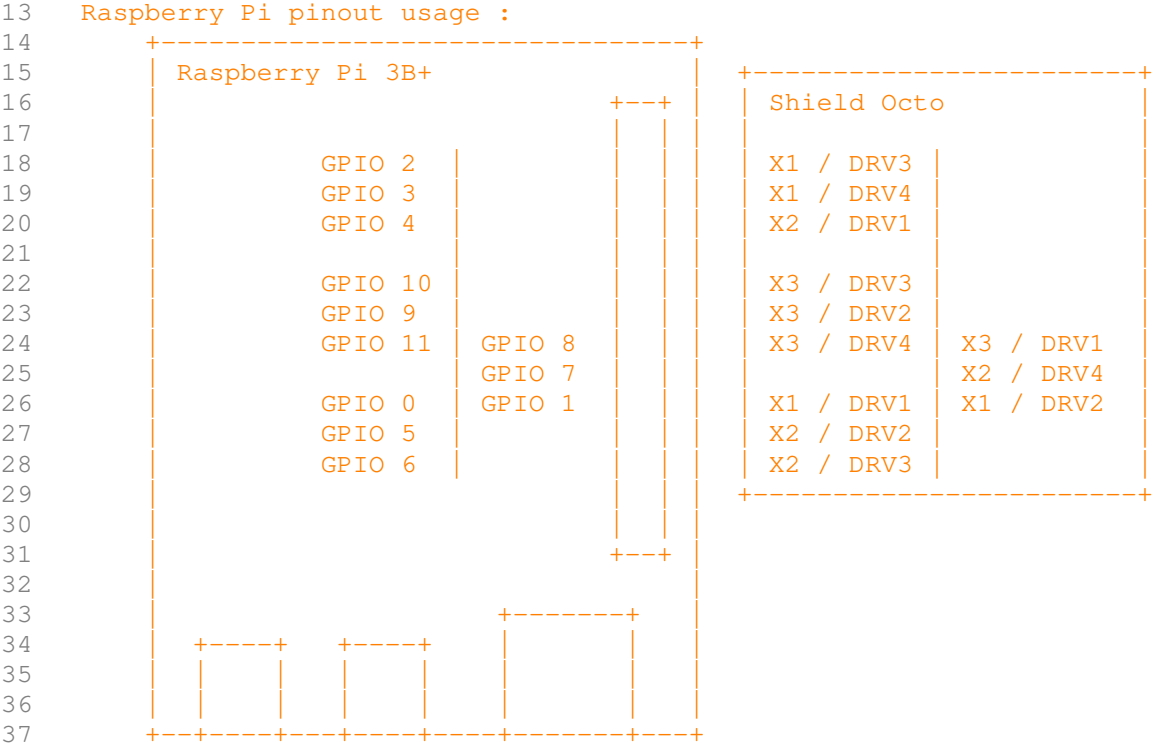
```
"""main.py | Robin Forestier | 2021 / 2022

Goal : Create a user interface to control the light of the ELO's workshop.

Explanation :   I use Flask for create a Web server.
                A shield is plug on the Raspberry Pi (server) with opto-isolator to
                ABB module with the GPIO.

TPI : For my IPT, I have to create a settings page controlling the time range where
the cameras have the right
      to request a modification of the lights.
      I have to create a manual mode, to modify the date and time. I also need to
      see the status of the cameras and be
      able to change the passwords.

Raspberry Pi pinout usage :
```




```
GitLab : http://172.16.32.230/Forestier/controle-des-lumieres-knx
         http://172.16.32.230/Forestier/tpi_forestier_gestion_lumiere_knx
XWiki :

    https://xwiki.serverelo.org/xwiki/bin/view/Centre%20de%20Formation%20ELO/Projets/Controle%20des%20lumières%20KNX/

    https://xwiki.serverelo.org/xwiki/bin/view/Centre%20de%20Formation%20ELO/Travaux-diplome/
    TPI_Forestier_Gestion_Lumiere_KNX/Documentation/
"""

# Importing the necessary modules to run the code.
import logging
import os
import pickle
import subprocess
import threading
import time
from datetime import datetime, timedelta
import numpy as np

# Import flask
from flask import Flask, render_template, request, session, url_for, redirect,
flash, abort
from flask_session import Session
from flask_sqlalchemy import SQLAlchemy
from werkzeug.exceptions import HTTPException

# Creating a class called Table.
from table.table import Table
```

```
65  # Creating a class called Table2 that inherits from Table.
66  from table.table import Table2
67
68  # The above code is importing the RPi.GPIO module and setting it to use the BCM pin
    numbering scheme.
69  try:
70      import RPi.GPIO as GPIO
71
72      GPIO.setmode(GPIO.BCM)
73      GPIO.setwarnings(False)
74
75      # GPIO 0 to 11 in output mode at 0
76      for i in range(12):
77          GPIO.setup(i, GPIO.OUT, initial=GPIO.LOW)
78
79      # it's a raspberry pi !
