

Day 7 assignment

Name – Soham Sarkar

College Name- MCKV INSTITUTE OF ENGINEERING



June 25, 2025

**📝 Assignment Question – Day 7**

**Q: Create a Node.js application that connects to a MongoDB database using both the MongoDB Native Driver and Mongoose (in two separate files). Implement full CRUD (Create, Read, Update, Delete) operations for a students collection.**

**💡 Instructions:**

1. **Using MongoDB Native Driver:**
   * Create a connection to MongoDB.
   * Implement the following functions:
     + createStudent() – Insert a new student document.
     + getStudents() – Retrieve all student documents.
     + updateStudent() – Update a student by ID.
     + deleteStudent() – Delete a student by ID.
2. **Using Mongoose:**
   * Define a Student schema with fields: name, email, age, course.
   * Perform the same CRUD operations as above using Mongoose methods.

**📁 Folder structure:**

Day7-Assignment/

│

├── mongodb\_native.js      // CRUD using native driver

├── mongoose\_crud.js       // CRUD using mongoose

├── package.json

**✅ Expected Output:**

* Insert, retrieve, update, and delete operations should be performed through function calls.
* Use meaningful console messages to show success or error during operations.

**Folder Structure:**

Day7-Assignment/

├── mongodb\_native.js // CRUD using MongoDB Native Driver

├── mongoose\_crud.js // CRUD using Mongoose

├── package.json

**Console:**

npm init -y

npm install mongodb mongoose

code mongodb\_native.js

code mongoose\_crud.js

**mongodb\_native File:**

* // Import MongoClient and ObjectId from the MongoDB driver
* const { MongoClient, ObjectId } = require('mongodb');
* // Define the MongoDB URI with authentication and admin database
* const uri = 'mongodb://soham:soham123@localhost:27017/admin';
* // Create a new MongoClient instance using the URI
* const client = new MongoClient(uri);
* // Define the name of the database to use
* const dbName = 'school';
* // Function to insert a new student into the "students" collection
* async function createStudent(student) {
* try {
* // Connect to MongoDB
* await client.connect();
* // Get a reference to the "school" database
* const db = client.db(dbName);
* // Insert the student document into the "students" collection
* const result = await db.collection('students').insertOne(student);
* // Print success message with inserted ID
* console.log('Student inserted with ID:', result.insertedId);
* } catch (err) {
* // Print error message if something goes wrong
* console.error('Error inserting student:', err);
* }
* }
* // Function to retrieve all students from the "students" collection
* async function getStudents() {
* try {
* // Connect to MongoDB
* await client.connect();
* // Get a reference to the "school" database
* const db = client.db(dbName);
* // Find all documents in the "students" collection and convert to array
* const students = await db.collection('students').find().toArray();
* // Print the list of students
* console.log('All Students:', students);
* } catch (err) {
* // Print error message if something goes wrong
* console.error('Error retrieving students:', err);
* }
* }
* // Function to update a student document by its ID
* async function updateStudent(id, updates) {
* try {
* // Connect to MongoDB
* await client.connect();
* // Get a reference to the "school" database
* const db = client.db(dbName);
* // Update the student document with the given ID using $set operator
* const result = await db.collection('students').updateOne(
* { \_id: new ObjectId(id) },  // Match by ObjectId
* { $set: updates }           // Apply updates
* );
* // Print success message if document was modified
* console.log('Student updated:', result.modifiedCount > 0);
* } catch (err) {
* // Print error message if something goes wrong
* console.error('Error updating student:', err);
* }
* }
* // Function to delete a student document by its ID
* async function deleteStudent(id) {
* try {
* // Connect to MongoDB
* await client.connect();
* // Get a reference to the "school" database
* const db = client.db(dbName);
* // Delete the student document with the given ID
* const result = await db.collection('students').deleteOne({ \_id: new ObjectId(id) });
* // Print success message if document was deleted
* console.log('Student deleted:', result.deletedCount > 0);
* } catch (err) {
* // Print error message if something goes wrong
* console.error('error deleting student:', err);
* }
* }
* // Example function calls.
* // Call createStudent() to insert a new student document
* // createStudent({ name: 'Alice', email: 'alice@example.com', age: 21, course: 'Math' });
* // Call getStudents() to retrieve and display all student documents
* // getStudents();
* // Call updateStudent() with an ID and updated fields
* // updateStudent('685aebf53c9c7411e95aa0d8', { age: 22 });
* // Call deleteStudent() with an ID to remove a student document.
* // deleteStudent('685aebf53c9c7411e95aa0d8');

**mongoose\_crud.js File:**

* // Import mongoose library to interact with MongoDB using Mongoose ORM
* const mongoose = require('mongoose');
* // Connect to MongoDB using Mongoose with a plain connection URI
* // This connects to a local MongoDB database named "school"
* // connection to MongoDB Atlas
* mongoose.connect('mongodb+srv://soham123:soham123@cluster0.9jnaofk.mongodb.net/school')
* .then(() => console.log('MongoDB connected'))
* .catch(err => console.error('Connection error:', err));
* // Define a schema for the "students" collection
* // Fields: name, email, age, course
* const studentSchema = new mongoose.Schema({
* name: String,     // Student's name
* email: String,    // Student's email address
* age: Number,      // Student's age
* course: String    // Course the student is enrolled in
* });
* // Create a model from the schema (used to interact with the collection)
* const Student = mongoose.model('Student', studentSchema);
* // Function to create a new student document
* async function createStudent(data) {
* try {
* const student = new Student(data);       // Create a new Student instance with provided data
* const saved = await student.save();      // Save the student document to the database
* console.log('Student saved:', saved); // Log the saved student object
* } catch (err) {
* console.error('Error:', err);         // Log any error that occurs
* }
* }
* // Function to retrieve all student documents
* async function getStudents() {
* try {
* const students = await Student.find();       // Fetch all documents in the "students" collection
* console.log('All Students:', students);   // Log the array of students
* } catch (err) {
* console.error('Error:', err);             // Log any error that occurs
* }
* }
* // Function to update a student document by its ID
* async function updateStudent(id, updates) {
* try {
* // Find the student by ID and apply the updates; return the updated document
* const updated = await Student.findByIdAndUpdate(id, updates, { new: true });
* if (updated) {
  + console.log('Student updated:', updated); // Log the updated document
* } else {
  + console.log('No student found');          // Inform if the student wasn't found
* }
* } catch (err) {
* console.error('Error:', err);                 // Log any error that occurs
* }
* }
* // Function to delete a student document by its ID
* async function deleteStudent(id) {
* try {
* // Find the student by ID and delete it
* const deleted = await Student.findByIdAndDelete(id);
* if (deleted) {
  + console.log('Student deleted:', deleted); // Log the deleted document
* } else {
  + console.log('No student found');           // Inform if no matching document was found
* }
* } catch (err) {
* console.error('Error:', err);                 // Log any error that occurs
* }
* }
* // Example calls — Uncomment only one at a time to test each operation
* // createStudent({ name: 'Anu', email: 'anu@example.com', age: 22, course: 'CS' }); // Insert a new student
* // getStudents(); // Retrieve and display all students
* // updateStudent('685af3b6bcb30547b998b11e', { age: 23 }); // Update student age (replace with real ID)
* // deleteStudent('685af3b6bcb30547b998b11e'); // Delete student by ID (replace with real ID)

**Run :**

node mongodb\_native.js

node mongoose\_crud.js