**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

**Факультет программной инженерии и компьютерной техники**

**КУРСОВОЙ ПРОЕКТ**

**ПО ДИСЦИПЛИНЕ «ВСТРОЕННЫЕ СИСТЕМЫ»**

**Блок SDK**

Выполнили:

Ларочкин Глеб Игоревич, группа P3400

Рыбаков Степан Дмитриевич, группа P3400

Преподаватель: Ключев Аркадий Олегович

Санкт-Петербург

2020

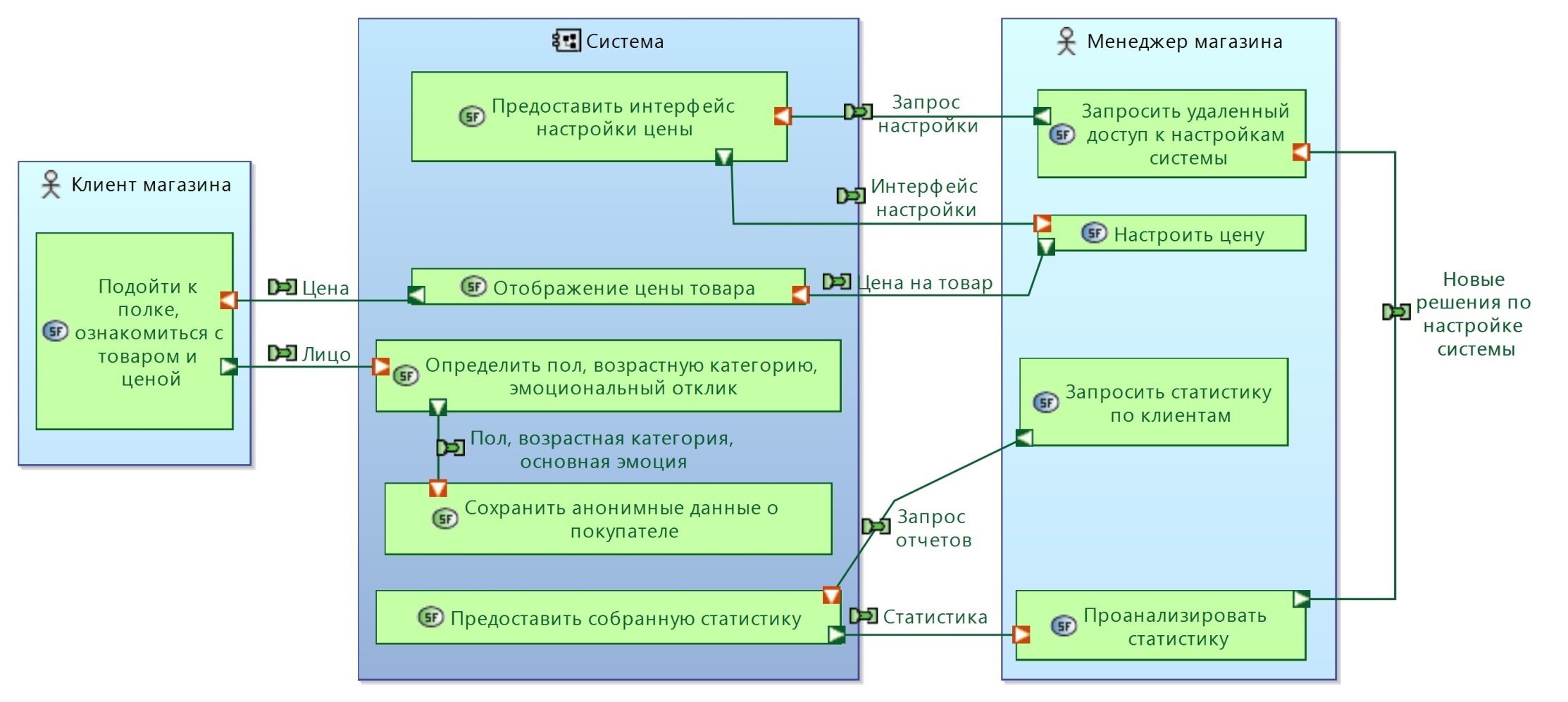
**Оглавление**

1. **Общее техническое задание к системе**
   1. **Описание системы**

Название разрабатываемой системы – «Умная полка». Система представляет собой магазинную полку, на которой размещается один или несколько электронных ценников, а также камера, кадры с которой анализируются для сбора анонимных данных о покупателях магазина (пол, возрастная категория, эмоциональный отклик). На основе собранных данных формируется статистка, которую менеджмент магазина может получить посредством Android приложения. Статистика может использоваться для улучшения вида упаковки товара, ценовой политики и организации торговых залов.

