Uploading Files to MongoDB - Traversy Media

Sources:

- Video Traversy Media: https://www.youtube.com/watch?v=3f5Q9wDePzY
- Video, the other guy: https://www.youtube.com/watch?v=OvbRLY1QRIk
- Multer-gridfs-storage: https://github.com/aheckmann/gridfs-stream
- React replacement of ejs: https://www.youtube.com/watch?v=KoWTJ5XiYm4
- Upload to Folder - client side is identical here: https://docs.google.com/document/d/1Xf9NyEh5gIZ6-cJ_8NEjxJFR6c9hjXiDXkq6AC65yjE/edit#
- **MY VIDEO ON THIS**: https://youtu.be/4WT5nvfXcbs
- **My REPO**: https://github.com/dmitrisanzharov/react-file-uploads

Rules / Reminders:

Core to display images:	• If I can display Image in BROWSER window as a SINGLE IMAGE • I can then use to reference that URL that displays that image, and it will appear • You CAN use React instead of .ejs, see REACT title
enctype="multipart/form-data"	This is needed when every I am using INPUT TYPE FILE, otherwise doesn't work. I can do it via AXIOS: `const req = await axios.post('/upload', formData, { headers: { 'Content-type': 'multipart/form-data' } });`

	OR via a form: `<form action="/upload" method="POST" enctype="multipart/form-data">`
Uploads.chunks uploads.files	Under this setup GridFS will automatically, create 2 databases chunks and files. This is needed, because it will break down LARGE files into smaller 16 mb chunks. The main one is uploads.files, that I need
`res.redirect('/');`	In express server, can be used to redirect client BACK to a certain page
`// @route GET /files` `// @desc Display all files in JSON`	Very good idea to showcase DESCRIPTION of what each request does
FIND ALL FILES: `gfs.files.find().toArray((err, files) => {  // this is the code to return ALL files` `if (!files || files.length === 0) { // if there are no files, then returns 404` `return res.status(404).json({` `err: 'No files exist'` `})` `}` `return res.json(files); // this will send file array` `});`	When I am working with GridFS, it can be used as Mongoose Schema, it effectively replaces the Schema, so it works the same, thus I can use .find(); .delete() on it When dealing with MANY files, you need to convert to an ARRAY, otherwise does NOT work
Find SINGLE FILE: Here is full code:	Here we are using filename in req.params.filename

```
app.get('/files/:filename', (req, res) => {

    gfs.files.findOne({ filename: req.params.filename }, (err, file) => {
        if (!file || file.length === 0) { // if there are no files, then returns 404
            return res.status(404).json({
                err: 'No files exist'
            })
        }

        return res.json(file); // this will send file array
    })

});
```

So the URL should look like this: http://localhost:5000/files/0ccd7399006ab65ddb6f79c4dc350b1e.pdf

NOTE the syntax in .findOne, we DO NOT need to convert into Array in here, instead, we are just passing Object which indicates what Property from database to use for Search AND then (err, file =>

Note2: its best to look for FILENAME, because it is a string, since other things, like ID are NOT strings, so you need to change them a bit for it to work, this is why file name is the easiest.

VIEW FILES / IMAGES IN BROWSER:

To view a file on Server side, I can use this code:

```
            const readstream = gridfsBucket.openDownloadStreamByName(file.filename); // if I want to use FILENAME I have to use: openDownloadStreamByName, for everything else I need to use: openDownloadStream(file._id)
            readstream.pipe(res);
```

I also need to initialise it, here:

`` let gfs, gridfsBucket; conn.once('open', () => { // initialise our Stream gridfsBucket = new mongoose.mongo.GridFSBucket(conn.db, { bucketName: 'uploads' }); gfs = Grid(conn.db, mongoose.mongo); // Grid here means, MODULE Grid, see above gfs.collection('uploads'); //! I am assuming that this is the name of the database, there could be an error here, if doesn't work }); ``` See gridBuckets	

Dependencies:

Full:

npm i express ejs body-parser mongoose multer multer-gridfs-storage gridfs-stream method-override

Npm i -D nodemon

express	server
ejs	Html display thing
body-parser	Does something with the form

mongoose	For mongoDB
multer	For file uploads in express
multer-gridfs-storage	This is for storing files larger than 16mb in MongoDB
gridfs-stream	Not sure what this does
method-override	Not sure what this does
nodemon	As Dev Dependency
cors	For cors errors

Step by step:

