Multer

ex expressjs.com/en/resources/middleware/multer.html

Note: This page was generated from the multer README.
Multer is a node.js middleware for handling `multipart/form-data`, which is primarily used for uploading files. It is written on top of busboy for maximum efficiency.

NOTE: Multer will not process any form which is not multipart (`multipart/form-data`).

Translations

This README is also available in other languages:

Installation

```
$ npm install --save multer
```

Usage

Multer adds a `body` object and a `file` or `files` object to the `request` object. The `body` object contains the values of the text fields of the form, the `file` or `files` object contains the files uploaded via the form.

Basic usage example:

Don't forget the `enctype="multipart/form-data"` in your form.

```
<form action="/profile" method="post" enctype="multipart/form-data">
  <input type="file" name="avatar" />
</form>
```

```
const express = require('express')
const multer  = require('multer')
const upload = multer({ dest: 'uploads/' })

const app = express()

app.post('/profile', upload.single('avatar'), function (req, res, next) {
  // req.file is the `avatar` file
  // req.body will hold the text fields, if there were any
})

app.post('/photos/upload', upload.array('photos', 12), function (req, res, next) {
  // req.files is array of `photos` files
  // req.body will contain the text fields, if there were any
})

const cpUpload = upload.fields([{ name: 'avatar', maxCount: 1 }, { name: 'gallery', maxCount: 8 }])
app.post('/cool-profile', cpUpload, function (req, res, next) {
  // req.files is an object (String -> Array) where fieldname is the key, and the value is array of files
  //
  // e.g.
  //  req.files['avatar'][0] -> File
  //  req.files['gallery'] -> Array
  //
  // req.body will contain the text fields, if there were any
})
```

In case you need to handle a text-only multipart form, you should use the `.none()` method:

```
const express = require('express')
const app = express()
const multer  = require('multer')
const upload = multer()

app.post('/profile', upload.none(), function (req, res, next) {
  // req.body contains the text fields
})
```

Here's an example on how multer is used an HTML form. Take special note of the `enctype="multipart/form-data"` and `name="uploaded_file"` fields:

```
<form action="/stats" enctype="multipart/form-data" method="post">
  <div class="form-group">
    <input type="file" class="form-control-file" name="uploaded_file">
    <input type="text" class="form-control" placeholder="Number of speakers" name="nspeakers">
    <input type="submit" value="Get me the stats!" class="btn btn-default">
  </div>
</form>
```

Then in your javascript file you would add these lines to access both the file and the body. It is important that you use the `name` field value from the form in your upload function. This tells multer which field on the request it should look for the files in. If these fields aren't the same in the HTML form and on your server, your upload will fail:

```
const multer  = require('multer')
const upload = multer({ dest: './public/data/uploads/' })
app.post('/stats', upload.single('uploaded_file'), function (req, res) {
   // req.file is the name of your file in the form above, here 'uploaded_file'
   // req.body will hold the text fields, if there were any
   console.log(req.file, req.body)
});
```

API

File information

Each file contains the following information:

Key	Description	Note
`fieldname`	Field name specified in the form	
`originalname`	Name of the file on the user's computer	
`encoding`	Encoding type of the file	
`mimetype`	Mime type of the file	
`size`	Size of the file in bytes	
`destination`	The folder to which the file has been saved	`DiskStorage`
`filename`	The name of the file within the `destination`	`DiskStorage`
`path`	The full path to the uploaded file	`DiskStorage`
`buffer`	A `Buffer` of the entire file	`MemoryStorage`

`multer(opts)`

Multer accepts an options object, the most basic of which is the `dest` property, which tells Multer where to upload the files. In case you omit the options object, the files will be kept in memory and never written to disk.

By default, Multer will rename the files so as to avoid naming conflicts. The renaming function can be customized according to your needs.

The following are the options that can be passed to Multer.

Key	Description
`dest` or `storage`	Where to store the files
`fileFilter`	Function to control which files are accepted
`limits`	Limits of the uploaded data
`preservePath`	Keep the full path of files instead of just the base name

In an average web app, only `dest` might be required, and configured as shown in the following example.

```
const upload = multer({ dest: 'uploads/' })
```

If you want more control over your uploads, you'll want to use the `storage` option instead of `dest`. Multer ships with storage engines `DiskStorage` and `MemoryStorage`; More engines are available from third parties.

`.single(fieldname)`

Accept a single file with the name `fieldname`. The single file will be stored in `req.file`.

