

# Projet chef d'oeuvre : détecteur de signatures

Titre professionnel "Développeur en intelligence artificielle" de niveau 6 enregistré au RNCP sous le n°34757

Passage par la voie de la formation - parcours de 19 mois achevé le 20 octobre 2020

Cédric Soares

Décembre 2020

<b>Contexte projet</b>	<b>4</b>
Les équipes 3MU et CATOP	4
Besoin exprimé pour le projet de détecteur de signatures manuscrites	5
Préconisation de solution	5
<b>Gestion de projet</b>	<b>6</b>
Pilotage projet	6
Méthodologie projet	6
Outils mis en oeuvre durant le projet	8
Reporting	9
<b>Analyse du besoin</b>	<b>9</b>
Reformulation des personas	9
Découpage fonctionnel	10
Le périmètre fonctionnel de détection :	11
Le périmètre fonctionnel d'entraînement	11
Diagramme d'activité	12
Backlog produit	13
<b>Choix de la solution d'intelligence artificielle</b>	<b>13</b>
Analyse des sources	14
Benchmark des modèles de détection d'objets en temps réel	14
Comparaison des performances dans l'article de recherche	15
Adoption du modèle par la communauté	16
Architecture de YoloV4	16
<b>Données utilisées pour l'entraînement du modèle</b>	<b>17</b>
Construction de la base de données analytique	17
Exploration des données	18
Nettoyage des données pour l'entraînement	21
Entreposage dans une base de données pour l'entraînement	22
<b>Entraînement du modèle</b>	<b>24</b>
Process mis en oeuvre	24
Analyse des résultats	25
<b>Intégration du modèle dans une API</b>	<b>26</b>
Fonctionnement de l'API	26
<b>Conception et implémentation d'une base de données relationnelle</b>	<b>27</b>
Conception et normalisation	27
Choix de solution technique et implémentation	28
<b>Développement du backend de l'application</b>	<b>31</b>
Gestion des sessions utilisateurs	32
<b>Design et intégration du frontend</b>	<b>33</b>
<b>Définition des éléments critiques et monitoring</b>	<b>35</b>
Monitoring de l'application	36

<b>Stratégie de tests</b>	<b>37</b>
<b>Stratégie de déploiement</b>	<b>41</b>
Conteneurisation	41
Déploiement	41
<b>Annexe I : Expression de besoin</b>	<b>43</b>
<b>Annexe II : Sprints Backlogs</b>	<b>49</b>
<b>Annexe III : E-mails de compte-rendus des Sprints Reviews</b>	<b>51</b>
Compte-rendu du sprint 0	51
Compte-rendu du sprint 1	53
Compte-rendu du sprint 2	56
Compte-rendu du sprint 3	59
Compte-rendu du sprint 4	60
Compte-rendu du sprint 5	62
<b>Annexe IV : Notebook de visualisations sur le base de donnée analytique</b>	<b>65</b>
<b>Annexe V : Script d'ingestion des données et des fichiers de configuration et de poids</b>	<b>70</b>
<b>Annexe VI : Script d'entraînement du modèle</b>	<b>78</b>
<b>Annexe VII: Mockups du frontend</b>	<b>82</b>
Home avec un utilisateur non authentifié	83
Sign up	83
Login	83
Home avec un utilisateur authentifié	84
Predict après sélection d'image	85
Predict après récupération du résultat	85
About	85
Your stats	86
<b>Annexe VIII : Code des tests unitaires</b>	<b>87</b>
<b>Annexe IX : Dockerfiles et Docker-compose</b>	<b>91</b>
Dockerfile du conteneur d'entraînement	92
Docker-compose d'entraînement	92
Dockerfile de l'interface graphique	95
Dockerfile de l'api	96
Docker-compose de l'ensemble interface graphique et api	99

# 1. Contexte projet

Le présent projet a été réalisé dans le cadre de mon alternance au sein de l'opérateur de télécommunications Orange. De début novembre 2019 à fin octobre 2020 j'ai intégré l'équipe 3MU au sein de l'entité DEVRAP. Sur la même période Vitali Shchutski, autre apprenant de la formation développeur en intelligence artificielle à quant à lui intégré une autre équipe de DEVRAP : CATOP . Nous avons tous les deux travaillé sur le projet.

## Les équipes 3MU et CATOP

L'entité DEVRAP (acronyme de développements rapides) fait partie de la Direction de la Technique et du Système d'Information (DTSI) d'Orange France. Composée de 850 personnes, les équipes assurent la maîtrise d'œuvre de projets de développement informatique sur des projets à échéance court voir moyen terme. L'échelle de temps s'échelonne de quelques semaines à quelques mois.

Les équipes 3MU et CATOP sont spécialisées en développement d'automates logiciels utilisant des technologies RPA (Robotic Automation Process) . 3MU développe des automates déployés sur des postes métiers. Ceux développés par CATOP sont déployés sur des serveurs.

À titre d'exemple, certains automates sont utilisés en boutique, lors de la souscription de nouveaux forfaits, pour récupérer les données clients dans différentes applications métiers.

Les équipes 3MU et CATOP sont positionnées comme des centres d'optimisation de coûts. Elles travaillent de concert et partagent le même management. Elles sont composées au total de 16 personnes. Dans le détail :

- 2 managers
- 9 développeur.s.e.s d'automates
- 2 développeur d'intelligences artificielles (Vitali et moi)
- 1 Administrateur système
- 1 DevOps
- 1 Product owner

Courant 2019, les équipes 3MU et CATOP ont fait le choix de monter en compétence sur des sujets d'intelligence artificielle. En effet, les automates sont limités dans leurs capacités au traitement d'informations existantes et à l'application de règles métier. Certains besoins remontés par les entités métier d'Orange France ont mis en lumière la nécessité d'implémenter des briques d'intelligence artificielles,

Vitali et moi-même avons été recrutés afin d'accompagner les équipes dans leur montée en compétence et réaliser des prototypes.

## Besoin exprimé pour le projet de détecteur de signatures manuscrites

La direction DC2P France d'Orange souhaite optimiser son workflow de souscription de ses offres sur le segment Pro PME (professions libérales, autoentrepreneurs, PME). Ce segment se caractérise par un process de ventes à la fois moins direct et plus compliqué que le marché des particuliers. D'une part, le souscripteur d'une offre n'est pas forcément l'utilisateur de cette dernière. D'autre part, les contraintes réglementaires sont plus importantes. En résulte un nombre de documents à traiter plus important que pour la clientèle grand public. De plus, sur ce marché, plusieurs démarches, tel que la signature des contrats sont totalement numérisées. Ce qui n'est pas le cas sur le segment Pro PME.

Jusqu'à présent, l'activation des contrats est conditionnée par une vérification manuelle de la signature des documents. Étant donné le contexte, ce procédé génère à la fois un coût important mais aussi un goulot d'étranglement au niveau des étapes de vérification.

Afin d'optimiser l'existant, l'opérateur souhaite automatiser la détection des signatures sur les différents documents commerciaux utilisés sur le segment Pro PME. La direction commerciale d'Orange France a demandé aux DEVRAP (Développements Rapides) 3MU et CATOP de la direction de la technique et du système d'information de réaliser l'étude d'une solution possible et son prototypage.

L'expression de besoin émise la direction DC2P est mise à disposition en annexe I.

## Préconisation de solution

Pour répondre à ce besoin, les équipes DEVRAP préconisent de développer un détecteur d'objets basé sur un algorithme de classification mis en œuvre par un modèle deep learning de Computer Vision (vision assistée par ordinateur). Sa finalité est de détecter si les documents scannés contiennent une ou plusieurs signatures. De plus, le cas échéant l'algorithme devra déterminer leur nombre et leur emplacement sur chaque document.

Le prototype sera livré sous forme d'API. Celle-ci pourra être interrogée via une application web ou un démonstrateur motorisé par un kit de développement Jetson Nano équipé d'une webcam.

La solution sera évaluée à la fois par la performance de ses résultats et sa réactivité.

## 2. Gestion de projet

### Pilotage projet

Conformément à l'expression de besoin, la maîtrise d'ouvrage a été prise en charge par le management des équipes DEVRAP à savoir :

- Philippe Bernard (CATOP)
- Samy Damien (3MU)

Vitali et moi-même avons réalisé la maîtrise d'œuvre du projet de manière étendue. Celle-ci comprend :

- La définition des méthodologies de projet à mettre en oeuvre et leur application
- Le benchmark de la solution d'IA qui motorise l'API
- Les différents entraînements de la solution d'IA
- Le développement du prototype
- La conteneurisation et le déploiement du prototype
- L'administration du prototype

### Méthodologie projet

Au vu de la contrainte temps (livraison en 5 mois d'un prototype fini) et le besoin de montée en compétences sur une grande partie des technologies et processus à mettre en œuvre, nous avons fait le choix d'appliquer les principes des méthodes agiles. Ceux-ci préconisent des livraisons de blocs fonctionnels par itérations. Celles-ci permettent de structurer le projet et d'avoir des retours échelonnés dans le temps.

Notre choix s'est plus précisément arrêté sur la méthode Scrum. Toutefois, n'étant que deux porteurs des travaux, nous avons dû adapter certains artefacts.

Dans la méthode Scrum originelle plusieurs acteurs interviennent dans le projet :

- Le Scrum master : est garant des principes du Scrum.
- Le Product owner : est responsable du périmètre fonctionnel du produit et reporte aux équipes métier.
- Les développeurs : assurent le développement du produit.

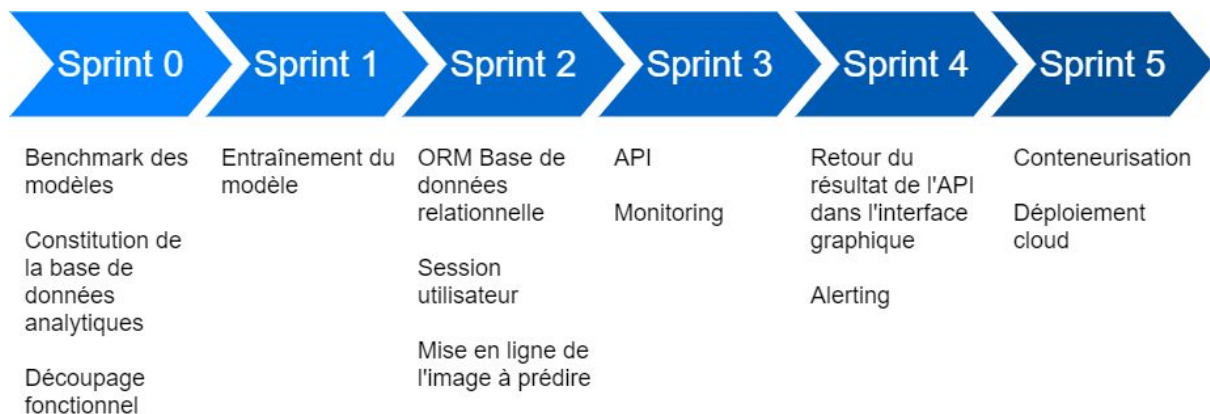
N'étant que deux, Vitali et moi, nous sommes consacrés au rôle de développeurs. La maîtrise d'ouvrage étant réalisée au sein de l'équipe. Le besoin de reporting auprès des équipes métier devient inexistant.

Nous avons conservé :

**Le découpage projet en sprints** : nous avons choisi une durée de 3 semaines par itération. Ce temps permet :

- La délimitation fonctionnelle du périmètre fonctionnel du sprint
- Le développement du périmètre fonctionnel
- La réalisation de tests fonctionnels et le cas échéant de tests techniques liés au périmètre fonctionnel
- La présentation du périmètre fonctionnel

Le projet a été réalisé en 6 sprints : 1 sprint de benchmark / d'essais de solutions et 5 sprints de développement. Voici comment le découpage des sprints dans le temps :

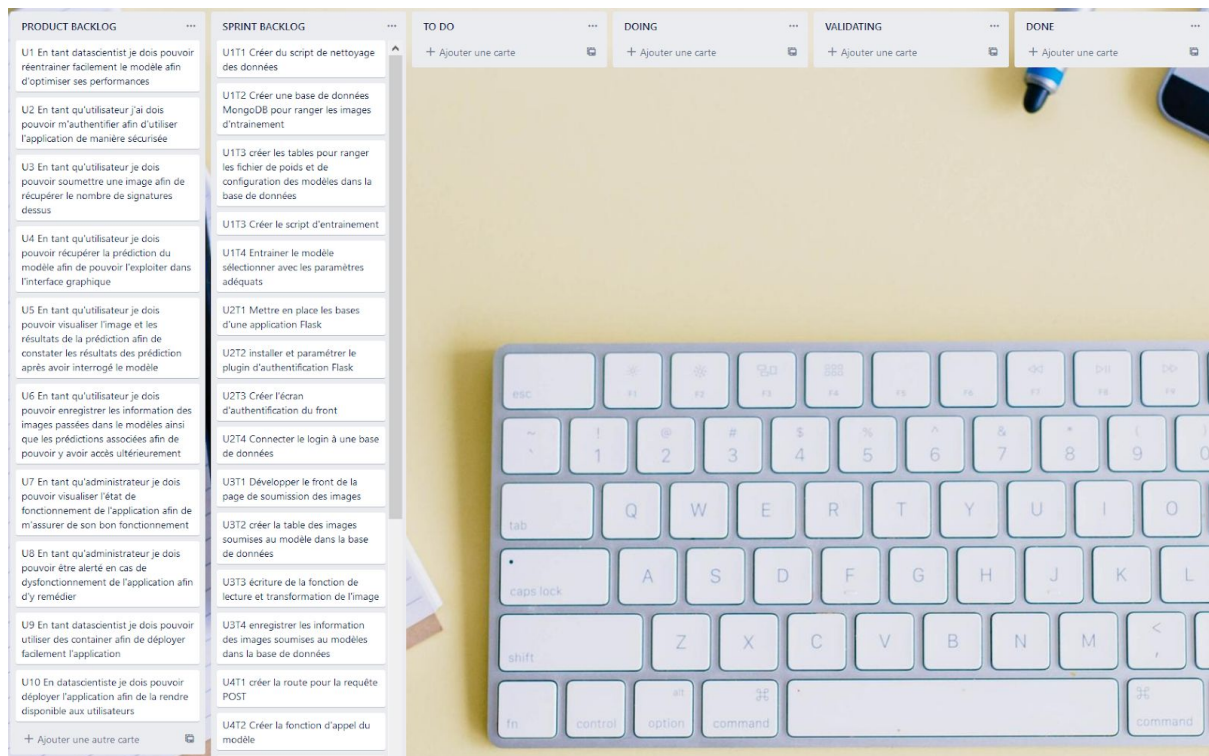


**Le scrum board** : Ce dernier comprend le périmètre fonctionnel de l'application découpé sous forme d'user stories. Il comprend également un sprint backlog regroupant le périmètre le découpage en tâche des user story développée dans l'itération. Enfin des colonnes de suivi suivantes :

- TO DO : tâche assignée mais non encore commencée
- DOING : tâche en cours de développement
- VALIDATING : tâche en cours de validation (test fonctionnel et/ou technique)
- DONE : Tâche développée et validée

Chaque tâche se voit attribuer une pondération de 1 à 12 relative à sa difficulté estimée. Nous considérons que ce procédé reflète une charge temps perçue pour réaliser la tâche.

Ci-dessous le Scrum board au début du projet



Scrum board du projet à son début

L'ensemble des user story est détaillée dans l'analyse du besoin client en partie 3. Une capture de chaque Sprint Backlog est mise à disposition dans l'annexe II.

**Les instances de réunion :** nous avons conservé une majorité des rituels de la méthode Scrum :

- Le daily meeting : point quotidien 15 minutes sur le réalisé de la veille et le à réaliser du jour.
- La planification de sprint : à chaque début de sprint l'équipe définit le périmètre fonctionnel qu'elle va réaliser, les priorités et réalise les l'estimation de charge de chaque tâche.
- La revue de sprint : L'équipe réalise la présentation du périmètre fonctionnel implémenté au management DEVRAP en charge de la maîtrise d'ouvrage.

Faute de présence d'un Scrum master et vu la taille réduite, nous n'avons pas conservé la rétrospective du sprint. Ce rituel permet, dans une utilisation rigoureuse du scrum, de faire le point sur l'analyse du sprint sous le prisme des pistes d'amélioration à mettre en œuvre dans les pratiques de l'équipe.

Outils mis en oeuvre durant le projet

- Trello : pour le scrum board
- Gitlab : pour versionning et le partage du code
- Virtual Studio Code : comme IDE pour la production du code



## Reporting

À l'issue de chaque revue de sprint, un compte-rendu a été envoyé à Philippe Bernard et Samy Damien. ce dernier contenait :

- Les points principaux abordés pendant la revue de sprint
- La capture d'écran du scrum board
- Le burndown chart permettant de visualiser la vélocité de l'équipe

L'ensemble des comptes-rendu est mis à disposition en annexe III.

## 3. Analyse du besoin

### Reformulation des personas

L'expression de besoin (présente en annexe I) définit trois types d'utilisateurs : opérateur, administrateur et mainteneur. Pour les besoins de l'analyse fonctionnelle nous avons adapté la nomenclature et précisé les rôles et leur profile comme suit:

- Opérateur → utilisateur : utilise l'interface graphique pour soumettre des documents scannés et obtenir le résultat de la détection signatures. Il ne dispose pas de compétences en développement ou datasience.
- Mainteneur → datascientist : est en charge des entraînements nécessaires à la vie du prototype. Il a des connaissances générales en datasciences afin d'apprécier les performances du modèle. Par contre, il n'a pas forcément la connaissance des outils / frameworks mis en œuvre pour entraîner le modèle.
- Administrateur → administrateur: supervise le fonctionnement de l'application via un outil de monitoring et reçoit les alertes lors des dysfonctionnements. Il comprend les items monitorés et maîtrise l'expérience utilisateur attendue de l'interface graphique. Par contre, il n'a pas nécessairement de compétences en développement ou datasciences.

## Découpage fonctionnel

L'expression de besoin a été traduite en diagramme UML de cas d'usages

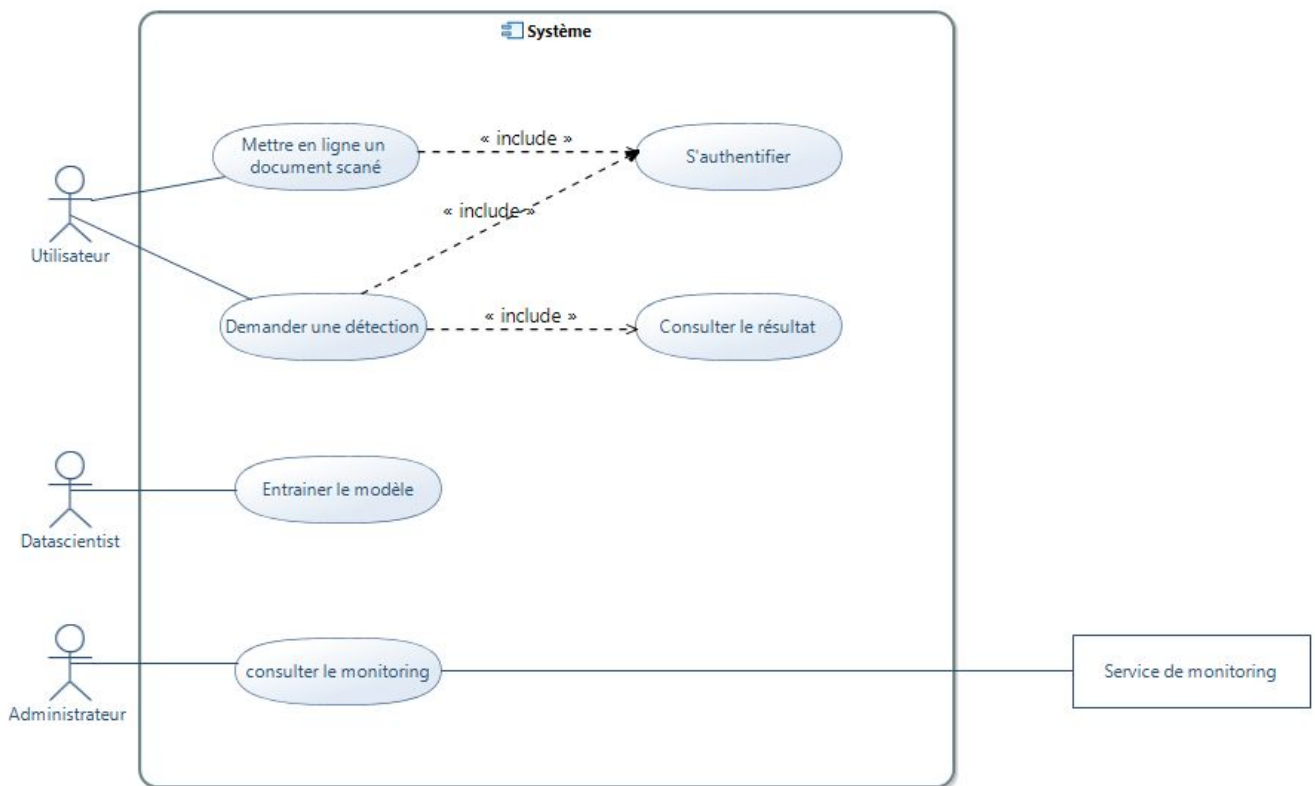


Diagramme de cas d'usage. Source: *UML Designer*

Il apparaît qu'il n'y a aucun héritage entre les acteurs et qu'aucun n'utilise les mêmes cas d'usage. De plus les cas d'usage liés à l'utilisateur et ceux au datascientists nécessitent des ressources différentes :

- Cas d'usages liés à l'utilisateur : Nécessitent une base de données relationnelle afin de gérer les sessions utilisateur, stocker les métadonnées des images et les résultats de la détection. Ils ne nécessitent pas l'utilisation d'un GPU.
- Cas d'usage lié au datascientist : Nécessite l'utilisation d'une base de données analytique pour stocker les documents nécessaires à l'entraînement du modèle. L'utilisation d'un modèle de deep learning de computer vision s'appuie sur l'utilisation du GPU.

