**EX.NO:** 3

**DATE:** 07.07.25

**Student Management System using Node.js, Express, MongoDB, HTML, and CSS**

**AIM:**

**To create a Node.js server using Express.js that performs Create, Read, Update, and Delete (CRUD) operations on student details, storing the data in a MongoDB database.**The data should be collected through an HTML form and styled using CSS.

**Algorithm:**

1.Start the Project:

* Create a new directory (e.g., EX3) for the project.
* Open terminal in the directory and run npm init -y to initialize a Node.js project.

2.Install Required Dependencies:

* Install required packages using: npm install express mongoose body-parser.

3.Set Up Project Structure:

* Create folders:
  + models/ → to store Mongoose schema (Student.js)
  + public/ → to store index.html and style.css

4.Create Mongoose Schema (models/Student.js):

* Define a schema with fields: name, email, age.
* Export the Mongoose model as Student.

5.Connect to MongoDB:

* In server.js, connect to MongoDB using Mongoose: mongoose.connect('mongodb://localhost:27017/studentDB')

6.Create Express App (server.js):

* Import necessary modules: express, mongoose, body-parser, path.
* Create an Express app using const app = express();.

7.Add Middleware:

* Use body-parser to parse form data.
* Use express.static to serve static files from public/.

8.Create HTML Form (public/index.html):

* Create a form to take inputs: name, email, age.
* Set action="/add" and method="POST" for the form.
* Link style.css for design.

9.Design the Form with CSS (public/style.css):

* Style the form, input fields, and buttons.
* Ensure a responsive and user-friendly design.

10.Handle Form Submission - Add Student (POST):

* In server.js, create a POST route /add that:
  + Accepts data from the form.
  + Saves a new student document in MongoDB.

11.Create Route to View Students (GET):

* Define a GET route /students that:
  + Fetches all student records from the database.
  + Sends them to the browser as a table or list.

12.Add Edit Functionality:

* Create a GET route /edit/:id to show a pre-filled form.
* Create a POST route /edit/:id to update student details in MongoDB.

13.Add Delete Functionality:

* Create a GET route /delete/:id to delete the selected student from MongoDB.

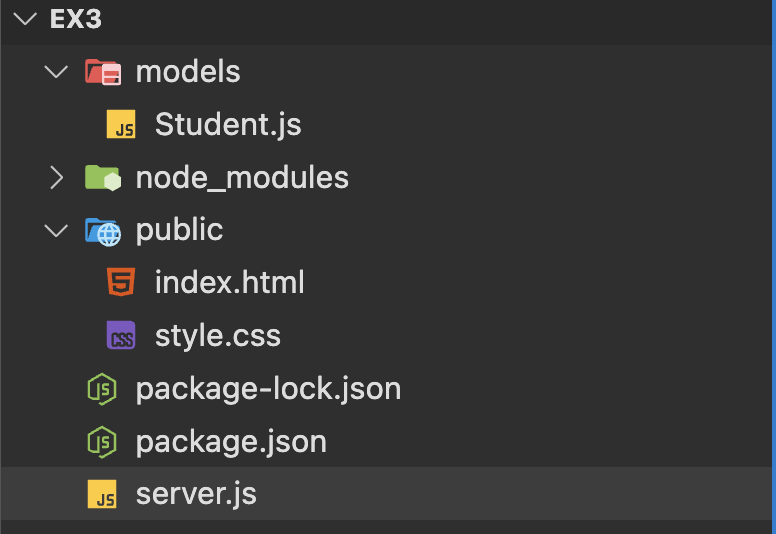
14.Start the Server:

* In server.js, add: app.listen(3000, () => console.log("Server running on http://localhost:3000"));

15.Test All Functionalities:

* + Open the browser and test:
    - Adding a new student.
    - Viewing all students.
    - Editing and updating a student.
    - Deleting a student.

**Design:**



This is the project root containing all code files.

**1. models/Student.js**

* Contains the **Mongoose schema** and model definition for student records (name, email, age).

