

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"

ДИСЦИПЛИНА

«Технология проектирования программного обеспечения»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1 и №2 - ВАРИАНТ 5131

«Проектирование сетевого сервиса для IoT»

Выполнил:

Магистрант гр. N42604

Ле Динь Хыонг

15.12.2022



дата/подпись

Проверил:

Преподаватель

Гирик Алексей Валерьевич

дата/подпись

Санкт-Петербург,

2022 г.

Содержание

1 Введение	3
2 Анализ требований	4
3 Техническое задание	5
3.1 Архитектура программного комплекса	5
3.2 Описание принципиальных технических решений	5
3.3 Описание формата хранения данных в файле устройства	6
3.4 Описание протокола взаимодействия клиентов и сервера	6
3.5 Описание процедуры установки зависимостей, запуска программ	9
4 Рабочий проект	9
5 Заключение	11
6 Список использованных источников	11
Приложения А	12
Приложения Б	15
Приложения В	17
Приложения Г	19

1 Введение

Интернет вещей (Internet of Things, IoT) – это множество физических объектов, подключенных к интернету и обменивающихся данными. Концепция IoT может существенно улучшить многие сферы нашей жизни и помочь нам в создании более удобного, умного и безопасного мира. Примеры Интернета вещей варьируются от носимых вещей, таких как умные часы, до умного дома, который умеет, например, контролировать и автоматически менять степень освещения и отопления. Также ярким примером служит так называемая концепция умного предприятия (Smart Factory), которое контролирует промышленное оборудование и ищет проблемные места, а затем перестраивается так, чтобы не допустить поломок. Интернет вещей занимает важное место в процессе цифровой трансформации в компаниях. Прогнозируется, что к 2030 году количество подключенных к сети устройств вырастет примерно до 24 млрд с годовой выручкой до 1,5 трлн долларов [1].

В этой лабораторной работе рассмотрим очень типичную структуру IoT системы, в которой самая важная задача является обеспечением стабильного и эффективного взаимодействий между объектами (серверы, клиенты, устройствами,...).

2 Анализ требований

Суть лабораторной работы состоит в том, чтобы обеспечить управление некоторым устройством по сети с помощью современного стека технологий и средств проектирования и разработки программного обеспечения.

В данной работе надо спроектировать и разработать две программы для ОС Linux на языке программирования Python:

1. Сервис, отслеживающий изменения в состоянии «умного» устройства, имитируемого локальным файлом, согласно варианту лабораторной работы, и получающий/передающий запросы и ответы по сетевому протоколу управления, заданному вариантом лабораторной работы.
2. Клиентскую программу для управления устройством через сервис (и проверки работы сервиса).

Требования задания представлены в таблице 1.

Таблица 1 – Требования задания.

Требование	Название	Описание
Устройство	Жалюзи	<i>Параметры:</i> 1. Процент сдвига полотна (0 .. 100). 2. Процент пропуска светового потока (0 .. 100 процентов). 3. Текущая освещенность с внешней стороны (0 .. 50000 лк). <i>Функции:</i> 1. Установить проценты сдвига полотна и пропуска светового потока. 2. Получить значения процентов сдвига полотна, пропуска светового потока и текущей освещенности с внешней стороны. 3. Получить уведомление об изменении процентов сдвига полотна и пропуска светового потока.
Формат хранения данных в файле устройства	Plain text	Данные в виде совокупности строк, содержащих печатные символы и символы-разделители строк (желательно «\n», LINE FEED).
Формат передачи по прикладному протоколу управления	JSON	Формат JavaScript Object Notation.
Транспортный протокол	TCP	Клиент получает уведомления по прикладному протоколу управления до тех пор, пока соединение остается открытым.

Сервис должен быть выполнен в виде консольного приложения, которое можно запустить либо интерактивно в терминале, либо с помощью подсистемы инициализации и управления службами `systemd`. Функциональность сервиса можно условно поделить на две части: имитация взаимодействия (опроса или получения уведомлений) с устройством и взаимодействие с клиентами по сети.

Клиенская программа может быть выполнена в виде консольного приложения, которое можно запустить интерактивно в терминале, или приложения с TUI/GUI. Любое количество клиентов может подключиться к серверной части (сервису) и согласованно управлять «устройством».