Ces deux types d'usages ont fait l'objet d'un développement de périmètres fonctionnels distincts . Par convention nous appelons périmètre fonctionnel de détection et périmètre fonctionnel d'entraînement.

## Le périmètre fonctionnel de détection :

L'expression de besoin demande explicitement que le périmètre fonctionnel détection comprenne graphique utilisable par un utilisateur ainsi qu'une Api pouvant être interrogé directement.

Le diagramme de package du périmètre fonctionnel de détection traduit cette exigence.

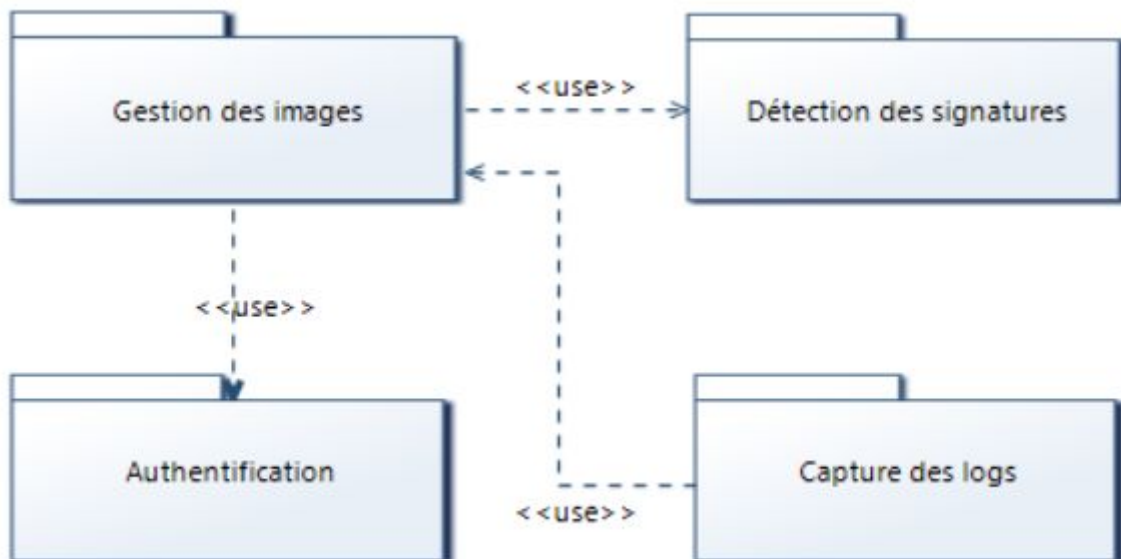


diagramme de package du périmètre fonctionnel de détection . Source : UML Designer

Dans ce périmètre, la gestion des images, l'authentification et la capture de logs sont intégrés dans l'application servant l'interface graphique. La détection des images est prise en charge par une Api.

Le monitoring à proprement parler est assuré par la plateforme Airbrake détaillée en partie 11. Elle s'appuie sur une brique logicielle intégrée dans l'interface graphique.

## Le périmètre fonctionnel d'entraînement

L'entraînement du modèle étant une activité ponctuelle mais utilisant des ressources propres, son implémentation est réalisée par l'utilisation d'un script simple appelant la base donnée analytique.

## Diagramme d'activité

La séquence de détection des signatures a été modélisée sous forme de diagramme d'activité.

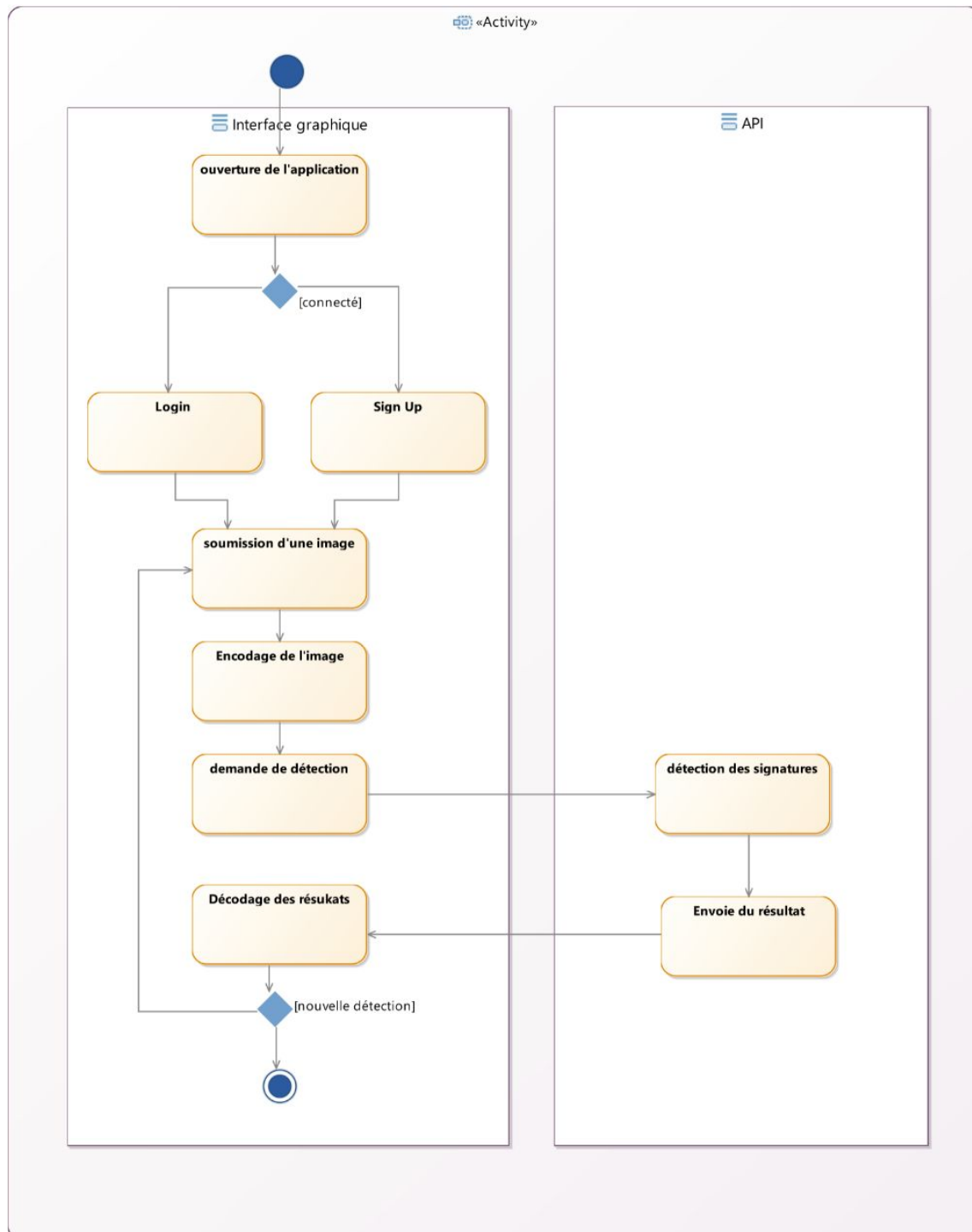


Diagramme d'activité du périmètre fonctionnel de détection. Source: UML Designer

Particularité : L'interface graphique encode les scans de documents en bytes pour les envoyer à l'Api. Elle décode également les cadre de signatures détectées par l'api qui lui sont renvoyé également sous cette forme.

## Backlog produit

Le travail d'analyse du besoin nous a amené à formaliser le backlog du produit découpé en user stories. Ci-dessous la vue des user story avec le périmètre fonctionnel auquel chacune renvoie.

User Story	Périmètre fonctionnel
U1 En tant datascientist je dois pouvoir (ré)entraîner facilement le modèle afin d'optimiser ses performances	Entraînement
U2 En tant qu'utilisateur j'ai dois pouvoir m'authentifier afin d'utiliser l'application de manière sécurisée	Détection
U3 En tant qu'utilisateur je dois pouvoir soumettre une image afin de récupérer le nombre de signatures dessus	Détection
U4 En tant qu'utilisateur je dois pouvoir récupérer la prédiction du modèle afin de pouvoir l'exploiter dans l'interface graphique	Détection
U5 En tant qu'utilisateur je dois pouvoir visualiser l'image et les résultats de la prédiction afin de constater les résultats des prédiction après avoir interrogé le modèle	Détection
U6 En tant qu'utilisateur je dois pouvoir enregistrer les information des images passées dans le modèles ainsi que les prédictions associées afin de pouvoir y avoir accès ultérieurement	Détection
U7 En tant qu'administrateur je dois pouvoir visualiser l'état de fonctionnement de l'application afin de m'assurer de son bon fonctionnement	Détection
U8 En tant qu'administrateur je dois pouvoir être alerté en cas de dysfonctionnement de l'application afin d'y remédier	Détection
U9 En tant que datascientist je dois pouvoir utiliser des container afin de déployer facilement l'application	Détection et Entraînement
U10 En tant que datascientist je dois pouvoir déployer l'application afin de la rendre disponible aux utilisateurs	Détection

Chaque user story a fait l'objet d'un découpage en tâches. Les tâches réalisées dans chaque Sprint sont listées dans les comptes-rendus mis à disposition dans l'annexe III.

## 4. Choix de la solution d'intelligence artificielle

Pour répondre au besoin de l'expression de besoin, nous avons préconisé de développer un détecteur d'objets basé sur un algorithme de classification mis en œuvre par un modèle

deep learning de Computer Vision (vision assistée par ordinateur). La solution sera évaluée à la fois par la performance de ses résultats et sa réactivité. Pour déterminer le meilleur modèle, nous avons réalisé une étude de l'état de l'art.

## Analyse des sources

Voici la liste des sources que l'équipe DEVRAP a utilisé pour procéder à l'étude :

- Papers with code
- arxiv
- Github

Dans le détail :

**Papers with code** : La plateforme destinée aux chercheurs en machine learning a été lancée en 2018. [Selon ses fondateurs Robert Stojnic et Ross Taylor](#), elle regroupait plus de 18 000 publications et 1 500 classements lors de son acquisition par le laboratoire d'intelligence artificielle de Facebook, en décembre 2019. La source réalise des benchmarks évaluant les meilleurs modèles sur des thématiques spécifiques des différents domaines d'application du machine learning.

**arXiv** : Le site regroupe plus 1,7 million d'articles scientifiques et est géré par l'université Américaine de Cornell. Il servira à avoir accès à la publication de recherche du modèle retenu.

**Github** : Le service racheté par Microsoft en 2018 héberge 100 millions de repository et est utilisé par 50 millions d'utilisateurs ([chiffres donnés par la plateforme en août 2019](#)). Github sera utilisé pour vérifier l'adoption du modèle retenu par la communauté des développeurs et data scientists.

Que ce soit par leur objet, leur audience ou leur propriétaire, ces trois sources s'avèrent être un gage de confiance quant aux informations que l'équipe DEVRAP a pu analyser.

## Benchmark des modèles de détection d'objets en temps réel

Pour évaluer le meilleur équilibre performance réactivité, l'équipe DEVRAP s'est basée sur le benchmark "Real-Time Object Detection on Coco" [réalisé par Papers with code en mai 2020](#).

Le classement évalue les modèles selon trois indicateurs :

- mAP (mean Average precision): moyenne du rapport précision sur recall sur l'ensemble des classes du dataset étudié.
- FPS (Frame par seconde): Nombre d'images par seconde que le modèle peut traiter
- Inference Time: Temps de réponse du modèle en millisecondes.

En s'appuyant sur les critères d'équilibre l'équipe DEVRAP s'est fixé, YoloV4 apparaît être le modèle offrant le meilleur équilibre performance / réactivité.

En effet, d'après le benchmark réalisé sur le dataset Microsoft COCO, YoloV4, sur des images d'entrée de dimensions 512x512, arrive 7e du top 20 pour la métrique avec un mAP à 43 alors que le premier EfficientDet à 55.1.

Sur ce simple critère, le modèle fait moins bien que d'autres comme EfficientDet par exemple et ne paraît donc pas être le meilleur choix. Par contre sur les deux items que sont le FPS et l'Inférence Time, suivant YoloV4 arrive premier du top 20

En considérant le nombre d'images par seconde, avec un FPS à 83 sur des images de 512x512, YoloV4 offre un écart significatif avec le modèle suivant dans le top 20 : CSPResNeXt. Ce dernier affiche une performance de 58 FPS.

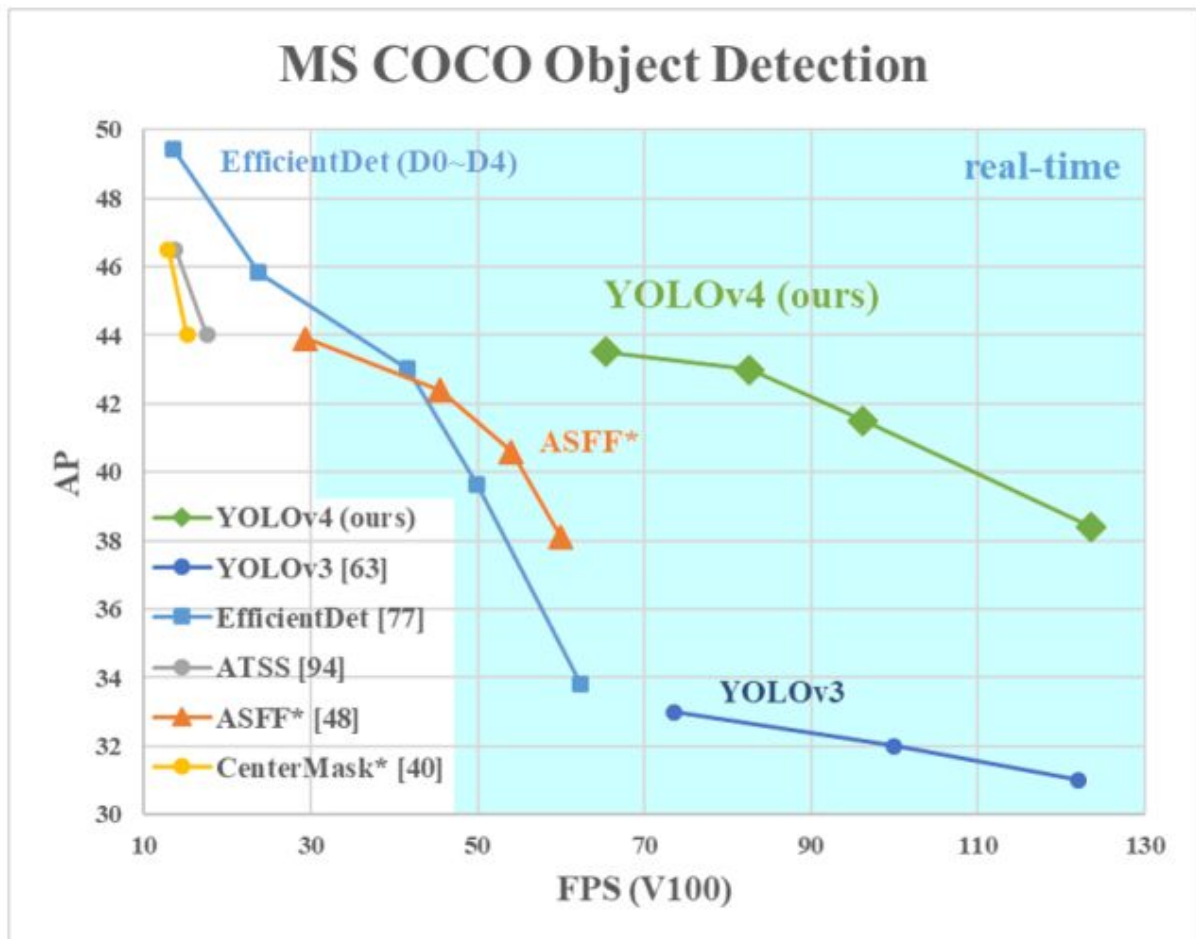
Voici le récapitulatif des informations extraites du benchmark.

	<b>Score mAP (Position)</b>	<b>Score FPS (Position)</b>	<b>Score Inference Time (Position)</b>
EfficientDet-D7x	55.1 (1e)	6.5 (18e)	Pas de mesure dans le benchmark
CSPResNeXt	33.4 (17e)	58 (3e)	17 (3e)
Yolov4 - 512	43 (7e)	83 (1e)	12 (1e)

## Comparaison des performances dans l'article de recherche

La [publication des travaux d' Alexey Bochkovskiy, Chien-Yao Wang et Hong-Yuan Mark Liao](#), co-créateurs de YoloV4, a été soumise sur arXiv le 23 avril 2020.

Dans leur article, les auteurs ont détaillé le rapport performance / FPS entre YoloV4 et d'autres modèles à la pointe de la recherche dont notamment EfficientDet plus performant selon le benchmark de Papers with code.



Comparaison du rapport performance / FPS avec un GPU Tesla V100 entre YoloV4 et d'autres modèles de deep learning à la pointe de la recherche. Source article de recherche [YOLOv4: Optimal Speed and Accuracy of Object Detection](#) (publié sur arXiv le 23 avril 2020)

Il apparaît au vu de la courbe que les performances de YoloV4 baissent beaucoup moins vite avec l'augmentation du FPS que EfficientDet. Le modèle reconnaît deux fois plus à un niveau de performance équivalent selon la publication.

## Adoption du modèle par la communauté

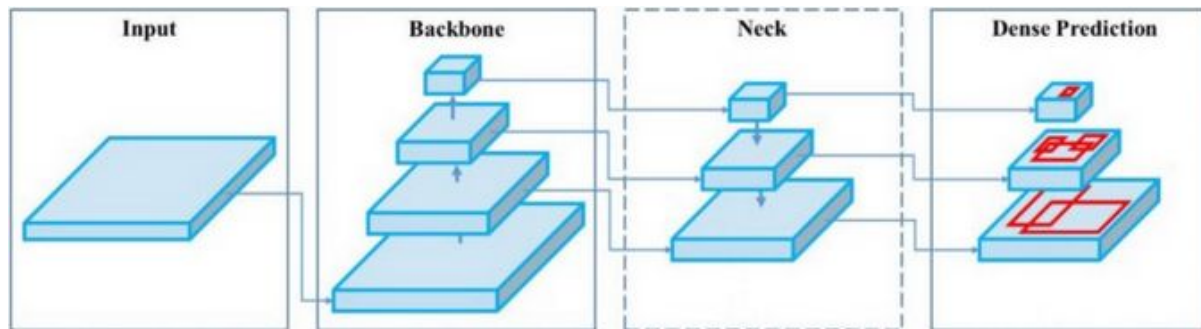
En plus de répondre au critère de sélection, le modèle est également largement utilisé par la communauté des data scientists. [D'après Papers with code](#), hormis la publication originale, depuis avril 2020, YoloV4 a été utilisé dans 144 autres articles de recherche.

De plus, d'après les statistiques Github le [repository lié aux travaux de recherche Alexey Bochkovskiy](#) est sauvegardé par 13 600 utilisateurs de la plateforme et copié pour modifications par 14 900 utilisateurs.

## Architecture de YoloV4

YoloV4 se décompose en trois blocs fonctionnels comme le détaille de le schéma tiré de la [publication d' Alexey Bochkovskiy, Chien-Yao Wang et Hong-Yuan Mark Liao](#).





Architecture du modèle VoloV4. Source article de recherche [YOLOv4: Optimal Speed and Accuracy of Object Detection](#) (publié sur arXiv le 23 avril 2020)

- **Backbone** : extrait les features de l'image soumise en entrée. Il fournit en sortie une carte de l'ensemble des features présentes dans l'image.
- **Neck** : identifie les features pertinentes.
- **Dense prediction** : identifie la position des features utiles en traçant des bounding box autour. Pour chaque, il donne la nature de l'objet détecté.

## 5. Données utilisées pour l'entraînement du modèle

### Construction de la base de données analytique

Dans le cadre du projet, la DC2P n'a pas fourni de scans de documents commerciaux. Ce choix s'explique par la présence de données personnelles liées à des clients d'Orange tel que : les coordonnées bancaires ou l'adresse postale par exemple. Il fait d'autant plus de sens que les entraînement des modèles n'ont pas été réalisés sur des matériels ou des machines virtuelles intégrées au système d'information de l'opérateur.

Nous avons donc recherché des sources de données répondants aux critères suivants:

- Données en accès publique
- Documents scannés de nature commerciale : contrats, baux
- Une partie des documents comporte des signatures manuscrites

La base de données analytiques que nous avons assemblées en compte trois :

- **[Tobacco-800](#)** : Dataset de 1290 documents, dont 789 annotés, issus de l'industrie du tabac. Constitué par Guangyu Zhu, Yefeng Zheng, David Doermann and Stefan Jaeger, chercheurs au laboratoire LAMP (Language and Media Processing) de l'université du Maryland aux Etats-Unis. Sa première version a été mise en ligne en 2006 et la dernière mise à jour a été réalisée en 2011.
- **[Nanonets](#)** : Dataset de 174 documents issus des moteurs de recherche Google et Bing mis à disposition par Nanonets sur son compte [Github](#). L'entreprise, spécialisée dans les OCR a utilisé le dataset dans un démonstrateur de son API en 2018.