* 1. **Функциональные требования**

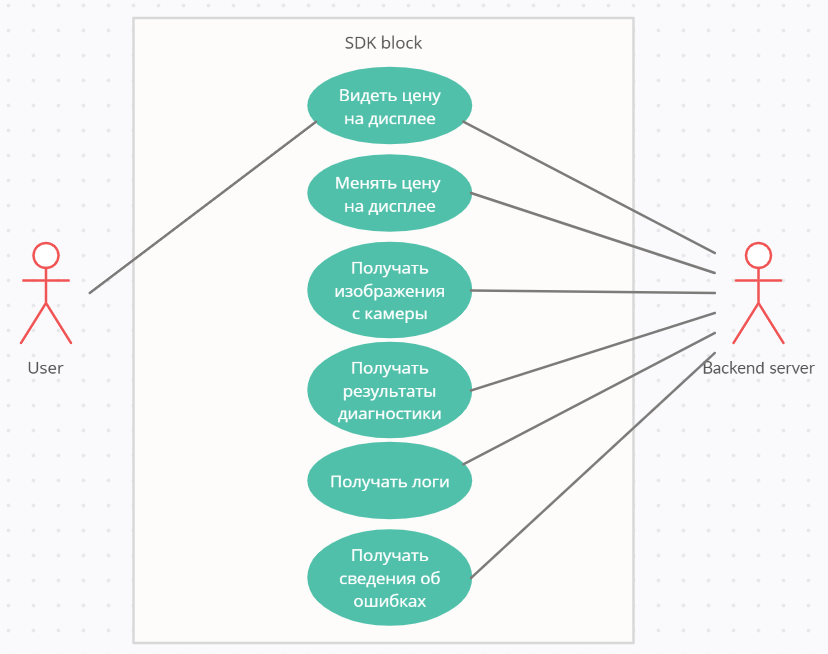
1. Система должна иметь 3 режима работы:
   1. Настройка. В этом режиме система взаимодействует с менеджером магазина и предоставляет функционал по установке цены на товар, которая будет демонстрироваться на электронном ценнике. Сбор данных и их анализ в этом режиме не осуществляется.
   2. Сбор данных. Система демонстрирует цену на товар, а также собирает анонимные данные о покупателях, которые подходят к полке.
   3. Предоставление собранной статистики. По запросу пользователя система предоставляет статистику, сформированную на основе собранных данных о покупателях.
2. Система должна собирать следующие анонимные данные о покупателях:
   1. Пол (женский, мужской).
   2. Возрастная категория (менее 11, 11 – 17, 18 – 24, 25 – 35, 36 – 45, 46 – 60, 61 – 75, старше 75).
   3. Эмоциональный отклик (удивление, радость, гнев, отвращение, грусть/спокойствие).
3. Система должна функционировать в соответствии с поведенческой моделью, представленной на рисунке 1.

****

*Рисунок 1. Схема взаимодействия системы с менеджером и клиентом магазина*

* 1. **Нефункциональные требования**

1. Применяемое аппаратное обеспечение должно включать в себя:
   * Стенд SDK-1.1M.
   * Компьютер общего назначения.
   * Мобильное устройство на базе Android.
   * Камера.
2. Требования к безопасности системы:
   * Настройка цены и просмотр статистики должны быть защищены от несанкционированного доступа.
   * Собираемые данные о покупателях (пол, возрастная группа, эмоциональный отклик) должны быть анонимны.
   * Система не должна сохранять фото/видео людей.
3. Интерфейсы:
   * Система должна предоставлять интерфейс для настройки системы и получения статистики на базе мобильного устройства Android.
   * Взаимодействие системы и пользователя должно осуществляться на русском языке.
   * Собираемые системой данные должны быть агрегированы и предоставляться менеджеру в виде таблиц, графиков и диаграмм.
4. Система должна собирать данные для статистики 1 раз в минуту.
5. Нештатные ситуации должны записываться во внутренний журнал с меткой даты и времени.
6. Должно быть разработано руководство пользователя системы, содержащее описание режимов работы системы и алгоритмов взаимодействия с ней в каждом из ее режимов.
7. **Описание блока SDK проекта**
   1. **Функциональные требования**
8. Блок должен позволять отображать цену товара на графическом дисплее.
9. Блок должен позволять менять цену товара на графическом дисплее.
10. Блок должен передавать данные (кадры) с камеры на сервер (backend) по расписанию.
11. Блок должен проводить диагностику по таймеру и/или запросу от сервера.
12. Блок должен передавать данные диагностики и логи по запросу от сервера или при обнаружении ошибки.



