Working app.py -

```
from flask import Flask, render_template, request, send_file
import subprocess
import os
from osgeo import gdal, ogr, osr
import zipfile
import tempfile
import io
import numpy as np
import logging


app = Flask(__name__)


logging.basicConfig(level=logging.DEBUG)


def clip_raster(dem_path, kml_path):
    logging.debug("Clipping the raster with dem_path: %s and kml_path: %s", dem_path, kml_path)
    tmp_dir = create_temp_dir()
    tmp_output_path = os.path.join(tmp_dir, 'tmp_clip.tif')
    if os.path.exists(tmp_output_path):
        os.remove(tmp_output_path)
    subprocess.run(['gdalwarp', '-cutline', kml_path, '-crop_to_cutline', dem_path, tmp_output_path], check=True)
    with open(tmp_output_path, 'rb') as f:
        clipped_data = f.read()
    logging.debug("Clipped raster data length: %d bytes", len(clipped_data))
    return clipped_data, tmp_dir
```

```
def transform_to_feet(clipped_dem_data, tmp_dir):
    logging.debug("Transforming clipped DEM data to feet")
    tmp_input_path = os.path.join(tmp_dir, 'tmp_clip.tif')
    with open(tmp_input_path, 'wb') as tmp_input:
        tmp_input.write(clipped_dem_data)
    input_ds = gdal.Open(tmp_input_path, gdal.GA_Update)
    dem_band = input_ds.GetRasterBand(1)
    dem_array = dem_band.ReadAsArray()

    # Mask the no-data values
    nodata_value = dem_band.GetNoDataValue()
    dem_array = np.ma.masked_equal(dem_array, nodata_value)
    logging.debug("Original DEM array stats - min: %f, max: %f", dem_array.min(),
dem_array.max())

    dem_array_feet = dem_array * 3.28084  # Conversion factor from meters to feet
    logging.debug("Converted DEM array stats - min: %f, max: %f", dem_array_feet.min(),
dem_array_feet.max())

    # Unmask the no-data values
    dem_array_feet = dem_array_feet.filled(nodata_value)

    tmp_output_path = os.path.join(tmp_dir, 'tmp_feet.tif')
    driver = gdal.GetDriverByName('GTiff')
    out_ds = driver.Create(tmp_output_path, input_ds.RasterXSize, input_ds.RasterYSize,
1, gdal.GDT_Float32)
    out_ds.SetGeoTransform(input_ds.GetGeoTransform())
    out_ds.SetProjection(input_ds.GetProjection())
```

```
    out_band = out_ds.GetRasterBand(1)
    out_band.WriteArray(dem_array_feet)
    out_band.SetNoDataValue(nodata_value)

    out_ds.FlushCache()
    out_ds = None  # Close the output dataset

    with open(tmp_output_path, 'rb') as f:
        transformed_data = f.read()
    logging.debug("Transformed DEM data length: %d bytes", len(transformed_data))
    return transformed_data, tmp_dir

def generate_contours(clipped_dem_feet_data, tmp_dir, interval=1):
    tmp_input_path = os.path.join(tmp_dir, 'tmp_feet.tif')
    input_ds = gdal.Open(tmp_input_path, gdal.GA_Update)
    raster_band = input_ds.GetRasterBand(1)
    proj = osr.SpatialReference(wkt=input_ds.GetProjection())
    dem_nan = raster_band.GetNoDataValue()

    tmp_output_path = os.path.join(tmp_dir, 'tmp_contours.shp')
    contour_ds = ogr.GetDriverByName("ESRI
Shapefile").CreateDataSource(tmp_output_path)
    contour_shp = contour_ds.CreateLayer('contour', proj,
geom_type=ogr.wkbLineString25D)
    field_def = ogr.FieldDefn("ID", ogr.OFTInteger)
    contour_shp.CreateField(field_def)
    field_def = ogr.FieldDefn("elev", ogr.OFTReal)
    contour_shp.CreateField(field_def)
```

```
    gdal.ContourGenerate(raster_band, interval, 0, [], 1, dem_nan, contour_shp, 0, 1)
    contour_ds = None  # Close the contour dataset

    with open(tmp_output_path, 'rb') as f:
        contour_data = f.read()