- **GSA Lease documents** : Ensemble des baux commerciaux contractés avec l'administration des services généraux du gouvernement américain afin de fournir des bureaux aux agents fédéraux. Les documents sont classés régions regroupant plusieurs Etats. Nous avons utilisé les bases des 6 régions les plus à l'est du pays. Cet ensemble représente 772 documents.

Dans le cadre du projet nous avons entraîné YoloV4 à trois reprises avec des fichiers de poids et de configuration initiaux afin d'en améliorer les performances. Le premier entraînement a été réalisé sur le dataset Tobacco-800 seul. Nous l'avons ensuite enrichi avec les deux autres sources lors des entraînements suivants. Les détails des performances sont fournis dans la section du rapport.

Les documents fournis Nanonets et GSA Lease documents n'étant pas annotés, nous avons utilisé la librairie [Labellmg](#).

## Exploration des données

Pour les besoin de l'exploration des données, nous avons regroupé les métadonnées des documents suivantes :

- **name** : suite hexadécimale représentant le nom du document
- **height** : hauteur du document
- **width** : largeur du document
- **is\_color** : indique si le scan document est en noir et blanc (0) ou en couleurs (1)
- **number\_of\_signatures** : nombre de signatures sur le document
- **source** : source où est tirée le document

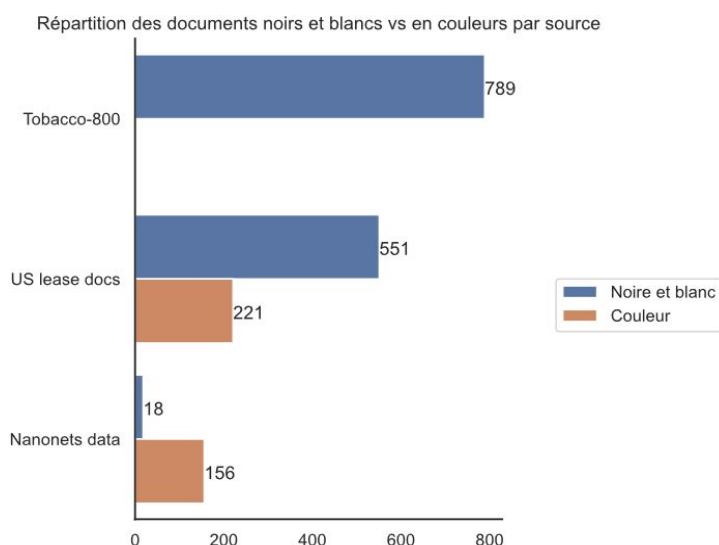
La dernière version de l'exploration présentée dans le rapport regroupe l'ensemble des sources utilisées.

En voici les statistiques descriptives.

	Unnamed: 0	height	width	is_color	number_of_signatures
<b>count</b>	1735.000000	1735.000000	1735.000000	1735.000000	1735.000000
<b>mean</b>	867.000000	2243.940634	1734.374640	0.217291	2.004035
<b>std</b>	500.995675	650.636099	507.187124	0.412521	1.215284
<b>min</b>	0.000000	408.000000	380.000000	0.000000	1.000000
<b>25%</b>	433.500000	2183.000000	1696.000000	0.000000	1.000000
<b>50%</b>	867.000000	2201.000000	1708.000000	0.000000	2.000000
<b>75%</b>	1300.500000	2292.000000	1728.000000	0.000000	3.000000
<b>max</b>	1734.000000	4200.000000	3506.000000	1.000000	8.000000

Statistique descriptive des documents du dataset d'entraînement. Source : notebook ad hoc

Premier constat : malgré la multiplication des sources, la proportion de documents en couleur est de 21,7%. Augmenter le nombre d'observations avec une modalité `is_color = 1` est non seulement difficile mais surtout très chronophage. D'une part, le nombre de datasets de documents signés accessibles sont rares. D'autre part, le détail de la coloration des documents n'est généralement pas détaillé. Il nécessite une exploration empirique de chaque dataset. En examinant plus en détail cette caractéristique.



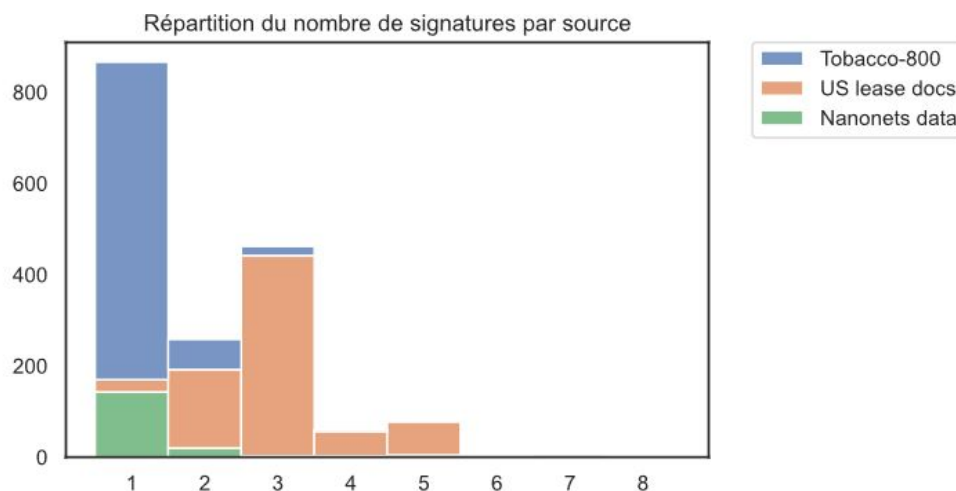
Répartition des documents noirs et blancs vs couleurs par source. Source : *notebook*

Sur le dataset nous avons initié ce travail de sélection d'images en couleurs. Notons l'absence de documents en couleur du dataset Tobacco-800. Nous avons émis deux hypothèse pouvant l'expliquer :

- Une normalisation en noir et blanc par l'équipe scientifique qui a constitué le dataset
- Un dataset plus ancien constitué de documents b2b

Nos travaux ne nous ont pas permis de valider l'un ou l'autre des hypothèses.

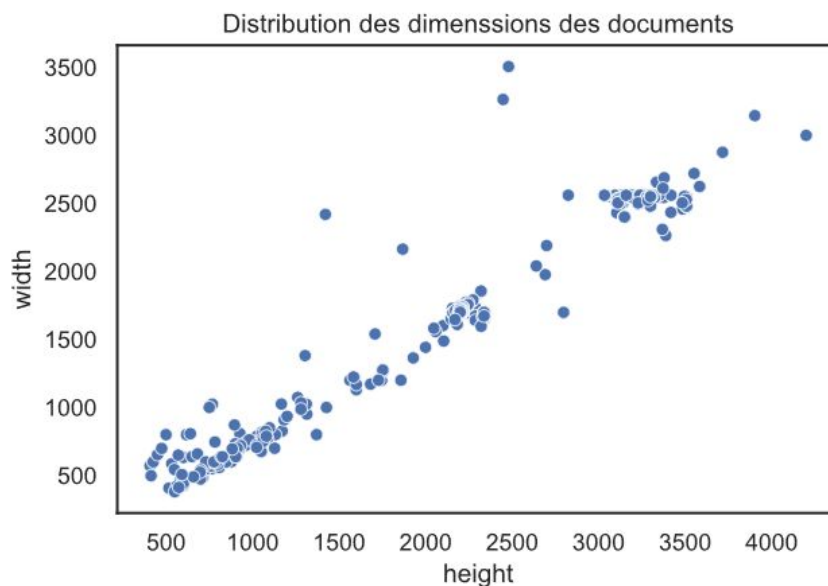
En observant la répartition du nombre de signatures par sources nous obtenons :



Répartition du nombre de signatures par sources. Source : *notebook*

Nous constatons un nombre de signatures par documents sensiblement plus élevé sur le dataset GSA US Lease. Le phénomène peut s'expliquer par la nature des documents. Les baux commerciaux nécessitent un nombre de signatures plus important par documents. De manière empirique nous avons également pu observer que le nombre de signataires par document est également plus important sur ce dataset.

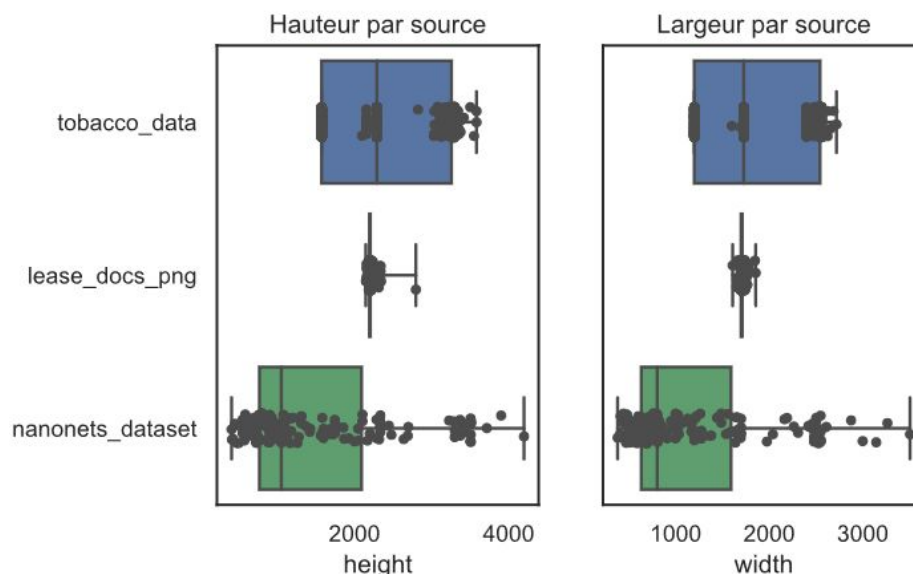
Observons maintenant la distribution des largeur par hauteur sur l'ensemble des documents.



Distribution des dimensions des documents. *Source: notebook*

On constate que la distribution du rapport hauteur / largeur est majoritairement linéaire. Cela s'explique par le fait, que malgré des dimensions différentes, les images, tirées de scans, sont quasiment toutes issues de documents au format A4.

Enfin observons les distribution des dimensions par source :



Distribution des dimensions par source. Source: notebook

Nous constatons une distribution très contenue des dimensions des documents issus du dataset GSA Lease. Le phénomène peut s'expliquer par le fait qu'une administration fédérale aurait tendance à normaliser la taille des documents qu'elle sauvegarde. A contrario la dispersion sur le dataset Nanonets, constitué à partir de résultats de moteurs de recherches, est beaucoup plus importante.

Le notebooks qui a servi à réaliser les visualisations est disponible en annexe IV.

## Nettoyage des données pour l'entraînement

Pour les besoins de l'entraînement, les actions de nettoyage et preprocessing se sont limitées à un redimensionnement des images en 608 x 608 pixels comme le préconise le github officiel du modèle. Pour atteindre cet objectif nous avons utilisé la version compilée pour Python d'OpenCV.

Voici la portion de script utilisée:

```
import os
import cv2

cwd = os.getcwd()

width = 608
height = 608

for filename in os.listdir(cwd):
    source_image = cwd + '/' + filename + '.png'
    source_raw_image = cwd + '/' + filename + '.png'
    destination_resized_image = cwd + 'resized_' + filename + '.png'
```

```
img = cv2.imread(source_image, cv2.IMREAD_UNCHANGED)
resized = cv2.resize(img, (width, height))
cv2.imwrite(destination_resized_image, resized)
```

## Entreposage dans une base de données pour l'entraînement

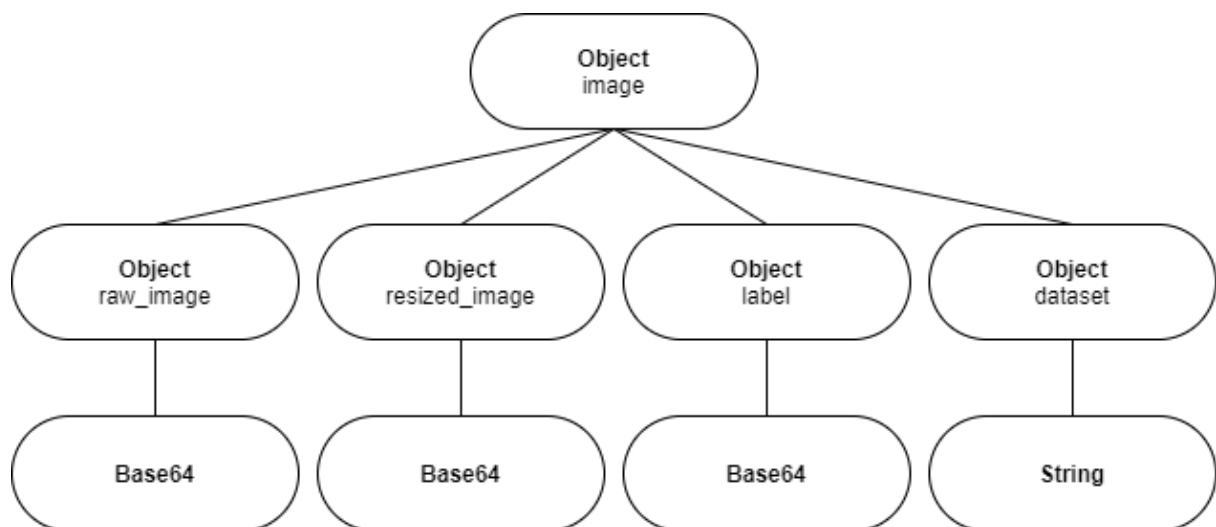
Afin de concevoir une solution, clé en main, déployable rapidement pour gérer les entraînements des modèles, nous avons fait le choix d'entreposer les images dans une base de données. Nos critères de choix pour la technologie à employer étaient les suivants :

- Pas de besoin de relations entre entités où sont entreposées les données
- Une base de données orientée documents
- Pouvant ingérer des documents ou les retourner avec un connecteur simple sans besoin d'implémenter un ORM

Notre choix s'est arrêté sur une base de données MongoDB. Elle permettra non seulement de stocker les images nécessaires à l'entraînement mais aussi les éléments nécessaires à l'entraînement du modèle. Voici la liste de collections que nous avons implémenté :

- **images** : images redimensionnées, utilisées pour l'entraînement
- **config** : fichier de configuration du modèle
- **weights** : fichier de poids du modèle. A noter que chaque fichier de poids pèse plus de 250 Mo. Les collections ne supportant pas les éléments de plus de 16 Mo, nous avons utilisé le système de GridFS qui découpe chaque document en chunks.

Voici la modélisation des collection `input_images_col` et `config_col`.



Modélisation de la collection image regroupant les images

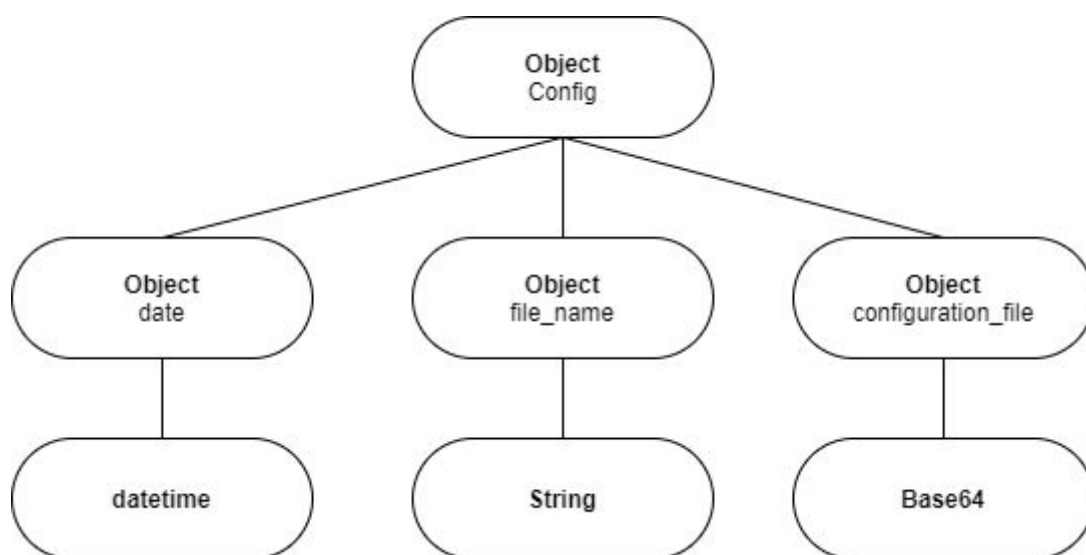
Détail de la collection images :

- **raw\_image** : image initiale encodé en Base64
- **resized\_image** : image redimenssionnée en Base64

- **label** : Fichier texte contenant les coordonnées des bounding box de signature labellisées. Encodé en Bas64
- **dataset** : nom du jeu de données. Peut prendre les valeurs “train”, “valid”, “test”

Soit sous la forme JSON :

```
{
  'raw_image' :
  'resized_image':
  'label':
  'dataset':
}
```



Modélisation de la collection des fichier

Détail de la collection config:

- **Date**: Date d'enregistrement du fichier de configuration
- **file\_name**: Nom du fichier de configuration
- **configuration\_file**: Fichier de configuration encodé en Base64

Soit sous la forme de JSON:

```
{
  'date':
  'file_name':
  'configuration_file':
}
```

Le script d'ingestion de données est disponible en annexe V

## 6. Entraînement du modèle

### Process mis en oeuvre

Le pilotage de l'entraînement réalisé par Darknet qui fait office de backbone de YoloV4. Le framework est appelé en lui passant les fichiers suivants :

- **detector** : Contenu dans le container du script d'entraînement. Objet contenant l'architecture de YoloV4..
- **obj.data** : Construit en extrayant l'ensemble des images des datasets train et valid de la collection images base de données. Il comprend : Des objets à détecter, le nombre de classes et les listings distincts des fichiers du dataset train et valid (images + labels).
- **culstom-yolov4-detector.cfg** : Extrait de la collection configs de la base de données. Fichier de configuration listant les hyperparamètres du modèles tels que les dimensions des images, leur angle de rotation, le learning rate ou le nombre maximum que le modèle va utiliser pour entraîner le modèle.

Darknet est compilé dans un container dédié à l'entraînement où sont également présent le notebook contenant le script. La base MongoDB est hébergée dans un autre container.

Le script d'entraînement est disponible en annexe VI.

### Analyse des résultats

Sur l'ensemble du projet, nous avons entraîné à trois reprises la version YOLOv4 préentraînée vierge :

- Premier entraînement sur la base dataset Tobacco-800 seul : 789 images
- Second entraînement sur la base dataset Tobacco-800 (789 images) complété par le dataset Nanonets (174 images) et 150 images collectée pour la dataset USA GSA Lease soit un total de 1113 images
- Troisième entraînement sur l'ensemble des images Tobacco-800, Nanonets et USA GSA Lease que nous avons collecté soit 1735 images

L'ajout d'images, avec parmi celles-ci des scans en couleur, a permis d'augmenter les performances du modèle. Voici le tableau regroupant les résultats des tests réalisés sur le dataset de validation au terme de chaque entraînement :

	<b>mAP</b>	<b>Precision</b>	<b>Recall</b>	<b>F1 Score</b>
Premier entraînement	29.57%	44%	25%	32%
Deuxième entraînement	65.06%	52%	75%	62%
Troisième entraînement	68.20%	71%	69%	70%



On constate que chaque nouvel entraînement a vu augmenter le mAP. Pour rappel il s'agit du rapport Precision / Recall moyen. L'ajout de scans en couleur et le passage de un à trois canaux de couleurs dans les hyperparamètres lors du second entraînement ont permis d'augmenter très sensiblement les performances du modèle sur l'ensemble des critères d'évaluation.

L'augmentation significative du nombre de scans lors du troisième entraînement s'est certes traduite par une augmentation du mAP, de la précision et du F1 Score. Le Recall a toutefois baissé. On en déduit que le modèle est moins capable de détecter un document où il y a effectivement une ou plusieurs signature(s) sur l'ensemble des documents qui sont effectivement signés. En ramenant le prototype à son enjeu business, il y a un risque plus important que des processus de souscription soient bloqués faute de signature détectés alors qu'ils auraient dû se poursuivre.

Lors des tests que nous avons réalisés nous avons pu constater que le modèle avait tendance à se tromper la détection ou le nombre de signatures détectées dans deux cas précis :

- Les signatures qui ont fait l'objet d'un coup de tampon
- Les signatures entourées par des annotations manuscrites. Ce cas se présente dans le cas de mentions légales recopiées par exemple.

Faute de contact avec les équipes métiers à la source du besoin, nous ne sommes pas en mesure de dire si ces cas sont courants ou non en situation réelle. De plus, le jeu de données que nous avons constitué comporte un biais vis-à-vis des documents collectés. Pour ces deux raisons, nous avons décidé de ne pas poursuivre les entraînements. Il serait judicieux de réentraîner le modèle avec un nombre significatif de documents issus de la direction DC2P.

