File Upload With Multer in Node.js and Express

by Esther Vaati 9 Nov 2018

Difficulty: Intermediate Length: Short Languages: English

Node.js Express Forms Back-End

When a web client uploads a file to a server, it is generally submitted through a form and encoded as `multipart/form-data`. Multer is middleware for Express and Node.js that makes it easy to handle this `multipart/form-data` when your users upload files.

In this tutorial, I'll show you how to use the Multer library to handle different file upload situations in Node.

How Does Multer Work?

As I said above, Multer is Express middleware. Middleware is a piece of software that connects different applications or software components. In Express, middleware processes and transforms incoming requests to the server. In our case, Multer acts as a helper when uploading files.

Project Setup

We will be using the Node Express framework for this project. Of course, you'll need to have Node installed.

Create a directory for our project, navigate into the directory, and issue `npm init` to create a **.json** file that manages all the dependencies for our application.

```
mkdir upload-express
cd  upload-express
npm init
```

Next, install Multer, Express, and the other dependencies necessary to bootstrap an Express app.

```
npm install express multer body-parser --save
```

Next, create a **server.js** file.

```
touch server.js
```

Then, in **server.js**, we will initialize all the modules, create an Express app, and create a server for connecting to browsers.

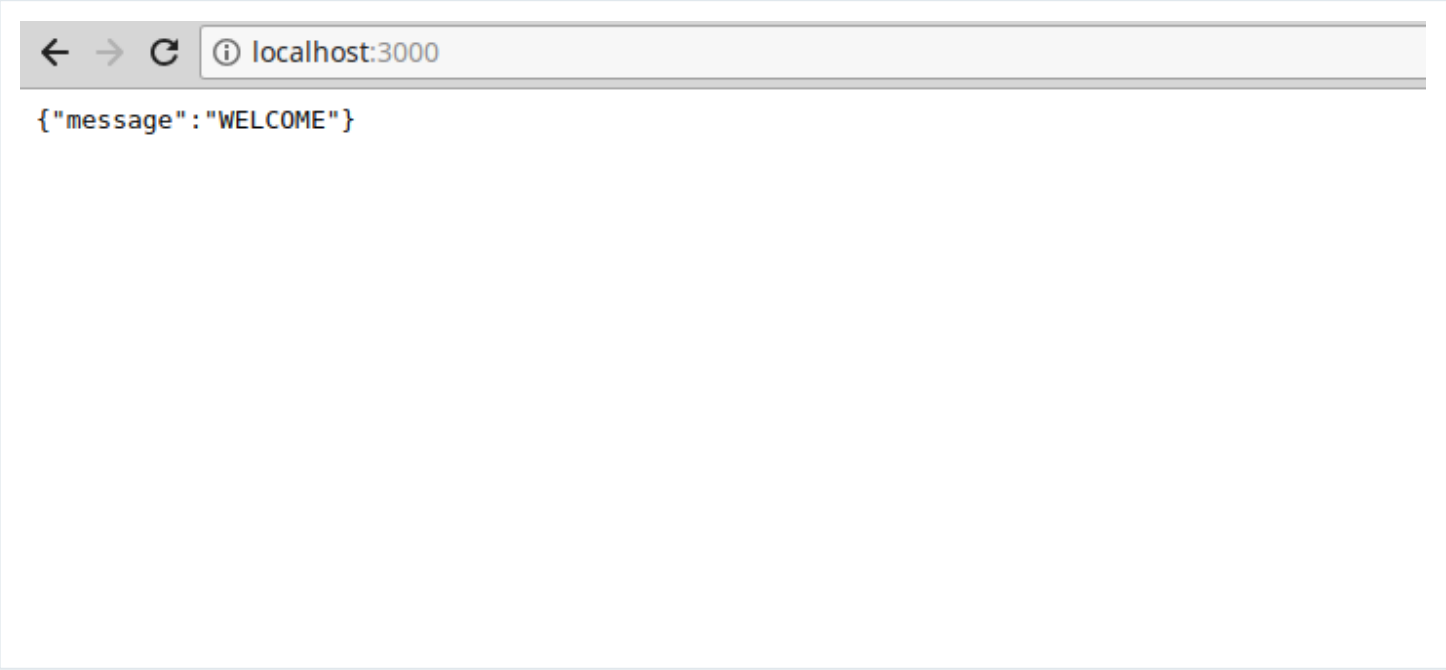
```
// call all the required packages
const express = require('express')
const bodyParser= require('body-parser')
const multer = require('multer');
app.use(bodyParser.urlencoded({extended: true}))

const app = express()