*Рисунок 2. Use-case диаграмма блока SDK.*

*Сюда можно еще активити бахнуть по следующей схеме:*

*Получает сигнал о смене цены от backend layer -> Отображает цену на графическом дисплее*

*Получает кадры с модуля камеры -> Настраивает TCP соединение с backend layer -> Передает кадры backend через соединение*

*Периодически проводит самодиагностику*

*Получает сигнал о диагностике -> Проводит диагностику*

*Находит ошибку -> отправляет сообщение на backend*

*Получает запрос на получение логов -> отправляет логи на backend*

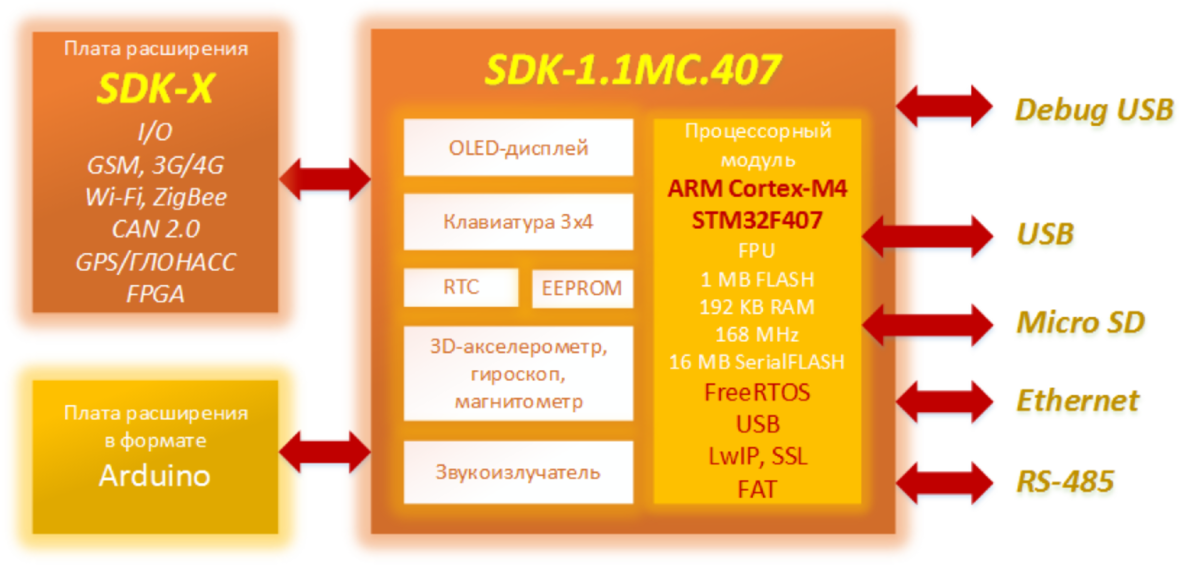
* 1. **Нефункциональные требования**

1. Программное обеспечение для блока SDK должно быть реализовано на языке C/C++.
2. В качестве стенда необходимо использовать стенд SDK-1.1M
3. Предполагается использование ОС для встраиваемых систем FreeRTOS
4. Для передачи данных предполагается использовать TCP/IP стек с открытым исходным кодом LwIP, предназначенный для встраиваемых систем.

Исходя из функциональных и нефункциональных требований поставлены следующие задачи:

1. Изучение возможностей стенда SDK-1.1M для реализации поставленных задач.
2. Изучение возможности LwIP для передачи данных на сервер.
3. Реализовать получение кадров с модуля камеры.
4. Реализовать API для передачи кадров и иных сообщений на сервер.
5. Реализовать проведение самодиагностики.
6. Реализовать логгирование блока.
7. Создать руководство по эксплуатации.
8. **Ход работы**