1	Initial the project Install All dependencies and Developer dependencies	Npm init -y npm i express ejs body-parser mongoose multer multer-gridfs-storage gridfs-stream method-override cors Npm i -D nodemon
2	In package.json set “main” to app.js	<pre>"description": "", "main": "app.js",</pre>
3	Package.json add the scripts to RUN the app	<pre>"scripts": { "start": "node app.js", "dev": "nodemon app.js" },</pre>
4	In ROOT create APP.js file Add the Express code AND view code, to display the HTML template	app.js <pre>const express = require('express'); const app = express();</pre>

		`` app.set('view engine', 'ejs'); // this sets up the VIEW, i.e. the ejs HTML display thing app.get('/', (req, res) => { res.render('index') }); // this is part of the EJS thing const port = 5000; app.listen(port, () => console.log(`Server started on port ${port}`)) ```
5	In ROOT create the VIEWS folder (this is so that EJS allows EJS, to look automatically in VIEWS folder and render the file that I asked it to… i.e. INDEX	Inside VIEW folder create a file called: Index.ejs Inside the INDEX.EJS add the Basic HTML template: ``` <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width,initial-scale=1"> <meta http-equiv="X-UA-Compatible" content="IE=edge"> <meta name="Dmitri" content="Dmitri Stuff"> <title>My New App</title> </head> <body> Mongo File Uploads ```

		`` </body> </html>``` **Note:** **This code here is essentially REACT / CLIENT:** ```app.set('view engine', 'ejs'); // this sets up the VIEW, i.e. the ejs HTML display thing app.get('/', (req, res) => { res.render('index') }); // this is part of the EJS thing``` **Also folder VIEWS and files inside views.**
6	Start the server	Npm run dev I should be able to see the HTML template on the port 5000 now
7	Add the form code into the HTML	``` <form action="/upload" method="POST" enctype="multipart/form-data"> <input type="file" name="file" id="file"> <input type="submit" value="Submit button"> </form>``` Remember the ENCTYPE for the FORM, cause I am using the TYPE FILE

8	In app.js bring ALL the required modules	`// ALL OTHER DEPENDENCIES` `const path = require('path');` `const crypto = require('crypto'); // this is to generate file names, it is CORE node.js module, so no need for it` `const bodyParser = require('body-parser');` `const mongoose = require('mongoose'); // note we do NOT need to create schemas in here, because GRIDFS takes care of that for us` `const multer = require('multer');` `const { GridFsStorage } = require('multer-gridfs-storage');` `const Grid = require('gridfs-stream');` `const methodOverride = require('method-override');` `const cors = require('cors');`
9	Add the middleware	`// Middleware` `app.use(bodyParser.json());` `app.use(methodOverride('_method'));` `app.use(cors());`
10	Connect to MongoDB, short way	`// MONGO STUFF` `const mongoURI = 'mongodb+srv://testUser1:BtNuk3jO9myBOupf@rest.qks7o.mongodb.net/practiceUploads?retryWrites=true&w=majority';` `const conn = mongoose.createConnection(mongoURI); // short way to connect to MongoDB`
11	Initialize our GFS stream and buckets	`// Init GFS` `let gfs, gridfsBucket;`

		`` conn.once('open', () => { // initialise our Stream gridfsBucket = new mongoose.mongo.GridFSBucket(conn.db, { bucketName: 'uploads' }); gfs = Grid(conn.db, mongoose.mongo); // Grid here means, MODULE Grid, see above gfs.collection('uploads'); // note here GFS will automatically break down UPLOADS into 2 uploads.chunks and uploads.files ... MOST OF THE TIME: I will be using UPLOADS in code, but if fails try: UPLOADS.FILES OR GFS.FILES / GFS }); ```
12	Create the storage engine in the app.js	``` // Create storage engine const storage = new GridFsStorage({ url: mongoURI, file: (req, file) => { return new Promise((resolve, reject) => { crypto.randomBytes(16, (err, buf) => { if (err) { return reject(err); } const filename = buf.toString('hex') + path.extname(file.originalname); const fileInfo = { filename: filename, bucketName: 'uploads' // the BUCKET NAME, should match the gfs.collection(NAME)... i.e. they should be ```

		the same ``` }; resolve(fileInfo); }); }); } }); const upload = multer({ storage }); // NOTE: this is the variable that we will use for the MIDDLEWARE, in the POST requests ```
13	Create a post request	I can do it, under the APP.LISTEN, it will still work ``` // POST REQUESTS app.post('/upload', upload.single('file'), (req, res) => { // node the UPLOAD is the MULTER variable from line 55; // in line 74, upload.single(NAME) -> here name comes from HTML INPUT FILE, i.e. <input type="file" name="file" id="file"> console.log('upload successful'); res.json({ file: req.file }) }) ``` Ones, the upload is successful, this will return me to /UPLOADS page Here, I will see the JSON of the file that I have uploaded.
14	Add the redirect back to HOME PAGE when file is uploaded	``` res.redirect('/'); ```
15	Add the GET request to get all the files	``` // @route GET /files // @desc Display all files in JSON app.get('/files', (req, res) => { gfs.files.find().toArray((err, files) => { // this is the code to return ALL files if(!files || files.length === 0){ // ```

