Projet FitSize : étude du Backend existant

Florian CHAPPAZ, Valentin DE OLIVEIRA, Sami IFAKIREN, Clément NGUYEN

I. Introduction

Le projet FitSize a déjà été commencé l'année dernière par d'autres camarades d'INFO5. Nous nous intéresserons ici seulement à la partie Backend du projet. En revanche, nous savons d'or et déjà que nous ne pouvons pas baser entièrement notre travail sur ce qui a été fait. En effet, l'année dernière, l'objectif du projet était que les utilisateurs indiquent à la main les points de mesure du vêtement. Depuis, l'entreprise est partie sur une autre piste, basée sur du Machine Learning, où les points d'ancrage sont déterminés automatiquement avec la photo donnée en entrée, donnant moins de complexité cognitive à l'utilisateur.
Nous avons donc besoin de déterminer les changements à effectuer ainsi que les choses que nous pouvons conserver.

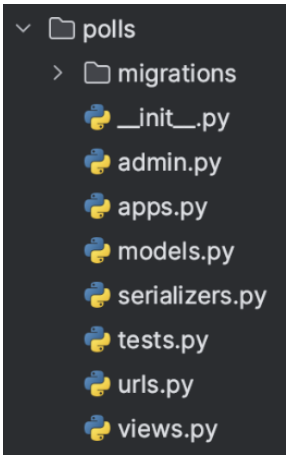
II. Le projet existant

Le projet existant est disponible à l'adresse suivante :
https://github.com/pfefitsize/FitSize---Backend

L'équipe précédente nous a également laissé un rapport technique expliquant les différents endpoints de l'API ainsi que le modèle de données :
https://air.imag.fr/images/1/1e/Rapport_technique.pdf

Analysons maintenant les différentes parties du projet.

Première remarque en survolant le code : on notifie l'utilisation récurrente du package *djangorestframework*, qui avait été évoqué par notre tuteur.



On trouve d'abord un premier dossier nommé *polls*, qui correspond au cœur du back-end. C'est l'unique application du projet. Cependant il faudra changer son nom, car celui-ci est directement tiré du tutoriel Django, et il n'est pas représentatif du contenu. De plus, nous serons peut-être amené à créer une seconde application avec d'autres endpoints pour gérer le traitement des images par l'IA (cf. Partie III.).

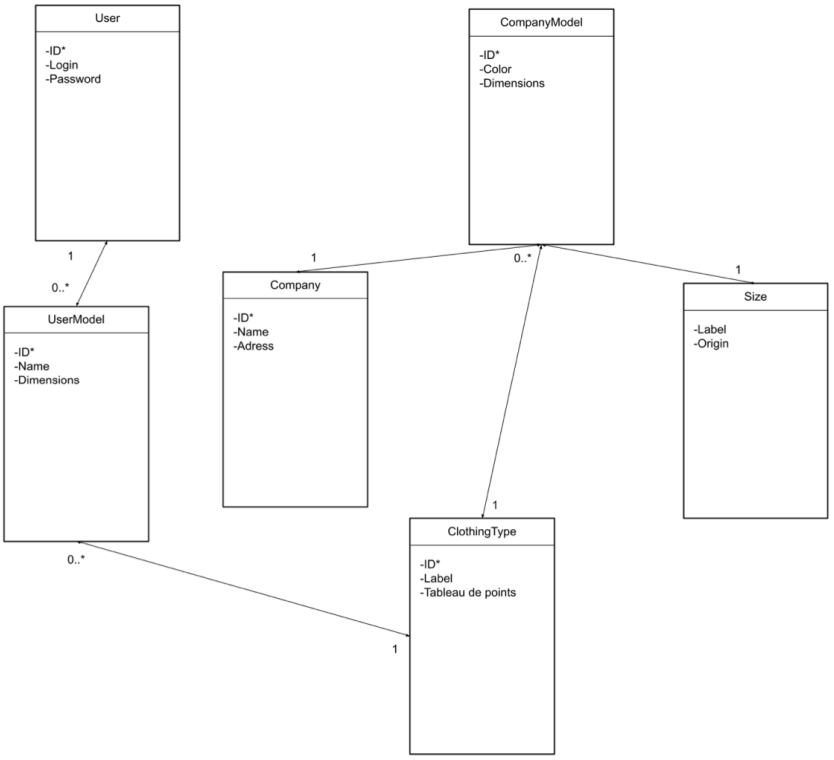
On constate également la présence de données fictives, qui peuvent s'avérer utiles si on souhaite conserver le modèle de données existant.

