**Topics Covered:**

1. **What is MongoDB?**
2. **How it is different from SQL databases?**
3. **Installing and Setting Up MongoDB**
4. **Basic MongoDB Commands with Examples**
5. **CRUD Operations (Create, Read, Update, Delete)**
6. **Using MongoDB with Node.js (Step-by-Step Explanation)**

**Introduction to MongoDB for Primary School Students**

**What is MongoDB?**

MongoDB is a type of database that stores data in a flexible format called **JSON-like documents** instead of tables (like SQL databases). Imagine a big **digital notebook** where each page contains different types of information but still stays organized.

**Example Comparison with DBMS (SQL vs. MongoDB)**

| **Feature** | **SQL (Relational Database)** | **MongoDB (NoSQL Database)** |
| --- | --- | --- |
| Data Storage | Tables (Rows & Columns) | Documents (JSON format) |
| Schema | Fixed Structure | Flexible Structure |
| Example Use | Banking Systems | Social Media, E-commerce |

Think of a **school register**:

* A SQL database is like a traditional attendance register **(structured rows and columns)**.
* MongoDB is like a **student diary** where you can write different things (name, hobby, age) without following a fixed structure.

**Installing and Setting Up MongoDB**

To use MongoDB, you first need to install it on your computer. Here are the steps:

1. **Download MongoDB Community Server** from [MongoDB's website](https://www.mongodb.com/try/download/community).
2. **Install MongoDB** and set up the database path.
3. **Start MongoDB Server** using the command:
4. mongod
5. **Open MongoDB Shell** and start using commands:
6. mongo

**Basic MongoDB Commands with Examples**

**1. Creating a Database**

To create a new database, type:

use schoolDB

➡️ This creates a database called schoolDB. Think of it like creating a **new school folder** where we will store information.

**2. Creating a Collection (Table Equivalent in SQL)**

db.createCollection("students")

➡️ This creates a collection named **students** where we will store student records.

**3. Inserting Data into a Collection**

db.students.insertOne({ name: "Alice", age: 10, grade: "5th" })

➡️ This adds a student named **Alice** into our students collection.

🔹 **Example Explained for Kids:** Imagine writing a student’s name, age, and class on a piece of paper and adding it to a file.

**4. Viewing Data**

db.students.find()

➡️ This displays all students stored in the database.

**5. Updating a Student's Age**

db.students.updateOne({ name: "Alice" }, { $set: { age: 11 } })

➡️ This updates Alice’s age to 11.

**6. Deleting a Record**

db.students.deleteOne({ name: "Alice" })

➡️ This removes Alice's record from the database.

**db.students.find({name:’alice’},{age:0}).pretty()**

**Using MongoDB with Node.js**

To connect MongoDB with a Node.js server:

**1. Install MongoDB Package**

npm install mongodb

➡️ This installs the required package to use MongoDB in our Node.js program.

**2. Creating a Simple Node.js Server**

const { MongoClient } = require("mongodb");

const url = "mongodb://localhost:27017";

const client = new MongoClient(url);

async function connectDB() {

await client.connect();

console.log("Connected to MongoDB");

const db = client.db("schoolDB");

const students = db.collection("students");

// Insert a student record

await students.insertOne({ name: "Alice", age: 10, grade: "5th" });

console.log("Student record inserted successfully!");

}

connectDB();

➡️ **Explanation:**

1. We import MongoClient class from mongodb package which is a collection of multiple classes and function and so inorder to bring only mongoClient class we use {}.
2. We specify the **MongoDB server URL** (localhost:27017).
3. We connect to MongoDB and select the **schoolDB** database.
4. We create a students collection.
5. We insert a student record and print confirmation.
6. If everything is correct, we see Connected to MongoDB in the console.

**Conclusion**

✅ **MongoDB is a NoSQL database** that stores data in flexible JSON-like documents.

✅ **It is different from SQL** because it does not use tables and schemas.

✅ **It is used in modern applications** like e-commerce, social media, and real-time tracking.

✅ **We can connect it with Node.js** to build powerful applications.

🎉 **Practice Task for Students:**

1. Insert your name, age, and favorite subject into a students collection.
2. Retrieve and display all student records.
3. Update your favorite subject.
4. Delete your record and verify it is removed.

