**Build Node.js, MongoDB, Express RESTful API From Scratch**

Creating web, mobile, or any software application using REST API is a better way to manage the scalability and performance of the web application. APIs are reusable code and considered best for scalability.

We all use various softwares, web, and mobile applications to make our life easy. We browse the internet without knowing the harm it could give to us, and it also doesn’t matter whether you are a software developer or not. Mostly every software application work on API pattern.

**API (Application Programming Interface)** is “a set of subroutine definitions, communication protocols, and tools for building software. … Good APIs makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.”

### **Understand REST in Software Engineering**

REST stands for Representational State Transfer, and it is used to fetch and manage the data using various stateless services.

CRUD (Create, Read, Update, Delete) operations are the internal part of the HTTP protocol, and these operations allow us to manipulate the data on the server through **RESTful APIs**.

#### HTTP Services Types:

|  |  |
| --- | --- |
| **Property** | **Detail** |
| **POST** | Create data or individual item |
| **GET** | Get an item or single data |
| **PUT** | Create or replace data |
| **PATCH** | Update data |
| **DELETE** | Delete a resource |

We are going to build RESTful APIs for school management system based on the CRUD operation pattern. We will create the REST endpoints with Node, Express, and MongoDB.

## Node.js RESTful API Project Setup

In the first step, create a project folder, where will keep all our code. Enter the below command in the terminal to create the project folder and enter into it.

mkdir node-express-rest-api && cd node-express-rest-api

Run **npm init** command to initialize the project via Node js, as soon as you enter the command.

You will be asked following questions along with entry point we are using index.js but you can name it anything you want:

{

"name": "express-rest-api",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC"

}

In next step, we are installing required npm modules to create RESTful APIs.

npm install --save express mongoose cors body-parser

Next, install **nodemon module**. It’ll save us from restarting the server every-time we make the changes in the project.

It observes changes in project files and modules and restarts the server as soon as any change occurs

npm install nodemon --save-dev

## Step 3: MongoDB Set up for REST API Tutorial

In this step, we are going to learn how to use Mongoose to set up MongoDB database connection in Node server.

Head over to **index.js** file, import the mongoose library at the top most part of the server file.

const mongoose = require("mongoose");

# Build Node.js, MongoDB, Express RESTful API From Scratch

Creating web, mobile, or any software application using REST API is a better way to manage the scalability and performance of the web application. APIs are reusable code and considered best for scalability.

We all use various softwares, web, and mobile applications to make our life easy. We browse the internet without knowing the harm it could give to us, and it also doesn’t matter whether you are a software developer or not. Mostly every software application work on API pattern.

**API (Application Programming Interface)** is “a set of subroutine definitions, communication protocols, and tools for building software. … Good APIs makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.”

### Understand REST in Software Engineering

REST stands for Representational State Transfer, and it is used to fetch and manage the data using various stateless services.

CRUD (Create, Read, Update, Delete) operations are the internal part of the HTTP protocol, and these operations allow us to manipulate the data on the server through **RESTful APIs**.

#### HTTP Services Types:

|  |  |
| --- | --- |
| **Property** | **Detail** |
| **POST** | Create data or individual item |
| **GET** | Get an item or single data |
| **PUT** | Create or replace data |
| **PATCH** | Update data |
| **DELETE** | Delete a resource |

We are going to build RESTful APIs for school management system based on the CRUD operation pattern. We will create the REST endpoints with Node, Express, and MongoDB.

## Step 1: Getting Started

Install node js

Also, pass the MongoDB database URL to the **mongoose.connect(‘…’)** method in the **backend/index.js** file.

## Step 2: Node.js RESTful API Project Setup

In the first step, create a project folder, where will keep all our code. Enter the below command in the terminal to create the project folder and enter into it.

**Node JS REST API** project name: node-express-rest-api

mkdir node-express-rest-api && cd node-express-rest-api

Run **npm init** command to initialize the project via Node js, as soon as you enter the command.

You will be asked following questions along with entry point we are using index.js but you can name it anything you want:

{

"name": "express-rest-api",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC"

}

In next step, we are installing required npm modules to create RESTful APIs.

npm install --save express mongoose cors body-parser

Next, install **nodemon module**. It’ll save us from restarting the server every-time we make the changes in the project.

It observes changes in project files and modules and restarts the server as soon as any change occurs.

npm install nodemon --save-dev

## Step 3: MongoDB Set up for REST API Tutorial

In this step, we are going to learn how to use Mongoose to set up MongoDB database connection in Node server.

Head over to **index.js** file, import the mongoose library at the top most part of the server file.

const mongoose = require("mongoose");

In this file we will use the **mongoose.connect()** method to connect the Node app to the MongoDB database.

Add code in the **index.js** file:

// Connecting MongoDB

async function mongoDbConnection() {

await mongoose.connect(

"mongodb://127.0.0.1:27017/restapi",

{

useNewUrlParser: true,

useUnifiedTopology: true,

},

6000,

);

}

mongoDbConnection().then(() => {

console.log("MongoDB successfully connected.");

}),

(err) => {

console.log("Could not connected to database : " + err);

};

Here, mongoDB works on port **27017** locally, and test is the database name.

## Step 4: Creating Student Schema Module

We have already installed Mongoose from NPM library to make the mongoDB connection. Mongoose is based on object data modeling (ODM). It helps in creating the data model using the schema pattern.