80      rpi = True
81  except ImportError:
82      # it's not a raspberry pi !
83      rpi = False
84
85  # The above code is creating a log file called log.log and setting the format of the
    log file to the time, the level of
86  # the log message and the message itself.
87  logging.basicConfig(filename="log.log", format='%(asctime)s - %(levelname)s -
    %(message)s', level=logging.INFO)
88
89  # This is creating a new Flask object.
90  app = Flask(__name__)
91
92  # This is the configuration for the application.
93  app.config.update(
94      SECRET_KEY='somesecretkeythatonlyishouldknow',
95      SQLALCHEMY_DATABASE_URI="sqlite:///db.sqlite3",
96      SQLALCHEMY_TRACK_MODIFICATIONS=False,
97      SESSION_TYPE='sqlalchemy',
98      SESSION_COOKIE_NAME='MyBeautifulCookies',
99      SESSION_COOKIE_SAMESITE='Strict',
100 )
101
102 # This is creating a database object that will be used to access the database.
103 db = SQLAlchemy(app)
104 # This is telling Flask to use SQLAlchemy as the database.
105 app.config['SESSION_SQLALCHEMY'] = db
106 # This is creating a session object that will be used to store and retrieve session
    data from the user's browser.
107 sess = Session(app)
108 db.create_all()
109
110 # Ip of the server (for the local user) use to skip authentication.
111 # Checking if the IP address of the machine that is running the script is in the
    list of IP addresses that are
112 # automatically login.
113 authorize_ip = ["localhost", "127.0.0.1", "172.16.32.133"]
114
115 # Value of the buttons and the colors for the map light, each element is a string
    "off".
116 buttonSts_p1 = ["off"] * 8
117 buttonSts_p2 = ["off"] * 8
118 color = ["#333333"] * 8
119 # possible warning (like temp alert).
120 warning = ""
121
122
123 class User:
124     """User : create different user for the login"""
125
126     def __init__(self, id, username, password):
127         """
128         The __init__ function is a constructor method that is called when an object
            is created
129
```

```
130             :param id: The id of the user
131             :type id: int
132             :param username: The username of the user
133             :type username: str
134             :param password: The password of the user
135             :type password: str
136             """
137 
138             self.id = id
139             self.username = username
140             self.password = password
141 
142 
143     # For the page: login 2
144     mot_de_pass = []
145 
146     mode_manuel = 0
147     """mode_manuel (int) is set if a user activate it on the settings page. (0 ->
        disable, 1 -> enable)"""
148 
149     mode_heure_manuel = [0, 0, 0]
150     """mode_heure_manuel (list) is use to store:
151     1. (boolean) set if mode heure manuel activated.
152     2. (int) current day. 0 to 6 -> 0 is monday.
153     3. () delta time.
154     """
155 
156     cam_connect = [[False, "", "b8:27:eb:26:5a:95", 0], [False, "", "b8:27:eb:ce:4f:78",
        0]]
157     """cam_connect (list) of the authorized camera.
158     To check the authorisation, I used the mac adresse.
159     1. (str) True if connected.
160     2. (str) Ip adresse.
161     3. (str) Mac adresse.
162     """
163 
164     # This is a timedelta object. It is a duration expressing the difference between two
        dates.
165     TURNOFFTIME = timedelta(minutes=1)
166     """TURNOFFTIME (timedelta) is the time duration before the light turn off."""
167 
168     cam_send_turn_off = []
169 
170     number_of_people = 0
171     """number_of_people (int) is the number of people in the room."""
172 
173     # Try to open password.pickle. this file contains the 3 passwords.
174     try:
175         # This code is opening a pickle file and reading it.
176         with open('password.pickle', 'rb') as f:
177             # This code is loading the pickle file.
178             password = pickle.load(f)
179     except OSError:
180         # This code is creating a pickle file that contains a list of passwords.
181         with open('password.pickle', 'wb') as f:
182             password = ["elo", "admin", "1234"]
183             pickle.dump(password, f, pickle.HIGHEST_PROTOCOL)
184 
185     users = []
186     """
187     user (list) is a list of User objects.
188         elo - basic user / can only modifie light
189         admin - all access (settings access)
190         local - only used for the raspberry why the touchscreen (server) / can access to
            settings with login page 2
191                 / don't have to login to control the light.
192     """
193     # Create the users.
194     users.append(User(id=1, username='elo', password=password[0]))
195     users.append(User(id=2, username='admin', password=password[1]))
196     users.append(User(id=3, username='local', password=password[2]))
197 
```

```
198
199 def gpio_modif():
200     """gpio_modif : modif all gpio"""
201     if rpi:
202         for i in range(8):
203             if buttonSts_p1[i] == "off":
204                 # OFF
205                 GPIO.output(i, 0)
206             else:
207                 # ON
208                 GPIO.output(i, 1)
209                 # time.sleep(1)     #security time for fuses
210     else:
211         pass
212
213
214 def get_time():
215     """Get the current time and day
216
217     :return: A tuple containing the current time and the current day of the week.
