Data Cleaning Challenge

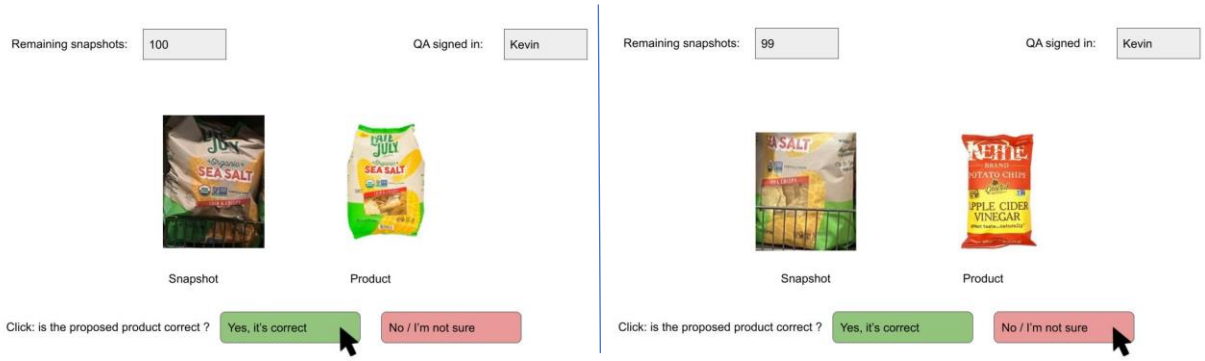
1. Context

As a Computer Vision leader in the retail industry, Trax needs to maintain a global database of product snapshots and use them to train recognition models.

2. Objectives

The aim of this challenge is to design and build an efficient and cost-effective Backend architecture dedicated to the cleaning of a snapshots database.

The design pre-supposes an existing one-page web-app (we show here 2 consecutive views).



After the QA Kevin signed in, he needs to review a random set of 100 snapshots. Each snapshot is displayed next to the reference image of its current product attribution. Kevin needs to confirm if the two correspond, then automatically proceeds to the next page, until there are no more snapshots to validate.

3. Tasks

1. Once a snapshot has been reviewed, it no longer re-appears for review on the web-app. Later stages of the challenge will change this behavior (see below).
    a. Design and build data models for users, products, snapshots. Provided with the challenge description are example products and snapshots.
    b. Implement a simple server that serve those, and all the endpoints necessary for the web-app platform to work, through a RESTful API.
2. After a few days of testing the platform, you realize QAs are not 100% accurate. You propose a solution: to implement a voting system in the Backend. The web-app front-end does not change, however the same snapshot must now be reviewed by 3 different QAs. The status of the snapshot-product association then corresponds to the majority.
    a. Update the data models
    b. Update the server code to implement this voting mechanism
3. To increase the process reliability, you suggest that we incentivize QAs that perform better than others.
    a. How would you rank 3 QAs after a session of 100 snapshots?
    b. Update the server code to implement this mechanism and expose an endpoint to get the QAs leaderboard
4. (Bonus)
    a. Formulate ideas to reduce QA-time while maintaining the same accuracy. You are free to update the web-app page design to some extent if it helps.

4. Implementation:

Requirements:

- All the code is written in Python
- Each service is running in its own Docker container
- A container orchestration tool is used (docker-compose, Kubernetes)
- The provided APIs are RESTfull
- The code is well structured and documented

(Bonus)

- Provide examples of unit tests