    logging.debug("Generated contour data length: %d bytes", len(contour_data))
    return contour_data, tmp_dir

def convert_shapefile_to_dxf(shapefile_data, tmp_dir):
    tmp_input_path = os.path.join(tmp_dir, 'tmp_contours.shp')
    tmp_output_path = os.path.join(tmp_dir, 'tmp_contours.dxf')
    subprocess.run(['ogr2ogr', '-f', 'DXF', '-zfield', 'elev', tmp_output_path, tmp_input_path],
check=True)
    with open(tmp_output_path, 'rb') as f:
        dxf_data = f.read()
    logging.debug("Converted DXF data length: %d bytes", len(dxf_data))
    return dxf_data, tmp_dir

def raster_to_points(clipped_dem_data, tmp_dir):
    tmp_input_path = os.path.join(tmp_dir, 'tmp_clip.tif')
    input_ds = gdal.Open(tmp_input_path, gdal.GA_Update)
    band = input_ds.GetRasterBand(1)
    nodata = band.GetNoDataValue()
    gt = input_ds.GetGeoTransform()

    tmp_output_path = os.path.join(tmp_dir, 'tmp_points.csv')
    with open(tmp_output_path, 'w', newline='') as tmp_output:
```

```
        tmp_output.write("X,Y,Z\n")
        for y in range(band.YSize):
            for x in range(band.XSize):
                value = band.ReadAsArray(x, y, 1, 1)[0][0]
                if value != nodata:
                    px = gt[0] + x * gt[1] + y * gt[2]
                    py = gt[3] + x * gt[4] + y * gt[5]
                    tmp_output.write(f"{px},{py},{value}\n")

    with open(tmp_output_path, 'rb') as f:
        points_data = f.read()
    logging.debug("Generated points data length: %d bytes")
    return points_data, tmp_dir

def create_temp_dir():
    temp_dir = os.path.join(tempfile.gettempdir(), 'terrain_processing_temp')
    os.makedirs(temp_dir, exist_ok=True)
    return temp_dir

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload():
    logging.debug("Upload route accessed")
    try:
        # Get uploaded files
```

```
dem_file = request.files['dem_file']
kml_file = request.files['kml_file']

# Save uploaded files to a temporary directory
temp_dir = create_temp_dir()
dem_path = os.path.join(temp_dir, dem_file.filename)
kml_path = os.path.join(temp_dir, kml_file.filename)
dem_file.save(dem_path)
kml_file.save(kml_path)

# Process the uploaded files
clipped_dem_data, tmp_dir = clip_raster(dem_path, kml_path)
clipped_dem_feet_data, tmp_dir = transform_to_feet(clipped_dem_data, tmp_dir)
contour_shp_data, tmp_dir = generate_contours(clipped_dem_feet_data, tmp_dir)
dxf_data, tmp_dir = convert_shapefile_to_dxf(contour_shp_data, tmp_dir)
csv_data, tmp_dir = raster_to_points(clipped_dem_data, tmp_dir)

# Create an in-memory zip file
zip_buffer = io.BytesIO()
with zipfile.ZipFile(zip_buffer, 'w') as zipf:
    zipf.writestr("clipped_dem.tif", clipped_dem_data)
    zipf.writestr("clipped_dem_feet.tif", clipped_dem_feet_data)
    zipf.writestr("contours.shp", contour_shp_data)
    zipf.writestr("contours.dxf", dxf_data)
    zipf.writestr("pvsyst_shading_file.csv", csv_data)

zip_buffer.seek(0)
```

```
        # Clean up temporary files
        for file in os.listdir(temp_dir):
            file_path = os.path.join(temp_dir, file)
            if os.path.isfile(file_path):
                os.remove(file_path)
        os.rmdir(temp_dir)

        # Return the zip file as a response
        logging.debug("Returning the generated zip file")
        return send_file(zip_buffer, mimetype='application/zip', as_attachment=True, download_name='output.zip')
    except Exception as e:
        logging.error("Error in upload route: %s", e)
        return "An error occurred during processing", 500

if __name__ == "__main__":
    app.run(debug=True)
```

