

# Implémentation de Multer dans une API Node.JS



Julien M. · Follow

5 min read · Mar 7, 2023

 Share



Se connecter à Medium avec Google



Radoniaina Michael ANDRIAMBOLA  
andriambolaradoniainamichael@gmail.com



Caleb Dev  
calebdev777@gmail.com



## Introduction

**Multer** est une bibliothèque Node.js qui permet de gérer facilement l'envoi et la réception de fichiers dans une API avec Express.JS. Elle simplifie la gestion des fichiers en offrant une interface simple et intuitive pour travailler avec des fichiers téléchargés.

Lorsque vous créez une API, il est souvent nécessaire de permettre aux utilisateurs de télécharger des fichiers comme des images, des vidéos, des documents, etc. Cependant, gérer ces fichiers peut être un défi, notamment en raison des

contraintes de sécurité, de la gestion des erreurs et de la nécessité de stocker les fichiers sur le serveur.

C'est là que **Multer** entre en jeu. Cette bibliothèque simplifie grandement la gestion des fichiers en offrant des fonctionnalités telles que la vérification des types et des tailles de fichiers, le stockage de fichiers sur le serveur et la récupération des métadonnées de fichiers. Elle facilite également la manipulation de plusieurs fichiers simultanément.

## Installation et configuration

Avant de pouvoir utiliser **Multer** dans votre API, vous devez l'installer en utilisant le gestionnaire de paquets Node.js, NPM. Pour cela, exécutez la commande suivante :

```
npm install multer
```

Cela installera **Multer** et toutes ses dépendances dans votre application.

Une fois que Multer est installé, vous devez d'abord le configurer avant de pouvoir créer une instance et utiliser la méthode `multer()` .

### Configuration de la partie Storage

```
const storage = multer.diskStorage({
  destination: (
    _: ExpressRequest,
    __: Express.Multer.File,
    callback: CallableFunction,
  ) => {
    callback(null, 'uploads');
  },
  filename: (_, file, callback) => {
    const extArray = file.mimetype.split('/');
    const extension = extArray[extArray.length - 1];
    callback(null, `${uuid()}.${extension}`);
  },
});
```

Dans cet exemple, nous avons créé un stockage sur disque personnalisé en utilisant la méthode `diskStorage()` qui stocke les fichiers téléchargés dans le dossier `uploads/`. Nous avons également défini un format de nom de fichier personnalisé en ajoutant un uuid aléatoire suivi de l'extension du fichier uploadé.

- `uploads` : Permet de spécifier le dossier dans lequel seront uploadés les différents fichiers
- `callback(null, `${uuid()}.${extension}`);` : Permet de spécifier le nom du fichier. Dans cet exemple, on choisit volontairement de modifier le nom du fichier uploadé en utilisant le module `uuid`, qui génère un identifiant unique comme nom de fichier. Exemple : `2e20fcb6-bce3-11ed-afa1-0242ac120002.jpg`

### Configuration de la partie `FileFilter`

```
const fileFilter = (
  _: ExpressRequest,
  file: Express.Multer.File,
  callback: CallableFunction,
) => {
  if (!file.originalname.match(/\.(jpg|jpeg|png)$/))
    return callback(
      new Error('Only image files are allowed'),
      false,
    );
  callback(null, true);
};
```

Open in app ↗

Sign up

Sign in

Medium

🔍 Search



Dans cet exemple, nous verrons que seuls les fichiers avec une extension `.jpg`, `.JPEG` ou `.PNG` seront valides. Sinon la fonction renverra une erreur de type.

Il ne reste plus qu'à instancier **Multer** avec les options qui ont été définies juste au dessus.

