Avant le 15/02 :

Ce qui a été fait :

- Installation des outils (Android-Studio, SDK, Flutter, DART)
- Cours de base (Langage DART, Programmation Fonctionnelle)
- Début du cours sur les widget

15/02/2022 (DONSEZ Didier) :

EXIF : informations sur la prise des photos (localisation ….)
Application Ruler (iphone) : prise de mesure sur photos
Exemple de projet avec Flutter (covoiturage solidaire) :
https://github.com/OliDesu/MoBiPa/tree/main/app
https://github.com/OliDesu/MoBiPa/blob/main/README.md -> traitement des images
On peut retrouver les rapports des projets faits en Flutter des années précédentes (Jorigine & MOBiPa) : https://air.imag.fr/index.php/Projets_2020-2021

Éventuellement utiliser Jhipster Spring pour générer code

OpenCV : bibliothèque graphique de traitement d'images en temps réel
https://opencv.org/

======> TODO :
Noter ce qu'on a fait jusqu'à présent
Penser au plan d'action en terme de travail
Regarder si Flutter est le bon système, si il permet de faire les 2 systèmes d'exploitation (iOS & Android)

16/02/2022 :

- Quelle technique utilisée ? = Object témoin utilisé sur l'application ruler (dans ce cas comment reconnaître l'objet témoin ?)
  = ou plus utilisation de type ruler qui utilise distance focale de caméra (utiliser camera-ruler))
  Besoin d'une BDD pour stocker les tailles ?
- Détailler les technos (par exemple, utiliser OpenCV pour reconnaître où mesurer les distances)
- Comment se répartir le travail

Daily Meeting

Méthodologie de travail (AGILE, SCRUM)
Discussion sur le planning
Ceremonial à mettre en oeuvre régulièrement —--> Daily meeting pour mettre à jour les points bloquants
Codage Météo : SQLite (pas JSON et API —--> Inutile)
App musique

Convention envoyée
Informations mises à jour, inspirations de Fidèle sur le Drive (ce qui a déjà été fait …)
Trop tôt pour discuter de technologie
Maquette Flutter Python sur lesquels on peut s'appuyer (techniques -> ARCoreProject)

Schéma dans formalisme à définir
Demande du modèle de référence (carte de visite pour objet témoin) -> demande de ça dans le premier écran
Deuxième écran : Prise de vue de modèle témoin, à plat élément de ref, puis redimensionnement afin d'être sur surface réelle → Identifier surface réelle et pouvoir comparer

Notre proposition : Utiliser éventuellement technologie Ruler basée sur distance focale
Grandeur à retenir pour la taille (auto ou manuelle)
→ Points clés = deux endroits afin de vectoriser les distances sur le vêtement
Auto : personne normale
Manuel : utilisation de l'user, si formes différentes (personnes voutées, etc…)
(Teva) Semi-auto : user peut intervenir sur points mal placés par algo auto

2 Étapes dans processus du programme :
Modélisation du contour
Taille haute, Taille basse, hauteur (spécifique à chaque acheteur)

Pas de fonctionnement avec pixels car lumière influence beaucoup
Superposition avatar avec vecteurs
Utiliser vecteurs —> superposer avatar avec d'autres modèles témoins

Proposition qui semble la mieux : Utiliser un modèle de vecteurs pour le témoin

Proposition de Léo : Développeur métaverse Blockchain

Préfère stocker vecteur (à la place des distances des vêtements) car si on prend différentes mesures, on peut appliquer facilement sur tous les modèles vectoriels.

18/02/2022 :

PRESENTATION MI-PARCOURS

Base de donnée suggérée :

PostGre
MongoDB
===> Pas H2 (pas fait pour la prod, fait pour le dev)

Déploiement en production à voir

TRAVAIL EN AUTONOMIE :
Recherche d'un framework backend Python (veille technologique) :
JOrigine → utilisent Firebase pas de Python => Pas adapté
https://firebase.google.com/

Django éventuellement
https://www.djangoproject.com/

Comment envoyer les informations au serveur ?
Dans le cas de la prise de l'objet témoin + vêtement (sur un seul écran) :
L'utilisateur de l'application Flutter prend une photo, indique les points de mesure du vêtement et de l'objet témoin, et envoie le tout au serveur (points + photo).

Django vs Flask :
https://hackr.io/blog/flask-vs-django

Django ne supporte pas MangoMDB —-> PostGreSQL
Est-ce que PostGreSQL supporte bien le stockage d'images ?

