

NEW App Platform: reimagining PaaS to make it simpler for you to build, deploy, and scale apps.



Community



TUTORIAL

How To Perform CRUD Operations with Mongoose and MongoDB Atlas

MongoDB Node.js

By Joshua Hall

Last Validated on March 22, 2021 • Originally Published on July 7, 2019 © 57.4k

Introduction


Mongoose is one of the fundamental tools for manipulating data for a Node.js and MongoDB backend.

In this article, you will be looking into using Mongoose with the MongoDB Atlas remote database. The example in this tutorial will consist of a list of food and their caloric values. Users will be able to create new items, read items, update items, and delete items.

Prerequisites

- Familiarity with using Express for a server. You can consult this article on Express and the official Express documentation.
- Familiarity with setting up REST API without any authentication. You can consult this article on routing with Express and this article on `async` and `await`.

Downloading and installing a tool like Postman is recommended for testing API endpoints.

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up

SCROLL TO TOP 

After creating an account and signing in, [follow these steps to deploy a free tier cluster](#).

Once you have set a cluster, a database user, and an IP address you will be prepared for later acquiring the connection string as you set up the rest of your project.

Step 1 — Setting Up the Project

In this section, you will create a directory for your project and install dependencies.

Create a new directory for your project:

```
$ mkdir mongoose-mongodb-atlas-example
```

Navigate to the newly created directory:

```
$ cd mongoose-mongodb-atlas-example
```

At this point, you can initialize a new npm project:

```
$ npm init -y
```


Next, install `express` and `mongoose`:

```
$ npm install express@4.17.1 mongoose@5.11.12
```

At this point, you will have a new project with `express` and `mongoose`.

Step 2 — Setting Up the Server

In this section, you will create a new file to run the Express server, connect to the MongoDB Atlas database, and import future routes.

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up

```
const foodRouter = require("./routes/foodRoutes.js");

const app = express();

app.use(express.json());

mongoose.connect(
  "mongodb+srv://madmin:<password>@clustername.mongodb.net/<dbname>?retryWrites=true",
  {
    useNewUrlParser: true,
    useFindAndModify: false,
    useUnifiedTopology: true
  }
);

app.use(foodRouter);

app.listen(3000, () => {
  console.log("Server is running...");
});
```


Pay attention to the connection string. This is the connection string that is provided by MongoDB Atlas. You will need to replace the administrator account (`madmin`), password, cluster name (`clustername`), and database name (`dbname`) with the values that are relevant to your cluster:

```
mongodb+srv://madmin:<password>@clustername.mongodb.net/<dbname>?retryWrites=true
```

`mongoose.connect()` will take the connection string and an object of configuration options. For the purpose of this tutorial, the `useNewUrlParser`, `useFindAndModify`, and `useUnifiedTopology` configuration settings are necessary to avoid a deprecation warning.

At this point, you have the start of an Express server. You will next need to define the schema and handle routes.

Step 3 — Building the Schema

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up

Create a new `models` directory:

```
$ mkdir models
```

Inside of this new directory, create a new `food.js` file and add the following lines of code:

```
./models/food.js
```

```
const mongoose = require("mongoose");

const FoodSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true,
    lowercase: true,
  },
  calories: {
    type: Number,
    default: 0,
    validate(value) {
      if (value < 0) throw new Error("Negative calories aren't real.");
    },
  },
});


const Food = mongoose.model("Food", FoodSchema);

module.exports = Food;
```

This code defines your `FoodSchema`. It will consist of a `name` value that is of type `String`, it will be `required`, `trim` any whitespace, and set to `lowercase` characters. It will also consist of a `calories` value that is of type `Number`, it will have a `default` of 0, and `validate` to ensure no negative numbers are submitted.

Step 4 — Building the Read Route

Once you have your data model set up, you can start setting up routes to use it. This

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up

Create a new `routes` directory:

```
$ mkdir routes
```

Inside of this new directory, create a new `foodRoutes.js` file and add the following lines of code:

```
./routes/foodRoutes.js
```

```
const express = require("express");
const foodModel = require("../models/food");
const app = express();

app.get("/foods", async (request, response) => {
  const foods = await foodModel.find({});


  try {
    response.send(foods);
  } catch (error) {
    response.status(500).send(error);
  }
});

module.exports = app;
```

This code establishes a `/foods` endpoint for GET requests (note the plural 's'). The Mongoose query function `find()` returns all objects with matching parameters. Since no parameters have been provided, it will return all of the items in the database.