```

```
09 //CREATE EXPRESS APP
10 const app = express();
11
12 //ROUTES WILL GO HERE
13 app.get('/', function(req, res) {
14     res.json({ message: 'WELCOME' });
15 });
16
17 app.listen(3000, () => console.log('Server started on port 3000'));
```

Running `node server.js` and navigating to `localhost:3000` on your browser should give you the following message.



Create the Client Code

The next thing will be to create an **index.html** file to write all the code that will be served to the client.

```
touch index.html
```

This file will contain the different forms that we will use for uploading our different file types.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>MY APP</title>
</head>
<body>


 <!--  SINGLE FILE -->
<form action="/uploadfile" enctype="multipart/form-data" method="POST">
   <input type="file" name="myFile" />
   <input type="submit" value="Upload a file"/>
</form>


<!-- MULTIPLE FILES -->

<form action="/uploadmultiple"  enctype="multipart/form-data" method="POST">
  Select images: <input type="file" name="myFiles" multiple>
  <input type="submit" value="Upload your files"/>
</form>

  <!--   PHOTO-->

<form action="/upload/photo" enctype="multipart/form-data" method="POST">
  <input type="file" name="myImage" accept="image/*" />
  <input type="submit" value="Upload Photo"/>
</form>



</body>
</html>
```

Open **server.js** and write a GET route that renders the **index.html** file instead of the "**WELCOME**" message.

```
// ROUTES
app.get('/',function(req,res){
  res.sendFile(__dirname + '/index.html');

});
```

Multer Storage

The next thing will be to define a storage location for our files. Multer gives the option of storing files to disk, as shown below. Here, we set up a directory where all our files will be saved, and we'll also give the files a new identifier.

```
//server.js


// 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 })
```

Handling File Uploads

Uploading a Single File

In the **index.html** file, we defined an action attribute that performs a POST request. Now we need to create an endpoint in the Express application. Open the **server.js** file and add the following code:

```
app.post('/uploadfile', upload.single('myFile'), (req, res, next) => {
  const file = req.file
  if (!file) {
    const error = new Error('Please upload a file')
    error.httpStatusCode = 400
    return next(error)
```

```
  }
    res.send(file)

})
```

Note that the name of the file field should be the same as the `myFile` argument passed to the `upload.single` function.

Uploading Multiple Files

Uploading multiple files with Multer is similar to a single file upload, but with a few changes.

```
//Uploading multiple files
app.post('/uploadmultiple', upload.array('myFiles', 12), (req, res, next) => {
  const files = req.files
  if (!files) {
    const error = new Error('Please choose files')
    error.httpStatusCode = 400
    return next(error)
  }

    res.send(files)

})
```

Uploading Images

Instead of saving uploaded images to the file system, we will store them in a MongoDB database so that we can retrieve them later as needed. But first, let's install MongoDB.

```
npm install mongodb --save
```

We will then connect to MongoDB through the `Mongo.client` method and then add the MongoDB URL to that method. You can use a cloud service like Mlab, which offers a free plan, or simply use the locally available connection.

```
const MongoClient = require('mongodb').MongoClient
const myurl = 'mongodb://localhost:27017';

MongoClient.connect(myurl, (err, client) => {
  if (err) return console.log(err)

```

```
  db = client.db('test')
  app.listen(3000, () => {
    console.log('listening on 3000')
  })
})
```

Open **server.js** and define a POST request that enables the saving of images to the database.

