Department of Artificial Intelligence and Machine Learning
B.Tech. Sem: V Subject: Full Stack Development Laboratory (DJS22AML504)

Experiment 11 & 12

Name: Shubham Mourya **SAP ID: 60017230110**

Name:- Shubham Mourya
AOGS
60017230110

Experiment No:- 11 & 12

Aim:- Create an application to demonstrate integration of Node JS & Mongo DB & React JS

Theory:-

1) Node.JS - Backend Server

Node JS is a server-side runtime built on Chrome's V8 engine. It's designed for building fast and scalable applications, thanks to its event-driven, non-blocking I/O model.

- Single-threaded:- Handles many requests concurrently without blocking
- Asynchronous:- Non-blocking I/O improves performance.
- NPM:- Provides a vast collection of libraries to extend functionality.

2) MongoDB - NoSQL Database

It stores data in flexible, JSON-like documents, which makes it ideal for handling unstructured data.

- Document-oriented:- Data is stored as JSON-like documents.
- Scalable:- Supports horizontal scaling for handling large data volumes.
- CRUD operations.

(3) React JS : Frontend Library

React JS is a Javascript library for building dynamic user interfaces. It allows developers to create reusable components & efficiently update the UI.

- Component - based : UI is split into reusable, independent components.
- Virtual DOM :- Efficiently updates only parts of UI
- Unidirectional Data Flow : Makes state management easier.

Eg :-

1) Set Up Node.JS & Express :- Create backend server that handles HTTP requests.

2) Integrate MongoDB :- Store to-do data using MongoDB

3) Create Rest Full APIs :- Implement endpoints for CRUD operations (add, update, delete & fetch to-dos).

4) Build the Frontend in React :- Use React to create UI that interacts with Node JS. backend.

5) Connect Frontend and Backend :- Use axios & fetch in React to make API calls to Node JS server.

Conclusion :- Thus, we have successfully implemented project using Node JS, React JS, MongoDB.

Aim	**Create an application to demonstrate integration of Node JS and MongoDB and React JS**
Software	VS Code
Pre-requisite	Active internet connection

Code	**Frontend**
	
	Components->Note.jsx
	
	import React from 'react';
	
	function Note({ note, onDelete, onEdit }) {
	return (
	
	<h2 className="text-xl font-semibold text-gray-800 mb-3">{note.title}</h2>
	 {note.description}
	
	{note.tag}
	
	{new Date(note.date).toLocaleDateString()}
	<button onClick={() => onEdit(note)} className="text-blue-500 hover:text-blue-700">
	Edit
	</button>
	<button onClick={() => onDelete(note._id)} className="text-red-500 hover:text-red-700">
	Delete
	</button>
	
	);
	}
	
	export default Note;
	
	NoteForm.jsx
	
	import React, { useState, useEffect } from 'react';
	
	function NoteForm({ onAddNote, onUpdateNote, currentNote }) {
	const [title, setTitle] = useState('');
	const [description, setDescription] = useState('');

```
  const [tag, setTag] = useState('General');

  useEffect(() => {
    if (currentNote) {
      setTitle(currentNote.title);
      setDescription(currentNote.description);
      setTag(currentNote.tag);
    }
  }, [currentNote]);

  const handleSubmit = (e) => {
    e.preventDefault();
    const note = { title, description, tag };
    if (currentNote) {
      onUpdateNote({ ...note, _id: currentNote._id });
    } else {
      onAddNote(note);
    }
    resetForm();
  };

  const resetForm = () => {
    setTitle('');
    setDescription('');
    setTag('General');
  };

  return (
    <form onSubmit={handleSubmit} className="space-y-4">
      <div>
        <input
          type="text"
          value={title}
          onChange={(e) => setTitle(e.target.value)}
          placeholder="Note Title"
          className="w-full px-4 py-2 border border-gray-300 rounded-lg
focus:ring-2 focus:ring-blue-500 focus:border-transparent"
          required
        />
      </div>
      <div>
        <textarea
          value={description}
          onChange={(e) => setDescription(e.target.value)}
          placeholder="Note Description"
          rows="4"
          className="w-full px-4 py-2 border border-gray-300 rounded-lg
focus:ring-2 focus:ring-blue-500 focus:border-transparent"
```

```
          required
        />
      </div>
      <div>
        <input
          type="text"
          value={tag}
          onChange={(e) => setTag(e.target.value)}
          placeholder="Tag (Optional)"
          className="w-full px-4 py-2 border border-gray-300 rounded-lg
focus:ring-2 focus:ring-blue-500 focus:border-transparent"
        />
      </div>
      <button
        type="submit"
        className="w-full bg-blue-500 text-white py-2 px-4 rounded-lg
hover:bg-blue-600 transition-colors duration-300"
      >
        {currentNote ? 'Update Note' : 'Add Note'}
      </button>
    </form>
  );
}

export default NoteForm;
```