## 7. Intégration du modèle dans une API

Conformément à l'expression de besoins, nous avons développé une API contenant le modèle. Nous avons utilisé le framework python FastAPI pour le réaliser. Celui-ci propose plusieurs avantages :

- Une définition des payload attendus par requête simplifiée, sous forme de classes.
- Une documentation OpenAPI/swagger intégrée
- Un temps nécessaire au développement de la solution plus court que Flask

### Fonctionnement de l'API

L'API prend en entrée un JSON contenant une image transformée en String Base64. Elle retourne sous forme de JSON les coordonnées des bounding box des signatures détectées et leur nombre. L'opération est réalisée via la route /predict.

La prédiction est réalisée à l'aide d'openCV qui utilise le modèle YoloV4 entraîné. Nous avons réalisé à la fois des tests unitaires et fonctionnels :

- Tests unitaires : api renvoie bien un code HTTP 200 ainsi qu'un JSON contenant des coordonnées lorsqu'elle est appelée
- Test fonctionnel : lorsqu'une image encodée est envoyée à l'API, l'application front est capable de décoder les bounding box renvoyées par l'API et les afficher sur l'image envoyée.

L'ensemble des tests de l'API et de l'interface graphique sont détaillées dans la section 12 du rapport .

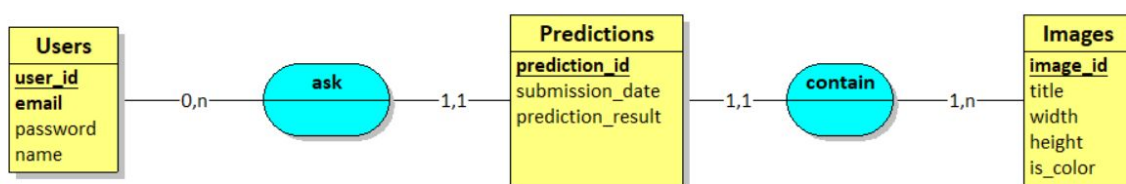
## 8. Conception et implémentation d'une base de données relationnelle

### Conception et normalisation

Afin de stocker les informations nécessaires au fonctionnement des applications nous avons implémenté une base de données SQLite. Celle-ci comprend trois tables :

- Users: stocke le mail l'identifiant et un mot de passe haché de chaque utilisateur
- Predictions: stocke la date des prédictions, le nombre de signatures détectée, l'id de l'utilisateur qui a réalisé la prédiction ainsi que l'id de l'image qui a servi à la prédiction.
- Images: stocke les métadonnées liées aux images soumises pour prédiction

Voici le MCD de la base de données :



MCD de la base de données. Source : Looping

Quelques remarques sur les cardinalités :

- Un utilisateur n'a pas besoin d'avoir réalisé une prédiction pour être enregistré en base de données. Il peut donc créer un compte et se connecter et réaliser sa première prédiction dans deux sessions différentes.
- Chaque prédiction ne peut être basée que sur une image et être associée qu'à un seul utilisateur

Il en découle deux jointures :

- Entre Predictions et Users
- Entre Predictions et Images

Voici le MLD déduit de la précédente modélisation

```
Users = (user_id INT, email VARCHAR(100), password VARCHAR(100), name  
VARCHAR(1000));
```

```
Images = (image_id INT, title VARCHAR(50), width INT, height INT,  
is_color LOGICAL);
```

```
Predictions = (prediction_id INT, submission_date DATE,  
prediction_result INT, #image_id, #user_id);
```

Les cardinalités se traduisent par la présence de clés étrangères user\_id et image\_id dans la table Predictions.

Enfin la traduction sous forme de MPD

```
CREATE TABLE Users(  
  user_id INT NOT NULL AUTO_INCREMENT,  
  email VARCHAR(100) NOT NULL,  
  password VARCHAR(100) NOT NULL,  
  name VARCHAR(1000) NOT NULL,  
  PRIMARY KEY(user_id),  
  UNIQUE(email)  
);  
  
CREATE TABLE Images(  
  image_id INT NOT NULL AUTO_INCREMENT,  
  title VARCHAR(50) NOT NULL,  
  width INT NOT NULL,  
  height INT NOT NULL,  
  is_color LOGICAL NOT NULL,  
  PRIMARY KEY(image_id)  
);  
  
CREATE TABLE Predictions(  
  prediction_id INT NOT NULL AUTO_INCREMENT,  
  submission_date DATE NOT NULL,  
  prediction_result INT NOT NULL,  
  image_id INT NOT NULL,  
  user_id INT NOT NULL,  
  PRIMARY KEY(prediction_id),  
  FOREIGN KEY(image_id) REFERENCES Images(image_id),  
  FOREIGN KEY(user_id) REFERENCES Users(user_id)  
);
```

## Choix de solution technique et implémentation

Les choix de la technologie de bases de données relationnelles découle de la nature du projet. Cette dernière ne comporte que trois tables. Notre finalité est de livrer un prototype. De ce fait, les tables stockeront que peu d'entrées au final. Partant de ces deux constats nous avons fait le choix d'une base de données qui peut être intégrée à même une application : SQLite.

Nous avons fait le choix de gérer la base de données via un ORM : SQLAlchemy. La solution nous permet de manipuler la base relationnelle via une abstraction sous forme de classes.

Voici comment les tables ont été implémentés via SQLAlchemy dans le fichier models.py :

```
class User(UserMixin, db.Model, Base):
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(100))
    name = db.Column(db.String(1000))
    prediction = db.relationship("Prediction")

    def __repr__(self):
        return "<User(email='%s', password='%s', name='%s')>" % (
            self.email, self.password, self.name)

class Prediction(db.Model, Base):
    __tablename__ = "prediction"
    id = db.Column(db.Integer, primary_key=True)
    submission_date = db.Column(db.String(100))
    prediction_result = db.Column(db.Integer)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"))
    image_id = db.Column(db.Integer, db.ForeignKey("image.id"))
    image = db.relationship("Image", back_populates="prediction")

    def __repr__(self):
        return "<Prediction(submission_date='%s', prediction_result='%s',
user_id='%s', image_id='%s')>" % (
            self.submission_date,
self.prediction_result, self.user_id, self.image_id)

class Image(db.Model, Base):
    __tablename__ = "image"
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100))
    width = db.Column(db.Integer)
    height = db.Column(db.Integer)
    is_color = db.Column(db.Integer)
    prediction = db.relationship("Prediction", back_populates="image")
```

```
def __repr__(self):
    return "<Image(title='%s', width='%s', height='%s', is_color='%s')>"
% (
    self.title, self.width, self.height,
    self.is_color)
```

A noter que la transposition sous forme de classe est complétée dans chacune d'elle par une méthode `__repr__` permettant la représentation des tables appelées dans les requêtes. Voici un tableau récapitulant les requêtes SQL qui seront misent en œuvre dans l'application et leur transposition via SQLAlchemy.

Requête	SQL	Transposition vis SQLAlchemy
Vérifier qu'un email est déjà présent dans la table Users	<pre>SELECT email FROM Users WHERE email LIKE email;</pre>	<pre>User.query.filter_by(email=email) .first()</pre>
Ajouter un nouvel utilisateur dans la table User	<pre>INSERT INTO User VALUES (email, password, name);</pre>	<pre>user = User(email=email, name=name, password=password)* db.session.add(user) db.session.commit()</pre>
Ajouter une image dans la table Images	<pre>INSERT INTO Images VALUES (title, width, is_color);</pre>	<pre>image = db_image( title=session['img_name'], width=session['width'] height=session['height'] is_color=session['is_color']) db.session.add(image) db.session.commit()</pre>
Ajouter une prédiction dans la table Predictions	<pre>INSERT INTO Predictions VALUES (submission_date, prediction_result, user_id, image_id);</pre>	<pre>prediction = Prediction(submission_date=date time.datetime.today().strftime( "%Y-%d-%m %H:%M:%S"), prediction_result=number_of_det ections, user_id=current_user.id, image_id= db.session.query(db_image).orde r_by(db_image.id.desc()).first( ).id ) db.session.add(prediction) db.session.commit()</pre>

Requête	SQL	Transposition vis SQLAlchemy
Récupérer toutes les prédictions réalisées par l'utilisateur connecté	<pre>SELECT title, width, height, is_color, submission_date, prediction_result FROM Users as u, Images as i, Predictions as p WHERE u.user_id = i.user_id AND i.prediction_id = p.prediction_id AND u.user_id = current_user_id ORDER BY submission_date;</pre>	<pre>user_prediction= db.session.query( Prediction, db_image ).join( db_image ).filter( Prediction.user_id==current_use r.id ).order_by(desc(Prediction.subm ission_date) ).with_entities( Prediction.submission_date, db_image.title, db_image.height, db_image.width, db_image.is_color, Prediction.prediction_result)</pre>

A noter :

- Dans le cadre de l'ajout de l'utilisateur nous avons en plus implémenter la hachage du password.
- Pour la prédiction. Nous sommes partis du postulat qu'un utilisateur réalise une prédiction dans la foulée de la mise en ligne de l'image et que deux utilisateurs ne peuvent faire une mise en ligne d'image et une demande de prédiction en même temps. Dans ce cadre, nous récupérons l'image\_id en allant chercher l'id de la dernière image soumise. Ce postulat ne tient que dans le cadre d'un prototype manipulé par un nombre restreint d'utilisateurs.

## 9. Développement du backend de l'application

Nous avons fait le choix de développer l'application à l'aide de Flask. Le framework python est léger, flexible et n'impose aucune contrainte de conception fonctionnelle, l'application n'ayant pour vocation principale que de fournir une interface et une base de données simple, ce framework est largement suffisant pour répondre au besoin.

Voici la liste des briques fonctionnelles implémentées dans le backend de l'application :

- La gestion des routes liées aux vues du front-end et à l'appel de l'API.
- L'implémentation de la base de données via l'ORM SQLAlchemy
- L'implémentation d'un système de Login via la librairie Flask Login
- L'implémentation logger (capteur de traces) pour la plateforme de monitoring et d'alerting Airbrake

Le backend contient également un fichier de définition des tests unitaires

Voici l'arbre du dossier de l'application:

```
.
├── Dockerfile
├── app.py
├── auth.py
├── db.sqlite
├── docker-compose.yml
├── main.py
├── manage.py
├── models.py
├── requirements.txt
├── start.sh
├── static
│   ├── css
│   ├── favicon.ico
│   └── welcome_page.jpg
├── templates
│   ├── about.html
│   ├── error.html
│   ├── home.html
│   ├── login.html
│   ├── predict.html
│   ├── result.html
│   ├── signup.html
│   └── stats.html
├── test_app.py
├── tests
│   ├── fake_image.png
│   ├── test_image.png
│   └── too_big.jpg
└── utils.py
```

Arborescence de l'application d'interface graphique

## Gestion des sessions utilisateurs

La librairie Flask Login permet de déterminer les algorithmes de login, de log out et de sign up. Elle permet également de déterminer les pages nécessitant un login de la part d'un utilisateur. Nous avons complété le dispositif par un password haché en sha256 implémenté grâce à la librairie werkzeug et un cookie de session de 5 minutes implémenté grâce à la module session de Flask.

Voici la liste des routes de l'application leur utilisation et la nécessité de login ou non

Route	Usage	méthode	nécessité d'être logué : oui / non
/	racine de l'application	GET	non
/about	présentation de l'application	GET	non
/login	gestion du login	GET et POST	non
/signup	gestion du sign up	GET et POST	non
/logout	gestion du log out	GET	oui
/stats	fournir les prédiction réalisée par l'utilisateur connecté	GET	oui
/home	mise en ligne de l'image pour détection de signatures	GET et POST	oui
/predict	encodage de l'image en Base64, appel de l'API, et décodage du résultat	GET et POST	oui

## 10. Design et intégration du frontend

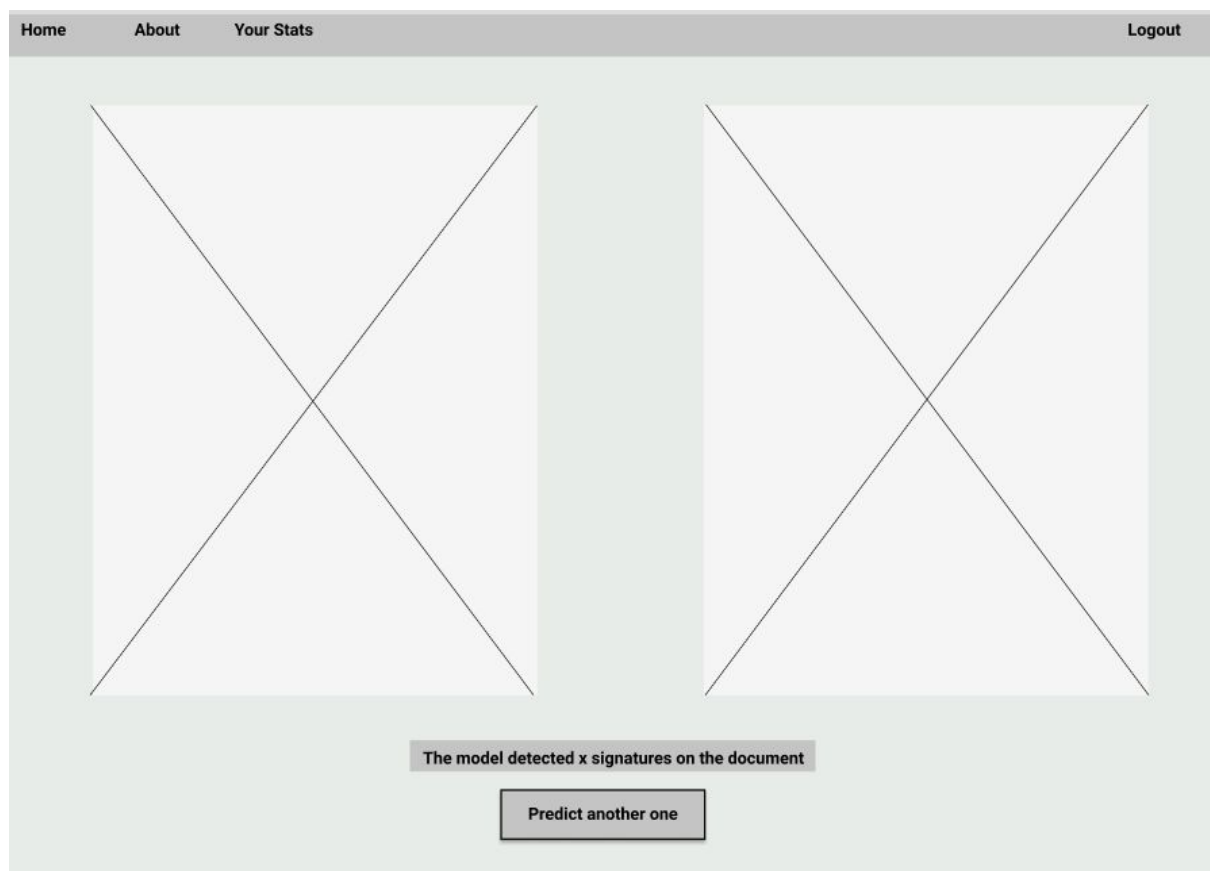
Le listing des templates des pages est assez similaire aux routes avec quelques exceptions.

Template	Usage	Contenu
/about	présentation de l'application	Contenu texte et image
/error	présentation d'un message d'erreur	texte
/login	gestion du login	formulaire de demande Email et Password
/signup	gestion du sign up	formulaire de demande d'Email, nom et password
/stats	fournir les prédiction réalisée par l'utilisateur connecté	texte / tableau html



Template	Usage	Contenu
/home	mode déconnecté : invitation au login ou sign up  mode connecté : soumission de l'image	mode déconnecté : bouton signin et sign up  mode connecté : bouton browse et submit
/predict	Après soumission de l'image : affichage de l'image mise en ligne  Après la demande de prédiction : affichage de l'image originale et de l'image avec application des bounding boxes	Après soumission image : visuel image soumise et bouton predict  Après demande de prédiction : affichage des image originale et de résultat et bouton pour demander une nouvelle prédiction

Chaque page, voir état de page, a fait l'objet d'un mockup. A titre d'exemple voici le mockup du template predict, après demande de prédiction, avec l'affichage du résultat.



L'ensemble des mockup est mis à disposition dans l'annexe VII

## 11. Définition des éléments critiques et monitoring

Nous avons déterminé les éléments critiques de l'ensemble regroupant l'interface graphique et API appliquant la matrice d'évaluation des risques.

Probabilité du risque					
Certaine	4	8	12	16	
Fort probable	3	6	9	12	
Probable	2	4	6	8	
Peu probable	1	2	3	4	
	Peu grave	Grave	Très grave	Fatal	Gravité du risque

Matrice d'évaluation des risques

Risque	Gravité	Probabilité	Score
Pas de réponse de l'API	Fatal	Probable	8
L'interface graphique ne répond pas	Fatal	Probable	8
Erreur d'enregistrement en BDD	Grave	Probable	4
Trop de requêtes concurrentes sur l'API	Très grave	Peu probable	3
Trop de requêtes concurrentes en BDD	Grave	Peu probable	2

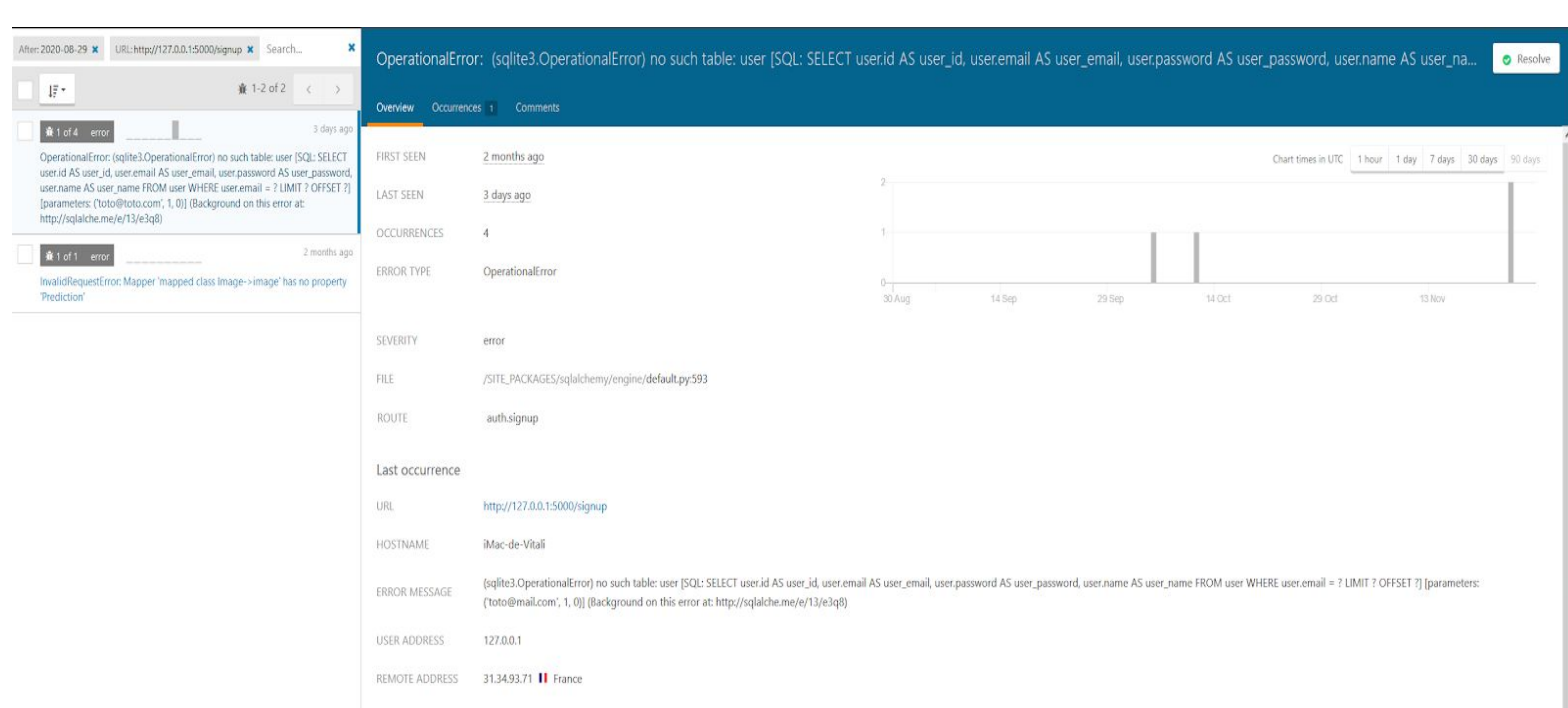
## Monitoring de l'application

Afin de suivre l'apparition de ces risques, nous avons implémenté un système de monitoring en charge de journaliser l'activité de l'application et générer des alertes le cas échéant. Pour sa simplicité d'implémentation sur une application Flask, nous avons retenu la solution Airbrake. Son intégration passe par l'installation de la librairie python pybrake de prise de trace. Il suffit ensuite de passer un identifiant et une clé de l'API Airbrake pour que les traces soient envoyées sur la plateforme du service.

Le service permet de journaliser les erreurs et les trier selon plusieurs filtres :

- route
- fichier dans l'application
- messages d'erreur
- modification du code source en utilisant une intégration avec l'hébergeur du repository
- utilisateur de l'application

Voici une erreur sur la route /signup liée à une table Users absente suite au montage des conteneurs en local



Capture d'une trace d'erreur sur la route /signup. *Source Airbrake*

Sur l'écran sont présents :

- La première / dernière occurrence de l'erreur
- Le nombre d'occurrence
- La sévérité
- Le type d'erreur
- La route sur laquelle est survenu l'erreur
- L'environnement sur lequel a été déployé
- L'ip où l'application est déployée

La plateforme permet également d'évaluer les performances de chaque route en donnant la volumétrie des codes réponses HTTP et le temps de réponse de la route. Ci-dessous une capture de la mesure de performances.



