CANARA ENGINEERING COLLEGE
Benjanapadavu, Mangaluru - 574219

LABORATORY MANUAL

MERN
(BDSL456C)

PREPARED BY

Mr. Nagaraj Gadagin
Assistant Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND

CANARA ENGINEERING COLLEGE

Sudhindra Nagar, Benjanapadavu,
Bantwal Taluk, D.K., Karnataka- 574 219

VISION OF THE INSTITUTE

To be an Engineering Institute of highest repute and produce world- class engineers catering to the needs of mankind.

MISSION OF THE INSTITUTE

- Provide the right environment to develop quality education for all, irrespective of caste, creed or religion to produce future leaders.
- Create opportunities for pursuit of knowledge and all-round development.
- Impart value education to students to build sense of integrity, honesty and ethics.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

VISION OF THE DEPARTMENT

To be identified as learning centre in the domain of Artificial Intelligent and Machine Learning education that delivers competent and professional engineers to meet the needs of industry, society and the nation.

MISSION OF THE DEPARTMENT

- To impart skill-based (Artificial Intelligent and Machine Learning) education through competent teaching-learning process.
- To establish a research and innovation ecosystem that provides solution for technological challenges of industry, society and the nation.
- To set-up industry-institute interface for overall development of students through practical internship and team work activities.
- To promote innovation and start-up culture among staff and students for addressing the challenges of needy.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

PROGRAM EDUCATIONAL OBJECTIVES

The graduates of BE-AIML program four years after graduation will

1. Design and develop learning-based intelligent systems in the field of artificial intelligent, machine learning, and allied engineering sectors.
2. Apply skills and knowledge of computer science to address relevant industry and societal problems or pursue higher education and research
3. Graduates will design and deploy software that meets the needs of individuals and the industries
4. Engage in lifelong learning, career advancement and adoption of changing professional and societal needs

PROGRAMME SPECIFIC OUTCOMES

1. Intelligent Systems: Select appropriate technologies to analyse, design, implement, and deployment of smart and intelligent systems
2. Contemporary Systems: Design, and development of efficient IT solutions for challenging issues through experiential learning

Laboratory Manual of MERN

The Laboratory Manual of MERN is an all-encompassing guide created for students who are studying and applying web application development technologies. This manual is a resource for conducting practical experiments related to Full Stack Web Application Development (MERN), providing students with hands-on experience in this fast-evolving field.

Purpose

The primary purpose of the Laboratory Manual of MERN is to provide students with a structured framework for learning and implementing MERN concepts and technology in real-world scenarios. Through a series of carefully designed experiments, students can explore the concepts, api's, tools, and applications of MERN, ultimately enhancing their understanding and proficiency in this cutting-edge technology.

Quality Assurance

The content of the Laboratory Manual of MERN has been developed and reviewed by experienced faculty members and subject matter experts in the field of Full Stack Web Development. Every effort has been made to ensure the accuracy, relevance, and reliability of the information presented in the manual, focusing on providing students with a high-quality learning experience.

MERN LABORATORY

Course Code	BDSL456C	Semester	IV
Teaching Hours/ Week (L:T:P:S)	0:0:2:0	CIE MARKS	50
Total Hours of Pedagogy	24	SEE MARKS	50
Credits	01	Total Marks	100

COURSE DESCRIPTION

The MERN Laboratory course immerses students in a dynamic learning environment focusing on the MERN (MongoDB, Express.js, React.js, Node.js) stack. Through hands-on experiments, students delve into full-stack development, understanding each component of the MERN stack. They gain knowledge and apply it in database management with MongoDB, server-side scripting with Node.js and Express.js, and front-end development with React.js.

COURSE OBJECTIVES

This course will enable the students to:

1. Understand and apply critical web development languages and tools to create dynamic and responsive web applications.
2. To build server-side applications using Node.js and Express
3. Develop user interfaces with React.js,
4. Manage data using MongoDB, and integrate these technologies to create full-stack apps
5. Understanding APIs and routing.

COURSE OUTCOMES

After completion of the course, the students will be able to:

1. Apply the fundamentals of MongoDB, such as data modelling, CRUD operations, and basic queries to solve given problem.
2. Use constructs of Express.js, including routing, software and constructing RESTful APIs to solve real world problems.
3. Develop scalable and efficient RESTful APIs using NodeJS.
4. Develop applications using React, including components, state, props, and JSX syntax.

SYLLABUS

SL. No.	Experiments	HOURS
1.	Using MongoDB, create a collection called transactions in database usermanaged (drop if it already exists) and bulk load the data from a json file, transactions.json. Upsert the record from the new file called transactions_upsert.json in Mongodb.	**2**
2.	Query MongoDB with Conditions: [Create appropriate collection with necessary documents to answer the query] a. Find any record where Name is Somu b. Find any record where total payment amount (Payment.Total) is 600. c. Find any record where price (Transaction.price) is between 300 to 500. d. Calculate the total transaction amount by adding up Payment.Total in all records.	**2**
3.	a. Write a program to check request header for cookies. b. Write node.js program to print the a car object properties, delete the second property and get length of the object.	**2**
4.	a. Read the data of a student containing usn, name, sem, year_of_admission from node js and store it in the mongodb b. For a partial name given in node js, search all the names from mongodb student documents created in Question(a)	**2**
5.	Implement all CRUD operations on a File System using Node JS.	**2**
6.	Develop the application that sends fruit name and price data from client side to Node.js server using Ajax.	**2**
7.	Develop an authentication mechanism with email_id and password using HTML and Express JS (POST method)	**2**
8.	Develop two routes: find_prime_100 and find_cube_100 which prints prime numbers less than 100 and cubes less than 100 using Express JS routing mechanism.	**2**
9.	Develop a React code to build a simple search filter functionality to display a filtered list based on the search query entered by the user.	**2**
10.	Develop a React code to collect data from rest API.	**2**

Suggested Learning Resources:

1. Vasan SubramanianPro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node, Apress; 1st ed. edition (1 April 2017)
2. Eddy Wilson Iriarte Koroliova,MERN Quick Start Guide, Packt Publishing (31 May 2018),
3. https://www.geeksforgeeks.org/mern-stack/
4. https://blog.logrocket.com/mern-stack-tutorial/

COURSE PRE-REQUISITES

- Proficiency in programming is crucial, preferably in web page development.

ASSESSMENT DETAILS (BOTH CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are 50 Marks.

The split-up of CIE marks for record/ journal and test are in the ratio 60:40.

- Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to 30 marks (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- The department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to 20 marks (40% of the maximum marks).

The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the

answer script to be strictly adhered to by the examiners. OR based on the course requirement evaluation rubrics shall be decided jointly by examiners.

- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
- General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)
- Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero

The minimum duration of SEE is 02 hours

COURSE CONTINUOUS AND COMPREHENSIVE ASSESSMENT PLAN

Sl. No.	Assessment Type	Maximum Marks	Minimum Marks	Assessment		
				Methods	Marks	Weightage (%)
1	CIE – CCAs	30	12	WRITEUP & CONDUCTION RECORD VIVA TOTAL	10 10 10 30	60
2	CIE - IAT	20	08	IAT (60:40) SCALEDOWN	100 20	40
	TOTAL CIE	**50**	**20**	**-**	**-**	**100**

CONTENT

Introduction

Full Stack Web Development

What is the Full Stack Web Development?

- Full stack web development refers to the building of web applications that encompass both frontend and backend components.
- A full-stack developer is proficient in frontend technologies, such as HTML, CSS, and JavaScript, as well as backend technologies, such as server-side programming languages, databases, and server management.

Technologies

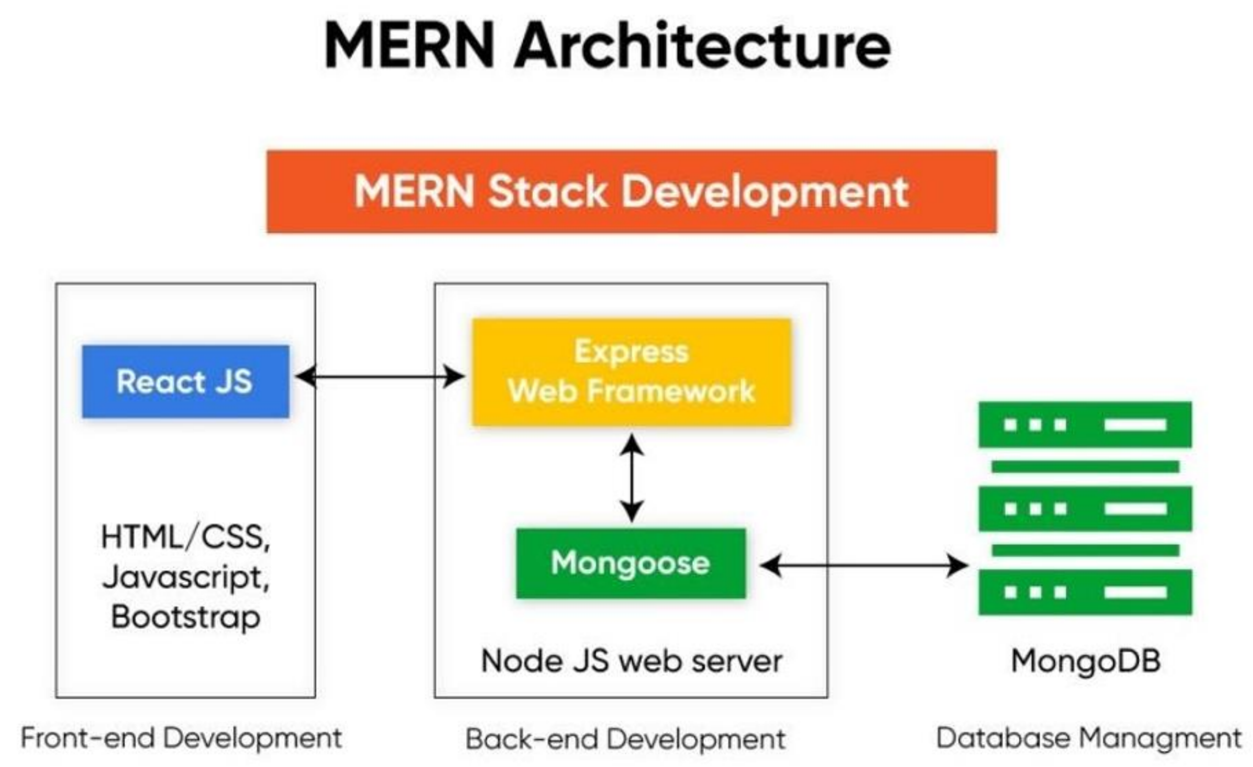
- **MERN (MongoDB, Express.js, React and Node.js)**
- **MEAN ((MongoDB, Express.js, Angular and Node.js)**
- **MEVN (MongoDB, Express.js, Vue and Node.js)**
- **LAMP (Linux, Apache, MySQL, PHP)**