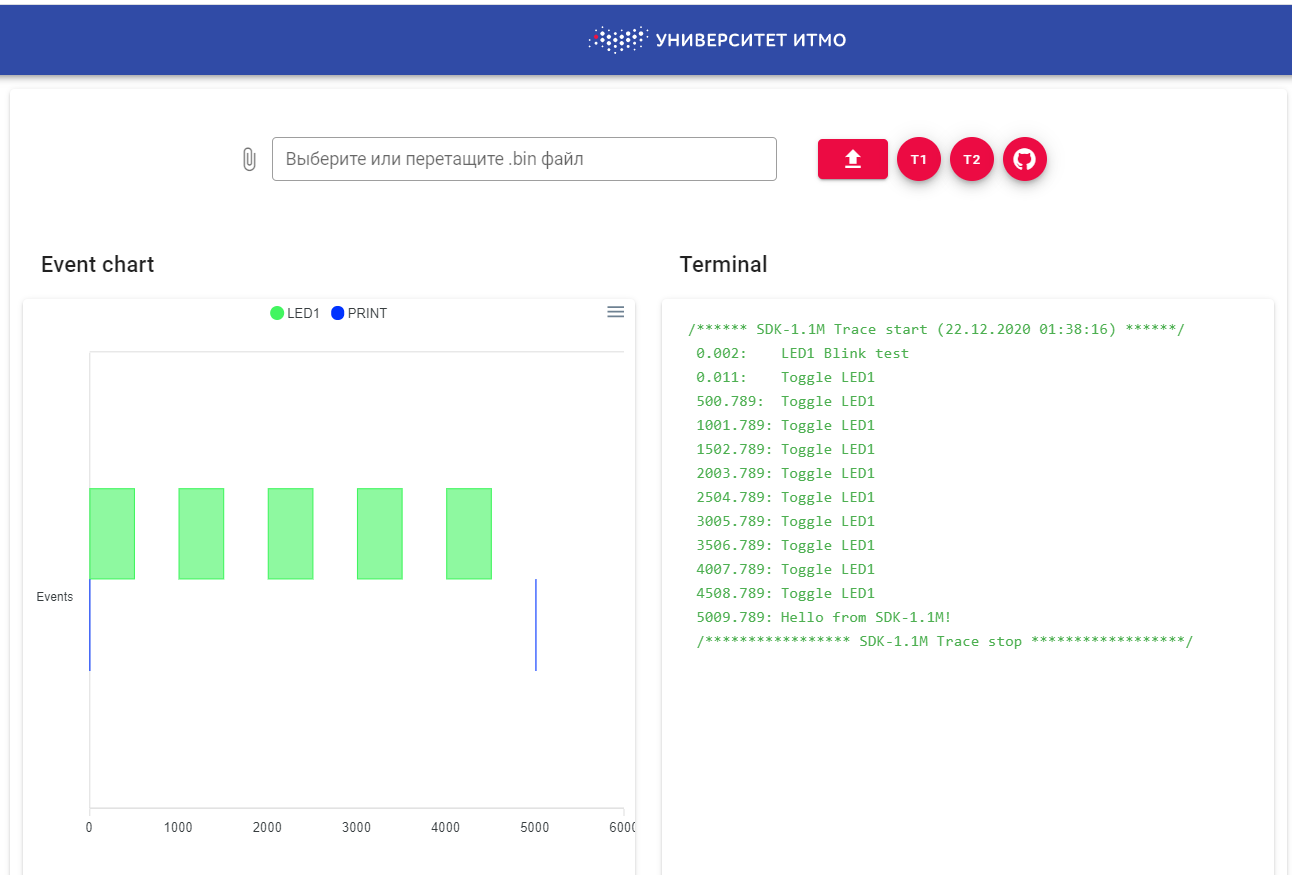
В ходе выполнения данной работы, сразу же были встречены некоторые трудности.



*Рисунок 3. Структура стенда SDK-1.1M.*

Использование стенда SDK-1.1M не представляется возможным в качестве основного стенда системы «Умная полка», поскольку на данный момент функционал онлайн-версии стенда, предлагаемый нам на платформе CLAB в условиях дистанционного обучения, предполагает использование преимущественно светодиодов и таймеров. Соответственно о модулях камеры и Wi-Fi/Ethernet не может быть и речи, поскольку банально отсутствует возможность подключения платы расширения или кабелей Ethernet, USB модулей камеры.

Исходя из сложившейся ситуации, в условиях невозможности отладки кода и тестирования на реальной плате, было принято следующее решение – реализовать максимально возможную часть поставленных требований и задач с помощью стенда CLab, а конкретнее, реализовать вывод значений (цен) на экран, а остальную часть работы проэмулировать путем создания соответствующей программы.

