Repository Content

.gitkeep

```
```

.gitignore

```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
lerna-debug.log*
.pnpm-debug.log*

# Diagnostic reports (https://nodejs.org/api/report.html)
report.[0-9]*.[0-9]*.[0-9]*.[0-9]*.json

# Runtime data
pids
*.pid
*.seed
*.pid.lock

# Directory for instrumented libs generated by jscoverage/JSCover
lib-cov

# Coverage directory used by tools like istanbul
coverage
*.lcov

# nyc test coverage
```

package-lock.json

```
[object Object]
```

package.json

```
[object Object]
```

.gitkeep

```
```

app.js

```
import express from 'express'
import cors from 'cors'
import cookieParser from 'cookie-parser'

const app = express();

app.use(cors({
    origin: process.env.CORS_ORIGIN,
    credentials:true
}))

app.use(express.urlencoded({
    extended: true,
    limit: '10mb'
}))

app.use(express.static('public'))

app.use(cookieParser())

import imageRouter from './routes/images.routes.js'

app.use('/api/v1/images', imageRouter)

export { app }
```

constants.js

```
export const DB_NAME = 'image_manipulation'
```

.gitkeep

```
```

imagesUpload.controller.js

```
import path from 'path'
import { fileURLToPath } from 'url';
import { uploadOnCloudinary } from '../utils/cloudinary.js'
import { Image } from '../models/image.model.js';
import {
    handlingImageBlur,
    handlingImageFormatConversion,
    handlingImageGreyscale,
    handlingImageNegative,
    handlingImageRotation,
    handlingImageTint
} from '../utils/sharp.js';
import { asyncHandler } from '../utils/AsyncHandler.js';
import { ApiError } from '../utils/ApiError.js';
import { ApiResponse } from '../utils/ApiResponse.js'

const generatingFileName = (format) => {
    const currentDate = new Date().toISOString().split('T')[0]
    const randomInteger = Array.from({length: 19}, () => Math.floor(Math.random() * 10)).join('').padStart(20,
Math.floor(Math.random() * 9) + 1)
    const uniqueName = `${currentDate}-${randomInteger}`
    const convertedFileName = `${uniqueName}.${format}`
    const __dirname = path.dirname(fileURLToPath(import.meta.url));
    const convertedLocalPath = path.join(__dirname, '../../converted', convertedFileName)

    return convertedLocalPath
```

.gitkeep

```
```

index.js

```
import { connect } from 'mongoose'
import { DB_NAME } from '../constants.js'

const connectDB = async () => {
    try {
        const connectionInstance = await connect(`${process.env.MONGO_URI}/${DB_NAME}`)
        console.log(`::: MONGO DB CONNECTED :: DB HOST :: ${connectionInstance.connection.host} :: PORT ::
${connectionInstance.connection.port} :: NAME :: ${connectionInstance.connection.name} :::`);

    } catch (error) {
        console.log(`::: MONGODB CONNECTION FAILED :: ${error} :::`);
        process.exit(1)
    }
}

export default connectDB
```

index.js

```
import { app } from "./app.js";
import dotenv from 'dotenv'
import connectDB from "./db/index.js";

dotenv.config({
    path: './.env'
})

connectDB()
.then(() => {
    app.on('error', (error) => {
        console.log(`::: Error :: ${error} :::`);
        throw error
    })
    app.listen(process.env.PORT || 8000, () => {
        console.log(`::: Server is running at Port :: ${process.env.PORT} :::`);
    })
})
.catch((error) => {
    console.log(`::: MONGO DB CONNECTION FAILED!!! :: ${error} :::`);
})
```