218     :rtype: tuple
219     """
220
221     # The above code is creating a global variable called mode_heure_manuel.
222     global mode_heure_manuel
223
224     # This code is checking if the first element of the list mode_heure_manuel is
    equal to 1.
225     # If the first element is 1 the mode hour manuel is set.
226     # The current hour become current hour + delta time.
227     if mode_heure_manuel[0] == 1:
228         t = datetime.now() + mode_heure_manuel[2]
229         current_time = t.strftime("%H:%M")
230         current_day = mode_heure_manuel[1]
231     else:
232         # Creating a time object.
233         t = time.localtime()
234         # The above code is creating a string of the current time in hours and
        minutes (10:10).
235         current_time = time.strftime("%H:%M", t)
236         # This code is checking the current day of the week and then using that to
        determine
237         # which day of the week it is (0 is monday, 6 is sunday).
238         current_day = datetime.today().weekday()
239
240     return current_time, current_day
241
242
243 def myping(host):
244     """myping : check if a host is up
245
246     :param host: The host to check.
247     :type host: str
248     :return: True if the host is up, False if not.
249     :rtype: bool
250     """
251
252     # Checking if the host is empty. If it is empty, it will return False.
253     if host == "":
254         return False
255
256     if rpi:
257         # This code is using the subprocess module to run the ping command on a host.
258         response = str(subprocess.check_output(["ping", "-c", "1", host]))
259     else:
260         # This code is using the os.popen() function to ping the host.
261         response = os.popen("ping -n 1 " + host).read()
262
263     # The above code is checking to see if the ping was successful.
264     if "0 received" in response:
265         return False
266     else:
```

```
267             return True
268
269
270     def getmacadd(host):
271         """getmacadd : get the mac adresse of a host
272
273         :param host: The host to check.
274         :type host: str
275         :return: The mac adresse of the host.
276         :rtype: str
277         """
278
279         # This code is using the ping function to ping the host.
280         rep = myping(host)
281         if rep:
282             # The above code is using the subprocess module to run the arp command on
                the host machine. The arp command is
283             # used to display the ARP (Address Resolution Protocol) table. The arp table
                contains the hardware (MAC) address
284             # for each IP address on the local area network.
285             answer = str(subprocess.check_output(["arp", "-a", host]))
286
287             if "aucune" in answer:
288                 return False
289             else:
290                 # Splitting the answer into a list of words.
291                 answer = answer.split(" ")
292                 # Return only the MAC address
293                 return answer[3]
294         else:
295             return False
296
297
298     @app.errorhandler(HTTPException)
299     def handle_exception(e):
300         """handle_exception : handle http error
301
302         :param e: The http error.
303         :type e: HTTPException
304         :return: The HTML error page.
305         :rtype: render_template
306         """
307
308         e = str(e)  # 404 not ...  :  The request URL ...
309         code = e.split(":")  # 404 Not Found
310
311         # This is a function that takes in an error code and returns the error page for
            that error.
312         return render_template('404.html', error=code[1], title=code[0]), code[0][0:3]
313
314
315     @app.route("/login", methods=['POST', 'GET'])
316     @app.route("/", methods=['POST', 'GET'])
317     def login():
318         """login : login page
319         Receive from POST : username / password
320         We first check if username existe and after password.
321         If it's a bad username or pass we flash the error.
322
323         :return: The login page.
324         :rtype: render_template
325         """
326
327         # Creating a variable called current_time that is equal to the current time.
328         current_time, _ = get_time()
329         # This code is getting the user's IP address and returning it.
330         ip = request.environ.get('HTTP_X_REAL_IP', request.remote_addr)
331
332         # check for authorized ip
333         for i in authorize_ip:
334             if ip == i:
335                 logging.info("create local user" + str(ip))
```

```
336                 session['user_id'] = 3
337                 return redirect(url_for('page1'))
338 
339         # This code is checking to see if the user is logged in.
340         # If they are, then they will be directed to the page 1.
341         if session.get("user_id") is not None:
342             return redirect(url_for('page1'))
343 
344         # This code is checking to see if the request method is a POST. If it is, it
        will run the code below.
345         if request.method == 'POST':
346             # This code is removing the user_id from the session.
347             session.pop('user_id', None)
348 
349             username = request.form['username']
350             password = request.form['password']
351 
352             try:
353                 # This code is using a list comprehension to search for a user with a
                    given username.
354                 user = [x for x in users if x.username == username][0]
355                 # The above code is checking if the user exists and if the password is
                    correct.
356                 if user and user.password == password:
357                     # Login accepted
358                     session['user_id'] = user.id
359                     logging.info("New login username : " + username + " ip : " + str(ip))
360                     return redirect(url_for('page1'))
361                 elif user.password != password:
362                     # bad password, logging it and flash the error
363                     logging.warning("bad password : " + username + " ip : " + str(ip))
364                     flash("Bad password")
365 
366                 return redirect(url_for('login'))
367 
368             except IndexError:
369                 # bad username, logging it and flash the error
370                 logging.warning("bad username : " + username + " ip : " + str(ip))
371                 flash("Bad username")
372                 return redirect(url_for('login'))
373 
374         return render_template("login.html", time=current_time, warning=warning)
375 
376 
377     @app.route("/login2", methods=['POST', 'GET'])
378     def login2():
379         """The login2 function is the second page of the login page.
