Проект «Ми-Шень»

Предварительное ТЗ на Управляющее Устройство

Версия 1.5

Внесены замечания от разработчиков и заказчика:

- уточнения по комплектующим мишени
- уточнения по виду топиков
- использована логика и переменные от Lua кода прошивки
- общие орг замечания
- исправление ошибок (убрана описка про MQTT коннектор)

1. Общее описание

Проект Ми-Шень реализует систему управления несколькими электронными устройствами по протоколу MQTT. Электронное устройство, далее — **Мишень**, устанавливает соединение с **Управляющим Устройством**, далее — УУ образуя двунаправленный канал связи, по которому УУ передаёт команды Мишени, а Мишень уведомляет УУ о наступлении ряда событий. В результате УУ согласованно управляет и принимает уведомления от нескольких Мишеней сразу. В УУ реализована сложная логика управления, которую можно запрограммировать и сохранить в профиль. В ходе работы пользователь выбирает один из профилей и стартует систему, запуская уникальный цикл работы, далее — **Тренировка**. В ходе и по окончании Тренировки — пользователю доступна информация о ней. Также, для всех прошедших Тренировок доступна **Статистика тренировок**.

Код проекта открытый. Лицензия не определена. Будет выложено ли на github.com/cadrspace (спросить Артёма — что для этого потребуется).
Часть проекта, общее описание или работы по данной тематике планируется осветить в инет-блогах и соцсетях.

1.1 Мишени

Мишень представляет собой автономное устройство, реализованное на платформе популярного семейства микроконтроллеров ESP. Мишень имеет датчик, индикацию и кнопку(тумблер) включения питания. А также биппер, сигнализирующий о попадании.

После включения питания мишень начинает устанавливать соединение. Установление соединения обозначается индикатором синего цвета («Connection»).

(В процессе установки соединения интенсивно мигает индикатор «Connection». После установки соединения этот индикатор горит постоянно. В случае потери соединения мишень вновь пытается установить соединение. Если соединение не устанавливается после 10ой

попытки — система перестаёт делать попытки соединения и запускает длительное мигание «Connection». Для дальнейшей работы необходимо перезагрузить мишень (включение/выключение питания).)

При установленном соединении мишень способна активироваться по сигналу от УУ. При активации загорается зеленый индикатор («Active») и раздается сигнал с биппера. В этом режиме мишень способна уведомлять о попадании в неё используя сигнал пьезодатчика.

~~Для уведомления УУ о готовности пользователя к некоторому режиму работы на мишени есть кнопка «User Action». Нажатие на неё уведомляет УУ о запуске некоторого режима работы. Событие User Action может быть отправлено в любом состоянии мишени при условии наличия соединения с УУ.~~

Мишень умеет сигнализировать о критическом уровне питания, для уведомления пользователя о скором прекращении работы. Реализация аппаратная.

1.1.1 Взаимодействие с УУ по MQTT

Для коннекта в коде мишени константно прописывается hostname или IP адрес брокера (adrMQTT). Порт стандартный. Для коннекта используется авторизация по login/password. QoS минимальный (0), шифрования нет.

После коннекта мишень подписывается на сигнальный топик от УУ (app/appclient/signal).

Мишень также умеет публиковать сообщения timestamp и mishen_status, определяющие временной отрезок с момента старта активации и уведомления о нахождении в сети. Последнее необходимо УУ для информирования УУ об информации о устройстве, например его ID, имя, серийный номер или любую другую информацию.

Требуется узнать — возможно ли определение ID устройства при отправке публикаций и возможно ли обнаружение этого ID при коннекте-дисконнекте. При подтверждении наличия возможности можно отказаться от этого топика

Публикация происходит в топики mishen/timestamp и mishen/mishen_status соответственно.

На данный момент не известно — как будут идентифицироваться источники публикаций. Требуется курить MQTT протокол на эту тему.

