**Cloudinary**

**Cloudinary** is a cloud-based service that provides a comprehensive solution for managing and delivering images and videos. It allows developers to upload, store, manage, manipulate, and deliver media assets in web and mobile applications. Cloudinary offers powerful features such as automatic optimization, responsive delivery, and various media transformations, such as resizing, cropping, and format conversion.

**Key Features of Cloudinary:**

1. **Image and Video Uploading**: Cloudinary supports easy uploads for images and videos from local files, URLs, and even directly from user devices (via APIs).
2. **Media Transformation**: You can apply transformations to media assets, such as resizing, cropping, rotating, adding filters, and changing formats (e.g., from PNG to JPG, from MOV to MP4).
3. **CDN Delivery**: Cloudinary uses a Content Delivery Network (CDN) for fast and efficient delivery of images and videos to users worldwide. It automatically serves optimized versions based on device characteristics.
4. **Automatic Optimization**: Cloudinary automatically optimizes images and videos to ensure the best balance between quality and file size, improving loading times.
5. **Management Interface**: Cloudinary provides a user-friendly dashboard to manage your media files, apply transformations, and track usage.
6. **APIs and SDKs**: Cloudinary offers APIs and SDKs for a variety of platforms, including JavaScript, Node.js, Ruby, Python, and mobile SDKs for iOS and Android.

Installation:

npm i cloudinary

**Real world application code**

import { v2 as cloudinary } from "cloudinary";

import fs from "fs";

cloudinary.config({

    cloud\_name: process.env.CLOUDINARY\_NAME,

    api\_key: process.env.CLOUDINARY\_API\_KEY,

    api\_secret: process.env.CLOUDINARY\_API\_SECRET,

});

const uploadOnCloudinary = async (localFilePath) => {

    try {

        if (!localFilePath) return null;

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

            resource\_type: "auto",

        });

        //? file has been uploaded successfully

        return response;

    } catch (error) {

        console.log("ERROR OCCURED WHILE UPLOADING ON CLOUDINARY ::: ", error);

        // remove the locally saved file as the uplod operaiton done

        fs.unlinkSync(localFilePath);

    } finally {

        fs.unlinkSync(localFilePath);

    }

};

export { uploadOnCloudinary };

This code defines a utility function for uploading files to Cloudinary using the cloudinary package, with additional functionality to clean up temporary files stored locally after the upload operation.

**Code Explanation**

1. **Cloudinary Configuration**

cloudinary.config({

    cloud\_name: process.env.CLOUDINARY\_NAME,

    api\_key: process.env.CLOUDINARY\_API\_KEY,

    api\_secret: process.env.CLOUDINARY\_API\_SECRET,

});

* + **cloud\_name**, **api\_key**, and **api\_secret**: These credentials are retrieved from environment variables (process.env). They authenticate the application with Cloudinary.
  + Ensure these variables are correctly set in your .env file:

CLOUDINARY\_NAME=your-cloudinary-name

CLOUDINARY\_API\_KEY=your-cloudinary-api-key

CLOUDINARY\_API\_SECRET=your-cloudinary-api-secret

1. **The Upload Function**

const uploadOnCloudinary = async (localFilePath) => {

* + **Input**: localFilePath - The file path of the local file to be uploaded.
  + **Output**: Returns the response from Cloudinary if the upload is successful or null if no file path is provided.

1. **Upload Logic**

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

            resource\_type: "auto",

        });

        //? file has been uploaded successfully

        return response;

* + **cloudinary.uploader.upload()**: This method uploads the file to Cloudinary.
    - **localFilePath**: Path to the file to upload.
    - **Options**:
      * resource\_type: "auto": Automatically detects the file type (e.g., image, video, raw).
  + **Return Value**: The response contains details about the uploaded file, such as its URL, public ID, file type, etc.

1. **Error Handling**

catch (error) {

        console.log("ERROR OCCURED WHILE UPLOADING ON CLOUDINARY ::: ", error);

        // remove the locally saved file as the uplod operaiton done

        fs.unlinkSync(localFilePath);

* + Logs any error that occurs during the upload.
  + **fs.unlinkSync(localFilePath)**: Deletes the local file if an error occurs, ensuring temporary files don’t accumulate.

1. **File Cleanup**

finally {

        fs.unlinkSync(localFilePath);

* + **Purpose**: Regardless of success or failure, the local file is deleted after the upload attempt to prevent unnecessary storage usage.