services->NoteService.js

```
const API_URL = 'http://localhost:5000/api/notes';

const getNotes = async () => {
  const response = await fetch(`${API_URL}/fetchallnotes`);
  return await response.json();
};

const addNote = async (note) => {
  const response = await fetch(`${API_URL}/addnote`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(note),
  });
  return await response.json();
};

const updateNote = async (note) => {
  const response = await fetch(`${API_URL}/updatenote/${note._id}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
```

```
    body: JSON.stringify(note),
  });
  return await response.json();
};

const deleteNote = async (id) => {
  await fetch(`${API_URL}/deletenote/${id}`, { method: 'DELETE' });
};

export default { getNotes, addNote, updateNote, deleteNote };
```

App.jsx

```
import React, { useState, useEffect } from 'react';
import Note from './components/Note';
import NoteForm from './components/NoteForm';
import NoteService from './services/NoteService';

function App() {
  const [notes, setNotes] = useState([]);
  const [loading, setLoading] = useState(true);
  const [currentNote, setCurrentNote] = useState(null);

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

  const fetchNotes = async () => {
    try {
      setLoading(true);
      const data = await NoteService.getNotes();
      setNotes(data);
    } catch (error) {
      console.error('Error fetching notes:', error);
    } finally {
      setLoading(false);
    }
  };

  const addNote = async (newNote) => {
    try {
      const savedNote = await NoteService.addNote(newNote);
      setNotes([...notes, savedNote]);
    } catch (error) {
      console.error('Error adding note:', error);
    }
  };
```

```
  const updateNote = async (updatedNote) => {
    try {
      const savedNote = await NoteService.updateNote(updatedNote);
      setNotes(notes.map(note => note._id === savedNote._id ? savedNote :
note));
      setCurrentNote(null);
    } catch (error) {
      console.error('Error updating note:', error);
    }
  };

  const deleteNote = async (id) => {
    try {
      await NoteService.deleteNote(id);
      setNotes(notes.filter(note => note._id !== id));
    } catch (error) {
      console.error('Error deleting note:', error);
    }
  };

  return (
    <div className="min-h-screen bg-gradient-to-br from-gray-50 to-gray-
100 p-6">
      <div className="max-w-4xl mx-auto">
        <h1 className="text-4xl font-bold text-center text-gray-800 mb-
8">✍️ Notes App</h1>
        <div className="bg-white rounded-xl shadow-lg p-6 mb-6">
          <NoteForm onAddNote={addNote} onUpdateNote={updateNote}
currentNote={currentNote} />
        </div>
        {loading ? (
          <div className="text-center py-4">Loading...</div>
        ) : (
          <div className="grid gap-4 md:grid-cols-2">
            {notes.map(note => (
              <Note
                key={note._id}
                note={note}
                onDelete={deleteNote}
                onEdit={() => setCurrentNote(note)}
              />
            ))}
          </div>
        )}
      </div>
    </div>
  );
```

```
}

export default App;
```

Backend

Models->Note.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const NoteSchema = new Schema({
   title: {
      type: String,
      required: true
   },
   description: {
      type: String,
      required: true
   },
   tag: {
      type: String,
      default: "General"
   },
   date: {
      type: Date,
      default: Date.now
   }
});

module.exports = mongoose.model('Note', NoteSchema);
```

routes->notes.js

```
const express = require('express');
const router = express.Router();
const Note = require('../models/Note');
const { body, validationResult } = require('express-validator');

// ROUTE 1: Get all notes using: GET "/api/notes/fetchallnotes"
router.get('/fetchallnotes', async (req, res) => {
   try {
      const notes = await Note.find();
      res.json(notes);
   } catch (error) {
      console.error(error.message);
      res.status(500).send("Internal Server Error");
   }
});
```

```
// ROUTE 2: Add a new note using: POST "/api/notes/addnote"
router.post('/addnote', [
    body('title', 'Enter a valid title').isLength({ min: 3 }),
    body('description', 'Description must be at least 5 characters').isLength({
min: 5 }),
], async (req, res) => {
    try {
        const { title, description, tag } = req.body;