**MongoDB User Authentication System with Node.js**

**Introduction**

This project implements a simple user authentication system using **Node.js**, **MongoDB**, and **JWT (JSON Web Token)**. It includes:

* **User Registration & Login** with MongoDB
* **Serving HTML Pages** for SignUp, Login.
* **API Testing using Postman**

**1️⃣ Setup Instructions**

**Step 1: Install Required Software**

* **Node.js 20** (JavaScript runtime)
* **MongoDB** (Database)
* **Visual Studio Code** (Code editor)
* **Postman** or **Thunder Client** (For API testing)

**Step 2: Initialize the Node.js Project**

Run the following command:

npm init -y

This creates a package.json file.

**Step 3: Install Required Modules**

npm install express nodemon fs path jsonwebtoken mongodb

This installs:

* **express** → For handling API requests
* **nodemon** → To auto-restart the server
* **fs & path** → For file handling
* **jsonwebtoken** → To manage user authentication
* **mongodb** → To connect Node.js with MongoDB

**2️⃣ MongoDB Connection and User Model**

Create a file database.js.

**database.js (MongoDB Connection & User Model)**

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

const url = 'mongodb://127.0.0.1:27017';

const client = new MongoClient(url);

let collection;

async function connectDB(dbname, table) {

    let result = await client.connect();

    let db = result.db(dbname);

    collection = db.collection(table);

    console.log("Database Connected...");

    return collection;

}

exports.getData = async function (email) {

    collection = await connectDB("usersDB", "users");

    let response = await collection.find({ email: email }).toArray();

    return JSON.stringify(response);

};

exports.insertData = async function (user) {

    collection = await connectDB("usersDB", "users");

    let response = await collection.insertOne({ name: user.name, email: user.email, password: user.password });

    return JSON.stringify(response);

}

**Database Module (database.js)**

This module connects to MongoDB and provides functions for inserting and retrieving user data.

**1. Import Dependencies**

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

* **MongoClient**: The official MongoDB driver for Node.js.

**2. Connect to MongoDB**

const url = 'mongodb://127.0.0.1:27017';

const client = new MongoClient(url);

let collection;

* **MongoDB URL**: Connects to local MongoDB instance (localhost:27017).
* **client**: Creates a new MongoDB client.
* **collection**: Stores a reference to the selected collection.

**3. Database Connection Function**

async function connectDB(dbname, table) {

let result = await client.connect(); // Connects to MongoDB

let db = result.db(dbname); // Selects the database

collection = db.collection(table); // Selects the collection (table)

console.log("Database Connected...");

return collection;

}

* Uses async/await to **ensure** the database is connected before executing queries.
* Even though the **database name (usersDB)** and **collection name (users)** are fixed in your use case, the function connectDB(dbname, table) is designed to be **flexible** for potential future use.

**4. Retrieve User Data**

exports.getData = async function (email) {

collection = await connectDB("usersDB", "users"); // Ensure connection

let response = await collection.find({ email: email }).toArray(); // Fetch data

return JSON.stringify(response);

};

* **find({ email: email })**: Searches for a user by email.
* **toArray()**: Converts results into an array.

**5. Insert User Data**

exports.insertData = async function (user) {

collection = await connectDB("usersDB", "users");

let response = await collection.insertOne({ name: user.name, email: user.email, password: user.password });

return JSON.stringify(response);

};

* **insertOne({ ... })**: Adds a new user record.

**Key Points**

* **async/await ensures proper database connection** before running queries.
* **Database name**: "usersDB"
* **Collection name**: "users"
* **Two functions**:
  1. getData(email): Retrieves user details.
  2. insertData(user): Adds a new user.

**3️⃣ Creating API Endpoints**

Create a server.js file to handle user authentication.

**server.js (Handling User Authentication & Serving HTML Pages)**

const express = require("express");

const db = require("./database.js");

const jwt = require("jsonwebtoken");

const path = require("path");

const app = express();

const port = 3000;

app.use(express.json());

app.use(express.static("public"));

app.get('/', (req, res) => res.sendFile(path.join(\_\_dirname, "public", "index.html")));

app.get('/signup', (req, res) => res.sendFile(path.join(\_\_dirname, "public", "signup.html")));

app.get('/login', (req, res) => res.sendFile(path.join(\_\_dirname, "public", "login.html")));

app.get('/welcome', (req, res) => res.sendFile(path.join(\_\_dirname, "public", "welcome.html")));

app.post("/register", async (req, res) => {

const { name, email, password } = req.body;

let userExists = await db.getData(name, password);

if (JSON.parse(userExists).length > 0) {

res.json({ signup: false, error: "User already registered" });

