

**Министерство науки и высшего образования Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**«Национальный исследовательский университет ИТМО»**  
**(Университет ИТМО)**

**Факультет      Безопасности Информационных Технологий**

**Образовательная программа: Конструирование и технология электронных средств**

**Направление подготовки (специальность): Разработка безопасных беспилотных транспортных средств**

**Дисциплина:**  
**«Технология проектирования программного обеспечения»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1 и 2**  
**«Проектирование сетевого сервиса для IoT»**

Выполнил:  
Магистрант гр. N42603  
Нгуен Мань Кыонг



Проверил:  
Гирик Алексей Валерьевич, к.т.н., доцент

---

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	10
5. Заключение	11
Список использованных источников	11
Приложение А	12
Приложение Б	14
Приложение В	15
Приложение Г	16
Приложение Д	17

## **1. Введение**

Интернет вещей (англ. Internet of Things, IoT) – это концепция развития технологий сетей Интернет в сторону автоматизации, и исключения участия человека из большинства процессов работы IT-инфраструктуры. Интернет-вещи посредством обмена информацией между различными датчиками и сенсорами должны полностью автоматизировать процессы управления группами устройств.

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.

## 2. Анализ требований

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, и получающий/передающий запросы и ответы по сетевому протоколу управления.

2. Клиентскую программу для управления устройством через сервис (и проверки работы сервиса).

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 с устройством и взаимодействие с клиентами по сети.

Клиентская 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».

Устройство, управление которым обеспечивает сервис, представляется в виде файла с заданным форматом. Файл содержит данные о состоянии устройства. Внешнее изменение файла соответствует внешним изменениям, которые могут произойти с устройством. Сервис должен обнаруживать внешние изменения в состоянии устройства и отправлять уведомления о таких изменениях заинтересованным клиентам. Изменение состояния устройства может также быть выполнено с помощью клиентской программы через сеть.

Информация о устройстве представлена в таблице 1.

Таблица 1: Информация о устройстве

Название устройств	Параметры устройства и поддерживаемые протоколом управления функции
Жалюзи	Параметры: 1. Процент сдвига полотна (0 .. 100 процентов). 2. Процент пропуска светового потока (0 .. 100 процентов). 3. Текущая освещенность с внешней стороны (0 .. 50000 лк). Функции: 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.

### 3. Техническое задание

#### 3.1. Архитектура программного комплекса

Программный комплекс состоит из 3 частей:

- Серверное приложение (СП): программа обеспечивает имитацию взаимодействия с устройством и взаимодействие с клиентами по сети.
- Клиентское приложение (КП): программа обеспечивает подключение любого количества клиентов к серверной части и согласовано управление устройством.
- Файл устройства: файл содержит данные о состоянии устройства (иллюстрация работы реального устройства). Значение данных в файле могут быть получены или изменены с помощью протокола управления. Изменение данных в файле устройства вручную с помощью редактора соответствует внешнему изменению, которое должно быть обнаружено сервисом.

Общая схема взаимодействия показана на рисунке 1.

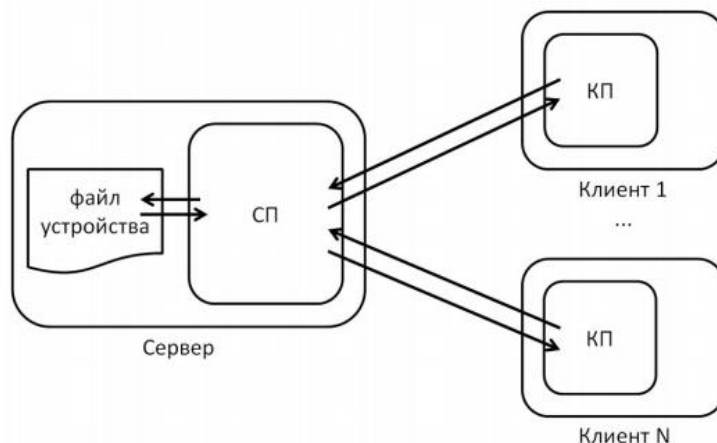


Рисунок 1. Общая схема взаимодействия компонентов программного комплекса (СП – серверное приложение, КП – клиентское приложение).

#### 3.2. Описание принципиальных технических решений

В качестве языка реализации обо серверной и клиентской программ использует Python3 (python 3.9.13).

Для лабораторной работы №1 используется протокол TCP для обеспечения связи между сервером и клиентом. В качестве программного интерфейса для обеспечения обмена данными между процессам используется модуль socket.

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 (кроме уведомлений). Проверка работы API должна быть возможна с помощью любого клиентского приложения, поддерживающего передачу HTTP-запросов. Для

обеспечения этих требования я использовал микрофреймворк Flask (flask 2.2.2), Flask-SocketIO (flask-socketio 5.3.1) для серверной программы и Socket.IO (python-socketio 5.7.2) для клиентской программы.