Présentation de notre choix d'implem :

User prend photo sur portable avec objet pour déterminer échelle
Entrer à la main les points d'intérêts du vêtement, entre lesquels on va prendre les grandeurs
On récupère sur application Flutter image + points utilisateur au backend
Calcul côté back de l'échelle
Vectorisation côté back
Une fois obtenu distances, on compare avec BDD

BDD : Images vêtements + modèle vectoriel

Appli stocker données scannées (cache pour prise de mesure vêtements) côté BDD ou front ?
Ce à quoi Jean-Marc avait pensé : Calcul pur dans le cloud via envoi dans endpoints, pas grand chose au final

Suggestion : inconvénient de introduction de mesure manuelle façon virtusize (lien dans dossier fonctionnel du drive)
Mesure manuelle d'abord puis introduction calcul auto des prises de mesures par AI

⇒ Moyen de stockage d'images vectorisées ? Points avec repère ou vecteurs ?
Images telles qu'elles ? Peut être lourd
Pour première itération, on devrait ne pas avoir besoin car on retrouve distances avec échelle
Nécessaire pour deuxième itération avec algo auto pour points de mesure

On peut retrouver les modèles vectoriels qui seront à stocker dans fonctionnel/concurrence/virtusize.zip

Retours de Fidèle nécessaire quand on sera à l'étape de comparaison des mesures clients avec les modèles du vendeur

https://www.i-digit.co.uk/
capture d'images par appareil photo

Docker permet de faire mise à dispo d'infrastructure (postgre, web ….)
s'appuie sur script pour automatiser la création d'infrastructure
docker-compose, kubernit

Faire un dépot privé sur GitHub

(Generating ssh key sur GitHub
Ajouter sur serveur identifiant SSH)

Ionos : configurer un serveur cloud

Version de Python 3.7 ou 3.8

02/03/2022 :

Installation des outils :
Python 3.10.2
Postgres 14.2
Django 4.0.3

Mise en place du repo GitHub

DAILY MEETING :

Mise au point -> installation des outils
Dépôt Git

Serveur de production :
flutter build-apk —--> Gérer le fichier exécutable

Mettre en mode dev pour propriétés du téléphone android, sinon on peut pas dl appli en dehors du android store

apk à mettre sur appareil mobile (à DL sur navigateur web)
Avoir un serveur web, télécharger apk dans répertoire du serveur web
A mettre dans un index.html
Cliquer sur lien dans index.html
Une fois enregistré —-> on peut exécuter APK, à mettre en compatibilité développeur

Besoins :
Prendre en photo + mesure d'une bague, paire de chaussures, lunettes (avec un élément de référence)
=====> Prendre objet au sol avec objet de ref à côté
=====> carré de 5cm utilisé le plus souvent comme objet de ref
Prime ruler : projet avec prise d'objet de mesure

Pièce de 2 euros : souci de monnaie européenne
Photo avec monnaie + règle précise pour ref

Menu Retouche avant livraison

Partir sur un type de vêtements pour prise de mesure (prise de mesure) : Tshirts puis étendre à slips

Travailler sur capture de modèle témoin, prise de photo

Flutter Build Production :
https://docs.flutter.dev/deployment/android#building-the-app-for-release

04/03/2022 :

1ère ébauche de base de données

Réalisation du poster en anglais

07/03/2022 :