*Exemple de fichier de configuration `storage.ts`*

```
import multer from 'multer';
import { Request } from 'express';
import { uuid } from 'uuidv4';

const storage = multer.diskStorage({
  destination: (
    _: Request,
    __: Express.Multer.File,
    callback: CallableFunction,
  ) => {
    callback(null, 'uploads');
  },
  filename: (_, file, callback) => {
    const extArray = file.mimetype.split('/');
    const extension = extArray[extArray.length - 1];
    callback(null, `${uuid()}.${extension}`);
  },
});

const fileFilter = (
  _: Request,
  file: Express.Multer.File,
  callback: CallableFunction,
) => {
  if (!file.originalname.match(/\.(jpg|jpeg|png)$/))
    return callback(
      new UnauthorizedError('Only image files are allowed'),
      false,
    );
  callback(null, true);
};

export const configurationStorage = () => multer({ storage, fileFilter });
```

## Utiliser Multer dans l'API

Prenons un exemple, vous avez développer une API Express.JS avec des routes permettant de récupérer, ajouter, modifier et supprimer des articles et aimeriez qu'à la création (POST) d'un article, une image soit uploadée avec **Multer**.

```
import PostController from '../controllers/PostController';
import express from "express";

const app = express();

// ... configuration de app
```

```
app.get("/posts", PostController.all);
app.get("/posts/:id", PostController.getById);
app.post("/posts", PostController.get);
app.put("/posts/:id", PostController.update);
app.delete("/posts/:id", PostController.delete);
```

Pour cela, il suffit d'instancier la classe **multer()**, qui utilise le système de storage et de `fileFilter` que nous avons déterminé plus haut, et spécifié un id particulier correspondant au champ à remplir pour l'upload de l'image.

```
// ...
import { configurationStorage } from './storage';
const multer = configurationStorage();

// ... routes de l'application

app.post("/posts", multer.single("picture"), PostController.get);
```

Nous avons créé une instance de Multer en utilisant la méthode `configurationStorage()`. Ensuite, nous avons configuré notre route de téléchargement en utilisant la méthode `upload.single()` qui indique à **Multer** de n'accepter qu'un seul fichier téléchargé.

- `single("picture")` : Permet d'uploader un seul fichier avec la clé `picture`
- `array("pictures", 10)` : Permet d'uploader plusieurs fichiers (10 au maximums) avec la clé `picture`

## Traiter le résultat de l'upload dans la requête

Quand vous utilisez votre API pour créer un article, une image est bien téléchargée dans le dossier `uploads`. Maintenant il ne reste plus qu'à traiter le résultat pour enregistrer le chemin de cette image dans la base de données par exemple.

```
import { Request, Response } from 'express';

export default class PostController {

  public async create(req: Request, res: Response) {
    if(!req.file.picture) throw new Error("Missing key picture");
```

```
const pictureFile = req.file.picture
// Fichier uploadé par Multer et passé dans la requête
// ... traitement annexe
res.sendStatus(201);
}

public async all() {
  // ...
}

public async getById() {
  // ...
}

public async update() {
  // ...
}

public async delete() {
  // ...
}
}
```

Dans cet exemple, nous utilisons la méthode **create** du PostController (utilisé dans les routes au dessus) pour créer un article. Multer a déjà uploadé le fichier dans le dossier `uploads` et envoi à la requête, les informations de l'image, sur la clé `picture` que nous avons spécifié dans la route.

Il ne reste plus qu'à choisir le traitement que vous voulez faire avec : enregistrer le nom du fichier ou le chemin en base de données, vérifier l'extension, etc...

## Liens utiles

- Documentation de Multer : <https://github.com/expressjs/multer#readme>
- Exemple complet d'API Express.JS utilisant **Multer** : <https://github.com/julienmaffar/boilerplate-nodejs-api>

# Implémenter Multer API Node.JS

```
callback: ()  
⇒ {  
  callback(nu  
  
  .ename: (__,  
  const extAr  
  const exten  
  callback(nu  
  
  : fileFilter  
  ExpressRec  
  .e: Express  
  .callback: Cal  
  {  
    (!file.ori  
    return call  
    new out  
    fa J.  
  }  
};
```

JavaScript

Typescript

Expressjs

Multer



Follow

**Written by Julien M.**

1 Follower

Fullstack JS Developer

More from Julien M.