MongoDB: A cross-platform document-oriented database program

Express.js: A web application framework for Node.js

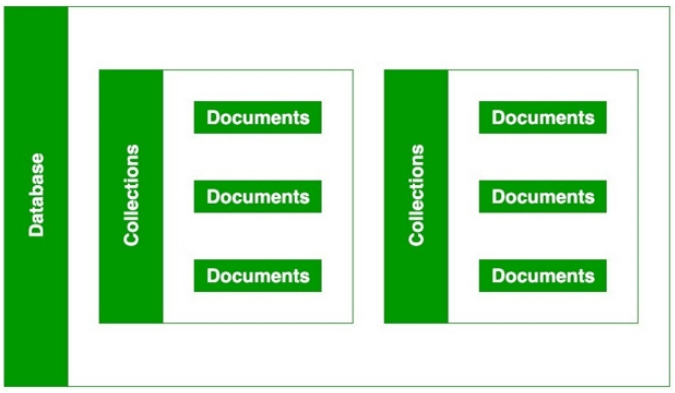
React: JavaScript library for building user interfaces

Node.js: An open source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser



What is it?

- MongoDB is a document-oriented NoSQL database system that provides high scalability, flexibility, and performance.
- It is categorized under the NoSQL database



MongoDB Features

- **Schema-less Database:**

  A Schema-less database means one collection can hold different types of documents in it. Or in other words, in the MongoDB database, a single collection can hold multiple documents and these documents may consist of the different numbers of fields, content, and size.

- **Document Oriented:**

  In MongoDB, all the data stored in the documents instead of tables like in RDBMS. In these documents, the data is stored in fields(key-value pair).

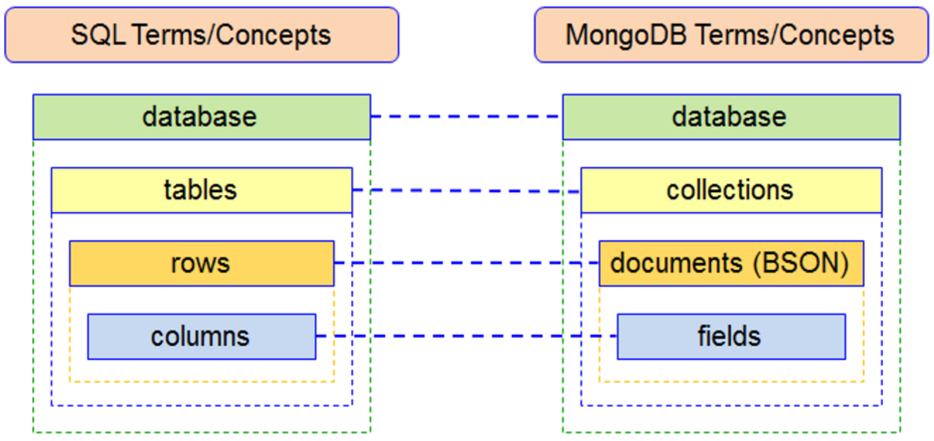
- **Scalability:**

  MongoDB provides horizontal scalability with the help of sharding. Sharding means to distribute data on multiple servers.

- **Indexing:**

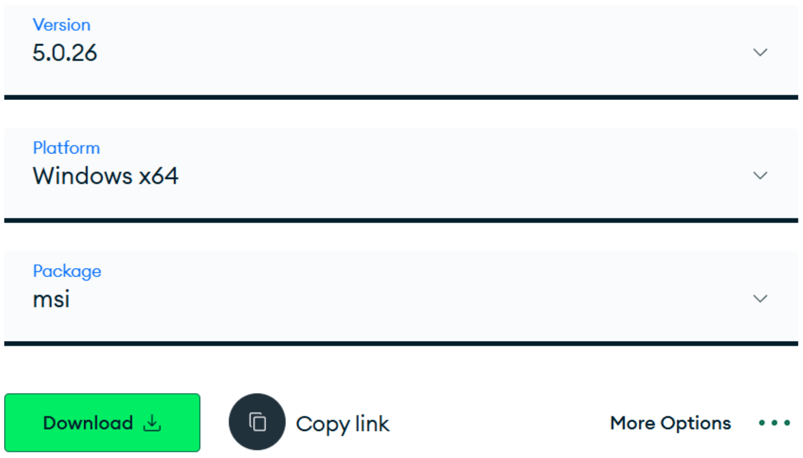
  Every field in the documents is indexed with primary and secondary indices this makes easier and takes less time to get or search data from the pool of the data.