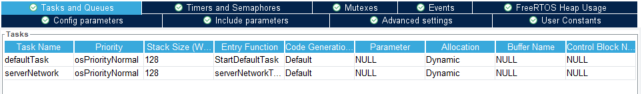


*Рисунок 4. Интерфейс и пример выполнения «мигания светодиода» на онлайн-версии SDK-1.1M.*

В качестве основного языка программирования был выбран язык Python. Это является хорошим выбором, поскольку он не входит в нашу учебную программу, то есть таким образом мы приобретаем дополнительные знания и изучаем новую для нас тему, а также в силу того, что другие группы работают с иными языками, например, Kotlin, и, все же часть работы выполняется на SDK-1.1M, то есть, происходит разработка программ на языке C, нами приобретается такой важный опыт, как обеспечение совместимости блоков программного обеспечения, реализованного при помощи различных языков программирования, в рамках одной системы, что крайне положительно сказывается на нашем профессиональном опыте.

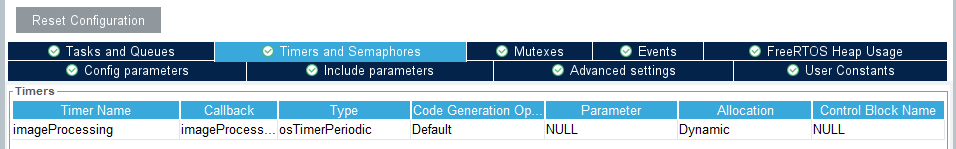
Реализация cLab

В качестве операционной системы была выбрана freertos для обеспечения планирования и выполнения нескольких задач. Это необходимо для обеспечения работы lwip (прослушивание трафика) и работы остальных компонент системы, таких как отправка изображений на сервер. Для операционной системы было две различных задачи *defaultTask* и *serverNetwork*. Первая задача циклически ожидает окончания 8-ми секунд и завершает трассировку SDK с помощью вызова функции *SDK\_TRACE\_Stop()*, это является обязательным условием для работы в виртуальной лаборатории cLab. Во второй задаче происходит прослушивание сетевого трафика, обработка соединений для последующей отправки-получения пакетов. На рисунке 5 можно наблюдать настройку данных задач.



*Рисунок 5. Задачи в операционной системе freertos*

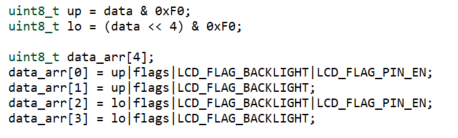
Для реализации периодической отправки изображений на сервер решено было использовать software timer, который входит в состав операционной системы. На рисунке 6 продемонстрированы параметры таймера *imageProcessing*. Данный таймер настроен на периодический вызов функции *imageProcessingCallback()* каждые 2 секунды. В данной функции происходит получение изображение камеры из памяти и отправка пакета с изображением серверу.



*Рисунок 6. Параметры software таймера imageProcessing*

Для работы с дисплеем был использован I2C LCD дисплей. Его функционала достаточно для отображения названия продукта, его цены и валюту, однако его функционал также невозможно использовать в виртуальном стенде.

Текстовые символы на LCD дисплей передаются в формате ASCII. При отправке каждого символа происходит передача четырёх байт. Первый байт содержит старшие 4 бита символа, а младшие биты хранят флаги. Второй байт хранит только флаги. Третий байт хранит младшие 4 бита символа и флаги. И четвертый также хранит только флаги конфигурации. На рисунке 7 представлен пример подготовки 4-х байт.



*Рисунок 7. Подготовка четырёх байт для передачи на LCD дисплей*

Также перед отправкой сообщения необходимо установить специальные символы, такие как установка курсора в начало строки, выключение визуализации курсора, очистка дисплея.

Так как возможности, предоставляемые виртуальной лабораторией ограничены, то функционал был реализован лишь частично. Работа с сетевым трафиком и передача символов на дисплей было реализовано в виде “заглушек” с некоторой временной задержкой. Однако при наличии реального оборудования с требуемыми возможностями/модулями можно с лёгкостью адаптировать данный проект, реализовав лишь отсутствующий функционал.

На рисунке 8 продемонстрировано выполнение в виртуальной лаборатории cLab. Как можно видеть, среди событий присутствуют события P1 (получение пакета с установкой товара и его цены) и P2 (периодическая передача изображения с камеры серверу). Событие P2 происходит каждые 2 секунды, как и требовалось. Также в трассировочном буфере можно наблюдать полезную информацию, такую как, установка товара и его цены на дисплей, отправка изображений с камеры серверу и лог ошибок, происходящих в системе, например, неправильно выставленный идентификатор валюты.