После коннекта мишень подписывается на единый для всех мишеней широковещательный топик /app/app_client и начинает обрабатывать все принимаемые сообщения выделяя из них адресованные себе и обрабатывая только их.

Мишень получает start и запускает таймер. При попадании значение таймера отсылается УУ. Таймер продолжает работать. Каждое следующее попадание при этом инициирует отправку сообщения с значением таймера. УУ само определяет когда мишень должна остановиться (сообщение stop).
Так как start отправляется в общий топик, сообщение должно сопровождаться ID мишени которое мишень проверяет при получении сообщения и отбрасывает в случае несовпадения.

Примечание: требуется подумать над проблемой избыточного энергопотребления при

обработке start в общем топике. Возможно лучше сделать индивидуальный сигнальный топик для каждой из мишеней. В этом случае мишени не будут срываться при каждом «не своем» сообщении.

1.1.2 Конструкция

Пластиковый короб с передней стенкой из оргстекла. Размеры 300х380мм, толщина 30-40мм.
Крепление — 4 винта с гайками.
На задней стенке два отверстия для крепления на стену.
~~Отсек питания (3 AAA? Акуум? DctoDC?~~).
Питание осуществляется от LiIon аккумулятора на >=600мА.
Заряд осуществляется через microUSB и схему зарада этих типов аккумуляторов.
Предположительно китайский шилд или свой велосипед на MCP...потом уточню.
Биппер располагается сбоку.
На дне корпуса — приспособление для устойчивого расположения на горизонтальной плоскости.
Есть предложение крепления для установки на штатив:

(*отверстие под 20 профиль заложить, позволит крепиться к плоской платформе например при установке "на землю"(устойчивость + компактность при транспортировке)или к любой высоты штативу с куском 20мм профильной трубы на конце*)

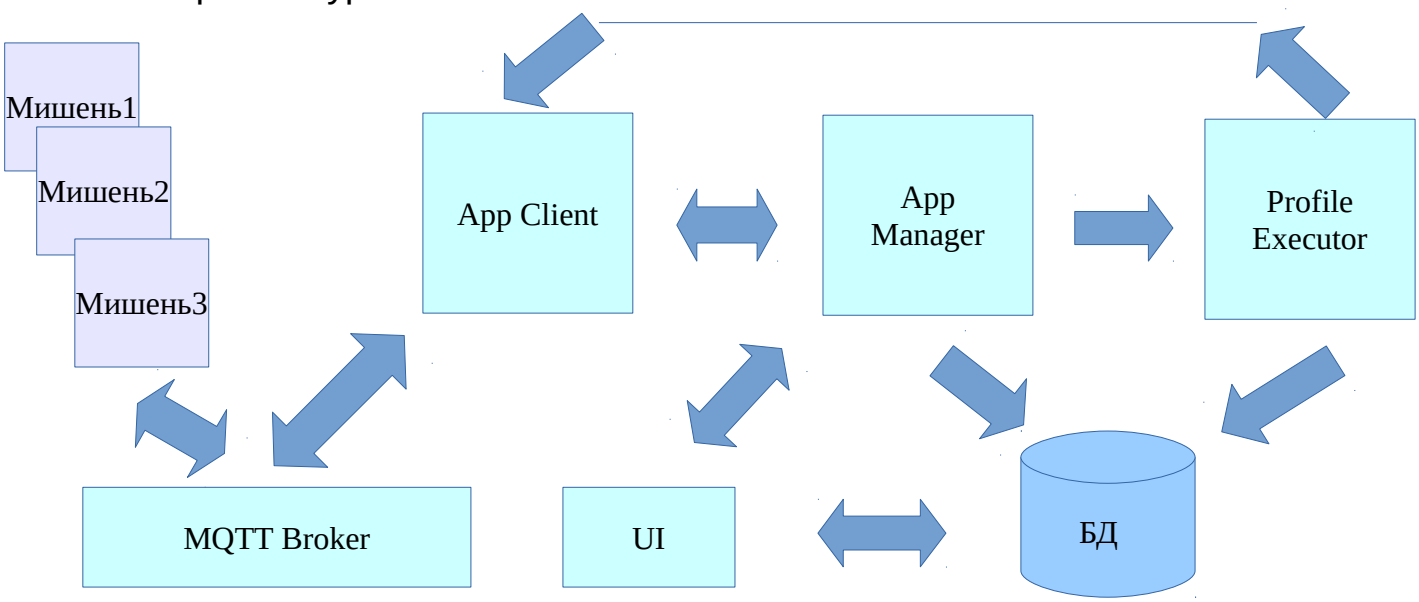
(требуется перечислить комплектующие для формирования примерной стоимости мишени)