} else {

const token = jwt.sign({ name, email }, "secret\_key");

let newUser = { name, email, password, token };

let result = await db.insertData(newUser);

res.json({ signup: true, token, result });

}

});

app.post("/login", async (req, res) => {

const { name, password } = req.body;

let user = await db.getData(name, password);

if (JSON.parse(user).length > 0) {

const token = jwt.sign({ name }, "secret\_key");

res.json({ login: true, token, data: JSON.parse(user) });

} else {

res.json({ login: false, error: "Invalid username or password" });

}

});

app.listen(port, () => {

console.log(`Server is running on http://localhost:${port}/`);

});

**Express Server (server.js)**

This file sets up an **Express.js server** that handles **user registration, login, and page routing**.

**1. Import Dependencies**

const express = require("express");

const db = require("./database.js");

const jwt = require("jsonwebtoken");

const path = require("path");

* **express** → Handles server requests and responses.
* **db** → Imports MongoDB functions from database.js.
* **jsonwebtoken (JWT)** → Generates authentication tokens.
* **path** → Helps serve HTML files.

**2. Initialize Express App**

const app = express();

const port = 3000;

* **Creates an Express application** and sets the server to run on **port 3000**.

**3. Middleware Setup**

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

app.use(express.static("public"));

* **express.json()** → Parses JSON data from requests (for API calls).
* **express.urlencoded({ extended: true })** → Parses form data (from <form> submissions).
* **express.static("public")** → Serves static files (HTML, CSS, JS) from the public folder.

***Both express.json() and express.urlencoded({ extended: true }) are middleware functions in Express.js that help parse incoming request bodies. However, they handle different types of data.***

### *****1️⃣***** *express.json()*

***app.use(express.json());***

***✅ Purpose: Parses incoming JSON data from the request body.  
✅ Use Case: When sending data as JSON format (e.g., API requests).***

#### **Example: Sending JSON Data**

***Client sends a POST request:***

***{***

***"name": "John",***

***"email": "john@example.com"***

***}***

* ***express.json() will parse this and make it available as req.body:***

***app.post("/data", (req, res) => {***

***console.log(req.body); // { name: 'John', email: 'john@example.com' }***

***res.send("Data received");***

***});***

### *****2️⃣***** *express.urlencoded({ extended: true })*

***app.use(express.urlencoded({ extended: true }));***

***✅ Purpose: Parses incoming form data (from HTML forms).  
✅ Use Case: When sending data using application/x-www-form-urlencoded (default form submission).***

#### **Example: Sending Form Data**

***If an HTML form submits data:***

***<form action="/register" method="POST">***

***<input type="text" name="name">***

***<input type="email" name="email">***

***<button type="submit">Submit</button>***

***</form>***

* ***The browser sends data as:***

***name=John&email=john@example.com***

* ***express.urlencoded({ extended: true }) parses it into an object:***

***app.post("/register", (req, res) => {***

***console.log(req.body); // { name: 'John', email: 'john@example.com' }***

***res.send("Form submitted");***

***});***

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

**4. Serve HTML Pages**

app.get('/', (req, res) => res.sendFile(path.join(\_\_dirname, "public", "index.html")));

app.get('/signup', (req, res) => res.sendFile(path.join(\_\_dirname, "public", "signup.html")));

app.get('/login', (req, res) => res.sendFile(path.join(\_\_dirname, "public", "login.html")));

app.get('/welcome', (req, res) => res.sendFile(path.join(\_\_dirname, "public", "welcome.html")));

* **Routes to serve HTML files** when users visit specific URLs.
* **Example**:
  + Visiting /signup → Loads signup.html.
  + Visiting /login → Loads login.html.

**5. User Registration (POST /register)**

app.post("/register", async (req, res) => {

const { name, email, password } = req.body;

let userExists = await db.getData(email);

if (JSON.parse(userExists).length > 0) {

res.json({ signup: false, error: "User already registered with this email" });

} else {

const token = jwt.sign({ name, email }, "secret\_key");

let newUser = { name, email, password, token };

let result = await db.insertData(newUser);

res.json({ signup: true, token, result });

}

});

✅ **Steps Explained**:

1. Extracts **name, email, and password** from the request body.
2. **Checks if the user already exists** in the database.
3. If **user exists**, returns { signup: false, error: "User already registered" }.
4. If **user does not exist**, creates a **JWT token** and inserts the new user.
5. Returns a **success response** with the JWT token.