Julien M.

## Le Design Pattern Repository au sein d'une API Node.JS

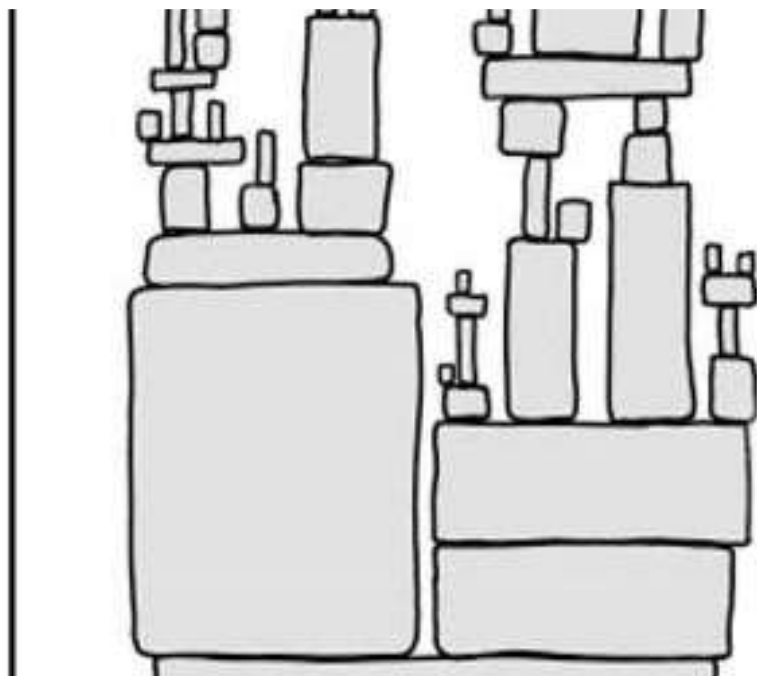
Le design pattern Repository et son implémentation concrète dans une API Node.JS


Mar 6, 2023

[See all from Julien M.](#)

Recommended from Medium





 Louis Trinh

## Mongoose Schema Modeling for E-commerce Products: Best Practices and Considerations

Mongoose Schema Model for Products in E-commerce

★ Jun 8 🖱 4



 user in JavaScript in Plain English

## Building a High-Performance Cache Manager in Node.js with TypeScript and ioredis

Introduction: In any performance-critical application, caching can play a pivotal role in reducing response times and easing database...

★ 3d ago 🖱 52



## Lists



### Stories to Help You Grow as a Software Developer

19 stories · 1447 saves



### General Coding Knowledge

20 stories · 1689 saves



### Generative AI Recommended Reading

52 stories · 1467 saves



### Visual Storytellers Playlist

61 stories · 529 saves

#### Amazon.com

##### Software Development Engineer

Seattle, WA

Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by \$25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

## Projects

#### NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

#### HeatMap (JavaScript)

- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay



Alexander Nguyen in Level Up Coding

## The resume that got a software engineer a \$300,000 job at Google.

1-page. Well-formatted.

★ Jun 1 🖱 24K 💬 478





 Vitaliy Korzhenko

## JavaScript Algorithm Interview Questions

### 1. Problem: Merge Sorted Arrays (Sorting Algorithm)

★ Oct 17 🖱 34



# *PDF Generation*



 habtesoft

## Generate pdf dynamically in nodejs

Generating PDFs dynamically in Node.js can be achieved using various libraries like pdfkit, puppeteer, or html-pdf. Each library has its...

★ Oct 5 🖱 7



```
console.log('start');
```

```
const promise1 = new Promise((resolve, reject) => {  
  console.log(1)  
  resolve(2)  
})
```

```
promise1.then(res => {  
  console.log(res)  
})
```

```
console.log('end');
```

**start** Shuai Li in Programming Domain

## Can You Answer This Senior Level JavaScript Promise Interview Question?

Most interviewees failed on it.

★ Aug 12 🖱 3.4K 💬 34

[See more recommendations](#)