1.2 УУ

УУ представляет собой исполняемую программу, умеющую работать с несколькими MQTT клиентами (через подписки и публикации), взаимодействовать с пользователем через UI (в частности отображать информацию в виде списков, таблиц, графиков, надписей и предоставлять соответствующие элементы управления и views), работать с локальной БД и иметь встроенный режим выполнения инструкций по алгоритмам, которые определяет пользователь.

Программно-аппаратная латформа УУ не существенна, но предпочтение отдается реализациям под Android 4.4+ и веб решению, способному работать на десктоп-машине. Все варианты должны поддержовать WiFi соединение, поскольку мишени работают только под этот тип сетевой организации.
Предполагаемые платформы: Linux, Android 4.4+
Предполагаемые языки: NodeJS, Python, Go
Предполагаемые MQTT реализации: Mosquitto или Paho

1.2.1 Архитектура УУ



App Client обеспечивает подписку на временной и статусный топики мишеней, а также публикует в сигнальный топик.

App Client — обеспечивает обертку кодирование/декодирование команд от УУ к каждой мишени широковещательным манером. (Внутри команды содержится ID получателя и отправителя, а также ID команды и дополнительной информация для неё)

MQTT предполагает наличие брокера и клиентов-издателей и подписчиков. Брокер работает на стороне УУ, клиентами являются все мишени и AppClient со стороны УУ.

Используются два типа подписок:
mishen/timer_done
mishen/user_active
mishen/gotcha (попадание)
mishen//bad_power

app/set_active
app/set_deactive
app/set_timer

App Manager — реализует общую логику управления режимами работы УУ. Отвечает за хранение текущего режима работы. Производит первичную обработку сообщений,

приходящих от AppClient и от UI.

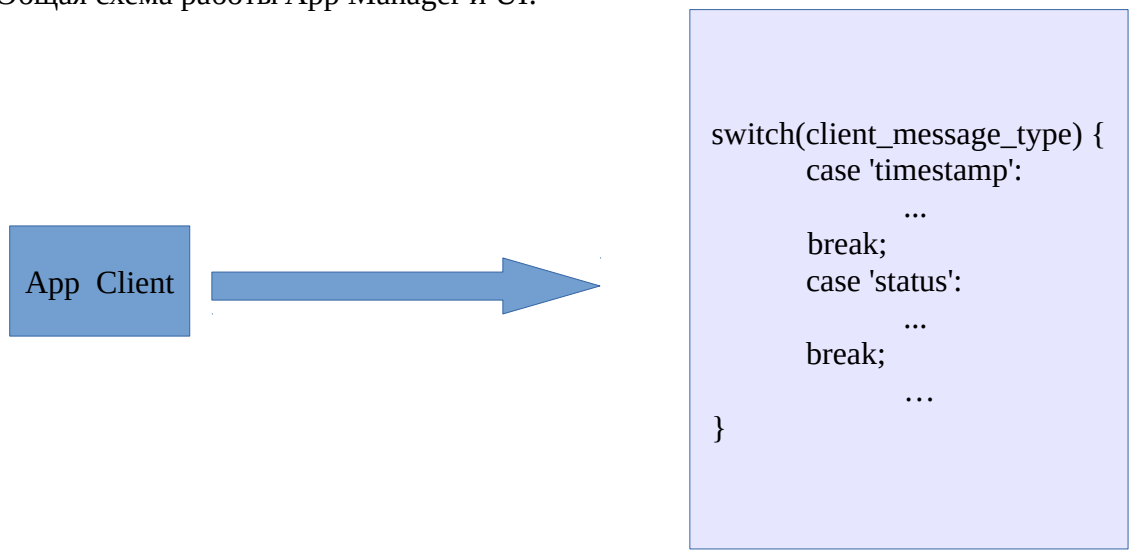
UI — необходимое для того чтобы показывать результат и принимать команды от пользователя при его работе с УУ. UI работает с БД и может запускать или останавливать процесс тренировки. Реализация зависит от платформы: web или Android.

