| **Experiment No. – 8** | | | | |
| --- | --- | --- | --- | --- |
| **Date of Performance:** | **9/4/25** | | | |
| **Date of Submission:** | **16/4/25** | | | |
| Program Execution/  formation/  correction/  ethical practices  (06) | Timely  Submission  (01) | Viva  (03) | Experiment  Total (10) | Sign with Date |
|  |  |  |  |  |

**8.1 Aim:** RESTful API integration using MongoDB.

**8.2 Course Outcome:** Apply MongoDB for frontend and backend connectivity using REST API

**8.3 Learning Objective:**Comprehend the architecture and design principles of MongoDB as a NoSQL database, install and configure it on a local machine, perform and test basic CRUD operations, connect and interact with MongoDB via RESTful APIs using Node.js, and compare SQL and NoSQL data storage mechanisms along with their use-cases.

**8.4 Requirement:**

**Software & Tools:**

* MongoDB Community Server
* MongoDB Shell (mongo)
* Node.js & npm
* Code Editor (Visual Studio Code / Notepad++)
* Terminal / Command Prompt
* Web Browser (for documentation)

**8.5 Theory :**

### **Introduction to MongoDB**

MongoDB is a high-performance, scalable, and schema-less NoSQL database that stores data in BSON (Binary JSON) format. Its flexible document model allows developers to handle complex data types and rapidly changing requirements.

### **MongoDB vs. Traditional SQL Databases**

| **SQL Database** | **NoSQL Database** |
| --- | --- |
| Table-Based | Document Based |
| Fixed Schema | Dynamic Schema |
| Uses SQL queries | Uses Jason-like queries |
| Vertical Scalability | Horizontal Scalability |

### **MongoDB Key Concepts**

* Database: A container for collections.
* Collection: Analogous to a table in SQL; holds multiple documents.
* Document: The fundamental unit of data, represented as a JSON-like structure.
* Indexing: Improves performance of queries.
* Replication and Sharding: Supports high availability and scalability.

**8.6 Procedure:**

**Installation Process**

### Step-by-Step Setup (Windows Example)

1. Download MongoDB:  
    Visit [MongoDB Community Download Page](https://www.mongodb.com/try/download/community) and select your operating system.
2. Run Installer:  
   Accept the license agreement.
3. Choose "Complete" installation.  
   Install MongoDB as a service.
4. Configure MongoDB:  
   Create a data directory:  
    mkdir C:\data\db  
   Add MongoDB to the system path.
5. Start MongoDB:  
   Launch the MongoDB server:  
    mongod  
   Open a new terminal to run the MongoDB shell:  
    mongo
6. Verify Installation:  
   Use mongo --version to confirm successful setup.

## **CRUD Operations with MongoDB**

CRUD operations form the backbone of data handling in RESTful applications:

* Create: Add new documents to a collection.
* Read: Query and retrieve documents.
* Update: Modify existing document data.
* Delete: Remove documents from the collection.

These operations can be executed via the MongoDB shell or through API calls integrated into backend services.

## **RESTful API Integration with MongoDB**

### **Overview**

To interact with MongoDB from a frontend application, a backend must expose endpoints that implement REST principles. A RESTful API provides a structured way to perform CRUD operations over HTTP.

### **Key Components**

* Node.js: Provides the runtime environment for building backend services.
* Express.js: A web framework for building RESTful routes/endpoints.
* MongoDB Driver / Mongoose: For connecting and interacting with MongoDB.

### **Typical API Routes**

| HTTP Method | Route | Operation | MongoDB Equivalent |
| --- | --- | --- | --- |
| POST | /users | Create | insertOne() / insertMany() |
| GET | /users | Read (all) | find() |
| GET | /users/:id | Read (specific) | findOne() |
| PUT/PATCH | /users/:id | Update | updateOne() |
| DELETE | /users/:id | Delete | deleteOne() / deleteMany() |

### **Workflow**

1. Frontend sends HTTP requests to API endpoints.
2. Backend processes the request and interacts with MongoDB to execute CRUD operations.
3. Responses are returned in JSON format to the client.

