books



2.Performing updation,deletion on actor database



**Conclusion :**In conclusion, integrating MongoDB with a Node.js application provides a powerful and flexible way to manage data using modern JavaScript tools. By leveraging the MongoDB Node.js driver, developers can seamlessly perform essential CRUD operations—such as creating, reading, updating, and deleting records—within their applications. This connection allows for dynamic and scalable data handling, especially well-suited for applications with flexible data requirements. With proper connection setup, efficient query methods, and structured error handling, a Node.js and MongoDB combination offers a reliable backend solution for both small-scale and enterprise-level projects