**2. public/index.html**

* HTML form to accept user input: name, email, age.
* Method: POST to /add.
* Linked with external CSS.

**3. public/style.css**

* Contains styling for the HTML form and layout.
* Improves user experience and layout design.

**4. server.js**

* Main Node.js server file.
* Uses Express to:
  1. Serve static files (HTML, CSS)
  2. Connect to MongoDB
  3. Handle CRUD routes for managing students.

**5. package.json / package-lock.json**

* Manage dependencies (express, mongoose, etc.)
* Track versions and scripts.

**Coding:**

**Student.js:**

// models/student.js

const mongoose = require('mongoose');

const studentSchema = new mongoose.Schema({

name: String,

age: Number,

email: String,

department: String

});

module.exports = mongoose.model('Student', studentSchema);

**index.html:**

<!DOCTYPE html>

<html>

<head>

<title>Student Form</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div class="form-container">

<h2>Add Student</h2>

<form action="/students" method="POST">

<input type="text" name="name" placeholder="Student Name" required>

<input type="number" name="age" placeholder="Age" required>

<input type="email" name="email" placeholder="Email" required>

<button type="submit">Submit</button>

</form>

<br>

<a href="/students">View Students</a>

</div>

</body>

</html>

**style.css:**

body {

font-family: Arial, sans-serif;

background: #f2f2f2;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.form-container {

background: white;

padding: 30px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

}

input, button {

display: block;

width: 100%;

margin: 10px 0;

padding: 10px;

}

**server.js:**

// server.js

// Import required modules

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const path = require('path');

// Import the student model

const Student = require('./models/Student');

// Initialize the express app

const app = express();

// Connect to MongoDB

mongoose.connect('mongodb+srv://sriraghavardhinim24mca:1234@cluster0.le3xbg9.mongodb.net/')

.then(() => console.log("MongoDB connected"))

.catch(err => console.error(err));

// Middleware

app.use(bodyParser.urlencoded({ extended: false }));

app.use(express.static(path.join(\_\_dirname, 'public')));

// Serve the HTML form

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

res.sendFile(path.join(\_\_dirname, 'public', 'form.html'));

});

// Create student

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

const { name, age, email } = req.body;

const student = new Student({ name, age, email });

await student.save();

res.redirect('/students');

});

// View students in formatted table

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

const students = await Student.find();

let html = `

<html>

<head>

<title>Student List</title>

<style>

body { font-family: Arial; margin: 40px; }

table { width: 80%; border-collapse: collapse; margin-bottom: 20px; }

th, td { border: 1px solid #ccc; padding: 10px; text-align: left; }

th { background-color: #f4f4f4; }

a.button { padding: 6px 12px; margin: 0 5px; background: #007BFF; color: white; text-decoration: none; border-radius: 4px; }

a.button:hover { background: #0056b3; }

</style>

</head>

<body>

<h1>Student Details</h1>

<table>

<tr>

<th>Name</th>

<th>Email</th>

<th>Age</th>

<th>Actions</th>

</tr> ;

students.forEach(student => {

html += `

<tr>

<td>${student.name}</td>

<td>${student.email}</td>

<td>${student.age}</td>

<td>

<a class="button" href="/edit/${student.\_id}">Edit</a>

<a class="button" href="/delete/${student.\_id}" style="background: #dc3545;">Delete</a>

</td>

</tr>

`;

});

html += `

</table>

<a href="/" class="button" style="background: #28a745;">Add New Student</a>

</body>

</html>

`;

res.send(html);

});

// Edit student form

app.get('/edit/:id', async (req, res) => {

const student = await Student.findById(req.params.id);

res.send(`

<h2>Edit Student</h2>

<form action="/update/${student.\_id}" method="POST">

<input type="text" name="name" value="${student.name}" required>

<input type="number" name="age" value="${student.age}" required>

<input type="email" name="email" value="${student.email}" required>

<button type="submit">Update</button>

</form>

<br><a href="/students">Cancel</a>

`);

