

Practical 7

Project: User Profile Manager

Objective: Set up MongoDB and Mongoose in a Node.js application and create a basic schema to perform simple database operations.

Tasks

- Install MongoDB locally or create a free cluster on MongoDB Atlas.
- Set up a new Node.js project and install the required dependencies (express, mongoose, dotenv).
- Create a .env file and store your MongoDB connection URI securely.
- Write a connection script in Node.js using Mongoose to connect to MongoDB.
- Define a User schema with fields like name, email, and age.
- Create a simple script to insert a user into the database and log the results in the console.
- Fetch all users from the database and display them in the console

Screenshot of code:

index.js



```
.env JS index.js X JS db.js JS User.js JS ...
User-Profile-Manager > JS index.js > ...
1  const express = require("express");
2  const connectDB = require("../db");
3  const User = require("../models/User");
4
5  const app = express();
6  app.use(express.json());
7
8  connectDB();
9
10 const insertUser = async () => {
11   try {
12     const user = new User({
13       name: "Yash Bhalodiya",
14       email: "d24it155@charusat.edu.in",
15       age: 19,
16     });
17     const result = await user.save();
18     console.log("User inserted:", result);
19   } catch (err) {
20     console.error("Error inserting user:", err.message);
21   }
22 };
23
24 const fetchUsers = async () => {
25   try {
26     const users = await User.find();
27     console.log("All users:", users);
28   } catch (err) {
29     console.error("Error fetching users:", err.message);
30   }
31 };
32
33 app.listen(3000, async () => {
34   console.log("Server running on http://localhost:3000");
35   await insertUser();
36   await fetchUsers();
37 });
38
```

db.js

```
.env JS index.js JS db.js X JS User.js db.txt
User-Profile-Manager > JS db.js > ...
1  const mongoose = require("mongoose");
2  require("dotenv").config();
3
4  const connectDB = async () => {
5    try {
6      await mongoose.connect(process.env.MONGODB_URI, {
7        useNewUrlParser: true,
8        useUnifiedTopology: true,
9      });
10     console.log("MongoDB connected successfully");
11   } catch (error) {
12     console.error("MongoDB connection failed:", error.message);
13     process.exit(1);
14   }
15 };
16
17 module.exports = connectDB;
18
```

models/User.js

```
.env JS index.js JS db.js JS User.js X
User-Profile-Manager > models > JS User.js > ...
1  const mongoose = require("mongoose");
2
3  const userSchema = new mongoose.Schema({
4    name: {
5      type: String,
6      required: true,
7    },
8    email: {
9      type: String,
10     required: true,
11     unique: true,
12   },
13   age: {
14     type: Number,
15     required: true,
16   },
17 });
18
19 const User = mongoose.model("User", userSchema);
20
21 module.exports = User;
22
```

Screenshot of Output:

```
PS D:\Semester 4\FSWD\Practical 7\User-Profile-Manager> node index.js
(node:18876) [MONGODB DRIVER] Warning: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:18876) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Server running on http://localhost:3000
MongoDB connected successfully
User inserted: {
  name: 'ABC',
  email: 'abc@mail.com',
  age: 20,
  _id: new ObjectId('67f3d708dd1dd672b6a1922e'),
  __v: 0
}
All users: [
  {
    _id: new ObjectId('67f3d5270988b3e3446793fb'),
    name: 'Yash Bhalodiya',
    email: 'd24it155@charusat.edu.in',
    age: 19,
    __v: 0
  },
  {
    _id: new ObjectId('67f3d708dd1dd672b6a1922e'),
    name: 'ABC',
    email: 'abc@mail.com',
    age: 20,
    __v: 0
  }
]
```

