

# Node.js CRUD Operations Using Mongoose and MongoDB Atlas

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

**Prerequisites:** Since we will be using Express to set up our basic server, We would recommend going through some articles on express and official express documents. Other requirements include MongoDB Atlas and Postman.

**Installation:** Install the mongoose and the express module through npm using the below command:

```
npm install express mongoose --save
```

## MongoDB Atlas Setup:

- Setup an account.
- Build a new cluster.
- 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.
- Whitelist your IP Address.Hit “Add Current IP address” and Confirm.
- Go to Clusters, if your cluster building is done then hit Connect, “Connect Your Application”, and copy the URL it gives you.

**Postman Setup:** We will be using Postman to manage our requests. Once it is downloaded, hit “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.

**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 request localhost:3000/api is received from the browser.

## File Name : Server.js

- Javascript

```
const express=require('express');
const bodyParser=require('body-parser');
const api = require('./api');

const port=3000;
const app=express();
```

```

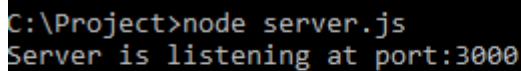
app.listen(port, function() {
  console.log("Server is listening at port:" + port);
});

// 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);

```



```

C:\Project>node server.js
Server is listening at port:3000

```

*Server running on desired port*



*This is geeksforgeeks*

*Sample output to check working of api route*

**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.

**Filename : studentschema.js**

- Javascript

```

var mongoose=require('mongoose');

var 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, 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 in api.js.

### **Advanced Routing and MongoDB Connections:**

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

- Javascript

```
var mongoose = require('mongoose');
var express = require('express');
var router = express.Router();
var StudentModel = require('./studentschema');

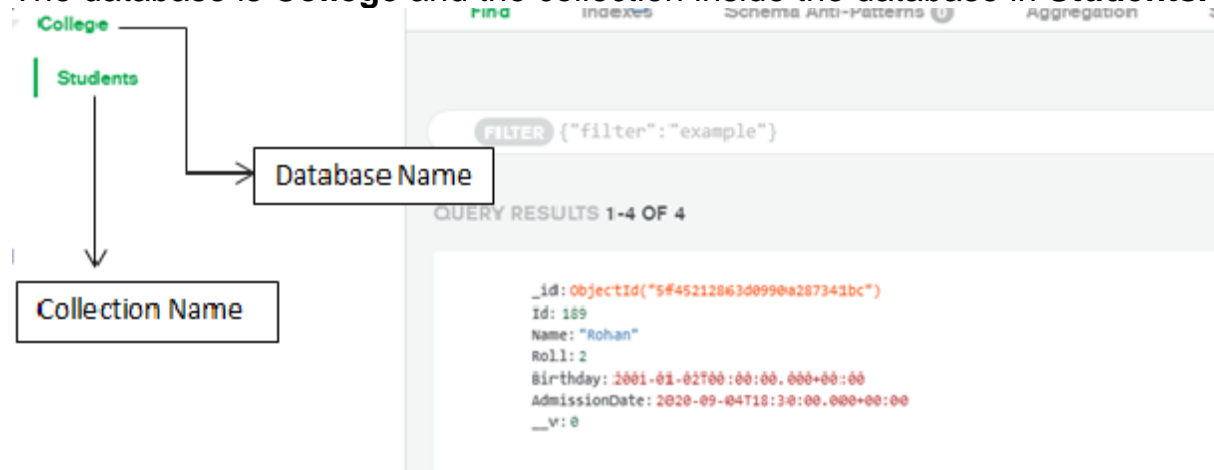
// Connecting to database
var query = 'mongodb+srv://Username:<password>'
  + '@student.tuufn.mongodb.net/College?'
  + 'retryWrites=true&w=majority'

const db = (query);
mongoose.Promise = global.Promise;

mongoose.connect(db, { useNewUrlParser : true,
  useUnifiedTopology: true }, function(error) {
  if (error) {
    console.log("Error!" + error);
  }
});

module.exports = router;
```

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



*A Glimpse of the Mongo Database*

## 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) {  
  var 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) {  
  var 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 request 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 match the condition **StudentId:\$gt:185** (\$gt means greater than).

- **Delete:** To delete a record from 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 admission to the College. The following code snippet does this job for us..

```
99  //retrieving the latest record
100  router.get('/latest',function(req,res){
101      StudentModel.find({}).sort({AdmissionDate : -1}).limit(1).exec(function(err,data){
102          if(err)
103          {
104              console.log(err);
105          }
106          else
107          {
108              res.send(data);
109          }
110      });
111  });
```

*Code Snippet that retrieves the latest data*

**NOTE:** `limit()` should not be used without `.sort()` as it may cause bugs later that are difficult to track down. This is because we can't otherwise guarantee the order of the result. We would get different records at the top of the results which isn't desirable. To make a query deterministic, they must give the same results every time they are executed.



## File Upload using Mongoose

### Create Model

Create Models directory and inside this directory create imageModel.js file; Then add following code into it:

```
1
2
3
4 const mongoose = require("../database");
5 var mongoose = require('mongoose');
6 var imageSchema = new mongoose.Schema({
7   name: String,
8   desc: String,
9   img:
10  {
11    data: Buffer,
12    contentType: String
13  }
14 });
15});
16//Image is a model which has a schema imageSchema
17module.exports = new mongoose.model('Image', imageSchema);
```

### Server Code:

```
const express = require("express");
const app = express();
const bodyParser = require("body-parser");
const path = require('path');
const fs = require("fs");
const multer = require("multer");
const mongoose = require("mongoose");
var imageModel = require('../models/imageModel');

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

app.set("view engine","ejs");
```

```

// SET STORAGE
var storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'uploads')
  },
  filename: function (req, file, cb) {
    cb(null, file.fieldname + '-' + Date.now())
  }
})

var upload = multer({ storage: storage })

app.get("/",(req,res)=>{
  res.render("index");
})

app.post("/uploadphoto",upload.single('myImage'),(req,res)=>{
  var img = fs.readFileSync(req.file.path);
  var encode_img = img.toString('base64');
  var final_img = {
    contentType:req.file.mimetype,
    image:new Buffer(encode_img,'base64')
  };
  imageModel.create(final_img,function(err,result){
    if(err){
      console.log(err);
    }else{
      console.log(result.img.Buffer);
      console.log("Saved To database");
      res.contentType(final_img.contentType);
      res.send(final_img.image);
    }
  })
})

//Code to start server
app.listen(3000,function () {
  console.log("Server Started at PORT 2000");
})

```