

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

[polytech.univ-tours.fr](http://polytech.univ-tours.fr)

**Projet Recherche & Développement**

**2018-2019**

# **Développement d'une application reconnaissance d'objets**

**Ajout de fonctionnalités**

**Tuteur académique**  
**Maxime MARTINEAU**

**Étudiant**  
**Pierrick BOBET (DI5)**

3 avril 2019



# Liste des intervenants

Nom	Email	Qualité
Pierrick BOBET	<a href="mailto:pierrick.bobet@etu.univ-tours.fr">pierrick.bobet@etu.univ-tours.fr</a>	Étudiant DI5
Maxime MARTINEAU	<a href="mailto:maxime.martineau@univ-tours.fr">maxime.martineau@univ-tours.fr</a>	Tuteur académique, Département Informatique



# Avertissement

Ce document a été rédigé par Pierrick Bobet susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Maxime Martineau susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



## Pour citer ce document

Pierrick Bobet, *Développement d'une application reconnaissance d'objets: Ajout de fonctionnalités*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
  author={Bobet, Pierrick},
  title={Développement d'une application reconnaissance d'objets: Ajout de fonctionnalités},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2018-2019}
}
```



# Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
Liste des tableaux	vi
<b>1 Introduction</b>	<b>1</b>
<b>2 Description générale</b>	<b>2</b>
0.1 Description de l'existant.....	2
0.2 Outils à disposition.....	4
1 Environnement du projet .....	4
2 Acteurs, en-jeux et contexte .....	5
2.1 Utilisateurs de l'application en général .....	6
3 Objectifs .....	6
4 Bases méthodologiques .....	7
4.1 Gestion de projet.....	7
4.2 Planification du projet.....	8
4.3 Conventions .....	9
4.4 Définition d'une fonctionnalité terminée .....	9
5 Caractéristiques des utilisateurs .....	10
5.1 Utilisateurs de API REST de l'application .....	10

5.2	Utilisateurs du déploiement continue & gestion modèles .....	10
6	Fonctionnalités du système .....	10
7	Structure générale du système .....	12
<b>3</b>	<b>État de l'art / veille</b> .....	<b>14</b>
1	Fonctionnalité 1 - API REST .....	14
1.1	Rappel objectif & attente .....	14
1.2	Explication des algorithmes .....	14
1.3	Formations .....	14
1.4	Découverte de API REST .....	15
1.5	Veille Framework .....	16
1.5.1	API Flask Python .....	16
1.5.2	Framework Django REST .....	16
1.5.3	Framework TastyPie .....	17
1.6	Choix du API REST .....	17
2	Fonctionnalité 2 - Déploiement continu .....	18
2.1	Veille .....	18
3	Fonctionnalité 3 - Gestion modèles reconnaissance .....	19
3.1	Veille .....	19
<b>4</b>	<b>Analyse et conception</b> .....	<b>20</b>
1	Fonctionnalité 1 - API REST .....	20
1.1	Prototype Python API .....	20
2	Fonctionnalité 2 - Déploiement continu .....	22
3	Fonctionnalité 3 - Gestion modèles reconnaissance .....	23
<b>5</b>	<b>Mise en oeuvre</b> .....	<b>24</b>
1	Préambule qualité code .....	24
2	Présentation des fonctionnalités .....	25
3	Fonctionnalité 1 - API REST .....	25
3.1	Outils et librairies .....	25
3.2	Implémentations .....	26
3.3	Qualité/Performance .....	28
3.4	Mise en place de tests .....	30
4	Fonctionnalité 2 - Déploiement continu .....	31
4.1	Outils et librairies .....	31
4.2	Implémentations .....	31
4.3	Qualité/Performance .....	31
5	Fonctionnalité 3 - Gestion modèles reconnaissance .....	32
5.1	Outils et librairies .....	32
5.2	Implémentations .....	32
5.3	Qualité/Performance .....	32

<b>6 Bilan et conclusion</b>	<b>34</b>
<b>7 Annexes</b>	<b>36</b>
1 Description des interfaces externes du logiciel .....	36
1.1 Interfaces matériel/logiciel .....	36
1.2 Interfaces logiciel/logiciel .....	36
2 Spécifications fonctionnelles.....	36
2.1 Définition de la fonction API REST reconnaissance.....	38
3 Définition de la fonction déploiement continu .....	39
4 Définition de la fonction administration modèle - Ajout/modification .....	40
5 Définition de la fonction administration modèle - Suppression.....	41
6 Spécifications non fonctionnelles.....	41
6.1 Contraintes de développement et conception.....	41
6.2 Contraintes de fonctionnement et d'exploitation .....	41
6.2.1 Sécurité .....	41
7 Mise en place prototype Django REST.....	42
8 Mise en place prototype FileBrowser .....	46
9 Ajustement veille techno API REST.....	47
10 Mode opératoire API reconnaissance.....	48
11 Mode opératoire déploiement continu.....	52
12 Plan de tests .....	63
13 Organisation et Diagramme GanTT.....	68
<b>Comptes rendus hebdomadaires</b>	<b>74</b>
<b>Webographie</b>	<b>81</b>

# Table des figures

## 2 Description générale

1	Contexte application - Page accueil .....	2
2	Contexte application - Page résultat .....	3
3	Diagramme de séquence environnement PRD & projet groupe DI5 SI.....	5
4	Exemple d'issue .....	7
5	Exemple tableau Kanban.....	8
6	Tableau des tâches GanTT.....	8
7	Diagramme des tâches GanTT .....	9
8	Environnement de base Django .....	11
9	Diagramme de cas d'utilisation (gestion réseaux neurones & déploiement).....	11
10	Schéma interaction de l'application reconnaissance objet .....	12
11	Diagramme environnement déploiement.....	13

## 3 État de l'art / veille

1	Schéma simpliste API REST .....	15
2	Logo API Flask.....	16
3	Logo Django REST Framework .....	16
4	Logo TastyPie REST Framework .....	17
5	Cheminement Pipeline GitLab CI.....	18

## 4 Analyse et conception

1	Schéma hiérarchique fichiers reconnaissance.....	23
---	--	----



**5 Mise en oeuvre**

1	Diagramme des cas d'utilisations .....	25
2	Diagramme de séquence API Reconnaissance.....	27
3	Requête API - liste des algorithmes .....	28
4	Requête API - résultat analyse image .....	29
5	Requête API - résultat analyse image avec erreur auth.....	29
6	Django - Ajout/Modification/Suppression modèle de reconnaissance .....	33

**7 Annexes**

1	Diagramme de cas d'utilisation .....	37
2	Page d'accueil Django REST .....	39
3	Page d'accueil Django REST .....	42
4	Page API "users" Django REST .....	43
5	Page API ajout "users" Django REST .....	44
6	Page API après ajout "users" Django REST .....	45
7	Récupération info avec CURL pour "users" Django REST.....	45
8	Exemple d'interface Django FileBrowser.....	46
9	Exemple d'interface Django FileBrowser.....	46
10	Exemple d'interface Django FileBrowser.....	47



# Liste des tableaux

## 3 État de l'art / veille

1	Veille gestion modèle.....	19
---	----------------------------	----