The screenshot displays the MongoDB Atlas web interface. The top navigation bar includes the Atlas logo, a dropdown menu for 'd24it155's O...', and links for 'Access Manager' and 'Billing'. On the right, there are links for 'All Clusters', 'Get Help', and a dropdown for 'd24it155'. The main interface is divided into a left sidebar and a main content area. The sidebar has a 'Practical 7' dropdown and a 'Data Services' tab. The main content area has a 'Collections' tab selected, showing the 'test.users' collection. The collection details include 'STORAGE SIZE: 36KB', 'LOGICAL DATA SIZE: 1798', 'TOTAL DOCUMENTS: 2', and 'INDEXES TOTAL SIZE: 72KB'. Below this, there are tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. The 'Find' tab is active, showing a query filter bar with the text 'Type a query: { field: 'value' }'. Below the filter bar, the 'QUERY RESULTS: 1-2 OF 2' are displayed, showing two documents. The first document is:

```
{
  "_id": "67f3d5270988b3e3446793fb",
  "name": "Yash Bhalodiya",
  "email": "d24it155@charusat.edu.in",
  "age": 19,
  "__v": 0
}
```

 The second document is:

```
{
  "_id": "67f3d708dd1dd672b6a1922e",
  "name": "ABC",
  "email": "abc@mail.com",
  "age": 20,
  "__v": 0
}
```

Project: Task Manager API

Objective: Build a RESTful API with Express and Mongoose to manage tasks in a MongoDB collection.

Tasks

- Define a Task schema with fields like title, description, status (e.g., "Pending", "Completed"), and dueDate.
- Set up the following API endpoints:
- POST /tasks: Add a new task to the database.
- GET /tasks: Retrieve all tasks.
- GET /tasks/:id: Retrieve a specific task by ID.
- PUT /tasks/:id: Update task details by ID.
- DELETE /tasks/:id: Delete a task by ID.
- Test the API endpoints using Postman or Thunder Client.
- Use filters to query tasks based on their status or dueDate.
- Handle basic errors, such as invalid task IDs or missing fields.

Screenshot of Code:

index.js

```
JS index.js x
Task-Manager-API > JS index.js > ...
1  const express = require("express");
2  const connectDB = require("./db");
3  const taskRoutes = require("./routes/taskRoutes");
4  require("dotenv").config();
5
6  const app = express();
7  app.use(express.json());
8
9  connectDB();
10
11 app.use("/tasks", taskRoutes);
12
13 const PORT = process.env.PORT || 3000;
14 app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
15
```

db.js

```
JS db.js x
Task-Manager-API > JS db.js > [?] connectDB
1  const mongoose = require("mongoose");
2  require("dotenv").config();
3
4  const connectDB = async () => {
5    try {
6      await mongoose.connect(process.env.MONGODB_URI);
7      console.log("MongoDB connected");
8    } catch (error) {
9      console.error("MongoDB connection failed:", error.message);
10     process.exit(1);
11   }
12 };
13
14 module.exports = connectDB;
15
```