.gitkeep

```

```

multer.middleware.js

```
import multer from 'multer'

const storage = multer.diskStorage({

    // destination where the images will upload in the server

    destination: function(req,file,cb) {
        cb(null, './public/temp') // Directory where the file gets saved
    },
    filename: function (req, file, cb) {

        // Generating a unique name for the uploaded file
        const currentDate = new Date().toISOString().split('T')[0];
        const randomSixDigit = Array.from({length: 19}, () => Math.floor(Math.random() *
10)).join('').padStart(20, Math.floor(Math.random() * 9) + 1);
        const uniqueSuffix = currentDate + '-' + randomSixDigit;

        // Using the file extension
        const fileExtension = file.originalname.split('.').pop();

        // Save with unique name
        cb(null,uniqueSuffix + '.' + fileExtension);
    }
})
```

.gitkeep

```

```

image.model.js

```
import { Schema, model } from "mongoose";
import jwt from 'jsonwebtoken';

const imageSchema = new Schema(
    {
        imageUrl: {
            type: String,
            required: true,
        },
        format: {
            type: String,
            enum: ["jpg", "png", "gif", "avif", "webp"],
            required: false,
        },
        publicID: {
            type: String,
            required: true,
        },
        token: {
            type: String,
            required: false
        }
    },
    {
        timestamps: true,
    }
```

.gitkeep

```
```

images.routes.js

```
import { Router } from "express";
import {
    changingImageFormat,
    changingImageGreyScale,
    changingImageTint,
    changingImageRotation,
    changingImageBlur,
    changingImageNegative
} from "../controllers/imagesUpload.controller.js";
import upload  from '../middlewares/multer.middleware.js'

const router = Router()

router.route('/image-format').post(
    upload.fields([
        {
            name: 'image',
            maxCount: 1
        }
    ]),changingImageFormat)

router.route('/image-greyscale').post(
    upload.fields([
        {
            name: 'image',
            maxCount: 1
```

.gitkeep

```
```

ApiError.js

```
class ApiError extends Error {
    constructor(
        statusCode,
        message = "Something went wrong",
        errors = [],
        stack = ""
    ) {
        super(message)
        this.statusCode = statusCode
        this.data = null
        this.message
        this.success = false;
        this.errors = errors

        if(stack) {
            this.stack = stack
        } else {
            Error.captureStackTrace(this, this.constructor)
        }
    }
}

export {ApiError}
```

ApiResponse.js

```
class ApiResponse {
    constructor(
        statusCode,
        data,
        message = "Success"
    ) {
        this.statusCode = statusCode,
        this.data = data,
        this.message = message,
        this.success = statusCode < 400
    }
}

export { ApiResponse }
```

AsyncHandler.js

```
const asyncHandler = (requestHandlerFunction) => async (req,res,next) => {
    try {
        return await requestHandlerFunction(req,res,next)
    } catch (error) {
        res.status(error.code || 500).json({
            success: false,
            message: error.message
        })
    }
}

export {asyncHandler}
```

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

        const uniquePublicId = Math.floor(Math.random() * 999999)

        // upload file on cloudinary with presets
        const response = await cloudinary.uploader.upload(localFilePath,{
            resource_type: 'auto',
            public_id: uniquePublicId,
            folder: 'to_be_converted',
            allowed_formats: ['jpg','png','webp','tiff','gif','avif']
        })
        console.log(`::: File is Uploaded in Cloudinary :::`);
        fs.unlinkSync(localFilePath)
        return response;
```

sharp.js

```
import sharp from 'sharp'

//* Output Options

export const handlingImageFormatConversion = async (inputFilePath,outputFilePath,outputFormat) => {
    try {
        await sharp(inputFilePath)
            .toFormat(outputFormat)
            .toFile(outputFilePath)
        console.log(`::: Your file saved in ${outputFilePath} :::`);
    } catch (error) {
        throw new Error(`::: Error changing format :: ${error.message} :::`)
    }
}

//* Image manipulation

export const handlingImageGreyscale = async (inputFilePath,outputFilePath) => {
    try {
        await sharp(inputFilePath)
            .greyscale()
            .toFile(outputFilePath)
        console.log(`::: Your file saved in ${outputFilePath} :::`);
    } catch (error) {
        throw new Error(`::: Error converting to greyscale :: ${error.message} :::`)
    }
```