Multiple File Upload with Apollo Server 3, React & GraphQL

AntoineGGG · Follow

Published in Geek Culture

5 min read · Sep 14, 2021

> *How to set up your multiple files upload with GraphQL*

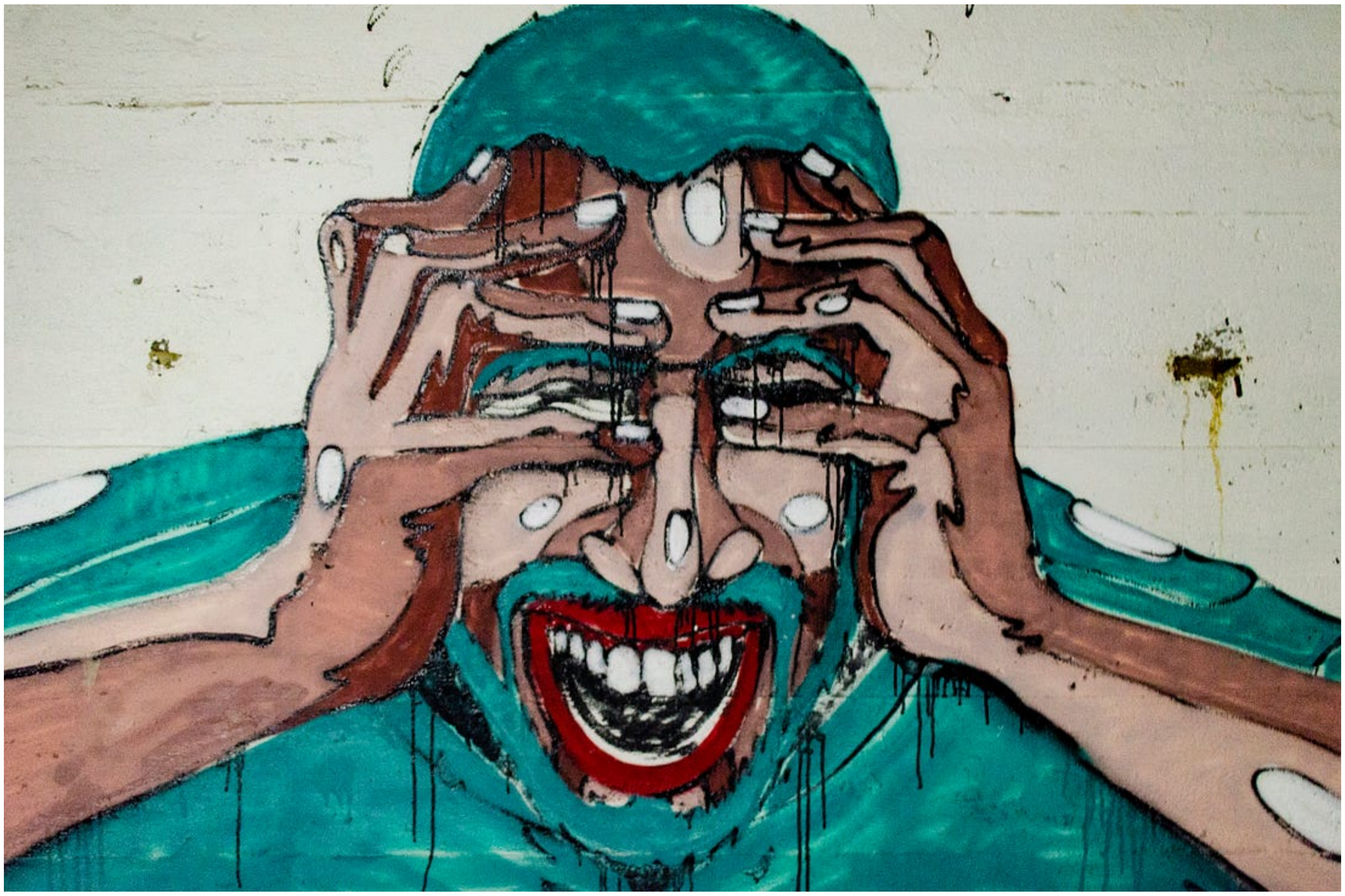


Photo by Aarón Blanco Tejedor on Unsplash

Introduction: In this paper, I would like to introduce the web application project I have been working on and explain the reasons behind its development. This project was undertaken as part of a school assignment, and we utilized the following technology stack:

- MongoDB: Used as the database to store our application's data.
- GraphQL with Apollo: Implemented for the backend and API functionality.
- React with GraphQL: Employed for the frontend development.

The primary objective of this project is to create a platform for painless resource aggregation and sharing. The aim is to provide users with a convenient way to share various types of resources, such as documents, tutorials, and other helpful information. Additionally, the application supports file uploads, enabling users to easily share relevant files with their friends and colleagues.

