**Batch: TY A2 Roll No.: 16010122041**

**Experiment / assignment / tutorial No.\_\_8\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| Title: Implementation of CRUD operations with Mongoose. |

**AIM:** **Demonstrate the use of Mongoose with CRUD operation.**

**Problem Definition:**

**1) Generate Database model**

**2) Create RESTful API**

**3) Demonstrate the Endpoints.**

**Resources used:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 2:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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**

* Mongoose CRUD operation
* RESTFul API

**Mongoose CRUD Operations**

**Mongoose** is an Object Data Modeling (ODM) library for MongoDB and Node.js, which simplifies interactions with the MongoDB database by providing schema-based data models. CRUD stands for Create, Read, Update, and Delete — the basic operations that are performed in a database.

**1. Create (Insert) Operation**

The Create operation is used to add new documents to a MongoDB collection. With Mongoose, you can create a new record by defining a model and using the .save() method or .create().

// Define a schema for a User

const userSchema = new mongoose.Schema({

name: String,

email: String,

age: Number

});

// Create a Mongoose model from the schema

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

// Create a new user and save it to the database

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

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

try {

const newUser = new User({ name, email, age });

await newUser.save();

res.status(201).json({ message: 'User created', newUser });

} catch (err) {

res.status(500).json({ message: 'Error creating user', error: err });

}

});

**2. Read (Find) Operation**

The Read operation retrieves documents from the MongoDB collection. Mongoose provides methods such as .find(), .findById(), and .findOne() for querying data.

// Get all users

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

try {

const users = await User.find(); // Fetch all users from the database

res.status(200).json(users);

} catch (err) {

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

}

});

// Get a specific user by ID

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

try {

const user = await User.findById(req.params.id); // Fetch a user by ID

if (!user) {

return res.status(404).json({ message: 'User not found' });

}

res.status(200).json(user);

} catch (err) {

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

}

});

**3. Update Operation**

The Update operation modifies an existing document. Mongoose provides methods like .findByIdAndUpdate() or .updateOne() for updating data.

// Update a user's details by ID

app.put('/users/:id', async (req, res) => {

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

try {

const updatedUser = await User.findByIdAndUpdate(req.params.id, { name, email, age }, { new: true });

if (!updatedUser) {

return res.status(404).json({ message: 'User not found' });

}

res.status(200).json({ message: 'User updated', updatedUser });

} catch (err) {

res.status(500).json({ message: 'Error updating user', error: err });

}

});

In the above code, { new: true } ensures the response contains the updated document.

**4. Delete Operation**

The Delete operation removes a document from the collection. You can use methods like .findByIdAndDelete() or .deleteOne().

// Delete a user by ID

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

try {

const deletedUser = await User.findByIdAndDelete(req.params.id);

if (!deletedUser) {

return res.status(404).json({ message: 'User not found' });

}

res.status(200).json({ message: 'User deleted', deletedUser });

} catch (err) {

res.status(500).json({ message: 'Error deleting user', error: err });

}

});

**RESTful API**

A **RESTful API** (Representational State Transfer) is a set of rules and conventions for designing and interacting with web services. It uses HTTP methods (like GET, POST, PUT, DELETE) for CRUD operations and is stateless, meaning the server does not store client session data between requests.

A RESTful API has the following characteristics:

1. **Stateless**: Every request from a client must contain all the necessary information for the server to understand and process it.
2. **Uniform Interface**: Resources (like users, posts, products) are identified in a consistent manner using URIs.
3. **HTTP Methods**: RESTful APIs commonly use the following methods for performing CRUD operations:
   * **GET**: Retrieve data (Read operation)
   * **POST**: Create new data (Create operation)
   * **PUT**: Update existing data (Update operation)
   * **DELETE**: Delete data (Delete operation)
4. **Resource-based**: Resources like users, products, or files are represented by URIs (e.g., /users or /products).

**RESTful API Example in Express.js**

Here’s an example of a RESTful API for managing users using Express.js and Mongoose.

// Import required modules

const express = require('express');

const mongoose = require('mongoose');

const app = express();

app.use(express.json()); // Middleware to parse JSON request bodies

// MongoDB connection (replace with your own connection string)

mongoose.connect('mongodb://localhost:27017/restfulAPIExample', { useNewUrlParser: true, useUnifiedTopology: true })

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

.catch(err => console.log('Error connecting to MongoDB:', err));

// Define a schema and model

const userSchema = new mongoose.Schema({

name: String,

email: String,

age: Number

});

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

// CRUD routes

// 1. Get all users (Read)

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

try {

const users = await User.find();

res.status(200).json(users);

} catch (err) {

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

}

});

// 2. Get a specific user by ID (Read)

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

try {

const user = await User.findById(req.params.id);

if (!user) return res.status(404).json({ message: 'User not found' });

res.status(200).json(user);

} catch (err) {

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

}

});

// 3. Create a new user (Create)

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

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

try {

const newUser = new User({ name, email, age });

await newUser.save();

res.status(201).json({ message: 'User created', newUser });

} catch (err) {

res.status(500).json({ message: 'Error creating user', error: err });

}

});

// 4. Update an existing user by ID (Update)

app.put('/users/:id', async (req, res) => {

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

try {

const updatedUser = await User.findByIdAndUpdate(req.params.id, { name, email, age }, { new: true });

if (!updatedUser) return res.status(404).json({ message: 'User not found' });

res.status(200).json({ message: 'User updated', updatedUser });

} catch (err) {

res.status(500).json({ message: 'Error updating user', error: err });

}

});

// 5. Delete a user by ID (Delete)