# 1

## Introduction

Ce document a pour but de décrire l'ensemble des spécifications du Projet de Recherches et Développement ainsi que son cheminement. C'est un support qui a pour objectif d'être le plan d'analyse et de développement du projet. Il s'appuie sur les besoins du projet avec divers échanges entre les participants du projet, Maxime MARTINEAU au statut de doctorant ainsi que Barthélémy SERRES au statut de Responsable CETU ILIAD3 Chercheur associé au LIFAT. Ces encadrants apportent des réponses précises et détaillées pour mener à bien le projet. Le document est relu et approuvé par Maxime MARTINEAU afin de confirmer la cohérence avec les attentes du projet.

De plus, une partie fonctionnelle, technique et planning sont incluses dans ce document. Le projet abordé est la continuité d'un ancien projet de fin d'étude de l'étudiant en année 5 Polytech Tours Informatique par Pierre JUILLIE.

Le présent sujet porte sur une application web de reconnaissance d'objets. Le projet est réalisé par Pierrick BOBET (Maître d'œuvre). Ce cahier de spécification fera l'objet d'un contrôle par le professeur Nicolas RAGOT.

# 2

## Description générale

### 0.1 Description de l'existant

Ce projet est une suite d'un autre projet de recherche et développement. La précédente application développée par l'étudiant a satisfait les besoins du client. Certains besoins n'ont pas pu être aboutis et seront poursuivis par un groupe de DI5 SI lors d'un projet collectif.

L'application est en réalité un site internet qui est développée sous le langage Python et couplé avec le Framework Django. La page d'accueil du site web est une page d'information expliquant le projet et le but de l'application. Un bouton est présent pour tester l'application de reconnaissance d'objet.



Figure 1 – Contexte application - Page accueil

Le bouton "Testez maintenant !" redirige vers une autre page qui permet d'importer une image. Ainsi que des explications y sont présentes pour améliorer la prise de photo.

Après validation de l'importation, une page de résultat affiche la probabilité de reconnaissance d'objet :

[Innophyt](#) [Accueil](#) [Reconnaissance](#)

## Résultat du traitement de l'image

### Votre image



Nom de l'espèce	Pourcentage de proximité
Tipule	<div><div></div></div> 100,0%

[Retour !](#)

Figure 2 – Contexte application - Page résultat

## 0.2 Outils à disposition

Les outils qui m'ont été fournis au début de mon projet sont :

- code source du projet
- cahier du développeur : Explication du code, structure du système, informations importantes
- déploiement serveur
- documentation d'installation
- plan de tests

Requiert la mise en place d'une base de données MySQL avec une table nommée "insectdb".

## 1 Environnement du projet

Une application a été développée dans un précédent Projet de Recherches et Développement. Le langage de programmation est Python avec le Framework Django. La structure du logiciel permet sa réutilisation totale. Cette application est un site web permettant d'importer une image et d'exécuter un algorithme de reconnaissance. Le résultat en sortie de l'exécution permet de sortir une liste contenant des probabilités de reconnaissance d'insectes.

Le site web assure une liaison entre les utilisateurs et l'algorithme de reconnaissance d'insecte. Des documentations techniques et sources du projet sont fournis au démarrage de l'analyse de ce projet de fin d'étude.

Le projet de fin d'étude doit permettre d'élargir la liaison par le biais d'une application mobile. Cette application mobile ne fait pas partie de ce projet mais elle est affectée à un autre groupe d'étudiant DI5 SI dans le cadre d'un projet d'option. Cependant, il est nécessaire de mettre en place un moyen d'interaction entre le programme de reconnaissance et une éventuelle application externe (application mobile). Tout cela par le biais d'une API. Il sera nécessaire d'analyser le code de l'application Django existante afin de concevoir une API.

La figure suivante représente un diagramme de séquence. Il permet de situer l'environnement de ce projet de fin d'étude par rapport à un autre projet parallèle dirigé par un groupe de DI5 SI.

Ce groupe de SI est composé :

- Un premier sous-groupe qui traite la partie API paiement/authentification et historique. Ils gèrent les moyens de paiement pour offrir des accès privilégiés aux membres de l'application.
- Un autre sous-groupe est sur la partie application mobile.
- La partie qui concerne ce projet est l'API AI et le programme Python.

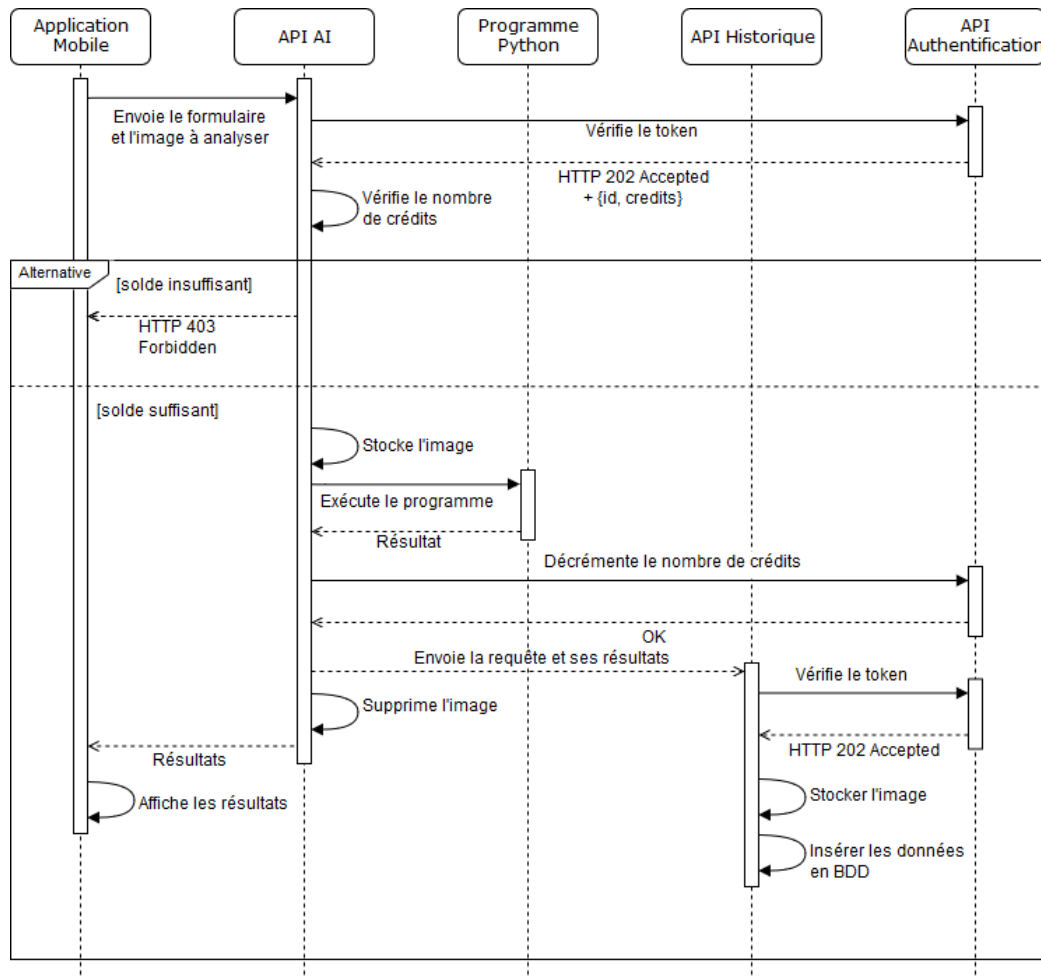


Figure 3 – Diagramme de séquence environnement PRD & projet groupe DI5 SI

## 2 Acteurs, en-jeux et contexte

Le précédent projet de fin d'étude avait comme contexte de mettre à disposition sous forme d'hébergement web un algorithme de reconnaissance d'insectes. Cette demande était formulée suite à un partenariat au sein de l'université de Tours ainsi que Damien MUNIER, en titre de client, travaillant au CETU Innophyt.

De plus, cette année 2018, un groupe d'étudiant en 5ème année à Polytech Tours Informatique Option Système Informations effectuent un projet sur ce sujet. Ils ont pour objectif de développer

une application mobile, et de mettre en place un API permettant de gérer l'authentification. L'application doit être divisée en deux parties concernant les utilisateurs. Une partie concernant des utilisateurs ayant payés et ainsi pour avoir des avantages privilégiés. Un espace de connexion doit être mis en place pour gérer les utilisateurs sur l'application.

Cette année, le projet est encadré par Maxime MARTINEAU et il est effectué par moi-même Pierrick BOBET. Les acteurs qui ont un lien avec le projet sont des chercheurs Donatello CONTE, Romain RAVEAUX, Barthélémy SERRES et Gilles VENTURINI. Le dernier étudiant qui a mené ce projet était Pierre JUILLIE dans le cadre de son Projet de Recherches et Développement.



## 2.1 Utilisateurs de l'application en général

L'application générale contient trois types d'utilisateurs, le sujet de ce projet de fin d'études n'inclut pas ses utilisateurs dans son contexte :

- Simple : Peuvent être des personnes sans aucune expérience du site ou de l'informatique. Ils peuvent naviguer sans difficulté sur les différentes pages.
- Payant : Correspond à un utilisateur simple ayant payé l'évolution de son statut. Il est donc nécessairement passé par un profil simple. Il possède toutes les capacités d'un utilisateur simple mais débloque des informations supplémentaires sur les résultats. Les différentes pages de l'application ne seront que peu ou pas impactées en terme de rendu entre un utilisateur simple et payant.