380         It is only for the touch screen. It is called when the user want to go in the
            settings page.
381         It checks if the user has entered the correct password. If the user has entered
            the correct
382         password, the user is redirected to the settings page.
383 
384         :return: The login2.html template is being returned.
385         :rtype: render_template
386         """
387 
388         # Creating a function that will return the current time.
389         current_time, _ = get_time()
390         # This code is getting the user's IP address and returning it.
391         ip = request.environ.get('HTTP_X_REAL_IP', request.remote_addr)
392 
393         # The above code is checking if the IP address of the user is in the list of
            authorized IP addresses.
394         if ip not in authorize_ip:
395             return redirect(url_for('login'))
396 
397         global mot_de_pass
398 
399         # This code is checking to see if the request method is a POST. If it is, it
            will run the code below.
400         if request.method == 'POST':
```

```
401             button_click = request.form.get('btn')
402 
403             # Checking if the button click is a number between 0 and 9.
404             if button_click in ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]:
405                 # The above code is adding in the list all the buttons that are clicked.
406                 mot_de_pass.append(button_click)
407 
408             # Checking if the button click is 10 (Effacer)
409             if button_click == "10":
410                 # Clear the list
411                 mot_de_pass = []
412 
413             # Checking if the button click is 11 (Submit)
414             if button_click == "11":
415                 # The code above is taking the list of characters and joining them
                    together to form a string.
416                 mot_de_pass = "".join(mot_de_pass)
417                 # This code is using a list comprehension to find the user named "local"
                    and then returning that user.
418                 user = [x for x in users if x.username == "local"][0]
419                 # The above code is checking if the password entered by the user is the
                    same as the password stored in the
420                 # database.
421                 if mot_de_pass == user.password:
422                     mot_de_pass = []
423                     return redirect(url_for('settings', authorized=1))
424 
425                 mot_de_pass = []
426 
427         return render_template('login2.html', mdp=mot_de_pass, time=current_time,
            warning=warning)
428 
429 
430 @app.route("/page1", methods=['POST', 'GET'])
431 def page1():
432         """This function is used to display the page 1
433         It is the first page of the application.
434 
435         :return: The html page is being returned.
436         :rtype: render_template
437         """
438 
439         # Creating a function that will return the current time.
440         current_time, _ = get_time()
441 
442         # This code is checking to see if the user is logged in. If they are not logged
            in, then they are redirected to the
443         # login page.
444         if session.get("user_id") is None:
445             return redirect(url_for('login'))
446 
447         # This code is checking if all the elements in the list are equal to "on".
448         if all(elem == "on" for elem in buttonSts_p1):
449             # Button "all on" in page 2 is turn to "on"
450             buttonSts_p2[0] = "on"
451         else:
452             buttonSts_p2[0] = "off"
453 
454         # This code is checking to see if the request method is a POST. If it is, it
            will run the code below.
455         if request.method == 'POST':
456             # Checking if the button has been pressed.
457             if request.form['button_p1'] == '1':
458                 if buttonSts_p1[0] == "on":
459                     buttonSts_p1[0] = "off"
460                     color[0] = "#333333"
461                 else:
462                     buttonSts_p1[0] = "on"
463                     color[0] = "#FFFFFF"
464             elif request.form['button_p1'] == '2':
465                 if buttonSts_p1[1] == "on":
466                     buttonSts_p1[1] = "off"
```

```
467                 color[1] = "#333333"
468             else:
469                 buttonSts_p1[1] = "on"
470                 color[1] = "#FFFFFF"
471         elif request.form['button_p1'] == '3':
472             if buttonSts_p1[2] == "on":
473                 buttonSts_p1[2] = "off"
474                 color[2] = "#333333"
475             else:
476                 buttonSts_p1[2] = "on"
477                 color[2] = "#FFFFFF"
478         elif request.form['button_p1'] == '4':
479             if buttonSts_p1[3] == "on":
480                 buttonSts_p1[3] = "off"
481                 color[3] = "#333333"
482             else:
483                 buttonSts_p1[3] = "on"
484                 color[3] = "#FFFFFF"
485         elif request.form['button_p1'] == '5':
486             if buttonSts_p1[4] == "on":
487                 buttonSts_p1[4] = "off"
488                 color[4] = "#333333"
489             else:
490                 buttonSts_p1[4] = "on"
491                 color[4] = "#FFFFFF"
492         elif request.form['button_p1'] == '6':
493             if buttonSts_p1[5] == "on":
494                 buttonSts_p1[5] = "off"
495                 color[5] = "#333333"
496             else:
497                 buttonSts_p1[5] = "on"
498                 color[5] = "#FFFFFF"
499         elif request.form['button_p1'] == '7':
500             if buttonSts_p1[6] == "on":
501                 buttonSts_p1[6] = "off"
502                 color[6] = "#333333"
503             else:
504                 buttonSts_p1[6] = "on"
505                 color[6] = "#FFFFFF"
506         elif request.form['button_p1'] == '8':
507             if buttonSts_p1[7] == "on":
508                 buttonSts_p1[7] = "off"
509                 color[7] = "#333333"
510             else:
511                 buttonSts_p1[7] = "on"
512                 color[7] = "#FFFFFF"
513         elif request.form['button_p1'] == 'page_2':
514             return redirect(url_for('page2'))
515         else:
516             pass
517 
518         gpio_modif()
519 
520     return render_template('page1.html', button=buttonSts_p1, color=color,
    time=current_time, warning=warning)
521 
522 
523 @app.route("/page2", methods=['POST', 'GET'])
524 def page2():