In the following sections, I will delve into the technical details of the project and discuss the implementation process, key features, and the overall user experience.

As someone who primarily works with REST APIs and the Multer package to handle form data, I found myself facing difficulties when transitioning to GraphQL and Apollo for a project. Despite investing two days of effort into resolving the issue, I was unable to successfully extract data from the file input in my front-end application. I scoured through numerous tutorials, consulted the Apollo documentation, explored the GraphQL-upload GitHub repository, and even turned to search engines and Stack Overflow for answers (as I'm sure many developers can relate).

In this paper, I aim to share the solution I discovered a few days ago after much trial and error.

Initially, our project began with Apollo Server V2. However, as we approached July, an important update arrived: Apollo Server V3. With the previous version, I encountered several issues, including an empty object being sent from the front-end, JSON errors with a misplaced "P" at position 0, and CreateReadStream errors due to deprecation in Node.js versions above 14.

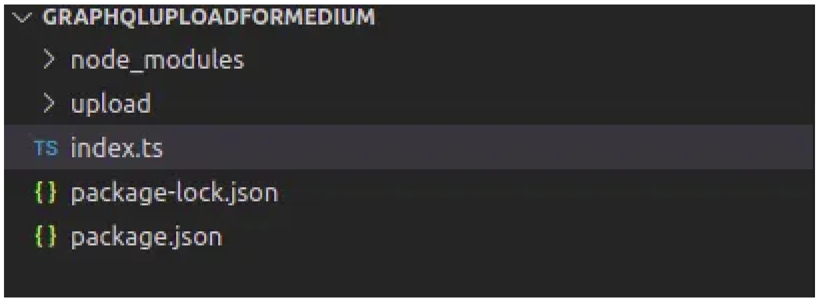
Let's delve into the steps I took to resolve these challenges and successfully integrate file uploads into our GraphQL and Apollo project.

Photo by Joseph Frank on Unsplash

The server:

- npm init -y
- npm i apollo-server apollo-server-express graphql graphql-upload typescript ts-

These commands will install all the necessary dependencies for this demo, with a specific focus on Apollo Server 3.


```
GRAPHQLUPLOADFORMEDIUM
  > node_modules
  > upload
  TS index.ts
  {} package-lock.json
  {} package.json
```

Please refer to the structure shown above, and find the contents of our index.ts file below:

```
const { ApolloServer, gql } = require("apollo-server-express");
const express = require("express");
import * as path from "path";
import * as fs from "fs";
import { GraphQLUpload, graphqlUploadExpress } from "graphql-upload";
const typeDefs = gql`
    scalar Upload
    type File {
      url
    }
    type Query {
      otherFields: Boolean!
    }
    type Mutation {
      fileUpload(file: [Upload]!): [File]!
    }
 `;
const fileRenamer = (filename: string): string => {
    const queHoraEs = Date.now();
    const regex = /[\s_-]/gi;
    const fileTemp = filename.replace(regex, ".");
    let arrTemp = [fileTemp.split(".")];
  return `${arrTemp[0]
  .slice(0, arrTemp[0].length - 1)
  .join("_")}${queHoraEs}.${arrTemp[0].pop()}`;
};
const resolvers = {
    Upload: GraphQLUpload,
    Mutation: {
      fileUpload: async (parent, { file }) => {
      let url = [];
      for (let i = 0; i < file.length; i++) {
      const { createReadStream, filename, mimetype } = await    file[i];
      const stream = createReadStream();
      const assetUniqName = fileRenamer(filename);
      const pathName = path.join(__dirname,   `./upload/${assetUniqName}`);
      await stream.pipe(fs.createWriteStream(pathName));
      const urlForArray = `http://localhost:4000/${assetUniqName}`;
      url.push({ url: urlForArray });
    }
   return url;
  },
 },
};
async function startServer() {
      const server = new ApolloServer({ typeDefs, resolvers });
      await server.start();
      const app = express();
      app.use(graphqlUploadExpress());
      server.applyMiddleware({ app });
      app.use(express.static(path.join(__dirname, "./upload")));
      await new Promise((r) => app.listen({ port: 4000 },     r));console.log(`🚀
    }
startServer();
```

Please pay attention to the following important details:

1. We are using `apollo-server-express` as the server framework.
2. The `path` and `fs` modules are utilized for file manipulation.
3. We are incorporating `graphql-upload` for file upload functionality.
4. Make sure to include the necessary Scalar Upload typeDefs for file uploads.
5. Similarly, include the GraphQLUpload resolver to handle file uploads.
6. The `fileRenamer` function is implemented to manage special characters (such as spaces, underscores, and dashes) in file names.
7. The `singleUpload` mutation is designed to accept an array of files, which are then processed by renaming and moving them to the appropriate folder.
8. The `startServer` function initiates the server with a middleware for file upload, which handles the necessary operations behind the scenes. Don't forget to use `express.static` to ensure your folders are accessible from the frontend.

