**FILE UPLOAD IN NODE JS**

<https://scotch.io/tutorials/express-file-uploads-with-multer>

File upload is a common feature that almost every website needs. We will go through step by step on how to handle single and multiple file(s) upload with Express, save it to database (LokiJs), and retrieve the saved file for viewing.

The complete sourcecode is available here: <https://github.com/chybie/file-upload-express>.

We will be using Typescript throughout this tutorial.

[**Install Required Dependencies**](https://scotch.io/tutorials/express-file-uploads-with-multer#toc-install-required-dependencies)

// using npm

npm install express cors multer lokijs del --save

Notes:-

* [express](http://expressjs.com/): We will develop our API using ExpressJs
* [cors](https://github.com/expressjs/cors): A node.js package that provides an Express/Connect middleware to enable Cross Origin Resource Sharing (CORS)
* [multer](https://github.com/expressjs/multer): Node.js middleware for handling multipart/form-data.
* [loki](http://lokijs.org/): LokiJs, a fast, in-memory document-oriented datastore for node.js, browser and cordova
* [del](https://github.com/sindresorhus/del): Delete files and folders

**Development Dependencies**

Since we are using Typescript, we need to install typings files in order to have auto-complete function (intellesense) during development.

// or using npm

npm install typescript @types/express @types/multer @types/lokijs @types/del --save-dev

[Setup](https://scotch.io/tutorials/express-file-uploads-with-multer#toc-setup)

A couple of setup steps to go before we start.

Typescript Configuration

Add a typescript configuration file. To know more about Typescript configuration, visit <https://www.typescriptlang.org/docs/handbook/tsconfig-json.html>.

// tsconfig.json

{

"compilerOptions": {

"module": "commonjs",

"moduleResolution": "node",

"target": "es6",

"noImplicitAny": false,

"sourceMap": true,

"outDir": "dist"

}

}

Notes:-

1. The compiled javascript code will be output to dist folder.
2. Since Node Js 7.5+ support ES6 / 2015, we will set the target as es6.

**Start Script**

Add the following scripts.

// package.json

{

...

"scripts": {

"prestart": "tsc",

"start": "node dist/index.js"

}

...

}

Later on we can run yarn start or npm start to start our application.

Notes:-

1. When we run yarn start, it will trigger prestart script first. The command tsc will read the tsconfig.json file and compile all typescript files to javascript in dist folder.
2. Then, we will run the compiled index file dist/index.js.

[**Starting Express Server**](https://scotch.io/tutorials/express-file-uploads-with-multer#toc-starting-express-server)

Let's start creating our Express server.

// index.ts

import \* as express from 'express'

import \* as multer from 'multer'

import \* as cors from 'cors'

import \* as fs from 'fs'

import \* as path from 'path'

import \* as Loki from 'lokijs'

// setup

const DB\_NAME = 'db.json';

const COLLECTION\_NAME = 'images';

const UPLOAD\_PATH = 'uploads';

const upload = multer({ dest: `${UPLOAD\_PATH}/` }); // multer configuration

const db = new Loki(`${UPLOAD\_PATH}/${DB\_NAME}`, { persistenceMethod: 'fs' });

// app

const app = express();

app.use(cors());

app.listen(3000, function () {

console.log('listening on port 3000!');

});

The code is pretty expressive itself. We allow Cross-Origin Resource Sharing (CORS), set the connection port to 3000, and start the server.

[**Upload Single File**](https://scotch.io/tutorials/express-file-uploads-with-multer#toc-upload-single-file)

Let's create our first route. We will create a route to allow users to upload their profile avatar.

Route

// index.ts

...

import {

loadCollection

} from './utils';

...

app.post('/profile', upload.single('avatar'), async (req, res) => {

try {

const col = await loadCollection(COLLECTION\_NAME, db);

const data = col.insert(req.file);

db.saveDatabase();

res.send({ id: data.$loki, fileName: data.filename, originalName: data.originalname });

} catch (err) {

res.sendStatus(400);

}

})

Notes:

1. This is a HTTP POST function.
2. upload.single('avatar') is Multer middleware. It means we accept a single file with the field name avatar. File upload will be handled by Multer.
3. Multer will add a file property for request when it's a single file upload.
4. We will then load the LokiJs collection/table(we will create this function next), and insert the request file req.file to the collection.