525     """This function is used to display the page 2
526     It is the second page of the application.
527 
528     :return: The page is being rendered.
529     :rtype: render_template
530     """
531 
532     # Creating a variable called current_time that is equal to the current time.
533     current_time, _ = get_time()
534 
535     # This code is checking to see if the user is logged in. If they are not logged
    in, then they are redirected to the
536     # login page.
```

```
537         if session.get("user_id") is None:
538             return redirect(url_for('login'))
539 
540         # This code is checking to see if the request method is a POST. If it is, it
            will run the code below.
541         if request.method == 'POST':
542             # All on
543             if request.form['button_p1'] == '1':
544                 buttonSts_p2[0] = "on"
545                 for i in range(8):
546                     buttonSts_p1[i] = "on"
547                     color[i] = "#FFFFFF"
548             # All off
549             elif request.form['button_p1'] == '2':
550                 buttonSts_p2[1] = "off"
551                 for i in range(8):
552                     buttonSts_p1[i] = "off"
553                     buttonSts_p2[i] = "off"
554                     color[i] = "#333333"
555             # 1 / 2
556             elif request.form['button_p1'] == '3':
557                 buttonSts_p2[0] = "off"
558                 buttonSts_p2[1] = "off"
559 
560                 for i in range(0, 8, 2):
561                     color[i] = "#FFFFFF"
562                     color[i + 1] = "#333333"
563 
564                     buttonSts_p1[i] = "on"
565                     buttonSts_p1[i + 1] = "off"
566             # Left
567             elif request.form['button_p1'] == '4':
568                 buttonSts_p2[0] = "off"
569                 buttonSts_p2[1] = "off"
570 
571                 for i in range(0, 8):
572                     if i < 3:
573                         color[i] = "#FFFFFF"
574                         buttonSts_p1[i] = "on"
575                     else:
576                         color[i] = "#333333"
577                         buttonSts_p1[i] = "off"
578             # Right
579             elif request.form['button_p1'] == '5':
580                 buttonSts_p2[0] = "off"
581                 buttonSts_p2[1] = "off"
582 
583                 for i in range(0, 8):
584                     if i > 3 and i != 7:
585                         color[i] = "#FFFFFF"
586                         buttonSts_p1[i] = "on"
587                     else:
588                         color[i] = "#333333"
589                         buttonSts_p1[i] = "off"
590 
591             elif request.form['button_p1'] == 'page_1':
592                 return redirect(url_for('page1'))
593             else:
594                 pass
595 
596             gpio_modif()
597 
598         return render_template('page2.html', button=buttonSts_p2, color=color,
            time=current_time, warning=warning)
599 
600 
601     @app.route("/settings", methods=['POST', 'GET'])
602     def settings():
603         """The settings page is used to change the settings of the application.
604 
605         :return: The settings.html template is being returned.
606         :rtype: render_template
```

```
607         """
608
609         id = session.get("user_id")
610         authorized = request.args.get('authorized')
611
612         # Get the id for the session.
613         # If == None --> go to th page login.
614         # If == 3 (local) --> go to the page login 2 or settings if password ok
615         # If == 1 (elo) --> remove the id / error 418
616         if id is None:
617             return redirect(url_for('login'))
618         elif id == 3 and authorized != "1":
619             return redirect(url_for('login2'))
620         elif id == 1:
621             session.pop("user_id")
622             abort(418)
623
624         # Calling a function that will return the current time.
625         current_time, _ = get_time()
626         # This code is getting the user's IP address and returning it.
627         ip = request.environ.get('HTTP_X_REAL_IP', request.remote_addr)
628
629         global mode_manuel
630         global mode_heure_manuel
631         global cam_connect
632
633         t = Table()
634         t2 = Table2()
635
636         # This code is checking to see if the request method is a POST. If it is, it
            will run the code under it.
637         if request.method == 'POST':
638
639             # This code is getting the data from the form and putting it into variables.