- Administrateur : Demande un niveau de compréhension minimum du site. Il faut bien connaître son intérêt, la notion d'utilisateur et éventuellement le fonctionnement de l'algorithme mais surtout ses résultats. Cet utilisateur garde un contrôle général sur l'application et sera de ce fait responsable du bon fonctionnement de celle-ci et notamment des fonctionnalités payantes. Celui-ci se verra accéder l'accès à des pages supplémentaires inaccessibles par les autres types d'utilisateurs

## 3 Objectifs

Ce projet fait suite à plusieurs projets de création d'une plate-forme web de reconnaissance d'insectes. Ces projets ont abouti à une première application qui permet une reconnaissance image d'insectes. Il s'agit d'une plate-forme d'aide à la décision. Ce projet de recherche et développement a pour nature d'améliorer le système existant. Diverses fonctionnalités et modifications restent à faire telles que :

- Concevoir une interface de programmation d'application API. La solution souhaitée est une API REST.
- Mettre en place l'intégration et le déploiement continu en production.
- Ajouter des fonctionnalités à l'interface administrateur de la plate-forme.

Le système est constitué au niveau matériel d'un serveur Linux Ubuntu 16 en production, ainsi qu'un accès réseaux. Du côté logiciel, le système est composé du langage Python avec le framework Python Django, une base de données MySQL est utilisée avec une table nommée "insectdb".



## 4 Bases méthodologiques

Pour mener à bien le projet, ayant un seul développeur, la mise en place d'une méthode Agile Kanban semble le plus pertinent.

Cette méthode nécessite un cahier des charges détaillé et validé du besoin du client en entrée de réalisation.

Étant une méthode risquée pour l'interaction avec le client durant le projet, il est important de faire valider chaque prototype de fonctionnalité pour vérifier la conformité.

Le projet se découpe bien en tâche/sous-tâche :

- Mise en place d'une gestion de projet au vu des fonctionnalités à implémenter (Découpage sous forme de tâche)
- Étude de l'existant, Analyse des librairies possibles pour aider à l'implémentation
- Veille technologique, test de librairies, prise en compte des contraintes techniques et fonctionnelles
- Enregistrement, étiquetage et trie/classement dans Zotero

### 4.1 Gestion de projet

La gestion de projet se fait par le biais de GitLab.com, cette plate-forme permet d'être au plus proche du code et ainsi centraliser la gestion et les sources du projet. Le projet se découpe en tâche et parfois en sous-tâche. Ces tâches sont représentées sous forme d'issue sous GitLab. Une issue contient un descriptif de la tâche ainsi que des données d'estimation de temps, d'importance avec un poids, et le temps réellement passé. Ce choix est justifié par la simplicité de mettre des annotations dans la description des issues/tâches, par exemple en citant une branche de développement, un morceau de code, des intervenants ou encore lier des tâches entre elles.

La figure suivante représente un exemple de tâche du projet qui a été représentée sous forme d'issue GitLab.

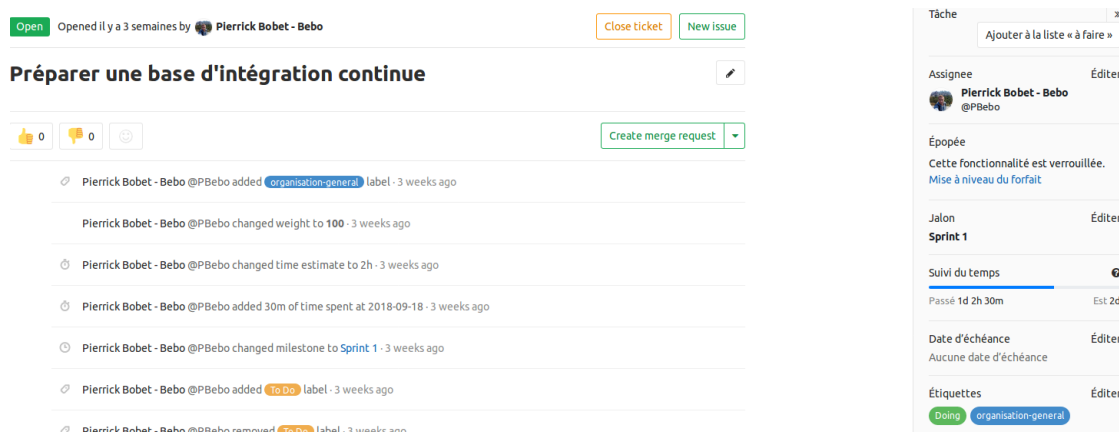


Figure 4 – Exemple d'issue

La figure suivante représente un exemple de tableau Kanban avec des issues/tâches.

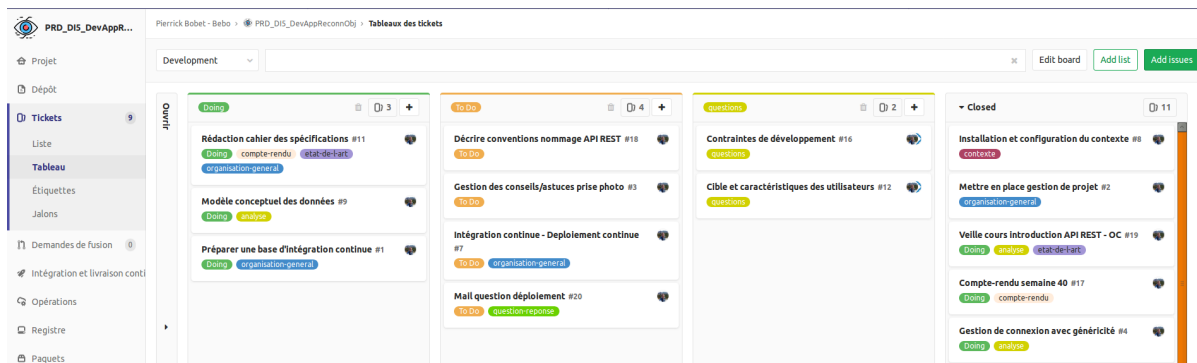


Figure 5 – Exemple tableau Kanban

## 4.2 Planification du projet

Cette sous-partie permet d'évaluer la faisabilité du projet et ainsi de découper les fonctionnalités à réalisés sous forme de tâche et sous-tâches. Ci-dessous, les trois figures représentent les estimations et le taux de réalisation effectué par rapport à la version de ce document.

	①	Name	Duration	Start	Finish	Predecessors
1		<b>Phase d'analyse</b>	<b>62.75 days?</b>	<b>9/19/18 8:00 AM</b>	<b>12/14/18 3:00 PM</b>	
2	✓	<b>Appropriation du projet</b>	<b>18 days</b>	<b>9/19/18 8:00 AM</b>	<b>10/12/18 5:00 PM</b>	
3	✓	Gestion de projet	2 days	9/19/18 8:00 AM	9/20/18 5:00 PM	
4	✓	Découverte de l'existant + test	13 days	9/26/18 8:00 AM	10/12/18 5:00 PM	3
5	✓	Veille - formation API REST	8 days?	10/9/18 8:00 AM	10/18/18 5:00 PM	4
6	✓	Intégration continue	7.125 days?	10/15/18 8:00 AM	10/24/18 9:00 AM	5
7		Cahier de spécification + rapp...	62.75 days?	9/19/18 8:00 AM	12/14/18 3:00 PM	
8	✓	<b>Veille Framework API</b>	<b>6.125 days?</b>	<b>10/18/18 2:00 PM</b>	<b>10/26/18 3:00 PM</b>	
9	✓	Veille Framework TastyPie	6 days?	10/18/18 2:00 PM	10/26/18 2:00 PM	
10	✓	Veille Framework Django API ...	6.125 days?	10/18/18 2:00 PM	10/26/18 3:00 PM	
11		Veille déploiement continu	6 days?	10/29/18 8:00 AM	11/5/18 5:00 PM	
12		Veille gestion modèles admin	7 days?	11/7/18 8:00 AM	11/15/18 5:00 PM	
13		<b>Phase développement</b>	<b>66.625 days?</b>	<b>1/7/19 8:00 AM</b>	<b>4/9/19 2:00 PM</b>	
14		Rapport PRD	64 days?	1/7/19 8:00 AM	4/4/19 5:00 PM	
15		Implémentation API reconnaiss...	24.375 days?	1/7/19 8:00 AM	2/8/19 11:00 AM	
16		Implémentation gestion modèl...	12 days?	2/11/19 8:00 AM	2/26/19 5:00 PM	
17		Implémentation déploiement c...	20 days?	2/27/19 8:00 AM	3/26/19 5:00 PM	
18		Rédaction documentations	10 days?	3/26/19 2:00 PM	4/9/19 2:00 PM	

Figure 6 – Tableau des tâches GanTT

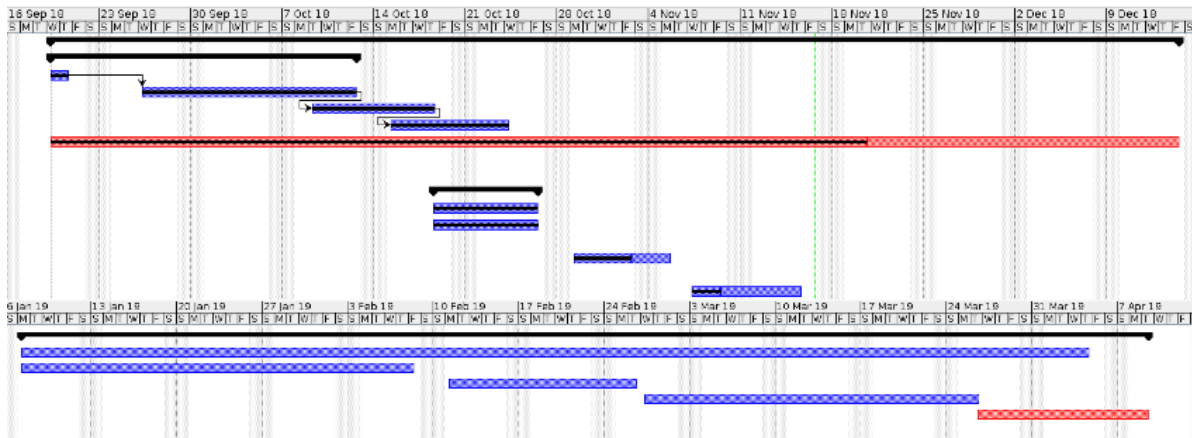


Figure 7 – Diagramme des tâches GanTT

### 4.3 Conventions

D'autre part, concernant l'architecture REST, il est important d'avoir de bonnes pratiques et des conventions à respecter. REST se base sur les URI (Uniform Resource Identifier) afin d'identifier une ressource. Tout en tenant compte des contraintes REST, le choix précis de l'URI/URL une application doit permettre de faciliter l'usage de l'API créée.

La réponse envoyée n'est pas une ressource, c'est la représentation d'une ressource. Ainsi, une ressource peut avoir plusieurs représentations dans des formats divers : HTML, XML, CSV, JSON, etc.

C'est au client de définir quel format de réponse il souhaite recevoir via l'entête Accept. Il est possible de définir plusieurs formats.

### 4.4 Définition d'une fonctionnalité terminée

Une fonctionnalité terminée se définit par plusieurs critères :

- Doit répondre aux besoins énoncés dans le cahier de spécifications, que ce soit au niveau du langage, des contraintes associées. Le résultat qui était prévu initialement doit correspondre avec ce qui a été énoncé dans le cahier de spécification .