		`if there are no files, then returns 404` `return res.status(404).json({` `err: 'No files exist'` `})` `}` `return res.json(files); // this will send file array` `});` `});`
16	Add to find one code	`// @route GET /files/:filename` `// @desc  Display single file` `app.get('/files/:filename', (req, res) => {` `gfs.files.findOne({ filename: req.params.filename }, (err, file) => {` `if (!file || file.length === 0) { // if there are no files, then returns 404` `return res.status(404).json({` `err: 'No files exist'` `})` `}` `return res.json(file); // this will send file array` `})` `});`
16a	Server code to display images:	`// @route GET image/:filename` `// @desc  Display image` `app.get('/image/:filename', (req, res) => {` `gfs.files.findOne({ filename: req.params.filename }, (err, file) => {` `// Check if file` `if (!file || file.length === 0) {`

		`` return res.status(404).json({ err: 'No file exists' }); } // Check if image if (file.contentType === 'image/jpeg' || file.contentType === 'image/png') { // Read output to browser const readstream = gridfsBucket.openDownloadStreamByName(file.filename); // if I want to use FILENAME I have to use: openDownloadStreamByName, for everything else I need to use: openDownloadStream(file._id) readstream.pipe(res); } else { res.status(404).json({ err: 'Not an image' }); } }); }); ```
17	Client code to Display images - like this:	Single image URL: http://localhost:5000/image/a6dde456f8330a3832a838e8097d65d8.png In HTML: ``` <img src="http://localhost:5000/image/a6dde456f8330a3832a838e8097d65d8.png" alt="blah"> ```
18	**Delete file** Can be _ID or filename	``` // @route DELETE /files/:filename // @desc Delete file app.delete('/files/:filename', async (req, res) => { ```

		`` const file = await gfs.files.findOne({ filename: req.params.filename }); const gsfb = new mongoose.mongo.GridFSBucket(conn.db, { bucketName: 'uploads' }); gsfb.delete(file._id, function (err, gridStore) { if (err) { res.status(404).send('no file found') } res.status(200).send('deleted successfully') }); }); ```

