**Geo\_processor\_app**

Teacher: Myself and the universe

Date started: Good Friday (7 April) 2023

**Contents**

[1 Introduction 1](#_Toc131757970)

[1.1 What does the app do 1](#_Toc131757971)

[2 File upload 1](#_Toc131757972)

[2.1 Testing in Postman 3](#_Toc131757973)

[3 List all the files in public/uploads 3](#_Toc131757974)

[4 Reading an uploaded file 3](#_Toc131757975)

[5 Projecting coordinates from MGA to lat/long 4](#_Toc131757976)

[5.1 POST function 4](#_Toc131757977)

[6 Extract elevation along a path using Google Maps API 5](#_Toc131757978)

# Introduction

Documentation of the creation of my own geo\_processor\_app.

This app is made in node-express, with ejs template engine for the frontend.

For this learning process I am not using react or nextjs.

## What does the app do

To be completed at the end

# File upload

The package ‘express-fileupload’ must be imported into app.js.

app.js:

import 'dotenv/config';

import 'express-async-errors';

import router from './routes/routes.js';

import express from 'express'

import cors from 'cors'

const app = express();

app.use(cors({origin: 'http://localhost:5000'}))

import fileUpload from 'express-fileupload' //required to read file upload request.body

// // error handler

// import notFoundMiddleware from './middleware/not-found.js';

// import errorHandlerMiddleware from './middleware/error-handler.js';

app.set('view engine','ejs') //template engine

app.use('/api/v1', express.static('./public')) //public folder to serve static files

//json middleware

app.use(express.json())

//req body processing middleware

app.use(express.urlencoded({ extended: true }))

//fileupload middleware

app.use(fileUpload({useTempFiles: true}))

app.get('/', (req, res) => {

res.send('Welcome to the google maps testing app. This is not working yet, I am not convinced by the google maps api');

});

//root path

app.use('/api/v1/', router)

// // middleware

// app.use(notFoundMiddleware);

// app.use(errorHandlerMiddleware);

const port = process.env.PORT || 5000;

//server listening

app.listen(port,console.log(`Server is listening on port ${port}....`))

Then in controllers/fileUpload.js:

import { StatusCodes } from 'http-status-codes'

export const uploadGeoJsonFile = async (req, res) => {

//upload a GeoJson file to the public folder

console.log(req.files)

let myFile = req.files.myFile

let myFilePath = new URL ('../public/uploads/'+`${myFile.name}`, import.meta.url).pathname.slice(1)

await myFile.mv(myFilePath)

return res.status(StatusCodes.OK)

.json(

{uploadedFile: {src: `/uploads/${myFile.name}`}}

)

}

We must also create a folder public/uploads to allow this (POST request) to work.

routes/routes.js:

import express from 'express'

import { uploadGeoJsonFile } from '../controllers/fileUpload.js'

import { welcome } from '../controllers/views.js'

const router = express.Router()

router.route('/').get(welcome)

router.route('/upload').post(uploadGeoJsonFile)

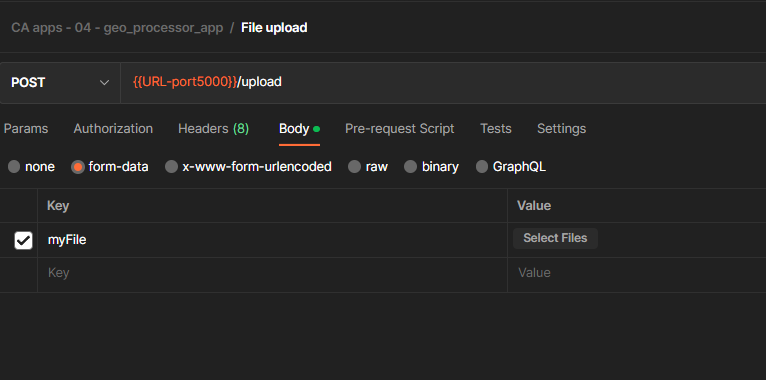
export default router

## Testing in Postman

We create a post request with a form-data body, where we add in a key of type file.

We can then select a file and test it out.

We then see our selected file uploaded to the public/uploads folder



# List all the files in public/uploads

We create a function to return an object of the available files in public/uploads.

Controllers/views.js

import fs from 'fs'

export const welcome = async(req, res) => {

res.send('Welcome to the index page')

}

export const listAllFiles = async (req, res) => {

const myFiles = await fs.promises.readdir('./public/uploads/')

myFiles.map(item => console.log(item))

res.json(myFiles)

}

And we add this as a route in routes/routes.js:

router.route('/listfiles').get(listAllFiles)

# Reading an uploaded file

We create a function (in controllers/geoprocessing.js) to read the contents of a file in the uploads folder and return an object.

import fs from 'fs'

async function readFile(fileName) {

const filePath = new URL ('../public/uploads/'+`${fileName}`, import.meta.url).pathname.slice(1)

let data = await fs.promises.readFile(filePath,'utf-8')

data = JSON.parse(data)

return data

}

# Projecting coordinates from MGA to lat/long

We create a function (in controllers/geoprocessing.js) to transform coordinates of a geojson object from MGA to lat/long.

We use the proj4 library: <https://github.com/proj4js/proj4js>

Npm install proj4

We also require epsg strings in proj4 format. These can be fetched from the web for example:

async function fetchEpsgString(epsg\_number) {

//fetch epsg proj4 string from web

let code = await fetch(`https://epsg.io/${epsg\_number}.proj4`)

return code.text()

}

This site is a good reference for epsg strings: <https://epsg.io/>

To avoid fetching every time we create an object (in controllers/geoprocessing.js) with some important ones:

const epsg\_strings\_proj4 = {

'4326': '+proj=longlat +datum=WGS84 +no\_defs +type=crs',

'7855': '+proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no\_defs +type=crs',

'7856': '+proj=utm +zone=56 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no\_defs +type=crs'

}

Now we can create the projection function (in controllers/geoprocessing.js):

async function dataToLatLong(data) {

//data is a geojson object

//get input data epsg code

let epsg\_code = data.crs.properties.name

epsg\_code = epsg\_code.slice(epsg\_code.length - 4, epsg\_code.length)

//extract coords

let coords = data.features[0].geometry.coordinates[0]

//project coords from input epsg to lat/lon

let coordsProj = coords.map(item =>

proj4(epsg\_strings\_proj4[epsg\_code],epsg\_strings\_proj4['4326'],item))

//create new object

let newData = JSON.parse(JSON.stringify(data))

newData.features[0].geometry.coordinates[0] = coordsProj

newData.crs.properties.name = 'urn:ogc:def:crs:EPSG::4326' //re-label the crs properties name

return newData

## POST function

We now create a post function which reads a requested geojson file, and then transforms the coordinates from input crs to lat/long (epsg: 4326).

export const projectToLatLong = async(req, res) => {

//read geojson file in the public folder and reproject coords to lat/long

//TODO ERROR CHECKING:

// 1. must check it is a geojson file

//read input data

let data = await readFile(req.body.fileName)

//project to lat/long

let newData = await dataToLatLong(data)

//send processed data to client

res.json(newData)

}

This is now included as an app route in routes/routes.js:

router.route('/process/project').post(projectToLatLong)

# Extract elevation along a path using Google Maps API

TODO: Complete this section on extraction of elevation along line using backend. (getElevation and splitLine functions)