640             btn_tbl_1 = request.form.get('btn_tbl_1')
641             btn_tbl_2 = request.form.get('btn_tbl_2')
642             nouvelle_heure = request.form.get('time')
643             supprimer = request.form.get('supp')
644
645             manuel = request.form.get('manuel')
646             heure_man = request.form.get('heure_manuel')
647             heure_man_btn = request.form.get('heure_manuel_btn')
648
649             modif_mdp = request.form.get('modif_mdp')
650             password = request.form.get('password')
651             new_password = request.form.get('new_password')
652
653             if btn_tbl_1:
654                 # The above code is converting the number of table's cell into a row and
                    column number.
655                 btn_click = float(btn_tbl_1)
656                 row = int(btn_click)
657                 column = round((btn_click - row) * 10)
658
659                 # This code is calling "modif_table".
660                 t.modif_table(row, column)
661                 t.set_table()
662
663             if btn_tbl_2:
664                 # The above code is converting the number of table's cell into a row and
                    column number.
665                 btn_click = float(btn_tbl_2)
666                 row = int(btn_click)
667                 column = round((btn_click - row) * 10)
668
669                 # This code is calling "modif_table" for table 2.
670                 t2.modif_table(row, column)
671                 t2.set_table()
672
673             if nouvelle_heure:
674                 # Adding a custom line to the table.
675                 t.add_custom_line(nouvelle_heure)
```

```
676                 t.set_table()
677                 t.set_colonne_heure()
678 
679             if supprimer:
680                 # Deleting the custom line in the table 1.
681                 t.del_custom_line(supprimer)
682 
683             if manuel:
684                 # This code is inverting the manual mode.
685                 mode_manuel = 1 - int(manuel)
686 
687             if heure_man_btn:
688                 # This code is inverting the manual mode.
689                 mode_heure_manuel[0] = 1 - int(heure_man_btn)
690 
691                 if mode_heure_manuel[0] == 1:
692                     # Converting the time from the website into a datetime object.
693                     # Then it is subtracting the current time from the time from the
                        website.
694                     # Then it is converting the timedelta object into a string.
695                     x = datetime.fromisoformat(heure_man)
696                     diff = x - datetime.strptime(datetime.now().strftime("%H:%M"),
                        "%H:%M")
697                     diff = timedelta(seconds=diff.seconds)
698 
699                     # Store the new day (0 is monday).
700                     mode_heure_manuel[1] = x.weekday()
701                     # Store the delta time
702                     mode_heure_manuel[2] = diff
703 
704                 else:
705                     # The above code is setting mode_heure_manuel to 0.
706                     mode_heure_manuel = [0, 0, 0]
707 
708                 # Creating a function that will be used to get the current time.
709                 current_time, _ = get_time()
710 
711             if modif_mdp:
712                 # This code is searching the list of users for the user with the same
                    username as the one who is trying to
713                 # change their password.
714                 user = [x for x in users if x.username == modif_mdp][0]
715                 # The above code is checking if the user exists and if the password is
                    correct.
716                 if user and user.password == password:
717                     # Checking if the user has selected the local option and if the new
                        password is not only number.
718                     if modif_mdp == "local" and not new_password.isdigit():
719                         flash("New password for local user can only contains numbers")
720                     else:
721                         logging.info("Modification password : " + modif_mdp + " new : "
                            + new_password + " ip : " + str(ip))
722                         user.password = new_password
723 
724                         # Save the new password on the pickle file
725                         with open('password.pickle', 'rb') as f:
726                             passw = pickle.load(f)
727                         with open('password.pickle', 'wb') as f:
728                             passw[int(user.id) - 1] = new_password
729                             pickle.dump(passw, f, pickle.HIGHEST_PROTOCOL)
730 
731                 elif user.password != password:
732                     # bad password, logging it and flash the error
733                     logging.warning("Try to modify the password but introduce wrong
                        password : " + modif_mdp + " ip : " + str(ip))
734                     flash("Wrong password")
735 
736         table = t.get_table()
737         heure = t.get_colonne_heure()
738         new_heure = t.get_new_colonne_heure()
739 
740         table2 = t2.get_table()
```

```
741         name_t2 = t2.get_name()
742 
743         return render_template('settings.html', table=table, heure=heure,
            new_heure=new_heure, table2=table2,
744                                name_t2=name_t2, mode_manuel=mode_manuel,
                                   mode_heure_manuel=mode_heure_manuel[0],
745                                cam_conn=cam_connect, time=current_time, warning=warning)
746 
747 
748     @app.route('/camera', methods=['POST', 'GET'])
749     def camera():
750         """The camera is only here to receive data from the camera. The script is
            checking mac address to verify if the
751         sender is a RPi camera. If it is, the server is checking if the data is
            corresponding with the ELO protocol
752         (the first byte is 0x55 and the last byte is 0xAA). If it is, the server is
            checking if the data is corresponding.
753 
754         :return: The server is returning the string '$,RPWCOK,002,ok,0*' or
            '$,RPWCER,005,error,0*'  to the camera.
755         :rtype: str
756         """
757 
758         global cam_connect
759         global buttonSts_p1
760         global color
761         global cam_send_turn_off
762         global number_of_people
763 
764         # This code is getting the user's IP address and returning it.