3 Техническое задание

3.1 Архитектура программного комплекса

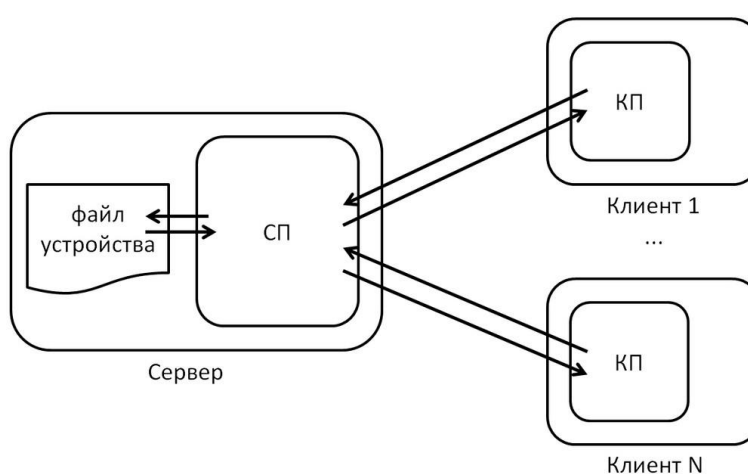


Рисунок 1 – Общая схема взаимодействия компонентов программного комплекса (СП – серверное приложение, КП – клиентское приложение).

Программный комплекс состоит из трех частей:

1. Серверное приложение – это программа, которая взаимодействует с клиентами и устройствам (файл устройства).
2. Клиентское приложение – это программа, с помощью которой пользователь может узнать про изменения состояний устройства или он может настроить это через сервер.
3. Файл устройства – это файл для иллюстрации работы реального устройства.

3.2 Описание принципиальных технических решений

В основном при реализации обо клиентской и серверной программ использует *Python3* (version *Python 3.9.12*) в качестве языка программирования.

В лабораторной работе №1 для обеспечения связи между клиентом и сервером используется протокол *TCP*, для обеспечения обмена данными между процессам используется модуль *socket*, нет внешних библиотеки.

В дополнение ко всем требованиям, изложенным в задании к лабораторной работе №1, в лабораторной работе №2 сервис должен поддерживать управление «устройством» с помощью REST API. Проверка работы API должна быть возможна с помощью любого клиентского приложения, поддерживающего передачу HTTP-запросов. Для обеспечения связи между клиентом и сервером, используются *Flask* (version *Flask 2.2.2*) и *Flask-SocketIO* (version *Flask-SocketIO 5.3.1*) на серверной программе, *SocketIO* на клиентской программе.

Для обеспечения связи между сервером и устройством используется подходящий метод чтения/записи соответствующий формату файл устройства.

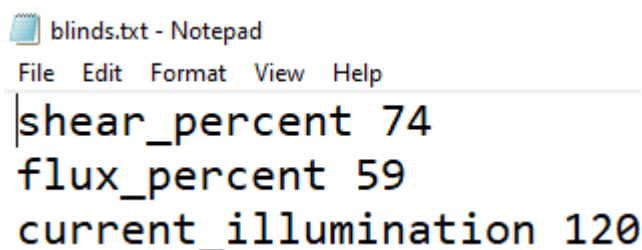
Для текстового редактора используется *Visual Studio Code*.

3.3 Описание формата хранения данных в файле устройства

Устройство используется в данной работе является жалюзи.

Формат хранения данных в файле устройства – plain text. Данные в виде совокупности строк, содержащих печатные символы и символы-разделители строк (желательно «\n», LINE FEED).

Для иллюстрации работы этого устройства файл «blinds.txt» используется.



```
blinds.txt - Notepad
File Edit Format View Help
shear_percent 74
flux_percent 59
current_illumination 120
```

Рисунок 2 – Формат хранения данных в файле устройства жалюзи

Параметры:

- «shear_percent»: процент сдвига полотна (0 .. 100)
- «flux_percent»: процент пропуска светового потока (0 .. 100 процентов).
- «current_illumination»: текущая освещенность с внешней стороны (0 .. 50000 лк).

Число в конце каждой строки показывает значения соответственного параметра.

