МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего

образования

«Санкт-Петербургский государственный университет аэрокосмического приборостроения»

Кафедра информационно-сетевых технологий

Дисциплина «Интернет вещей»

Автор – ассистент Д.О. Шевяков

Лабораторная работа № 3

ОРГАНИЗАЦИЯ ПОЛУЧЕНИЯ И ВЫВОДА ДАННЫХ С УДАЛЕННОГО ОБОРУДОВАНИЯ

Цель работы – приобретение навыков организации получения данных с удаленного оборудования.

Методические указания

Для организации обмена данными с оборудованием обычно использую GET-запросы, в которых с оборудования приходит информация для мониторинга, а от пользователя команды для управления.

В качестве оборудования будем использовать функцию, которая будет автоматически обновлять поля нашего объекта класса. В реальных условиях, между системой и физической вещью необходимо будет выстраивать взаимодействие при помощи веб-запросов, также как и между системой и интерфейсом. На рисунке ниже показана функция эмуляции, которая обновляет поле объекта класса, задавая ему случайно значение в заданном диапазоне.

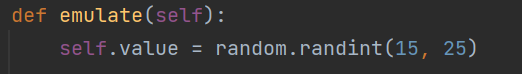


Рисунок 1 – Функция эмуляции для вещи Sensor

Эмулятор готов, теперь напишем к нему функцию подключения на стороне сервера. Для этого создадим функцию по аналогии с «hello\_world», но укажем другой путь роутинга, в моем случае – это «/connect». Внутри этой функции вызовем метод connect нашей вещи и передадим его результат в return. Для этого модифицируем метод connent таким образом, чтобы он возвращал данные, которые обновляются функцией emulate. Вызовем в методе connect метод emulate. После этого необходимо упаковать данные в json и передать, как результат выполнения функции роутинга. Функция роутинга и модифицированный метод connect представлены ниже.

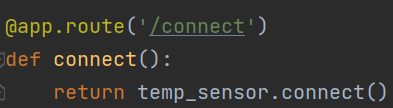


Рисунок 2 – Код функции connect

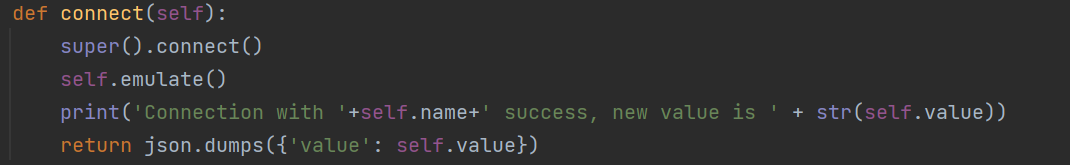


Рисунок 3 – Метод connect вещи Sensor

Для отображения информации, получаемой от эмулятора вещи необходимо разработать веб-страницу с полями для заполнения и ajax – запрос для получения этих данных из системы. Ниже представлен пример страницы с полем, в которое записываются данные и код ajax – запроса для получения данных.

https://psv4.userapi.com/c240331/u82898989/docs/d12/cdc128eed695/1.png?extra=84j4zaG9xCKTy8ZFqPfSUm6lMpJ8GqTKjx55Q33RIiVCl6fo5zju-ykXK7zg6t7CGKwPHLiH1nN0q8jwHTcfoP-NxQNuGjpXEHEq9v4fz5ybg3UzKdU_buPKxt7B5TMJII-F7jDbgtvXy6pE148

Рисунок 4 – Страница эмулятора вещи Sensor



Рисунок 5 – Код ajax – запроса для отправки данных

Структура этого запроса следующая:

* type – это тип запроса, в нашем случае – GET
* url – адрес, на который будет отправлен запрос
* dataType – тип данных, который ожидаем от сервера, в нашем случае - json
* contentType– тип передаваемых данных, в нашем случае – json
* data –данные запроса, которые будут переданы в систему.
* success – это функция, которая будет исполнена в случае, если запрос будет успешно исполнен и получит ответ. В нашем случае ответ – это json, в котором передаётся всего один параметр value, который передается в элемент с соответствующим названием.

Далее необходимо структурировать наш интерфейс для удобства работы с ним. Запишем наш js-код в отдельный файл, добавив к нему функцию SetInterval, которая будет вызывать нашу функцию подключения каждую секунду, чтобы подключение выполнялось регулярно, и поместим его в папку js в директории static, нашу страницу поместим в папку templates, добавим ссылки на соответствующие файлы, а также подключим ajax для корректной работы js-скрипта. В конце добавим стилей в папку css, чтобы эмулятор выглядел опрятней. Код страницы эмулятора можно увидеть ниже.



Рисунок 6– html-код страницы эмулятора

Далее нам необходимо запустить наш интерфейс на сервере, чтобы он корректно работал с нашей системой. Для этого поменяем return нашей функции «hello\_world» на return render\_template(‘sensor\_emulator.html’). Для этого в первую строку файла app.py надо добавить render\_template. Теперь при заходе по ссылке 127.0.0.1:5000 будет рендерится наш интерфейс.