Capture de l'écran de mesure de performance des routes. Source : Airbrake

## 12. Stratégie de tests

Afin de valider le bon fonctionnement de l'application, nous avons implémenté des tests unitaires et réalisé des tests fonctionnels.

Voici le récapitulatif des tests unitaires implémentée

Test	périmètre sur lequel le test a été implémenté
L'appel des route renvoie un code HTTP 200	routes: <ul style="list-style-type: none"> <li>• /home (utilisateur non authentifié)</li> <li>• /login</li> <li>• /signup</li> <li>• /predict (après sélection de l'image)</li> </ul>
Présence dans le code HTML d'un texte attendu spécifique à la page	routes: <ul style="list-style-type: none"> <li>• /home (utilisateur non authentifié et authentifié)</li> <li>• /about</li> <li>• /signup</li> <li>• /login</li> <li>• /predict</li> </ul>
Présence dans le code HTML d'un message d'alerte attendu	route: <ul style="list-style-type: none"> <li>• /login</li> </ul>
Test de redirection effective	<ul style="list-style-type: none"> <li>• Après l'entrée d'un mail / identifiant / password redirection vers /login</li> <li>• Après l'entrée d'un mail / passwords valides sur /login vers /home authentifié</li> <li>• Après l'obtention d'un résultat sur /predict, redirection vers /home lors du clic pour demander une nouvelle prédiction</li> </ul>

Le fichier de tests unitaires est mis à disposition en annexe VIII

Les tests fonctionnels conçus sur la base des users story du backlog produit. En voici le récapitulatif:

Test fonctionnel	référence user story / périmètre fonctionnel
Les images sont bien présentes dans la BDD MongoDB après ingestion	U1 Entraînement du modèle
Les fichiers de configuration du modèle sont bien présents dans BDD MongoDB après ingestion	U1 Entraînement du modèle
Le fichier de poids est bien présent dans la base MongoDB après ingestion	U1 Entraînement du modèle

<b>Test fonctionnel</b>	<b>référence user story / périmètre fonctionnel</b>
L'entraînement du modèle arrive à son terme	U1 Entraînement du modèle
Un utilisateur renseignant un email non présent dans la BDD SQLite lors du signin se fait signifier par un message d'alerte que le mail n'existe pas	U2 Authentification
Un utilisateur renseignant un mail déjà présent en BDD SQLite lors du signup se fait signifier par une alerte que son email est déjà existant	U2 Authentification
Un utilisateur non loggué ne peut avoir accès qu'aux pages about et home. Cette dernière page affiche des boutons vers le sign up et le signin	U2 Authentification
Un utilisateur loggué ne peut avoir accès à toutes les pages. La page home permet de sélectionner et mettre en ligne une image	U2 Authentification
Après un login, sur la page home, un clic de l'utilisateur sur le bouton "browse" se traduit par l'ouverture de l'explorateur de fichiers de du système du browser	U3 Soumission d'une image
Sur la home, après un clic sur le bouton submit, l'utilisateur voit l'image qu'il a soumis dans l'application	U3 Soumission d'une image
Après appui sur le bouton predict, un utilisateur voit deux images sur la page	U5 Affichage du résultat
Après appui sur le bouton predict, un utilisateur, voit sur l'image de droite, des cadres autour des signatures détectées le cas échéant	U5 Affichage du résultat
Après appui sur le bouton predict, un utilisateur, voit en haut de page le nombre de signatures détectées	U5 Affichage du résultat
A la de la séquence de prédiction, un utilisateur demandant un nouvelles prédiction via le bouton "predict another one" se voit redirigé vers la page de soumission d'images	U5 Affichage du résultat
Un utilisateur loggué, voit la liste des prédiction effectuées, avec les information des images associées lors qu'il va sur la page "Your stats"	U6 Statistiques utilisateur

<b>Test fonctionnel</b>	<b>référence user story / périmètre fonctionnel</b>
Un administrateur a accès à un dashboard de journalisation des erreurs de l'application et des performances de cette dernière	U7 monitoring
En tant qu'utilisateur, je reçois un e-mail si l'application rencontre une erreur critique	U8 Alerting
L'interface graphique fonctionne correctement, à l'identique qu'en local sur le système d'exploitation, lorsqu'elle fonctionne dans des conteneurs docker	U9 conteneurisation
L'API fonctionne correctement, à l'identique qu'en local sur le système d'exploitation, lorsqu'elle fonctionne dans des conteneurs docker	U9 conteneurisation
L'interface communique avec l'api lorsque les deux composants sont déployés dans des conteneurs docker distincts	U9 conteneurisation
L'utilisateur a accès à l'interface graphique en utilisant un lien web de l'application hébergée dans un service cloud	U10 déploiement
L'utilisateur peut avoir la même expérience de bout-en-bout lorsque l'interface graphique et l'API sont déployées sur un service cloud	U10 déploiement

## 13. Stratégie de déploiement

### Conteneurisation

Voici la liste des conteneurs que nous avons utilisés pour l'entraînement, l'interface graphique et l'api.

Usage	Détails
Scripts d'ingestion et d'entraînement	<ul style="list-style-type: none"><li>• image : nvidia/cuda:10.2-cudnn7-devel</li><li>• Compilation lors de la construction du conteneur d'OpenCV et Darknet</li></ul>
Base de donnée analytique MongoDB	<ul style="list-style-type: none"><li>• image : mongo:4-bionic</li></ul>
Interface graphique	<ul style="list-style-type: none"><li>• image : python:3.8.5-slim-buster</li></ul>
Api	<ul style="list-style-type: none"><li>• image : tiangolo/uvicorn-gunicorn-fastapi:python3.7</li></ul>

Les Dockerfiles de chaque container et les deux docker-compose sont mis à disposition en annexe IX.

### Déploiement

Les conteneurs utilisés pour l'entraînement sont utilisés en local. En effet, les entraînements nécessitent l'utilisation de capacités en GPU et de connaître les spécificités de ce dernier afin de pouvoir paramétrer la compilation d'OpenCV et de Darknet lors de la construction du conteneur.

Nous avons dans un premier lieu déployé les conteneurs de l'interface graphique et de l'api ensemble dans une VM sur Google Cloud Platform (GCP). La solution s'est avérée peu robuste et contraire à l'esprit de l'expression de besoin du client. En effet, le document demande expressément que ces deux éléments restent modulaires.

Pour y remédier nous avons déployé l'api sur GCP via le service Cloud run. L'interface graphique est quant à elle déployée localement.

Nous sommes prêts à étudier le déploiement modulaire de l'interface graphique et de l'api sur GCP en utilisant le service Google Kubernetes Engine.



# Annexes

## Annexe I : Expression de besoin

# Expression de besoin : détection automatisée de signatures manuscrites sur des documents commerciaux

Direction DC2P : Pro PME — Orange France

Mai 2019

## Table des matières

<b>Contexte</b>	<b>3</b>
La DC2P — Direction des marchés Pro PME d'Orange France	3
Le besoin	3
Les objectifs	4
Les futurs utilisateurs	4
<b>Développement logiciel</b>	<b>4</b>
Périmètre fonctionnel	4
Contraintes techniques	5
<b>Pilotage du projet</b>	<b>5</b>
<b>Planning et livrables</b>	<b>5</b>
Livrables	5
Planning	5

# Contexte

## La DC2P — Direction des marchés Pro PME d'Orange France

La DC2P d'Orange France adresse les marchés Pro regroupant les professions libérales, autoentrepreneurs et celui des PME. Les terminaux et forfaits commercialisés sur ces segments de marché sont proches, voire similaires, par ceux proposés sur le marché grand public. Toutefois certaines particularités caractérisent ces marchés :

- Les niveaux de QoS (Qualité de service) et SLA (niveau de support) attendus sont supérieurs au marché grand public
- Les produits peuvent être commercialisés dans le réseau grand public, mais aussi dans des agences spécialisées
- Les process de ventes sont moins directs que sur le marché grand public. Le prescripteur, l'acheteur, l'utilisateur et le payeur ont une probabilité d'être distincts beaucoup plus forte.
- Les contraintes réglementaires sur ce marché sont plus élevées.

## Le besoin

Les contraintes liées aux marchés Pro PME font qu'à date, le traitement des documents commerciaux relatifs à la signature de forfaits se fait exclusivement de manière manuelle à date. Le processus mis en œuvre est à la fois consommateur d'un nombre important d'ETP (équivalent temps plein) et peut engendrer un nombre significatif d'erreurs lors du traitement.

Dans ce cadre, la validation de la présence de signatures est un prérequis à l'initiation de la validation des contrats. L'opération est positionnée en amont et sur le chemin critique des autres opérations. Toute anomalie peut dès lors être synonyme d'allongement des process voire d'incohérences dans le système d'information.

La DC2P souhaite donc mandater les équipes DEVRAP 3MU et CATOP pour étudier la possibilité d'automatiser la détection des signatures.

## Les objectifs

Les équipes DEVRAP devront étudier une solution et présenter un prototype répondant aux objectifs suivants :

- Présenter une solution technique retenue
- Développer un prototype de solution permettant de détecter les signatures manuscrites sur des documents
- La solution devra indiquer la position des signatures sur les documents, leur nombre et le cas échéant l'absence de signature.
- La solution devra pouvoir être supervisée par un opérateur ou être intégrée comme une brique de traitement dans le système d'information.

## Les futurs utilisateurs

Le prototype devra pouvoir être utilisé par trois types de publics :

- Opérateur : utilise le prototype pour détecter la présence de signatures
- Administrateur : supervise le bon fonctionnement de l'application et assure le support de niveau 1
- Mainteneur : assure l'évolution du prototype, des mises à jour fonctionnelles et techniques et du support en cas d'impossibilité d'un administrateur de l'assurer.

## Développement logiciel

### Périmètre fonctionnel

- Une interface graphique est mise à disposition des opérateurs
- L'application propose des sessions d'utilisateurs distinctes
- Les opérateurs soumettant un document scanné peuvent visualiser l'emplacement de signature, le nombre de signatures présentes et l'absence le cas échéant dans l'interface graphique
- L'application peut être interrogée sans utiliser son interface graphique
- L'application peut être appelée depuis un autre applicatif métier
- Les administrateurs ont accès à une interface pour surveiller l'état de l'application et reçoivent des alertes
- en cas de dysfonctionnement de l'application

## Contraintes techniques

- L'interface graphique doit pouvoir être interrogeable via les navigateurs web suivants : Edge, Chrome, Firefox, Safari
- L'interface répond [aux standards W3C](#)
- Le moteur est exposé sous la forme d'une API REST
- Les différents composants de l'application : interface graphique et API sont conteneurisées

## Pilotage du projet

La maîtrise d'ouvrage est confiée au management des équipes 3MU et CATOP de l'entité DEVRAP.

La maîtrise d'œuvre devra être prise en charge intégralement par les équipes 3MU et CATOP de l'entité DEVRAP.

## Planning et livrables

### Livrables






- Les conteneurs de l'API et de l'interface graphique
- Le code source de l'application
- Des comptes-rendus de suivi des travaux. Le choix de la forme et périodicité de ces derniers sont laissés à l'appréciation aux équipes 3MU et CATOP. Les équipes devront néanmoins en valider la nature en début de projet.

### Planning

Le prototype répondant à l'ensemble du périmètre fonctionnel et des exigences techniques devra être livré et présenté au 4 décembre 2020.

## Annexe II : Sprints Backlogs



SPRINT 3 BACKLOG	...	SPRINT BACKLOG 1	...	SPRINT 2 BACKLOG	...	SPRINT 4 BACKLOG	...	SPRINT 5 BACKLOG	...
U4T1 créer les routes pour l'API FastApi		U1T1 Créer du script de nettoyage des données		U2T1 Mettre en place les bases d'une application Flask		U5T1 créer une page de résultats qui affiche côte-à-côte l'image envoyée à l'API et l'image retournée par cette dernière		U9T1 Conteneuriser le script d'entraînement et la base de données	
U4T2 Créer la fonction d'appel du modèle		U1T2 Créer les collections MongoDB pour ranger les images d'entraînement		U2T2 installer et paramétrer le plugin d'authentification Flask		U5T2 créer une fonction permettant d'afficher les prédictions		U9T2 Conteneuriser l'interface graphique	
U7T1 installer le plugin de monitoring Flask		U1T3 créer les collections MongoDB pour ranger les fichier de poids et de configuration des modèles dans la base de données		U2T3 Créer l'écran d'authentification du front		U6T1 créer des tables dans la base de données pour les prédictions		U9T3 Conteneuriser l'api	
U7T2 connecter le plugin à l'application		U1T3 Créer le script d'entraînement		U2T4 Connecter le login à une base de données		U6T2 enregistrer les prédictions dans les tables correspondantes		U10T1 Paramétrer une plateforme cloud	
+ Ajouter une autre carte		U1T4 Entraîner le modèle sélectionner avec les paramètres adéquats		U3T1 Développer le front de la page de soumission des images		U6T3 créer des requêtes afin pouvoir récupérer des statistiques issues des prédictions		U10T2 Déployer le modèle	
		+ Ajouter une autre carte		U3T2 créer la table des images soumises au modèle dans la base de données		U8T1 paramétrer le système d'alertes dans le plugin de monitoring		+ Ajouter une autre carte	
				U3T3 écriture de la fonction de lecture et transformation de l'image		+ Ajouter une autre carte			
				U3T4 enregistrer les information des images soumises au modèles dans la base de données					
				+ Ajouter une autre carte					

## Annexe III : E-mails de compte-rendus des Sprints Reviews

# Compte-rendu du sprint 0

Bonjour Philippe, Samy,

Voici le compte-rendu de la review du Sprint 0.

## **Périmètre du Sprint 0 :**

- Benchmark des modèles
- Constitution de la base de données analytiques
- Découpage fonctionnel

Concernant le modèle, notre choix s'est arrêté sur Yolov4. Après analyse de l'expression de besoin il apparaît que c'est celui qui offre le meilleur équilibre performance / temps de réactivité. l'état de l'art se trouve en pièce jointe du mail.

Sur le point de la base de données analytique faite de scans fournis par la DC2P nous avons dû trouver une source pouvant correspondre au besoin. Notre choix s'est arrêté sur le dataset Tobacco-800. Celui-ci se compose de 1290 documents, dont 789 annotés. Ils sont issus de l'industrie du tabac et sont compilés par des chercheurs au laboratoire LAMP de l'université du Maryland aux Etats-Unis. Nous attirons à nouveau votre attention sur le fait que la source peut générer un biais lors de l'entraînement du modèle. Disposer de données issue du métier serait d'une aide précieuse quant à l'analyse des résultats des futurs entraînements.

Conformément à l'expression de besoin fournie par DC2P nous avons décidé découper le besoin en trois briques fonctionnelles :

- Une solution d'entraînement
- Une interface graphique servant à soumettre des images pour détection
- Une API en charge de la réalisation de la détection

Nous restons Vitali et moi-même à votre disposition pour tout complément d'information.

Bonne journée,

Cédric

# Compte-rendu du sprint 1

Bonjour Philippe, Samy,

Voici le compte-rendu du de la review du Sprint 1.

Pour rappel, nous avons fait le choix de nous inspirer en grande partie de la méthode Scrum pour mener à bien le projet. Dans ce cadre nous organisons le travail en sprint de 3 semaines, soit 15 jours de travail utile.

De ce fait ce compte-rendu et les suivants suivront le même formalisme:

- Rappel du périmètre du sprint
- Listing du découpage en tâche des user stories traitées avec score attribué à chacune d'elle par l'équipe. Ce score, entre 1 et 12 reflète la difficulté et le temps perçu pour accomplir la tâche.
- Liste des tests fonctionnels passés / réussis pendant le sprint
- Burndown chart du sprint
- Remarques

## **Périmètre fonctionnel du Sprint 1:**

- Entraînement du modèle, équivaut la l'User Story U1 → En tant datascientist je dois pouvoir (ré)entraîner facilement le modèle afin d'optimiser ses performances

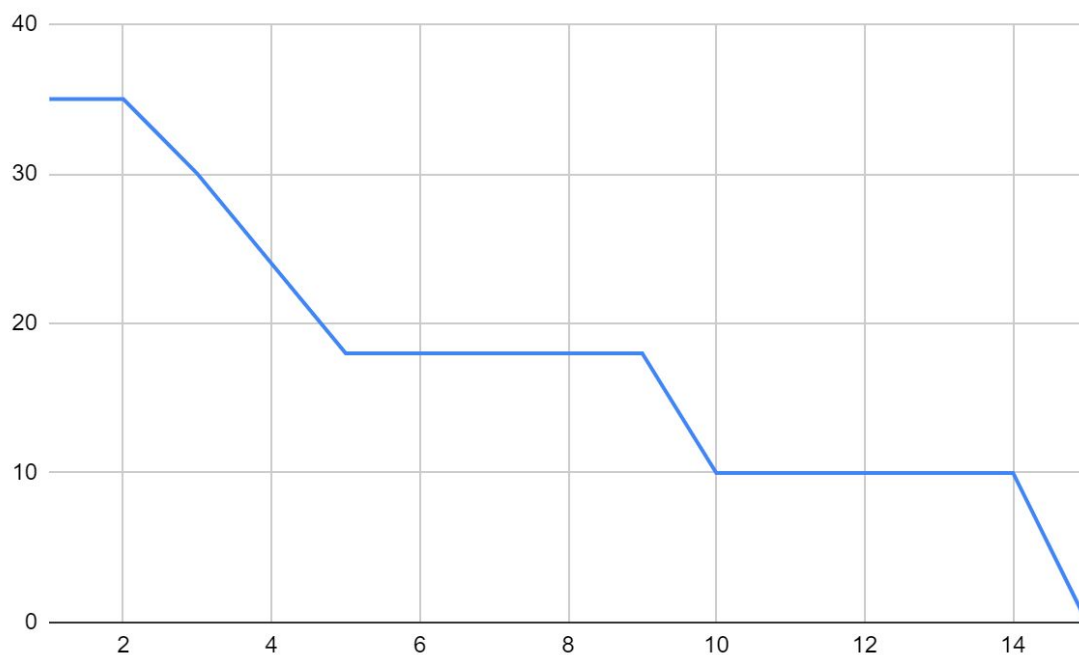
## **Découpage en tâches:**

Tâche	Score
U1T1 Créer du script de nettoyage des données	5
U1T2 Créer les collections MongoDB pour ranger les images d'entraînement	6
U1T3 créer les collections MongoDB pour ranger les fichier de poids et de configuration des modèles dans la base de données	6
U1T3 Créer le script d'entraînement	8
U1T4 Entraîner le modèle sélectionné avec les paramètres adéquats	10

### Test fonctionnel mis en oeuvre:

Test	Statut
Les images sont bien présentes dans la BDD MongoDB après ingestion	OK
Les fichiers de configuration du modèle sont bien présents dans BDD MongoDB après ingestion	OK
Le fichier de poids est bien présent dans la base MongoDB après ingestion	OK
L'entraînement du modèle arrive à son terme	OK

### Burndown chart :



### Remarques :

Le premier entraînement sur le dataset Tobacco-800 seul n'a pas donné des performances acceptables pour considérer implémenter le modèle entraîné. Notamment à cause de l'absence de documents scannés en couleur. Pour y remédier nous avons procédé par itérations en ajoutant des images issu de deux nouvelles sources :

- [Nanonets](#) : Dataset de 174 documents issus des moteurs de recherche Google et Bing mis à disposition par Nanonets sur son compte [Github](#).

- [GSA Lease documents](#) : Ensemble des baux commerciaux contractés avec l'administration des services généraux du gouvernement américain afin de fournir des bureaux aux agents fédéraux.

Voici le tableau récapitulatif des résultats obtenus

	<b>mAP</b>	<b>Precision</b>	<b>Recall</b>	<b>F1 Score</b>
Premier entraînement	29.57%	44%	25%	32%
Deuxième entraînement	65.06%	52%	75%	62%
Troisième entraînement	68.20%	71%	69%	70%

Nous n'avons pas poursuivi au-delà du troisième entraînement dû à un début de baisse du Recall. Il pourrait être judicieux en fin de projet de réaliser en entraînement de réaliser des données métiers si la DC2P peut en fournir d'ici là.

Bonne fin de journée,

Cédric

## Compte-rendu du sprint 2

Bonjour Philippe, Samy

Vous trouverez ci-dessous le compte-rendu du sprint 2.