3.4 Описание протокола взаимодействия клиентов и сервера

В лабораторной работе №1 используется протокол TCP для для передачи данные между сервером и клиентами.

Протокол управления передачей (Transmission Control Protocol – TCP) – сетевая модель, описывающая процесс передачи цифровых данных [2]. Сервер может сам отправить данные к клиенту и обратно. Связь между сервером и клиентом является постоянным. Схема взаимодействия сервера с клиентом через интерфейс сокетов приведена на рисунке 3.



Рисунок 3 – Схема установления связи и передачи данных между клиентом и сервером.

Для создания соединения TCP/IP необходимо два сокета: один на локальной машине, а другой на удаленной. Таким образом, каждое сетевое соединение характеризуется IP-адресом локальной машины и портом на локальной машине, IP-адресом и портом на удаленной машине.

При реализации модели "клиент-сервер", сервер обычно слушает (`listen`) порт с определенным номером. Дождавшись запроса от клиента на этот порт, сервер и клиент устанавливают соединение, используя свободные порты [3].

Данные отправлены из клиента к серверу в виде Json:

```

{
    'command' : <команд>,
    'parameter' : <параметр>,
    'value' : <значение>
}
  
```

'command' может быть `get` (для получить текущее значение параметра устройства) или `set` (для изменить значения параметра устройства). Если значение 'command' является `get` то нам не нужно поля 'value'.

Данные отправлены из сервер к клиенту просто в виде String чтобы клиент просто сразу выразил результат на экран (terminal).

Во второй лабе протокол HTTP используется для того чтобы клиент отправил команды на сервер. Передача состояния представления (Representational State Transfer – REST) – это архитектурный стиль взаимодействия компонентов распределённого приложения в сети [4]. Под REST (RESTful) подразумевают основные принципы, по которым приложение или сетевой ресурс взаимодействует с сервером при помощи протокола HTTP.

Собственно API – это программный интерфейс, предназначенный для взаимодействия между программами. REST — это интерфейс связи, работающий благодаря протоколу HTTP.

Сама по себе архитектура REST не привязана к использованию конкретных технологий и протоколов. Однако современная Web реальность такова, что для построения RESTful API практически всегда используется HTTP плюс какой-то распространенный формат представления ресурсов

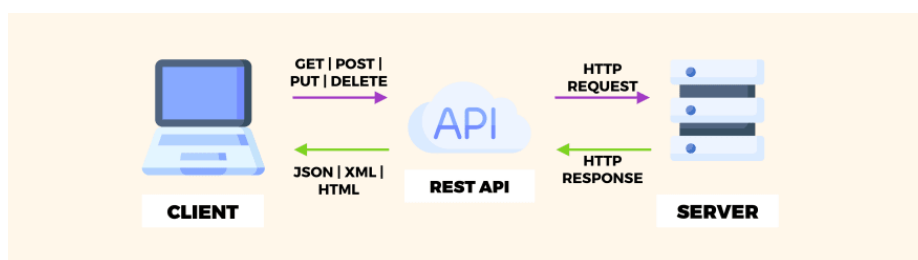


Рисунок 4 – Пример REST API.

Способ взаимодействия с сервером состоит минимум из четырех действий:

1. Прием информации с сервера (не должен изменять его состояние).
2. Внесение новых сведений.
3. Изменение уже имеющихся.
4. Удаление информации.

Такого количества достаточно, хотя для конкретной задачи их может быть и больше. Каждый вид требует своего метода запроса – соответственно GET, POST, PUT и DELETE [5].

В данной работе использует один простой метод, при котором только GET method используется. Это называется Query String. Суть этого метода заключается в том что все команды вы прикрепим в URL у запроса:

Чтобы получить значения процентов сдвига полотна, пропуска светового потока и текущей освещенности с внешней стороны, в браузере открыть ссылку:

<http://<ip>:<port>/blinds/<parameter>>

Где ip - адрес; port - порт; parameter - shaer, flux или illumination.

Пример: <http://127.0.0.1:5000/blinds/illumination>

Чтобы установить проценты сдвига полотна пропуска светового потока, , в браузере открыть ссылку:

<http://<ip>:<port>/blinds?<parameter>=<X>>

Где ip - адрес; port - порт; parameter - shaer или flux, X = 0 .. 100.