Working index.html -

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>TERRAIN DATA GENERATOR</title>
    <link href="https://fonts.googleapis.com/css2?family=Avenir+Next+LT+Pro:wght@400;700&display=swap" rel="stylesheet">
    <style>
        body {
            font-family: 'Avenir Next LT Pro', sans-serif;
```

```
        background-image: url("{{ url_for('static',
filename='images/terrain_background.png') }}");
        background-size: cover;
        background-repeat: no-repeat;
        color: #ffffff;
        margin: 0;
        padding: 0;
        display: flex;
        flex-direction: column;
        height: 100vh;
    }
    .header {
        background-color: rgba(0, 0, 0, 0.7);
        padding: 10px;
        text-align: center;
        font-size: 24px;
        font-weight: bold;
        box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
    }
    .container {
        max-width: 600px;
        margin: auto;
        padding: 20px;
        background-color: rgba(0, 176, 240, 0.9);
        border-radius: 10px;
        box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
        margin-top: 10vh;
        position: relative;
```

```
}
h1 {
    text-align: center;
    color: #ffffff;
}
form {
    background-color: rgba(255, 255, 255, 0.9);
    padding: 20px;
    border-radius: 10px;
}
label {
    display: block;
    margin-bottom: 10px;
    color: #000000;
    font-weight: bold;
}
input[type=file] {
    margin-bottom: 15px;
    padding: 10px;
    border: 1px solid #cccccc;
    border-radius: 5px;
    width: 100%;
    background-color: #ffffff;
}
button[type=submit] {
    padding: 10px 20px;
    background-color: #00B0F0;
    color: #ffffff;
```

```
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s;
}
button[type=submit]:hover {
    background-color: #008CBA;
}
.footer {
    margin-top: auto;
    text-align: center;
    padding: 10px;
    background-color: rgba(0, 0, 0, 0.7);
    color: #ffffff;
    font-size: 14px;
}
.logo {
    position: absolute;
    top: 20px;
    right: 20px;
}
.logo img {
    width: 150px;
    height: auto;
}
.progress {
    display: none;
```

```
            margin-top: 20px;
            height: 20px;
            background-color: #f3f3f3;
            border-radius: 5px;
            overflow: hidden;
            position: relative;
        }
        .progress-bar {
            height: 100%;
            width: 0;
            background-color: #4caf50;
            text-align: center;
            color: #ffffff;
            white-space: nowrap;
            transition: width 0.4s ease;
        }
        .message {
            text-align: center;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <div class="header">
        Terrain Data Generator
    </div>
    <div class="container">
        <h1>Upload Your Files</h1>
```

```
    <form id="uploadForm" action="/upload" method="POST"
enctype="multipart/form-data">
      <label for="dem_file">Upload DEM (Raster) File:</label>
      <input type="file" id="dem_file" name="dem_file" accept=".tif,.tiff"><br>

      <label for="kml_file">Upload KML File:</label>
      <input type="file" id="kml_file" name="kml_file" accept=".kml"><br>

      <button type="submit">Process Files</button>
    </form>
    <div class="progress">
      <div class="progress-bar" id="progress-bar">Bringing the terrain to you...</div>
    </div>
    <div class="message" id="message"></div>
  </div>
  <div class="footer">
    &copy; 2024 Terrain Data Generator. All rights reserved.
  </div>
  <div class="logo">
    <img src="{{ url_for('static', filename='images/CIR.jpg') }}" alt="CIR Logo">
  </div>
  <script>
    document.getElementById('uploadForm').addEventListener('submit',
function(event) {
      event.preventDefault();
      var formData = new FormData(this);
      var xhr = new XMLHttpRequest();
      var progressBar = document.getElementById('progress-bar');
      var progressContainer = document.querySelector('.progress');
```

```
var message = document.getElementById('message');

xhr.open('POST', '/upload', true);
xhr.responseType = 'blob';

xhr.upload.addEventListener('progress', function(e) {
  if (e.lengthComputable) {
    var percentComplete = (e.loaded / e.total) * 100;
    progressBar.style.width = percentComplete + '%';
    if (percentComplete < 100) {
      progressContainer.style.display = 'block';
    }
  }
});

xhr.addEventListener('load', function() {
  if (xhr.status === 200) {
    message.textContent = 'Files processed successfully!';
    message.style.color = 'green';
    // Trigger file download
    var blob = xhr.response;
    var link = document.createElement('a');
    link.href = window.URL.createObjectURL(blob);
    link.download = 'output.zip';
    link.click();
  } else {
    message.textContent = 'Error processing files.';
    message.style.color = 'red';
```

```
            }
            progressContainer.style.display = 'none';
        });

        xhr.send(formData);
    });
  </script>
</body>
</html>
```