**8.7 Program and Output:**

**Code :**

**Fronted Part:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"

integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">

<title>Webx Practical</title>

</head>

<body>

<h1 class="text-center mt-2">Student Form</h1>

<h2 class="mt-3">Add Student</h2>

<form action="/addUser" method="POST">

<input type="text" name="name" placeholder="Enter Name" class="form-control w-50" required>

<input type="email" name="email" placeholder="Enter Email" class="form-control mt-2 w-50" required> <br>

<button type="submit" class="btn btn-primary">Add User</button>

</form>

<h2 class="mt-3">Delete Student</h2>

<form action="/deleteUser" method="POST">

<input type="email" name="email" placeholder="Enter Email to Delete" class="form-control mt-2 w-50" required> <br>

<button type="submit" class="btn btn-primary">Delete Student</button>

</form>

<h2 class="mt-3">All Student</h2>

<a href="/getUsers">Refresh users</a>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"

integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"

crossorigin="anonymous"></script>

</body>

</html>

**Backend Part :**

**Node:**

const express = require('express');

const MongoClient = require('mongodb').MongoClient;

const app = express();

const port = 3000;

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

const dbName = "stuDB";

let db;

// Middleware

app.use(express.json())

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

// Connect to MongoDB

MongoClient.connect(url)

.then(client => {

db = client.db(dbName);

console.log("Connected to MongoDB");

})

.catch(err => console.error(err));

app.get('/', (req, res) => {

res.sendFile(\_\_dirname + "/index.html");

});

// Create User (Insert)

app.post('/addUser', (req, res) => {

const newUser = { name: req.body.name, email: req.body.email };

db.collection("users").insertOne(newUser)

.then(() => res.redirect('/'))

.catch(err => res.send("Error inserting user: " + err));

});

// Read Users (Fetch Data)

app.get('/getUsers', (req, res) => {

db.collection("users").find().toArray()

.then(users => res.send(users))

.catch(err => res.send("Error fetching users: " + err));

});

app.post('/deleteUser', (req, res) => {

const query = { email: req.body.email };

db.collection("users").deleteOne(query)

.then(() => res.redirect('/'))

.catch(err => res.send("Error deleting user: " + err));

});