Full Server code:

```
const express = require('express');
const app = express();
const cors = require('cors');

// additional modules
const path = require('path');
const crypto = require('crypto'); // this is to generate file names, it is CORE node.js module, so no need for it
const bodyParser = require('body-parser');
const mongoose = require('mongoose'); // note we do NOT need to create schemas in here, because GRIDFS takes care of that for us
const multer = require('multer');
const { GridFsStorage } = require('multer-gridfs-storage');
const Grid = require('gridfs-stream');
const methodOverride = require('method-override');
```

```
// Middleware
app.use(bodyParser.json());
app.use(methodOverride('_method'));
app.use(cors());


// MONGO STUFF
const mongoURI =
'mongodb+srv://testUser1:BtNuk3jO9myBOupf@rest.qks7o.mongodb.net/grid-fs-uploads?
retryWrites=true&w=majority';
const conn = mongoose.createConnection(mongoURI); // short way to connect to
MongoDB


// Init GFS
let gfs, gridfsBucket;

conn.once('open', () => {
    // initialise our Stream
    gridfsBucket = new mongoose.mongo.GridFSBucket(conn.db, {
        bucketName: 'uploads'
    });
    gfs = Grid(conn.db, mongoose.mongo); // Grid here means,  MODULE Grid, see
above
    gfs.collection('uploads'); // note here GFS will automatically break down
UPLOADS into 2  uploads.chunks and uploads.files ...  MOST OF THE TIME: I will be
using  UPLOADS in code, but if fails try:  UPLOADS.FILES OR GFS.FILES / GFS
});



// Create storage engine
const storage = new GridFsStorage({
    url: mongoURI,
    file: (req, file) => {
        return new Promise((resolve, reject) => {
            crypto.randomBytes(16, (err, buf) => {
                if (err) {
                    return reject(err);
```

```
                }
                const filename = buf.toString('hex') +
path.extname(file.originalname);
                const fileInfo = {
                    filename: filename,
                    bucketName: 'uploads' // the BUCKET NAME, should match the
gfs.collection(NAME)... i.e. they should be the same
                };
                resolve(fileInfo);
            });
        });
    }
});
const upload = multer({ storage }); // NOTE: this is the variable that we will
use for the MIDDLEWARE, in the POST requests




// VIEW STUFF
app.set('view engine', 'ejs'); // this sets up the VIEW, i.e. the ejs  HTML
display thing
app.get('/', (req, res) => {
    res.render('index')
}); // this is part of the EJS thing


const port = 5000;
app.listen(port, () => console.log(`Server started on port ${port}`));


// POST REQUESTS
app.post('/upload', upload.single('file'), (req, res) => { // node the UPLOAD is
the MULTER variable from line 55;
    // in line 74,   upload.single(NAME) -> here name comes from HTML INPUT FILE,
i.e.   <input type="file" name="file" id="file">
    console.log('upload successful');
```

```
    res.json({ file: req.file })
    // res.redirect('/');

});


// @route GET /files
// @desc  Display all files in JSON
app.get('/files', (req, res) => {
    gfs.files.find().toArray((err, files) => {  // this is the code to return ALL
files
        if (!files || files.length === 0) { // if there are no files, then
returns 404
            return res.status(404).json({
                err: 'No files exist'
            })
        }

        return res.json(files); // this will send file array
    });
});


// @route GET /files/:filename
// @desc  Display single file object
app.get('/files/:filename', (req, res) => {

    gfs.files.findOne({ filename: req.params.filename }, (err, file) => {
        if (!file || file.length === 0) { // if there are no files, then returns
404
            return res.status(404).json({
                err: 'No files exist'
            })
        }

        return res.json(file); // this will send file array
    })

});
```

```
// @route GET image/:filename
// @desc  Display image
app.get('/image/:filename', (req, res) => {
    gfs.files.findOne({ filename: req.params.filename }, (err, file) => {
        // Check if file
        if (!file || file.length === 0) {
            return res.status(404).json({
                err: 'No file exists'
            });
        }

        // Check if image
        if (file.contentType === 'image/jpeg' || file.contentType === 
'image/png') {
            // Read output to browser
            const readstream = 
gridfsBucket.openDownloadStreamByName(file.filename); // if I want to use 
FILENAME I have to use: openDownloadStreamByName, for everything else I need to 
use: openDownloadStream(file._id)
            readstream.pipe(res);
        } else {
            res.status(404).json({
                err: 'Not an image'
            });
        }
    });
});


// @route DELETE /files/:id
// @desc  Delete file
app.delete('/files/:id', (req, res) => { // to remove it can be _ID  OR FILENAME
    gfs.remove({ _id: req.params.id, root: 'uploads' }, (err, gridStore) => { // 
uploads is the NAME of the collection
        if (err) {
            return res.status(404).json({ err: err });
        }
```

```
        res.redirect('/');
    });
});
```

Full ejs code:

```
<!DOCTYPE html>

<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="Dmitri" content="Dmitri Stuff">
    <title>My New App</title>
</head>

<body>

    <h1>Mongo File Uploads</h1>

    <hr>

    <!-- note when I am using a FILE TYPE input field, I need to add    enctype =
MULTIPART/FORM-DATA, otherwise does NOT work -->
    <form action="/upload" method="POST" enctype="multipart/form-data">
        <input type="file" name="file" id="file">
        <br>
        <br>
        <input type="submit" value="Submit button">
    </form>
    <hr>
```

```
    <img src="http://localhost:5000/image/a6dde456f8330a3832a838e8097d65d8.png"
alt="blah">



</body>

</html>
```

React instead of .ejs

Source: https://www.youtube.com/watch?v=KoWTJ5XiYm4

Pure HTML code:

```
const FileUpload = () => {
    return (
        <div>
            <form action="http://localhost:5000/upload" method='POST'
encType="multipart/form-data">
                <input type="file" name='file' />
                <input type="submit" value='submit' />
            </form>
        </div>
    )
}
```

React INPUT=NAME replacer

Core:

- To replicate input name='name' , you need to use formData.append('name', 'nameToMatchMulter'
- It must be BEFORE .append('file', file), otherwise might bug

Here is full code:

```
import React, { useState, useEffect } from 'react'
import axios from 'axios';

const FileUpload = () => {

    const [file, setFile] = useState(null);



    const onSubmit = async (e) => {

        e.preventDefault();
        const formData = new FormData();

        formData.append('name', 'file'); // this must match multer; ALSO, NAME
MUST COME BEFORE THE FILE, otherwise ill fail
        formData.append("file", file);

        await axios.post("http://localhost:5000/upload", formData, {
            headers: {
                "Content-type": "multipart/form-data",
            },
        });

    }



    return (
        <div>
            <form onSubmit={onSubmit}>
                <input type="file" onChange={e => setFile(e.target.files[0])} />
```

```
                <input type="submit" value='submit' />
            </form>

        </div>
    )
}

export default FileUpload;
```