- La solution doit avoir été testée manuellement avec plusieurs cas, que ce soit un usage normal et erroné. Cela permet dans un premier temps de tester rapidement la conformité du fonctionnement. Idéalement des tests unitaires peuvent être mis en place pour assurer la conformité en profondeur en traitant plus de cas d'utilisation avancés.
- Une documentation doit être intégrée au code. Avec une description simple et pertinente en en-tête de fonction.
- Une documentation plus spécifique à des parties dans le code doit être rédigée afin d'améliorer la compréhension du code. Par exemple à l'intérieur d'une fonction pour décrire une boucle d'itération.
- Lors d'un "push" sur une branche, les tests d'intégration continu sur Gitlab doivent être approuvés comme fonctionnels.

## 5 Caractéristiques des utilisateurs

### 5.1 Utilisateurs de API REST de l'application

- Développeur d'application externe par API :

**Connaissance de l'informatique :**

Requiert des compétences de développement applicatif afin d'intégrer l'API de ce projet dans son propre environnement. Par exemple les développeurs de l'application mobile pour la reconnaissance d'objet.

**Expérience de l'application :**

Utilisateur avancé qui connaît bien les actions de l'application pour trouver un intérêt à l'usage de l'API.

**Utilisateurs occasionnels :**

en terme de développement, l'API n'a pas besoin d'être constamment mise à jour, sauf en cas d'ajout de fonctionnalité.

en terme d'utilisation, des utilisateurs indirecte lié à l'API, par exemple les utilisateurs de l'application mobile font des transactions sans le savoir par le biais de l'API.

**Droits d'accès utilisateurs :**

Token authentifié par le biais d'un autre API en cours de développement par un groupe de DI5 SI.

Vérification d'un crédit d'utilisation de l'application. Toujours par le biais de l'API authentification.

### 5.2 Utilisateurs du déploiement continue & gestion modèles

- Administrateur ou développeur :

**Connaissance de l'informatique :**

Requiert des compétences d'utilisation avancée d'un système Linux Ubuntu. Des commandes ainsi que des services tels que Apache2 MySQL doivent être maîtrisés. Cet utilisateur devra être authentifié et restreint à l'environnement du système.

Requiert des compétences dans les domaines des réseaux de neurones et modèle de classe afin de savoir manipuler les fichiers modèles afin de les ajouter/supprimer/modifier.

Requiert des compétences au niveau de GitLab en général pour administrer l'intégration/-déploiement continu.

**Expérience de l'application :**

Doit comprendre le cheminement de l'application, dans le cas où des erreurs apparaissent afin de déboguer le programme.

**Utilisateurs occasionnels :**

Ces fonctionnalités sont utilisées régulièrement notamment le déploiement continu pour chaque nouvelle version.

**Droits d'accès utilisateurs :**

Utilisateur avec des droits avancées (administrateur)

## 6 Fonctionnalités du système

La figure suivante schématise un environnement avec le Framework Django Python :

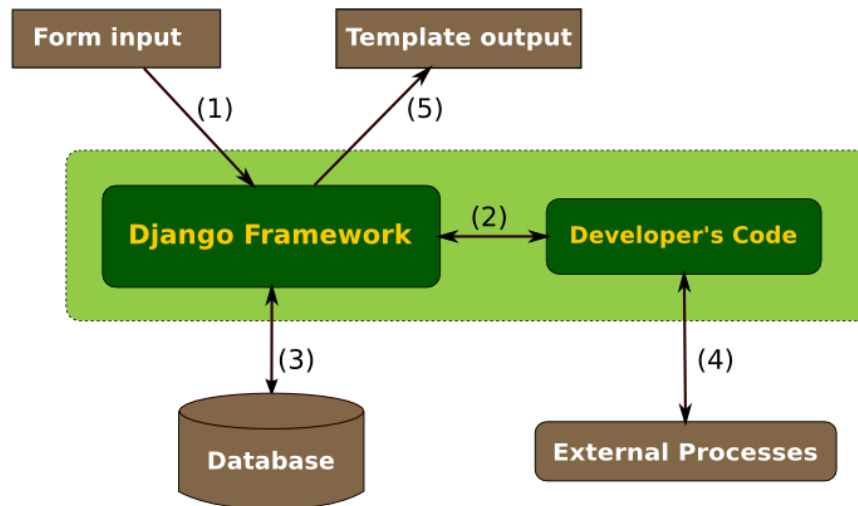


Figure 8 – Environnement de base Django

Chaque flèche de ce diagramme montre une interaction entre le Framework Django principal et les applications qui interagissent avec Django. Les flèches indiquent la direction des échanges.

Pour simplifier, "Form Input" représente tout type de données soumis à Django et, de la même manière, "Template output" pour représenter toutes les données possibles que Django peut renvoyer à un client.

Les fonctions peuvent se représenter par un diagramme de cas d'utilisation :

- La gestion des modèles de réseaux de neurones
- La fonctionnalité de déploiement continu.

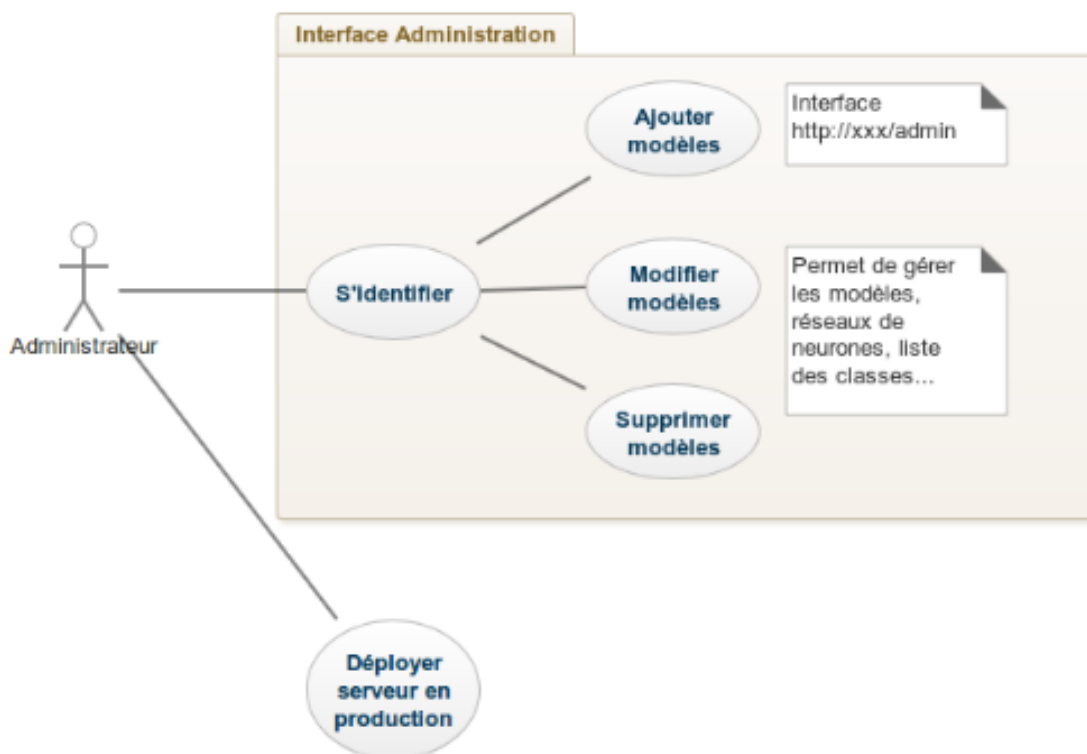
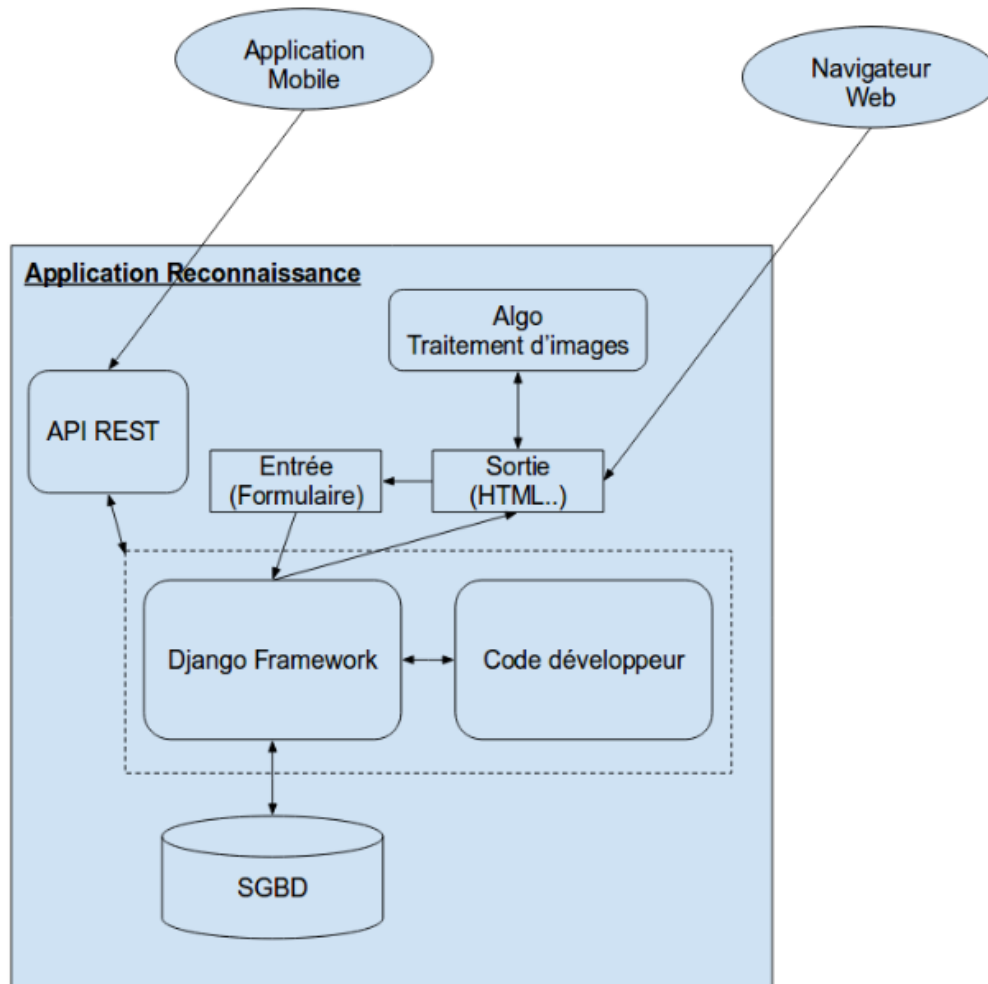


Figure 9 – Diagramme de cas d'utilisation (gestion réseaux neurones &amp; déploiement)

## 7 Structure générale du système

La figure suivante représente un schéma correspondant à l'application de reconnaissance ainsi que ses deux moyens d'interactions spécifiés :

- Application Mobile (par le groupe d'étudiant DI5 SI)
- Navigateur Web (actuellement l'existant)

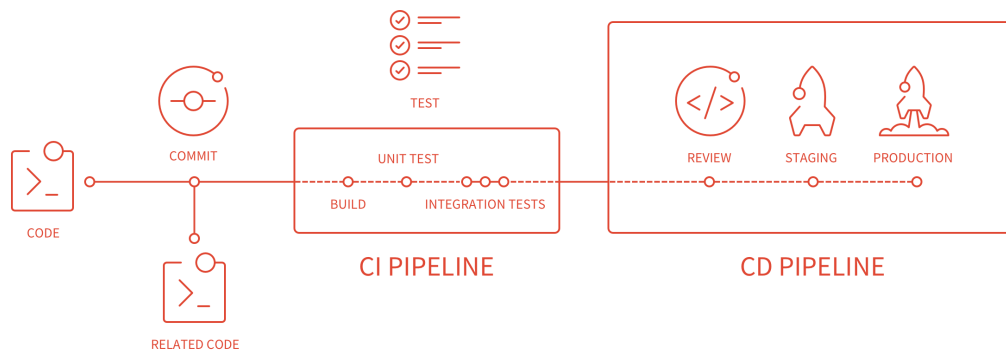


**Figure 10** – Schéma interaction de l'application reconnaissance objet

L'application mobile interagit par le biais de requête avec l'API REST reconnaissance de ce projet. L'API REST reconnaissance s'occupe de faire le lien avec les algorithmes de reconnaissance.

Le navigateur web interagit comme dans son contexte avec la sortie HTML grâce au Framework Django de l'application.

L'environnement concernant le déploiement continu peut se représenter de la manière suivante :



**Figure 11** – *Diagramme environnement déploiement*

Le code de l'application de reconnaissance est versionné sous un gestionnaire de source GitLab. De ce fait, il existe l'intégration continue et son déploiement continu sur plusieurs outils d'intégration continue (GitLab CI, Jenkins...)

L'intégration continue présente d'énormes avantages lorsque l'automatisation fait partie intégrante du flux de travail. Les outils d'intégration permettent un support intégré d'intégration continue, de déploiement continu et de livraison continue pour créer, tester et déployer votre application.

# 3

## État de l'art / veille

### 1 Fonctionnalité 1 - API REST

#### 1.1 Rappel objectif & attente

Le rôle de cette fonctionnalité est d'effectuer une transaction qui permet le lancement des algorithmes de reconnaissance et d'en ressortir un tuple de prédictions dans l'ordre des pourcentages les plus élevés sous format JSON.

#### 1.2 Explication des algorithmes

Les algorithmes utilisés pour la reconnaissance d'images utilise des fichiers de modèles au nombre de trois stockés dans le serveur de l'application Django. Ces fichiers seront utilisés par la librairie Tensorflow afin de calculer le résultat final. Il est important de bien comprendre que l'image d'entrée de l'algorithme subira quelques traitements définis dans la classe Algorithm d'application, il est donc pas nécessaire de forcer l'utilisateur à donner une image dite "parfaite" mais simplement de l'encourager.

#### 1.3 Formations

Afin de comprendre le concept et le fonctionnement d'un API (Application Programming Interfaces). Il a été nécessaire de suivre une formation durant la phase d'analyse. Les API REST en particulier sont de plus en plus utilisées car elles fonctionnent de la même manière que le reste du web. Ce cours est la propriété de Openclassroom [6], il s'intitule "**Utilisez des API REST dans vos projets web**"