Пока что наша функция не может обратиться к цифровому двйонику нашей вещи, поскольку он объявлен локально внутри функции hello\_world, поэтому перенесем объявление в начало скрипта, чтобы объект стал глобальным. Результат работы функции представлен ниже. Можно заметить, что сообщения о создании вещи теперь показываются перед запуском сервера.

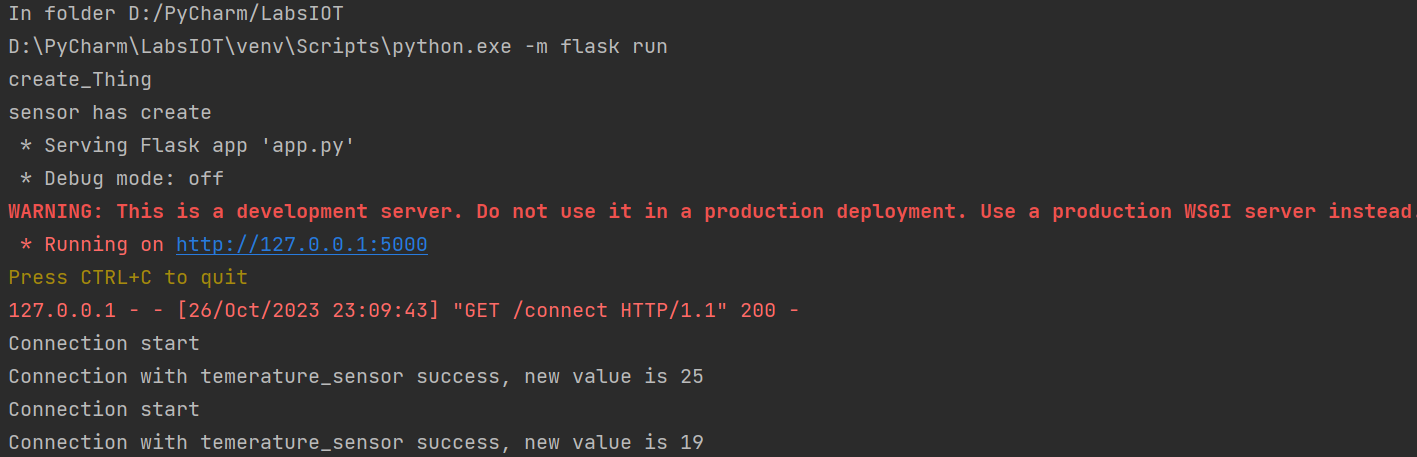


Рисунок 7 – Результат подключения

Код измененных модулей прикреплён в приложениях

Задание

Реализовать эмуляторы вещей для всех классов, созданных в предыдущей лабораторной. Разработать один или несколько интерфейсов, в которые будут передаваться данные от эмуляторов. В рамках данной лабораторной работы необходимо передать в интерфейс только данные для мониторинга. Передача команд, которые осуществляют управление вещью, будет рассмотрена в следующей лабораторной. Для каждой вещи необходимо реализовать собственный ajax-запрос и собственную функцию подключения.

Содержание отчёта:

1. Титульный лист
2. Задание
3. Код классов и приложения
4. Скриншоты
5. Выводы

Приложение 1 Код app.py

from flask import Flask, request, render\_template

import things

app = Flask(\_\_name\_\_)

temp\_sensor = things.Sensor('C', 'temerature\_sensor')

@app.route('/connect')

def connect():

return temp\_sensor.connect()

@app.route('/')

def hello\_world(): # put application's code here

return render\_template('sensor\_emulator.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

Приложение 2 Код things.py

import abc

import json

import random

class Thing(abc.ABC):

def \_\_init\_\_(self, name):

self.name = name

print('create\_Thing')

@abc.abstractmethod

def connect(self, \*args):

print('Connection start')

class Sensor(Thing):

def \_\_init\_\_(self, unit, name):

super().\_\_init\_\_(name)

self.unit = unit

self.value = 0

print('sensor has create')

def connect(self):

super().connect()

self.emulate()

print('Connection with '+self.name+' success, new value is ' + str(self.value))

return json.dumps({'value': self.value})

def emulate(self):

self.value = random.randint(15, 25)

Приложение 3 Код index.js

setInterval(get\_data, 1000)

function get\_data() {

$.ajax({

type: 'GET',

url: '/connect',

dataType: 'json',

contentType: 'application/json',

data: {},

success: function (response) {

document.getElementById("value").value = response["value"]

}

});

}

Приложение 4 Код sensor\_emulator.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Sensor Emulator</title>

<link rel="stylesheet" type="text/css" href="../static/css/main.css">

<script src="http://code.jquery.com/jquery-3.3.1.js"></script>

<script src="../static/js/index.js"></script>

</head>

<body>

<div id="monitoring\_data">

<input type="text" id="value" placeholder="Значение">

</div>

</body>

</html>