**Load LokiJs Collection**

A generic function to retrieve a LokiJs collection if exists, or create a new one if it doesn't.

// utils.ts

import \* as del from 'del';

import \* as Loki from 'lokijs';

const loadCollection = function (colName, db: Loki): Promise<LokiCollection<any>> {

return new Promise(resolve => {

db.loadDatabase({}, () => {

const \_collection = db.getCollection(colName) || db.addCollection(colName);

resolve(\_collection);

})

});

}

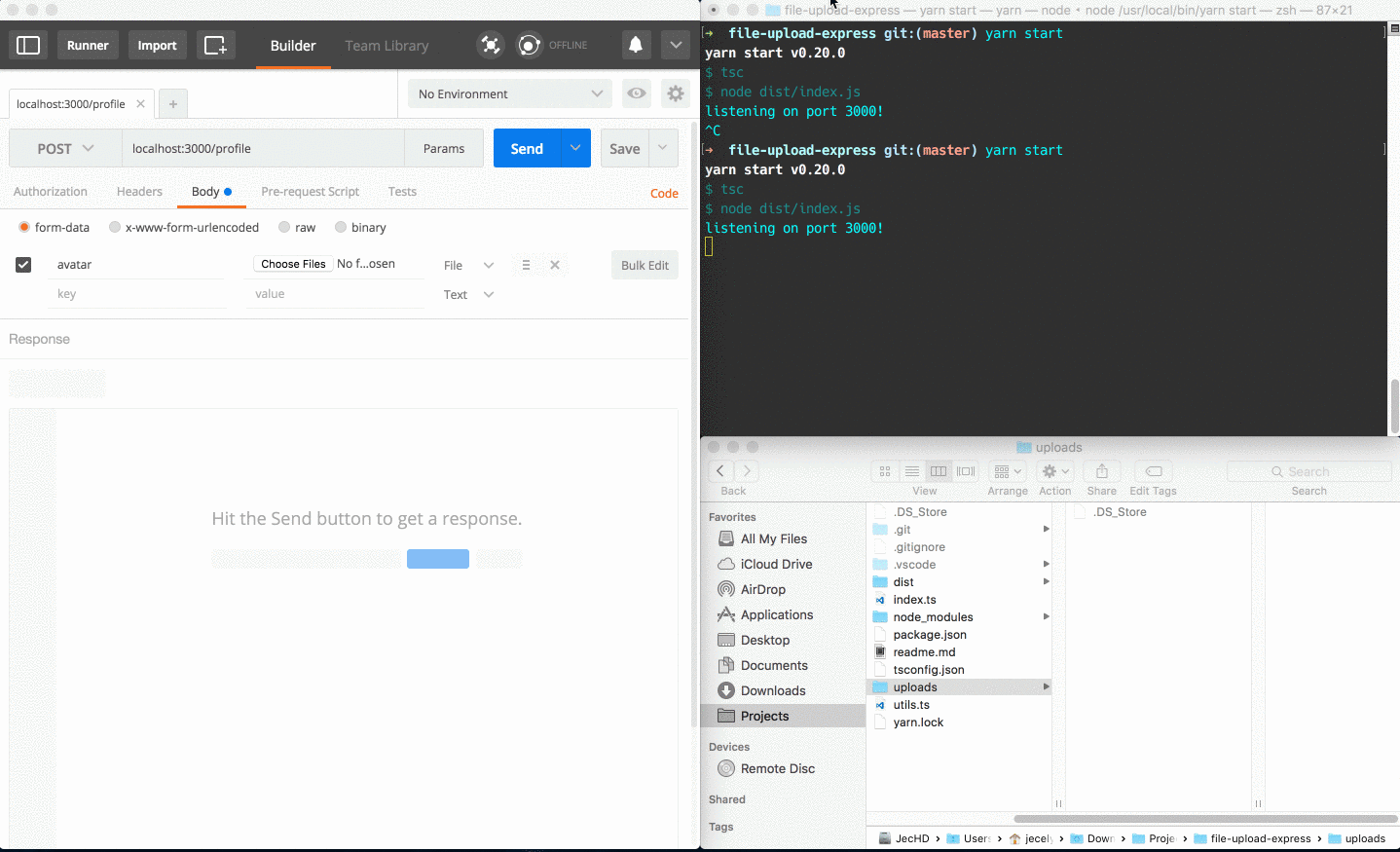
export { loadCollection }

[Run Our Application](https://scotch.io/tutorials/express-file-uploads-with-multer" \l "toc-run-our-application)

You may run the application with yarn start. I try to call the locahost:3000/profile API with (Postman)[<https://www.getpostman.com/apps>], an GUI application for API testing.

