CODE > NODE.JS

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.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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()

//CREATE EXPRESS APP
const app = express();

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

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 PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Advertisement

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>
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
   <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=
  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="PO
  <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){
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
  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:

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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, r
  const files = req.files
  if (!files) {
    const error = new Error('Please choose files')
    error.httpStatusCode = 400
    return next(error)
  }

    res.send(files)

})
```

Uploading Images

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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 databa

 var finalImg = {
      contentType: req.file.mimetype,
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        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) => {

      const imgArray= result.map(element => element._id);
            console.log(imgArray);

```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    if (err) return console.log(err)
    res.send(imgArray)

  })
});
```

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.

```
app.get('/photo/:id', (req, res) => {
var filename = req.params.id;

db.collection('mycollection').findOne({'_id': ObjectId(filename) }, (

     if (err) return console.log(err)

    res.contentType('image/jpeg');
    res.send(result.image.buffer)


  })
})
```

Conclusion

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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.

Advertisement

Esther Vaati

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

FEED LIKE FOLLOW

Weekly email summary

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

Email Address

Update me weekly

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Advertisement

Native

12 Comments Tuts+ Hub 1 Login

Recommend 1 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Tyav Moses • 5 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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

4 ^ | ˅ 1 • Reply • Share ›

Christopher Ochsenreither • a month ago

I think it would make a lot more sense to make the filename function use the actual name of the file (file.originalname property) with the date prepended to retain the file extension instead appending the date to the fieldname, which will always be the same.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
var storage = multer.diskStorage({
destination: (req, file, cb) => {
cb(null, 'uploads');
},
filename: (req, file, cb) => {
console.log(file);
cb(null, `${Date.now()}-${file.originalname}`);
}
});
```

1 ^ | ˅ • Reply • Share ›

Bigeard • 2 months ago

Better code : https://github.com/Bigeard/...

^ | ˅ • Reply • Share ›

Thomas Dolhanty • 4 months ago

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

^ | ˅ • Reply • Share ›

Aravind Reddy • 5 months 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

^ | ˅ • Reply • Share ›

linda alibi • 5 months ago • edited

there is a duplication of 'app'
this works:
// call all the required packages
const express = require('express')

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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'));

Reply • Share ›

Pankaj Kr • 6 months ago

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

Reply • Share ›

teja b • 6 months ago

thank you, it was useful to me.

Reply • Share ›

Daniel Goje • 8 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 • 10 months ago

how to upload video using nodejs and mysql

Reply • Share ›

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Tom Larry • a year 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 • 10 months ago

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

1 • Reply • Share ›

Subscribe   Add Disqus to your site   Disqus' Privacy Policy   DISQUS

Advertisement

Create PDF in your applications with the Pdfcrowd HTML to PDF API   PDFCROWD

LOOKING FOR SOMETHING TO HELP KICK START YOUR NEXT PROJECT?

Envato Market has a range of items for sale to help get you started.

WordPress Plugins

From $5

PHP Scripts

From $5

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Unlimited Downloads
From $16.50/month

Get access to over one million creative assets on Envato Elements.

Over 9 Million Digital Assets

Everything you need for your next creative project.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Envato.com Our products Careers Sitemap

© 2019 Envato Pty Ltd. Trademarks and brands are the property of their respective owners.

Follow Envato Tuts+ Facebook Twitter Pinterest

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD