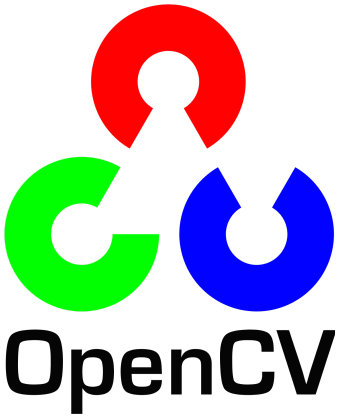
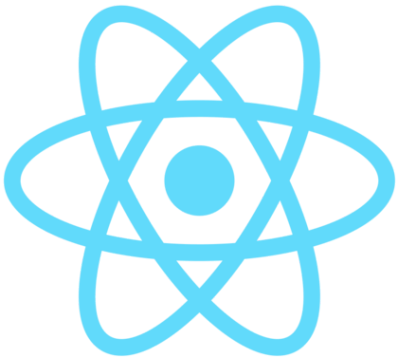
Hourdin Franck - Legru Florent

Application mobile de recherche d'images





I. Introduction.

L'objectif de ce projet était de concevoir une application compatible avec l'OS Android capable d'effectuer une "recherche inversée" d'image. Il fallait donc concevoir un côté "Client" ainsi qu'un côté "Serveur".
Concernant le côté serveur, les contraintes techniques à respecter était l'architecture REST, le langage PYTHON, la librairie OPENCV pour le travail sur image.
Côté Smartphone, nous avons choisi d'utiliser REACT NATIVE car celle-ci était connue de Franck.

II. Principe de fonctionnement

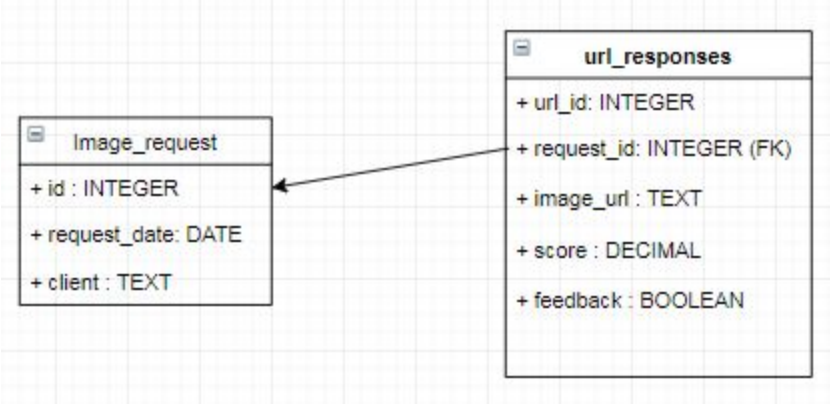
L'idée principale est d'effectuer une requête côté client en envoyant une image au format "base64" donc une chaîne de caractère, qui devra être reconstituée côté serveur. Le serveur va ensuite calculer les points SIFT de cette image, et les comparer à la base d'image déjà existante pour établir un "score", ainsi une liste d'image sera renvoyée par le serveur au client, toujours sous forme de chaîne de caractère et les images seront classées par score de correspondance avec l'image requête. Le client effectue deux requêtes asynchrones : La première pour envoyer son image, la deuxième pour récupérer la liste correspondante.
Les résultats côté serveur sont stockés dans une base de données afin de pouvoir retrouver ses requêtes sans avoir à générer à nouveau un travail d'OPENCV.

III. Serveur

Lien git du code : https://github.com/Adralith1/recognition_rest

a) Le modèle de données

Les requêtes du client devant être stockées pour respecter le principe du serveur REST, le modèle suivant de stockage en base de données a été choisi :



La table image_request correspond à la requête initiale du client, on garde donc la date de la requête, ainsi que l'adresse IP du client. La requête en base_64 du client n'est pas stockée puisqu'il n'est à aucun moment nécessaire de renvoyer l'image requête du client.
La table url_response contient plus d'informations, il peut y avoir plusieurs url_id pour un seul request_id. Chaque url_id correspond en fait à une image parmi la liste de celles classées par score. Il y a également une donnée "feedback" qui peut être remontée par le client en un simple click sur l'image afin de donner son avis sur le classement, un 0 ne désigne aucun feedback (donc classement considéré correct) un 1 désigne une mauvaise image (ou mauvais classement).

