Module – Node

Node with MongoDB

Q1. What is MongoDB.

ANS.

MongoDB is a popular NoSQL database that stores data in a flexible, JSON like format called documents. Unlike traditional SQL database that uses tables and rows, MongoDB uses collections and documents. Which makes it more flexible and scalable, especially for handing large amounts of unstructured or semi–structured data.

Why use MongoDB ? : For its flexibility, document-Oriented data model, scalability, performance and developer – friendly features.

Q2. What is difference between mongo DB and SQL.

ANS.

SQL :

SQL databases are ideal for structured data, complex queries, and scenarios where data integrity is critical.

Generally better for complex queries involving multiple joins, aggregate functions, and high data integrity.

Vertical Scalability: Traditionally scales vertically by adding more resources (CPU, RAM) to the existing server.

MongoDB:

MongoDB is better suited for handling unstructured data, fast development cycles, and scenarios requiring high scalability.

Often faster for simple queries and can handle large volumes of read/write operations efficiently.

Horizontal Scalability: Designed for horizontal scaling through sharding, which distributes data across multiple servers.

Q3. Create database for online shopping app.

Ans.

First create an Express.js project with an EJS template.

then Create a database, follow these steps:

```
const express = require('express');
const app = express();

// npm i mongodb // Step 1 - install mongodb
const { MongoClient } = require('mongodb'); // Step 2 - Require the MongoDB client from the
mongodb package
const url = 'mongodb://localhost:27017/'; // Step 3 - Define the MongoDB connection URL
(connecting to a local MongoDB server)

const path = require('path');

const client = new MongoClient(url); //  Step 4 - Create a new instance of MongoClient to
interact with the MongoDB database using the provided URL.
const dbName = 'onlineShopping'; // Step 5 - Define the name of the database that you want to
use or create

app.set('view engine','ejs')
app.set('views', path.join(__dirname, 'views'));
app.use(express.urlencoded({extended: false }));
```

```
app.use(express.static('public'))

let db; // Step 6

const main = async ()=>{
    await client.connect(); // Step 7 - Connect to the MongoDB database using the client
instance; this is an asynchronous operation
    console.log('connected')
    db = client.db(dbName) // Step 8 - Assign the connected database to the 'db' variable,
allowing further operations on the database
}

main()

app.get("/", (req, res) => {
    res.send('hy')
})

app.listen(4000);
```

The database will not appear in MongoDB Compass if it's just been created but no collections or documents have been added.

Q4. Create Require collections for online shopping app and documents.

i. User

ii. Product category

iii. Product

iv. Order

v. Review

Ans.

Create all collections and add data from users

```
// Create User document
app.post('/add-user', (req, res) => {
    // step-1 Access the 'users' collection from the connected database.
    const usersCollection = db.collection('users'); // collection name = 'users' (check
MongoDB Compass)

    // step-2. Create a new user object with properties 'name', 'email', and 'password'.
    // These properties are extracted from the request body, which contains the form data
submitted by the user.
    const newUser = {  // create format
        name: req.body.name,      // Extract the 'name' field from the form data
        email: req.body.email,    // Extract the 'email' field from the form data
        password: req.body.password // Extract the 'password' field from the form data
    };

    // step-3. The 'insertOne' method is used to add a single document (newUser) to the
collection.
    usersCollection.insertOne(newUser) // insert data
        res.redirect('/')
});

// Similarly add all collection

// Create category document
app.post('/add-category', (req, res)=>{
    const categoryCollection = db.collection('category');
    const newCat = {
        name: req.body.name,
        description: req.body.description
    };
    categoryCollection.insertOne(newCat)
    res.redirect('/category')
})

// Create product document
app.post('/add-product', (req, res)=>{
    const productCollection = db.collection('product');
    const newProduct = {
        name: req.body.name,
        price: req.body.price,
        category: req.body.category,
        description: req.body.description
    };
    productCollection.insertOne(newProduct)
    res.redirect('/product')
})

// Create Order document
app.post('/add-order', (req, res)=>{
```

```
    const orderCollection = db.collection('order');
    const newOrder = {
        userId: req.body.userId,
        productId: req.body.productId,
        quantity: req.body.quantity,
        status: req.body.status
    };
    orderCollection.insertOne(newOrder)
    res.redirect('/order')
})

// Create Review document
app.post('/add-review', (req, res)=>{
    const reviewCollection = db.collection('review');
    const newReview = {
        productId: req.body.productId,
        userId: req.body.userId,
        rating: req.body.rating,
        comment: req.body.comment
    };
    reviewCollection.insertOne(newReview)
    res.redirect('/review')
})
```

Q5. Write command to show all data from product collections and short in ascending order.

Ans.

Show data in show-data page and shorting in ascending order

```
// Show and short data
app.get('/show-data', async (req,res)=>{
    const showCollection = db.collection('product');
    let products = await showCollection.find({}).sort({ name: 1 }).toArray();//
This returns an array of products // sort() function use for shorting (short by
name)
    res.render('show-data', { products:  products});// Pass the array to the view
})
```

```
<div class="container">
        <a href="/product">Go back</a>
```

```
        <h1>Product List</h1>
        <div class="product-list">
            <% products.forEach(product => { %>
                <div class="product-item">
                    <h2 class="product-name"><%= product.name %></h2>
                    <p class="product-price">Price: $<%= product.price %></p>
                    <p class="product-category">Category: <%=
product.category %></p>
                    <p class="product-description"><%= product.description %></p>
                </div>
            <% }) %>
        </div>
    </div>
```

Q6. Update product price for particular product.

Ans.

Add update page and Update product price and other data.

```
// update price
// step - 1 add old data on update page
app.get('/update/:id', async (req, res) => {
    const productCollection = db.collection('product');
    const productId = req.params.id; // Get the product ID from the URL

    const product = await productCollection.findOne({ _id: new
ObjectId(productId) }); // Use ObjectId to convert the string ID to an ObjectId
    res.render('update', { product: product });
});

//step - 2 data updated and show
app.post('/update-product/:id',async (req, res) => {
    const productCollection = db.collection('product');
    const productId = req.params.id;
    const updatedProduct = {price: req.body.price};
    await productCollection.updateOne(
        { _id: new ObjectId(productId) }, // Find the product by ID
        { $set: updatedProduct } // Update the product fields
```

```
    );
    res.redirect('/show-data'); // Redirect back to the list of products
});
```

Q7. Write command to delete particular document and collection.

Ans.

Delete particular data in documents.

```
// delete collection

app.get('/delete/:id', async (req, res) => {
    const productCollection = db.collection('product');
    const productId = req.params.id;

    await productCollection.deleteOne({ _id: new ObjectId(productId) }); // Use
ObjectId to find the document by ID

    res.redirect('/show-data')
})
```