Ce cours a permis d'en savoir plus sur :

- ce qu'est une API et ses cas d'utilisations
- les critères qui définissent une API REST
- la différence entre les API SOAP et REST
- comment est construite une API REST



- comment envoyer des requêtes à une API externe
- les bases pour construire sa propre API

## 1.4 Découverte de API REST

API REST est comme un ensemble de conventions tirant parti du protocole HTTP pour fournir un comportement CRUD (Créer, Lire, Mettre à jour et Supprimer) sur les objets et leurs collections.

Principes directeurs de REST

- Client - serveur - En séparant les problèmes d'interface utilisateur des problèmes de stockage de données, REST améliore la portabilité de l'interface utilisateur sur plusieurs plates-formes et améliorons l'évolutivité en simplifiant les composants du serveur.
- Sans état - Chaque demande client-serveur doit contenir toutes les informations nécessaires à la compréhension de la demande et ne peut tirer parti d'aucun contexte stocké sur le serveur. L'état de la session est donc entièrement conservé sur le client.
- Cacheable - Les contraintes de cache exigent que les données d'une réponse à une demande soient étiquetées implicitement ou explicitement comme pouvant être mises en cache ou non. Si une réponse peut être mise en mémoire cache, un cache client reçoit le droit de réutiliser ces données de réponse pour des requêtes équivalentes ultérieures.
- Interface uniforme - En appliquant le principe de génie logiciel général à l'interface composant, l'architecture globale du système est simplifiée et la visibilité des interactions est améliorée. Pour obtenir une interface uniforme, plusieurs contraintes architecturales sont nécessaires pour guider le comportement des composants. REST est défini par quatre contraintes d'interface : identification des ressources ; manipulation des ressources à travers des représentations ; messages auto-descriptifs ; et hypermédia en tant que moteur de l'état de l'application.
- Système en couches - Le style de système en couches permet à une architecture d'être composée de couches hiérarchiques en limitant le comportement des composants de sorte que chaque composant ne puisse pas «voir» au-delà de la couche immédiate avec laquelle il interagit.
- Code à la demande (facultatif) - REST permet d'étendre les fonctionnalités du client en téléchargeant et en exécutant du code sous la forme d'applets ou de scripts. Cela simplifie les clients en réduisant le nombre de fonctionnalités devant être pré-implémentées. [8]

La figure suivante représente de manière simpliste une architecture API REST. Un navigateur web interagi avec l'API, l'API fait le lien avec le serveur web et la base de données.

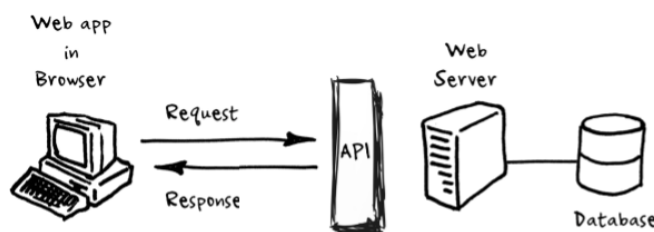


Figure 1 – Schéma simpliste API REST

## 1.5 Veille Framework

### 1.5.1 API Flask Python



Figure 2 – Logo API Flask

Flask est un framework très populaire et puissant pour la construction d'applications Web. Au cours des dernières années, les utilisateurs l'ont utilisée pour créer une API REST qui fonctionne bien avec les applications front-end découplées et modernes.

Apprendre "Flask" est plus facile et plus rapide. C'est très facile à configurer et à faire fonctionner. Contrairement à Django (qui est plus lourd), beaucoup des fonctionnalités disponibles ne seront pas utiles pour ce projet.

Les équipes de développement back-end sont souvent confrontées à un défi : permettre aux développeurs front-end, qu'ils soient internes ou distants, de créer facilement des clients conformes à l'API (application Web, application mobile ou même des outils CLI ...)

### 1.5.2 Framework Django REST



Figure 3 – Logo Django REST Framework

La structure Django REST est une boîte à outils puissante et flexible pour la création d'API Web.

Certaines des raisons pour lesquelles vous pouvez utiliser le framework REST :

- L'API navigable sur le Web est un gain énorme en termes de convivialité pour vos développeurs.
- Stratégies d'authentification, y compris les packages pour OAuth1a et OAuth2.
- Sérialisation prenant en charge les sources de données ORM et non ORM.
- Personnalisable à fond - utilisez simplement des vues standard basées sur des fonctions si vous n'avez pas besoin des fonctionnalités plus puissantes.
- Une documentation complète et un excellent soutien de la communauté.
- Utilisé et approuvé par des entreprises de renommée internationale, notamment Mozilla, Red Hat, Heroku et Eventbrite.

La documentation de Django REST Framework [2] offre une section destinée à simplifier les premiers pas dans la création d'un API REST. L'annexe "Mise en place Django REST" présente les résultats de la création d'un API.

### 1.5.3 Framework TastyPie



Figure 4 – Logo TastyPie REST Framework

Tastypie [7] est un Framework API de service Web pour Django. Il fournit une abstraction à la fois pratique, puissante et hautement personnalisable pour la création d'interfaces de style REST.

Tastypie est une application réutilisable (c'est-à-dire qu'elle ne repose que sur son propre code et vise uniquement à fournir une API de style REST) et qu'elle est adaptée à la fourniture d'une API à toute application sans modifier les sources de celle-ci.

Les besoins de tous ne sont pas les mêmes. Tastypie s'efforce donc de fournir une flexibilité pour modifier ou étendre son fonctionnement.

Il existe d'autres Frameworks API pour Django. Voici quelques raisons de choisir Tastypie :

- Avoir le besoin d'une API qui est RESTful et utilise bien HTTP.
- Ne pas à avoir à écrire son propre sérialiseur pour que la sortie soit correcte.
- Avoir une infrastructure API peu magique, très flexible et bien adaptée au domaine problématique.
- Besoin d'une sérialisation XML traitée de la même manière que JSON (et YAML aussi).

### 1.6 Choix du API REST

Il y a un vaste choix dans les solutions pour concevoir une API REST. Il est donc nécessaire de faire un choix sur la solution. Pour ce faire, il était intéressant de rechercher directement une solution au plus proche de l'application et des algorithmes de reconnaissance. La solution API REST Django était la solution la plus proche de l'application de reconnaissance d'objet. Cependant après avoir conçu des prototypes dans un environnement à part, la solution est assez lourde et complexe à mettre en oeuvre. Elle utilise beaucoup de fonctionnalités qui ne sont pas utiles pour ce projet, comme par exemple un système de persistance pour stocker les résultats des requêtes API. Ce qui n'a pas de pertinence dans ce projet.

