```
import pandas as pd
import numpy as np
import os
import pickle
import sqlite3
from flask import Flask, redirect, url_for, render_template, request,
send_from_directory
import logging
import datetime

logging.basicConfig(filename="log.log",
                    format='%(asctime)s %(message)s',
                    filemode='a')

# Creating an object
logger = logging.getLogger()

# Setting the threshold of logger to DEBUG
logger.setLevel(logging.DEBUG)

class controller():
    def __init__(self):
        self.building_id_list = list(range(0,1449))
        self.year_upper_limit = 2018
        self.year_lower_limit = 2016
        self.service = service()

    def predict(self, building_id, year):
        try:
            assert isinstance(building_id, int), 'building id should be integer'
            assert building_id in self.building_id_list, 'please provide valid
building id'
            assert isinstance(year, int), 'year should be integer'
            assert (year >= self.year_lower_limit) & (year <=
self.year_upper_limit),\
                   f'year should within {(self.year_lower_limit,
self.year_upper_limit)}'
        except AssertionError as e:
            return e
        output = self.service.predict(building_id, year)
        return output

class dao():
    def __init__(self):
        self.conn = sqlite3.connect('data/input_data/ashrae.db',
check_same_thread=False)

    def get_meter_list(self, building_id):
        query = """
                SELECT meter FROM building_meter_map
                WHERE building_id==?;
                """
        cursor = self.conn.execute(query, (building_id,))
        self.conn.commit()
        return (cursor.fetchall())

    def get_preprocessed_data(self, building_id, year, meter, config):
        logger.debug("Fetching processed data...")
        feature = config['feature']
        feature_str = config['feature_str']
        query = """
                    SELECT
                        {}
                    FROM building_metadata
                    INNER JOIN weather_data ON
building_metadata.site_id==weather_data.site_id
                    WHERE building_metadata.building_id==? AND strftime('%Y',
weather_data.timestamp)==?
                    ORDER BY weather_data.timestamp;
                """.format(feature_str)
        cursor = self.conn.execute(query, (building_id,str(year)))
        df = pd.DataFrame(cursor.fetchall(), columns=feature)
        self.conn.commit()
        logger.debug("Completed!")
        return df

class service():
    def __init__(self):
        self.model_0_1 = pickle.load(open('model/model_0_1.pkl', 'rb'))
        self.model_0_1.fitted_ = True
        self.model_0_2 = pickle.load(open('model/model_0_2.pkl', 'rb'))
        self.model_0_2.fitted_ = True
        self.model_1 = pickle.load(open('model/model_1.pkl', 'rb'))
        self.model_1.fitted_ = True
        self.model_2 = pickle.load(open('model/model_2.pkl', 'rb'))
        self.model_2.fitted_ = True
        self.model_3 = pickle.load(open('model/model_3.pkl', 'rb'))
        self.model_3.fitted_ = True
        self.model_0_config = pickle.load(open('model/model_0_config.pkl', 'rb'))
        self.model_1_config = pickle.load(open('model/model_1_config.pkl', 'rb'))
        self.model_2_config = pickle.load(open('model/model_2_config.pkl', 'rb'))
        self.model_3_config = pickle.load(open('model/model_3_config.pkl', 'rb'))
        self.building_sample_1 = pickle.load(open('model/building_sample_1.pkl',
'rb'))
        self.building_sample_2 = pickle.load(open('model/building_sample_2.pkl',
'rb'))
        self.output_file = None
        self.dao = dao()

    def post_processing(self, x, year):
        x = np.clip(np.expm1(x), 0, None)
        index = pd.date_range(f'01-01-{year}', f'01-01-{year+1}',
                              freq='H', inclusive='left')
        index = index.format(formatter=lambda x: x.strftime('%d-%m-%Y %H:%M:%S'))
        x = list(zip(index, x))
        self.output_file = datetime.datetime.now().strftime('%d-%m-%Y_%H-%M-%S')
+ '.csv'
        pd.DataFrame(x, columns=['timestamp', 'meter_reading'])\
          .to_csv(os.path.join('main', 'output', self.output_file), index=False)
        return x

    def predict(self, building_id, year):
        #load data
        logger.debug("Fetching meter list...")
        meter_list = self.dao.get_meter_list(building_id)
        logger.debug(f"Meter list: {meter_list}")
        output = dict()
        for meter_type in meter_list:
            meter = meter_type[0]
            logger.debug(f"Predicting meter reading for meter type: {meter}")
            #get_processed_data
            if meter == 0:
                preprocessed_data = self.dao.get_preprocessed_data(building_id,
year, meter,

self.model_0_config)
                if building_id in self.building_sample_1:
                    output[meter] =
self.post_processing(self.model_0_1.predict(preprocessed_data),
                                                                 year)
                else:
                    output[meter] =
self.post_processing(self.model_0_2.predict(preprocessed_data),
                                                                 year)
            elif meter == 1:
                preprocessed_data = self.dao.get_preprocessed_data(building_id,
year, meter,

self.model_1_config)
                output[meter] =
self.post_processing(self.model_1.predict(preprocessed_data),
                                                             year)
            elif meter == 2:
                preprocessed_data = self.dao.get_preprocessed_data(building_id,
year, meter,

self.model_2_config)
                output[meter] =
self.post_processing(self.model_2.predict(preprocessed_data),
                                                             year)
            elif meter == 3:
                preprocessed_data = self.dao.get_preprocessed_data(building_id,
year, meter,

self.model_3_config)
                output[meter] =
self.post_processing(self.model_3.predict(preprocessed_data),
                                                             year)
        logger.debug("Return output data...")
        return output, self.output_file

app = Flask(__name__) #creating the Flask class object
```

```
input_obj = controller()

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

@app.route('/predict', methods=['GET'])
def predict(input_obj=input_obj):
    building_id = request.args['building_id']
    year = request.args['year']
    output, output_file = input_obj.predict(building_id=int(building_id),
year=int(year))
    logger.debug(f"Type of output data: {type(output)}")
    return render_template('predict.html', output=output,
output_file=output_file)

@app.route('/download', methods=['POST'])
def download():
    output_file = request.form['output_file']
    return send_from_directory(os.path.join(os.getcwd(), 'main', 'output'),
output_file,
                               as_attachment=True)

if __name__ =='__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)
```