npm install express mongoose --save

**File Name : 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); |

**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**

|  |
| --- |
| **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 in **Students.**

Graphical user interface, text, application

Description automatically generated

*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.