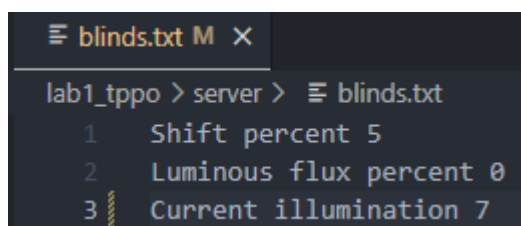
Для обеспечения связи между сервером и устройством я использовал подходящий метод чтения/записи соответствующий формату файл устройства.

В качестве текстового редактора кода используется Visual Studio Code.

### 3.3. Описание формата хранения данных в файле устройства

Формат хранения данных в файле устройства – plain text (данные в виде совокупности строк, содержащих печатные символы и символы-разделители строк “\n”).

На рисунке 2 представлен файл устройства blinds.txt.



```
blinds.txt M X
lab1_tppo > server > blinds.txt
1 Shift percent 5
2 Luminous flux percent 0
3 Current illumination 7
```

Рисунок 2. Формат хранения данных устройства.

Как показано на таблице 1, на файле сохраняют параметры устройства.

- «Shift percent» - процент сдвига полотна (0 .. 100 процентов)
- «Luminous 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 первой лабораторной работе используется протокол TCP для передачи данные между сервером и клиентами. Протокол управления передачей (TCP) - это стандарт связи, который позволяет прикладным программам и вычислительным устройствам обмениваться сообщениями по сети [1]. Протокол TCP имеет высокую надежность, поскольку позволяет не терять данные при передаче, запрашивает подтверждения о получении от принимающей стороны и в случае необходимости отправляет данные повторно. При этом отправляемые пакеты данных сохраняют порядок отправки, то есть можно сказать, что передача данных упорядочена. Минусом данного протокола является относительно низкая скорость передачи данных, за счет того что выполнение надежной и упорядоченной передачи занимает больше времени, чем в альтернативном протоколе UDP [2].

На рисунке 3 представлен типичный сценарий взаимодействия, происходящего между клиентом и сервером с помощью TCP-сокета.