        // If there are validation errors, return Bad request and the errors
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).json({ errors: errors.array() });
        }

        const note = new Note({
            title, description, tag
        });

        const savedNote = await note.save();
        res.json(savedNote);
    } catch (error) {
        console.error(error.message);
        res.status(500).send("Internal Server Error");
    }
});

// ROUTE 3: Update an existing note using: PUT "/api/notes/updatenote/:id"
router.put('/updatenote/:id', async (req, res) => {
    const { title, description, tag } = req.body;
    try {
        // Create a newNote object
        const newNote = {};
        if (title) { newNote.title = title; }
        if (description) { newNote.description = description; }
        if (tag) { newNote.tag = tag; }

        // Find the note to be updated and update it
        let note = await Note.findById(req.params.id);
        if (!note) { return res.status(404).send("Not Found"); }

        note = await Note.findByIdAndUpdate(req.params.id, { $set: newNote
}, { new: true });
        res.json(note);
    } catch (error) {
        console.error(error.message);
        res.status(500).send("Internal Server Error");
```

```
    }
});

// ROUTE 4: Delete an existing note using: DELETE
"/api/notes/deletenote/:id"
router.delete('/deletenote/:id', async (req, res) => {
    try {
        // Find the note to be deleted and delete it
        let note = await Note.findById(req.params.id);
        if (!note) { return res.status(404).send("Not Found"); }

        note = await Note.findByIdAndDelete(req.params.id);
        res.json({ "Success": "Note has been deleted", note });
    } catch (error) {
        console.error(error.message);
        res.status(500).send("Internal Server Error");
    }
});

module.exports = router;
```

db.js

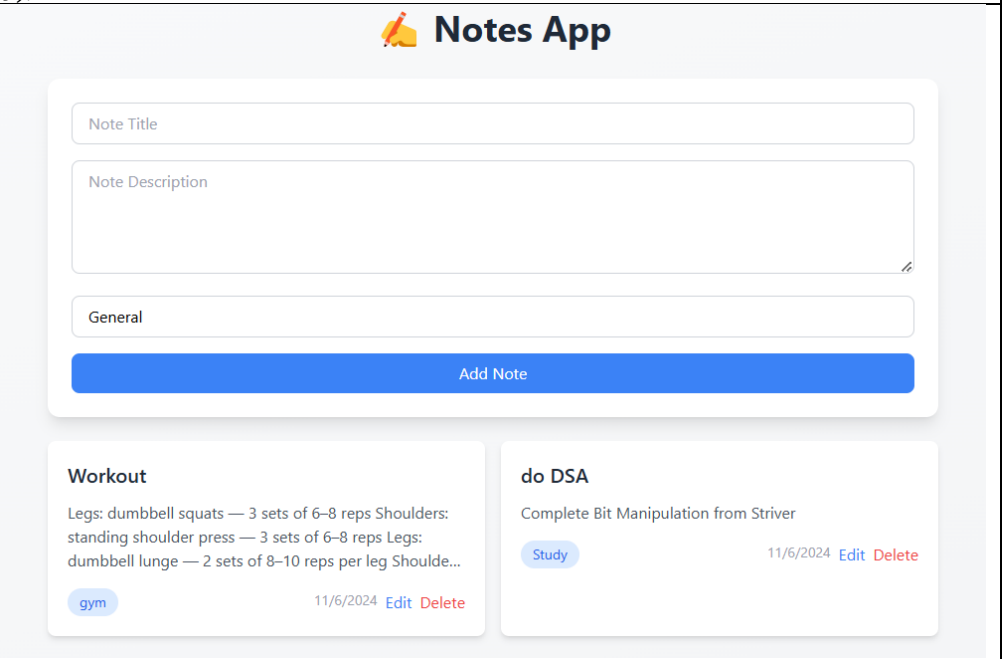
```
const mongoose = require('mongoose');

const MONGOURI = "mongodb://localhost:27017/notesapp";

const connectToMongo = () => {
    mongoose.connect(MONGOURI).then(() => {
        console.log("Connected to MongoDB successfully");
    }).catch((err) => {
        console.error("Error connecting to MongoDB: ", err);
    });
};