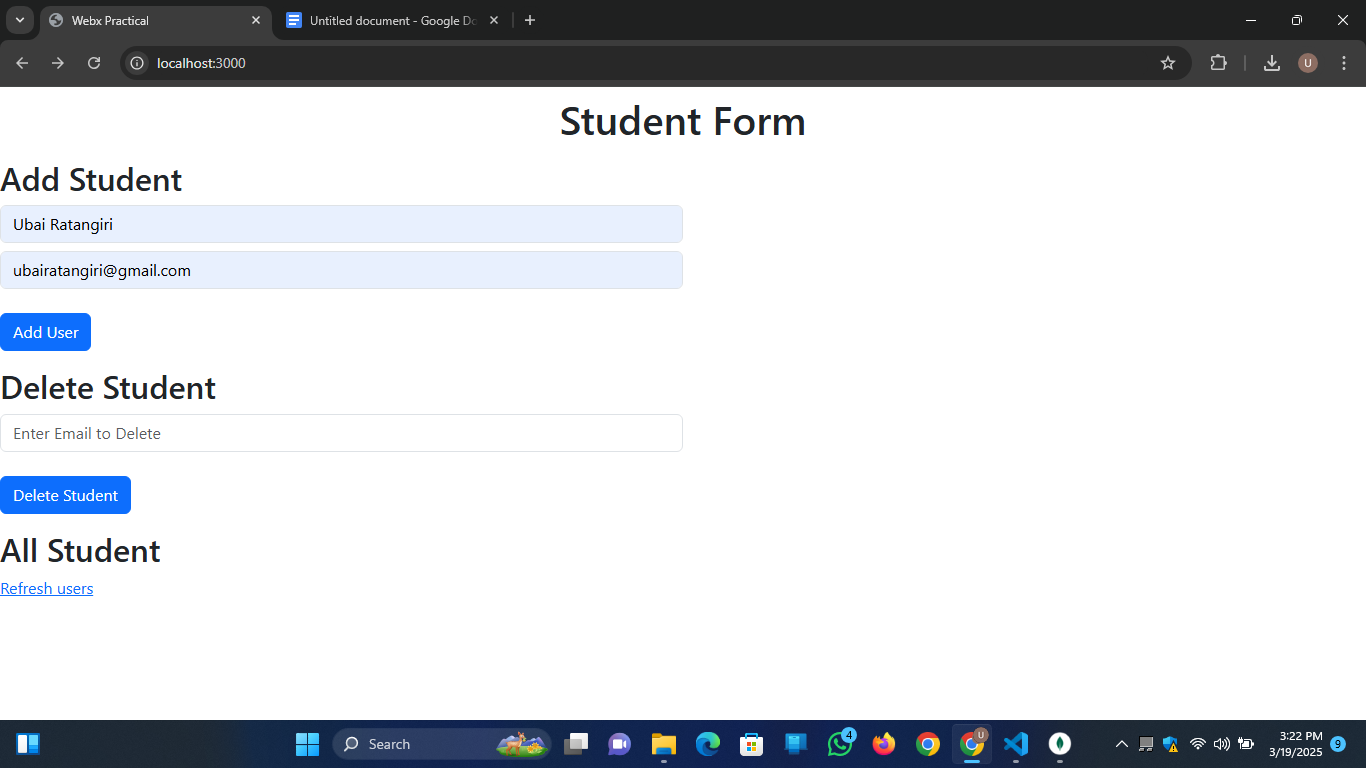
// Start Server

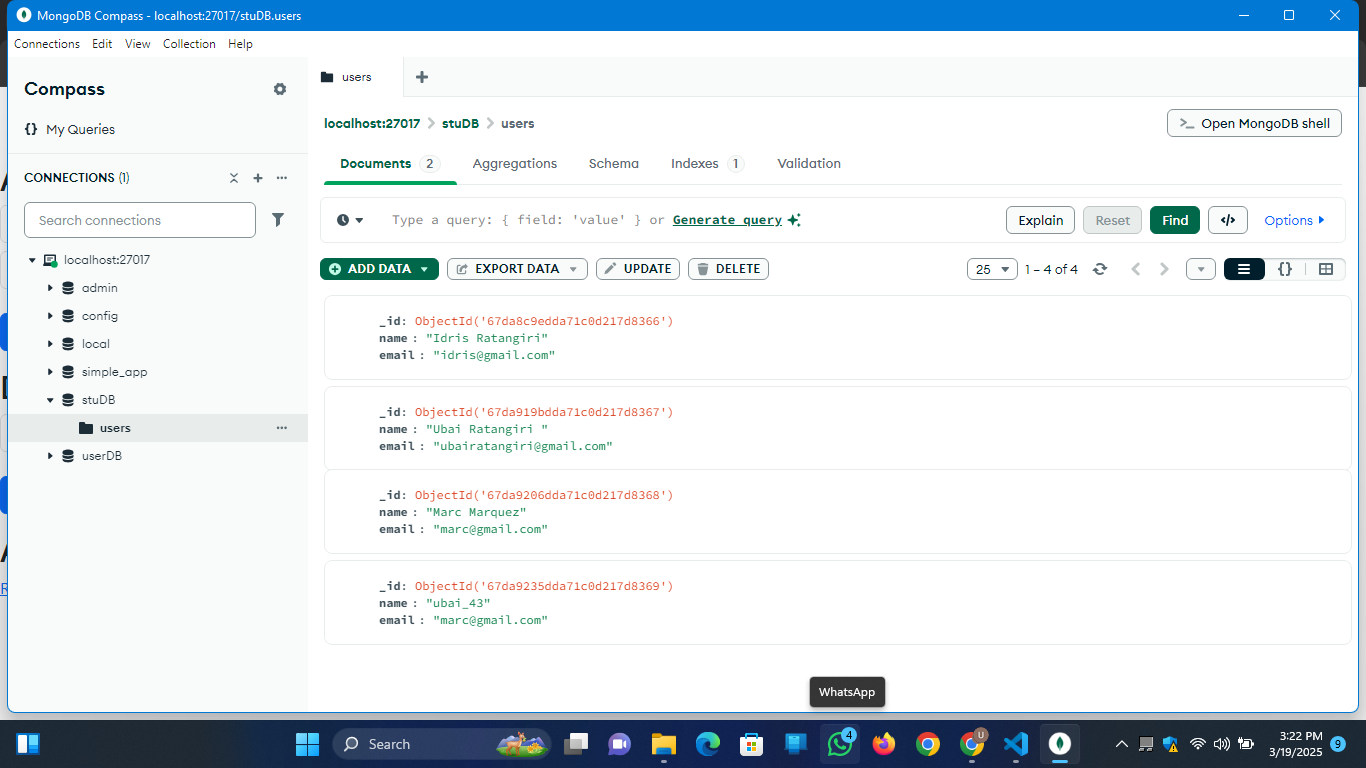
app.listen(port, () => {

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

});

**Output:**





**8.8 Conclusion:**

In this study, we explored MongoDB’s integration with RESTful APIs to build scalable and modern web apps. Its flexible schema and Node.js compatibility make it ideal for dynamic projects. Understanding CRUD operations and REST principles is key for full-stack development and real-world deployment.

**8.9 Question:**

1. What is a RESTful API and how does it interact with MongoDB?

A RESTful API uses HTTP methods to interact with resources. It connects with MongoDB to perform CRUD operations and sends back responses, usually in JSON format.

1. Why is Node.js commonly used for building RESTful APIs with MongoDB?

Node.js is fast, scalable, and works well with MongoDB’s JSON-like data format, making it ideal for building high-performance APIs.

1. What is the role of Express.js in this experiment?

Express.js simplifies the creation of RESTful API routes and managing HTTP requests between the frontend and MongoDB.