Il y a également un dossier contenant des configurations pour tester les endpoints avec Postman. Cela est également très utile pour tester ce qui existe, donc on souhaite aussi conserver ce dossier.

Enfin des tests sont également présents.

Mais ces différents éléments ne sont intéressants que si le modèle de données existant est cohérent avec ce que l'on souhaite apporter à l'application.

III. Le modèle de données existant



Modèle de données UML du rapport technique

Ce premier schéma n'entre pas dans les types et les structures de données, mais il nous permet d'avoir une première approche sur le travail réalisé. Fondamentalement, cela correspond bien à l'idée que nous nous faisions de l'application. Le login des utilisateurs est déjà géré, ce qui nous permettra de nous concentrer sur le fonctionnel.

On remarque simplement une chose : notre tuteur nous a indiqué que l'un des scénarios à effectuer en priorité est l'ajout d'une collection de vêtements par une entreprise. Il faudra donc modifier légèrement ce modèle pour que le représentant d'une entreprise puisse se connecter.

A part l'ID, tous les champs des entités sont des chaînes de caractères. La plupart de ces champs ont un nom qui explique très bien leur contenu. Cependant, deux d'entre eux méritent plus d'explications :

- **Dimensions**: aussi bien dans l'entité UserModel que CompanyModel. Que ce soit dans le rapport technique ou le rapport final, nous n'avons pas trouvé de documentation par rapport à ce champ. Or, quand nous allons voir un exemple dans les données fictives, nous obtenons cet exemple :

  ```
  "dimensions": "1.0,0.1,4.0,2.5,4.0KP0.0,0.5,8.0,7.0,6.0,5.5,1.2,7.8,8.8,0.0,9.4,8.0",
  ```

  L'exemple ne nous permet pas de deviner la structure de données facilement.
  Mais en allant décortiquer la fonction saveFromKeyPoints() :
  - Les 5 floats avant le token 'KP' représentent l'objet témoin avec les deux coordonnées de la première extrémité, les deux coordonnées de la seconde extrémité, ainsi que la taille réelle.
  - Les floats après le token 'KP' représentent les coordonnées des keypoints du vêtement. Mais il est assez difficile de comprendre comment ils sont utilisés, et pourquoi ils sont traités par groupes de 4.

  Nous ne nous attarderons pas à comprendre le fonctionnement exact de la structure de données, car nous verrons dans la partie suivante que nous serons amenés à le changer.

- **Points**: dans l'entité ClothingType. Ce champ est censé nous indiquer quels sont les keypoints à mesurer. Mais dans les données fictives, on trouve seulement cette valeur pour tous les vêtements :

  ```
  "points": "1 2 6 7"
  ```

  On ne comprend pas vraiment comment ce champ représente les keypoints, mais encore une fois, nous n'en avons pas l'utilité, car cette représentation est amenée à changer avec le modèle de Machine Learning fourni par l'entreprise.

IV. Le nouveau modèle ML de l'entreprise

L'entreprise FitSize nous a fourni un modèle de Machine Learning qui a pour but de détecter automatiquement les keypoints avec une simple photo du vêtement. Nos tuteurs nous ont également précisé que ce modèle retourne des dimensions réelles (en millimètres par exemple) que nous sommes directement capables de traiter.

Nous n'allons pas rentrer dans les détails de fonctionnement du modèle, qui est basé sur OpenCV et sur l'outil AILIA entre autres.

Nous allons plutôt nous intéresser aux résultats retournés par ce modèle, en particulier leur format.
Deux scripts sont fournis :

- *run_det.py* : est censé détecter automatiquement le type de vêtement qui est pris en photo. Mais après un test non concluant, et le peu de temps que nous avons devant nous, nous allons directement opter pour le script suivant.
- *run_no_det.py* : le type de vêtement est pour le moment indiqué dans le fichier *CONFIG.py*. Nous allons utiliser ce script qui semble être très fonctionnel.

Tout d'abord, l'exécution du script fourni nous donne en sortie une image avec les keypoints placés, pour nous donner une visualisation rapide de la sortie.

