|  |  |  |  |
| --- | --- | --- | --- |
|  | **Course Name: Advanced Web Technology** | **EXPERIMENT NO. 3** | |
| **Course Code: 20CP314P**  **Faculty: Komal Singh** | **Branch: CSE** | **Semester: VI** |
| **(To be filled by Student)**  **Submitted by:**  **Roll no:** | | | |

**Objective:** Setting up a MongoDB Database (Connecting MongoDB to your application)

**Experiment 3**: Create a JavaScript file with MongoDB queries for operations such as insert, update, and delete while also establishing a connection to the MongoDB database.

In this experiment, I initiated by installing MongoDB, meticulously configuring it to ensure optimal performance on my system. Leveraging various installation methods, including Homebrew for macOS, I established a robust MongoDB environment, laying the foundation for subsequent database operations. With MongoDB up and running, I seamlessly transitioned into database management tasks, creating and connecting to a local database instance effortlessly. This initial setup phase facilitated a smooth transition into executing CRUD operations, wherein I honed my skills in inserting, querying, updating, and deleting data using MongoDB's flexible querying capabilities.

Furthermore, I delved deeper into MongoDB's functionality, experimenting with JavaScript and the Mongoose library to interact with the database programmatically. Through practical exercises, I gained invaluable insights into manipulating data structures, performing complex queries, and updating multiple records simultaneously. This hands-on experience not only solidified my understanding of MongoDB's core concepts but also underscored its significance in modern web development paradigms. Armed with this knowledge, I am better equipped to architect robust database solutions and optimize data management workflows in future projects.

FILE REPO - [REPO](https://github.com/goffycoder/Advance_web_-Technology_Assignments/tree/main/Practicals/p3)