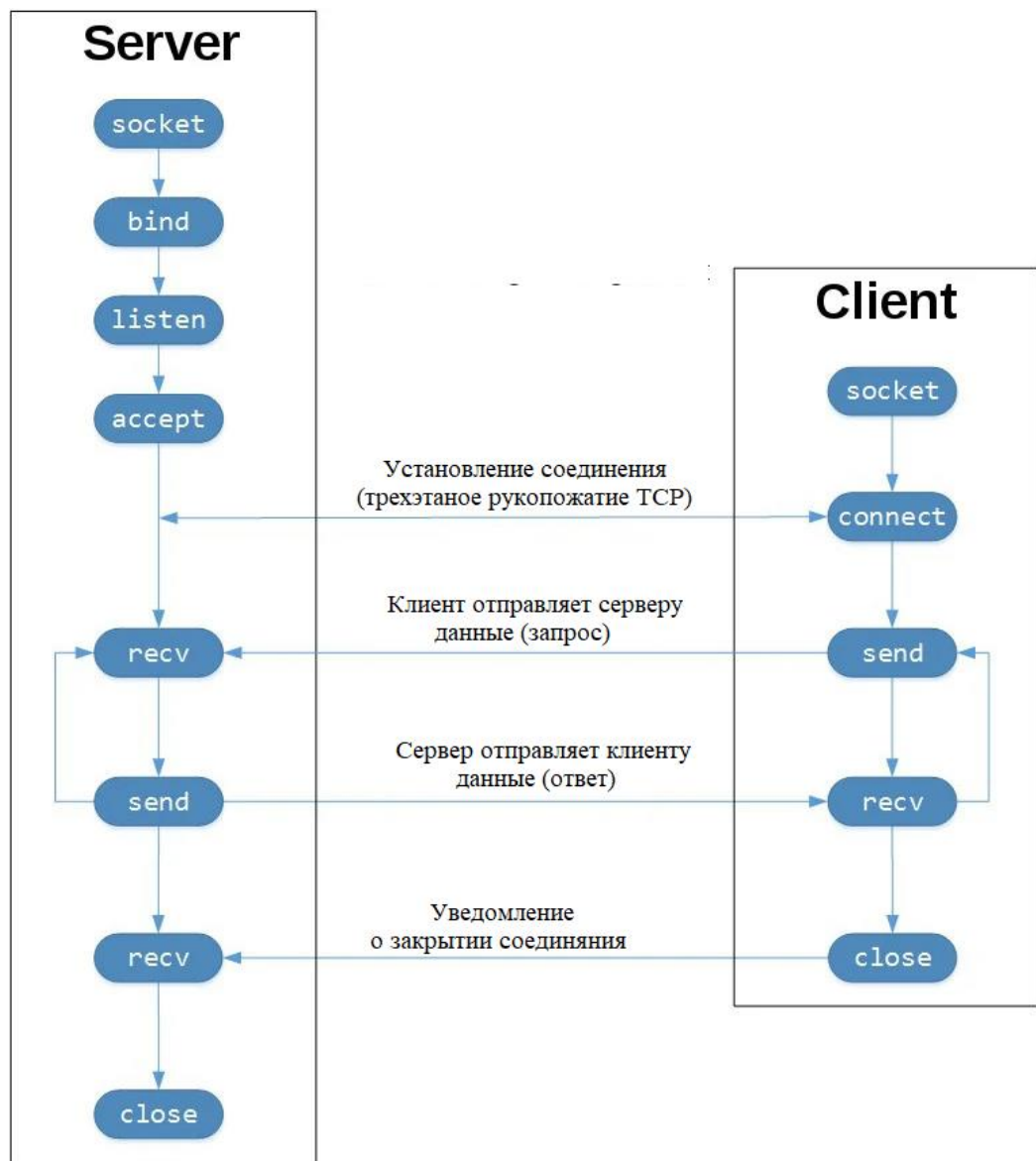


Рисунок 3. Блок-схема коммуникации TCP-сокета

Данные отправлены из клиента к серверу в бинарном виде с помощью методом encode() строки (возвращает закодированную версию данной строки) и decode() для декодирования при получении.

Структура сообщения из клиента к серверу:

- Установить проценты сдвига полотна: set shift\_percent <значение>
- Установить проценты сдвига пропуска светового потока: set flux\_percent <value>
- Получить значения процентов сдвига полотна: get shift\_percent
- Получить значения процентов пропуска светового потока: get flux\_percent
- Получить значения текущей освещенности с внешней стороны: get current\_ill
- Отключить от сервера: exit

Сервер отвечает за соответственный запрос клиенту в виде строки.

В дополнение ко всем требованиям, изложенным в задании к лабораторной работе № 1, во второй лабораторной работе сервис должен поддерживать управление «устройством» с помощью REST API (кроме уведомлений).

REST (Representational State Transfer) в переводе - это передача состояния представления. Технология позволяет получать и модифицировать данные и состояния удаленных приложений, передавая HTTP-вызовы через интернет или любую другую сеть.

API (Application Programming Interface), или программный интерфейс приложения - это набор инструментов, который позволяет одним программам работать с другими.

Если проще, то REST API - это когда серверное приложение дает доступ к своим данным клиентскому приложению по определенному URL. REST API позволяет использовать для общения между программами протокол HTTP (зашифрованная версия — HTTPS), с помощью которого мы получаем и отправляем большую часть информации в интернете. В данной работе используется протокол HTTP. Посмотрим на работу протокола HTTP на примере. Допустим, есть адрес `http://localhost:12345/blind/shift_percent`. Он состоит из двух частей: первая - это адрес сервера, то есть `http://localhost:12345`. Вторая - адрес ресурса на удаленном сервере, в данном примере - `/blind/shift_percent`.

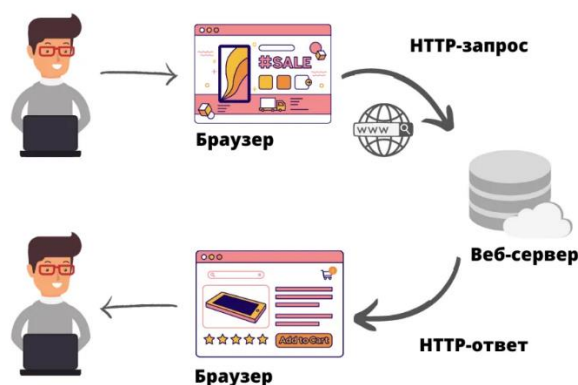


Рисунок 4. Пример HTTP-запроса к серверу

Чтобы ресурс, который мы запрашиваем, выполнял нужные действия, используют разные способы обращения к нему. В API-системе пять основных классических методов: GET, POST, PUT, PATCH, DELETE. В данной работе использует самый популярный метод: GET, который запрашивает информацию из указанного источника и не влияет на его содержимое. Одним из способов передачи данных в приложение представляет использование параметров строки запроса (query string).

Итак чтобы узнать состояние устройства нужно отправить запрос GET с следующим URL: `GET http://{host}:{port}/blind/{param}`. Чтобы изменить состояние устройства нужно отправить запрос GET с следующим URL: `GET http://{host}:{port}/blind?{param}={value}`.

Для уведомления о изменениях состояния устройства заинтересованным клиентам использоваться библиотека Socket.IO, которая обеспечивает двустороннюю связь в реальном времени между клиентом и сервером. Socket.IO главным образом использует протокол WebSocket.