module.exports = connectToMongo;
```

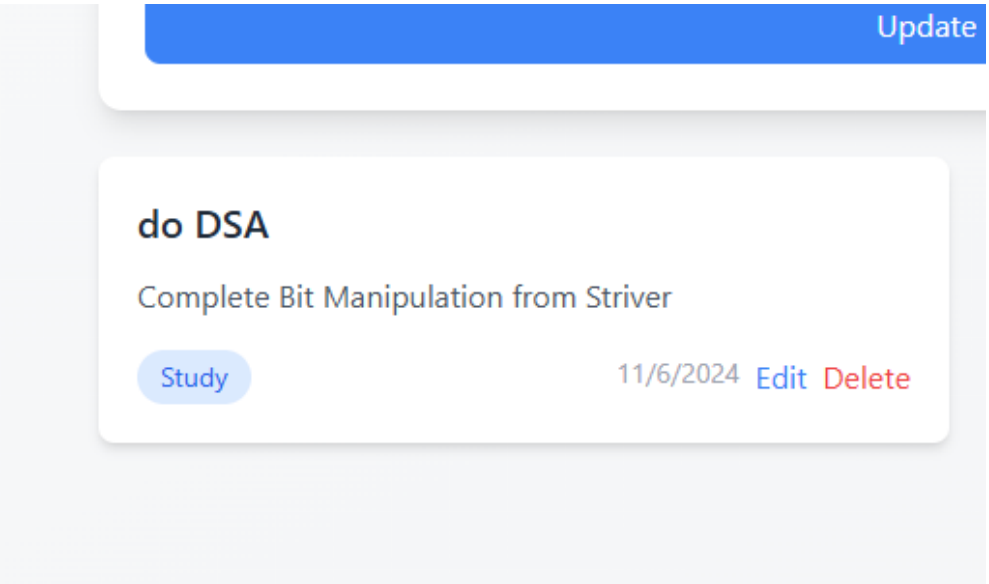
index.js

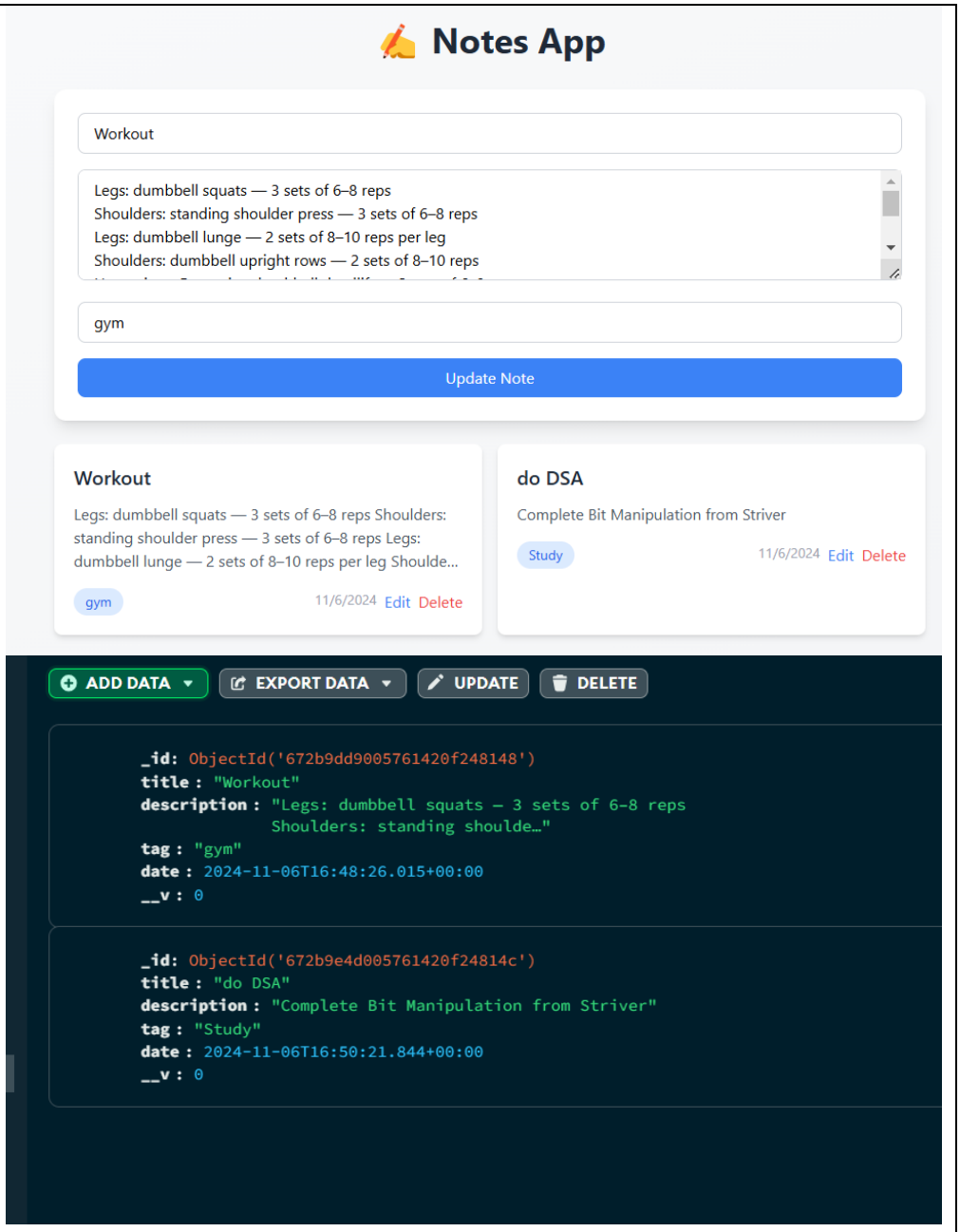
```
const connectToMongo = require('./db');
const express = require('express');
const cors = require('cors');

