**Node.js CRUD Operations Using Mongoose and MongoDB Atlas**

Last Updated : 09 Oct, 2024

Before we look into the major operations and functionality of Mongoose, let us get a brief idea about what it is and how it makes our Node.js project more flexible and user-friendly.

**Prerequisites:**

* [Node.js and NPM](https://www.geeksforgeeks.org/node-js-npm-node-package-manager/)
* [Mongoose](https://www.geeksforgeeks.org/mongoose-tutorial/)
* [MongoDB](https://www.geeksforgeeks.org/mongodb-tutorial/)
* [Postman](https://www.geeksforgeeks.org/introduction-postman-api-development/)

**MongooseJS**

Mongoose is basically a package that serves as a mediator between the **NodeJS application** and the **MongoDB server**. It is an **Object Document Mapper(ODM)** that allows us to define objects with a strongly-typed schema that is mapped to a MongoDB document. Mongoose supports all the **CRUD operations –** **Creating**, **Retrieving**, **Updating**, and **Deleting**.

Performing CRUD operations with Mongoose and MongoDB Atlas is essential for building data-driven applications. To master database management in full-stack applications, the [**Full Stack Development with Node JS**](https://gfgcdn.com/tu/SY7/)**course** provides hands-on tutorials on connecting, querying, and managing MongoDB databases in Node.js applications.

**Approach**

To implement crud operation in Node using Mongoose and MongoDB atlas, we will first set-up a node project and install the mongoose module using NPM. Then setup the mongoDB atlas and define schema for users/students. At last, create the routes to make API calls and define the respective CRUD operations.

**Steps to Implement CRUD using Mongoose and MongoDB in Node**

**Step 1: Initialize Node project**

npm init

**Step 2: Mongoose Installation**

Install the Mongoose and the express module through npm using the below command:

npm install express mongoose --save

**can try it or install mongodb and mongodbcompass**

**Step 3: MongoDB Atlas SetUp**

1. Set up an account.
2. Build a new cluster.
3. Go to Database Access and hit **“Add New User**“. Add a username and password, if you autogenerate a password make sure you copy it, we’ll need it later.
4. Whitelist your IP Address. Hit “**Add Current IP address**” and Confirm.
5. Go to Clusters, if your cluster building is done then hit Connect, **“Connect Your Application**“, and copy the URL it gives you.

**Step 4: Postman Setup**

We will be using Postman to manage our requests. Once it is downloaded, hit the “**Create a request**” option. Every time we make a new API endpoint we’ll be setting up another request for it. This will help you manage everything so you don’t have to **copy/paste HTTP requests** everywhere.

**Step 5: Server Setup**

Here, we’ll set up our server on **port 3000** and call the express function that returns a server object in a variable named app. Then we start the listener saying app. Listen with the port address. Finally, we create the /api route which will be triggered once the request**localhost:3000/api** is received from the browser.

**Example:**This example demonstrates setting up basic express app.

JavaScript



1

// Filename - Server.js

**const express = require('express');**

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

**//const api = require('./api');**

**const port = 3000;**

**const app = express();**

**app.listen(port, function () {**

**console.log("Server is listening at port:" + port);**

**});**

**// Parses the text as url encoded data**

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

**// Parses the text as json**

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

**//app.use('/api', api);**

**after it in terminal**

**node server.js**

**Step 6: Create Schema**

Schema is a representation of the structure of the data. It allows us to decide exactly what data we want, and what options we want the data to have as an object.

**Example:**This example creates mongoose schema for Student having student id, name, roll no, birthday and address.

JavaScript



1

// Filename : studentschema.js

2

​

3

// Filename : studentschema.js

const mongoose = require('mongoose');

const StudentSchema = new mongoose.Schema({

StudentId: Number,

Name: String,

Roll: Number,

Birthday: Date,

Address: String

});

module.exports = mongoose.model(

'student', StudentSchema, 'Students');

A schema named “**StudentSchema”** is created that accepts the **fields Id,** **Name**, **Roll**, **Birthday**, and **Address**.

Models basically provide a list of predefined methods that are used to manipulate the **data for inserting**, **updating**, **deleting**, and **retrieving**from the database collection.

With that basic pattern, we’ll use the mongoose.model method to make it usable with actual data and export it so that we can use it in api.js.

**Step 6: Advanced Routing and MongoDB Connections:**

**Example:** When you make a request to **localhost:3000/api**, express will search for api route and execute the api.js file.

JavaScript



1

// Filename : api.js

2

​

3

const mongoose = require('mongoose');

4

const express = require('express');

5

const router = express.Router();

6

const StudentModel = require('./studentschema');

7

​

8

// Connecting to database

9

const query = 'mongodb+srv://Username:<password>'

10

+ '@student.tuufn.mongodb.net/College?'