Comparison between SQL & MongoDB Terms/Concepts



Installation of MongoDB

1. Go to the MongoDB Download Center to download the MongoDB Community Server. (Preferably V5.X)

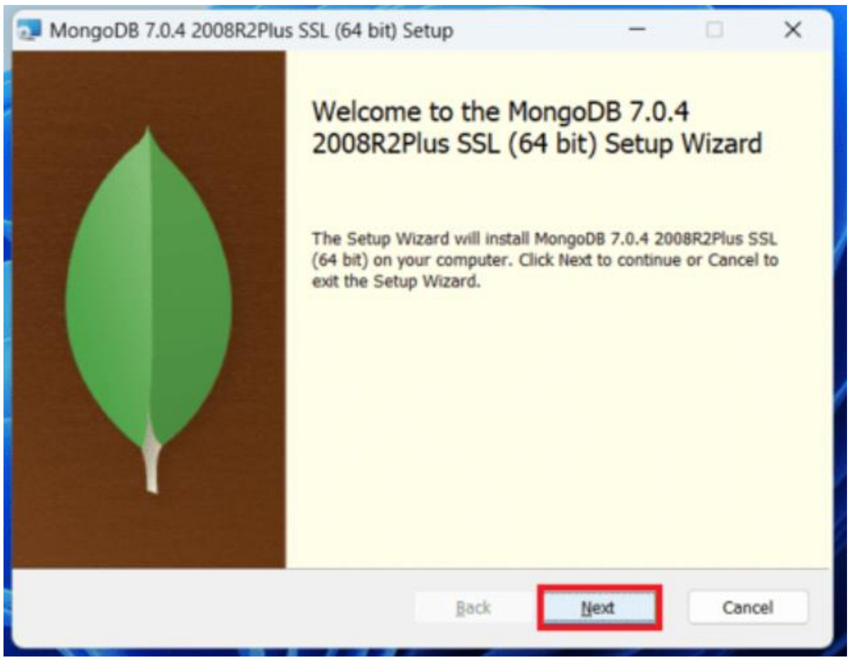


Version
5.0.26

Platform
Windows x64

Package
msi

2. When the download is complete open the msi file and click the *next button* in the startup screen:



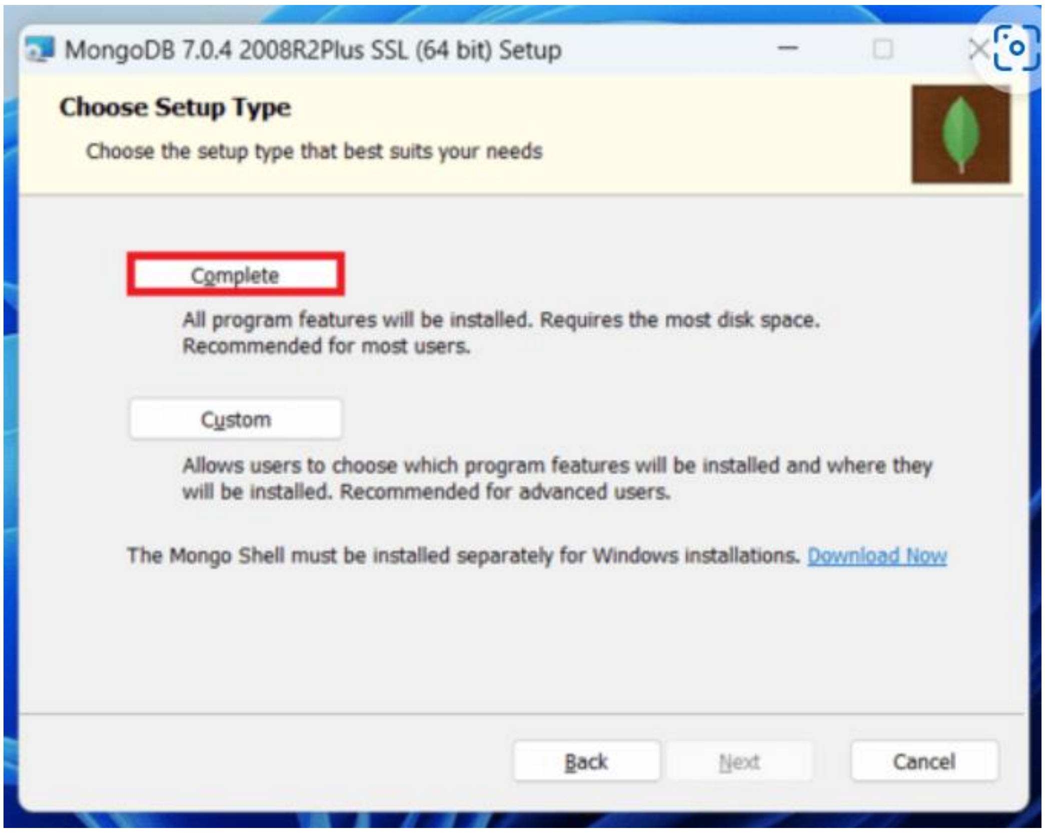
MongoDB 7.0.4 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 7.0.4 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 7.0.4 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

3. Now select install mongodb as a Service and accept the End-User License Agreement and click the next button , :

4. Now select the complete option to install all the program features. Here, if you can want to install only selected program features and want to select the location of the installation, then use the Custom option:


MongoDB 7.0.4 2008R2Plus SSL (64 bit) Setup

Choose Setup Type

Choose the setup type that best suits your needs

Complete

All program features will be installed. Requires the most disk space. Recommended for most users.

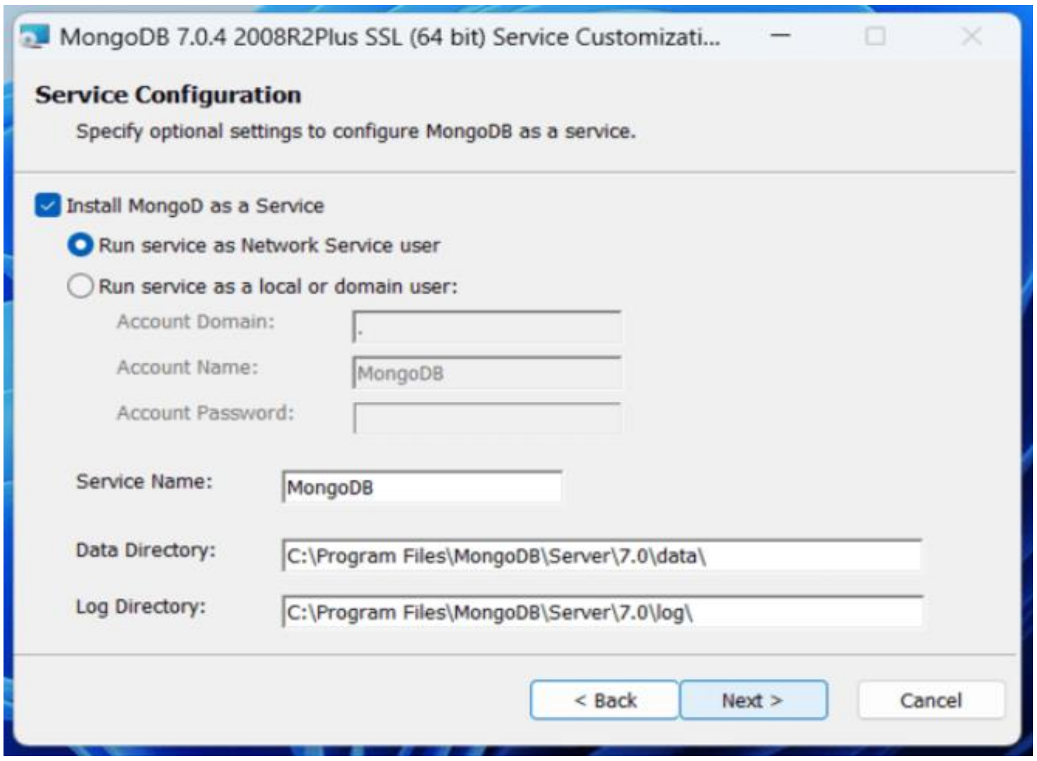
Custom

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

The Mongo Shell must be installed separately for Windows installations. Download Now

Back | Next | Cancel

5. Select "Run service as Network Service user" and copy the path of the data directory. Click Next:


MongoDB 7.0.4 2008R2Plus SSL (64 bit) Service Customizati...

Service Configuration

Specify optional settings to configure MongoDB as a service.

☑ Install MongoD as a Service

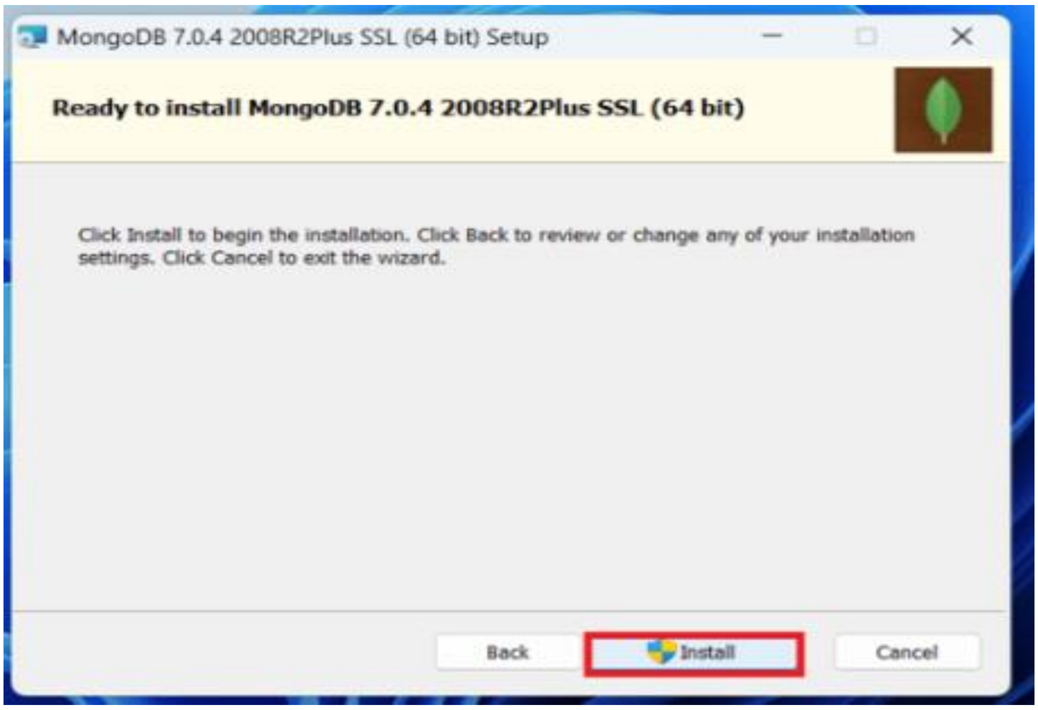
[✓ Run service as Network Service user]

Run service as a local or domain user:

Account Domain:	.
Account Name:	MongoDB
Account Password:	
Service Name:	MongoDB
Data Directory:	C:\Program Files\MongoDB\Server\7.0\data\
Log Directory:	C:\Program Files\MongoDB\Server\7.0\log\

< Back | Next > | Cancel

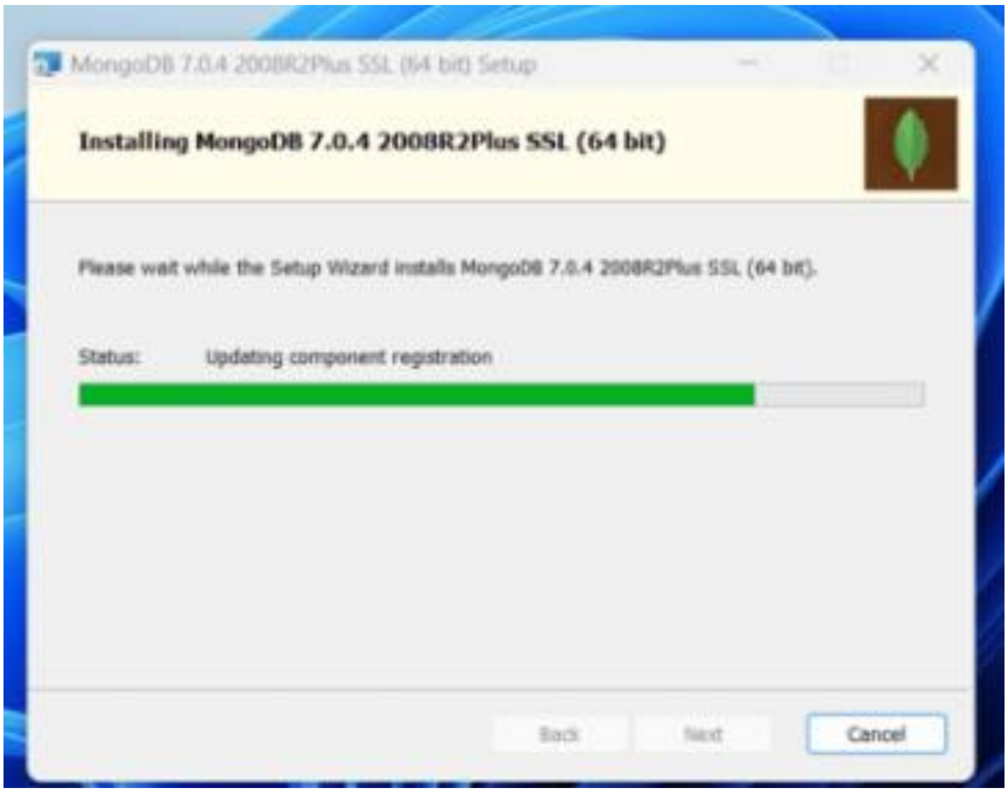
6. Click the *Install button* to start the MongoDB installation process:


MongoDB 7.0.4 2008R2Plus SSL (64 bit) Setup

Ready to install MongoDB 7.0.4 2008R2Plus SSL (64 bit)

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

7. After clicking on the install button installation of MongoDB begins:


MongoDB 7.0.4 2008R2Plus SSL (64 bit) Setup

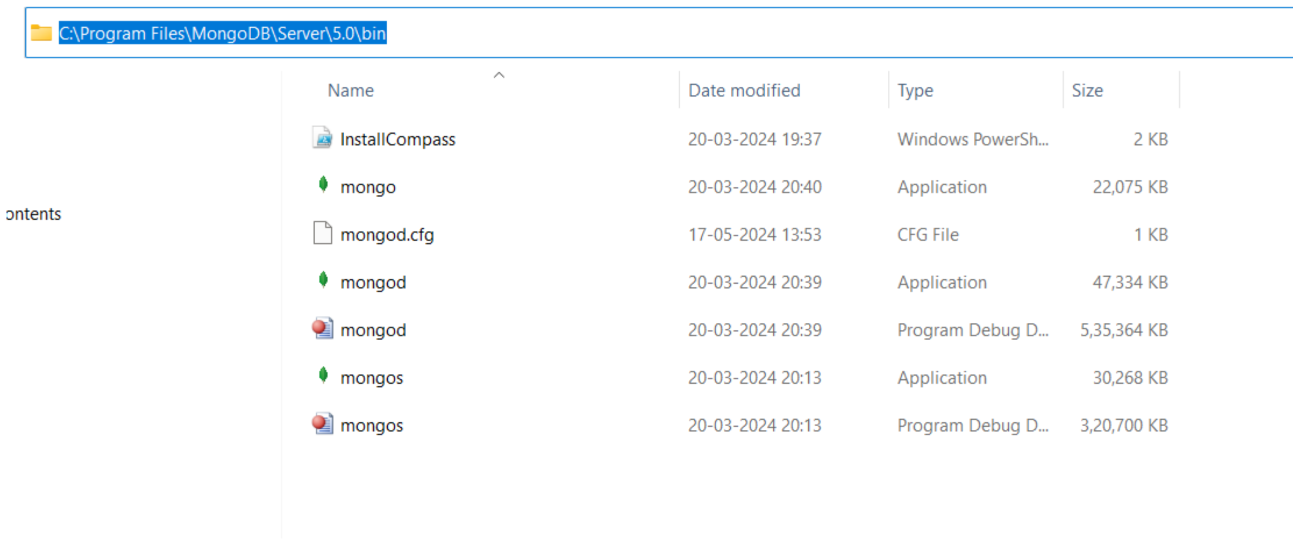
Installing MongoDB 7.0.4 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 7.0.4 2008R2Plus SSL (64 bit).

Status: Updating component registration

8. Now click the **Finish button** to complete the MongoDB installation process

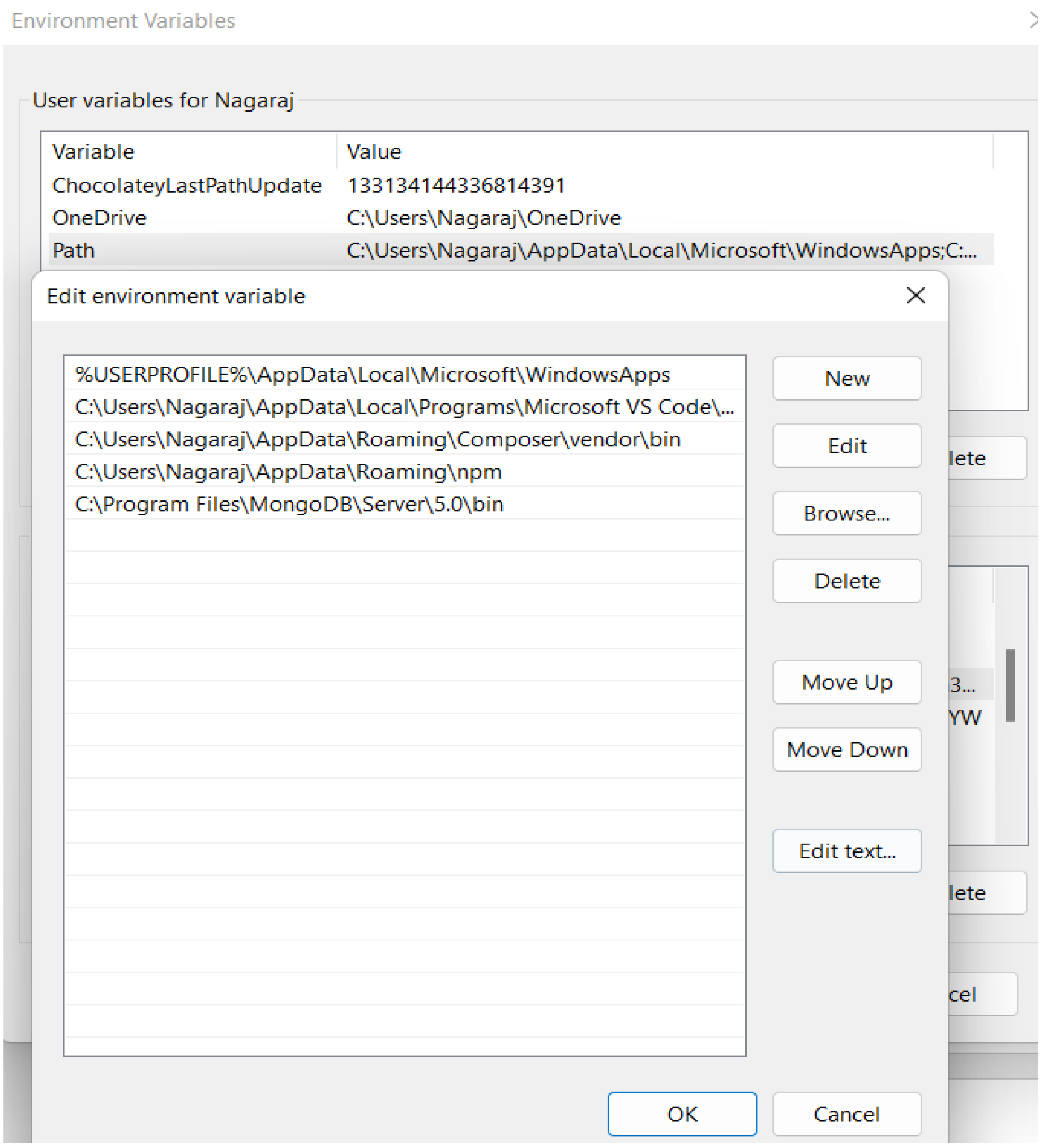
9. Now we go to the location where MongoDB installed in step 5 in your system and copy the bin path:



C:\Program Files\MongoDB\Server\5.0\bin

Name	Date modified	Type	Size
InstallCompass	20-03-2024 19:37	Windows PowerSh...	2 KB
mongo	20-03-2024 20:40	Application	22,075 KB
mongod.cfg	17-05-2024 13:53	CFG File	1 KB
mongod	20-03-2024 20:39	Application	47,334 KB
mongod	20-03-2024 20:39	Program Debug D...	5,35,364 KB
mongos	20-03-2024 20:13	Application	30,268 KB
mongos	20-03-2024 20:13	Program Debug D...	3,20,700 KB

10. Now, to create an environment variable open system properties >> Environment Variable >> System variable(user variable) >> path >> Edit Environment variable and paste the copied link to your environment system and click Ok:



Environment Variables

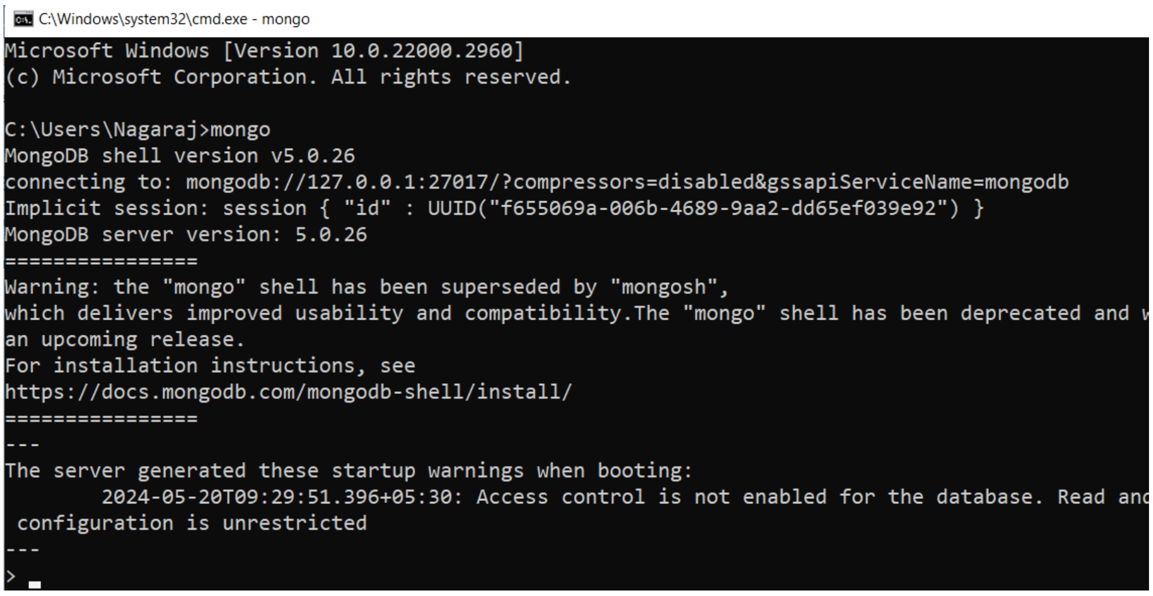
User variables for Nagaraj

Variable	Value
ChocolateyLastPathUpdate	133134144336814391
OneDrive	C:\Users\Nagaraj\OneDrive
Path	C:\Users\Nagaraj\AppData\Local\Microsoft\WindowsApps;C:...

Edit environment variable

```
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Nagaraj\AppData\Local\Programs\Microsoft VS Code\...
C:\Users\Nagaraj\AppData\Roaming\Composer\vendor\bin
C:\Users\Nagaraj\AppData\Roaming\npm
C:\Program Files\MongoDB\Server\5.0\bin
```

11. After setting the environment variable, we will run the Mongo. So, open the command prompt and run the following command: mongo



```
Microsoft Windows [Version 10.0.22000.2960]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nagaraj>mongo
MongoDB shell version v5.0.26
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("f655069a-006b-4689-9aa2-dd65ef039e92") }
MongoDB server version: 5.0.26
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and w
an upcoming release.
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
---
The server generated these startup warnings when booting:
        2024-05-20T09:29:51.396+05:30: Access control is not enabled for the database. Read and
 configuration is unrestricted
---
> 
```

MongoDB Commands

1. To view existing databases
   show dbs/show databases

2. To create a new database/switch to a database
   use database_name

3 to view current working database
   db

4. create a new collection
   db.createCollection("collection_name")

5 . to view existing collections in a database
   show collections/show tables

6. to view contents of a collection
   db.collection_name.find()

7. inserting documents into a collection
   db.collection_name.insert(json)
   db.collection_name.insertOne()
   db.collection_name.insertMany()

8. to view perticular collection/filter the data
   db.collection_name.find(filtering condition)

9. to update collection/documents

```
db.collection_name.update(filtering condition,value to be upadated)
db.collection_name.updateOne(filtering condition,value to be upadated)
db.collection_name.updateMany(filtering condition,value to be upadated)
```

10. to delete documents

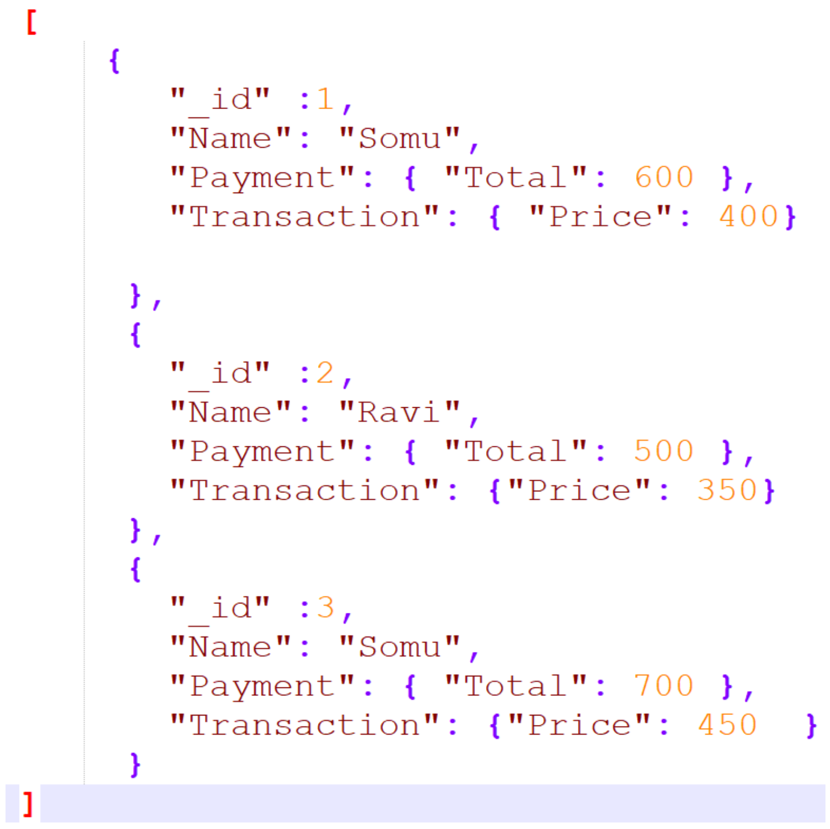
```
db.collection_name.delete(filtering condition)
db.collection_name.deleteOne(filtering condition)
db.collection_name.deleteMany(filtering condition)
```

Experiment 1:
Using MongoDB, create a collection called transactions in database usermanaged (drop if it already exists) and bulk load the data from a json file, transactions.json. Upsert the record from the new file called transactions_upsert.json in Mongodb.

Execution steps:
1: create two json file transactions.json and transactions_upsert.json containing set of documents.
2. Execute following mongodb commands in mongo shell.

```
// Step 1: Connect to MongoDB and switch to the usermanaged database
use usermanaged

// Step 2: Drop the transactions collection if it already exists
db.transactions.drop()

// Step 3: Create a new transactions collection
db.createCollection("transactions")

// Step 4: Bulk load the data from transactions.json into the transactions collection
mongoimport --db usermanaged --collection transactions --file transactions.json --jsonArray

// Step 5: Upsert the records from transactions_upsert.json into the transactions collection
mongoimport --db usermanaged --collection transactions --file transactions_upsert.json --jsonArray
--mode upsert
```

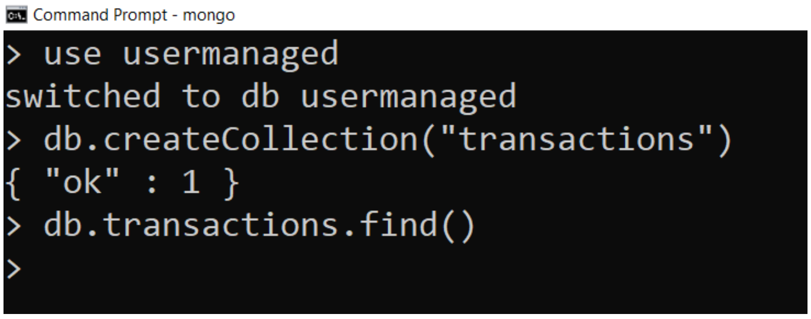
1. Create a transactions.json



```
[
  {
      "_id" :1,
      "Name": "Somu",
      "Payment": { "Total": 600 },
      "Transaction": { "Price": 400}

   },
   {
      "_id" :2,
      "Name": "Ravi",
      "Payment": { "Total": 500 },
      "Transaction": {"Price": 350}
   },
   {
      "_id" :3,
      "Name": "Somu",
      "Payment": { "Total": 700 },
      "Transaction": {"Price": 450   }
   }
]
```

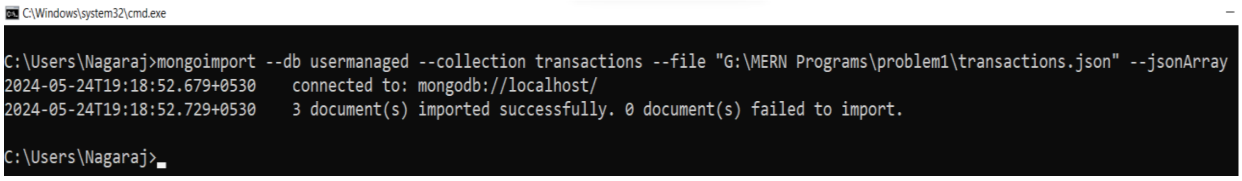
2. Open

Before loading data into the collection transactions, mongo shell prompts as there are no documents in it