БД — локальная база данных, скорее всего SQLite. В ней хранятся конфигурация, сохраненные алгоритмы работы Тренировок, а также результаты Тренировок, принятые от Мишеней. Можно сделать через абстрактный слой.

Profile Executor

- принимает и хранит тренировочные сообщения от мишеней
- читает по очереди алгоритм текущего профиля
- посылает команды мишеням сразу через App Client
- организует триггер ожидания комбинации событий приостанавливая выполнение алгоритма
- реализует искусственную задержку перед переходом к следующей инструкции алгоритма

Общая схема работы App Manager и UI:



```
switch(ui_message_type) {
        case 'update':
        ...
        break;
        case 'delete':
        ...
        break;
        ...
        case 'create':
        ...
        break;
        case 'read':
        ...
        break;
        ...
}
```

1.2.2 Формат записи алгоритмов Тренировок

Алгоритм это ряд сообщений, отправляемых мишеням и и синхронизация с её поведением. Для определения логики отправки необходим способ записи в виде понятного текстового кода. При необходимости в дальнейшем можно создать более удобные конструкции для формирования этого кода автоматически.

Общий подход к программированию профиля Тренировки:

Пользователь выбирает профиль Тренировки. Профиль это выполнение рабочих циклов несколько раз.

Рабочий цикл
В каждом из циклов активируются несколько мишеней. При попадании или при окончании таймаута мишени — алгоритм принимает решение о продолжении работы или выхода из рабочего цикла.
При запуске цикла должен быть определен параметр (или структура), благодаря которому алгоритм будет понимать — какие из мишеней ему доступны, а также хранить текущее состояние.

Реализация рабочего цикла это выполнение ряда микроинструкций:
SendToClient – посылка сообщения мишени
WaitEvent – ожидание сообщения системы.
Delay – запуск таймера, приостанавливающего работу цикла.
StartAppTimer — запуск таймера по частному событию, в принципе замена Delay

Пример:

Задача последовательно зажигать мишени (3 шт), переключая их активность при попадании или при окончания времени стрельбы.

// Деактивация всех мишеней SendToClient(Client_Id, Deactivate)

```
SendToClient(101, 'Stop')
SendToClient(102, 'Stop')
SendToClient(103, 'Stop')

// Активировать 1 мишень
SendToClient(101, 'start')
//запуск срабатываение события через временной период
StartAppTimer('TooLong101', SHOOT_TIMER)
// Ждем пока в нее не попадут или пока не кончится таймер
WaitEvent(101, 'time_stamp','TooLong101')
SendToClient(101, 'Stop')

// Активировать 2 мишень
SendToClient(102, 'start')
//запуск срабатываение события через временной период
StartAppTimer('TooLong102', SHOOT_TIMER)
// Ждем пока в нее не попадут или пока не кончится таймер
WaitEvent(103, 'time_stamp','TooLong102')
SendToClient(102, 'Stop')

// Активировать 3 мишень
SendToClient(103, 'start')
//запуск срабатываение события через временной период
StartAppTimer('TooLong103', SHOOT_TIMER)
// Ждем пока в нее не попадут или пока не кончится таймер
WaitEvent(103, 'time_stamp','TooLong103')
SendToClient(103, 'Stop')

Delay(5000)
```

Продолжение следует...