Передмова

Наша команда працює над автоматизацією спеціальних програм які використовують на етапі виробництва наших датчиків.

Важливо розуміти, що всі логи та код, які ми надаємо в цьому завданні, є придуманими спеціально для цього завдання.

Це завдання складатиметься з 2х частин перша на перевірку бази пітона, друга за пайтестом, до речі, перше завдання починається прямо зараз, удачі!

Завдання 1

Кожна програма пише лог-файл, з якого нам періодично необхідно щось отримати, і це завдання покаже нам, наскільки ти в цьому хороший.

Для виконання завдання ми прикріплюємо файли: ● Файл із логами: **app_2.log**

Порожній файл для твого розв'язання цього завдання: do_it_yourself.py – ми не прикріплюємо

На виробництві датчики спілкуються зі спеціальною централлю і надсилають їй свої статусні повідомлення зі свідченнями, це може бути температура, рівень батареї тощо.

Приклад рядка, який може надіслати датчик:

У прикладі показаний рядок з хендлером 'BIG', з ним і працюватимемо.

Інші рядки нам сьогодні не потрібні, все, що знаходиться з лівого боку рядка (time, type, etc) до роздільника ">" є сервісною частиною повідомлення, ця частина нам також не потрібна.

> 'HANDLER;13;ID;1;36;39;7463;-92;-57;1;2;129;5;0;0;671;1;199;1;STATE;\r\n'

● HANDLER - тип повідомлення

У лог-файлі зустрічатимуться хендлери (BIG, BAD, WOLF), кожен містить у собі різні дані, але позиція кожного аргументу всередині хендлера та його кількість незмінні.

● ID – унікальний ідентифікатор датчика

Програма відрізняє кожен фізичний пристрій саме за ID

● STATE - стан датчика У цьому прикладі будуть 2 стани:

02 - датчик ОК

DD - датчик почувається погано

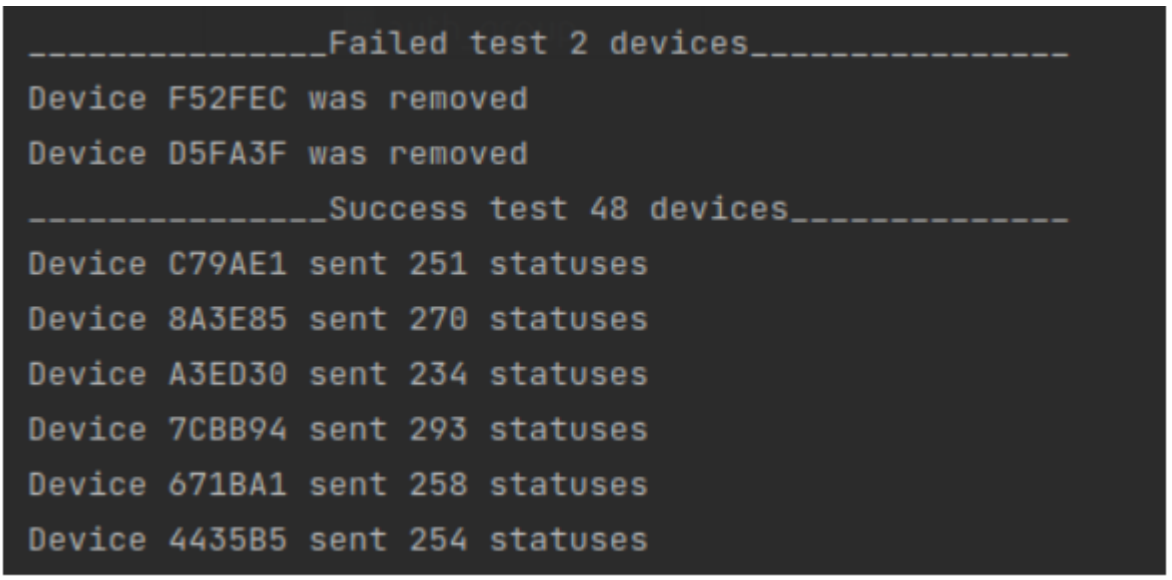
Що із цим робити?

Завдання стоїть нетривіальне, необхідно реалізувати функцію, яка вважає кількість повідомлень з хендлером 'BIG' для кожного датчика, який ОК. Датчики, які надсилають STATE:DD - рахувати не потрібно.

Додаткове завдання:

Порахувати кількість датчиків, які успішно пройшли тест і датчиків, які зазнали фіаско

Поради щодо виконання завдання



- Результат роботи функції можна переглянути на скріншоті

- В результаті правильного вирішення завдання буде 40 девайсів, які пройшли тест та 10 бракованих.

- Обмежень у кількості рядків та імпорті бібліотек немає, але чим простіше реалізація – тим краще.

● Висновок статистики можна просто принтити.

Важливо:

Датчик може надсилати STATE:02, але в процесі тесту зрозуміти що він несправний, відправити STATE:DD і після продовжить надсилати STATE:02 Такі датчики не повинні входити в статистику, датчик, який одного разу надіслав несправність, вважається - несправним.

Завдання 2:

Для того щоб розуміти, який перед нами датчик і якого кольору ми будемо

використовувати клас 'CheckQr', який тобі доведеться протестувати. Для виконання завдання ми прикріплюємо файли:

● Файл із кодом, який потрібно протестувати: scanner_handler.py

Порожній файл для твого вирішення цього завдання: test_name.py - ми не прикріплюємо

Робота з цим класом починається з методу check_scanned_device метод приймає рядок (QR), перевіряє чи є девайс в базі і який у нього колір (за довжиною QR) і якщо девайсу немає в базі, або за його довжиною не знайдено кольору - повертає помилку.

Особливості

● Не можна редагувати та змінювати код у файлі scanner_handler.py.

● Метод check_in_db буде викликати виняток, уяви, що це реальне підключення до прод-бази та краще з ним не експериментувати. Потрібно придумати, як це обійти. Ця функція повинна повертати True - якщо девайс знайдено і None - якщо девайса немає в базі.

● Методи can_add_device, send_error у цій задачі просто приймають помилку/успіх в аргументах і повертають її в нікуди. Це зроблено для підвищення складності додаткового завдання.

● Усі тести пишуться як функціональні, не як юніт.

● Метод check_len_color записує у змінну color колір або None

● Годувати QR безпосередньо в метод check_len_color або інші методи - погана ідея, краще піти флоу програми і написати функціональний тест.

Кейси для покриття тестами

● Необхідно просканувати QR-коди різної довжини, які є в БД і перевірити, чи програма призначає правильний колір залежно від довжини QR-коду.

● Негативний кейс, в якому ми скануємо QR-код для довжини якого немає кольору

Додаткові завдання:

1. Перевірити сканування QR, якого немає БД.

2. Написати тести для випадку не успішного сканування та перевірити, що метод send_error був викликаний з потрібними аргументами.

3. Написати тести для успішного сканування і перевірити, що метод can_add_device повертає повідомлення у разі успішного сканування, за аналогією з тестом для send_error.