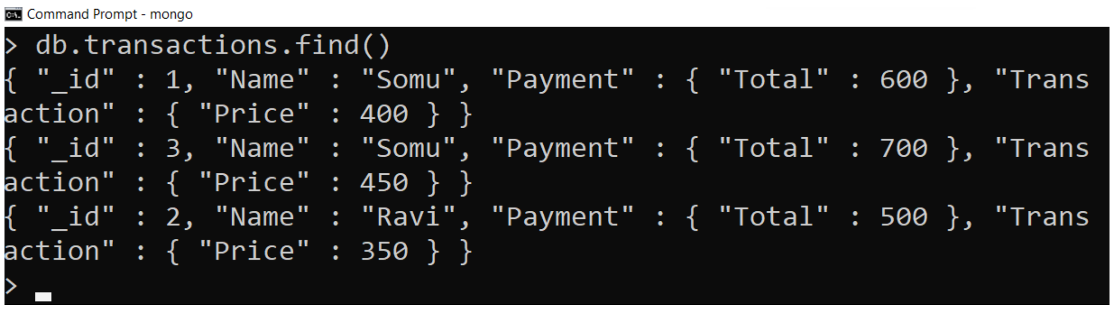
```
> use usermanaged
switched to db usermanaged
> db.createCollection("transactions")
{ "ok" : 1 }
> db.transactions.find()
>
```

3. **Load bulk data from transactions.json to database using mongoimport tool, open the command prompt enter the following**



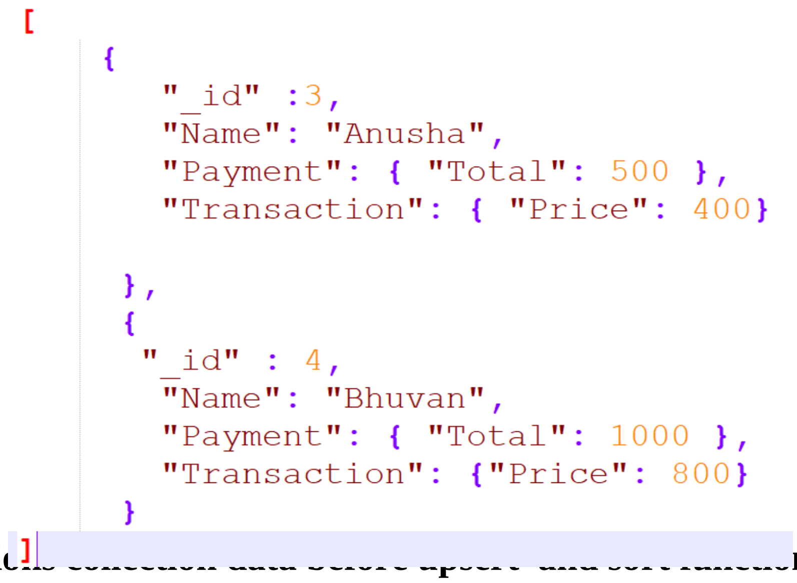
```
C:\Users\Nagaraj>mongoimport --db usermanaged --collection transactions --file "G:\MERN Programs\problem1\transactions.json" --jsonArray
2024-05-24T19:18:52.679+0530    connected to: mongodb://localhost/
2024-05-24T19:18:52.729+0530    3 document(s) imported successfully. 0 document(s) failed to import.

C:\Users\Nagaraj>_
```

4. **Documents in a collection after loading data.**



```
> db.transactions.find()
{ "_id" : 1, "Name" : "Somu", "Payment" : { "Total" : 600 }, "Trans
action" : { "Price" : 400 } }
{ "_id" : 3, "Name" : "Somu", "Payment" : { "Total" : 700 }, "Trans
action" : { "Price" : 450 } }
{ "_id" : 2, "Name" : "Ravi", "Payment" : { "Total" : 500 }, "Trans
action" : { "Price" : 350 } }
> _
```

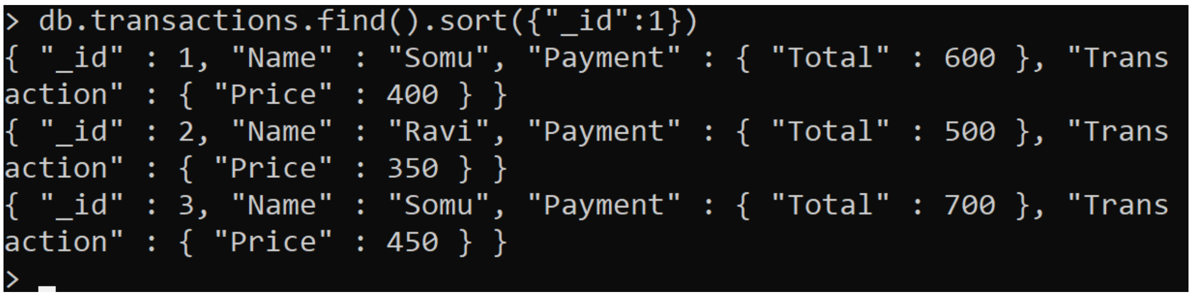
5. **create transactions_upsert.json file**



```
[
    {
        "_id" :3,
        "Name": "Anusha",
        "Payment": { "Total": 500 },
        "Transaction": { "Price": 400}

     },
     {
      "_id" : 4,
        "Name": "Bhuvan",
        "Payment": { "Total": 1000 },
        "Transaction": {"Price": 800}
     }
]
```

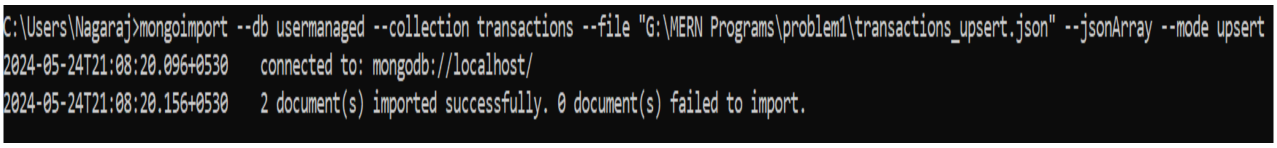
6. **transactions collection data before upsert and sort function sorts documents**

in ascending order based on id _



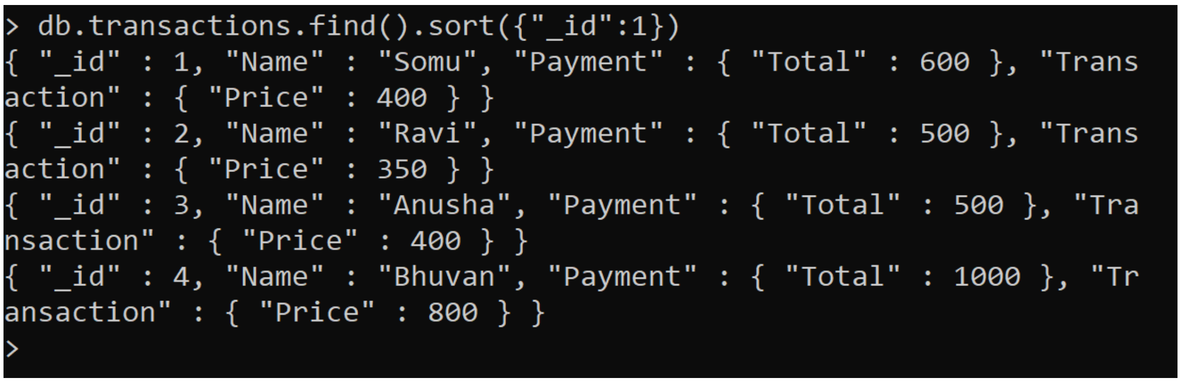
```
> db.transactions.find().sort({"_id":1})
{ "_id" : 1, "Name" : "Somu", "Payment" : { "Total" : 600 }, "Trans
action" : { "Price" : 400 } }
{ "_id" : 2, "Name" : "Ravi", "Payment" : { "Total" : 500 }, "Trans
action" : { "Price" : 350 } }
{ "_id" : 3, "Name" : "Somu", "Payment" : { "Total" : 700 }, "Trans
action" : { "Price" : 450 } }
> _
```

7. upsert the data from transactions_upsert.json



```
C:\Users\Nagaraj>mongoimport --db usermanaged --collection transactions --file "G:\MERN Programs\problem1\transactions_upsert.json" --jsonArray --mode upsert
2024-05-24T21:08:20.096+0530    connected to: mongodb://localhost/
2024-05-24T21:08:20.156+0530    2 document(s) imported successfully. 0 document(s) failed to import.
```

8. transactions collection data after upsert



```
> db.transactions.find().sort({"_id":1})
{ "_id" : 1, "Name" : "Somu", "Payment" : { "Total" : 600 }, "Trans
action" : { "Price" : 400 } }
{ "_id" : 2, "Name" : "Ravi", "Payment" : { "Total" : 500 }, "Trans
action" : { "Price" : 350 } }
{ "_id" : 3, "Name" : "Anusha", "Payment" : { "Total" : 500 }, "Tra
nsaction" : { "Price" : 400 } }
{ "_id" : 4, "Name" : "Bhuvan", "Payment" : { "Total" : 1000 }, "Tr
ansaction" : { "Price" : 800 } }
>
```

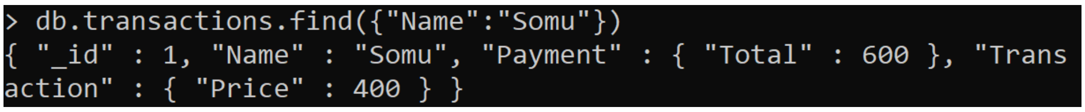
Experiment 2:

Query MongoDB with Conditions: [Create appropriate collection with necessary documents to answer the query]

a. Find any record where Name is Somu
b. Find any record where total payment amount (Payment.Total) is 600.
c. Find any record where price (Transaction.price) is between 300 to 500.
d. Calculate the total transaction amount by adding up Payment.Total in all records.

