CWEB280 -wk3

# Recommended Quiz: 20 minutes

Review : <https://handlebarsjs.com/guide/expressions.html>

Review: <https://handlebarsjs.com/examples/literal-segments.html>

# Multer Options - fileFilter and limits

Multer uses a set of options to configure the instance of the upload handler. We previously used the ‘dest’ option to specific the temporary upload directory

## limits

The limits option, as the name implies, set limits to the uploaded file they include: <https://www.npmjs.com/package/multer#limits>

| **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 |
| headerPairs | For multipart forms, the max number of header key=>value pairs to parse | 2000 |

Example: Use limits.fileSize to only allow files of 2Megabytes or less

limits: {  
 // limit file size to 2MB -> 2\*1024\*1024  
 fileSize: 2\*1024\*1024,  
},

## fileFilter

The fileFilter option is an anonymous function that checks file properties to filter which files are accepted for upload.

The function takes in 3 parameters

1. The request (req)
2. The file information (file)
3. A callback function that reports if the file meets the criteria or is not allowed

Learn more: <https://www.npmjs.com/package/multer#filefilter>

Example: Use fileFilter to only allow image files

fileFilter: (req, file, callback) => {  
 // check mime type starts with 'image/'  
 if (file.mimetype.startsWith('image/')) {  
 callback(null, true); // file mime type is allowed  
 } else {  
 // file not allowed, so return callback with new error message  
 return callback(new ***Error***('Only images are allowed'));  
 }  
},

# Use Multer to ONLY Accept Images Files of 2MB or Less

**\routes\examples.js – add the code in violet**

const ***router*** = ***express***.Router();  
  
// add packages that will handle the file uploads  
// https://www.npmjs.com/package/multer#usage  
const multer = require('multer');  
// IMPORTANT: ensure you created the destination folder before using it below  
const upload = multer({// multer settings  
 dest: 'public/uploads/',  
 fileFilter: (req, file, callback) => {  
 // check mime type starts with 'image/'  
 if (file.mimetype.startsWith('image/')) {  
 callback(null, true); // file mime type is allowed  
 } else {  
 // file not allowed, so return callback with new error message  
 return callback(new ***Error***('Only images are allowed'));  
 }  
 },  
 limits: {  
 // limit file size to 2MB -> 2\*1024\*1024  
 fileSize: 2\*1024\*1024,  
 },  
});

Restart the webserver for the changes to take effect - Navigate to <http://localhost:3000/examples/upload/>

Exercise Try to upload invalid files:

What happens when you try to upload a word document?

What happens when you try to upload an image file larger than 2048KB?

# File Handling on the Server

To perform file IO actions on the server we can use the fs module included with node.js

First we need to require the module in the router file

// require package for file io  
const ***fs*** = require('fs');

The several fs functions become available for us to use: Some examples are: <https://nodejs.dev/learn/the-nodejs-fs-module>

* fs.access(): check if the file exists and Node.js can access it with its permissions
* fs.appendFile(): append data to a file. If the file does not exist, it's created
* fs.close(): close a file descriptor
* fs.copyFile(): copies a file
* fs.createReadStream(): create a readable file stream
* fs.createWriteStream(): create a writable file stream
* fs.mkdir(): create a new folder
* fs.open(): set the file mode
* fs.readFile(): read the content of a file.
* fs.realpath(): resolve relative file path pointers (., ..) to the full path
* fs.rename(): rename a file or folder

# Move Uploaded files from temporary directory to public/images

In this example we are going to move the uploaded files from the temp destination to the public/image folder

We first need to make a function that will take multer file info object and a new path-

The function will use the filename and originalname to make a new file name in the images folder.

The advantage to this that the file will have a proper file extension and still have a unique/ not easily guessed file name.

We use the ***fs.rename*** function both rename and move the file.

**\routes\examples.js – add the code in violet**

*/\*\*  
 \** ***@param*** *{MulterFileInfo} tempFile  
 \** ***@param*** *{string} newPath  
 \*/*function moveFile(tempFile, newPath) {  
 // append the files filename and originalname to the path  
 newPath += tempFile.filename + '-' + tempFile.originalname;  
 ***fs***.rename(tempFile.path, newPath,  
 (err)=> {  
 // if there is a file system error just throw the error for now  
 if (err) throw err;  
 // OPTIONAL: inspect new path in terminal  
 ***console***.log('File moved to ' + newPath);  
 });  
}  
  
// GET content for path: http://localhost:3000/examples/upload/  
***router***.get('/upload', (req, res, next) => {  
 res.render('upload-files', {  
 title: 'GET - Upload Form Example',  
 });  
});

In the post handler we need to traverse (loop through) the uploaded files in req.files and call the moveFile function

**\routes\examples.js – add the code in violet**

// POST submit form data to path : http://localhost:3000/examples/upload/  
// BEST PRACTICE: specify the only fields the app app will accept  
// https://www.npmjs.com/package/multer#usage  
***router***.post('/upload', upload.fields([  
 {name: 'file1', maxCount: 1},  
 {name: 'file2', maxCount: 1},  
]),  
(req, res, next) => {  
 // output file array info to console to see what is available  
 ***console***.log('uploaded files:\n');  
 ***console***.log(req.files);  
  
 // loop through the files object and then the nested arrays  
 for ( const [field, fileArray] of ***Object***.entries(req.files)) {  
 // OPRIONAL: inspect the objects in the terminal  
 ***console***.log('Field: '+ field + '\n');  
 ***console***.log(fileArray);  
 // loop through the files in the array  
 for (const tempFile of fileArray) {  
 // OPTIONAL: again inspect the files in the terminal  
 ***console***.log('Temp File:\n');  
 ***console***.log(tempFile);  
 // call the move file function to move the file to public/imafges folder  
 moveFile(tempFile, \_\_dirname + '/../public/images/');  
 }  
 }  
  
 // declare variables to store the uploaded file information  
 let file1;  
 let file2;  
 // let pictures; //part of exercise 6 solution  
 // check to see if the corresponding files were uploaded  
 // otherwise use new object  
 file1 = req.files['file1'] ? req.files['file1'][0] : {originalname: 'not uploaded'};  
 file2 = req.files['file2'] ? req.files['file2'][0] : {originalname: 'not uploaded'};  
  
  
  
 res.render('upload-files', {  
 title: 'POST - Upload Form Example',  
 isSubmitted: true, // check to see if the file title is filled in />  
 file1Title: req.body.file1Title,  
 file2Title: req.body.file2Title,  
 file1Description: req.body.desc1,  
 file2Description: req.body.desc2,  
 file1Info: file1,  
 file2Info: file2,  
 });  
});

Exercise Check files are being moved

What happens when you upload a valid image file? Where does it end up? How long does it take to move the file?