// Connect to MongoDB
connectToMongo();

const app = express();
const port = 5000;

// Middleware
```

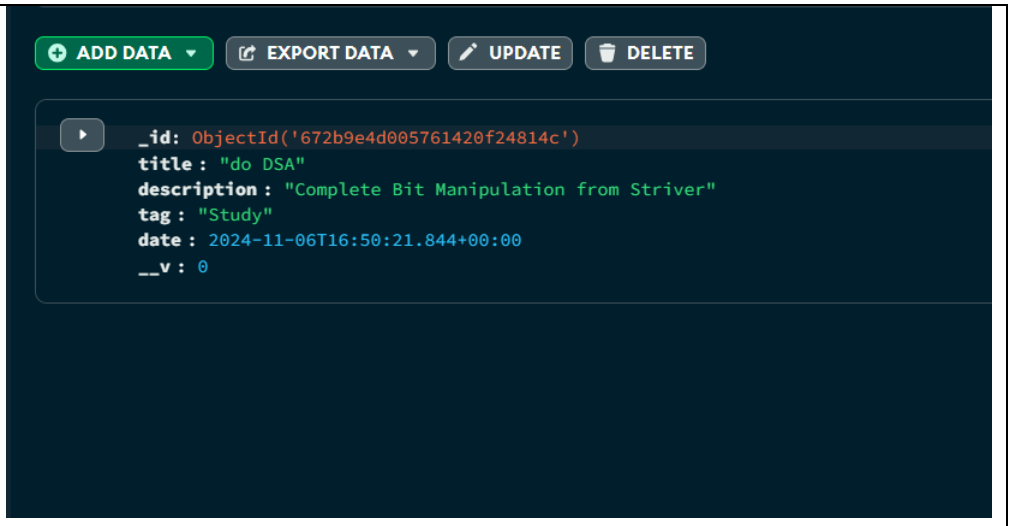
```
app.use(cors());
app.use(express.json());

// Routes
app.use('/api/notes', require('./routes/notes'));

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

Result






Notes App
Workout
Legs: dumbbell squats — 3 sets of 6–8 reps
Shoulders: standing shoulder press — 3 sets of 6–8 reps
Legs: dumbbell lunge — 2 sets of 8–10 reps per leg
Shoulders: dumbbell upright rows — 2 sets of 8–10 reps
gym
Update Note
Workout
Legs: dumbbell squats — 3 sets of 6–8 reps Shoulders: standing shoulder press — 3 sets of 6–8 reps Legs: dumbbell lunge — 2 sets of 8–10 reps per leg Shoulde...
gym
11/6/2024 Edit Delete
do DSA
Complete Bit Manipulation from Striver
Study
11/6/2024 Edit Delete
ADD DATA
EXPORT DATA
UPDATE
DELETE
_id: ObjectId('672b9dd9005761420f248148')
title : "Workout"
description : "Legs: dumbbell squats — 3 sets of 6-8 reps
Shoulders: standing shoulde…"
tag : "gym"
date : 2024-11-06T16:48:26.015+00:00
__v : 0
_id: ObjectId('672b9e4d005761420f24814c')
title : "do DSA"
description : "Complete Bit Manipulation from Striver"
tag : "Study"
date : 2024-11-06T16:50:21.844+00:00
__v : 0

	
Conclusion	Thus, we have successfully performed the integration of Node.js, MongoDB, and React.js to create a full-stack notes application. The frontend allows users to add, edit, and delete notes with real-time updates, while the backend handles data persistence and validation through RESTful APIs connected to MongoDB. This experiment demonstrates the seamless communication between the frontend and backend, showcasing CRUD operations in a practical web application.