b) Le traitement d'image :

Il y a une première indexation à effectuer grâce au fichier "indexing.py" qui récupère toutes les images et stocke les points SIFT dans un fichier "bof_retr.pkl". Ce fichier se charge également d'uploader toutes les images sur un hébergeur du nom de "NOELSHACK", de cette manière, le client ne recevra pas de résultats en base64 mais de simples URL à afficher, ce qui permet de ne pas faire de second traitement sur une liste d'images en b64, ni côté serveur, ni côté client.

c) Importer une nouvelle base d'images

Une url `url(r'img_searches/new_base',views.NewBase.as_view())[...]`
A été créée afin de pouvoir effectuer ses recherches à l'aide d'une nouvelle banque d'images, il faudra pour cela compresser toutes ses images dans un fichier au format ZIP, les anciennes images seront soit supprimées, soit gardées en fonction de l'option choisie par l'utilisateur.

III. Client

https://github.com/fhourdin/react_native_logo_recognition (Suivre README)

Concernant la partie mobile de l'application, par commodité, notre choix s'est porté sur la librairie React-Native. Celle-ci permet la créations d'applications natives en utilisant le Javascript, et plus spécifiquement la librairie React. Son intérêt réside d'une part dans la lisibilité du code, mais également du fait qu'une telle application peut à la fois être déployée sur Android, et sur iOS.

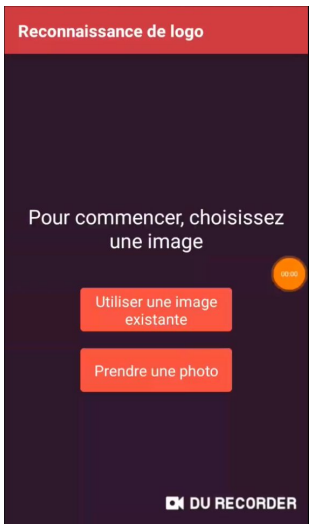
a) Environnement de développement

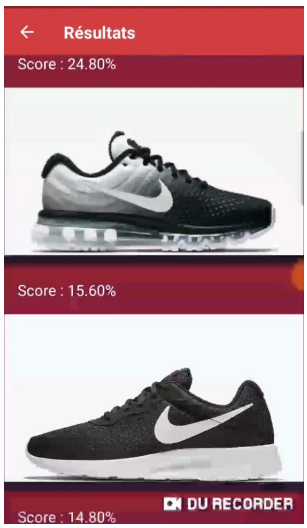
Pour plus de facilité de développement, mais aussi de présentation, notre application utilise Expo comme environnement de développement. Une fois Expo lancé sur un appareil mobile (Android ou iOS), on peut très facilement déployer le code local via USB, réseau local, ou réseau tunnelé. Le serveur de développement expo propose également un système très pratique de "Hot reloading", qui prends en compte chaque modification et redéploie automatiques celles-ci. L'inconvénient est qu'Expo ne permet pas l'utilisation des librairies natives (propres à iOS ou Android), mais uniquement les librairies JS.

b) UI/UX

Nous avons opté pour deux librairies graphiques très reconnues : react-native-ui-kitten, et Material-UI (crée par Google).

Ce type de librairie nous a permis d'utiliser des blocs logiques préalablement stylisés. Ce qui permet d'obtenir un résultat tout à fait acceptable en terme de design. Le travail fut simplement de définir les bases d'un thème (Couleurs, typographies, etc...)





c) Interfaçage API

Enfin, notre client utilise la très célèbre librairie whatwg-fetch pour communiquer avec le serveur. Javascript permet une utilisation simple des concepts asynchrones grâces aux “Promises” et à l’ “Async/Await”, L’interface utilisateur se met à jour en fonction des comportements utilisateurs et des réponses apportées par le serveur.

Les requêtes vers le serveur permettent alors d’hydrater les propriétés des composants React, et viennent mettre à jour les différentes vues. Le système de navigation est géré par la librairie “react-navigation”, proposant un ensemble de fonctions simples pour gérer les différentes vues.