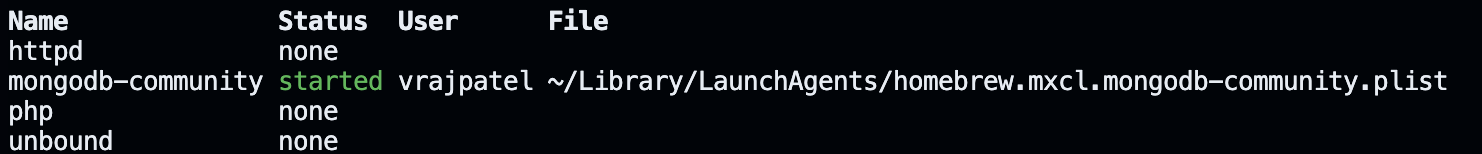
**Step 1: Install MongoDB**

Installed MongoDB using HOME BREW package Manager



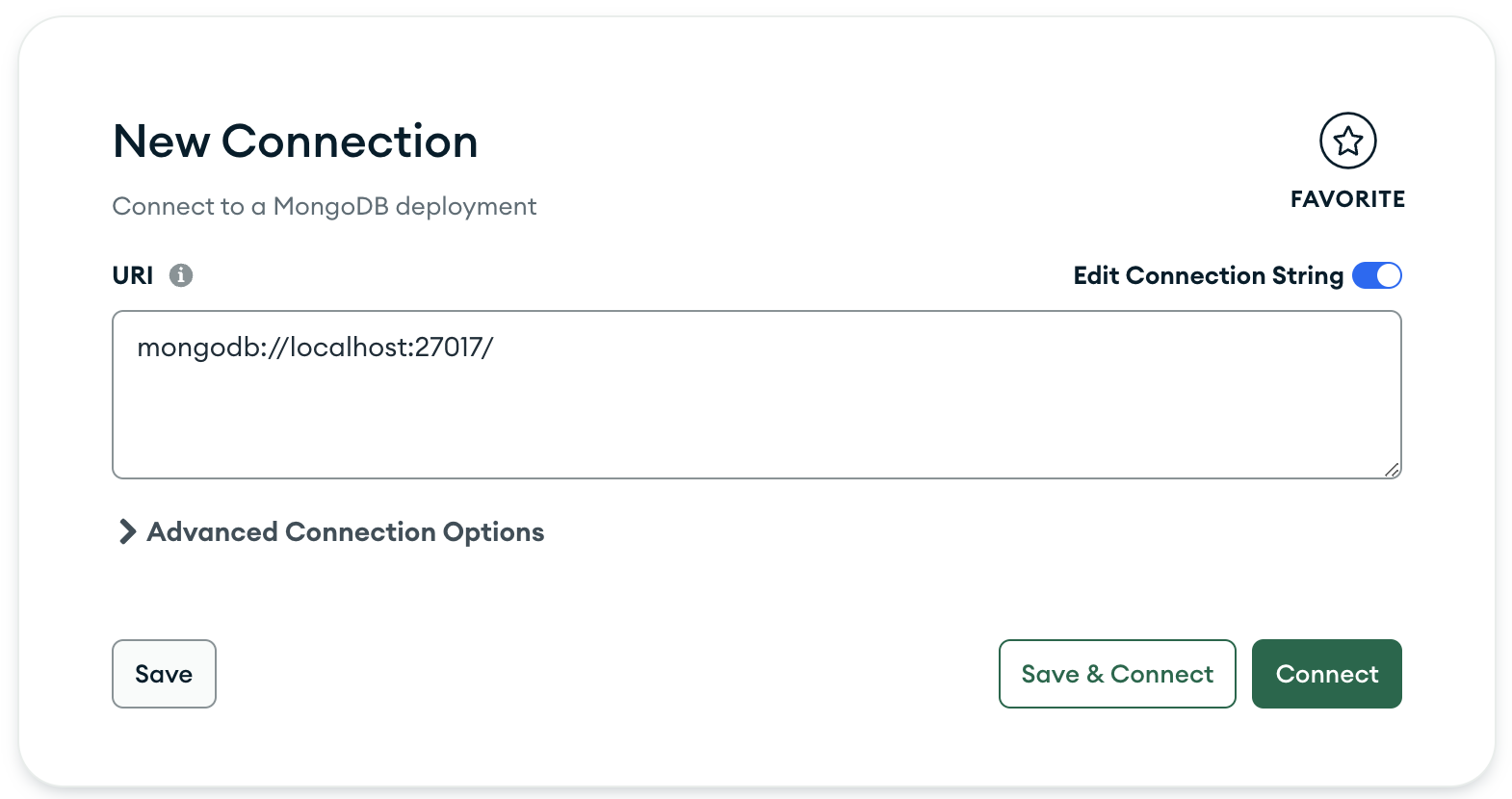
**Step 2: Start MongoDB Server**

mongoDB is Up and running and added in background Processes



**Step 3: Connect to MongoDB**

WE will connect with MongoDB



**Step 4: Connect from your Application**

* Step 1 = > npm install mongodb // will install all mongodb Drivers
* Step 2 => Connected my Application and created DATABASE mydb



**Script-**

const { MongoClient } = require('mongodb'); const uri = 'mongodb://localhost:27017/mydb'; const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true }); async function connectToMongoDB() { try { await client.connect(); console.log('Connected'); } catch (error) { console.error('Error connecting to MongoDB');; } } connectToMongoDB(); async function inserto(client, newdoc) { { const result = await client.db('mydb').collection('awt').insertOne(newdoc); console.log(`New listing created with the following id: ${result.insertedId}`); } } data = { name: "Patel Vraj Chetankumar", age: 21, city: "california" }; inserto(client, data);

**Added new Entry in mydb.awt**

name: "Patel Vraj Chetankumar", age: 21, city: "california"

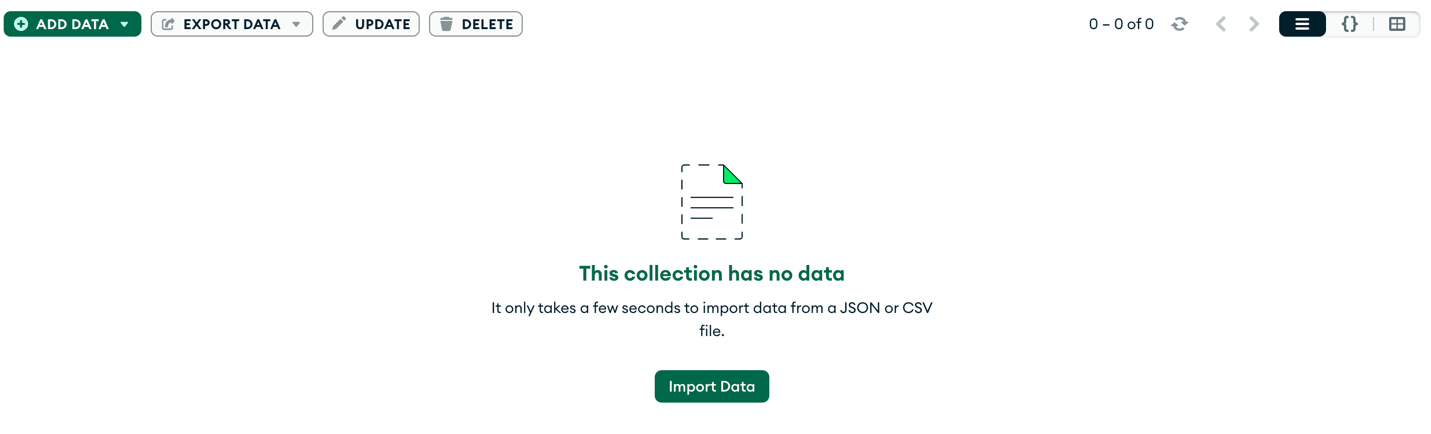
We Have successfully created Database {mydb.awt} and added new entries to it listing id : as objectID