### Périmètre fonctionnel du Sprint 2 :

- Implémentation de l'ORM SQLAlchemy sur une base d'application Flask
- Implémentation de la gestion de la session utilisateur. Équivaut à l'user story U2 → En tant qu'utilisateur j'ai dois pouvoir m'authentifier afin d'utiliser l'application de manière sécurisée
- Gestion de la mise en ligne de l'image à prédire. Équivaut à l'user story U3 → En tant qu'utilisateur je dois pouvoir soumettre une image afin de récupérer le nombre de signatures dessus

### Découpage en tâches :

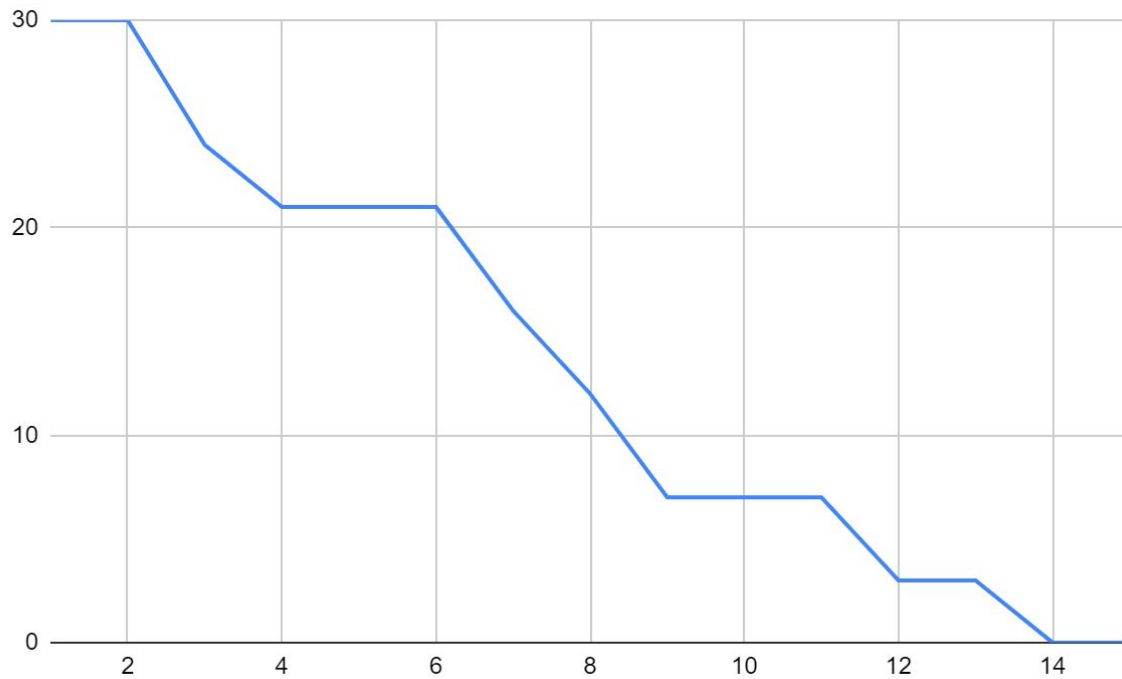
Tâche	Score
U2T1 Mettre en place les bases d'une application Flask	1
U2T2 installer et paramétrer le plugin d'authentification Flask	6
U2T3 Créer l'écran d'authentification du front	3
U2T4 Connecter le login à une base de données	5
U3T1 Développer le front de la page de soumission des images	4
U3T2 créer la table des images soumises au modèle dans la base de données	5
U3T3 écriture de la fonction de lecture et transformation de l'image	4
U3T4 enregistrer les information des images soumises au modèles dans la base de données	3

**Tests fonctionnels :**

Test	Statut
Un utilisateur renseignant un email non présent dans la BDD SQLite lors du signin se fait signifier par un message d'alerte que le mail n'existe pas	OK
Un utilisateur renseignant un mail déjà présent en BDD SQLite lors du signup se fait signifier par une alerte que son email est déjà existant	OK
Un utilisateur renseignant de nouveaux identifiant / mail / password non présents dans la BDD SQLite lors du sign up se voit redirigé vers la page de login	OK
Un utilisateur renseignant un email existant dans la BDD SQLite lors du signin est redirigé vers la page de soumission d'une image	OK
Après 5 minutes d'inactivité, un utilisateur est déconnecté de l'application et doit se reconnecter	OK
Un utilisateur non loggué ne peut avoir accès qu'aux pages about et home. Cette dernière page affiche des boutons vers le sign up et le signin	OK
Un utilisateur loggué ne peut avoir accès à toutes les pages. La page home permet de sélectionner et mettre en ligne une image	OK
Après un login, sur la page home, un clic de l'utilisateur sur le bouton "browse" se traduit par l'ouverture de l'explorateur de fichiers de du système du browser	OK
Sur la home, après un clic sur le bouton submit, l'utilisateur voit l'image qu'il a soumis dans l'application	OK



### Burndown chart :



### Remarques:

- Nous avons fait le choix d'une base de données SQLite
- Nous avons fait le choix de SQLAlchemy comme solution OR

Bonne fin de journée,

Cédric

## Compte-rendu du sprint 3

Bonjour Philippe, Samy

Vous trouverez ci-dessous le compte-rendu de la review du sprint 3.

### Périmètre fonctionnel:

- Développement de l'API. Équivaut à l'user story U4 → En tant qu'utilisateur je dois pouvoir récupérer la prédiction du modèle afin de pouvoir l'exploiter dans l'interface graphique
- Implémentation du monitoring. Équivaut à l'user story U7 → En tant qu'administrateur je dois pouvoir visualiser l'état de fonctionnement de l'application afin de m'assurer de son bon fonctionnement

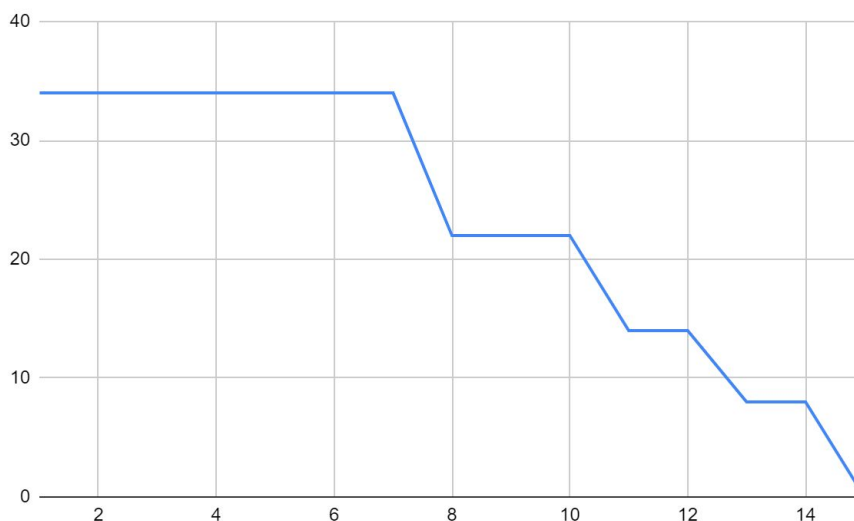
### Découpage en tâches:

Tâche	Score
U4T1 créer les routes pour l'API FastApi	12
U4T2 Créer la fonction d'appel du modèle	8
U7T1 installer le plugin de monitoring Flask	6
U7T2 connecter le plugin à l'application	8

### Tests fonctionnels:

Nous n'avons pas prévu de test fonctionnels de l'API. L'intégration de ses résultat en base de données de l'application interface graphique sera testé lors du sprint 4

### Burndown chart:



## Compte-rendu du sprint 4

Bonjour Philippe, Samy,

Vous trouverez ci-dessous le compte-rendu de la review du sprint 4.

### Périmètre fonctionnel:

- Intégration du résultat de l'API dans l'application. Équivaut à deux users stories :
  - U5 En tant qu'utilisateur je dois pouvoir visualiser l'image et les résultats de la prédiction afin de constater les résultats des prédictions après avoir interrogé le modèle
  - U6 En tant qu'utilisateur je dois pouvoir enregistrer les informations des images passées dans le modèle ainsi que les prédictions associées afin de pouvoir y avoir accès ultérieurement
- Paramétrage de l'alerting. Équivaut à l'user story U8 → En tant qu'administrateur je dois pouvoir être alerté en cas de dysfonctionnement de l'application afin d'y remédier

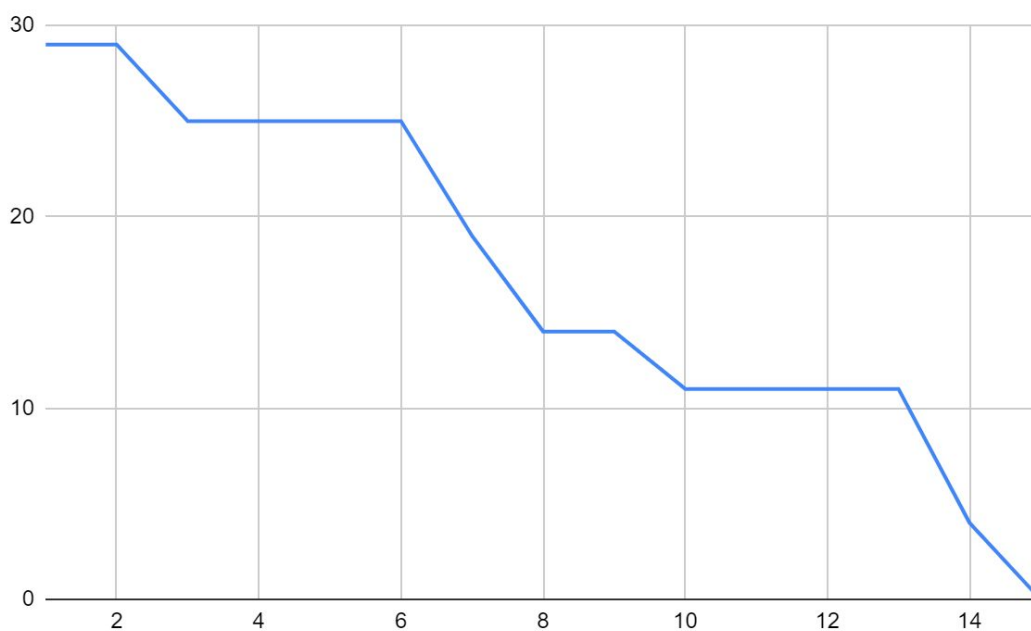
### Découpage en tâches:

Tâche	Score
U5T1 créer une page de résultats qui affiche côte-à-côte l'image envoyée à l'API et l'image retournée par cette dernière	4
U5T2 créer une fonction permettant d'afficher les prédictions	6
U6T1 créer des tables dans la base de données pour les prédictions	5
U6T2 enregistrer les prédictions dans les tables correspondantes	3
U6T3 créer des requêtes afin pouvoir récupérer des statistiques issues des prédictions	7
U8T1 paramétrer le système d'alertes dans le plugin de monitoring	4

### Tests fonctionnels:

Test	Statut
Après appui sur le bouton predict, un utilisateur voit deux images sur la page	OK
Après appui sur le bouton predict, un utilisateur, voit sur l'image de droite, des cadres autour des signatures détectées le cas échéant	OK
Après appui sur le bouton predict, un utilisateur, voit en haut de page le nombre de signatures détectées	OK
A la de la séquence de prédiction, un utilisateur demandant un nouvelles prédiction via le bouton "predict another one" se voit redirigé vers la page de soumission d'images	OK
Un utilisateur logué, voit la liste des prédiction effectuées, avec les information des images associées lors qu'il va sur la page "Your stats"	OK

### Burndown chart:



Bonne fin de journée,

Cédric

## Compte-rendu du sprint 5

Bonjour Philippe, Samy,

Vous trouverez ci-dessous le compte-rendu de la review du sprint 5.

### Périmètre fonctionnel:

- Conteneurisation. Équivalent de l'user story U9 → U9 En tant datascientist je dois pouvoir utiliser des container afin de déployer facilement l'application
- Mise en production. Equivalant de l'user story U10 → U10 En datascientiste je dois pouvoir déployer l'application afin de la rendre disponible aux utilisateurs

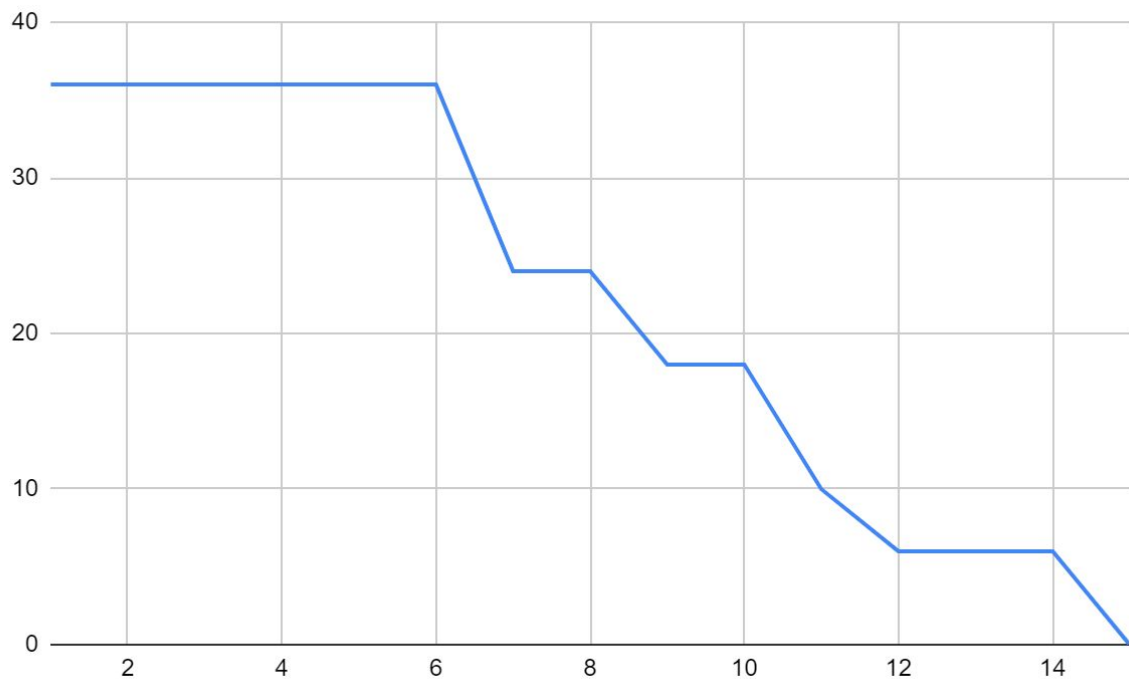
### Découpage en tâches:

Tâche	Score
U9T1 Conteneuriser le script d'entraînement et la base de données	12
U9T2 Conteneuriser l'interface graphique	6
U9T3 Conteneuriser l'api	8
U10T1 Paramétrer une plateforme cloud	4
U10T2 Déployer l'application et l'api	6

### Tests fonctionnels:

Test	Statut
L'interface graphique fonctionne correctement, à l'identique qu'en local sur le système d'exploitation, lorsqu'elle fonctionne dans des conteneurs docker	OK
L'API fonctionne correctement, à l'identique qu'en local sur le système d'exploitation, lorsqu'elle fonctionne dans des conteneurs docker	OK
L'interface communique avec l'api lorsque les deux composants sont déployés dans des conteneurs docker distincts	OK
L'utilisateur a accès à l'interface graphique en utilisant un lien web de l'application hébergée dans un service cloud	KO
L'utilisateur peut avoir la même expérience de bout-en-bout lorsque l'interface graphique et l'API sont déployées sur un service cloud	OK

### Burndown chart :



### Remarques :

Après plusieurs tests de mise en production, nous avons adopté la solution Google Cloud Run. Cette dernière nous a permis de mettre en ligne l'api sur Google Cloud Platform. L'interface graphique est à utiliser localement. Nous envisageons l'étude un déploiement total des deux via Google Kubernetes Engine pour une itération de projet ultérieure.

## Annexe IV : Notebook de visualisations sur le base de donnée analytique

```
In [1]: import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
sns.set_theme(style="white")
import math
import numpy as np

df = pd.read_csv('https://drive.google.com/uc?id=1-3TtbNmGJZeti05pX854G3RfI')
```

```
In [2]: df.head()
```

```
Out[2]:
```

	Unnamed: 0	name	height	width	is_color	number_of_signatures	source
0	0	b014c226	3300	2544	0	1	tobacco_data
1	1	bee1cedc	1575	1200	0	1	tobacco_data
2	2	6e367c98	1575	1200	0	1	tobacco_data
3	3	21f534e3	3300	2544	0	1	tobacco_data
4	4	cdfd883c	3150	2400	0	1	tobacco_data

```
In [4]: df.describe()
```

```
Out[4]:
```

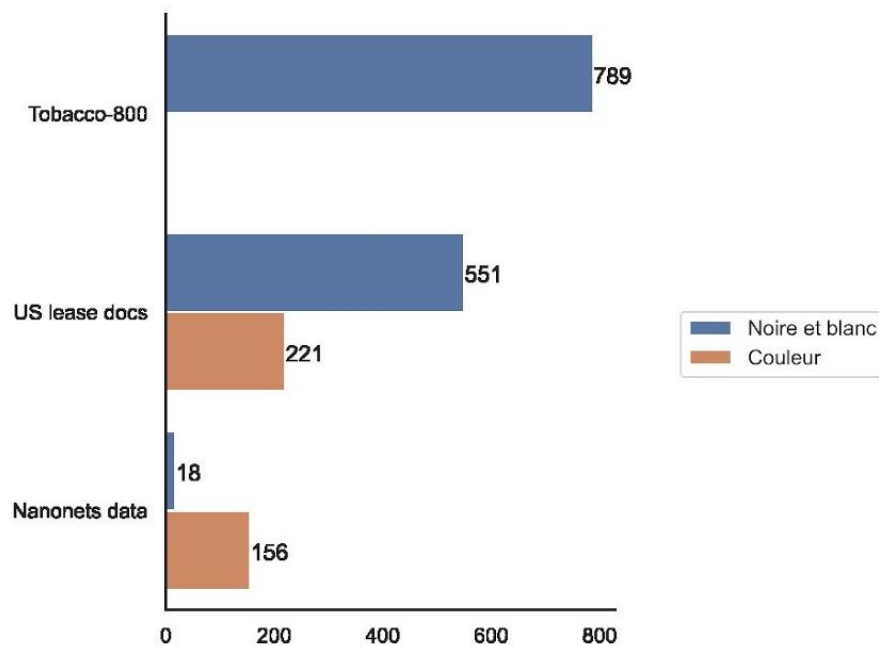
	Unnamed: 0	height	width	is_color	number_of_signatures
count	1735.000000	1735.000000	1735.000000	1735.000000	1735.000000
mean	867.000000	2243.940634	1734.374640	0.217291	2.004035
std	500.995675	650.636099	507.187124	0.412521	1.215284
min	0.000000	408.000000	380.000000	0.000000	1.000000
25%	433.500000	2183.000000	1696.000000	0.000000	1.000000
50%	867.000000	2201.000000	1708.000000	0.000000	2.000000
75%	1300.500000	2292.000000	1728.000000	0.000000	3.000000
max	1734.000000	4200.000000	3506.000000	1.000000	8.000000

Nombre d'images en couleur et noire et blanc en fonction d'une source de données



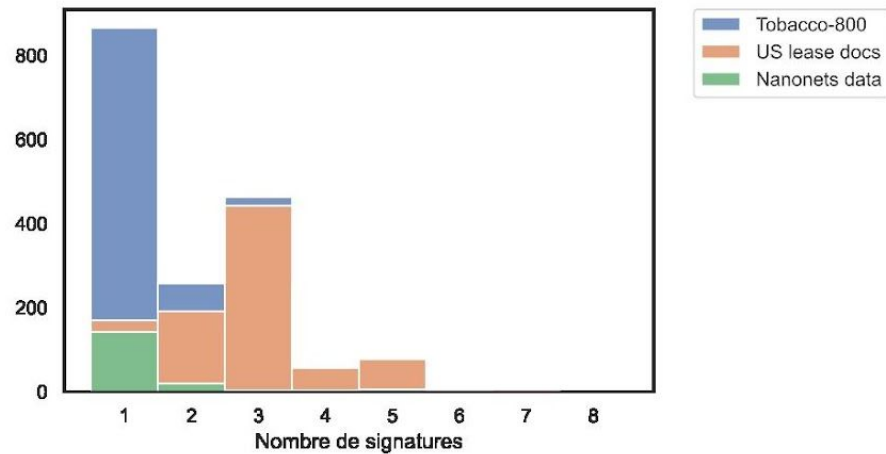
```
In [6]: plot_data = df.groupby('source')['is_color'].value_counts()
plot_data = plot_data.rename('count').reset_index()

g=sns.catplot(x='count', y='source', kind='bar', hue='is_color',
              order=['tobacco_data', 'lease_docs_png', 'nanonets_dataset'], c
g.set_yticklabels(['Tobacco-800', 'US lease docs', 'Nanonets data'])
g.set(xlabel="", ylabel = "")
handles, labels = g.ax.get_legend_handles_labels()
g.ax.legend(handles=handles, bbox_to_anchor=(1.6, 0.5), borderaxespad=0, label
for p in g.ax.patches:
    width=p.get_width()
    if math.isnan(width):
        continue
    else:
        txt = str(int(width))
        txt_x = p.get_width()
        txt_y = p.get_y() + 0.25
        g.ax.text(txt_x, txt_y, txt)
```



