Batch: D-2	**Roll No.:** 16010122151
Experiment / assignment / tutorial No.____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of MongoDB, Node.js and Express js.

AIM: Implementation of MongoDB, Node.js and Express js.

Problem Definition:

Design and develop a backend for a web application that connects to a MongoDB database using Node.js and Express.js. The system should handle HTTP requests, interact with the database to store and retrieve data, and return appropriate responses to the client.

Resources used:

- Node.js for the backend server
- Express.js as the web framework
- MongoDB as the database for storing data
- Mongoose for handling MongoDB connections and schema validation in Node.js
- Postman for testing API endpoints

Expected OUTCOME of Experiment:

CO 4: Test the concepts and components of various front-end, back-end web app

Books/ Journals/ Websites referred:

1. Shelly Powers Learning Node O' Reilly 2 nd Edition, 2016.

Pre Lab/ Prior Concepts:

Write details about the following content

- Mongo DB

MongoDB is a NoSQL, document-oriented database that stores data in flexible, JSON-like documents. It is schema-less, meaning each document can have a different structure. It is widely used for its scalability and flexibility.

- Connection using node js Express js And MongoDB

To connect to a MongoDB database from Node.js, the mongoose library is typically used. Express.js provides routing and middleware, which makes it easy to create APIs.

Steps to establish the connection:

- Install necessary libraries (express, mongoose, body-parser).
- Set up a Node.js server using Express.
- Use Mongoose to connect to the MongoDB database.
- Create schema and model for the MongoDB collections.
- Create Express routes to handle requests like GET, POST, PUT, DELETE for CRUD operations.

Implementation Details:

Students have to write stepwise details of implementation.

Home page

```
import React from 'react'
function Home(){
    return(
        <div style={{margin:'5%'}}>
        <h1>Hello this the home page</h1>
        <a href="/register">Click to Register</a><hr/>
        <a href="/data">Click to view data entries</a>
        </div>
    )
}
export default Home
```

Register Page

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

const RegistrationForm = () => {
  const [formData, setFormData] = useState({
    username: '',
    password: '',
  });
  const [errorMessage, setErrorMessage] = useState('');

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
```

```
    try {
      const response = await axios.post('http://localhost:5000/api/auth/register',
formData);
      console.log(response.data);

    } catch (error) {
      console.log(error);
      if (error.response) {

        setErrorMessage(error.response.data.message);
      } else {
        setErrorMessage('An error occurred. Please try again.');
      }
    }
  };

  return (
    <div>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          name="username"
          placeholder="Username"
          value={formData.username}
          onChange={handleChange}
        />
        <input
          type="password"
          name="password"
          placeholder="Password"
          value={formData.password}
          onChange={handleChange}
        />
        <button type="submit">Register</button>
      </form>
      {errorMessage && <p>{errorMessage}</p>}
      <a href="/">Click to Home Page</a>
      <hr />
      <a href="/data">Click to view data entries</a>
    </div>
  );
};
```

```
export default RegistrationForm;
```

Data Page

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

const RegistrationData = () => {
  const [entries, setEntries] = useState([]);

  useEffect(() => {
       axios.get('http://localhost:5000/api/data/entries')
      .then(response => {
        console.log(response);
        setEntries(response.data);
      })
      .catch(error => {
        console.error(error);
      });
  }, []);

  return (
    <div>
      <h1>Registered Entries</h1>
      <ul>
        {entries.map((entry, index) => (
          <li key={index}>{entry.username}</li>
        ))}
      </ul>
    </div>
  );
};

export default RegistrationData;
```

Schema

```
onst mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  username: { type: String, unique: true, required: true },
  password: { type: String, required: true },
});

const User = module.exports = mongoose.model('User', userSchema);

module.exports = User;
```

Registration logic

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcrypt');
const User = require('../models/user');

const register = async (req, res) => {
  console.log("Register");
  try {
    console.log(req.body);
    const { username, password } = req.body;



    const existingUser = await User.findOne( {username} );

    if (existingUser) {
      return res.status(400).json({ message: 'Username already exists' });
    }


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


    const newUser = new User({ username, password: hashedPassword });

    await newUser.save();

    res.status(201).json({ message: 'User registered successfully' });
  } catch (error) {
    console.error(error);
```

```
    res.status(500).json({ message: 'Server error' });
  }
};

module.exports = register;
```

Server Page

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const authRoutes = require('./routes/auth');
const dataRoutes = require('./routes/data');

const app = express();
app.use(cors());
app.use(express.json());

const port = process.env.PORT || 5000;

const mongoURI = "mongodb+srv://minavk:minav-
mk@cluster0.wecbstk.mongodb.net/?retryWrites=true&w=majority";
mongoose.connect(mongoURI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});

app.use('/api/auth/register', authRoutes);
app.get('/api/data/entries', dataRoutes);

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

Conclusion:

In this experiment, we successfully implemented a simple backend using Node.js, Express.js, and MongoDB. We explored how to connect to a MongoDB database and perform CRUD operations using Mongoose. This experiment helped understand how to build a scalable backend using these technologies and solidified the concepts of full-stack development.