When I upload a file, you can see that a new file created in uploads folder and the database file db.json is created as well.

When I issue a call without passing in avatar, error will be returned.



[**Filter File Type**](https://scotch.io/tutorials/express-file-uploads-with-multer#toc-filter-file-type)

We can handle file upload successfully now. Next, we need to limit the file type to image only. To do this, let's create a filter function that will test the file extensions.

// utils.ts

...

const imageFilter = function (req, file, cb) {

// accept image only

if (!file.originalname.match(/\.(jpg|jpeg|png|gif)$/)) {

return cb(new Error('Only image files are allowed!'), false);

}

cb(null, true);

};

...

export { imageFilter, loadCollection }

**Apply the Image Filter**

We need to tell the Multer to apply our image filter function. Add it in upload variable.

// index.ts

...

import { imageFilter, loadCollection } from './utils';

...

// setup

...

const upload = multer({ dest: `${UPLOAD\_PATH}/`, fileFilter: imageFilter }); // apply filter

...

Restart the application, try to upload a non-image file and you should get error.

[Upload Multiple Files](https://scotch.io/tutorials/express-file-uploads-with-multer" \l "toc-upload-multiple-files)

Let's proceed to handle multiple files upload now. We will create a new route to allow user upload their photos.

Route

...

app.post('/photos/upload', upload.array('photos', 12), async (req, res) => {

try {

const col = await loadCollection(COLLECTION\_NAME, db)

let data = [].concat(col.insert(req.files));

db.saveDatabase();

res.send(data.map(x => ({ id: x.$loki, fileName: x.filename, originalName: x.originalname })));

} catch (err) {

res.sendStatus(400);

}

})

...

The code is similar to single file upload, except we accept a field photos instead of avatar, limit the total file to 12, accept an array of files as input and reply result as array.

[Retrieve List of Images](https://scotch.io/tutorials/express-file-uploads-with-multer#toc-retrieve-list-of-images)

Next, create a route to retrieve all images.

// index.ts

...

app.get('/images', async (req, res) => {

try {

const col = await loadCollection(COLLECTION\_NAME, db);

res.send(col.data);

} catch (err) {

res.sendStatus(400);

}

})

...

The code is super easy to understand.

[Retrieve Image by Id](https://scotch.io/tutorials/express-file-uploads-with-multer" \l "toc-retrieve-image-by-id)

Next, create a route to retrieve an image by id.

// index.ts

...

app.get('/images/:id', async (req, res) => {

try {

const col = await loadCollection(COLLECTION\_NAME, db);

const result = col.get(req.params.id);

if (!result) {

res.sendStatus(404);

return;

};

res.setHeader('Content-Type', result.mimetype);

fs.createReadStream(path.join(UPLOAD\_PATH, result.filename)).pipe(res);

} catch (err) {

res.sendStatus(400);

}

})