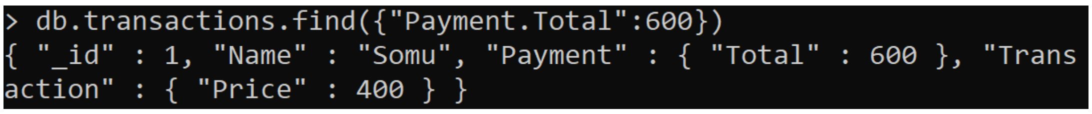
Queries:

a. Find any record where Name is Somu
db.transactions.find({ "Name": "Somu" })



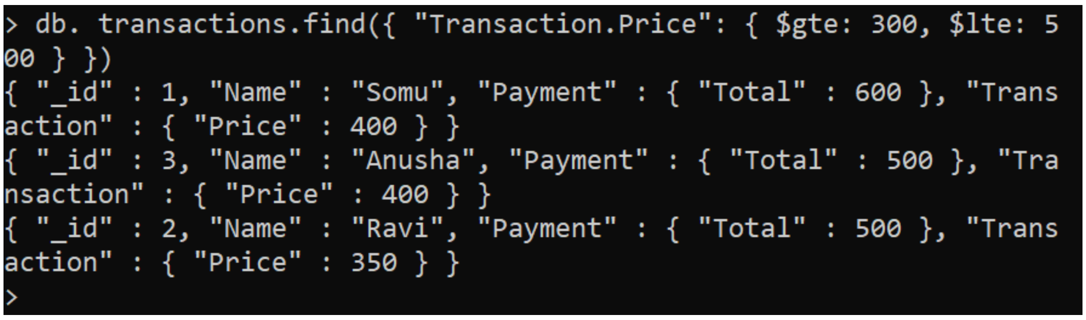
```
> db.transactions.find({"Name":"Somu"})
{ "_id" : 1, "Name" : "Somu", "Payment" : { "Total" : 600 }, "Trans
action" : { "Price" : 400 } }
```

b. Find any record where total payment amount (Payment.Total) is 600.
db.transactions.find({ "Payment.Total": 600 })



```
> db.transactions.find({"Payment.Total":600})
{ "_id" : 1, "Name" : "Somu", "Payment" : { "Total" : 600 }, "Trans
action" : { "Price" : 400 } }
```

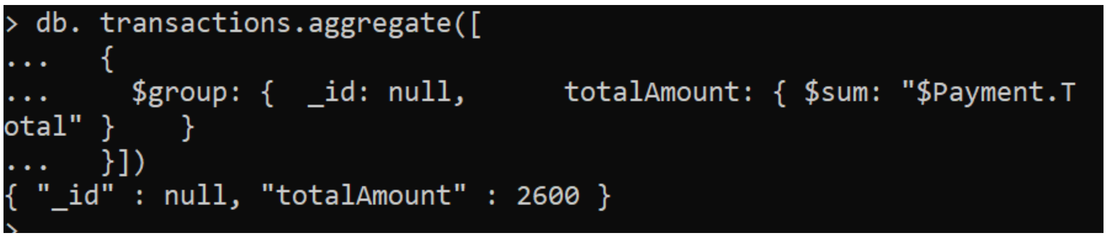
c. Find any record where price (Transaction.price) is between 300 to 500.
db. transactions.find({ "Transaction.Price": { $gte: 300, $lte: 500 } })



```
> db. transactions.find({ "Transaction.Price": { $gte: 300, $lte: 5
00 } })
{ "_id" : 1, "Name" : "Somu", "Payment" : { "Total" : 600 }, "Trans
action" : { "Price" : 400 } }
{ "_id" : 3, "Name" : "Anusha", "Payment" : { "Total" : 500 }, "Tra
nsaction" : { "Price" : 400 } }
{ "_id" : 2, "Name" : "Ravi", "Payment" : { "Total" : 500 }, "Trans
action" : { "Price" : 350 } }
>
```

d. Calculate the total transaction amount by adding up Payment.Total in all records.

```
db. transactions.aggregate([
 {
  $group: {  _id: null,     totalAmount: { $sum: "$Payment.Total" }    }
 }])
```



```
> db. transactions.aggregate([
...   {
...     $group: {  _id: null,      totalAmount: { $sum: "$Payment.T
otal" }     }
...   }])
{ "_id" : null, "totalAmount" : 2600 }
>
```

Experiment 3:

a. Write a program to check request header for cookies.
b. Write node.js program to print the car object properties, delete the second property and get length of the object.

a. Write a program to check request header for cookies.

```
const http = require("http");

// Create a server
const server = http.createServer((req, res) => {
  // Get request headers
  const headers = req.headers;
  console.log(headers, ".....");
  // Check if 'cookie' header exists
  if (headers.cookie) {
    console.log("Cookies:", headers.cookie);
  } else {
    console.log("No cookies found in the request header.");
  }

  res.end("Check console for cookies information.");
});

// Listen on port 3000
server.listen(3000, () => {
  console.log("Server running at http://localhost:3000/");
});
```

b. Write node.js program to print the car object properties, delete the second property and get length of the object.

```
// Define a car object
const car = {
    make: 'Toyota',
    model: 'Camry',
    year: 2020,
    color: 'blue'
};

// Print the properties of the car object
console.log('Original car object:');
for (const property in car) {
    console.log(`${property}: ${car[property]}`);
}

// Delete the second property
const secondProperty = Object.keys(car)[1];
delete car[secondProperty];

console.log('\nCar object after deleting the second property:');
```

```
for (const property in car) {
   console.log(`${property}: ${car[property]}`);
}
```

// Get the length of the object

```
const length = Object.keys(car).length;
console.log(`\nLength of the car object after deleting the second property: ${length}`);
```

Experiment 4:

a. Read the data of a student containing usn, name, sem, year_of_admission from node js and store it in the mongodb
b. For a partial name given in node js, search all the names from mongodb student documents created in Question(a)

a. Read the data of a student containing usn, name, sem, year_of_admission from node js and store it in the mongodb.

```
//studentdata.json file

[
  {
    "usn": "4CB22AI002",
    "name": "Bhuvan",
    "sem": 3,
    "year_of_admission": 2022
  },
  {
    "usn": "4CB22AI004",
    "name": "CHANDAN",
    "sem": 4,
    "year_of_admission": 2021
  },
  {
    "usn": "4CB22AI025",
    "name": "HARSHITA",
    "sem": 2,
    "year_of_admission": 2023
  }
]
```

```
//program.js
const express = require("express");
const MongoClient = require("mongodb").MongoClient;
const fs = require("fs");

const app = express();
const port = 3000;

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

// Database Name
const dbName = "studentDB";

// File path for student data
const studentDataFilePath = "studentData.json";

// Middleware to parse JSON requests
app.use(express.json());

// API endpoint to store student data in MongoDB
app.post("/api/students", (req, res) => {
  // Read student data from the request body
  const studentData = req.body;

  // Connect to MongoDB
  MongoClient.connect(url, function (err, client) {
    if (err) {
      console.error("Failed to connect to MongoDB:", err);
      res.status(500).json({ error: "Failed to connect to MongoDB" });
      return;
    }

    console.log("Connected successfully to server");

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

    // Insert student data into MongoDB collection
    collection.insertMany(studentData, function (err, result) {
      if (err) {
        console.error("Error inserting documents:", err);
        res.status(500).json({ error: "Error inserting documents" });
      } else {
        console.log(
          `Inserted ${result.insertedCount} documents into the collection`
        );
        res
          .status(201)
          .json({ message: `Inserted ${result.insertedCount} documents` });
      }
      client.close();
    });
  });
});

// Start the server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

b. For a partial name given in node js, search all the names from mongodb student documents created in Question(a).