Name : as string

Age : as int

City : as string

**Step 5: Performing Database Operations**

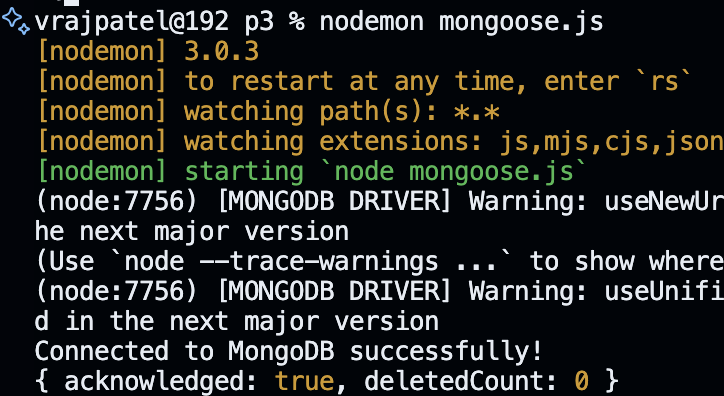


Lets us insert data into our mydb.kris instance with format of

* Name
* Email
* Age
* Registered Date

MAIN BASE SCRIPT

const mongoose = require('mongoose'); mongoose.connect('mongodb://localhost/mydb', { useNewUrlParser: true, useUnifiedTopology: true }) .then(() => console.log('Connected to MongoDB successfully!')) .catch((error) => console.error('Error connecting to MongoDB', error)); const krischema = new mongoose.Schema({ name: { type: String, required: true }, email: { type: String, required: true, unique: true }, age: { type: Number, default: 0 }, deregistered: { type: Date, default: Date.now } }); const obj = mongoose.model('kri', krischema);



SINGLE Entry in DB WITH Traditional APPROACH

const newobj = new obj(

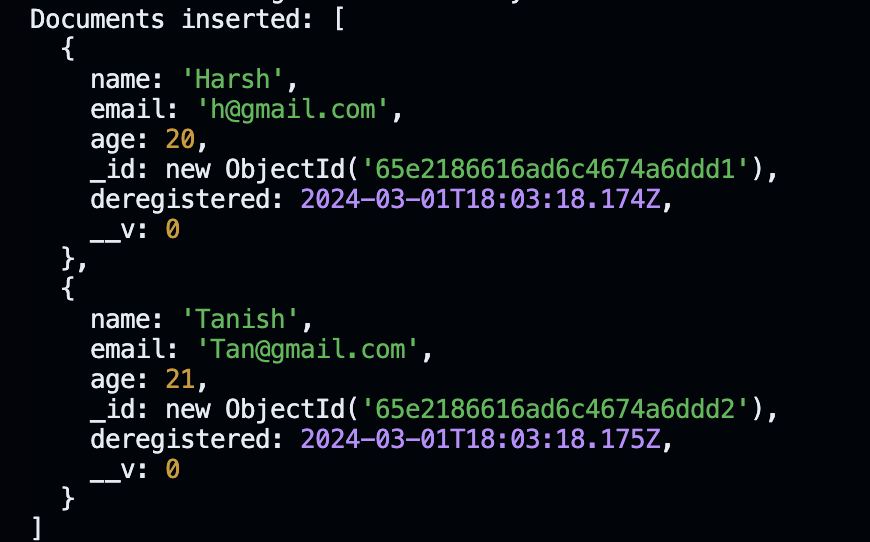
{ name: 'Vraj Patel', email: 'vraj.pce21@sot.pdpu.ac.in', age: 20 })