```
app.post('/uploadphoto', upload.single('picture'), (req, res) => {
    var img = fs.readFileSync(req.file.path);
 var encode_image = img.toString('base64');
 // Define a JSONobject for the image attributes for saving to database

 var finalImg = {
      contentType: req.file.mimetype,
      image:  new Buffer(encode_image, 'base64')
   };
db.collection('quotes').insertOne(finalImg, (err, result) => {
    console.log(result)

    if (err) return console.log(err)

    console.log('saved to database')
    res.redirect('/')


  })
})
```

In the above code, we first encode the image to a base64 string, construct a new buffer from the base64 encoded string, and then save it to our database collection in JSON format.

We then display a success message and redirect the user to the index page.

Retrieving Stored Images

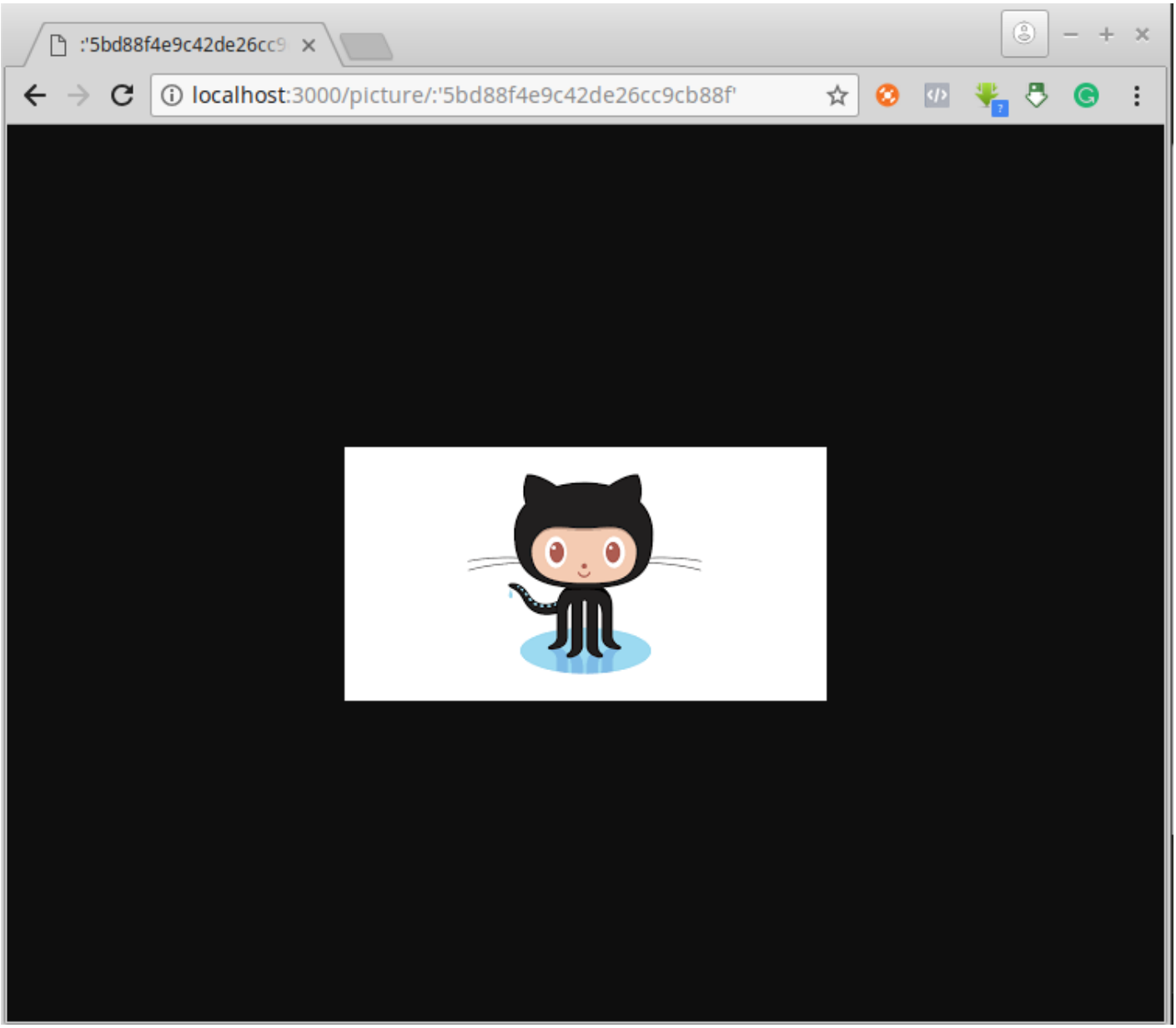
To retrieve the stored images, we perform a MongoDB search using the `find` method and return an array of results. We then go on and obtain the `_id` attributes of all the images and return them to the user.