Image en sortie d'un vêtement avec les keypoints

On remarque en premier lieu que les keypoints ont des noms bien spécifiques.
En effet, quand nous allons consulter le fichier de configuration utilisé par le modèle, on tombe sur cette variable :

```
KEYPOINTS = {
        'blouse': ['neckline_left', 'neckline_right', 'center_front', 'shoulder_left', 'shoulder_right',
                   'armpit_left', 'armpit_right', 'cuff_left_in', 'cuff_left_out', 'cuff_right_in',
                   'cuff_right_out', 'top_hem_left', 'top_hem_right'],
        'outwear': ['neckline_left', 'neckline_right', 'shoulder_left', 'shoulder_right', 'armpit_left',
                    'armpit_right', 'waistline_left', 'waistline_right', 'cuff_left_in', 'cuff_left_out'
                    'cuff_right_in', 'cuff_right_out', 'top_hem_left', 'top_hem_right'],
        'trousers': ['waistband_left', 'waistband_right', 'crotch', 'bottom_left_in', 'bottom_left_out',
                     'bottom_right_in', 'bottom_right_out'],
        'skirt': ['waistband_left', 'waistband_right', 'hemline_left', 'hemline_right'],
        'dress': ['neckline_left', 'neckline_right', 'center_front', 'shoulder_left', 'shoulder_right',
                  'armpit_left', 'armpit_right', 'waistline_left', 'waistline_right', 'cuff_left_in',
                  'cuff_left_out', 'cuff_right_in', 'cuff_right_out', 'hemline_left', 'hemline_right']
    }
```

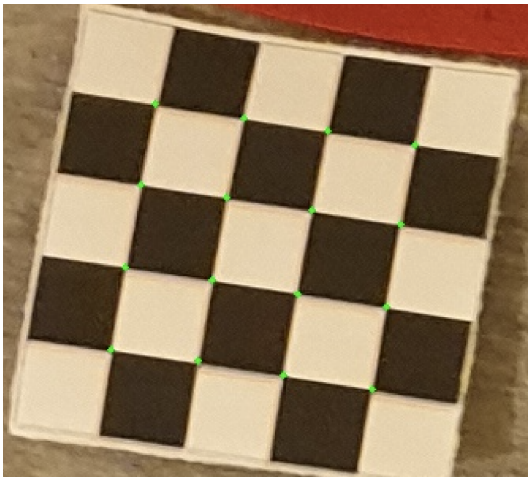
Il faudra donc dans un premier temps modifier le modèle afin que les différents keypoints soient déterminés de la même manière dans l'entité ClothingType.

Pour cela, il est important de comprendre la sortie retournée par ce script. En voici un exemple :

```
{'waistband_left': array([467, 978,   1], dtype=int16), 'waistband_right': array([2522, 1065,    1],
dtype=int16), 'crotch': array([1380, 2616,    1], dtype=int16), 'bottom_left_in': array([1065, 2932,
1], dtype=int16), 'bottom_left_out': array([   2, 2435,    1], dtype=int16), 'bottom_right_in':
array([1672, 2861,    1], dtype=int16), 'bottom_right_out': array([2932, 2624,    1], dtype=int16)}
52.17978800104307
```

On comprend bien que le premier objet correspond aux coordonnées des différents keypoints affichés en vert sur l'image du caleçon plus haut dans ce document. Malheureusement, aucune documentation ne précise ce format de sortie (unité : pixels ou millimètres ?). La dernière ligne est censée correspondre à la taille de l'échiquier posé en échelle de référence. Nous avons tenté de faire plusieurs calculs de distance en partant de plusieurs hypothèses, mais nous n'arrivions pas à trouver la taille réelle du caleçon.

Il faut donc aller explorer le code plus en détail.



Dans le script, on constate un appel à la fonction *utils.helper.get_checkerboard_corners*

Cette même fonction fait appel à la fonction `findChessboardCorners()` de la librairie OpenCV2. Elle retourne la liste des coordonnées des points verts.

La fonction du helper calcule la moyenne des distances entre chaque point vert (1cm), en pixel.

Donc dans notre exemple, **1cm équivaut à 52.17978800104307px**.

Les coordonnées des keypoints étant bien des coordonnées en pixels, il suffit donc de calculer une distance en pixels entre deux keypoints et de diviser par le 52,18px. Dans notre exemple, on obtient un caleçon d'une largeur de 39 cm.

V. Conclusion

Nous pouvons donc partir sur le backend de l'année dernière, et modifier légèrement le modèle à notre guise, en particulier la manière de stocker les dimensions qui doit correspondre aux mêmes structures de données que les résultats retournés par l'IA.
Il nous reste donc à finaliser l'API pour faire le lien avec le front, mais nous devons aussi créer une API permettant de recevoir une image, de la traiter avec l'intelligence artificielle, et de récupérer les dimensions fournies.