Enter the command to create the folder structure for student schema module from the root of your project folder

mkdir model && cd model && touch student.model.js

Go to **model/student.model.js** file and add the following code:

const mongoose = require("mongoose");

const Schema = mongoose.Schema;

let studentSchema = new Schema(

{

name: {

type: String,

},

email: {

type: String,

},

subject: {

type: String,

},

},

{

collection: "students",

},

);

module.exports = mongoose.model("StudentSchema", studentSchema);

## **Step 5: Build RESTful API with Express & Node JS**

Next, we will build RESTful APIs with Express js in Node js app. First, we’ll create routes folder and create a student.routes.js file in it.

Make sure to come out from the previous (model) directory. Then, run command to build the routes folder and **student.routes.js** file.

mkdir routes && cd routes && touch student.route.js

Then, in the student.route.js file, we will import express js and student schema model.

We can use the express js method to build robust and secure RESTful APIs, as mentioned below:

const express = require("express");

const studentExpressRoute = express.Router();

const cors = require("cors");

let StudentSchema = require("../model/student.model");

// CORS OPTIONS

var whitelist = ["http://localhost:8100", "http://localhost:4000"];

var corsOptionsDelegate = function (req, callback) {

var corsOptions;

if (whitelist.indexOf(req.header("Origin")) !== -1) {

corsOptions = {

origin: "\*",

methods: "GET,HEAD,PUT,PATCH,POST,DELETE",

};

} else {

corsOptions = { origin: false }; // disable CORS for this request

}

callback(null, corsOptions);

};

// Get users

studentExpressRoute

.route("/", cors(corsOptionsDelegate))

.get(async (req, res, next) => {

await StudentSchema.find()

.then((result) => {

res.json({

data: result,

message: "Data successfully fetched!",

status: 200,

});

})

.catch((err) => {

return next(err);

});

});

// Create user

studentExpressRoute.route("/create-student").post(async (req, res, next) => {

await StudentSchema.create(req.body)

.then((result) => {

console.log(result);

res.json({

data: result,

message: "Data successfully added.",

status: 200,

});

})

.catch((err) => {

return next(err);

});

});

// Get single user

studentExpressRoute.route("/get-student/:id").get(async (req, res, next) => {

await StudentSchema.findById(req.params.id, req.body)

.then((result) => {

res.json({

data: result,

message: "Data successfully retrieved.",

status: 200,

});

})

.catch((err) => {

return next(err);

});

});

// Update user

studentExpressRoute.route("/update-student/:id").put(async (req, res, next) => {

await StudentSchema.findByIdAndUpdate(req.params.id, {

$set: req.body,

})

.then((result) => {

res.json({

data: result,

msg: "Data successfully updated.",

});

})

.catch((err) => {

return next(err);

});

});

// Delete student

studentExpressRoute.route("/remove-student/:id").delete(async (req, res) => {

await StudentSchema.findByIdAndRemove(req.params.id)

.then(() => {

res.json({

msg: "Data successfully updated.",

});

})

.catch((err) => {

return next(err);

});

});

module.exports = studentExpressRoute;

## **Step 6: Node App Server Settings**

First, create a index.js file in the root of your project. In this file, we will keep our server settings. Add the given below code in server js file:

const express = require("express");

const mongoose = require("mongoose");

const cors = require("cors");

const bodyParser = require("body-parser");

const createError = require("http-errors");

// Connecting MongoDB

async function mongoDbConnection() {

await mongoose.connect(

"mongodb://127.0.0.1:27017/test",

{

useNewUrlParser: true,

useUnifiedTopology: true,

},

6000

);

}

mongoDbConnection().then(() => {

console.log("MongoDB successfully connected.");

}),

(error) => {

console.log("Could not connected to database : " + err);

};

const userRoute = require("./routes/student.route");

const app = express();

app.use(

bodyParser.urlencoded({

extended: true,

}),

bodyParser.json()

);

// CORS

app.use(cors());

// RESTful API root

app.use("/endpoint", userRoute);

// PORT

const port = process.env.PORT || 8080;

app.listen(port, () => {

console.log("PORT Connected on: " + port);

});

// Find 404 and hand over to error handler

app.use((req, res, next) => {

next(createError(404));

});

// error handler

app.use(function (err, req, res, next) {

console.error(err.message);

if (!err.statusCode) err.statusCode = 500;

res.status(err.statusCode).send(err.message);

});

As you can see in above file, we have performed the following tasks:

* MongoDB connection
* Express JS settings
* Defined REST API root
* Created PORT for Node app
* Express.js route error handling
* Setting up build location via express

Now, we have successfully produced Node, Express and MongoDB RESTful API with the following structure:

* name
* email
* subject

Start the following command to start the nodemon server.

npx nodemon server

**Given below table includes REST API’s and their respective methods:**

|  |  |
| --- | --- |
| **API Method** | **Endpoint** |
| **POST** | endpoint/create-student |
| **GET** | endpoint/(list students) |
|  |  |
| **GET** | endpoint/get-student/:id(get student from students list) |
| **PUT** | endpoint/update-student/:id(update specific student data) |
| **DELETE** | endpoint/remove-student/:id(delete specific student data) |

## **Step 7: Testing Express JS REST API**

In this last step, we are going to test our RESTful APIs locally. Follow the given below process to get started with testing.

**Express.js REST API URL:** <http://localhost:8080/endpoint>

## Step 8: Testing Node JS RESTful API with Postman

In order to test REST API in postman you need to have installed Postman in your machine. You can visit [here](https://www.getpostman.com/) to download the Postman API testing tool.

Once you are done downloading Postman then open it and select the HTTP method from the top left part. Then enter the REST API in the search bar and if you are getting the response similar as in the screenshot below, it means our REST API is working.