Структура сообщения из клиента к серверу во второй лабораторной работе:

- Установить проценты сдвига полотна: `set shift_percent <значение>`
- Установить проценты сдвига пропуска светового потока: `set flux_percent <value>`
- Получить значения процентов сдвига полотна: `get shift_percent`



- Получить значения процентов пропуска светового потока: `get flux_percent`
- Получить значения текущей освещенности с внешней стороны: `get current_ill`

### **3.5. Описание процедуры установки зависимостей, запуска программ**

В лабораторной работе №1 нет никаких внешних библиотек, поэтому можно запускать программы прямо:

- для запуска серверной программы: `python3 tppo_server_5121.py 127.0.0.1 12345`
- для запуска клиентской программы: `python3 tppo_client_5121.py 127.0.0.1 12345`

Во лабораторной работе №2 нужно установить несколько внешних библиотек, список которых написан в файле `requirements.txt`:

- установить необходимые библиотеки: `pip3 install -r requirements.txt`
- для запуска серверной программы: `python3 tppo_server_5121.py 127.0.0.1 12345`
- для запуска клиентской программы: `python3 tppo_client_5121.py 127.0.0.1 12345`

#### 4. Рабочий проект

В каждой лабораторной работе проект состоит из двух частей: серверное, клиентское приложение. В серверной папке файл blinds.txt используется для иллюстрации работы устройства «жалюзи»; файл blinds.py содержит класс Blinds(), в котором описываются методы для работы с файлом устройства; файл requirements.txt показывает необходимые библиотеки, которые нужно установить до запуска программы. Структура исходного кода показана на рисунке 5, исходные код можно смотреть в Приложениях А-Д.

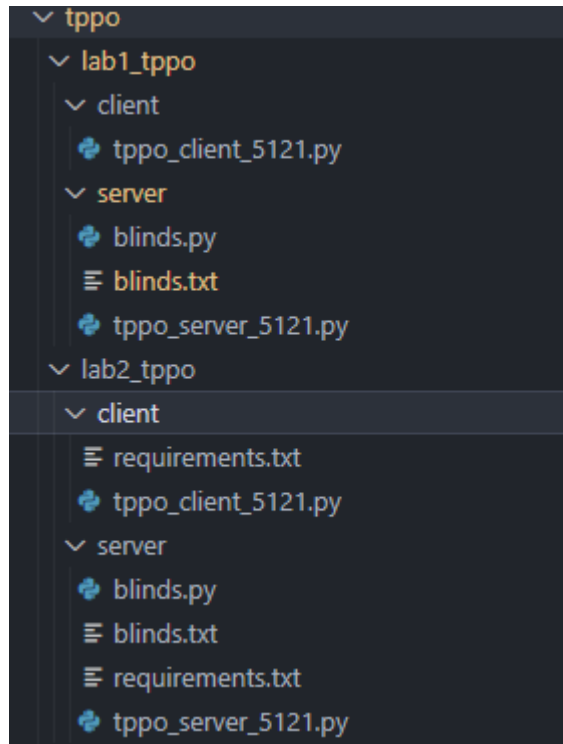


Рисунок 5. Структура исходных кодов

Результат работы программ показан на рисунках 6, 7.