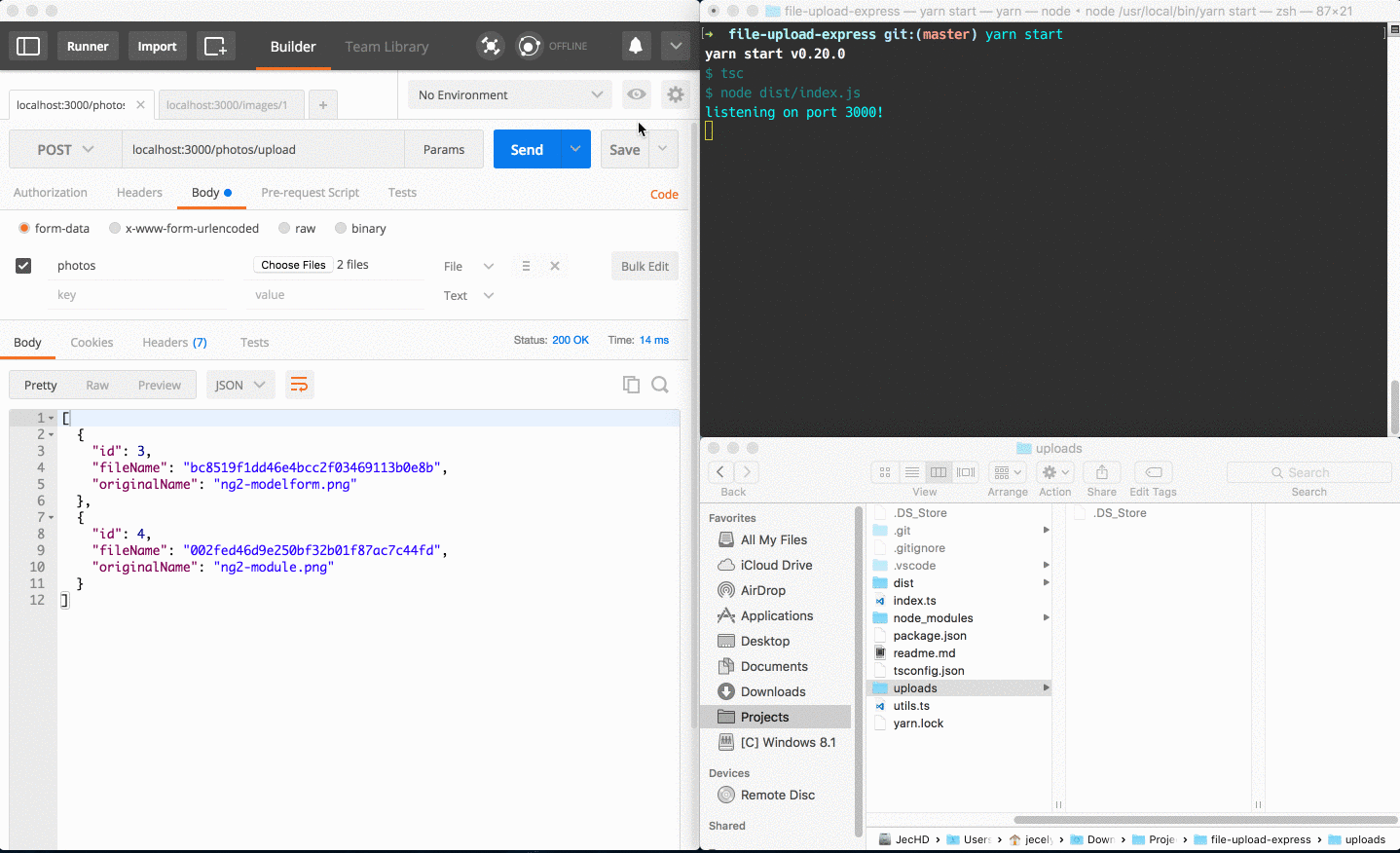
...

Notes:-

1. We will return 404 if image not exist in database.
2. We will stream the file as output, set the content-type correctly so our client or browser know how to handle it.

[Run the Application](https://scotch.io/tutorials/express-file-uploads-with-multer#toc-run-the-application)

Now restart the application, upload a couple of images, and retrieve it by id. You should see the image is return as image instead of json object.



[Clear All Data When Restart](https://scotch.io/tutorials/express-file-uploads-with-multer" \l "toc-clear-all-data-when-restart)

Sometimes, you might want to clear all the images and database collection during development. Here's a helper function to do so.

// utils.ts

....

const cleanFolder = function (folderPath) {

// delete files inside folder but not the folder itself

del.sync([`${folderPath}/\*\*`, `!${folderPath}`]);

};

...

export { imageFilter, loadCollection, cleanFolder }

Use it in our application.

// index.ts

...

import { imageFilter, loadCollection, cleanFolder } from './utils';

...

// setup

...

// optional: clean all data before start

cleanFolder(UPLOAD\_PATH);

...

[Summary](https://scotch.io/tutorials/express-file-uploads-with-multer" \l "toc-summary)

Handle file(s) upload with Express is easy with the help of Multer middleware. Multer is very flexible and configurable. You can go though the documentation and add more configuration as you need.

The complete sourcecode is available here: <https://github.com/chybie/file-upload-express>.

That's it. Happy coding.