});

// Update student

app.post('/update/:id', async (req, res) => {

const { name, age, email } = req.body;

await Student.findByIdAndUpdate(req.params.id, { name, age, email });

res.redirect('/students');

});

// Delete student

app.get('/delete/:id', async (req, res) => {

await Student.findByIdAndDelete(req.params.id);

res.redirect('/students');

});

// Start the server

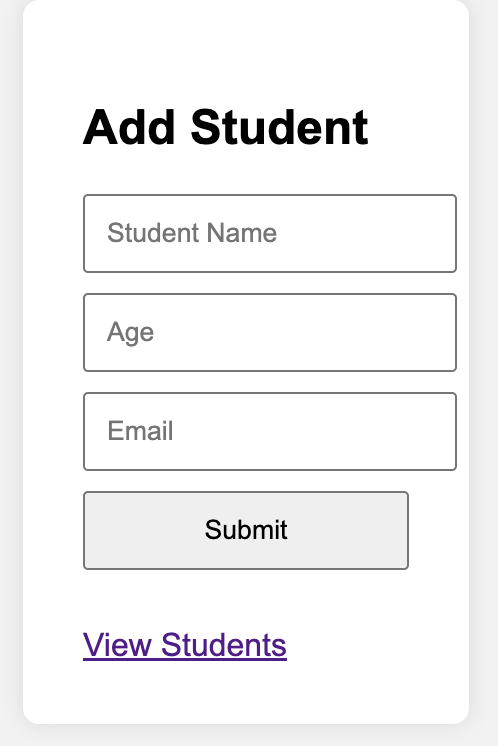
const PORT = 3000;