D'autre part, la solution API Flask semble la plus pertinente et adaptée à l'application. Elle est populaire notamment sur la fréquence de mise à jour de son code source. Sa documentation et des exemples sur internet permettent de mettre en place sans difficulté un API REST puissant et stable.

Durant cette phase de veille technologique, un premier prototype d'API REST a été mis en place avec la solution API Flask.

## 2 Fonctionnalité 2 - Déploiement continu

### 2.1 Veille

Django permet de simplifier la vie des développeurs Web, cependant il ne sert à rien si on ne le déploie pas sur un vrai serveur de production. Django a facilité son déploiement, il peut utiliser le module de Apache2 `mod_wsgi`.

Dans la plupart des cas, ce sera le choix de déploiement le plus simple, le plus rapide et le plus stable.

Le déploiement de Django avec Apache et `mod_wsgi` est une manière éprouvée de mettre Django en production.

`mod_wsgi` est un module Apache qui peut héberger n'importe quelle application WSGI Python, y compris Django. Django fonctionne avec toute version d'Apache qui prend en charge `mod_wsgi`.

La documentation officielle de `mod_wsgi` représente la source ultime pour tout détail sur la manière d'utiliser `mod_wsgi`. Le point de départ le plus pertinent est la documentation d'installation et de configuration. [1]

Durant la phase de veille technologique, une virtualisation Docker de l'environnement de production a été mis en place. Cette virtualisation permet de représenter le cas réel avec la même configuration que le serveur de production. De ce fait, plusieurs exemple dont un très intéressant a été mis en place avec l'application de reconnaissance d'objet. Cette mise en place de prototype a été un succès et dont reste une solution retenue pour la phase de conception.

D'autre part, le projet est actuellement géré via la gestion de source GitLab. De ce fait, une recherche a été faite du côté de la mise en place de l'outil d'intégration continue et déploiement continu intégré à GitLab.

Le schéma suivant représente un cheminement d'intégration continu / déploiement continu. Il permet lors d'un commit par exemple, d'exécuter les tests unitaire/intégration/fonctionnel. Puis de déployer le code dans un environnement de test et de production.

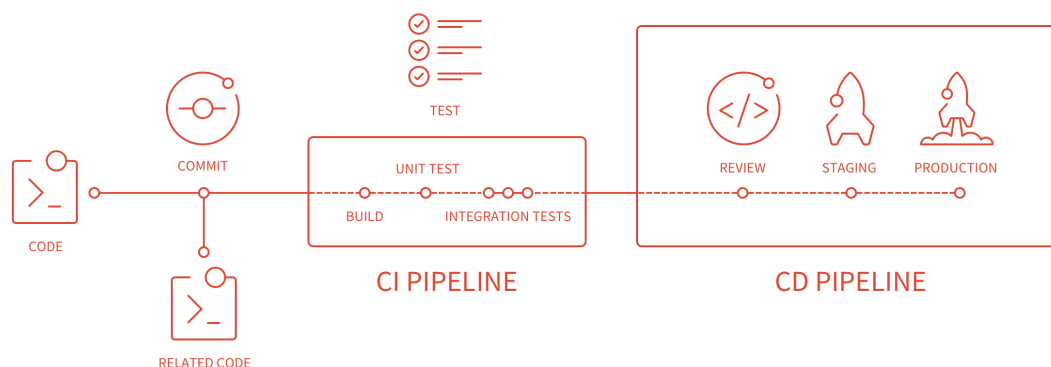


Figure 5 – Cheminement Pipeline GitLab CI

### 3 Fonctionnalité 3 - Gestion modèles reconnaissance

Cette fonctionnalité permet de mettre en place la généricité de l'application. C'est-à-dire que dans son contexte, l'application permet uniquement d'être utilisé pour la reconnaissance d'insecte.

Afin de rendre l'application générique, les algorithmes de reconnaissance avec les réseaux de neurones et modèles sont fournis.

Il faut donc adapté les fonctions actuellement développées. Et aussi proposer une façon de pouvoir gérer les fichiers de reconnaissance (réseaux de neurones et modèles).

#### 3.1 Veille

Plusieurs librairies sont disponible pour mener à bien la mission. Afin de déterminer la librairie qui sera utilisé lors de la conception, le choix se basera essentiellement sur les spécifications (comme la licence, l'état stable...) ainsi que sur la régularité des mises à jour et la popularité de la librairie.

**Table 1 – Veille gestion modèle**

PACKAGE	DJANGO-FILEBROWSER	DJANGO-FILER
Description	Media-Management with Grappelli	File and Image Management Application for django
Python 3?	Oui	Oui
Status Dev	Production/Stable	Production/Stable
Dernière mise à jour	Nov. 2, 2018, 12 :12 p.m.	June 2, 2018, 7 :40 a.m.
Popularité GitHub (Star)	714	1045

Un exemple d'utilisation de la librairie Django Filebrowser est disponible en annexe [Section 8](#) (Chapitre 7) . Elle permet de créer modifier supprimer des modèles et réseaux de neurones permettant la reconnaissance.

Il est possible de concevoir soit même sa propre page permettant d'importer des fichiers ou un fichier compressé contenant les 3 fichiers de modèle/réseau de neurones. Et de plus offrir un menu déroulant permettant de sélectionner la catégorie d'objet à reconnaître pour pouvoir supprimer. Cette solution est plutôt coûteuse en temps de conception.

# 4

## Analyse et conception

### 1 Fonctionnalité 1 - API REST

Le souhait est de pouvoir accéder à l'application de reconnaissance d'objet par le biais d'une requête API REST. L'API REST doit permettre d'importer une image et d'en sortir comme résultat les prédictions sous forme de liste de pourcentage avec les classes correspondantes. Le paramètre d'entrée est une image. Puis en sortie d'exécution c'est un tuple de prédictions dans l'ordre des prédictions en pourcentages des plus élevés. De plus, l'API REST doit permettre de lister les différentes classes qu'il est capable d'analyser (Insectes, plantes...).

Parmi les méthodes d'un API REST tel que GET, PUT, DELETE, POST. Celle qui correspond au besoin est la méthode POST. Elle permet notamment de représenter l'import d'un fichier vers le serveur et d'avoir un résultat en retour.

Une fois avoir choisi POST, il faut faire coller les méthodes permettant de lancer l'exécution de l'algorithme de reconnaissance. De ce fait, en suivant la documentation du code, il est indiqué que la classe "mainapp/algorithm.py" contient les méthodes souhaitées. À partir de là, la fonction "def get\_class\_list(self,img\_path):" est la méthode la plus haute hiérarchiquement dans l'appel des fonctions. Cette fonction permet donc le lancement de l'algorithme de reconnaissance et ainsi en faisant appel aux différentes fonctions de cette classe au préalable pour assurer le traitement de l'image.

En ayant eu toutes les informations précédentes, la phase de conception peut débuter. Il faut intégrer dans le contexte du projet le framework API REST retenu de la veille technologique. Puis créer une nouvelle transaction POST qui prends en paramètre d'entrée une image et en postcondition l'appel à la fonction "def get\_class\_list(self,img\_path):".

#### 1.1 Prototype Python API

Le code Python suivant est un premier prototype élaboré pendant la veille technologique de ce projet. Il permet en entrée de fournir une image et un identifiant d'algorithme de reconnaissance. Ensuite, il permet d'analyser l'image fourni par rapport au type d'algorithme de reconnaissance indiqué en paramètre. Puis de ressortir une liste de tuple sous format JSON.