```
const express = require("express");
const MongoClient = require("mongodb").MongoClient;

const app = express();
const port = 3000;

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

// Database Name
const dbName = "studentDB";

// Middleware to parse JSON requests
app.use(express.json());

// API endpoint to search students by partial name
app.get("/api/students", (req, res) => {
  // Get the partial name from the query parameters
  const partialName = req.query.partialName;

  // Connect to MongoDB
  MongoClient.connect(url, function (err, client) {
    if (err) {
      console.error("Failed to connect to MongoDB:", err);
      res.status(500).json({ error: "Failed to connect to MongoDB" });
      return;
    }

    console.log("Connected successfully to server");

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

    // Search students by partial name
    collection
      .find({ name: { $regex: partialName, $options: "i" } })
      .toArray(function (err, result) {
        if (err) {
          console.error("Error searching documents:", err);
          res.status(500).json({ error: "Error searching documents" });
        } else {
          console.log("Students matching the partial name:");
          console.log(result);
          res.status(200).json(result);
        }
        client.close();
      });
  });
});

// Start the server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Experiment 5:
Implement all CRUD operations on a File System using Node JS

```
/*
!run one function at a time
*/

const fs = require("fs");

const createFile = (fileName, data) => {
  fs.writeFile(fileName, data, (err) => {
    if (err) {
      console.error("Error creating file:", err);
    } else {
      console.log("File created successfully.");
    }
  });
};

// createFile("example.txt", "This is a sample text file.");

const readFile = (fileName) => {
  fs.readFile(fileName, "utf8", (err, data) => {
    if (err) {
      console.error("Error reading file:", err);
    } else {
      console.log("File content:");
      console.log(data);
    }
  });
};

// readFile("example.txt");

const appendFile = (fileName, newData) => {
  fs.appendFile(fileName, newData, (err) => {
    if (err) {
      console.error("Error appending to file:", err);
    } else {
      console.log("Data appended to file successfully.");
    }
  });
};

// appendFile("example.txt", "\nAdditional data appended.");

const updateFile = (fileName, newData) => {
  fs.writeFile(fileName, newData, (err) => {
    if (err) {
      console.error("Error updating file:", err);
    } else {
      console.log("File updated successfully.");
    }
  });
};

// updateFile("example.txt", "This is updated content.");

const deleteFile = (fileName) => {
  fs.unlink(fileName, (err) => {
    if (err) {
      console.error("Error deleting file:", err);
    } else {
      console.log("File deleted successfully.");
    }
  });
};

// deleteFile("example.txt");
```

Experiment 6:
Develop the application that sends fruit name and price data from client side to Node.js server using Ajax.

//Client side
//index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

//App.js

```
import React, { useState } from "react";
import axios from "axios";

function App() {
  const [name, setName] = useState("");
  const [price, setPrice] = useState("");
  const url = "http://localhost:5000";
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post(`${url}/api/fruits`, { name, price });
      console.log("Data sent successfully");
      // Optionally, you can reset the form fields here
    } catch (error) {
      console.error("Error sending data:", error);
    }
  };

  return (
    <div>
      <h1>Fruit Data Form</h1>
      <form onSubmit={handleSubmit}>
        <label>
          Fruit Name:
          <input
            type="text"
            value={name}
            onChange={(e) => setName(e.target.value)}
          />
        </label>
        <br />
        <label>
          Price:
          <input
            type="text"
            value={price}
            onChange={(e) => setPrice(e.target.value)}
          />
        </label>
        <br />
        <button type="submit">Submit</button>
      </form>
    </div>
  );
}

export default App;
```

//server side

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const app = express();
const PORT = 5000;

app.use(bodyParser.json());
app.use(cors());
// Handle POST request to '/api/fruits'
app.post("/api/fruits", (req, res) => {
  const { name, price } = req.body;
  console.log(`Received data from client: Name - ${name}, Price - ${price}`);
  // Here, you can process the data (e.g., save to database) and send back a response if
needed
  res.status(200).send("Data received successfully");
});

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Experiment 7:
Develop an authentication mechanism with email_id and password using HTML and Express JS (POST method)

```
//client side
//index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);


//App.js
import React, { useState } from "react";
import axios from "axios";

function App() {
  const [registerData, setRegisterData] = useState({ email: "", password: "" });
  const [loginData, setLoginData] = useState({ email: "", password: "" });
  const [message, setMessage] = useState("");

  const handleRegister = async (e) => {
```

```
  e.preventDefault();
  try {
    await axios.post("http://localhost:3000/register", registerData);
    setMessage("User registered successfully");
  } catch (error) {
    setMessage("Error registering user");
    console.error("Error registering user:", error);
  }
};

const handleLogin = async (e) => {
  e.preventDefault();
  try {
    await axios.post("http://localhost:3000/login", loginData);
    setMessage("Login successful");
  } catch (error) {
    setMessage("Invalid credentials");
    console.error("Error logging in user:", error);
  }
};

return (
  <div>
    <h2>User Registration</h2>
    <form onSubmit={handleRegister}>
      <label>Email:</label>
      <input
        type="email"
        value={registerData.email}
        onChange={(e) =>
          setRegisterData({ ...registerData, email: e.target.value })
        }
        required
      />
      <br />
      <label>Password:</label>
      <input
        type="password"
        value={registerData.password}
        onChange={(e) =>
          setRegisterData({ ...registerData, password: e.target.value })
        }
        required
      />
      <br />
      <button type="submit">Register</button>
    </form>

    <h2>User Login</h2>
    <form onSubmit={handleLogin}>
      <label>Email:</label>
      <input
```

```
          type="email"
          value={loginData.email}
          onChange={(e) =>
            setLoginData({ ...loginData, email: e.target.value })
          }
          required
        />
        <br />
        <label>Password:</label>
        <input
          type="password"
          value={loginData.password}
          onChange={(e) =>
            setLoginData({ ...loginData, password: e.target.value })
          }
          required
        />
        <br />
        <button type="submit">Login</button>
      </form>

      {message && <p>{message}</p>}
    </div>
  );
}

export default App;
```

//Server side

```
const express = require("express");
const bcrypt = require("bcrypt");
const bodyParser = require("body-parser");
const cors = require("cors");

const app = express();
const PORT = 3000;

// Middleware to parse JSON requests
app.use(bodyParser.json());
app.use(cors());
// Sample user data (replace this with your database)
const users = [];

// Route to handle user registration
app.post("/register", async (req, res) => {
  try {
    const { email, password } = req.body;

    // Check if email is already registered
    const existingUser = users.find((user) => user.email === email);
    if (existingUser) {
      return res.status(400).send("Email already exists");
    }

    // Hash the password
    const hashedPassword = await bcrypt.hash(password, 10);

    // Store the user data (in memory for now, replace this with database storage)
    users.push({ email, password: hashedPassword });

    res.status(201).send("User registered successfully");
  } catch (error) {
    console.error("Error registering user:", error);
    res.status(500).send("Internal server error");
  }
});

// Route to handle user login
app.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;

    // Find the user by email
    const user = users.find((user) => user.email === email);
    if (!user) {
      return res.status(401).send("Invalid credentials");
    }

    // Compare the password
    const passwordMatch = await bcrypt.compare(password, user.password);
    if (!passwordMatch) {
      return res.status(401).send("Invalid credentials");
    }

    res.status(200).send("Login successful");
  } catch (error) {
    console.error("Error logging in user:", error);
    res.status(500).send("Internal server error");
  }
});

// Start the server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Develop two routes: find_prime_100 and find_cube_100 which prints prime numbers less than 100 and cubes less than 100 using Express JS routing mechanism

```
const express = require("express");
const app = express();
const PORT = 3000;

// Route to find prime numbers less than 100
app.get("/find_prime_100", (req, res) => {
const primes Experiment 8:
= findPrimes(100);
  res.json({ primes });
});

// Route to find cubes less than 100
app.get("/find_cube_100", (req, res) => {
  const cubes = findCubes(100);
  res.json({ cubes });
});

// Function to find prime numbers less than n
function findPrimes(n) {
  const primes = [];
  for (let i = 2; i < n; i++) {
    let isPrime = true;
    for (let j = 2; j <= Math.sqrt(i); j++) {
      if (i % j === 0) {
        isPrime = false;
        break;
      }
    }
    if (isPrime) {
      primes.push(i);
    }
  }
  return primes;
}

// Function to find cubes less than n
function findCubes(n) {
  const cubes = [];
  for (let i = 1; i < n; i++) {
    const cube = i * i * i;
    if (cube < n) {
      cubes.push(cube);
    } else {
      break;
    }
  }
  return cubes;
}

// Start the server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Experiment 9:
Develop a React code to build a simple search filter functionality to display a filtered list based on the search query entered by the user.

//index.js
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

//App.js
```
import React from "react";
import SearchFilter from "./SearchFilter";

function App() {
  return (
    <div>
      <SearchFilter />
    </div>
  );
}

export default App;
```

//SearchFilter.jsx

```
import React, { useState } from "react";
import jsonData from "./data.json"; // Import JSON data

function SearchFilter() {
  const [searchQuery, setSearchQuery] = useState("");
  const [filteredItems, setFilteredItems] = useState([]);

  // Function to handle search input change
  const handleSearchChange = (e) => {
    const query = e.target.value.toLowerCase();
    setSearchQuery(query);
    // Filter the items based on the search query
    const filtered = jsonData.filter((item) =>
      item.toLowerCase().includes(query)
    );
    setFilteredItems(filtered);
  };

  return (
    <div>
      <h2>Search Filter</h2>
      <input
        type="text"
        placeholder="Search..."
        value={searchQuery}
        onChange={handleSearchChange}
      />
      <ul>
        {filteredItems.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );
}

export default SearchFilter;
```

Experiment 10:
Develop a React code to collect data from rest API.

//index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

//App.js

```
import React, { useState, useEffect } from "react";
import axios from "axios";
```

```
function DataCollector() {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetchData();
  }, []);

  const fetchData = async () => {
    try {
      const response = await axios.get(
        "https://dummy.restapiexample.com/api/v1/employees"
      );
      setData(response.data.data);
    } catch (error) {
      console.error("Error fetching data:", error);
    }
  };

  return (
    <div>
      <h2>Data Collection</h2>
      <button onClick={fetchData}>Fetch Data</button>
      <ul>
        {data.map((item, index) => (
          <li key={index}>{item.employee_name}</li>
        ))}
      </ul>
    </div>
  );
}

export default DataCollector;
```