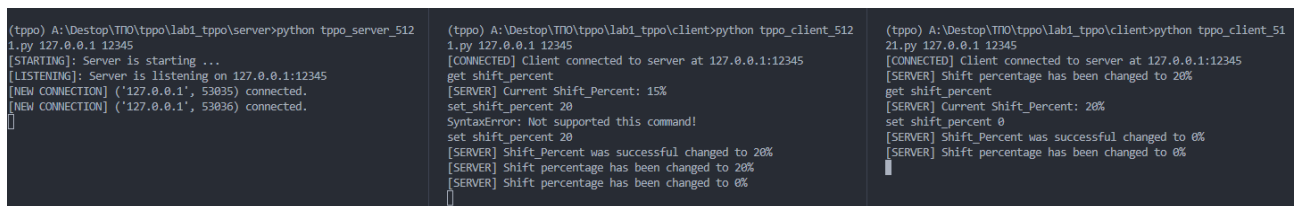


Рисунок 6. Скриншоты работы программ сервера, клиентов 1 и 2 в лаб 1

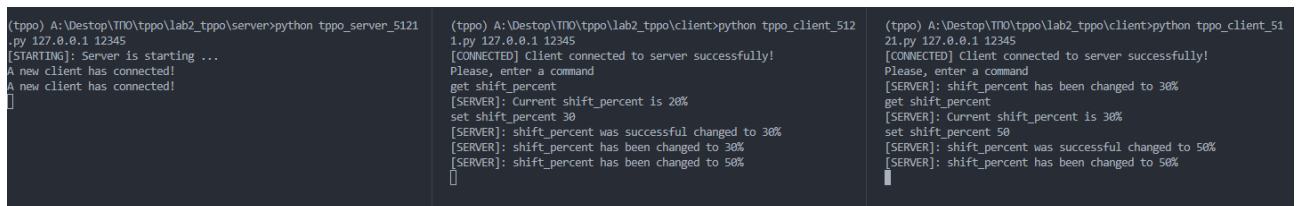


Рисунок 7. Скриншоты работы программ сервера, клиентов 1 и 2 в лаб 2



```
tpo > lab2.tpo > server > F blind.txt
1 | Shift percent 50
2 | Luminous flux percent 0
3 | Current illumination 10

(tppo) A:\Desktop\tpo\lab2.tpo\server>python tpo_server_5121.py 127.0.0.1 12345
[STARTING]: Server is starting ...

(tppo) A:\Desktop\tpo\tpo>curl http://localhost:12345/blind/shift_percent
{"shift_percent":50}

(tppo) A:\Desktop\tpo\tpo>curl http://localhost:12345/blind?shift_percent=10
{"shift_percent":10}

(tppo) A:\Desktop\tpo\tpo>
```

Рисунок 8. Проверка работы API с “curl”

Проверка работы возможна со помощью curl (рис.8).

## 5. Заключение

В рамках лабораторных работ успешно разработаны программы для управления «жалюзи». Результаты лабораторных работ соответствуют всем требованиям. Для лабораторной работы №1 используется протокол TCP для обеспечения связи между сервером и клиентом. В лабораторной работе № 2 сервис поддерживает управление «устройством» с помощью REST API (кроме уведомлений).

### Список использованных источников

1. Что такое модель TCP/IP протокола управления передачей данных? URL: <https://www.fortinet.com/ru/resources/cyberglossary/tcp-ip> (Дата обращения: 10.10.2022)
2. TCP И UDP – В ЧЕМ РАЗНИЦА? URL: <https://wiki.merionet.ru/seti/23/tcp-i-udp-v-chem-raznica/> (Дата обращения: 10.10.2022)
3. Socket Programming in Python (Guide). URL: <https://realpython.com/python-sockets/> (Дата обращения: 25.10.2022)
4. An Intro to Threading in Python. URL: <https://realpython.com/intro-to-python-threading/> (Дата обращения: 25.10.2022)
5. Threading Mutex Lock in Python. URL: <https://superfastpython.com/thread-mutex-lock/> (Дата обращения: 25.10.2022)
6. Python Multithreading and Multiprocessing Tutorial. URL: <https://www.toptal.com/python/beginners-guide-to-concurrency-and-parallelism-in-python> (Дата обращения: 25.10.2022)
7. Введение в Rest API: что это простыми словами. URL: <https://mcs.mail.ru/blog/vvedenie-v-rest-api> (Дата обращения: 13.11.2022)
8. Python and REST APIs: Interacting With Web Services. URL: <https://realpython.com/api-integration-in-python/> (Дата обращения: 05.12.2022)
9. Типы HTTP-запросов и философия REST. URL: <https://habr.com/ru/post/50147/> (Дата обращения: 13.11.2022)

