**EX NO: 11**

**DATE:**

**CONTENT BEYOND SYLLABUS**

**IMPROVE THE REUSABILITY AND MAINTAINABILITY OF THE SOFTWARE SYSTEM BY APPLYING A SINGLETON DESIGN PATTERN**

**AIM:**

To improve the reusability and maintainability of the software system by applying a singleton design pattern.

**SINGLETON DESIGN PATTERN:**

**Overview:**

The Singleton pattern ensures that only one instance of a class is created and provides a global access point to that instance. It is useful when exactly one object is needed to coordinate actions across the system.

**Implementation in Node.js:**

For the given software system, we can apply the Singleton pattern to the **Database Adapter** class to ensure only one database connection is used throughout the application, improving efficiency and resource management.

**Singleton Database Adapter Implementation:**

const express = require('express');

const { ObjectId } = require('mongodb');

const cors = require('cors');

const DatabaseSingleton = require('./DatabaseSingleton');

const app = express();

const PORT = 5000;

app.use(cors());

app.use(express.json());

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

try {

const collection = await DatabaseSingleton.getCollection("students");

const students = await collection.find().toArray();

res.json(students);

} catch (error) {

console.error('Error fetching students:', error);

res.status(500).json({ message: 'Error fetching students', error }); }});

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

try {

const collection = await DatabaseSingleton.getCollection("students");

const newStudent = req.body;

await collection.insertOne(newStudent);

res.json(newStudent);

} catch (error) {

console.error('Error adding student:', error);

res.status(500).json({ message: 'Error adding student', error }); }});

app.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);});

const { MongoClient } = require('mongodb');

class DatabaseSingleton {

constructor() {

if (!DatabaseSingleton.instance) {

this.client = new MongoClient("your\_mongo\_uri", { useNewUrlParser: true, useUnifiedTopology: true });

DatabaseSingleton.instance = this; }

return DatabaseSingleton.instance; }

async connect() {

if (!this.client.topology || !this.client.topology.isConnected()) {

await this.client.connect(); }

return this.client.db("sis"); }

static async getCollection(collectionName) {

const instance = new DatabaseSingleton();

const db = await instance.connect();

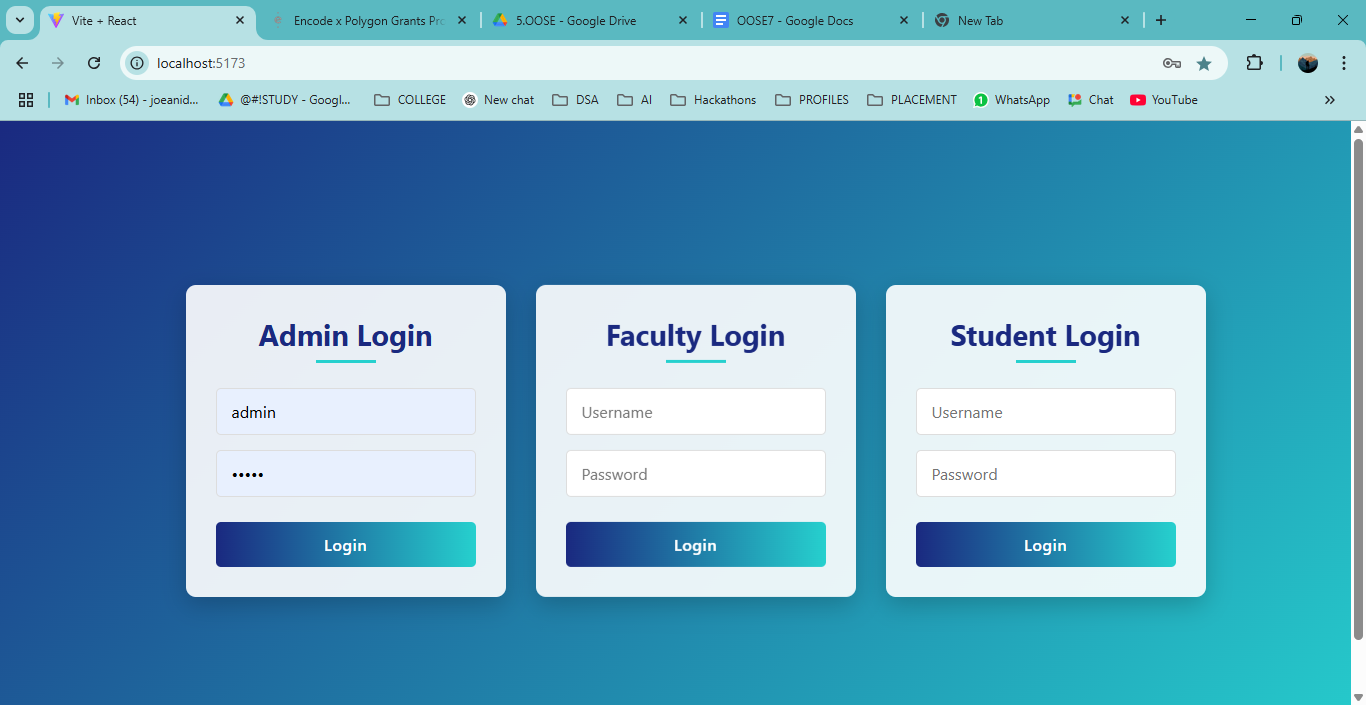
return db.collection(collectionName); }}