Multi Upload:

Identical to Single ONLY MAP through FILES, SEE my other solution here:
https://docs.google.com/document/d/1Xf9NyEh5gIZ6-cJ_8NEjxJFR6c9hjXiDXkq6AC65yjE/edit#

Here is full code:

```
import React, { useState, useEffect } from 'react'
import axios from 'axios';

const MultiUpload = () => {

    const [file, setFile] = useState(null);


    const handleChange = e => {
        setFile(Array.prototype.slice.call(e.target.files));
    }


    const onSubmit = async (e) => {

        e.preventDefault();

        file.map(async el => {
            const formData = new FormData();
```

```
            formData.append('name', 'file'); // this must match multer; ALSO,
NAME MUST COME BEFORE THE FILE, otherwise ill fail
            formData.append("file", el);

            await axios.post("http://localhost:5000/upload", formData, {
                headers: {
                    "Content-type": "multipart/form-data",
                },
            });
        })

    }


    return (
        <div>
            <form onSubmit={onSubmit}>
                <input type="file" onChange={handleChange} multiple={true} />
                <input type="submit" value='submit' />
            </form>

        </div>
    )
}

export default MultiUpload;
```

Display Single and Many

- Key in here is to create a single GET request on server side that will display image as a URL in the browser, e.g.: http://localhost:5000/image/a6dde456f8330a3832a838e8097d65d8.png

Here is BASIC CODE FOR IT:

```
// @route GET image/:filename
// @desc  Display image
app.get('/image/:filename', (req, res) => {
    gfs.files.findOne({ filename: req.params.filename }, (err, file) => {

        const readstream = gridfsBucket.openDownloadStreamByName(file.filename);
// if I want to use FILENAME I have to use: openDownloadStreamByName, for
everything else I need to use: openDownloadStream(file._id)
        readstream.pipe(res);
    });
});
```

- To display we need FILENAME, so we get that from requesting ALL data, and filtering by file name
- We then map this data and display it in <img>, like this:

```
            {data && data.map(el => {
                return <img src={`http://localhost:5000/image/${el}`}
alt="" key={el} />
            })}
```

  Its exactly the same is displaying EXTERNAL URLS, cause when on google we use ‘get image url’ we querying external URL, so here we are queuing SERVER THAT WE CREATED, its the same thing basically.

Here is full code:

```
import React, { useState, useEffect, useMemo } from 'react';
import axios from 'axios';

export default function DisplayNDelete() {

    const [data, setData] = useState(null);

    const getData = async () => {
        const res = await axios.get('http://localhost:5000/files');
        const finalData = res.data.map(el => el.filename);
        console.log("finalData: ", finalData);
```

```
        setData(finalData);
    }


    useEffect(() => {
        getData();
    }, [])




    return (
        <div>
            <h1>Single image</h1>
            <img
src={'http://localhost:5000/image/a6dde456f8330a3832a838e8097d65d8.png'} alt=""
/>
            <hr />

            <h2>Many</h2>
            {data && data.map(el => {
                return <img src={`http://localhost:5000/image/${el}`} alt=""
key={el} />
            })}
        </div>
    )
}
```

Deleting the image:

- SOURCE: https://stackoverflow.com/questions/71989447/unhandledpromiserejectionwarning-typeerror-cannot-read-property-unlink-of-un/74792137#74792137
- This one is a copy and paste code:

- So were are creating an API for deleting the code and then MAPPing through the images and passing filename for deletion (see component code)
- REMINDER to re-render the setData, by calling getData function (see component code below)

```
// @route DELETE /files/:filename
// @desc  Delete file
app.delete('/files/:filename', async (req, res) => {

    const file = await gfs.files.findOne({ filename: req.params.filename });
    const gsfb = new mongoose.mongo.GridFSBucket(conn.db, { bucketName: 'uploads'
});
    gsfb.delete(file._id, function (err, gridStore) {
        if (err) {
            res.status(404).send('no file found')
        }
        res.status(200).send('deleted successfully')
    });
});
```

Function to do it:

```
    const delImage = async (fileName) => {
        await axios.delete(`http://localhost:5000/files/${fileName}`);
        console.log('fileDeleted');
        getData();
    }
```