These considerations are crucial for ensuring proper file upload functionality within your application.

The front App

We will be utilizing an incredibly fast React app builder:

```
npx create-react-app client --template clean-cra
```

To install all of these packages, run the following command:

```
npm i @apollo/client apollo-upload-client
```

The structure:

Configure your App.js as follows:

```
import React from "react";
import { ApolloProvider, ApolloClient, InMemoryCache } from "@apollo/client";
import { createUploadLink } from "apollo-upload-client";
import FileUpload from "./FileUPload";
function App() {
      const uploadLink = createUploadLink({
      uri: "http://localhost:4000/graphql" });
      const client = new ApolloClient({
         link: uploadLink,
         cache: new InMemoryCache(),
   });
return (
    <div className="App">
     <ApolloProvider client={client}>
     <FileUpload />
     </ApolloProvider>
   </div>
  );
}
export default App;
```

The most important part here is the usage of `createUploadLink`, which comes from the `apollo-upload-client` package. This package manages the multipart request created by GraphQL for us.

Here's the implementation of the `FileUpload` component:

```
   import React from "react";
   import { useMutation, gql } from "@apollo/client";
   const UPLOAD_FILE = gql`
         mutation fileUpload($file: [Upload]!) {
         fileUpload(file: $file) {
            url
        }
      }
   `;
   const FileUPload = () => {
         const [fileUpload] = useMutation(UPLOAD_FILE, {
             onCompleted: (data) => console.log(data),
         });
         const handleFileChange = (e) => {
           const file = e.target.files;
           if (!file) return;
           fileUpload({ variables: { file } });
         };
      return (
      <>
       <input
         type="file"
         name="GraphQLUploadForMedium"
         onChange={handleFileChange}
       />
     </>
```

```
  );
};
export default FileUPload;
```

And there you have it! With this method, you can successfully send multiple files to your GraphQL API and access them using their respective URLs.

If you have any suggestions or advice on how to improve this process further, please feel free to comment or reach out to me. I'm always open to feedback and eager to enhance the functionality.

GraphQL React Apollo Server Upload JavaScript

Follow

Written by AntoineGGG

288 Followers · Writer for Geek Culture

Tech Padawan

More from AntoineGGG and Geek Culture

AntoineGGG in Geek Culture

Using prevState with React, basic and complex

How to use the famous “prevState” to modify our React state without overriding it !

· 4 min read · Jun 18, 2021

220 1

Joanna in Geek Culture

Understanding the 10 Most Difficult Python Concepts

Python has a simple syntax and is easy to learn, making it an ideal choice for beginners. However, as you delve deeper into Python, you may...

· 8 min read · Apr 4

531 2

Jacob Bennett in Geek Culture

The 5 paid subscriptions I actually use in 2023 as a software engineer

Tools I use that are cheaper than Netflix

· 4 min read · Mar 25

4.2K 37

AntoineGGG

How to get ansible vscode extension working…

Or how to lint your ansible yaml files properly !

· 2 min read · Apr 17

12

See all from AntoineGGG

See all from Geek Culture

Recommended from Medium

Gen in Dev Genius

Upload Files to S3 on Frontend

Here is the guide to show you how to set up an S3 bucket including the bucket access + CORS configuration, and get the user access key +...

6 min read · Mar 7

14 1

Daniel Craciun

Integrate Google reCAPTCHA v2 With Next.js 13 In Under 10 Minutes

reCAPTCHA is a widely known form of authentication used worldwide to filter and monitor incoming bot traffic towards websites.

3 min read · Jul 10

4 1

Lists

Stories to Help You Grow as a Software Developer
19 stories · 321 saves

General Coding Knowledge
20 stories · 266 saves

Medium Publications Accepting Story Submissions
147 stories · 476 saves

Modern Marketing
33 stories · 98 saves

Jeswanth Reddy in Version 1

Difference Between Promise and Async/Await

If you're reading this, you probably understand how the promise and async/await are different in the execution context.

2 min read · May 12

791 8

Theodore John.S

Load, Unload, and BeforeUnload Event Listeners in React: Significance, Use Cases, Best Practices…

Lets implement load, unload, and beforeunload event listeners to create dynamic and interactive web applications. Master the art of…

4 min read · Jun 29

5

Rehan choudhury in JavaScript in Plain English

Next.js Secure Authentication Using http-only Cookie (GraphQL or REST)

When it comes to user authentication we need to make sure our application is secured from any potential threats. To implement such type of...

6 min read · May 12

37

Olga Green

Building Dynamic Forms with React Hook Form

In this article, we will be exploring how to use React Hook Form to create dynamic forms that can adapt to changing input requirements.

12 min read · Mar 17

3

See more recommendations