765         ip = request.environ.get('HTTP_X_REAL_IP', request.remote_addr)
766         # Calling a function that will return the current time.
767         current_time, current_day = get_time()
768         # This code is retrieving the user's id from the session.
769         id = session.get("user_id")
770 
771         # Check if the user have an id
772         # if yes => 404
773         if id is not None:
774             abort(404)
775         # else check is mac add
776         else:
777             # This code is using the getmacadd function to get the mac address of the ip
                address.
778             mac_add = getmacadd(ip)
779 
780             # The above code is checking to see if the MAC address of the camera is in
                the list of MAC addresses of the
781             # cameras that are connected to the system.
782             # If the MAC address is not in the list, error 404.
783             if not any(mac_add in x for x in cam_connect) or mac_add is False:
784                 abort(404)
785 
786         # Checking if the request method is POST. If it is, it will run the code below.
787         if request.method == 'POST':
788             data = request.form.get('data')
789             # Checking if the data is not empty.
790             if data:
791                 # Verify if data received corresponding with ELO communication protocol
792                 if data[0] == "$" and data[::-1][0] == "*":
793                     # The above code is reading the data from the file and splitting it
                        into a list.
794                     data = data.split(',')
795                     # RPWCSD is the header (Raspberry Pi Wi-Fi Camera Send Data)
796                     if data[1] == "RPWCSD":
797                         # Length of the data
798                         length = int(data[2])
799                         # Time when it was sent
800                         h = data[3][0:5]
801                         # Temperature of the camera
802                         cam_temp = int(data[3][5:length - 1])
803                         # data : 1 entry, 0 exit
```

```
                    d = data[3][length - 1]

                    # The code is checking to see if the temperature is greater than
                    70 degrees.
                    # If it is, it will update the temperature in the cam_connect
                    list. If not, it will update
                    # the temperature in the cam_connect list as 0.
                    if cam_temp > 70:
                        cam_connect[[mac_add in x for x in
                        cam_connect].index(True)][3] = cam_temp
                    else:
                        cam_connect[[mac_add in x for x in
                        cam_connect].index(True)][3] = 0

                    # The code is checking to see if the data is 1. If it is, it
                    will increment the number of people
                    # else it will decrement the number of people.
                    if d == "1":
                        number_of_people = number_of_people + 1
                    elif d == "0" and number_of_people > 0:
                        number_of_people = number_of_people - 1

                    logging.info(" *** Number of people : " + str(number_of_people)
                    + " ***")

                    # if manuel mode is disabled check for the request
                    # Checking if the user has selected the manual mode. If they
                    have, then the program will not check
                    # the Table value.
                    if not mode_manuel:
                        # Creating two table object.
                        t = Table()
                        table = t.get_table()
                        table_heure = t.get_colonne_heure()

                        t2 = Table2()
                        table2 = t2.get_table()

                        # The above code is splitting the current_time string into
                        two variables: current_h and
                        # current_min.
                        current_h, current_min = [x for x in current_time.split(":")]

                        # Find out in which time slot you are in.
                        for i in reversed(range(len(table_heure))):
                            # Splitting the string into two parts, one for the hours
                            and one for the minutes.
                            h, m = table_heure[i].split(":")
                            if current_h == h and current_min >= m:
                                # if the box is orange
                                if table[i][int(current_day)] == 1 or
                                table[i][int(current_day)] == 3:
                                    # Turn on the light
                                    if d == "1":
                                        # Taking the transpose of the table2 array.
                                        array_t2 = np.array(table2).T

                                        # The above code is selecting the correct
                                        row from the array_t2 based on the
                                        # current_day.
                                        array_t2 = array_t2[int(current_day)]
                                        # This code is converting the string values
                                        of the button status to integer
                                        # value of 0 or 1.
                                        array_comp = [0 if x == "off" else 1 for x
                                        in buttonSts_p1]
                                        # This code is taking the two arrays and
                                        comparing them to each other element by
                                        # element.
                                        # Then do an OR operation to turn on the new
                                        light without modifying the other.
                                        array_t2_or = [a or b for a, b in
```

```
                                        zip(array_t2, array_comp)]
861
862                                     # If the value is 0, it will be "off",
                                        otherwise it will be "on".
863                                     buttonSts_p1 = ["off" if x == 0 else "on"
                                        for x in array_t2_or]
864                                     # If the value is 0, it will be "#333333",
                                        otherwise it will be "#FFFFFF".
865                                     color = ["#333333" if x == 0 else "#FFFFFF"
                                        for x in array_t2_or]
866
867                                     # Reset cam_send_turn_off
868                                     cam_send_turn_off = []
869
870                                     # Turn on the light
871                                     gpio_modif()
872                                     logging.info(" *** Camera turn on the light.
                                        *** ")
873                                 elif d == "0" and number_of_people == 0:
874                                     # Add the current_time in cam_send_turn_off.