Here is full code:

```
import React, { useState, useEffect, useMemo } from 'react';
import axios from 'axios';

export default function DisplayNDelete() {

    const [data, setData] = useState(null);
```

```
    const getData = async () => {
        const res = await axios.get('http://localhost:5000/files');
        const finalData = res.data.map(el => el.filename);
        console.log("finalData: ", finalData.length);

        setData(finalData);
    }

    const delImage = async (fileName) => {
        await axios.delete(`http://localhost:5000/files/${fileName}`);
        console.log('fileDeleted');
        getData();
    }




    useEffect(() => {
        getData();
    }, [])




    return (
        <div>
            <h1>Single image</h1>
            <img
src={'http://localhost:5000/image/6f2888c360c0325427fc148fda2cf870.png'} alt=""
/>
            <hr />

            <h2>Many</h2>
            {data && data.map(el => {
                return (
                    <div key={el}>
                        <hr />
                        <img src={`http://localhost:5000/image/${el}`} alt="" />
```

```
                        <br />
                        <button onClick={() => delImage(el)}>delete</button>
                    </div>
                )
            })}
        </div>
    )
}
```

FINAL CODE:

Server code, app.js

```
const express = require('express');
const app = express();
const cors = require('cors');

// additional modules
const path = require('path');
const crypto = require('crypto'); // this is to generate file names, it is CORE node.js module, so no need for it
const bodyParser = require('body-parser');
const mongoose = require('mongoose'); // note we do NOT need to create schemas in here, because GRIDFS takes care of that for us
const multer = require('multer');
const { GridFsStorage } = require('multer-gridfs-storage');
const Grid = require('gridfs-stream');
const methodOverride = require('method-override');
```

```
// Middleware
app.use(bodyParser.json());
app.use(methodOverride('_method'));
app.use(cors());


// MONGO STUFF
const mongoURI =
'mongodb+srv://testUser1:BtNuk3jO9myBOupf@rest.qks7o.mongodb.net/grid-fs-uploads?
retryWrites=true&w=majority';
const conn = mongoose.createConnection(mongoURI); // short way to connect to
MongoDB


// Init GFS
let gfs, gridfsBucket;

conn.once('open', () => {
    // initialise our Stream
    gridfsBucket = new mongoose.mongo.GridFSBucket(conn.db, {
        bucketName: 'uploads'
    });
    gfs = Grid(conn.db, mongoose.mongo); // Grid here means,  MODULE Grid, see
above
    gfs.collection('uploads'); // note here GFS will automatically break down
UPLOADS into 2  uploads.chunks and uploads.files ...  MOST OF THE TIME: I will be
using  UPLOADS in code, but if fails try:  UPLOADS.FILES OR GFS.FILES / GFS
});



// Create storage engine
const storage = new GridFsStorage({
    url: mongoURI,
    file: (req, file) => {
        return new Promise((resolve, reject) => {
            crypto.randomBytes(16, (err, buf) => {
                if (err) {
                    return reject(err);
```

```
                }
                const filename = buf.toString('hex') +
path.extname(file.originalname);
                const fileInfo = {
                    filename: filename,
                    bucketName: 'uploads' // the BUCKET NAME, should match the
gfs.collection(NAME)... i.e. they should be the same
                };
                resolve(fileInfo);
            });
        });
    }
});
const upload = multer({ storage }); // NOTE: this is the variable that we will
use for the MIDDLEWARE, in the POST requests




// VIEW STUFF
app.set('view engine', 'ejs'); // this sets up the VIEW, i.e. the ejs  HTML
display thing
app.get('/', (req, res) => {
    res.render('index')
}); // this is part of the EJS thing


const port = 5000;
app.listen(port, () => console.log(`Server started on port ${port}`));


// POST REQUESTS
app.post('/upload', upload.single('file'), (req, res) => { // node the UPLOAD is
the MULTER variable from line 55;
    // in line 74,   upload.single(NAME) -> here name comes from HTML INPUT FILE,
i.e.   <input type="file" name="file" id="file">
    console.log('upload successful');
```

```
    res.json({ file: req.file })
    // res.redirect('/');

});


// @route GET /files
// @desc  Display all files in JSON
app.get('/files', (req, res) => {
    gfs.files.find().toArray((err, files) => {  // this is the code to return ALL
files
        if (!files || files.length === 0) { // if there are no files, then
returns 404
            return res.status(404).json({
                err: 'No files exist'
            })
        }

        return res.json(files); // this will send file array
    });
});


// @route GET /files/:filename
// @desc  Display single file object
app.get('/files/:filename', (req, res) => {

    gfs.files.findOne({ filename: req.params.filename }, (err, file) => {
        if (!file || file.length === 0) { // if there are no files, then returns
404
            return res.status(404).json({
                err: 'No files exist'
            })
        }

        return res.json(file); // this will send file array
    })

});
```

```
// @route GET image/:filename
// @desc  Display image
app.get('/image/:filename', (req, res) => {
    gfs.files.findOne({ filename: req.params.filename }, (err, file) => {
        // Check if file
        if (!file || file.length === 0) {
            return res.status(404).json({
                err: 'No file exists'
            });
        }

        // Check if image
        if (file.contentType === 'image/jpeg' || file.contentType ===
'image/png') {
            // Read output to browser
            const readstream =
gridfsBucket.openDownloadStreamByName(file.filename); // if I want to use
FILENAME I have to use: openDownloadStreamByName, for everything else I need to
use: openDownloadStream(file._id)
            readstream.pipe(res);
        } else {
            res.status(404).json({
                err: 'Not an image'
            });
        }
    });
});


// @route DELETE /files/:filename
// @desc  Delete file
app.delete('/files/:filename', async (req, res) => {

    const file = await gfs.files.findOne({ filename: req.params.filename });
    const gsfb = new mongoose.mongo.GridFSBucket(conn.db, { bucketName: 'uploads'
});
    gsfb.delete(file._id, function (err, gridStore) {
```

```
        if (err) {
            res.status(404).send('no file found')
        }
        res.status(200).send('deleted successfully')
    });
});
```