```

1 IP_DU_SERVEUR_API_AUTH = '0.0.0.0' # adresse IP du serveur contenant l'API AUTH
2
3 # Vérification de l'utilisateur par rapport à son authentification et son crédit ↩
4   d'utilisation de l'app
5 url = 'http://IP_DU_SERVEUR_API_AUTH/auth/check'
6 headers = request.headers.get('Authorization')
7 response = requests.post(url, headers=headers)
8
9 # Si la réponse de l'API AUTH est autre que autorisé (202) alors arrêter l'exécution et ↩
10   retourner la réponse de l'API AUTH
11 if response.status_code != 202:
12     return response
13
14 # Le cas échéant, faire l'algo choisi avec l'image fourni
15 global test_algo
16
17 file = request.files['image']
18 algo = request.form['algorithm']
19 f = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
20
21 file.save(f)
22
23 # Permet de définir l'algorithme de reconnaissance qui sera utilisé par rapport à ↩
24   l'argument en entrée
25 test_algo = algorithm.Algorithm(algo)
26
27 # Permet de récupérer une liste contenant les résultats des probabilités de reconnaissance
28 list_class = test_algo.get_class_list(UPLOAD_FOLDER + file.filename)
29
30 # Permet d'arrondir et d'éliminer les résultats non pertinent pour une nouvelle liste en ↩
31   sortie
32 list_classFiltre = []
33 # Et on la remplit avec nos valeurs modifiées.
34 for name in list_class:
35     value = name[1] * 100
36     value = np.around(value, decimals=2)
37
38     if value != 0:
39         dict = {}
40         dict['species'] = name[0]
41         dict['probability'] = value
42         list_classFiltre.append(dict)
43
44 file.close()
45
46 # Retourne la liste de probabilité sous format JSON
47 return jsonify(list_classFiltre)

```

## 2 Fonctionnalité 2 - Déploiement continu

La plate-forme actuelle ne propose pas de fonctionnalité permettant d'offrir un déploiement continu en production. Ce souhait a été exposé par Maxime MARTINEAU et Barthelemy SERRES.

### Qu'est-ce que le déploiement continu ?

Le déploiement continu est une stratégie de développement logiciel dans laquelle toute validation réussissant les tests d'acceptation automatisés est automatiquement publiée dans le déploiement en production. [4]

Ce qu'il faut savoir c'est que lorsqu'on développe avec le Framework Django, l'utilisation du serveur de développement fourni par ce Framework n'est adapté que pour le développement. Ce n'est pas envisageable sur la mise en production dans une situation réelle. Ce serveur n'offre pas des conditions de sécurité et des performances suffisantes pour garantir un service stable. [3]

Les spécifications matérielles indiquent que le serveur de production a un système d'exploitation Linux Ubuntu Server 16 avec Apache2. Divers sites recommandent la mise en place de Apache2 avec mod\_wsgi. Ce dernier, le protocole WSGI est une spécification qui permet à un serveur web et une application web Python de communiquer ensemble et ainsi récupérer les pages web générés par le projet Django. [3]

Lors de la phase de veille de ce projet, cette fonctionnalité a été mis en œuvre par le biais d'un prototype. Un environnement virtuel Docker identique en terme de configuration et version de système au serveur actuellement en production. Cet environnement virtuel a permis de tester le déploiement en production par le biais de Apache2 avec le module apache\_wsgi. Grâce à cette veille, des documentations ont été étiquetées afin de mener à bien la partie réalisation.



### 3 Fonctionnalité 3 - Gestion modèles reconnaissance

La veille technologique a permis de faire ressortir une librairie très intéressante. De plus, elle a menée aussi à une autre solution, celle de concevoir soit même une solution sans passer par une librairie. Cependant, cela peut être très coûteux en temps. Après analyse de l'organisation et du temps accordé, il est préférable de mettre en place une solution sûre avec une librairie qui offre une stabilité. Si le temps le permet, il est possible de se pencher sur une conception manuelle.

Par le biais de la librairie, cela permet d'avoir une interface ergonomique. La représentation d'une catégorie de reconnaissance (exemple : insecte, animal, fleur...) peut se définir par un dossier à son nom (exemple : "insect"). Dans le dossier, on aurait les 3 fichiers de modèle, réseau de neurones...

La gestion de ces catégories de reconnaissance peuvent être fait par l'ajout d'un dossier et de ses fichiers, la suppression du dossier, la modification partielle d'un ou plusieurs fichiers.

Le schéma suivant permet d'illustrer l'idée pour hiérarchiser des différentes catégories de reconnaissance.

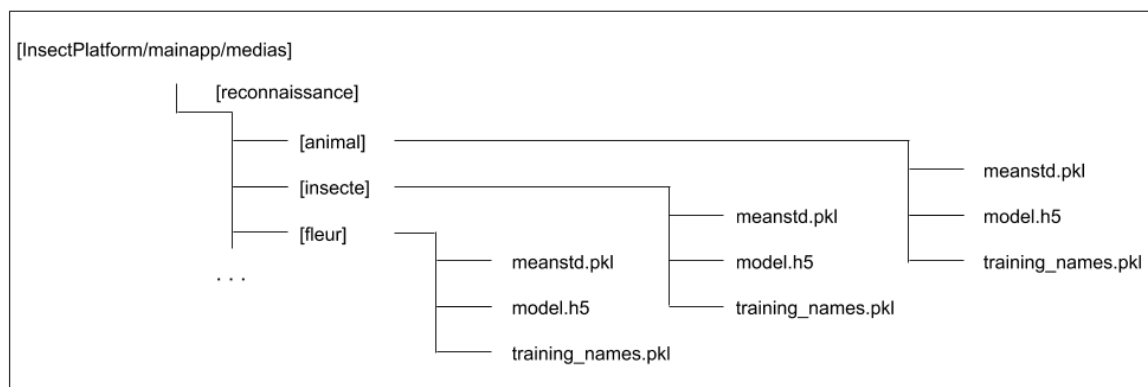


Figure 1 – Schéma hiérarchique fichiers reconnaissance

# 5

## Mise en oeuvre

### 1 Préambule qualité code

La production de code a respecté la précédente norme de l'ancien étudiant affecté au projet. Cela permet de garder une cohérence dans la lisibilité et qualité du code. Le nommage et convention respectent le principe de Python. Notamment dans les noms de variable/attribut et fonction/méthode. Par exemple 'mon\_attribut = 0' et 'void print\_result()'.  
↩

La structuration du code doit respecter les conventions de Django afin que le projet soit utilisable par le Framework Django.

La documentation du code utilise les conventions **docstring**. Chacune des méthodes/fonctions et classes sont commentés de la manière suivante :