875                                     # After TURNOFFTIME if any other action is
                                        arrive turn off all light
876                                     cam_send_turn_off.append(current_time)
877                                 break
878
879                     # This code is checking the list of camera IP addresses and if
                        the IP address is not in the list, it
880                     # will add it to the list.
881                     for i, x in enumerate(cam_connect):
882                         if mac_add in x and ip not in x:
883                             cam_connect[i][1] = ip
884                             logging.info(" *** New camera : " + ip + " *** ")
885
886                 # Return OK to the camera with respecting the protocol of
                    communication ELO.
887                 return '$,RPWCOK,002,ok,0*'
888
889     # Return error to the camera with respecting the protocol of communication ELO.
890     return '$,RPWCER,005,error,0*'
891
892
893 @app.before_first_request
894 def activate_job():
895     """activate_job : fonction to run job on background of the web server (with
        thread)
896
897     :return: None
898     :rtype: None
899     """
900
901     def run_job():
902         """run_job : run background job (run every minute).
903         Automatically turn off the light.
904         Check temperature of the Raspberry Pi
905
906         :return: None
907         :rtype: None
908         """
909
910         global mode_manuel
911         global mode_heure_manuel
912         global cam_connect
913         global cam_send_turn_off
914         global TURNOFFTIME
915         global number_of_people
916
917         current_time, _ = get_time()
918         saved_time = current_time
919
920         logging.debug(" *** Starting while loop *** ")
921
922         # While loop that will run forever.
```

```
923         while True:
924             # Wait a minute.
925             while saved_time == current_time:
926                 current_time, current_day = get_time()
927 
928             saved_time = current_time
929 
930             # The above code is checking if the current time is 00:00 (midnight).
931             # If it is, it sets the mode_manuel to 0 and the mode_heure_manuel to
                [0, 0, 0].
932             if current_time == "00:00":
933                 number_of_people = 0
934                 mode_manuel = 0
935                 mode_heure_manuel = [0, 0, 0]
936                 logging.info(" *** Reset midnight ***")
937 
938             # check for automatic off on the table 1 or if camera requested a shutdown
939             if not mode_manuel:
940                 # Getting all the value of the table 1.
941                 t = Table()
942                 table = t.get_table()
943                 table_heure = t.get_colonne_heure()
944 
945                 # Check for automatic turn off
946                 for i in range(len(table_heure)):
947                     # Check if current_time is equal to one of the times in the table.
948                     if current_time == table_heure[i]:
949                         # Ff the box in the table is blue.
950                         if table[i][int(current_day)] > 1:
951                             # Turn all off.
952                             logging.info(" *** Server turn all off automatically ***")
953 
954                             for x in range(8):
955                                 buttonSts_p1[x] = "off"
956                                 buttonSts_p2[x] = "off"
957                                 color[x] = "#333333"
958 
959                 # Checks if the camera has requested a shutdown
960                 for time_turn_off in cam_send_turn_off:
961                     # Compares the current time with the time of the requests add to
                    TURNOFFTIME.
962                     time_add = datetime.strptime(time_turn_off, '%H:%M') + TURNOFFTIME
963                     time_add = time_add.strftime("%H:%M")
964                     if current_time == time_add:
965                         # Clear the requests list
966                         cam_send_turn_off = []
967 
968                         # Split the current time
969                         current_h, current_min = [x for x in current_time.split(":")]
970 
971                         # Find out what time slot you are in
972                         for j in reversed(range(len(table_heure))):
973                             h, m = table_heure[j].split(":")
974                             if current_h == h and current_min >= m:
975                                 # If camera as authorisation
976                                 if table[j][int(current_day)] == 1 or
                                    table[j][int(current_day)] == 3:
977                                     # Turn all off.
978                                     logging.info(" *** Camera turn all the light off
                                        ***")
979 
980                                     for x in range(8):
981                                         buttonSts_p1[x] = "off"
982                                         buttonSts_p2[x] = "off"
983                                         color[x] = "#333333"
984 
985                 # Update GPIO
986                 gpio_modif()
987 
988             # The above code is reading the temperature of the CPU only if it's
                running on a RPi.
989             if rpi:
```

```
                with open('/sys/class/thermal/thermal_zone0/temp', 'r') as ftemp:
                    global warning
                    temp = int(ftemp.read()) / 1000
                    # This code is checking the temperature and if it is greater
                    than 60 it will send a warning.
                    if temp > 60:
                        logging.warning(" *** Temp = " + str(int(temp)) + "°C *** ")
                        warning = "Temp = " + str(int(temp)) + "°C"
                    else:
                        warning = ""

            # The above code is checking to see if the cameras are online.
            for i in range(len(cam_connect)):
                ip_cam = cam_connect[i][1]
                cam_connect[i][0] = myping(ip_cam)

    # This code is creating a thread that will run the run_job function.
    thread = threading.Thread(target=run_job)
    thread.start()


if __name__ == "__main__":
    logging.info(" *** Starting server *** ")
    app.run(host='0.0.0.0', port=80, debug=False)
    GPIO.cleanup()
    logging.info(" *** Server stopped *** ")
```