Single File Upload Component:

```
import React, { useState, useEffect } from 'react'
import axios from 'axios';

const FileUpload = () => {

    const [file, setFile] = useState(null);


    const onSubmit = async (e) => {

        e.preventDefault();
        const formData = new FormData();

        formData.append('name', 'file'); // this must match multer; ALSO, NAME
MUST COME BEFORE THE FILE, otherwise ill fail
        formData.append("file", file);

        await axios.post("http://localhost:5000/upload", formData, {
            headers: {
                "Content-type": "multipart/form-data",
            },
        });

    }


    return (
```

```
        <div>
            <form onSubmit={onSubmit}>
                <input type="file" onChange={e => setFile(e.target.files[0])} />
                <input type="submit" value='submit' />
            </form>

        </div>
    )
}

export default FileUpload;
```

Many files upload component:

```
import React, { useState, useEffect } from 'react'
import axios from 'axios';

const MultiUpload = () => {

    const [file, setFile] = useState(null);


    const handleChange = e => {
        setFile(Array.prototype.slice.call(e.target.files));
    }


    const onSubmit = async (e) => {

        e.preventDefault();

        file.map(async el => {
            const formData = new FormData();
```

```
            formData.append('name', 'file'); // this must match multer; ALSO,
NAME MUST COME BEFORE THE FILE, otherwise ill fail
            formData.append("file", el);

            await axios.post("http://localhost:5000/upload", formData, {
                headers: {
                    "Content-type": "multipart/form-data",
                },
            });
        })

    }


    return (
        <div>
            <form onSubmit={onSubmit}>
                <input type="file" onChange={handleChange} multiple={true} />
                <input type="submit" value='submit' />
            </form>

        </div>
    )
}

export default MultiUpload;
```

Display and DElete Component:

```
import React, { useState, useEffect, useMemo } from 'react';
import axios from 'axios';

export default function DisplayNDelete() {

    const [data, setData] = useState(null);
```

```
    const getData = async () => {
        const res = await axios.get('http://localhost:5000/files');
        const finalData = res.data.map(el => el.filename);
        console.log("finalData: ", finalData.length);

        setData(finalData);
    }

    const delImage = async (fileName) => {
        await axios.delete(`http://localhost:5000/files/${fileName}`);
        console.log('fileDeleted');
        getData();
    }




    useEffect(() => {
        getData();
    }, [])




    return (
        <div>
            <h1>Single image</h1>
            <img
src={'http://localhost:5000/image/6f2888c360c0325427fc148fda2cf870.png'} alt=""
/>
            <hr />

            <h2>Many</h2>
            {data && data.map(el => {
                return (
                    <div key={el}>
                        <hr />
                        <img src={`http://localhost:5000/image/${el}`} alt="" />
```

```
                        <br />
                        <button onClick={() => delImage(el)}>delete</button>
                    </div>
                )
            })}
        </div>
    )
}
```

FINAL CODE - SIMPLE:

Server:

```
const express = require('express');
const app = express();


// ALL OTHER DEPENDENCIES
const path = require('path');
const crypto = require('crypto'); // this is to generate file names, it is CORE node.js module, so no need for it
const bodyParser = require('body-parser');
const mongoose = require('mongoose'); // note we do NOT need to create schemas in here, because GRIDFS takes care of that for us
const multer = require('multer');
const { GridFsStorage } = require('multer-gridfs-storage');
const Grid = require('gridfs-stream');
const methodOverride = require('method-override');
const cors = require('cors');

// Middleware
app.use(bodyParser.json());
```

```
app.use(methodOverride('_method'));
app.use(cors());


// MONGO STUFF
const mongoURI =
'mongodb+srv://testUser1:BtNuk3jO9myBOupf@rest.qks7o.mongodb.net/youtubeDemo?retr
yWrites=true&w=majority';
const conn = mongoose.createConnection(mongoURI); // short way to connect to
MongoDB

// Init GFS, gridFSBuckets
let gfs, gridfsBucket;

conn.once('open', () => {
    gridfsBucket = new mongoose.mongo.GridFSBucket(conn.db, {
        bucketName: 'uploads'
    });
    gfs = Grid(conn.db, mongoose.mongo);
    gfs.collection('uploads');
});


// Create storage engine
const storage = new GridFsStorage({
    url: mongoURI,
    file: (req, file) => {
        return new Promise((resolve, reject) => {
            crypto.randomBytes(16, (err, buf) => {
                if (err) {
                    return reject(err);
                }
                const filename = buf.toString('hex') +
path.extname(file.originalname);
                const fileInfo = {
                    filename: filename,
                    bucketName: 'uploads'
                };
                resolve(fileInfo);
```

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




const port = 5000;

app.listen(port, () => console.log(`Server started on port ${port}`))


// @route GET /test
// @desc See if server works
app.get('/test', (req, res) => {
    res.send('works')
})


// @route POST /upload
// @decs Upload single file
app.post('/upload', upload.single('file'), (req, res) => {
    console.log('file was uploaded');
    res.send('all good');
});


// @route GET /all-files
// @decs display all files
app.get('/all-files', (req, res) => {
    gfs.files.find().toArray((err, files) => {

        return res.json(files);
    });

});
```

```
// @route GET image/:filename
// @desc  Display image
app.get('/display/:filename', (req, res) => {
    gfs.files.findOne({ filename: req.params.filename }, (err, file) => {

        const readstream = gridfsBucket.openDownloadStreamByName(file.filename);
        readstream.pipe(res);
    });

});

// @route DELETE /files/:filename
// @desc  Delete file
app.delete('/delete/:filename', async (req, res) => {

    const file = await gfs.files.findOne({ filename: req.params.filename });
    const gsfb = new mongoose.mongo.GridFSBucket(conn.db, { bucketName: 'uploads'
});
    gsfb.delete(file._id, function (err, gridStore) {
        if (err) {
            res.status(404).send('no file found')
        }
        res.status(200).send('deleted successfully')
    });
});
```

Upload many:

```
import React, { useState } from 'react';
import axios from 'axios';
```

```
function UploadSingleFile() {

    const [file, setFile] = useState(null);


    const onChange = e => {
        setFile(Array.prototype.slice.call(e.target.files));
    }


    const onSubmit = async e => {
        e.preventDefault();

        file.map(async el => {
            let formData = new FormData();

            formData.append('name', 'file');
            formData.append('file', el);


            await axios.post("http://localhost:5000/upload", formData, {
                headers: {
                    "Content-type": "multipart/form-data",
                },
            }
            )

            console.log('file uploaded to server')
        })

    }



    return (
        <div>
            <form onSubmit={onSubmit}>
                <input type="file" onChange={onChange} multiple={true} />
                <input type="submit" value='submit' />
```

```
            </form>
        </div>
    )
}

export default UploadSingleFile
```

Display and Delete:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

export default function DisplayNDelete() {

    const [data, setData] = useState(null);



    const fetchData = async () => {
        let req = await axios.get('http://localhost:5000/all-files');
        // console.log(req.data);

        let fileNames = req.data.map(el => el.filename);
        console.log("fileNames: ", fileNames);



        setData(fileNames);

    }



    const delFile = async (fileName) => {
        await axios.delete(`http://localhost:5000/delete/${fileName}`);
        fetchData();
```

```
    }


    useEffect(() => {
        fetchData();
    }, [])


    return (
        <div>
            <h1>display images</h1>
            {data && data.map(el => {
                return (
                    <div key={el}>
                        <hr />
                        <img src={`http://localhost:5000/display/${el}`} alt=""
height='200px' width='200px' />
                        <br />
                        <button onClick={() => delFile(el)}>delete</button>
                    </div>
                )
            })}
        </div>
    )
}
```