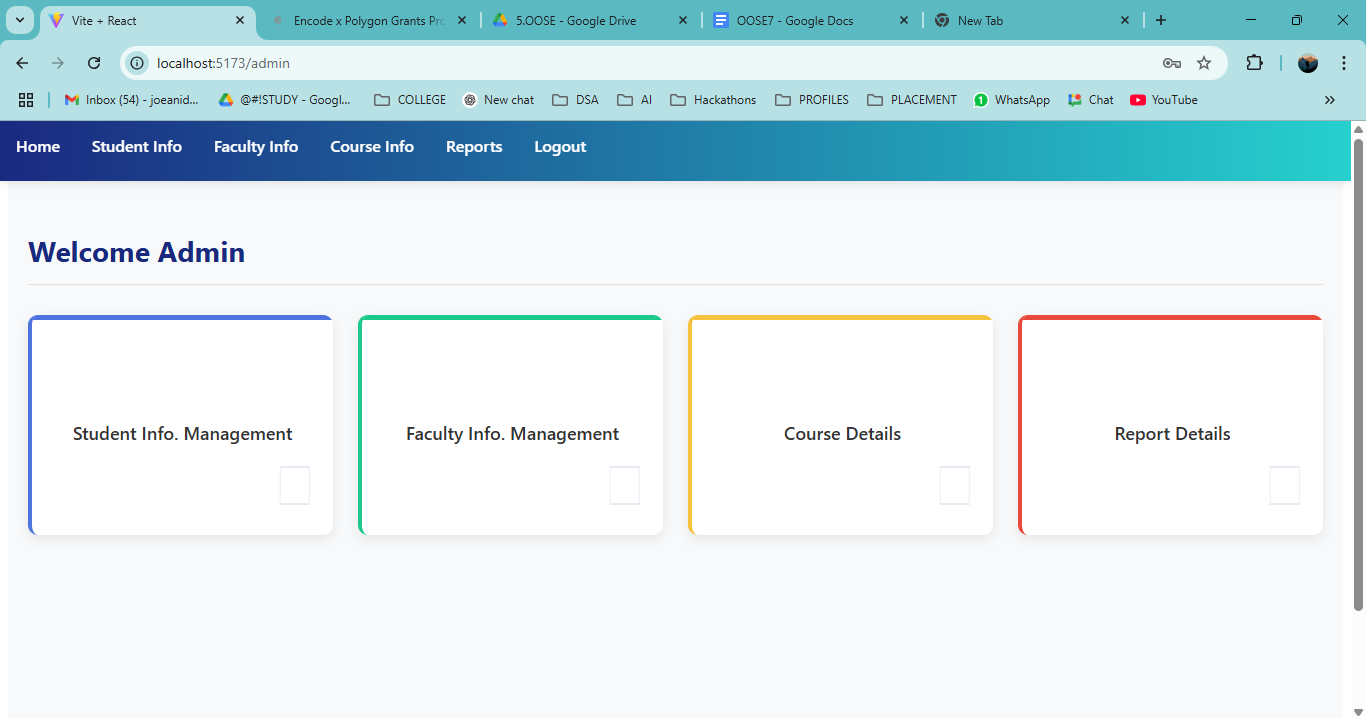
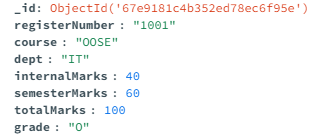
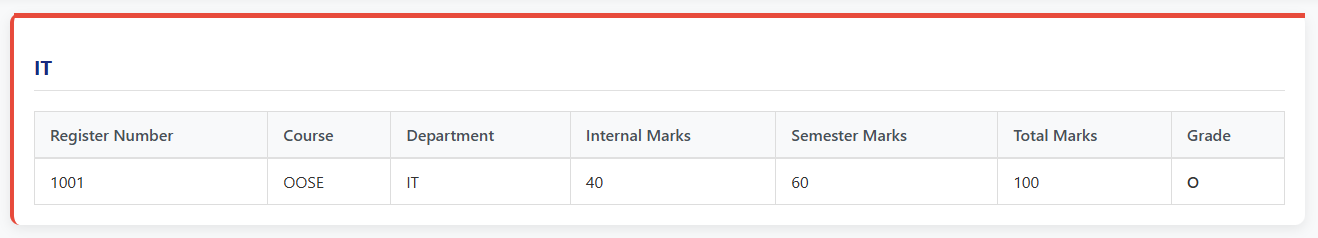
module.exports = DatabaseSingleton;

**Benefits of Using Singleton Pattern:**

* **Single Database Connection:** Avoids unnecessary multiple database connections.
* **Performance Improvement:** Reduces resource overhead.
* **Global Access Point:** Ensures consistent database access across the application.
* **Prevents Duplication:** Ensures only one database adapter instance is used.

**OUTPUT:**

****

****

**RESULT:**