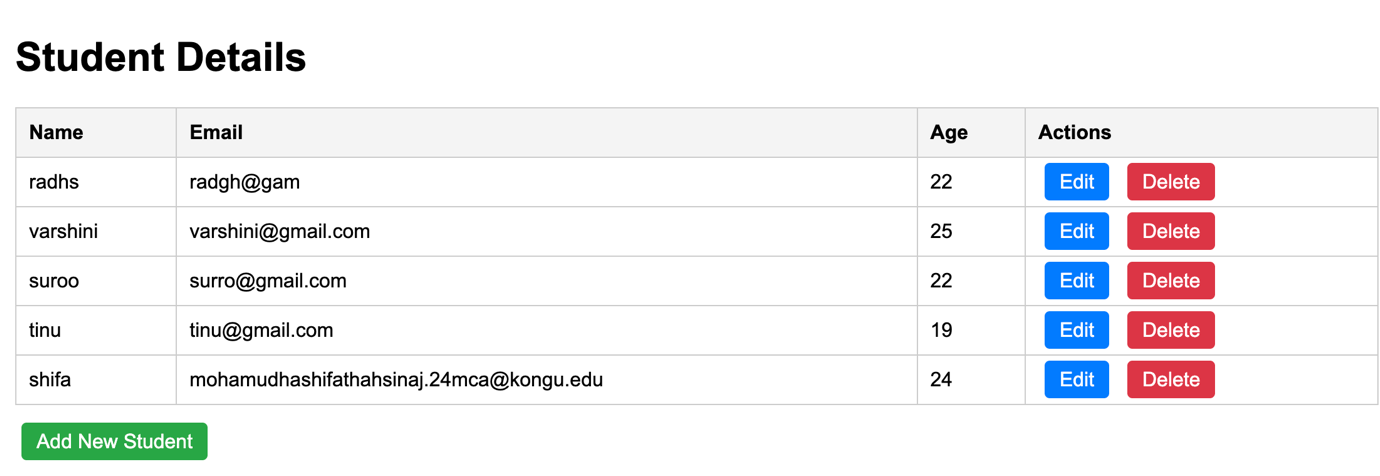
app.listen(PORT, () => {

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

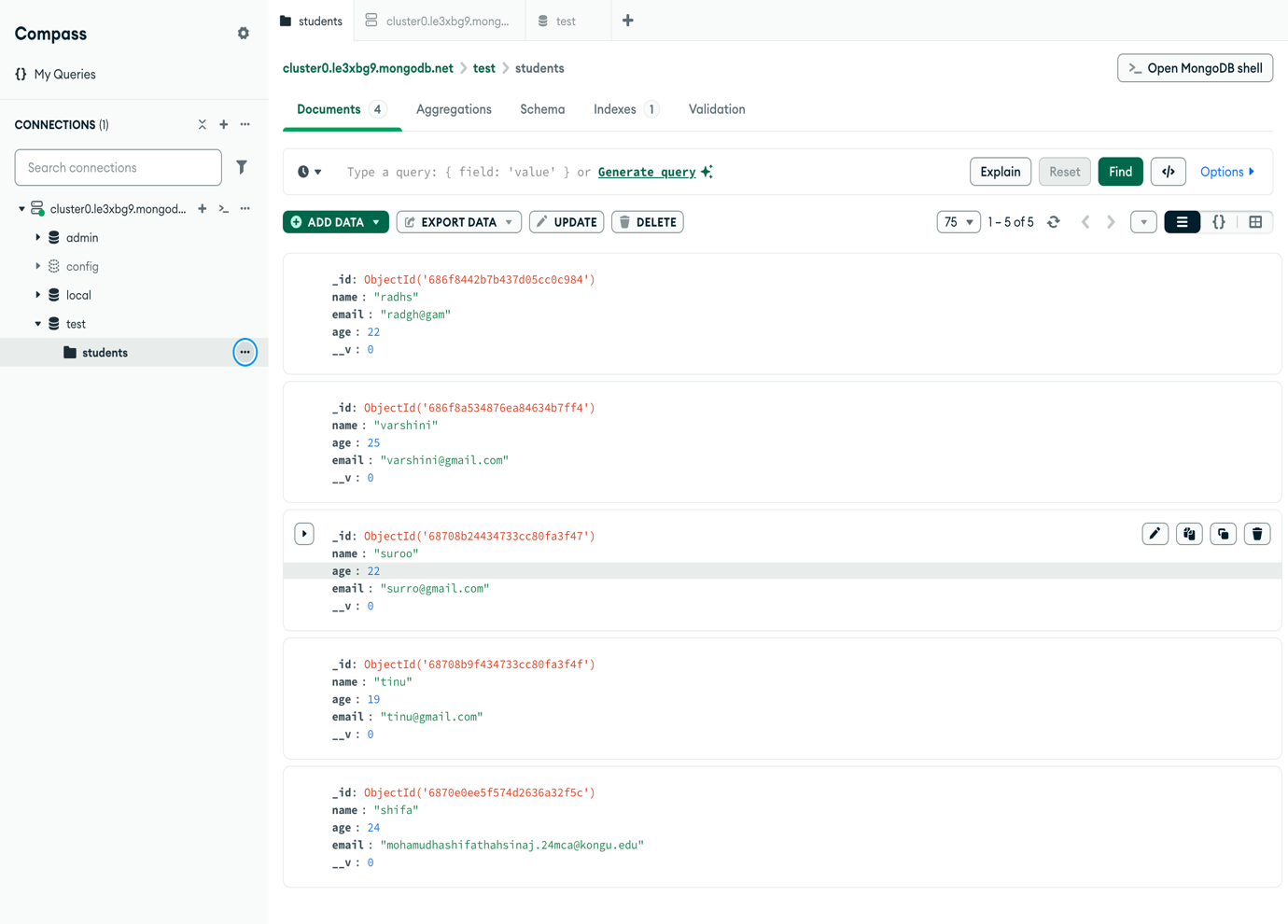
});

**OUTPUT:**

****

****

****

****

|  |  |
| --- | --- |
| **COE(30)** |  |
| **RECORD(20)** |  |
| **VIVA(10)** |  |
| **TOTAL(60)** |  |

**RESULT:**

Thus the above program has been successfully verified and executed.