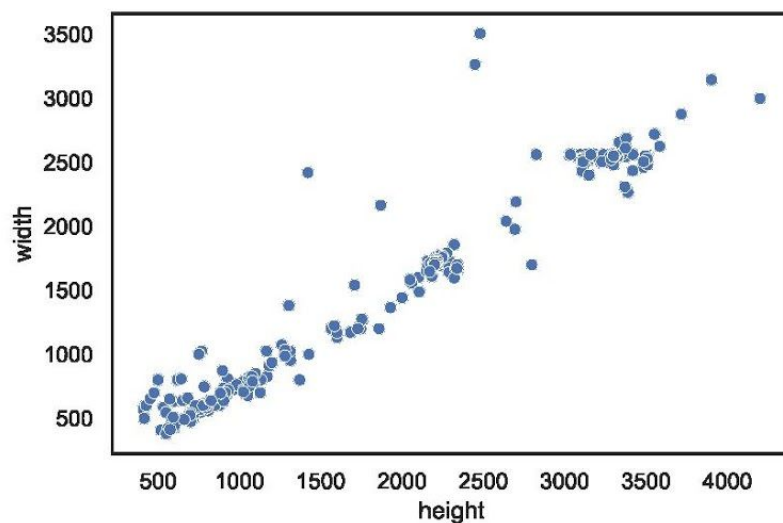
Le nombre de signatures sur les documents en fonction d'une source de données

```
In [7]: g = sns.histplot(data=df, x='number_of_signatures', hue='source', discrete=True)
g.set(xlabel='Nombre de signatures', ylabel = '')
leg = g.axes.get_legend()
leg.borderaxespad=0
leg.set_bbox_to_anchor((1.4, 1))
leg.set_title('')
new_labels = ['Tobacco-800', 'US lease docs', 'Nanonets data']
for t, l in zip(leg.texts, new_labels): t.set_text(l)
```



## Les dimensions des documents

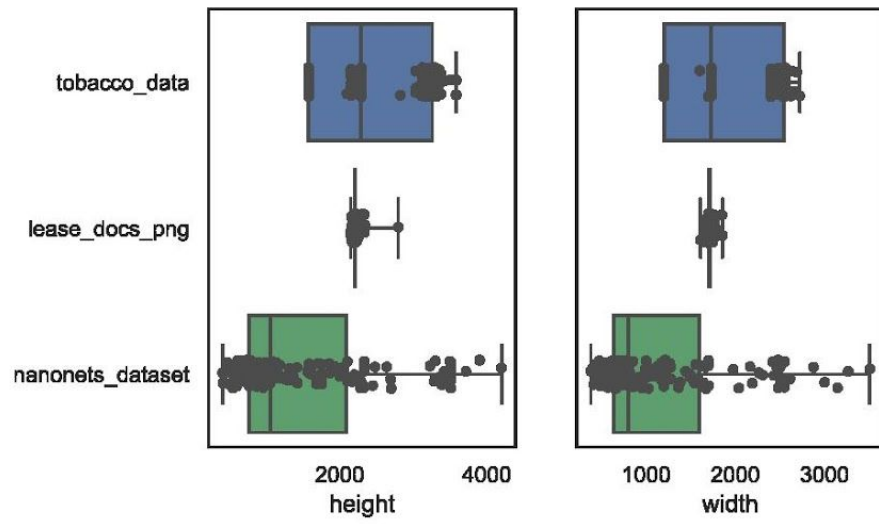
```
In [8]: g = sns.scatterplot(data=df, x='height', y='width')
```



```
In [9]: fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=True)

sns.boxplot(x="height", y="source", data=df, whis=np.inf, ax=ax1)
g1=sns.stripplot(x="height", y="source", data=df, color=".3", ax=ax1)
g1.set(ylabel='')

sns.boxplot(x="width", y="source", data=df, whis=np.inf, ax=ax2)
g2=sns.stripplot(x="width", y="source", data=df, color=".3", ax=ax2)
g2.set(ylabel='')
plt.show()
```



In [ ]:

## Annexe V : Script d'ingestion des données et des fichiers de configuration et de poids

```

from pymongo import MongoClient, errors
import datetime
import re

# global variables for MongoDB host (default port is 27017)
DOMAIN = 'mongodb://mongodb'
PORT = 27017

# use a try-except indentation to catch MongoClient() errors
try:
    # try to instantiate a client instance
    client = MongoClient(
        host = [ str(DOMAIN) + ":" + str(PORT) ],
        serverSelectionTimeoutMS = 3000, # 3 second timeout
        username = 'testuser',
        password = 'testpass',
    )

    # print the version of MongoDB server if connection successful
    print ('server version:', client.server_info()['version'])

    # get the database_names from the MongoClient()
    database_names = client.list_database_names()

except errors.ServerSelectionTimeoutError as err:
    # set the client and DB name list to 'None' and `[]` if exception
    client = None
    database_names = []

    # catch pymongo.errors.ServerSelectionTimeoutError
    print ('pymongo ERROR:', err)

db = client['db']
images = db['images']
config = db['config']
fs = gridfs.GridFS(db, 'weights')

# downloading images + annotations
get_ipython().system('unzip -qq Tobacco_dataset_png.zip; rm
Tobacco_dataset_png.zip')
get_ipython().system('unzip -qq anotations_darknet.zip; rm
anotations_darknet.zip')

# %%
import os

```

```

# some images doesn't have annotations
# we will ignore them
tobacco=os.listdir('Tobacco_dataset_png')
annot=os.listdir('anotations_darknet')

# %%
import random
# split train 60% valid 30% test 10%
random.shuffle(annot)

# getting list of filenames without extention
filenames = [os.path.splitext(filename)[0] for filename in annot]

# wringing names to dict
dataset = dict()
dataset['train'] = filenames[:int(round(len(filenames)*0.6,0))]
dataset['valid'] =
filenames[int(round(len(filenames)*0.6,0)):int(round(len(filenames)*0.9,0))]
dataset['test'] = filenames[int(round(len(filenames)*0.9,0)):]

# %%
#yolo_db = client['yolo_db']
images = yolo_db['input_col']

# %%

import base64
import cv2

width = 608
height = 608

data = []

# get path of the current working directory
cwd = os.getcwd()

for setname in dataset.keys():
    # create directory
    if not os.path.isdir(setname):
        os.mkdir(setname)

    # coping images and annotations
    for filename in dataset[setname]:

```

```

source_image = cwd + '/Tobacco_dataset_png/' + filename + '.png'
source_label = cwd + '/anotations_darknet/' + filename + '.txt'
destination_raw_image = cwd + '/' + setname + '/' + filename + '.png'
destination_resized_image = cwd + '/' + setname + '/resized_' + filename
+ '.png'
destination_label = cwd + '/' + setname + '/' + filename + '.txt'

# read and resize
img = cv2.imread(source_image, cv2.IMREAD_UNCHANGED)
resized = cv2.resize(img, (width, height))
cv2.imwrite(destination_resized_image, resized)

# copy of the annotation
os.rename(source_label, destination_label)
os.rename(source_image, destination_raw_image)

with open(destination_raw_image, 'rb') as binary_raw_image:
    binary_raw_image_data = binary_raw_image.read()
    base64_encoded_raw_image_data =
base64.b64encode(binary_raw_image_data)
    base64_raw_image_message =
base64_encoded_raw_image_data.decode('utf-8')

    with open(destination_resized_image, 'rb') as binary_resized_image:
        binary_resized_image_data = binary_resized_image.read()
        base64_encoded_resized_image_data =
base64.b64encode(binary_resized_image_data)
        base64_resized_image_message =
base64_encoded_resized_image_data.decode('utf-8')

    with open(destination_label, 'rb') as binary_label_file:
        binary_label_data = binary_label_file.read()
        base64_encoded_label_data = base64.b64encode(binary_label_data)
        base64_label_message = base64_encoded_label_data.decode('utf-8')

    data.append({
        'raw_image':base64_raw_image_message,
        'resized_image':base64_resized_image_message,
        'label':base64_label_message,
        'dataset':setname
    })

db['images'].insert_many(data)

# %%
#we build config dynamically based on number of classes

```

```

#we build iteratively from base config files. This is the same file shape as
cfg/yolo-obj.cfg

num_classes = 1
print("writing config for a custom YOLOv4 detector detecting number of
classes: " + str(num_classes))

#Instructions from the darknet repo
#change line max_batches to (classes*2000 but not less than number of
training images, and not less than 6000), f.e. max_batches=6000 if you train
for 3 classes
#change line steps to 80% and 90% of max_batches, f.e. steps=4800,5400
if os.path.exists('./cfg/custom-yolov4-detector.cfg'):
os.remove('./cfg/custom-yolov4-detector.cfg')

directory = "./cfg/"
filenames = []
binaries = []
data = []

if os.path.exists('./cfg/custom-yolov4-detector.cfg'):
os.remove('./cfg/custom-yolov4-detector.cfg')

with open('./cfg/custom-yolov4-detector.cfg', 'a') as f:
    f.write('[net]' + '\n')
    f.write('batch=64' + '\n')
    #####smaller subdivisions help the GPU run faster. 12 is optimal, but
you might need to change to 24,36,64####
    f.write('subdivisions=24' + '\n')
    f.write('width=608' + '\n')
    f.write('height=608' + '\n')
    f.write('channels=3' + '\n')
    f.write('momentum=0.949' + '\n')
    f.write('decay=0.0005' + '\n')
    f.write('angle=0' + '\n')
    f.write('saturation = 1.5' + '\n')
    f.write('exposure = 1.5' + '\n')
    f.write('hue = .1' + '\n')
    f.write('\n')
    f.write('learning_rate=0.001' + '\n')
    f.write('burn_in=1000' + '\n')
    #####you can adjust up and down to change training time####
    ##Darknet does iterations with batches, not epochs####
    #max_batches = num_classes*2000
    max_batches = 6000
    f.write('max_batches=' + str(max_batches) + '\n')
    f.write('policy=steps' + '\n')
    steps1 = .8 * max_batches

```



```

steps2 = .9 * max_batches
f.write('steps='+str(steps1)+','+str(steps2) + '\n')

#Instructions from the darknet repo
#change line classes=80 to your number of objects in each of 3
[yolo]-layers:
#change [filters=255] to filters=(classes + 5)x3 in the 3 [convolutional]
before each [yolo] layer, keep in mind that it only has to be the last
[convolutional] before each of the [yolo] layers.

with open('cfg/yolov4-custom2.cfg', 'r') as f2:
    content = f2.readlines()
    for line in content:
        f.write(line)
    num_filters = (num_classes + 5) * 3
    f.write('filters='+str(num_filters) + '\n')
    f.write('activation=linear')
    f.write('\n')
    f.write('\n')
    f.write('[yolo]' + '\n')
    f.write('mask = 0,1,2' + '\n')
    f.write('anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146,
142, 110, 192, 243, 459, 401' + '\n')
    f.write('classes=' + str(num_classes) + '\n')

with open('cfg/yolov4-custom2.cfg', 'rb') as binary_yolov4_custom2:
    binary_yolov4_custom2_data = binary_yolov4_custom2.read()
    base64_binary_yolov4_custom2_data =
base64.b64encode(binary_yolov4_custom2_data)
    base64_binary_yolov4_custom2_message =
base64_binary_yolov4_custom2_data.decode('utf-8')

binaries.append(base64_binary_yolov4_custom2_message)

with open('cfg/yolov4-custom3.cfg', 'r') as f3:
    content = f3.readlines()
    for line in content:
        f.write(line)
    num_filters = (num_classes + 5) * 3
    f.write('filters='+str(num_filters) + '\n')
    f.write('activation=linear')
    f.write('\n')
    f.write('\n')
    f.write('[yolo]' + '\n')
    f.write('mask = 3,4,5' + '\n')
    f.write('anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146,
142, 110, 192, 243, 459, 401' + '\n')
    f.write('classes=' + str(num_classes) + '\n')

```

```

with open('cfg/yolov4-custom3.cfg', 'rb') as binary_yolov4_custom3:
    binary_yolov4_custom3_data = binary_yolov4_custom3.read()
    base64_binary_yolov4_custom3_data =
base64.b64encode(binary_yolov4_custom3_data)
    base64_binary_yolov4_custom3_message =
base64_binary_yolov4_custom3_data.decode('utf-8')

binaries.append(base64_binary_yolov4_custom3_message)

with open('cfg/yolov4-custom4.cfg', 'r') as f4:
    content = f4.readlines()
    for line in content:
        f.write(line)
    num_filters = (num_classes + 5) * 3
    f.write('filters='+str(num_filters) + '\n')
    f.write('activation=linear')
    f.write('\n')
    f.write('\n')
    f.write('[yolo]' + '\n')
    f.write('mask = 6,7,8' + '\n')
    f.write('anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146,
142, 110, 192, 243, 459, 401' + '\n')
    f.write('classes=' + str(num_classes) + '\n')

with open('cfg/yolov4-custom4.cfg', 'rb') as binary_yolov4_custom4:
    binary_yolov4_custom4_data = binary_yolov4_custom4.read()
    base64_binary_yolov4_custom4_data =
base64.b64encode(binary_yolov4_custom4_data)
    base64_binary_yolov4_custom4_message =
base64_binary_yolov4_custom4_data.decode('utf-8')

binaries.append(base64_binary_yolov4_custom5_message)

with open('cfg/yolov4-custom5.cfg', 'r') as f5:
    content = f5.readlines()
    for line in content:
        f.write(line)

with open('cfg/yolov4-custom5.cfg', 'rb') as binary_yolov4_custom5:
    binary_yolov4_custom5_data = binary_yolov4_custom5.read()
    base64_binary_yolov4_custom5_data =
base64.b64encode(binary_yolov4_custom5_data)
    base64_binary_yolov4_custom5_message =
base64_binary_yolov4_custom5_data.decode('utf-8')

binaries.append(base64_binary_yolov4_custom5_message)

```

```

with open('./cfg/custom-yolov4-detector.cfg', 'rb') as
binary_custom_yolov4_detector:
    binary_custom_yolov4_detector_data =
binary_custom_yolov4_detector.read()
    base64_encoded_custom_yolov4_detector_data =
base64.b64encode(binary_custom_yolov4_detector_data)
    base64_encoded_custom_yolov4_detector_message =
base64_encoded_custom_yolov4_detector_data.decode('utf-8')

binaries.append( base64_encoded_custom_yolov4_detector_message)

for file in os.listdir(directory):
    name_pattern = re.compile(r"^.*\/(.*)\..*$")
    file_name = re.findall(name_pattern, string=file)[0]
    file_names.append(file_name)

for i in range(1,6):

    entry = {
        'date': datetime.datetime.now(),
        'file_name': entries[i],
        'config_file': binaries[i],
    }

    data.append(entry)

db['config'].insert_many(data)

with open('./darknet/yolov4.conv.137', 'rb') as weights :
    fs.put(data=weights, filename = 'yolov4.conv.137')

```

## Annexe VI : Script d'entraînement du modèle

```

from pymongo import MongoClient, errors
import datetime
import shutil
import os

# global variables for MongoDB host (default port is 27017)
DOMAIN = 'mongodb://mongodb'
PORT = 27017

# use a try-except indentation to catch MongoClient() errors
try:
    # try to instantiate a client instance
    client = MongoClient(
        host = [ str(DOMAIN) + ":" + str(PORT) ],
        serverSelectionTimeoutMS = 3000, # 3 second timeout
        username = 'testuser',
        password = 'testpass',
    )

    # print the version of MongoDB server if connection successful
    print ('server version:', client.server_info()['version'])

    # get the database_names from the MongoClient()
    database_names = client.list_database_names()

except errors.ServerSelectionTimeoutError as err:
    # set the client and DB name list to 'None' and `[]` if exception
    client = None
    database_names = []

    # catch pymongo.errors.ServerSelectionTimeoutError
    print ('pymongo ERROR:', err)

db = client['db']
images = db['images']
config = db['config']
fs = gridfs.GridFS(db, 'weights')

!cd /app/darknet/

cw = os.getcwd()

```

```

dirs = ['train', 'valid', 'test']

for directory in dirs :

    if not os.path.isdir(cw+'/data/obj/'+directory):
        os.mkdir(cw+'/data/obj/'+directory)

images = images.find(
    {},
    {'_id':0, 'resized_image':1, 'label':1,
'dataset':1})

#retrieve images
images_len = images.count_documents({})

for i in range(0, images_len):

    base64_img_bytes = images[i]['resized_image'].encode('utf-8')
    source_image= images[i]['dataset'] + f'{i}' + '.png'
    destination_image = '/obj/data/' + images[i]['dataset']
    with open(source_image, 'wb') as image_to_save:
        decoded_image_data = base64.decodebytes(base64_img_bytes)
        image_to_save.write(decoded_image_data)
    shutil.move('/app/darknet/' + source_image,
'/app/darknet/data/obj/' + images[i]['dataset'])

    base64_txt_bytes = images[i]['label'].encode('utf-8')
    source_label= images[i]['dataset'] + f'{i}' + '.txt'
    with open(source_label,'wb') as txt_to_save:
        decoded_txt_data = base64.decodebytes(base64_txt_bytes)
        txt_to_save.write(decoded_txt_data)
    shutil.move('/app/darknet/' + source_label,
'/app/darknet/data/obj/' + images[i]['dataset'] )

with open('data/obj.data', 'w') as out:
    out.write('classes = 1\n')
    out.write('train = data/train.txt\n')
    out.write('valid = data/valid.txt\n')
    out.write('names = data/obj.names\n')
    out.write('backup = backup/')

#write train file (just the image list)
with open('data/train.txt', 'w') as out:
    for img in [f for f in os.listdir('./data/obj/train') if
f.endswith('.png')]:

```

```

        out.write('data/obj/' + img + '\n')

#write the valid file (just the image list)
with open('data/valid.txt', 'w') as out:
    for img in [f for f in os.listdir('./data/obj/valid') if
f.endswith('png')]:
        out.write('data/obj/' + img + '\n')

configs = config.find(
    {},
    {'_id':0, 'file_name':1, 'config_file':1}
)
configs_len = config.count_documents({})

for i in range(0, configs_len):
    file_name = configs[i]["file_name"] + ".cfg"
    base64_file_bytes = configs[i]["configuration_file"].encode("utf-8")
    with open('./darknet/cfg/' + file_name, "wb") as file_to_extract:
        decoded_file_data = base64.decodebytes(base64_file_bytes)
        file_to_extract.write(decoded_file_data)
weights = db.weights.files.find({'filename':file_name})
uid = weights[0]['_id']
file_name = req[0]['filename'] + '.conv.137'
data = FS.get(uid).read()
with open(file_name, 'wb') as file_to_extract:
    file_to_extract.write(data)

#retrieve weights
reg = db.weights.files.find({'filename':file_name})
uid = req[0]['_id']
file_name = req[0]['filename'] + '.weights'
data = FS.get(uid).read()
with open(file_name, 'wb') as file_to_extract:
    file_to_extract.write(data)

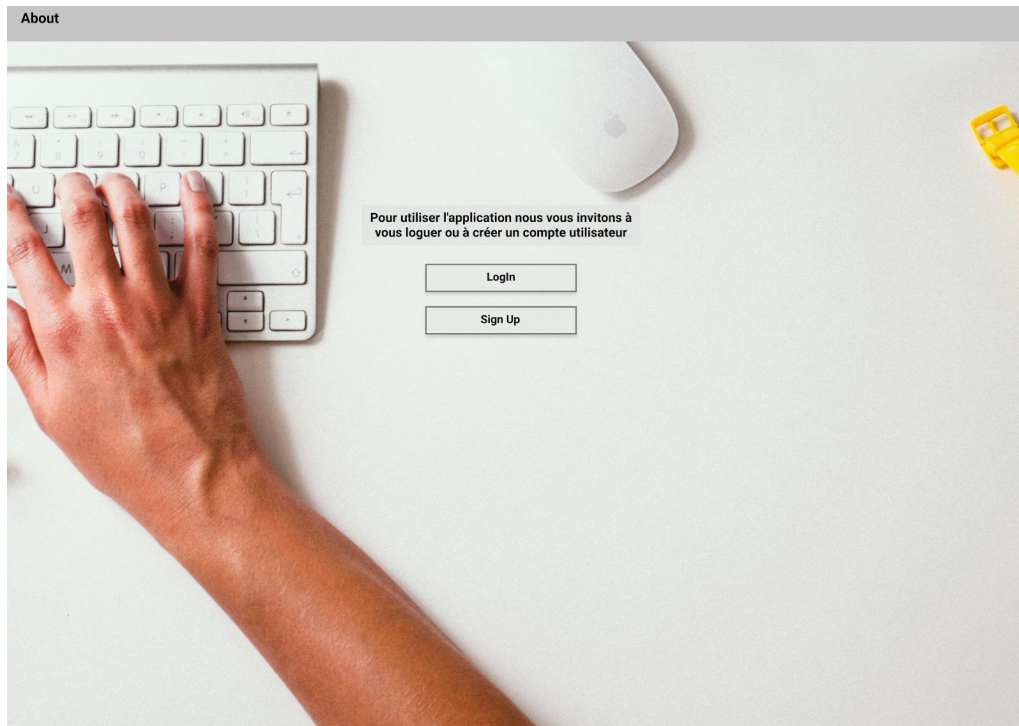
!./darknet detector train data/obj.data cfg/custom-yolov4-detector.cfg
yolov4.conv.137 -dont_show -map')