Since Mongoose functions are asynchronous, you will be using `async/await`. Once you have the data this code uses a `try/catch` block to send it. This will be useful to verify the data with Postman.

Navigate to the root of your project directory and run your Express server with the following command in your terminal:

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up

Note: If you need assistance navigating the Postman interface for requests, consult [the official documentation](#).

The Postman results will display an empty array.

Step 5 — Building the Create Route

Next, you will build the functionality to create a new food item and save it to the database.

Revisit the `foodRoutes.js` file and add the following lines of code between `app.get` and `module.exports`:

```
./routes/foodRoutes.js

// ...

app.post("/food", async (request, response) => {
  const food = new foodModel(request.body);


  try {
    await food.save();
    response.send(food);
  } catch (error) {
    response.status(500).send(error);
  }
});

// ...
```

This code establishes a `/food` endpoint for POST requests. The Mongoose query function `.save()` is used to save data passed to it to the database.

In Postman, create a new request called **Create New Food**. Ensure the request type is set to POST. Set the **request URL** to `localhost:3000/food`.

In the **Body** section, select **raw** and **JSON**. Then, add a new food item by constructing

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up

```
{
  "name": "cotton candy",
  "calories": 100
}
```

After sending the **Create New Food** request, send the **Read All Food** request again. The Postman results will display the newly added object.

Step 6 — Building the Update Route

Every object created with Mongoose is given its own `_id` and you can use this to target specific items. It will be a mix of alphabetical characters and letters. For example: `5d1f6c3e4b0b88fb1d257237`.

Next, you will build the functionality to update an existing food item and save the changes to the database.

Revisit the `foodRoutes.js` file and add the following lines of code between `app.post` and `module.exports`:


`./routes/foodRoutes.js`

```
// ...

app.patch("/food/:id", async (request, response) => {
  try {
    await foodModel.findByIdAndUpdate(request.params.id, request.body);
    await foodModel.save();
    response.send(food);
  } catch (error) {
    response.status(500).send(error);
  }
});

// ...
```

This code establishes a `/food/:id` endpoint for PATCH requests. The Mongoose

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up

identifying string of the food you created previously.

In the **Body** section, select **raw** and **JSON**. Then, modify your food item by constructing a JSON object with a `name` and `calories`:

```
{
  "calories": "999"
}
```

After sending the **Update Food** request, send the **Read All Food** request again. The Postman results will display the object with modified `calories`.

Step 7 — Building the Delete Route

Finally, you will build the functionality to remove an existing food item and save the changes to the database.

Revisit the `foodRoutes.js` file and add the following lines of code between `app.patch` and `module.exports`:

`./routes/foodRoutes.js`

```
// ...

app.delete("/food/:id", async (request, response) => {
  try {
    const food = await foodModel.findByIdAndDelete(request.params.id);

    if (!food) response.status(404).send("No item found");
    response.status(200).send();
  } catch (error) {
    response.status(500).send(error);
  }
});

// ...
```

This code establishes a `/food/:id` endpoint for DELETE requests. The Mongoose

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up

After sending the **Delete Food** request, send the **Read All Food** request again. The Postman results will display an array without the item that was deleted.

Note: Now that you have completed this tutorial, you may wish to **Terminate** any MongoDB Atlas clusters that you are no longer using.

At this point, you have an Express server using Mongoose methods to interact with a MongoDB Atlas cluster.

Conclusion

In this article, you learned how to use Mongoose methods you can quickly make and manage your backend data.

If you'd like to learn more about Node.js, check out [our Node.js topic page](#) for exercises and programming projects.

If you'd like to learn more about MongoDB, check out [our MongoDB topic page](#) for exercises and programming projects.

Was this helpful?

Yes

No



[Report an issue](#)

About the authors



Joshua Hall



Bradley Kouchi

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics.



Sign Up

Still looking for an answer?



Ask a question



Search for more help

RELATED



App Platform: Run Node apps without managing servers

[Product](#)

How To Configure Keyfile Authentication for MongoDB Replica Sets on Ubuntu 20.04

[Tutorial](#)

How To Configure a MongoDB Replica Set on Ubuntu 20.04

[Tutorial](#)

Comments

2 Comments

Leave a comment...

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics.



Sign Up

^ [LukeAvedon](#) September 11, 2020

1 Great tutorial!

Think I may have found one small error:

In "File Setup" you refer to the file `./models/FoodModel.js` but later refer to the same file as `./models/food.js`. Perhaps, I missed something obvious? Thanks for the very helpful tutorial.

[Reply](#) [Report](#)

^ [Yacie](#) November 21, 2020

0 Goodday sir

Do you have a git repo for this.

I wud much appreciate if you have it hosted too

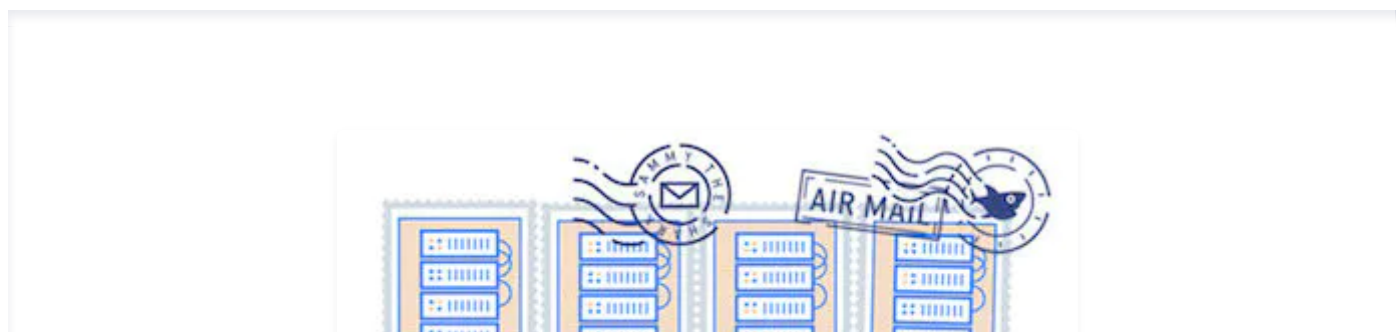
Or u want me to figure it out too

The challenge I been having with these tutorials is transitioning from 'localhost' to remote host. That's why I asked you the above questions

[Reply](#) [Report](#)

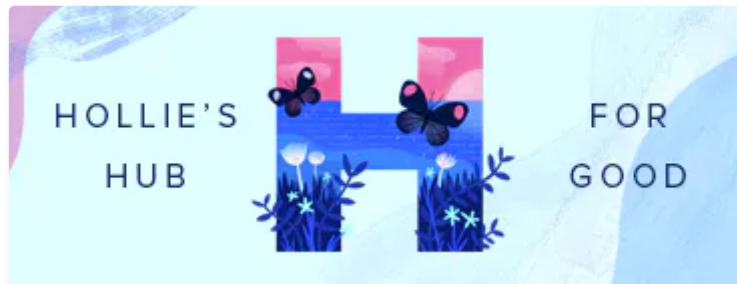


This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



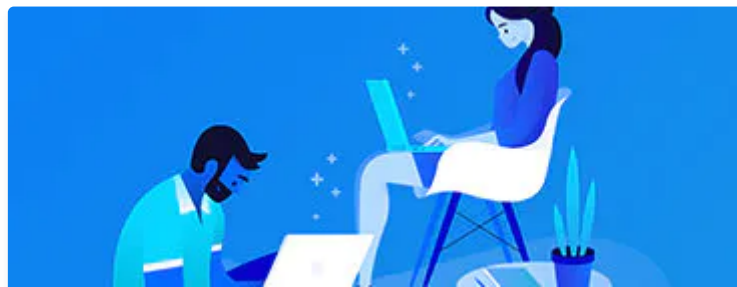
Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up



HOLLIE'S HUB FOR GOOD

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.



BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

Featured on [Community](#) [Kubernetes Course](#) [Learn Python 3](#) [Machine Learning in Python](#)
[Getting started with Go](#) [Intro to Kubernetes](#)

DigitalOcean Products [Virtual Machines](#) [Managed Databases](#) [Managed Kubernetes](#) [Block Storage](#)
[Object Storage](#) [Marketplace](#) [VPC](#) [Load Balancers](#)

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. ✕

Sign Up

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

[Learn More](#)



© 2021 DigitalOcean, LLC. All rights reserved.

Company

- [About](#)
- [Leadership](#)
- [Blog](#)
- [Careers](#)
- [Partners](#)
- [Referral Program](#)
- [Press](#)
- [Legal](#)
- [Security & Trust Center](#)

[Products](#)


[Community](#)

[Contact](#)

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. [×](#)

[Sign Up](#)

- | | |
|-------------------|------------------------|
| Spaces | Write for DigitalOcean |
| Marketplace | Presentation Grants |
| Load Balancers | Hatch Startup Program |
| Block Storage | Shop Swag |
| API Documentation | Research Program |
| Documentation | Open Source |
| Release Notes | Code of Conduct |

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics. 

Sign Up