## Приложение А

### Программный код северной программы лаб 1: tppo\_server\_5121.py

```
import sys
import socket
import threading
from blinds import *
import os

class Server():
    def __init__(self, host, port, device):
        self.host = host
        self.port = port
        self.SIZE = 1024
        self.FORMAT = "utf-8"
        self.device = device
        self.clients = []

    def setup_socket(self):
        print("[STARTING]: Server is starting ...")
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.bind((self.host, self.port))
        self.sock.listen()
        print(f"[LISTENING]: Server is listening on {host}:{port}")

    def response_get(self, params):
        if params == "Shift_Percent":
            msg = f"Current {params}: {self.device.get_shift_percentage()}%"
        elif params == "Flux_Percent":
            msg = f"Current {params}: {self.device.get_luminous_flux_percentage()}%"
        elif params == "Current_Ill":
            msg = f"{params}: {self.device.get_current_illumination()} lx"
        return msg

    def response_set(self, params, values):
        if params == "Shift_Percent":
            self.device.set_shift(values)
        elif params == "Flux_Percent":
            self.device.set_luminous_flux(values)
        return f"{params} was successful changed to {values}%"

    def handle_client(self, conn, addr):
        print(f"[NEW CONNECTION] {addr} connected.")
        connected = True
        while connected:
            recv_data = conn.recv(self.SIZE).decode(self.FORMAT)
            recv_data = recv_data.title().split()
            lock.acquire()
            if not recv_data:
                print(f"Closing connection to {addr}")
                connected = False
                conn.close()
                self.clients.remove(conn)
            else:
                if recv_data == ["Exit"]:
                    print(f"Closing connection to {addr}")
                    connected = False
                    self.clients.remove(conn)
                    conn.close()
                elif recv_data[0] == "Get":
                    msg = self.response_get(params = recv_data[1])
                    try:
                        conn.send(msg.encode(self.FORMAT))
                    except socket.error:
                        print(f"Closing connection to {addr}")
                        connected = False
                        self.clients.remove(conn)
                elif recv_data[0] == "Set":
                    msg = self.response_set(params = recv_data[1], values = recv_data[2])
                    try:
                        conn.send(msg.encode(self.FORMAT))
                    except socket.error:
                        print(f"Closing connection to {addr}")
                        connected = False
                        self.clients.remove(conn)
            lock.release()
```

```

def notify_changes(self):
    device_path = "../server/blinds.txt"
    cached_stamp = os.stat(device_path).st_mtime
    while True:
        stamp = os.stat(device_path).st_mtime
        if(stamp != cached_stamp):
            cached_stamp = stamp
            lock.acquire()
            shift_percentage = device.get_shift_percentage()
            luminous_flux_percentage = device.get_luminous_flux_percentage()
            current_illumination = device.get_current_illumination()
            lock.release()
            if (blinds_params['current_shift'] != shift_percentage):
                blinds_params['current_shift'] = shift_percentage
                for client in self.clients:
                    client.send(f"Shift percentage has been changed to {shift_percentage}%".encode(self.FORMAT))
            elif (blinds_params['current_luminous_flux'] != luminous_flux_percentage):
                blinds_params['current_luminous_flux'] = luminous_flux_percentage
                for client in self.clients:
                    try:
                        client.send(f"Luminous fluxpercentage has been changed to {luminous_flux_percentage}%".encode(self.FORMAT))
                    except socket.error:
                        print(f"Closing connection to {addr}")
                        connected = False
                        self.clients.remove(conn)
            elif (blinds_params['current_illumination'] != current_illumination):
                blinds_params['current_illumination'] = current_illumination
                for client in self.clients:
                    try:
                        client.send(f"Current illumination has been changed to {current_illumination} lx".encode(self.FORMAT))
                    except:
                        print(f"Closing connection to {addr}")
                        connected = False
                        self.clients.remove(conn)

def check_conn(self, conn):
    connExists = False
    for client in self.clients:
        if (str(conn) == str(client)):
            connExists = True
            break
    return connExists

def run(self):
    self.setup_socket()
    notify_thread = threading.Thread(target=self.notify_changes)
    notify_thread.daemon = True
    notify_thread.start()
    while True:
        conn, addr = self.sock.accept()
        if (self.check_conn(conn) == False):
            self.clients.append(conn)
        cli_thread = threading.Thread(target=self.handle_client, args=(conn, addr))
        cli_thread.daemon = True
        cli_thread.start()

lock = threading.Lock()
device = Blinds()
shift_percentage = device.get_shift_percentage()
luminous_flux_percentage = device.get_luminous_flux_percentage()
current_illumination = device.get_current_illumination()
blinds_params = {
    "current_shift": shift_percentage,
    "current_luminous_flux": luminous_flux_percentage,
    "current_illumination": current_illumination
}

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print(f"Usage: {sys.argv[0]} <host> <port>")
        sys.exit(1)
    host, port = sys.argv[1], int(sys.argv[2])
    server = Server(host, port, device)
    server.run()

```

## Приложение Б

### Программный код: blinds.py

```
import time

class Blinds():
    path = "../server/blinds.txt"
    @staticmethod
    def read_database(path):
        with open(path, "r") as file:
            data = file.readlines()
            file.close()
        return data

    def get_params(self, path = path):
        data = self.read_database(path)
        shift_percentage = int(data[0].split()[-1])
        luminous_flux_percentage = int(data[1].split()[-1])
        current_illumination = int(data[2].split()[-1])
        return shift_percentage, luminous_flux_percentage, current_illumination

    def __init__(self):
        self.shift_percentage, self.luminous_flux_percentage, self.current_illumination = self.get_params()

    def get_shift_percentage(self, path = path):
        data = self.read_database(path)
        self.shift_percentage = int(data[0].split()[-1])
        return self.shift_percentage

    def get_luminous_flux_percentage(self, path = path):
        data = self.read_database(path)
        self.luminous_flux_percentage = int(data[1].split()[-1])
        return self.luminous_flux_percentage

    def get_current_illumination(self, path = path):
        data = self.read_database(path)
        self.current_illumination = int(data[2].split()[-1])
        return self.current_illumination

    def set_shift(self, new_shift, path = path):
        """
        Установить проценты сдвига полотна

        """
        data_ = self.read_database(path)
        shift = data_[0].split()
        shift[-1] = str(new_shift)
        data_[0] = ' '.join(shift)+'\n'

        with open(path, 'w') as file:
            file.writelines(data_)
            file.close()
        self.shift_percentage = str(new_shift)

    def set_luminous_flux(self, new_luminous_flux, path = path):
        """
        Установить проценты сдвига полотна

        """
        data_ = self.read_database(path)
        luminous_flux = data_[1].split()
        luminous_flux[-1] = str(new_luminous_flux)
        data_[1] = ' '.join(luminous_flux)+'\n'

        with open(path, 'w') as file:
            file.writelines(data_)
            file.close()
        self.luminous_flux_percentage = str(new_luminous_flux)
```