Пример: <http://127.0.0.1:5000/blinds?flux=59>

3.5 Описание процедуры установки зависимостей, запуска программ

В лабораторной работе №1 не используется внешняя библиотека, поэтому запускаем программы прямо.

В окне терминала для запуска сервера,: `python tppo_server_5131.py ip port`

Где ip: адрес, port: порт

Пример: `python tppo_server_5131.py 127.0.0.1 5000`

В другой окне терминала для запуска клиента: `python tppo_client_5131.py ip port`

Пример: `python tppo_client_5131.py 127.0.0.1 5000`

Чтобы установить проценты сдвига полотна пропуска светового потока, `set shaer X` и `set flux X`. Где X = 0 .. 100

Чтобы получить значения процентов сдвига полотна, пропуска светового потока и текущей освещенности с внешней стороны, `get shaer`, `get flux` и `get illumination`

Пример: `set shaer 52`, `get flux`

В лабораторной работе №2 используются несколько внешних библиотек, поэтому нужно установить зависимости. В окне терминала: `pip install -r requirements.txt`

Дальше для запуска программ клиента и сервера команды аналогичны как для запуска программ лабораторной работы №1

4 Рабочий проект

Структура исходного кода:

```
.../tppo_5131_labs/  
    tppo_5131_lab_1/  
        tppo_server_5131.py  
        tppo_client_5131.py  
        blinds.txt  
    tppo_5131_lab_2/  
        tppo_server_5131.py  
        tppo_client_5131.py  
        blinds.txt  
        requirements.txt
```

Дальше посмотрим результат работы программы:





Рисунок 6 – Test REST API.

5 Заключение

Результаты лабораторных работ соответствуют всем требованиям. Видно что для того чтобы много клиентов можно подключить одновременно, использование многопоточный подход является очень важным.

В качестве жалюзи можно подключить любое устройство. При этом зависит от протокола работы устройства, соответствующие REST API будут использованы.

Хотя это довольно простая система интернета вещей но она дает нам общую картину и основные принципы о том как эта система работает и как применить ее в жизни.

6 Список использованных источников

1. Что такое IoT и что о нем следует знать. URL: <https://habr.com/ru/company/otus/blog/549550/> (Дата обращения: 15.12.2022).
2. Руководство по стеку протоколов TCP/IP для начинающих. URL: <https://selectel.ru/blog/tcp-ip-for-beginners/#baza> (Дата обращения: 15.12.2022).
3. Описание функций работы с сокетами. URL: <https://studfile.net/preview/2567136/page:8/> (Дата обращения: 15.12.2022).
4. REST. Материал из Национальной библиотеки им. Н. Э. Баумана. URL: <https://ru.bmstu.wiki/REST> (Дата обращения: 15.12.2022).
5. Что такое REST API? URL: <https://www.compuhome.ru/chto-takoe-rest-api.html> (Дата обращения: 15.12.2022).