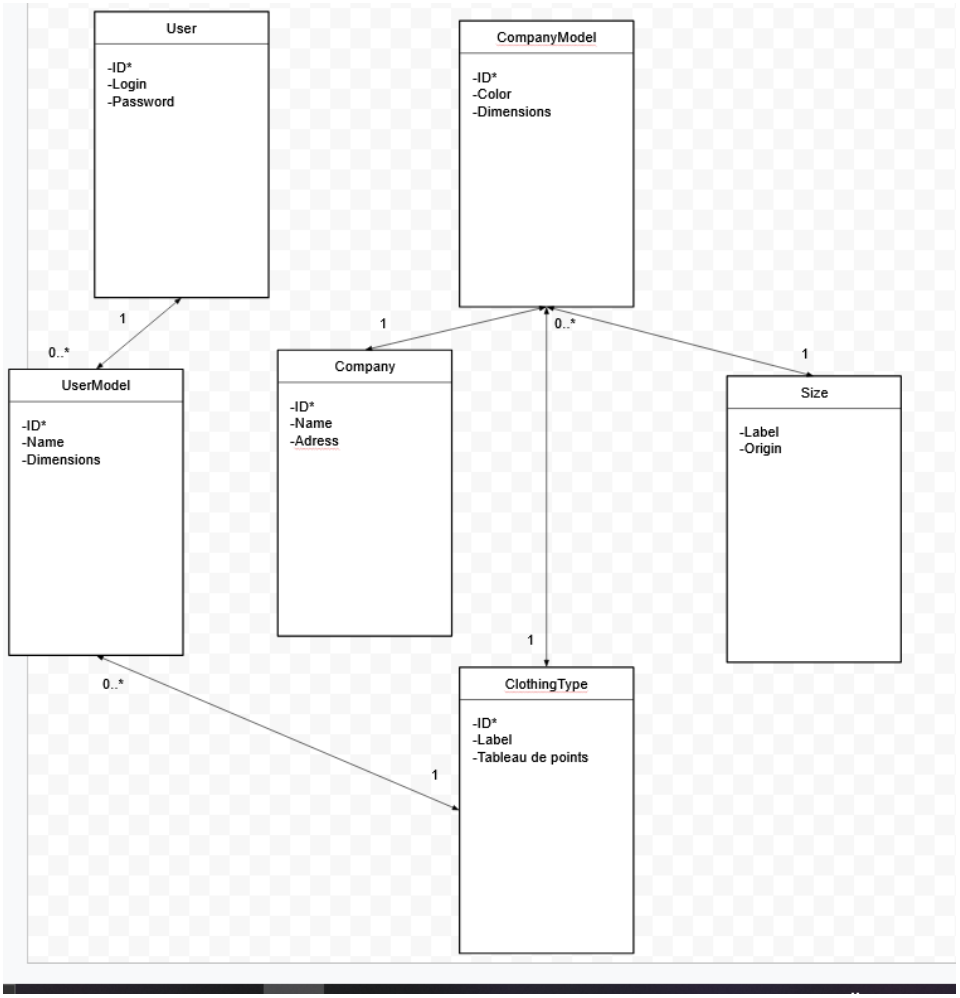
Django tutos first apps partie 1 & 2
Flutter page de connexion basique terminée

Questions à poser à la réunion de mardi : Qu'est ce qui serait utile au calcul des dimensions ?

Changement de database : SQLite (auparavant PostGreSQL) —> rapide et concret, par défaut sur l'appli DJango, fonctionne en local

Fini implémentation des modèles niveau backend, modèles peut être à revoir mais donne une idée de ce qui est stocké dans la database

On suit ce tuto en plus de la doc Django :
https://www.caktusgroup.com/blog/2019/02/01/creating-api-endpoint-django-rest-framework/



A FAIRE DEMAIN :
Installer Rest pour Django

Frontend :

Page de connexion et navigation vers page d'accueil contenant des profils de vêtements pour l'utilisateur (== USERMODEL dans le diagramme de classes)

8/03/2022

Rest framework installé
Analyse de Factory Boy pour créer des tests
Faker pour créer de fausses données

11/03/2022

Erreur à corriger sur le Backend :

