07. Сетевое взаимодействие по REST

Практические задания.

Цель: научиться разрабатывать программы, обменивающиеся данными по сети.
Задача: Разработать программу чтения данных в формате JSON по протоколу HTTP.

№ 07.0

1. Запустите редактор **Geany** из раздела «Программирование» в главном меню.
2. Откройте в редакторе **Geany** пример программы, читающей данные в формате JSON по протоколу HTTP (у модуля ESPeasy на базе ESP8266) **`~/CodeClub-IoT/samples/http_json.rb`**.
3. Уточните у преподавателя *IP-адрес* модуля и измените его в исходном тексте программы.
4. Запустите программу на выполнение из раздела меню «Сборка», пункт «Execute» (выполнить) и проверьте её работу. Закройте терминальное окно после завершения программы.

№ 07.1 - JSON

1. Доработайте программу **`MeteoStation`**, чтобы она запрашивала недостающие данные об атмосферном давлении от модуля ESPeasy на базе ESP8266 и выводила их на экран вместе с показаниями локального датчика DHT11.
2. Протестируйте программу; запустите её и проверьте правильность работы.
3. Предусмотрите в программе обработку ситуации, когда удалённый источник данных (ESP8266) по какой-то причине недоступен.
4. Пример обработки превышения времени ожидания ответа (таймаута) — в программе **`samples/http_timeout.rb`**.

№ 07.2 - REST

1. Откройте в редакторе **Geany** программу **`~/CodeClub-IoT/samples/http_server.rb`**. Это пример простого сервера, который в цикле принимает запросы по протоколу HTTP и возвращает текущее время, оформляя его в виде правильного HTTP-ответа.
2. Запустите эту программу из редактора и проверьте, как она работает: обратитесь из браузера по адресу **`http://localhost:5000`**.
3. Закройте терминальное окно выполнения программы (иначе при следующем запуске программы будет выдана ошибка, что порт 5000 уже занят).
4. На основе этого примера доработайте программу **`MeteoStation`**: добавьте в неё метод **`http_server`**, который будет принимать запросы в соответствии с рекомендациями REST и выдавать текущие значения указанных датчиков:
   `GET /meteostation/sensors/temperature` → `значение температуры`
   `GET /meteostation/sensors/humidity` → `значение влажности`
   `GET /meteostation/sensors/pressure` → `значение давления`
   *(Можно для быстроты скопировать последнюю версию программы **`MeteoStation`** в новый файл и удалить из неё описания и методы опроса датчика DHT11, чтобы сосредоточиться только на разработке и проверке метода **`http_server`**).*
5. Для выделения из запроса адреса (URL), удобно применить метод **`split()`**, который разделит строку на части и поместит их в элементы массива:

```
strings = request.split(" ")
# "GET /m/s/t HTTP/1.1\r\n" → ["GET", "/m/s/t", "HTTP/1.1"]
path = strings[1].split("/") # → ["", "m", "s", "t"]
```

6. Протестируйте, как поведёт себя ваша программа, если к ней обратятся с несуществующим адресом URL, например: `/meteostation/sensors/altitude` или `/meteostation/show/pressure`.
7. Измените программу **MeteoStation**, чтобы она принимала и другие управляющие команды. Например:
   PUT /meteostation/log → записать значения датчиков в файл протокола
   PUT /meteostation/stop → закончить выполнение программы

8. Для проверки произвольных HTTP-запросов (а не только GET и POST, которые может отправлять браузер), удобно написать тестовую клиентскую программу по примеру **~/CodeClub-IoT/samples/http_json.rb**. Например, для выдачи команды PUT можно воспользоваться таким кодом:
```
uri = URI.parse("http:localhost:5000/meteostation/stop")
http = Net::HTTP.new(uri.host, uri.port)
put_request = Net::HTTP::Put.new(uri.request_uri)
```
9. Какие ещё команды полезно предусмотреть для управления метеостанцией?