Приложения А

<tppo_5131_lab_1/tppo_server_5131.py>

```
import json
import socket
from threading import Lock
import os
import threading
import argparse

parser = argparse.ArgumentParser(description='Ip and port of Server')
parser.add_argument('server_ip', type=str, help='Ip of Server')
parser.add_argument('server_port', type=int, help='Port of Server')
args = parser.parse_args()

def read_txt():
    path = 'blinds.txt'
    with open(path, 'r') as file:
        data = file.readlines()
        shear_percent = int(data[0].split()[-1])
        flux_percent = int(data[1].split()[-1])
        current_illumination = int(data[2].split()[-1])
        vals = [shear_percent, flux_percent, current_illumination]
    return vals

def write_txt(data):
    path = 'blinds.txt'
    with open(path, 'w') as file:
        file.write(data)

vals = read_txt()

parameters = ['shaer', 'flux', 'illumination']

class Server:
    def __init__(self, host, port):
        self.sock = self._setup_socket(host, port)
        self.connections = []
        print("[STARTING] Server is listening on {}: {}".format(host, port))

    def run(self):
        blinds_changes_thread = threading.Thread(target=self.blinds_changes)
        blinds_changes_thread.daemon = True
        blinds_changes_thread.start()

        while(True):
            conn, addr = self.sock.accept()
            if (self._check_for_existing_conn(conn) == False):
                self.connections.append(conn)
            else:
                print("Connection has already existed")
```

```

print(f"[NEW CONNECTION] {addr} connected")

cmd_parsing_thread = threading.Thread(target=self.cmd_parsing,args=(conn, addr))

cmd_parsing_thread.daemon = True
cmd_parsing_thread.start()

def blinds_changes(self):
    deviceFileName = 'blinds.txt'
    _cached_stamp = os.stat(deviceFileName).st_mtime
    while (True):
        stamp = os.stat(deviceFileName).st_mtime
        if(stamp != _cached_stamp):
            _cached_stamp = stamp

            new_vals = read_txt()

            for idx in range(3):

                if new_vals[idx] != vals[idx]:
                    for connection in self.connections:
                        connection.send('{} has been changed to {}'.format(parameters[idx],
new_vals[idx])).encode())
                        vals[idx] = new_vals[idx]

def cmd_parsing(self, conn, addr):
    while(True):
        byteData = conn.recv(4096)
        stringData = byteData.decode("utf-8")
        print('CLIENT: ', stringData)
        f_json = json.loads(stringData)
        vals = read_txt()
        if f_json['command']=='set':
            idx = ['shaer', 'flux', 'illumination'].index(f_json['parameter'])
            vals[idx] = f_json['value']

            data = 'shear_percent {} \n flux_percent {} \n current_illumination {} \n'.format(vals[0],
vals[1], vals[2])
            write_txt(data)

            if f_json['command']=='get':
                idx = ['shaer', 'flux', 'illumination'].index(f_json['parameter'])
                conn.send("{}: {}".format(f_json['parameter'], vals[idx])).encode())

def _check_for_existing_conn(self, conn):
    connExists = False
    for connection in self.connections:
        if (str(conn) == str(connection)):
            connExists = True
            break
    return connExists

```

```
@staticmethod
def _setup_socket(host,port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind((host, port))
    sock.listen()
    return sock

if __name__ == "__main__":
    server = Server(args.server_ip, args.server_port)
    server.run()
```

Приложения Б

<tppo_5131_lab_1/tppo_client_5131.py>

```
import socket
from threading import Thread
import json
import argparse

parser = argparse.ArgumentParser(description='Ip and port of Server')
parser.add_argument('server_ip', type=str, help='Ip of Server')
parser.add_argument('server_port', type=int, help='Port of Server')
args = parser.parse_args()

class Client:
    def __init__(self, host, port):
        self.sock = self._setup_socket(host, port)
        thread = Thread(target=self.send_cmd)
        thread.daemon = True
        thread.start()

        print("[CONNECTED] Client connected to server at {}: {}".format(host, port))
        print("Parameters: shaer (0..100); flux (0..100); illumination (0..5000)")
        print("Syntax:\n\
get <parameter>\n\
set <parameter> <value>")
        while(True):
            data = self.sock.recv(4096)
            print("SERVER: ", data.decode())

    def send_cmd(self):
        while(True):
            try:
                text = input()
                elements = text.split()
                parameters = ['shaer', 'flux', 'illumination']
                if len(elements)==3 and elements[0]=='set' and elements[1] in parameters[0:2] and
0<=float(elements[2])<=100:
                    set_json = {
                        'command': elements[0],
                        'parameter': elements[1],
                        'value': elements[2]
                    }
                    self.sock.send(json.dumps(set_json).encode('utf-8'))
                elif len(elements)==2 and elements[0]=='get' and elements[1] in parameters:
                    get_json = {
                        'command': elements[0],
                        'parameter': elements[1],
                    }
                    self.sock.send(json.dumps(get_json).encode('utf-8'))
            else:
                print('Syntax error')
```

```
        except:
            print('Syntax error')

    @staticmethod
    def _setup_socket(host, port):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((host, port))
        return sock

if __name__ == "__main__":
    client = Client(args.server_ip, args.server_port)
```