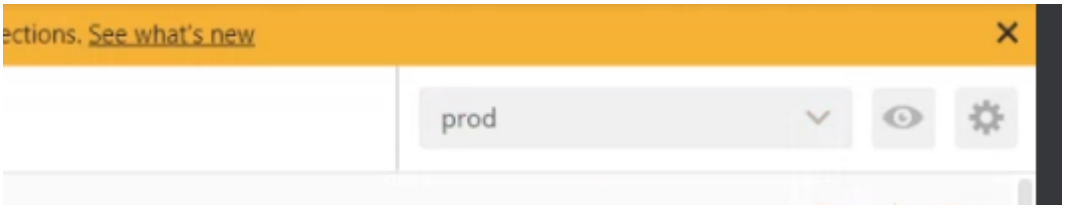
```
ERROR: testPost (tests.testViews.UserViewSetTestCase)
POST to create a User.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\jules\demo\lib\site-packages\django\db\backends\utils.py", line 89, in
_execute
    return self.cursor.execute(sql, params)
  File "C:\Users\jules\demo\lib\site-packages\django\db\backends\sqlite3\base.py", line 477,
in execute
    return Database.Cursor.execute(self, query, params)
sqlite3.IntegrityError: datatype mismatch

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "C:\FitSize\FitSize---Backend\tests\testViews.py", line 23, in testPost
    response = self.client.post(self.list_url, data=data_user)
  File "C:\Users\jules\demo\lib\site-packages\django\test\client.py", line 852, in post
    response = super().post(
  File "C:\Users\jules\demo\lib\site-packages\django\test\client.py", line 441, in post
    return self.generic(
  File "C:\Users\jules\demo\lib\site-packages\django\test\client.py", line 541, in generic
    return self.request(**r)
  File "C:\Users\jules\demo\lib\site-packages\django\test\client.py", line 810, in request
    self.check_exception(response)
  File "C:\Users\jules\demo\lib\site-packages\django\test\client.py", line 663, in
check_exception
    raise exc_value
  File "C:\Users\jules\demo\lib\site-packages\django\core\handlers\exception.py", line 55, in
inner
    response = get_response(request)
  File "C:\Users\jules\demo\lib\site-packages\django\core\handlers\base.py", line 197, in
_get_response
    response = wrapped_callback(request, *callback_args, **callback_kwargs)
  File "C:\Users\jules\demo\lib\site-packages\django\views\decorators\csrf.py", line 54, in
wrapped_view
    return view_func(*args, **kwargs)
  File "C:\Users\jules\demo\lib\site-packages\rest_framework\viewsets.py", line 125, in view
    return self.dispatch(request, *args, **kwargs)
```

```
  File "C:\Users\jules\demo\lib\site-packages\rest_framework\views.py", line 509, in dispatch
    response = self.handle_exception(exc)
  File "C:\Users\jules\demo\lib\site-packages\rest_framework\views.py", line 469, in
handle_exception
    self.raise_uncaught_exception(exc)
  File "C:\Users\jules\demo\lib\site-packages\rest_framework\views.py", line 480, in
raise_uncaught_exception
    raise exc
  File "C:\Users\jules\demo\lib\site-packages\rest_framework\views.py", line 506, in dispatch
    response = handler(request, *args, **kwargs)
  File "C:\Users\jules\demo\lib\site-packages\rest_framework\mixins.py", line 19, in create
    self.perform_create(serializer)
  File "C:\Users\jules\demo\lib\site-packages\rest_framework\mixins.py", line 24, in
perform_create
    serializer.save()
  File "C:\Users\jules\demo\lib\site-packages\rest_framework\serializers.py", line 212, in save
    self.instance = self.create(validated_data)
  File "C:\Users\jules\demo\lib\site-packages\rest_framework\serializers.py", line 962, in
create
    instance = ModelClass._default_manager.create(**validated_data)
  File "C:\Users\jules\demo\lib\site-packages\django\db\models\manager.py", line 85, in
manager_method
    return getattr(self.get_queryset(), name)(*args, **kwargs)
  File "C:\Users\jules\demo\lib\site-packages\django\db\models\query.py", line 514, in create
    obj.save(force_insert=True, using=self.db)
  File "C:\Users\jules\demo\lib\site-packages\django\db\models\base.py", line 806, in save
    self.save_base(
  File "C:\Users\jules\demo\lib\site-packages\django\db\models\base.py", line 857, in
save_base
    updated = self._save_table(
  File "C:\Users\jules\demo\lib\site-packages\django\db\models\base.py", line 1000, in
_save_table
    results = self._do_insert(
  File "C:\Users\jules\demo\lib\site-packages\django\db\models\base.py", line 1041, in
_do_insert
    return manager._insert(
  File "C:\Users\jules\demo\lib\site-packages\django\db\models\manager.py", line 85, in
manager_method
    return getattr(self.get_queryset(), name)(*args, **kwargs)
  File "C:\Users\jules\demo\lib\site-packages\django\db\models\query.py", line 1434, in
_insert
    return query.get_compiler(using=using).execute_sql(returning_fields)
```

```
  File "C:\Users\jules\demo\lib\site-packages\django\db\models\sql\compiler.py", line 1621,
in execute_sql
    cursor.execute(sql, params)
  File "C:\Users\jules\demo\lib\site-packages\django\db\backends\utils.py", line 67, in execute
    return self._execute_with_wrappers(
  File "C:\Users\jules\demo\lib\site-packages\django\db\backends\utils.py", line 80, in
_execute_with_wrappers
    return executor(sql, params, many, context)
  File "C:\Users\jules\demo\lib\site-packages\django\db\backends\utils.py", line 84, in
_execute
    with self.db.wrap_database_errors:
  File "C:\Users\jules\demo\lib\site-packages\django\db\utils.py", line 91, in __exit__
    raise dj_exc_value.with_traceback(traceback) from exc_value
  File "C:\Users\jules\demo\lib\site-packages\django\db\backends\utils.py", line 89, in
_execute
    return self.cursor.execute(sql, params)
  File "C:\Users\jules\demo\lib\site-packages\django\db\backends\sqlite3\base.py", line 477,
in execute
    return Database.Cursor.execute(self, query, params)
django.db.utils.IntegrityError: datatype mismatch
```