```

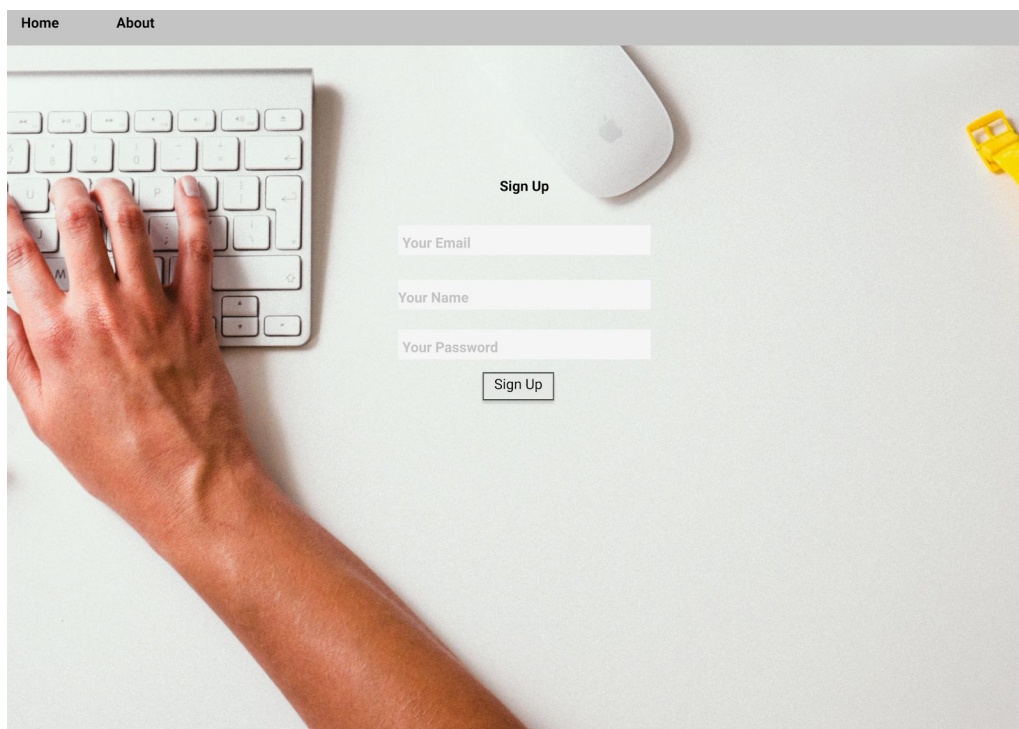
## Annexe VII: Mockups du frontend



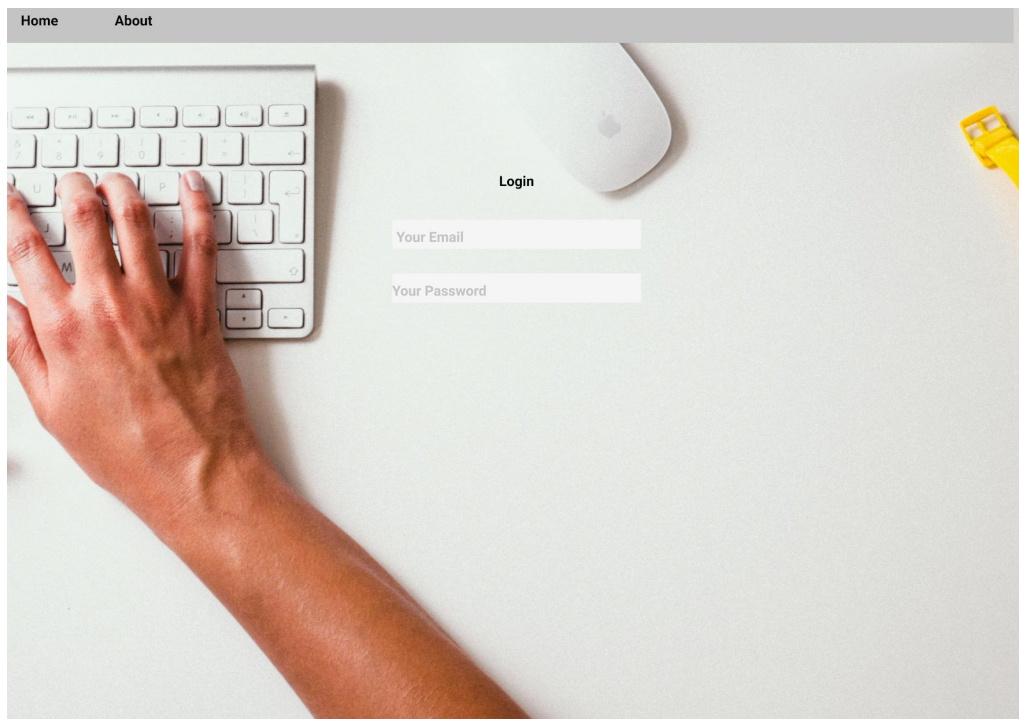
## Home avec un utilisateur non authentifié



## Sign up



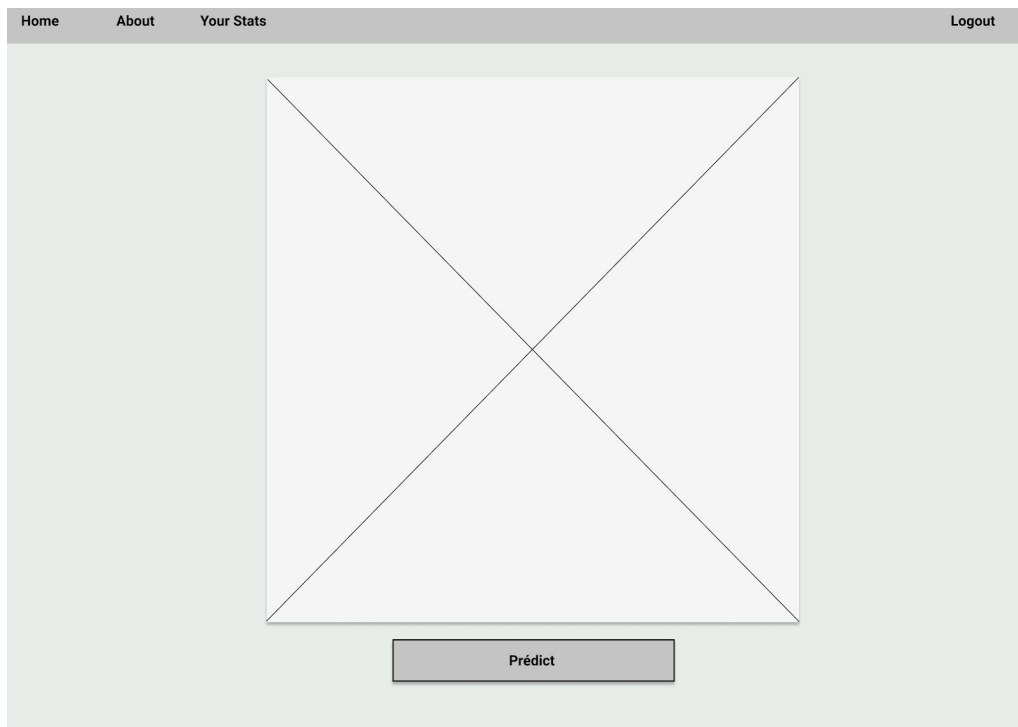
# Login



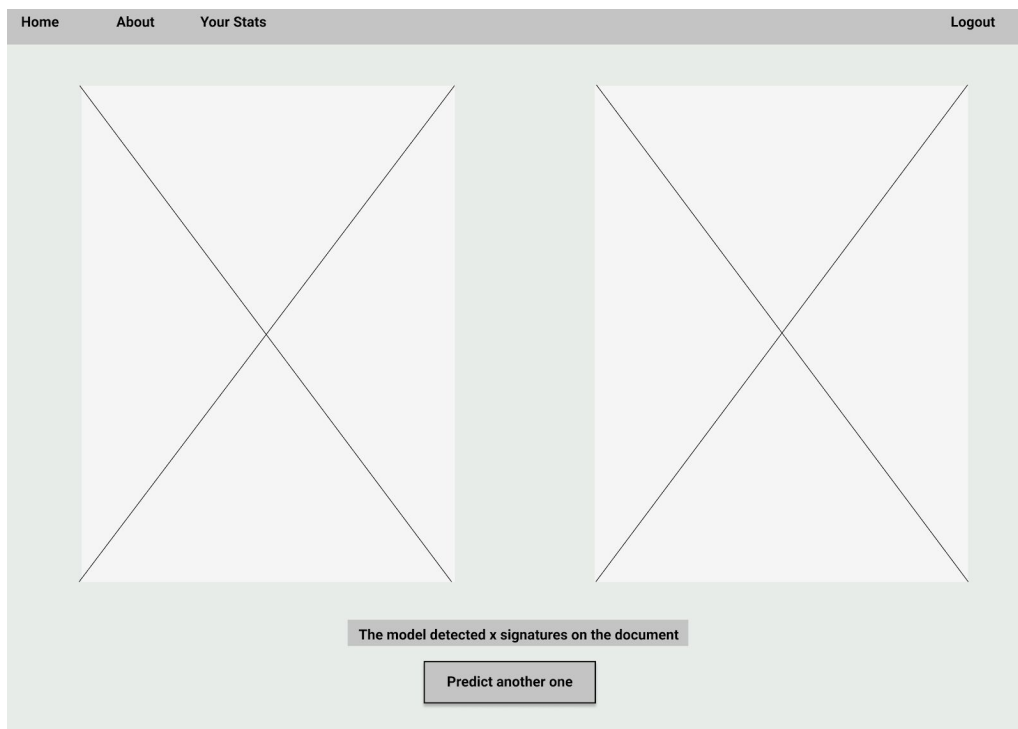
## Home avec un utilisateur authentifié



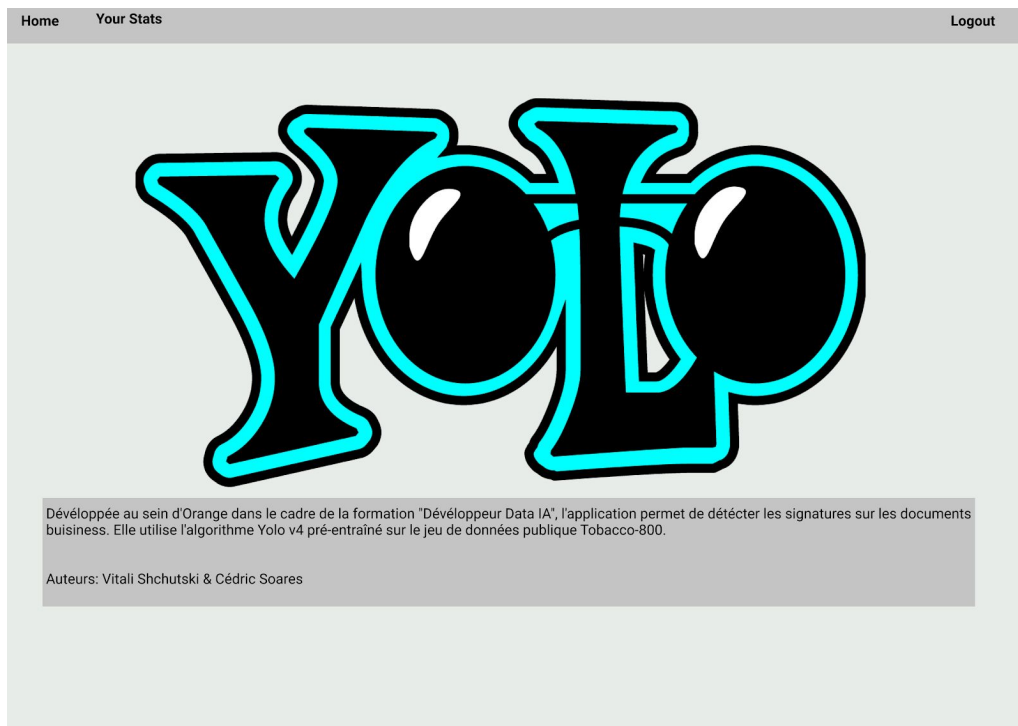
## Predict après sélection d'image



## Predict après récupération du résultat



## About



## Your stats



## Annexe VIII : Code des tests unitaires

```

from app import app
from models import *
import pytest
from werkzeug.security import generate_password_hash
import logging
import base64
import requests

@pytest.fixture
def client():
    app.config['TESTING'] = True
    log = logging.getLogger('pybrake')
    log.disabled = True
    with app.test_client() as client:
        with app.app_context() as cxt:
            db.drop_all()
            db.create_all(app=app)
            test_user = User(email='test@test.fr', name='test',
password=generate_password_hash('test', method='sha256'))
            db.session.add(test_user)
            db.session.commit()
            cxt.push()
        yield client

def login(client, username, password):
    return client.post('/login', data=dict(
        email=username,
        password=password
    ), follow_redirects=True)

def logout(client):
    return client.get('/logout', follow_redirects=True)

class MockResponse:

    @staticmethod
    def json():
        return {'prediction': [[1010, 1705, 217, 83], [344, 1637, 194, 84]]}

class TestApp():
    test_mail = 'test@test.fr'
    test_password = 'test'

    def test_home_not_authenticated(self, client):
        rv = client.get('/')
        assert rv.status_code == 200
        assert "Pour utiliser l'application nous vous invitons à vous loguer
ou à créer un compte utilisateur".encode("utf-8") in rv.data

```

```

def test_login_page(self, client):
    rv = client.get('/login')
    assert rv.status_code == 200
    assert "Login".encode("utf-8") in rv.data

def test_about_page(self, client):
    rv = client.get('/about')
    assert "Auteurs: Vitali Shchutski & Cédric Soares".encode("utf-8")
in rv.data

def test_signup_page(self, client):
    rv = client.get('/signup')
    assert rv.status_code == 200
    assert "Sign Up".encode("utf-8") in rv.data

def test_home_authenticated(self, client):
    rv = login(client, self.test_mail, self.test_password)
    assert "Please upload an image".encode("utf-8") in rv.data

def test_login_logout(self, client):
    rv = login(client, self.test_mail, self.test_password + 'x')
    assert "Your account does not exist or login details are note
correct please sign up or try again.".encode("utf-8") in rv.data
    login(client, self.test_mail, self.test_password)
    rv = logout(client)
    assert "Login".encode("utf-8") in rv.data

def test_new_user_signup_login(self, client):
    client.get('/')
    client.get('/signup')
    client.post('/signup', data=dict(
        email="new@test.fr",
        name="new",
        password="new"),
        follow_redirects=True)
    rv = client.post('/login', data=dict(
        email="new@test.fr",
        password="new"),
        follow_redirects=True)
    assert "Please upload an image".encode("utf-8") in rv.data

def test_predict_page_without_image(self, client):
    login(client, self.test_mail, self.test_password)
    rv = client.get('/predict')
    assert "Please, upload an image!".encode("utf-8") in rv.data

def test_predict_page_with_image(self, client):

```

```

login(client, self.test_mail, self.test_password)
with open('tests/test_image.png', 'rb') as test_file:
    rv = client.post('/home', data=dict(file=test_file,
filename='test_image.png'), follow_redirects=True)
    assert rv.status_code == 200
    assert "Predict".encode("utf-8") in rv.data

def test_result_page(self, client, monkeypatch):

    def mock_get(*args, **kwargs):
        return MockResponse()

    login(client, self.test_mail, self.test_password)
    with open('tests/test_image.png', 'rb') as test_file:
        client.post('/home', data=dict(file=test_file,
filename='test_image.png'), follow_redirects=True)

    monkeypatch.setattr(requests, "post", mock_get)
    rv = client.post('/predict', follow_redirects=True)
    assert "Predict another one".encode("utf-8") in rv.data

```



## Annexe IX : Dockerfiles et Docker-compose

## Dockerfile du conteneur d'entraînement

```
FROM nvidia/cuda:10.2-cudnn7-devel
ENV DEBIAN_FRONTEND noninteractive
ENV NVIDIA_VISIBLE_DEVICES all
ENV NVIDIA_DRIVER_CAPABILITIES compute,utility

LABEL maintainer="Vitali Shchutski and Cedric Soares"

#Linux libraries install
RUN \
    apt-get clean \
    && apt-get update \
    && apt-get install -y \
    apt-utils \
    software-properties-common \
    #&& add-apt-repository "deb http://security.ubuntu.com/ubuntu
bionic-security main" \
    && apt-get install -y \
    autoconf \
    automake \
    cmake \
    pkg-config \
    libavcodec-dev \
    libavformat-dev \
    libswscale-dev \
    libtool \
    build-essential \
    libopencv-dev \
    libv4l-dev \
    libxvidcore-dev \
    libx264-dev \
    libjpeg-dev \
    libpng-dev \
    libtiff-dev \
    openexr \
    libgtk-3-dev \
    gfortran \
    libatlas-base-dev \
    libtbb2 \
    libtbb-dev \
    libdc1394-22-dev \
    git \
    nano \
    wget \
    unzip \
```

```

python3 \
python3-pip \
python3-dev \
python3-numpy \
&& cd /usr/local/bin \
&& ln -s /usr/bin/python3 python \
&& python -m pip install --upgrade pip \
&& apt-get autoremove \
&& apt-get clean -y

#OpenCV, Darnet and Python packages install
WORKDIR /opt

RUN \
git clone https://github.com/opencv/opencv.git \
&& git clone https://github.com/opencv/opencv_contrib.git \
&& cd opencv \
&& mkdir build \
&& cd build \
&& cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D WITH_CUDA=OFF \
-D BUILD_opencv_python3=yes \
-D PYTHON_DEFAULT_EXECUTABLE=/usr/bin/python3 \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D OPENCV_EXTRA_MODULES_PATH=/opt/opencv_contrib/modules \
-D OPENCV_ENABLE_NONFREE=ON \
-D BUILD_EXAMPLES=ON .. \
&& make -j $(nproc) \
&& make install

WORKDIR /app

COPY requirements.txt yolov4-notebook.ipynb ./

RUN pip install --no-cache-dir -r requirements.txt \
&& git clone https://github.com/roboflow-ai/darknet.git

WORKDIR darknet

COPY ./Makefile ./

RUN \
make OPENCV=1 GPU=1 AVX=0 CUDNN=1 CUDNN_HALF=0 OPENMP=1 -j $(nproc) \
&& wget
https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optima
l/yolov4.conv.137 >/dev/null 2>&1

```

**WORKDIR** /app

**ENTRYPOINT** ["/bin/bash"]

## Docker-compose d'entraînement

```
version: '2.4'
services:
  signature-detector:
    build :
      context : .
      dockerfile : Dockerfile
    runtime: nvidia
    container_name: yolov4-train
    image: yolov4-darknet-cuda10-cudnn7
    stdin_open: true
    tty: true
    restart: unless-stopped
    ports:
      - "8888:8888"
    volumes:
      - train:/app
    depends_on:
      - mongodb
    networks:
      - backend
  mongodb:
    image: mongo:latest
    container_name: mongodb
    restart: unless-stopped
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: testuser
      MONGO_INITDB_ROOT_PASSWORD: testpass
      MONGODB_DATA_DIR: /data/db#Dem41ndev#
      MONGODB_LOG_DIR: /dev/null
    volumes:
      - mongodbbdata:/data/db

    networks:
      - backend

networks:
  backend:
    driver: bridge

volumes:
  mongodbbdata:
    driver: local
  train:
    driver: local
```

## Dockerfile de l'interface graphique

```
FROM python:3.8.5-slim-buster

LABEL maintainer="Vitali Shchutski and Cedric Soares"

# Install Linux packages
RUN \
    apt-get update \
    && apt-get install -y \
    apt-utils \
    build-essential \
    libopencv-dev \
    libjpeg-dev \
    libpng-dev \
    libgl1-mesa-glx \
    git \
    nano \
    && pip install --upgrade pip

WORKDIR /app

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

COPY . .

# Install python packages
RUN chmod +x start.sh
RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 5000

CMD ["/bin/bash", "-c", "chmod +x /app/start.sh && /app/start.sh"]
```

## Dockerfile de l'api

```
FROM tiangolo/uvicorn-gunicorn-fastapi:python3.7

#Linux libraries install
RUN \
    apt-get update \
    && apt-get install -y \
    apt-utils \
    software-properties-common \
    && add-apt-repository "deb http://security.ubuntu.com/ubuntu
xenial-security main" \
    && apt-get install -y \
    autoconf \
    automake \
    cmake \
    pkg-config \
    libavcodec-dev \
    libavformat-dev \
    libswscale-dev \
    libtool \
    build-essential \
    libopencv-dev \
    libv4l-dev \
    libxvidcore-dev \
    libx264-dev \
    libjpeg-dev \
    libpng-dev \
    libtiff-dev \
    openexr \
    libgtk-3-dev \
    gfortran \
    libatlas-base-dev \
    libtbb2 \
    libtbb-dev \
    libdc1394-22-dev \
    && apt-get autoremove \
    && apt-get clean -y

RUN pip install --upgrade pip \
    && pip install gdown \
    && gdown
https://drive.google.com/uc?id=1jH9r7Zl9gDkMoTd4uH0KKeh1lMDKroP1 \
    && gdown
https://drive.google.com/uc?id=1-1orZi-jV-iot3Vi3vp4SCRsk-QVl3Vm

COPY ./app /app
```

**COPY** requirements.txt .

**RUN** pip --no-cache-dir install -r requirements.txt

**EXPOSE** 8080

**WORKDIR** /app

**CMD** ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]



## Docker-compose de l'ensemble interface graphique et api

```
version : '3.7'
services:
  app:
    build :
      context: .
      dockerfile: Dockerfile_app
    container_name: app
    image: flask
    stdin_open: true
    tty: true
    restart: unless-stopped
    ports:
      - "5000:5000"
    depends_on:
      - api
    volumes:
      - ./app
    networks:
      - bridge

  api:
    build:
      context: .
      dockerfile: Dockerfile_api
    container_name: api
    image: fastapi
    stdin_open: true
    tty: true
    restart: unless-stopped
    ports:
      - "8080:8080"
    networks:
      - bridge
networks:
  bridge:
    driver: bridge

volumes:
  data:
```