메인 프로그램
KKBoobyTrap.py

KKBoobyTrap.py

```
from flask import Flask, render_template, send_from_directory, Response, send_file,
request, redirect, url_for
from flask_socketio import SocketIO
from pathlib import Path
from capture import capture_and_save
from piwebcamera import PiWebCamera
import argparse, logging, logging.config, conf
import os
from urllib.parse import parse_qs
from power import PowerStatus




app = Flask(__name__)
socketio = SocketIO(app)
archive_path = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'archive')

power = PowerStatus()

logging.config.dictConfig(conf.dictConfig)
logger = logging.getLogger(__name__)

picamera = PiWebCamera(video_source=0, do_display=False)
picamera.start()


@app.after_request
def add_header(r):
    """
    Add headers to both force latest IE rendering or Chrome Frame,
    and also to cache the rendered page for 10 minutes
    """
    r.headers["Cache-Control"] = "no-cache, no-store, must-revalidate"
    r.headers["Pragma"] = "no-cache"
    r.headers["Expires"] = "0"
    r.headers["Cache-Control"] = "public, max-age=0"
    return r
```

```
@app.route("/")
@app.route("/index.html")
def index():
    logger.debug("Requested /")
    return render_template("index.html")

@app.route("/video/last_video")
def last_video():
    logger.debug("Requested last video")
    for filename in sorted(os.listdir(archive_path), reverse=True):
        if not filename.startswith('.'):
            type = get_type(filename)
            if type == "video":
                return send_from_directory(archive_path, filename)


''' ##### Achive File Section ##### '''
@app.route('/archive')
def archive():
        return render_template('archive.html')

def get_type(filename):
        name, extension = os.path.splitext(filename)
        return 'video' if extension == '.mp4' else 'audio' if extension == '.wav' else
'audio' if extension == '.mp3' else 'photo'

@app.route('/archive/<string:filename>')
def archive_item(filename):
        name, extension = os.path.splitext(filename)
        type = get_type(filename)
        return render_template('record.html', filename=filename, type=type)

@app.route('/archive/delete/<string:filename>')
def archive_delete(filename):
        os.remove(archive_path + "/" + filename)
        return redirect(url_for('archive'))

@app.route('/archive/play/<string:filename>')
def archive_play(filename):
        return send_file('archive/' + filename)

def get_records():
        records = []
        for filename in sorted(os.listdir(archive_path), reverse=True):
                if not filename.startswith('.'):
                        type = get_type(filename)
                        size = byte_to_mb(os.path.getsize(archive_path + "/" +
filename))
                        record = {"filename": filename, 'size': size, 'type': type}
```

```
                        records.append(record)
        return records

def byte_to_mb(byte):
        mb = "{:.2f}".format(byte / 1024 / 1024)
        return str(mb) + " MB"

app.jinja_env.globals.update(get_records=get_records)

''' ##### Achive File Section ##### '''

def genpi(picamera):
    logger.debug("Starting PI stream")
    while True:
        frame = picamera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/png\r\n\r\n' + frame + b'\r\n')

@app.route("/pistream")
def pistream_page():
    logger.debug("Requested stream page")
    return render_template("pistream.html")


@app.route("/video_pi_feed")
def video_pi_feed():
    return Response(genpi(picamera),
        mimetype="multipart/x-mixed-replace; boundary=frame")

@app.route("/picapture")
def picapture():
    logger.debug("Requested PICAM capture")
    im = picamera.get_captureFrame()
    capture_and_save(im)
    return render_template("send_to_init.html")

@app.route("/temperature")
def temperature():
    content = os.popen("vcgencmd measure_temp").readline()
    content = content.replace("temp=", "")
    powerstatus = power.getPowerStatus()
    return Response(content+"["+powerstatus+"]", mimetype='text/xml')

@app.route("/favorit.ico")
def favorit_ico():
    logger.debug("Requested favorit.ico image")
    filename = "favorit.ico"
    return send_file(filename)
```

```
if __name__=="__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-p','--port',type=int,default=8081, help="Running port")
    parser.add_argument("-H","--host",type=str,default='0.0.0.0', help="Address to
broadcast")
    args = parser.parse_args()
    logger.debug("Starting server")
    socketio.run(app, log_output=True, host='0.0.0.0', port=8081, debug=True,
use_reloader=False)
```

Revision #1
★Created 2024-10-02 23:16:18 UTC by Hyeon Su Ryu
Updated 2024-10-02 23:31:09 UTC by Hyeon Su Ryu