REUNION :

Une fois app générée, accéder au serveur web
Accéder depuis smartphone au lien web
Essayage automatique "Skip Français" sur le Drive



Client Http Flutter pour utiliser les endpoints POST

pubspec.yaml

```
dependencies:
        flutter:
        sdk: flutter
```

http: ^0.12.0+2

Prise de photo avec flutter liens utiles :
https://www.digitalocean.com/community/tutorials/flutter-flutter-http
https://pub.dev/packages/camera (PLUGIN)

Stockage image ac objet témoin + stockage dimensions à faire

13/03/2022

Erreurs en lien avec le data format de la requête quand on essaye d'effectuer une création de user dans les tests :

```
def testPost(self):
    """POST to create a User."""
    data_user = {
        'id': '...'
        'login': 'New name',
        'password': 'New password',
    }
    self.assertEqual(User.objects.count(), 0)
    response = self.client.post(self.list_url, data=data_user)
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertEqual(User.objects.count(), 1)
    user = User.objects.all().first()
    for field_name in data_user.keys():
        self.assertEqual(getattr(user, field_name), data_user[field_name])
```

Solution => On enlève UUID dans les modèles,on doit seulement envoyer des requêtes POST avec en paramètre les autres champs (autre que id), et on utilise la génération d'identifiants de base dans Django
Mise en place du projet sur Postman afin d'effectuer des tests de endpoints

SimpleRouter

This router includes routes for the standard set of `list`, `create`, `retrieve`, `update`, `partial_update` and `destroy` actions. The viewset can also mark additional methods to be routed, using the `@action` decorator.

URL Style	HTTP Method	Action	URL Name
{prefix}/	GET	list	{basename}-list
	POST	create	
{prefix}/{url_path}/	GET, or as specified by `methods` argument	`@action(detail=False)` decorated method	{basename}-{url_name}
{prefix}/{lookup}/	GET	retrieve	{basename}-detail
	PUT	update	
	PATCH	partial_update	
	DELETE	destroy	
{prefix}/{lookup}/{url_path}/	GET, or as specified by `methods` argument	`@action(detail=True)` decorated method	{basename}-{url_name}

By default the URLs created by `SimpleRouter` are appended with a trailing slash. This behavior can be modified by setting the `trailing_slash` argument to `False` when instantiating the router. For example:

Router utilisé afin de générer automatiquement des endpoints, fournit les opérations de base (Get all, Get by id, Put, Delete, …)
A modifier éventuellement en fonction des besoins futurs

Indexs par défaut des endpoints ====> tableau récapitulatif des endpoints à faire

```
........

/polls/clothingtype/
/polls/usermodel/
/polls/company/
/polls/companymodel/
/polls/size/
/polls/user/
....
```

Nous avons terminé la création des endpoints de base pour les models et effectué des tests de non régression
(Créer, Supprimer, Obtenir un/des models)
Setup précis de Postman reste à faire demain

14/03/2022

Travail de conception : de quels endpoints auront nous besoin ?

Recherche sur comment créer une base de donnée contenant des fakedata

15/03/22

Setup Postman fait

Données par défaut pour la database, on peut maintenant supprimer et repeupler la BDD grâce à un fichier de fixtures
La procédure à suivre est détaillée dans un fichier sur le drive

16/03/22

Backend:
Ajout d'un endpoint pour le UserModel permettant le traitement des données lorsque l'application mobile envoie au backend les dimensions du vêtement de la garde robe de l'acheteur, avec les dimensions de l'objet témoin.
Le serveur effectue une modification du UserModel reçu avant de le sauvegarder dans la base de données afin d'obtenir les distances en centimètres entre les points indiqués sur le vêtement.
On peut ainsi retrouver la taille des épaules, des jambes … avec seulement les coordonnées des points sur le vêtement et celles de l'objet témoin (pour l'objet témoin, la distance entre les deux points est indiquée à la main pour le moment)

17/03/22

Rédaction du rapport et ajustements finaux pour le backend, ajout de quelques tests unitaires et pour le nouveau endpoint api/polls/usermodel/savedimensions/