`.array(fieldname[, maxCount])`

Accept an array of files, all with the name `fieldname`. Optionally error out if more than `maxCount` files are uploaded. The array of files will be stored in `req.files`.

`.fields(fields)`

Accept a mix of files, specified by `fields`. An object with arrays of files will be stored in `req.files`.

`fields` should be an array of objects with `name` and optionally a `maxCount`. Example:

```
[
  { name: 'avatar', maxCount: 1 },
  { name: 'gallery', maxCount: 8 }
]
```

`.none()`

Accept only text fields. If any file upload is made, error with code "LIMIT_UNEXPECTED_FILE" will be issued.

`.any()`

Accepts all files that comes over the wire. An array of files will be stored in `req.files`.

WARNING: Make sure that you always handle the files that a user uploads. Never add multer as a global middleware since a malicious user could upload files to a route that you didn't anticipate. Only use this function on routes where you are handling the uploaded files.

`storage`

`DiskStorage`

The disk storage engine gives you full control on storing files to disk.

```
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, '/tmp/my-uploads')
  },
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9)
    cb(null, file.fieldname + '-' + uniqueSuffix)
  }
})

const upload = multer({ storage: storage })
```

There are two options available, `destination` and `filename`. They are both functions that determine where the file should be stored.

`destination` is used to determine within which folder the uploaded files should be stored. This can also be given as a `string` (e.g. `'/tmp/uploads'`). If no `destination` is given, the operating system's default directory for temporary files is used.

Note: You are responsible for creating the directory when providing `destination` as a function. When passing a string, multer will make sure that the directory is created for you.

`filename` is used to determine what the file should be named inside the folder. If no `filename` is given, each file will be given a random name that doesn't include any file extension.

Note: Multer will not append any file extension for you, your function should return a filename complete with an file extension.

Each function gets passed both the request (`req`) and some information about the file (`file`) to aid with the decision.

Note that `req.body` might not have been fully populated yet. It depends on the order that the client transmits fields and files to the server.

For understanding the calling convention used in the callback (needing to pass null as the first param), refer to Node.js error handling

`MemoryStorage`

The memory storage engine stores the files in memory as `Buffer` objects. It doesn't have any options.

```
const storage = multer.memoryStorage()
const upload = multer({ storage: storage })
```

When using memory storage, the file info will contain a field called `buffer` that contains the entire file.

WARNING: Uploading very large files, or relatively small files in large numbers very quickly, can cause your application to run out of memory when memory storage is used.

`limits`

An object specifying the size limits of the following optional properties. Multer passes this object into busboy directly, and the details of the properties can be found on busboy's page.

The following integer values are available:

Key	Description	Default
`fieldNameSize`	Max field name size	100 bytes
`fieldSize`	Max field value size (in bytes)	1MB
`fields`	Max number of non-file fields	Infinity
`fileSize`	For multipart forms, the max file size (in bytes)	Infinity
`files`	For multipart forms, the max number of file fields	Infinity
`parts`	For multipart forms, the max number of parts (fields + files)	Infinity

Key	Description	Default
`headerPairs`	For multipart forms, the max number of header key=>value pairs to parse	2000

Specifying the limits can help protect your site against denial of service (DoS) attacks.

`fileFilter`

Set this to a function to control which files should be uploaded and which should be skipped. The function should look like this:

```
function fileFilter (req, file, cb) {

  // The function should call `cb` with a boolean
  // to indicate if the file should be accepted

  // To reject this file pass `false`, like so:
  cb(null, false)

  // To accept the file pass `true`, like so:
  cb(null, true)

  // You can always pass an error if something goes wrong:
  cb(new Error('I don\'t have a clue!'))

}
```

Error handling

When encountering an error, Multer will delegate the error to Express. You can display a nice error page using the standard express way.

If you want to catch errors specifically from Multer, you can call the middleware function by yourself. Also, if you want to catch only the Multer errors, you can use the `MulterError` class that is attached to the `multer` object itself (e.g. `err instanceof multer.MulterError`).

```
const multer = require('multer')
const upload = multer().single('avatar')

app.post('/profile', function (req, res) {
  upload(req, res, function (err) {
    if (err instanceof multer.MulterError) {
      // A Multer error occurred when uploading.
    } else if (err) {
      // An unknown error occurred when uploading.
    }

    // Everything went fine.
  })
})
```

Custom storage engine

For information on how to build your own storage engine, see Multer Storage Engine.