```
app.get('/photos', (req, res) => {
db.collection('mycollection').find().toArray((err, result) => {
```

```
03
04        const imgArray= result.map(element => element._id);
05              console.log(imgArray);
06
07     if (err) return console.log(err)
08     res.send(imgArray)
09
10    })
11  });
```

Since we already know the id's of the images, we can view an image by passing its id in the browser, as illustrated below.

```
01  app.get('/photo/:id', (req, res) => {
02  var filename = req.params.id;
03
04  db.collection('mycollection').findOne({'_id': ObjectId(filename) }, (err, result
05
06      if (err) return console.log(err)
07
08     res.contentType('image/jpeg');
09     res.send(result.image.buffer)
10
11
12    })
13  })
```



Conclusion

I hope you found this tutorial helpful. File upload can be an intimidating topic, but it doesn't have to be hard to implement. With Express and Multer, handling `multipart/form-data` is easy and straightforward.

You can find the full source code for the file upload example in our GitHub repo.

Esther Vaati

Software developer

Esther is a software developer based in Kenya. She is very passionate about technology. She is also a programming instructor at Envato Tuts+. When she is not coding, she loves to watch movies and do volunteer work.

Kaleiesther

View on GitHub

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by

Tyav Moses • 2 months ago

Hi, very interesting article. I have a question which got me looking up every page to find its answer. if a multipart form-data is sent from a page an includes texts and file, I am aware that multer converts the text values to an JSON-like object and stores it as req.body and the files as req.files. so I want to ask, if I am to get the text informations before using multer, how do I locate it? or am I to use multer before I can?

an example of my question would be

```
/**
*a multipart/form-data containing:
* 1 - image: file.png
* 2 - text1 : "question"
* 3 - text2 : "answer"
*/

app.post((req, res, next)=>{
console.log(req.body) // undefined
//question is how do I access text1 and text2 at this point
},
upload.single("image"),
(req, res, next)=>{
```

```
console.log(req.body) // {text1: "question", text2:"answer"}
//here req.body has been converted by multer.
})
```

thanks

3 ^ v • Reply • Share ›

Thomas Dolhanty • 25 days ago

Very helpful for this node newbie. Thank you very much.

^ v • Reply • Share ›

Aravind Reddy • a month ago

i have a small doubt im a beginner for multer and ihave followed the process defined above and is working perfectly but if i want to update a image with new image the display will not happen.
if any one is aware of the solution to this please help me out

^ v • Reply • Share ›

linda alibi • 2 months ago • edited

there is a duplication of 'app'
this works:

```
// call all the required packages
const express = require('express')
const bodyParser= require('body-parser')
const multer = require('multer');
let app = express();
app.use(bodyParser.urlencoded({extended: true}))

//CREATE EXPRESS APP

//ROUTES WILL GO HERE
app.get('/', function(req, res) {
res.json({ message: 'WELCOME' });
});

app.listen(3000, () => console.log('Server started on port 3000'));
```

^ v • Reply • Share ›

Pankaj Kr • 3 months ago

Nice Article, But the code which exactly worked for me is: https://jsonworld.com/demo/...

^ v • Reply • Share ›

teja b • 3 months ago

thank you, it was useful to me.

^ v • Reply • Share ›

Daniel Goje • 5 months ago

Isn't there a bug on line 5 of the first code snipper? You are using app before declaring it.

Reply • Share ›

Padmaputra Aravind • 7 months ago

how to upload video using nodejs and mysql

Reply • Share ›

Tom Larry • 7 months ago

Great article!
Downloaded and tested the single file upload, after clicking on <upload a="" file=""> button got:
Error: ENOENT: no such file or directory, open 'c:\file-upload-with-multer-in-node-master\uploads\myFile-1546735756004'
Set a breakpoint in upload.single(), didn't hit that.
Please advise.

Reply • Share ›

Tom Larry ➚ Tom Larry • 7 months ago

solved it by simply create a "uploads" folder in the app's root folder.

1 Reply • Share ›

QUICK LINKS - Explore popular categories

ENVATO TUTS+ +

JOIN OUR COMMUNITY +

HELP +

tuts+

27,990 Tutorials 1,257 Courses 39,815 Translations