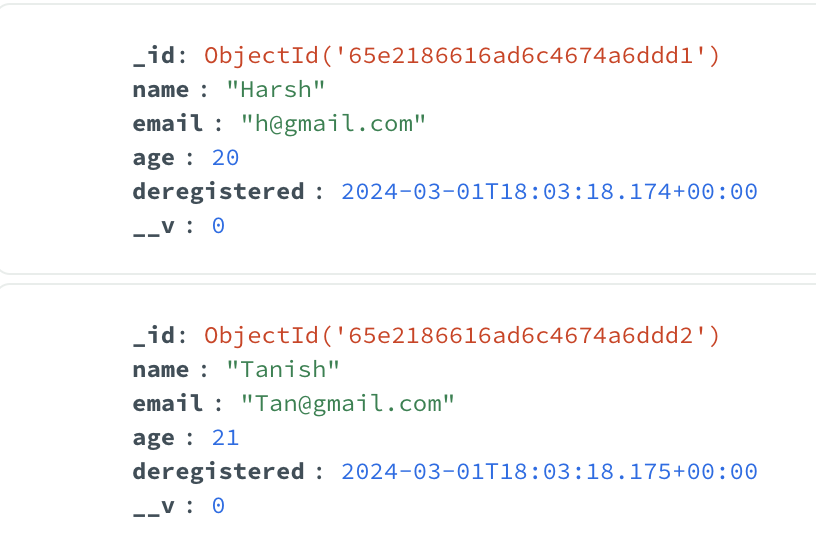
newobj.save()



Multiple Entry in DB WITH INSERT MANY APPROACH

*// Define an array containing the documents to insert* const newUsers = [{ name: 'Harsh', email: 'h@gmail.com', age: 20 }, { name: 'Tanish', email: 'Tan@gmail.com', age: 21 } ]; *// Insert many documents into the database* obj.insertMany(newUsers) .then((docs) => { console.log('Documents inserted:', docs); }) .catch((err) => { console.error('Error inserting documents:', err); });



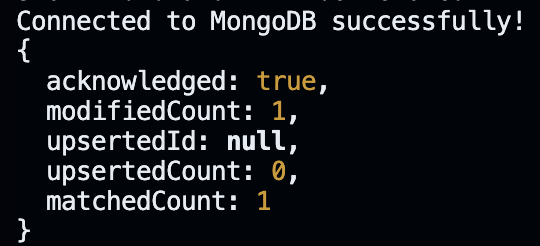


FIND USER BY NAME

obj.findOne({ name: 'Vraj Patel' }) .then((doc) => { if (doc) { console.log('Document found:', doc); } else { console.log('No document found with the given name.'); } }) .catch((err) => { console.error('Error finding document:', err); });



UPDATE ONE





obj.updateOne({ name: "Vraj Patel" }, { $set: { name: "Vraj Chetankumar Patel" } }) .then((docs) => { if (docs) { console.log(docs); } else { console.log("no such user exist"); } }).catch((err) => { console.log(err); })

UPDATE MANY

*// UPDATE MANY* *// obj.updateMany({age: 20},{$set:{ age: 21}})* *// .then((docs)=>{* *// if(docs) {* *// console.log(docs);* *// } else {* *// console.log("no such user exist");* *// }* *// }).catch((err)=>{* *// console.log(err);* *// })*



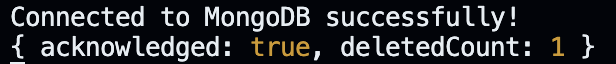


DELETE ONE

obj.deleteOne({ name: "Harsh" }) .then((docs) => { if (docs) { console.log(docs); } else { console.log("no such user exist"); } }).catch((err) => { console.log(err); })



DELETE MANY

obj.deleteMany({ name: "Tanish" }) .then((docs) => { if (docs) { console.log(docs); } else { console.log("no such user exist"); } }).catch((err) => { console.log(err); })

WE have Successfully installed Mongo DB and connected our data bases with our files. Then we proceed with our queries of insertion update delete find queries, and thus we have successfully completed our practical three.