**6. User Login (POST /login)**

app.post("/login", async (req, res) => {

const { email, password } = req.body;

let user = await db.getData(email, password);

if (JSON.parse(user).length > 0) {

const token = jwt.sign({ email }, "secret\_key");

res.json({ message: "Welcome to your profile", login: true, token, data: JSON.parse(user) });

} else {

res.json({ login: false, error: "Invalid username or password" });

}

});

✅ **Steps Explained**:

1. Extracts **email and password** from the request body.
2. Searches for a matching user in the database.
3. If **user exists**, generates a **JWT token** and sends { login: true } response.
4. If **user does not exist**, returns { login: false, error: "Invalid username or password" }.

**7. Start the Server**

app.listen(port, () => {

console.log(`Server is running on http://localhost:${port}/`);

});

* **Starts the Express server** on port 3000.
* Logs a message when the server is running.

**🔹 Summary**

* **Serves HTML pages** (/signup, /login, /welcome).
* **Handles user registration** (POST /register).
* **Handles user login** (POST /login).
* **Uses JWT tokens** for authentication.
* **Connects to MongoDB** via database.js.

**4️⃣ HTML Pages for Frontend**

Create a public folder and add these HTML files inside it.

**public/index.html**

<!DOCTYPE html>

<html>

<head><title>Home</title></head>

<body>

<h2>Welcome to User Authentication System</h2>

<a href="/signup">Sign Up</a> | <a href="/login">Login</a>

</body>

</html>

**public/signup.html**

<!DOCTYPE html>

<html>

<head><title>Sign Up</title></head>

<body>

<h2>Sign Up</h2>

<form action="/register" method="post">

Name: <input type="text" name="name"><br>

Email: <input type="email" name="email"><br>

Password: <input type="password" name="password"><br>

<button type="submit">Register</button>

</form>

</body>

</html>

**public/login.html**

<!DOCTYPE html>

<html>

<head><title>Login</title></head>

<body>

<h2>Login</h2>

<form action="/login" method="post">

Name: <input type="text" name="name"><br>

Password: <input type="password" name="password"><br>

<button type="submit">Login</button>

</form>

</body>

</html>

**public/welcome.html**

<!DOCTYPE html>

<html>

<head><title>Welcome</title></head>

<body>

<h2>Welcome User</h2>

<p>You have successfully logged in!</p>

</body>

</html>

**5️⃣ Final Steps**

**Run MongoDB**

mongod

**Start the Node.js Server**

node server.js

**Test in Browser**

Visit:

* http://localhost:3000/ (Home Page)
* http://localhost:3000/signup (SignUp Page)
* http://localhost:3000/login (Login Page)
* http://localhost:3000/welcome (After successful login)

**Conclusion**

✅ **MongoDB stores user data** ✅ **Node.js API handles user authentication** ✅ **JWT is used for secure logins** ✅ **Postman is used for API testing** ✅ **Frontend pages allow user interaction**

This setup fully implements a **User Authentication System** using **MongoDB + Node.js** 🚀

ERROR:

If your MongoDB is installed correctly but shows "files not found" in the command prompt, try these steps:

**1️⃣ Check MongoDB Installation**

Run:

mongod --version

If MongoDB is installed correctly, it should return the version.

**2️⃣ Check if MongoDB Service is Running**

Try starting the MongoDB service manually:

net start MongoDB

or

mongod

If you see an error, MongoDB might not be running.

**3️⃣ Check MongoDB Data Directory**

MongoDB stores data in the C:\data\db directory (Windows). If this folder is missing, create it manually:

mkdir C:\data\db

Then try running:

mongod

**4️⃣ Verify MongoDB Path**

Ensure MongoDB is added to the system PATH:

1. Open Command Prompt and run:

echo %PATH%

1. If MongoDB is not listed, add it manually:
   * Go to **Control Panel > System > Advanced System Settings > Environment Variables**.
   * Under **System Variables**, find Path and **Edit**.
   * Add: C:\Program Files\MongoDB\Server\6.0\bin (or your MongoDB version folder).
   * Restart your system.

**5️⃣ Try Running MongoDB in a New Command Prompt**

After adding it to PATH, open a new command prompt and run:

mongod

If it works, try:

mongo

to start the MongoDB shell.

To implement this is in postman follow the same steps as we did in week 6 and 7.