## Приложение В

### Программный код клиентской программы лаб 1: tpro\_client\_5121.py

```
import socket
import threading
import sys

class Client:
    def __init__(self, host, port):
        self._host = host
        self._port = port
        self.SIZE = 1024
        self.FORMAT = "utf-8"
        self._socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connected = False

    def _input_and_send_loop(self):
        try:
            while True:
                command = input()
                cmd = command.title().split()
                if not self.check_cmd(cmd):
                    self._socket.send(command.encode(self.FORMAT))
                    if cmd == ["Exit"]:
                        print(f"Closing connection!")
                        self.connected = False
                        self._socket.close()
                        sys.exit()
        except:
            self.connected = False
            sys.exit()

    def check_cmd(self, cmd):
        self.invalid_cmd = False
        if cmd == []:
            print("Please, enter a command")
            self.invalid_cmd = True
        elif len(cmd) == 1:
            if cmd[0] not in ["Get", "Set", "Exit"]:
                print("SyntaxError: Not supported this command!")
                self.invalid_cmd = True
            elif cmd[0] in ["Get", "Set"]:
                print("SyntaxError: Put a parameter's name")
                self.invalid_cmd = True
        elif len(cmd) == 2:
            if cmd[0] == "Set" and cmd[1] in ["Shift_Percent", "Flux_Percent", "Current_Ill"]:
                print(f"SyntaxError: Give a value for param {cmd[1]}")
                self.invalid_cmd = True
            elif cmd[0] != "Get" or cmd[1] not in ["Shift_Percent", "Flux_Percent", "Current_Ill"]:
                print("SyntaxError: Not supported this command!")
                self.invalid_cmd = True
        elif len(cmd) == 3:
            if cmd[0] != "Set":
                print("SyntaxError: Not supported this command!")
                self.invalid_cmd = True
            elif cmd[1] not in ["Shift_Percent", "Flux_Percent", "Current_Ill"]:
                print(f"SyntaxError: Not supported this parameter {cmd[1]}!")
                self.invalid_cmd = True
            elif not cmd[2].isdigit():
                print(f"SyntaxError: Give a value for param {cmd[1]}")
                self.invalid_cmd = True
        else:
            self.invalid_cmd = False
        return self.invalid_cmd

    def connect(self):
        self._socket.connect((self._host, self._port))
        self.connected = True
        print(f"[CONNECTED] Client connected to server at {self._host}:{self._port}")
        threading.Thread(target=self._input_and_send_loop).start()
        try:
            while self.connected:
                msg = self._socket.recv(self.SIZE).decode(self.FORMAT)
                if msg:
                    print(f"[SERVER] {msg}")
                else:
```

```

        sys.exit()
    except:
        sys.exit()

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print(f"Usage: {sys.argv[0]} <host> <port>")
        sys.exit(1)
    host, port = sys.argv[1:3]
    client = Client(host, int(port))
    client.connect()

```

## Приложение Г

### Программный код северной программы лаб 2: tppo\_server\_5121.py

```

from flask import Flask, request, abort
from flask_socketio import SocketIO
import threading
from blinds import *
import os
import sys

app = Flask(__name__)
socketio = SocketIO(app)
mutex = threading.Lock()

device = Blinds()
current_shift_percent = device.get_shift_percentage()
current_flux_percent = device.get_luminous_flux_percentage()
current_ill = device.get_current_illumination()

@app.route('/blind/<string:param>')
def main(param):
    global mutex
    if (param == "shift_percent"):
        mutex.acquire()
        value = device.get_shift_percentage()
        mutex.release()
    elif (param == "flux_percent"):
        mutex.acquire()
        value = device.get_luminous_flux_percentage()
        mutex.release()
    elif (param == "current_ill"):
        mutex.acquire()
        value = device.get_current_illumination()
        mutex.release()
    else:
        abort(404)
        return "Not supported this parameters!!"
    return {param: value}

@app.route('/blind')
def get_query_string():
    shift_percent = request.args.get('shift_percent')
    flux_percent = request.args.get('flux_percent')
    global mutex
    if (shift_percent):
        mutex.acquire()
        device.set_shift(shift_percent)
        mutex.release()
        return {"shift_percent": shift_percent}
    elif (flux_percent):
        mutex.acquire()
        device.set_luminous_flux(flux_percent)
        mutex.release()
        return {"flux_percent": flux_percent}
    else:
        abort(404)
        return "Not supported this parameters!!"

@socketio.on('connect')
def test_connect(auth):
    print("A new client has connected!")
    global notify_changes_thread