Приложения В

<tppo_5131_lab_2/tppo_server_5131.py>

```
from flask import Flask, request
from flask_socketio import SocketIO
import argparse
import os
import threading

parser = argparse.ArgumentParser(description='Ip and port of Server')
parser.add_argument('server_ip', type=str, help='Ip of Server')
parser.add_argument('server_port', type=int, help='Port of Server')
args = parser.parse_args()

app = Flask(__name__)
socketio = SocketIO(app)

def read_txt():
    path = 'blinds.txt'
    with open(path, 'r') as file:
        data = file.readlines()
    shear_percent = int(data[0].split()[-1])
    flux_percent = int(data[1].split()[-1])
    current_illumination = int(data[2].split()[-1])
    vals = [shear_percent, flux_percent, current_illumination]
    return vals

def write_txt(data):
    path = 'blinds.txt'
    with open(path, 'w') as file:
        file.write(data)

@app.route('/blinds/<parameter>')
def main(parameter):
    vals = read_txt()
    idx = ['shaer', 'flux', 'illumination'].index(parameter)
    return {parameter: vals[idx]}

@app.route('/blinds')
def blinds():
    shaer = request.args.get('shaer')
    flux = request.args.get('flux')
    vals = read_txt()

    if shaer!=None:
        vals[0] = shaer
        data = 'shear_percent { }\nflux_percent { }\ncurrent_illumination { }\n'.format(vals[0],
vals[1], vals[2])
        write_txt(data)
        return {"shaer":shaer}
    elif flux!=None:
```

```

        vals[1] = flux
        data = 'shear_percent {} \nflux_percent {} \ncurrent_illumination {} \n'.format(vals[0],
vals[1], vals[2])
        write_txt(data)
        return { "flux":flux }

@socketio.on('connect')
def connect():
    blinds_changes_thread = threading.Thread(target=blinds_changes)
    blinds_changes_thread.start()
    print("[NEW CONNECTION] connected!")

def blinds_changes():
    vals = read_txt()
    parameters = ['shaer', 'flux', 'illumination']
    deviceFileName = 'blinds.txt'
    _cached_stamp = os.stat(deviceFileName).st_mtime
    while (True):
        stamp = os.stat(deviceFileName).st_mtime
        if(stamp != _cached_stamp):
            _cached_stamp = stamp
            new_vals = read_txt()
            for idx in range(3):
                if new_vals[idx] != vals[idx]:
                    socketio.emit("blinds_changes", { "parameter":parameters[idx],
"value":new_vals[idx]}, broadcast=True)
                    vals[idx] = new_vals[idx]

if (__name__ == "__main__"):
    socketio.run(app, args.server_ip, args.server_port)

```

Приложения Г

<tppo_5131_lab_2/tppo_client_5131.py>

```
import socketio
import argparse
import requests

parser = argparse.ArgumentParser(description='Ip and port of Server')
parser.add_argument('server_ip', type=str, help='Ip of Server')
parser.add_argument('server_port', type=int, help='Port of Server')
args = parser.parse_args()

sio = socketio.Client()

@sio.event
def connect():
    print("[CONNECTED] Client connected to server at http://{ip}:{port}".format(args.server_ip,
args.server_port))
    print("Parameters: shaer (0..100); flux (0..100); illumination (0..5000)")
    print("Syntax:\n\
get <parameter>\n\
set <parameter> <value>")
    while(True):
        try:
            text = input()
            elements = text.split()
            parameters = ['shaer', 'flux', 'illumination']
            if len(elements)==3 and elements[0]=='set' and elements[1] in parameters[0:2] and
0<=float(elements[2])<=100:
                url = "http://{ip}:{port}/blinds?{param}={val}".format(args.server_ip, args.server_port,
elements[1], elements[2])
                response = requests.get(url)
                print("SERVER RESPONSE: ", response.json())
            elif len(elements)==2 and elements[0]=='get' and elements[1] in parameters:
                url = "http://{ip}:{port}/blinds/{param}".format(args.server_ip, args.server_port, elements[1])
                response = requests.get(url)
                print("SERVER RESPONSE: ", response.json())
            else:
                print('Syntax error')
        except:
            print('Syntax error')

@sio.on('blinds_changes')
def blinds_changes(data):
    print('NOTIFICATION SERVER: {param} has been changed to {val}'.format(data['parameter'],
data['value']))

sio.connect("http://{ip}:{port}".format(args.server_ip, args.server_port))
```