*Рисунок 8. Результат работы в виртуальной системе cLab*

Блаблабла про питон, сделаю, блаблабла про сдк, нужен код и скрины

1. **Вывод**

Ничего не работает, почему не понятно, но все было очень интересно, поставьте 5.

**Приложение 1. Код эмулятора SDK для проекта.**

from threading import Thread

import cv2

import os

import time

import requests

from http.server import HTTPServer, BaseHTTPRequestHandler

from http import HTTPStatus

import json

from jsonschema import validate

from base64 import b64encode

def log(message):

    print("\nSHELF LOG: %s" % message)

\_url = 'http://127.0.0.1/api/postImage'

\_product = {

    'productName': 'milk',

    'currency': 899,

    'value': 99.99

}

\_currency = {

    643: 'RUB',

    840: 'USD',

    978: 'EUR',

}

\_schema = {

    "type": "object",

    "required": ["productName", "currency", "value"],

    "properties": {

        "productName": {"type": "string"},

        "currency": {"type": "integer"},

        "value": {"type": "number"},

    },

}

class Server(BaseHTTPRequestHandler):

    def \_header(self):

        self.send\_header('Content-type', 'application/json')

        # Allow requests from any origin, so CORS policies don't

        # prevent local development.

        self.send\_header('Access-Control-Allow-Origin', '\*')

        self.end\_headers()

    def \_ok\_header(self):

        self.send\_response(HTTPStatus.OK.value)

        self.\_header()

    def \_deny\_header(self, message):

        self.send\_response(HTTPStatus.BAD\_REQUEST.value)

        self.\_header()

        self.wfile.write(json.dumps(

            {'success': False, 'message': message}).encode('utf-8'))

    def do\_GET(self):

        self.\_ok\_header()

        self.wfile.write(json.dumps(\_product).encode('utf-8'))

    def do\_POST(self):

        length = int(self.headers.get('content-length'))

        try:

            message = json.loads(self.rfile.read(length))

            try:

                validate(instance=message, schema=\_schema)

                try:

                    currency = \_currency[message['currency']]

                    \_product['productName'] = message['productName']

                    \_product['currency'] = message['currency']

                    \_product['value'] = message['value']

                    self.\_ok\_header()

                    self.wfile.write(json.dumps(

                        {'success': True}).encode('utf-8'))

                    log("set product to the shelf: %s - %d %s" %

                        (\_product['productName'], \_product['value'], currency))

                except KeyError:

                    self.\_deny\_header("invalid currency value")

            except Exception as ex:

                self.\_deny\_header(ex.message)

        except json.decoder.JSONDecodeError:

            self.\_deny\_header("invalid json")

    def do\_OPTIONS(self):

        # Send allow-origin header for preflight POST XHRs.

        self.send\_response(HTTPStatus.NO\_CONTENT.value)

        self.send\_header('Access-Control-Allow-Origin', '\*')

        self.send\_header('Access-Control-Allow-Methods', 'GET, POST')

        self.send\_header('Access-Control-Allow-Headers', 'content-type')

        self.end\_headers()

class Camera(object):

    def \_\_init\_\_(self, path):

        self.path = path

        self.cam = cv2.VideoCapture(path)

        fps = self.cam.get(cv2.CAP\_PROP\_FPS)

        frame\_count = int(self.cam.get(cv2.CAP\_PROP\_FRAME\_COUNT))

        # video duration time in mseconds

        self.duration = int(frame\_count/fps \* 1000)

        log("Load video %s duration %d " % (self.path, self.duration))

    def getFrame(self, msec):

        self.cam.set(cv2.CAP\_PROP\_POS\_MSEC, msec % self.duration)

        ret, frame = self.cam.read()

        return frame

def run\_server():

    server\_address = ('localhost', 8080)

    httpd = HTTPServer(server\_address, Server)

    log('serving at %s:%d' % server\_address)

    httpd.serve\_forever()

def run\_camera():

    camera = Camera("video.mp4")

    secs = 0  # seconds from start of video

    try:

        while(True):

            time.sleep(2)

            frame = camera.getFrame(secs \* 1000)

            name = './frame%d.jpg' % secs

            cv2.imwrite(name, frame)

            with open(name, 'rb') as file:

                payload = {

                    'file': b64encode(file.read()),

                }

                r = requests.post(\_url, payload)

            log('Send file %s' % name)

            os.remove(name)

            secs += 2

    except Exception:

        camera.cam.release()

        cv2.destroyAllWindows()

thread1 = Thread(target=run\_camera)

thread1.daemon = True

thread1.start()

run\_server()