routes/taskRoutes.js

JS taskRoutes.js X

Task-Manager-API > routes > JS taskRoutes.js > ...

```
1  const express = require("express");
2  const router = express.Router();
3  const Task = require("../models/Task");
4
5  // POST /tasks - Create a new task
6  router.post("/", async (req, res) => {
7    try {
8      const task = new Task(req.body);
9      await task.save();
10     res.status(201).json(task);
11   } catch (err) {
12     res.status(400).json({ error: err.message });
13   }
14 });
15
16 // GET /tasks - Get all tasks
17 router.get("/", async (req, res) => {
18   try {
19     const { status, dueDate } = req.query;
20     const filter = {};
21     if (status) filter.status = status;
22     if (dueDate) filter.dueDate = { $lte: new Date(dueDate) };
23
24     const tasks = await Task.find(filter);
25     res.json(tasks);
26   } catch (err) {
27     res.status(500).json({ error: err.message });
28   }
29 });
30
31 // GET /tasks/:id - Get a task by ID
32 router.get("/:id", async (req, res) => {
33   try {
34     const task = await Task.findById(req.params.id);
35     if (!task) return res.status(404).json({ error: "Task not found" });
36     res.json(task);
37   } catch (err) {
38     res.status(400).json({ error: "Invalid ID" });
39   }
40 });
41
42 // PUT /tasks/:id - Update a task
43 router.put("/:id", async (req, res) => {
44   try {
45     const task = await Task.findByIdAndUpdate(req.params.id, req.body, {
46       new: true,
47       runValidators: true,
48     });
49   }
```

```
48     });
49     if (!task) return res.status(404).json({ error: "Task not found" });
50     res.json(task);
51   } catch (err) {
52     res.status(400).json({ error: err.message });
53   }
54 });
55
56 // DELETE /tasks/:id - Delete a task
57 router.delete("/:id", async (req, res) => {
58   try {
59     const task = await Task.findByIdAndDelete(req.params.id);
60     if (!task) return res.status(404).json({ error: "Task not found" });
61     res.json({ message: "Task deleted" });
62   } catch (err) {
63     res.status(400).json({ error: "Invalid ID" });
64   }
65 });
66
67 module.exports = router;
68
```

models/Task.js

```
JS Task.js ×
Task-Manager-API > models > JS Task.js > ...
1  const mongoose = require("mongoose");
2
3  const taskSchema = new mongoose.Schema({
4    title: {
5      type: String,
6      required: true,
7    },
8    description: String,
9    status: {
10     type: String,
11     enum: ["Pending", "Completed"],
12     default: "Pending",
13   },
14   dueDate: {
15     type: Date,
16     required: true,
17   },
18 }, { timestamps: true });
19
20 const Task = mongoose.model("Task", taskSchema);
21
22 module.exports = Task;
23
```

Output:

```
PS D:\Semester 4\FSWD\Practical 7\Task-Manager-API> node index.js
Server running on port 3000
MongoDB connected
```

DATABASES: 1 COLLECTIONS: 2

+ Create Database

Q Search Namespaces

▼ test

tasks

users

test.tasks

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 202B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 4KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes

[Generate queries from natural language in Compass](#)

Filter

Type a query: { field: 'value' }

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('67f3dacfd7edb7aad9f5bb87')
title: "Finish Node.js project"
description: "Create REST API using Express and Mongoose"
status: "Completed"
dueDate: 2025-02-28T00:00:00.000+00:00
createdAt: 2025-04-07T14:01:51.317+00:00
updatedAt: 2025-04-07T14:01:51.317+00:00
__v: 0
```

HTTP <http://localhost:3000/tasks>

POST <http://localhost:3000/tasks>

Params Authorization Headers (8) Body Scripts Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "title": "Finish Node.js project",
3   "description": "Create REST API using Express and Mongoose",
4   "status": "Completed",
5   "dueDate": "2025-02-28"
6 }
7
```

Body Cookies Headers (7) Test Results

{ JSON Preview Visualize

```
1 {
2   "title": "Finish Node.js project",
3   "description": "Create REST API using Express and Mongoose",
4   "status": "Completed",
5   "dueDate": "2025-02-28T00:00:00.000Z",
6   "_id": "67f3dacfd7edb7aad9f5bb87",
7   "createdAt": "2025-04-07T14:01:51.317Z",
8   "updatedAt": "2025-04-07T14:01:51.317Z",
9   "__v": 0
10 }
```

GitHub: <https://github.com/BhalodiyaYash155/Practical-7>

Conclusion: From This Practical I learn the following topics in node.js

- **What is MongoDB? NoSQL vs SQL databases**
- **Installing MongoDB locally and an introduction to MongoDB Atlas**
- **Setting up a Node.js project**
- **Installing and configuring Mongoose for MongoDB integration**
- **Understanding schemas and models in Mongoose**
- **Implementing Create, Read, Update, and Delete operations with Mongoose**
- **Introduction to RESTful API routes using Express**
- **Querying documents with filters and projections**