```



```

    if not notify_changes_thread.is_alive():
        notify_changes_thread.start()

@socketio.on('disconnect')
def test_disconnect():
    print('Client disconnected')

def notify_changes():
    global current_shift_percent
    global current_flux_percent
    global current_ill
    global socketio
    device_path = "../server/blinds.txt"
    cached_stamp = os.stat(device_path).st_mtime
    while(True):
        stamp = os.stat(device_path).st_mtime
        if(stamp != cached_stamp):
            cached_stamp = stamp
            mutex.acquire()
            shift_percent = device.get_shift_percentage()
            flux_percent = device.get_luminous_flux_percentage()
            illumination = device.get_current_illumination()
            mutex.release()

            if (current_shift_percent != shift_percent):
                current_shift_percent = shift_percent
                socketio.emit("notify changes", {"params": "shift_percent", "value": current_shift_percent},
broadcast=True)
            elif (current_flux_percent != flux_percent):
                current_flux_percent = flux_percent
                socketio.emit("notify changes", {"params": "flux_percent", "value": current_flux_percent},
broadcast=True)

            elif (current_ill != illumination):
                current_ill = illumination
                socketio.emit("notify changes", {"params": "illumination", "value": current_ill},
broadcast=True)

notify_changes_thread = threading.Thread(target=notify_changes)

if (__name__ == "__main__"):
    if len(sys.argv) != 3:
        print(f"Usage: {sys.argv[0]} <host> <port>")
        sys.exit(1)
    print("[STARTING]: Server is starting ...")
    host, port = sys.argv[1], int(sys.argv[2])
    socketio.run(app, host=host, port=port, debug=False)

```

## Приложение Д

### Программный код клиентской программы лаб 2: tppo\_client\_5121.py

```

import socketio
import requests
import sys

sio_client = socketio.Client()

def get_param(param):
    global host
    global port
    api_url = f"http://{host}:{port}/blind/{param}"
    return requests.get(api_url)

def set_param(param, value):
    global host
    global port
    api_url = f"http://{host}:{port}/blind?{param}={value}"
    return requests.get(api_url)

def check_cmd(cmd):
    invalid_cmd = False
    if cmd == []:
        print("Please, enter a command")
        invalid_cmd = True

```

```

elif len(cmd) == 1:
    if cmd[0] not in ["get", "set"]:
        print("SyntaxError: Not supported this command!")
        invalid_cmd = True
    elif cmd[0] in ["get", "set"]:
        print("SyntaxError: Put a parameter's name")
        invalid_cmd = True

elif len(cmd) == 2:
    if cmd[0] == "set" and cmd[1] in ["shift_percent", "flux_percent", "current_ill"]:
        print(f"SyntaxError: Give a value for param {cmd[1]}")
        invalid_cmd = True
    elif cmd[0] != "get" or cmd[1] not in ["shift_percent", "flux_percent", "current_ill"]:
        print("SyntaxError: Not supported this command!")
        invalid_cmd = True
elif len(cmd) == 3:
    if cmd[0] != "set":
        print("SyntaxError: Not supported this command!")
        invalid_cmd = True
    elif cmd[1] not in ["shift_percent", "flux_percent", "current_ill"]:
        print(f"SyntaxError: Not supported this parameter {cmd[1]}!")
        invalid_cmd = True
    elif not cmd[2].isdigit():
        print(f"SyntaxError: Give a value for param {cmd[1]}")
        invalid_cmd = True
else:
    invalid_cmd = False
return invalid_cmd

@sio_client.event
def connect():
    global host
    global port
    print("[CONNECTED] Client connected to server successfully!")
    print("Please, enter a command")
    while(True):
        command = input()
        cmd = command.lower().split()
        if not check_cmd(cmd):
            if cmd[0] == "get":
                result = get_param(cmd[1]).json()
                if cmd[1] == "current_ill":
                    print(f"[SERVER]: Current illumination is {result[cmd[1]]} lx")
                else:
                    print(f"[SERVER]: Current {cmd[1]} is {result[cmd[1]]}%")
            if cmd[0] == "set":
                result = set_param(cmd[1], cmd[2]).json()
                if cmd[1] == "current_ill":
                    print(f"[SERVER]: {cmd[1]} was successful changed to {result[cmd[1]]} lx")
                else:
                    print(f"[SERVER]: {cmd[1]} was successful changed to {result[cmd[1]]}%")
            if cmd[0] == "exit":
                print("Disconnected!")
                break

@sio_client.on("notify changes")
def get_notice_changes(data):
    if data['params'] == "illumination":
        print(f"[SERVER]: {data['params']} has been changed to {data['value']} lx")
    else:
        print(f"[SERVER]: {data['params']} has been changed to {data['value']}%")

@sio_client.event
def disconnect():
    print("Disconnected!")

if (__name__ == "__main__"):
    if len(sys.argv) != 3:
        print(f"Usage: {sys.argv[0]} <host> <port>")
        sys.exit(1)
    host, port = sys.argv[1], int(sys.argv[2])
    sio_client.connect(f"http://{host}:{port}")

```