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

try {

const deletedUser = await User.findByIdAndDelete(req.params.id);

if (!deletedUser) return res.status(404).json({ message: 'User not found' });

res.status(200).json({ message: 'User deleted', deletedUser });

} catch (err) {

res.status(500).json({ message: 'Error deleting user', error: err });

}

});

// Start the Express server

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**Implementation Details:**

# App.js

import React, { useState, useEffect } from 'react'; import './App.css';

function App() { const [newTask, setNewTask] = useState(''); const [updateTask, setUpdateTask] = useState(''); const [tasks, setTasks]

= useState([]);

const [editTask, setEditTask] = useState({ id: null, title: '' });

useEffect(()

=> {

fetchTasks()

;

}, []);

const fetchTasks = async () => { try {

const response = await fetch('http://localhost:5000/api/tasks'); if (response.*ok*) {

const data = await response.json(); setTasks(data);

} else {

console.error('Failed to fetch tasks. Server returned:', response.*status*, response.*statusText*);

}

} catch (error) {

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

}

};

const handleAddTask = async () => { try {

const response = await fetch('http://localhost:5000/api/tasks', { method: 'POST', headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ task: newTask })

});

setTasks([...tasks,

data]); setNewTask('');

} else {

console.error('Failed to add task. Server returned:', response.*status*, response.*statusText*);

}

} catch (error) {

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

}

};

const handleUpdateTask = async () => { try {

const response = await fetch(`http://localhost:5000/api/tasks/${editTask.id}`, { method: 'PUT', headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ task: updateTask })

});

if (response.*ok*) {

const updatedTask = await response.json(); setUpdateTask(updatedTask.task); *// Update the updateTask state with the new value*

const updatedTasks = tasks.map((task) =>

task.\_id === updatedTask.\_id ? updatedTask : task

);

setTasks(updatedTasks); setEditTask({ id: null, title: '' });

} else {

console.error('Failed to update task. Server returned:', response.*status*, response.*statusText*);

}

} catch (error) {

console.error('Error updating task:', error);

}

};

const handleDeleteTask = async (id)

=> { try {

const response = await fetch(`http://localhost:5000/api/tasks/${id}`, { method: 'DELETE'

});

if (response.*ok*) {

*// If the deletion was successful on the server, update the state* const updatedTasks = tasks.filter(task

=> task.\_id !== id); setTasks(updatedTasks);

} else {

*// Handle server error or task not found*

console.error('Failed to delete task. Server returned:', response.*status*, response.*statusText*);

}

} catch (error) {

*// Handle network or other errors*

console.error('Error deleting task:', error);

}

};

return (

<div className="App">

<h1>Task List</h1>

<div>

<input type="tex t"

placeholder="Enter a task" value={newTask}

onChange={(e) => setNewTask(e.target.value)}

/>

<button onClick={handleAddTask}>Add Task</button>

</div>

<ul>

{tasks.map((task) => (

<li key={task.\_id}>

{task.task}{' '}

<button onClick={() => handleDeleteTask(task.\_id)}>Delete</button>

<button onClick={() => { setEditTask({ id: task.\_id, title:

task.task }); setUpdateTask(task.task);

}}>Edit</button>

{editTask.id === task.\_id && (

<div> <input type="text"

placeholder="En ter updated task" value={updateTask}

onChange={(e) => setUpdateTask(e.target.value)}

/>

<button onClick={handleUpdateTask}>Update</button>

</div>

)}

</li>

))}

</ul>

</div>

);

}

export default App;

# Server.js

const express = require('express'); const mongoose = require('mongoose'); const bodyParser = require('body-parser'); const cors = require('cors');

const dotenv = require('dotenv') dotenv.config();

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

const uri = process.env.MONGODB\_URI;

app.use(cors()); app.use(bodyParser.json());

mongoose.connect(uri

, { useNewUrlParser:

true, useUnifiedTopology:

true

});

const taskSchema = new mongoose.Schema({ task: { type: String, required: true

}

});

const Task = mongoose.model('Task', taskSchema); app.post('/api/tasks', async (req, res) => {

const { task } = req.body; const newTask = new Task({ task

}); await newTask.save(); res.json(newTask);

});

app.get('/tasks', async (req, res) => { const tasks = await Task.find(); res.json(tasks);

});

const { id } = req.params; const { task

} = req.body; try

{

const updatedTask = await Task.findByIdAndUpdate(id, { task }, { new: true }); if (!updatedTask) {

return res.status(404).json({ message: 'Task not found' });

}

res.json(updatedTask);

} catch (error) {

console.error('Error updating task:', error); res.status(500).json({ message: 'Internal server error' });

}

});

app.delete('/api/tasks/:id', async (req, res) =>

{ const { id } = req.params; try { const deletedTask = await

Task.findByIdAndDelete(id); if (!deletedTask)

{

console.log(`Task with ID ${id} not found.`);

return res.status(404).json({ message: 'Task not found' });

}

res.json({ message: 'Task deleted successfully' });

} catch (error) {

console.error('Error deleting task:', error); res.status(500).json({ message: 'Internal server error' });

}

});

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`); });

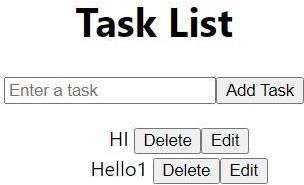
**Output: Read: -**

**Create: -**



**Delete:**



**Update: -**



**Conclusion:** Learnt and implemented database CRUD operations. The implementation of Mongoose CRUD operations with RESTful APIs simplifies data management in MongoDB.