11

+ 'retryWrites=true&w=majority'

12

​

13

const db = (query);

14

mongoose.Promise = global.Promise;

15

​

16

mongoose.connect(db, {

17

useNewUrlParser: true,

18

useUnifiedTopology: true

19

}, function (error) {

20

if (error) {

21

console.log("Error!" + error);

22

}

23

});

24

​

25

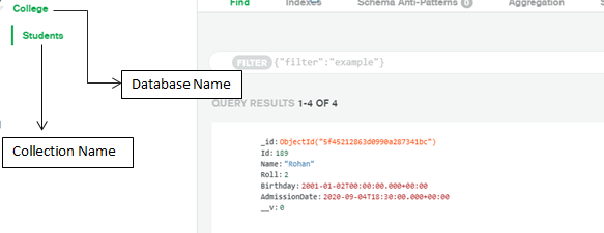
module.exports = router;

add in server.js

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

app.use('/api', api);

The database is **College**and the collection inside the database in **Students.**



*A Glimpse of the Mongo Database*

**Step 7: Implementing CRUD Operations**

* **Create:** We’ll be setting up a post request to ‘**/save**‘ and we’ll create a new student object with our model and pass with it the request data from **Postman**.

Once this is done, we will use **.save()** to save it to the database.

router.get('/save', function (req, res) {  
 const newStudent = new StudentModel({  
 StudentId: 101,  
 Name: "Sam", Roll: 1, Birthday: 2001 - 09 - 08  
 });  
  
 newStudent.save(function (err, data) {  
 if (err) {  
 console.log(error);  
 }  
 else {  
 res.send("Data inserted");  
 }  
 });  
});

A new instance of the student is created using StudentModel and the reference is stored in the variable newStudent. Using the newStudent variable we save the document of the new student to the database collection.

For achieving this, in Postman we will make a GET request **localhost:3000/api/save**

**Note:** We can even insert new documents without hardcoding the fields as done above. For that, we need to change the request from **GET** to **POST**and use the body-parser middleware to accept the new student’s data. This ensures that we can insert details of as many students as we need.

router.post('/save', function (req, res) {  
 const newStudent = new StudentModel();  
 newStudent.StudentId = req.body.StudentId;  
 newStudent.Name = req.body.Name;  
 newStudent.Roll = req.body.Roll;  
 newStudent.Birthday = req.body.Birthday;  
  
 newStudent.save(function (err, data) {  
 if (err) {  
 console.log(error);  
 }  
 else {  
 res.send("Data inserted");  
 }  
 });  
});

* **Retrieve:** To retrieve records from a database collection we make use of the **.find() function**.

router.get('/findall', function (req, res) {  
 StudentModel.find(function (err, data) {  
 if (err) {  
 console.log(err);  
 }  
 else {  
 res.send(data);  
 }  
 });  
});

In Postman, we make a new **GET reques**t with the URL**localhost:3000/api/findall** and hit send. It makes our **HTTP GET request** and returns documents of all the students from our database collection.

* To retrieve a single record or the first matched document we make use of the function **findOne()**.

router.get('/findfirst', function (req, res) {  
 StudentModel.findOne({ StudentId: { $gt: 185 } },  
 function (err, data) {  
 if (err) {  
 console.log(err);  
 }  
 else {  
 res.send(data);  
 }  
 });  
});

In Postman, we make a new GET request with the URL **localhost:3000/api/findfirst** and hit send.It makes our**HTTP GET request** and returns the first document that matches the condition **StudentId:$gt:185**($gt means greater than).

* **Delete:** To delete a record from the database, we make use of the function **.remove()**. It accepts a condition that is the parameter according to which it performs deletion. Here the condition is **Id:188**.

router.get('/delete', function (req, res) {  
 StudentModel.remove({ StudentId: 188 },  
 function (err, data) {  
 if (err) {  
 console.log(err);  
 }  
 else {  
 res.send(data);  
 }  
 });  
});

* We can also use the **.findByIdAndDelete()** method to easily remove a record from the database. Every object created with Mongoose is given its own **\_id**, and we can use this to target specific items with a **DELETE request**.

router.post('/delete', function (req, res) {  
 StudentModel.findByIdAndDelete((req.body.id),  
 function (err, data) {  
 if (err) {  
 console.log(err);  
 }  
 else {  
 res.send(data);  
 console.log("Data Deleted!");  
 }  
 });  
});

* **Update:** Just like with the delete request, we’ll be using the**\_id** to target the correct item. **.findByIdAndUpdate()** takes the target’s id, and the request data you want to replace it with.

router.post('/update', function (req, res) {  
 StudentModel.findByIdAndUpdate(req.body.id,  
 { Name: req.body.Name }, function (err, data) {  
 if (err) {  
 console.log(err);  
 }  
 else {  
 res.send(data);  
 console.log("Data updated!");  
 }  
 });  
});

**How to retrieve the latest record from database collection?**

To retrieve the latest record we need two basic functions:

* **.sort()** – It accepts a parameter according to which it sorts the data in **descending (-1)** or**ascending(1)** order.
* **.limit()** – It decides the number of documents needed to be retrieved.

**Example:** Suppose I want to fetch the record of the student who has most recently taken admitted to the College. The following code snippet does this job for us.

