utils/cloudinary.js

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

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

const uploadOnCloudinary = async (localfilepath) => {
  try {
    if (!localfilepath) return null;

    //upload the file on cloudinary
    const cloudinaryResponse = await cloudinary.uploader.upload(localfilepath, {
      resource_type: "auto",
    });
    //file has been uploaded successfully
    console.log("File Uploaded on Cloudinary Successfully", response.url);
    return response;
  } catch (error) {
    fs.unlinkSync(localfilepath); //remove the locally saved temporary file as the upload operation
    return null;
  }
};

export { uploadOnCloudinary };
```

Explaination

Cloudinary File Upload using Node.js

The given code is a Node.js module that utilizes the `cloudinary` library to upload a file to the Cloudinary cloud storage service. The `cloudinary` library is a popular solution for managing and delivering images and other media files.

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

// Configure Cloudinary with API credentials
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

// Function to upload a file to Cloudinary
```

```
// Function to upload a file to Cloudinary
const uploadOnCloudinary = async (localfilepath) => {
  try {
    if (!localfilepath) return null;

    // Upload the file on Cloudinary
    const cloudinaryResponse = await cloudinary.uploader.upload(localfilepath, {
      resource_type: "auto",
    });

    // File has been uploaded successfully
    console.log("File Uploaded on Cloudinary Successfully", cloudinaryResponse.url);
    return cloudinaryResponse;

  } catch (error) {
    // Remove the locally saved temporary file as the upload operation failed
    fs.unlinkSync(localfilepath);
    return null;
  }
};

export { uploadOnCloudinary };
```

Code Explanation:

Cloudinary Configuration:

- **The** `cloudinary.config` **method** is used to configure the Cloudinary SDK with the required API credentials, retrieved from environment variables.

`uploadOnCloudinary` Function:

- This asynchronous function takes a `localfilepath` parameter representing the path to the local file to be uploaded.

File Upload:

- The `cloudinary.uploader.upload` method is used to upload the file to Cloudinary. The `resource_type: "auto"` option allows Cloudinary to determine the resource type automatically.

Success Handling:

- If the upload is successful, the Cloudinary response is logged, and the response object is returned.

Error Handling:

- If an error occurs during the upload, the locally saved temporary file is removed using `fs.unlinkSync` , and `null` is returned.

Summary:

This code provides a modular function, `uploadOnCloudinary`, that allows for easy integration with Cloudinary to upload files. The function includes error handling to clean up locally saved files in case of upload failures.

middlewares/multer.middleware.js

```
import multer from "multer";

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "./public/temp");
  },
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() * 1e9);
    cb(null, file.fieldname + "-" + uniqueSuffix);
  },
});

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

Multer Configuration for File Upload

The given code configures the Multer middleware for handling file uploads in a Node.js application. Multer is a popular middleware for handling `multipart/form-data`, commonly used for file uploads.

```
import multer from "multer";

// Multer storage configuration
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "./public/temp");
  },
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() * 1e9);
    cb(null, file.fieldname + "-" + uniqueSuffix);
  },
});

// Exporting the configured multer middleware
export const upload = multer({ storage: storage });
```

Code Explanation: Multer Storage Configuration:

multer.diskStorage: Multer provides a disk storage engine that saves uploaded files to the server's disk.

destination: Specifies the directory where uploaded files will be stored. In this case, it's set to "./public/temp".

filename: Defines how the file should be named. It generates a unique filename based on the original fieldname and a unique suffix.

Exporting Multer Middleware:

The configured Multer middleware is exported as upload. This middleware can be used to handle file uploads in route handlers. Summary: This code sets up Multer to handle file uploads in a Node.js application. Uploaded files will be stored in the "./public/temp" directory with unique filenames generated based on the original filename and a timestamp. The upload middleware can be integrated into route handlers to process file uploads.