Short Notes on Cloudinary Upload and Multer - Backend Development

☁️ 1. Cloudinary File Upload (Image/File Storage)

✅ What is Cloudinary?

- A cloud service to store and manage images, videos, and other media files.
- Avoids storing heavy media files on your local server.

✅ Code Explanation

```
import { v2 as cloudinary } from "cloudinary";

import fs from "fs";
```

- cloudinary: SDK to interact with Cloudinary services.
- fs: Node's file system module to manage local files.

🔧 Configuration

```
cloudinary.config({

cloud_name: process.env.CLOUDINARY_NAME,

api_key: process.env.CLOUDINARY_API_KEY,

api_secret: process.env.CLOUDINARY_API_SECRET,

});
```

- Use environment variables to securely store credentials.
- These are required to connect your app to Cloudinary.

📤 Upload Function

```
const uploadOnCloudinary = async (localFilePath) => {

try {

if (!localFilePath) return null;

const response = await cloudinary.uploader.upload(localFilePath, {

resource_type: "auto",

});

console.log("file has been uploaded successfully", response.url);

return response;

} catch (error) {

fs.unlinkSync(localFilePath); // delete file from local temp folder

}

};
```

- Takes local file path and uploads to Cloudinary.
- resource_type: 'auto' automatically detects file type (image, video, etc.).
- On success, it returns the uploaded file’s URL.
- On error, it deletes the local temp file to save space.

🔁 Usage Flow

1. File is uploaded to your server (locally saved).
2. uploadOnCloudinary uploads it to Cloudinary.
3. On success, the file URL is returned.
4. The local file is deleted to save space.

📁 2. Multer (Local File Upload)

✅ What is Multer?

- A middleware for handling multipart/form-data, mainly used for uploading files.

✅ Code Explanation

```
import multer from "multer";
```

- multer: Required to set up file upload logic.

🧠 Multer Storage Setup

```
const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, "/public/temp");

},

filename: function (req, file, cb) {

cb(null, file.originalname);

},

});
```

- diskStorage: Saves uploaded files to disk (your local storage).
- destination: Where to store uploaded files (e.g., /public/temp).
- filename: Keeps the original filename. You can make it unique by appending timestamps if needed.

🔄 Exporting Upload Middleware

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

- This upload middleware will be used in your route to handle file uploads.
- Example usage:

```
router.post("/upload", upload.single("file"), handlerFunction);
```

🔗 Combined Use (Multer + Cloudinary)

1. Use Multer to upload the file to a local folder.
2. Call uploadOnCloudinary with the local path.
3. Upload it to Cloudinary.
4. On success, delete local file and use Cloudinary URL.

✅ Best Practices

- Always remove local files after uploading to Cloudinary.
- Store Cloudinary credentials in .env file.
- Use unique names for files if needed to avoid overwriting.
- Validate file types before uploading.

📌 Summary Table

Tool	Purpose	Where Used
Cloudinary	Cloud storage for media	Remote (cloud)
Multer	Handle file upload from frontend	Local (temporary folder)
fs	Manage local files (delete, etc.)	Node.js built-in module

Use this as a quick revision guide whenever you're handling image or file uploads in your backend project!