```
1 def test_api_reconnaissance_get_result(self):
2     """
3     Objectif : Permet de tester un appel à l'API de reconnaissance avec :
4     :param un token d'authentification valide
5     :param un modèle de reconnaissance d'insecte
6     :param une image de Parasitoïde codée en base64
7
8     :return: JsonResponse Attendu = Retour la liste de résultat avec le Parasitoïde avec ↩
9             une probabilité de 99\% et d'autre sous résultat.
10    """
11    model = ReconnaissanceModel()
12    model.nom = 'insect'
13    model.meanstd = 'data_test/meanstd.pkl'
14    model.model = 'data_test/model.h5'
15    model.training_names = 'data_test/training_names.pkl'
16    model.save() # Ajout d'un modèle de reconnaissance d'insecte dans la base de données ↩
17    Django Test
```

D'autre part, la qualité de code se justifie par l'utilisation de nombreuses librairies et Framework permettant l'aboutissement de chacune des fonctionnalités. Il s'agit donc de ne pas réinventer la roue et ainsi gagner en maintenabilité de l'application avec des outils plus travaillés.

## 2 Présentation des fonctionnalités

Les trois fonctionnalités qui ont été implémentés dans ce projet sont :

- Requêtes API REST
- Déploiement continu
- Généricité de la plate-forme

Le diagramme des cas d'utilisation pour un utilisateur développeur est représenté ci-dessous pour les trois fonctionnalité :

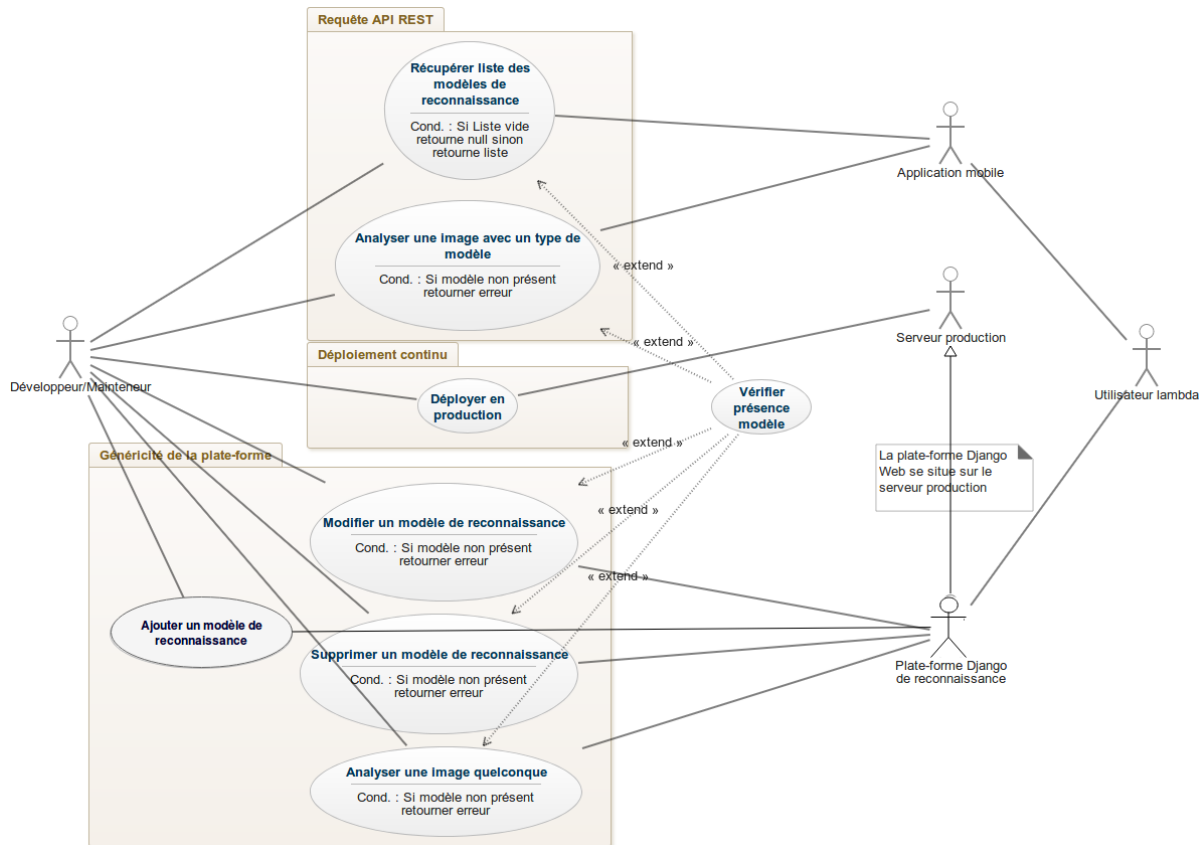


Figure 1 – Diagramme des cas d'utilisations

## 3 Fonctionnalité 1 - API REST

### 3.1 Outils et bibliothèques

Les outils et bibliothèques qui ont été nécessaires pour la partie conception de requête API REST sont Django Rest Framework et Requests.

Lorsqu'une requête lève une exception, par exemple un refus d'accès à la requête, la bibliothèque `from django.shortcuts import render, redirect, get_object_or_404` a été utilisée pour retourner des messages formatés. Ainsi que la bibliothèque `from django.http import HttpResponseServerError, HttpResponseNotFound`

Lorsqu'une requête s'exécute avec un retour souhaité de donnée, par exemple la reconnaissance d'objet avec un retour sous format JSON, la bibliothèque `from django.http import JsonResponse` a été utilisée afin de produire un résultat sous format JSON conforme aux attentes.

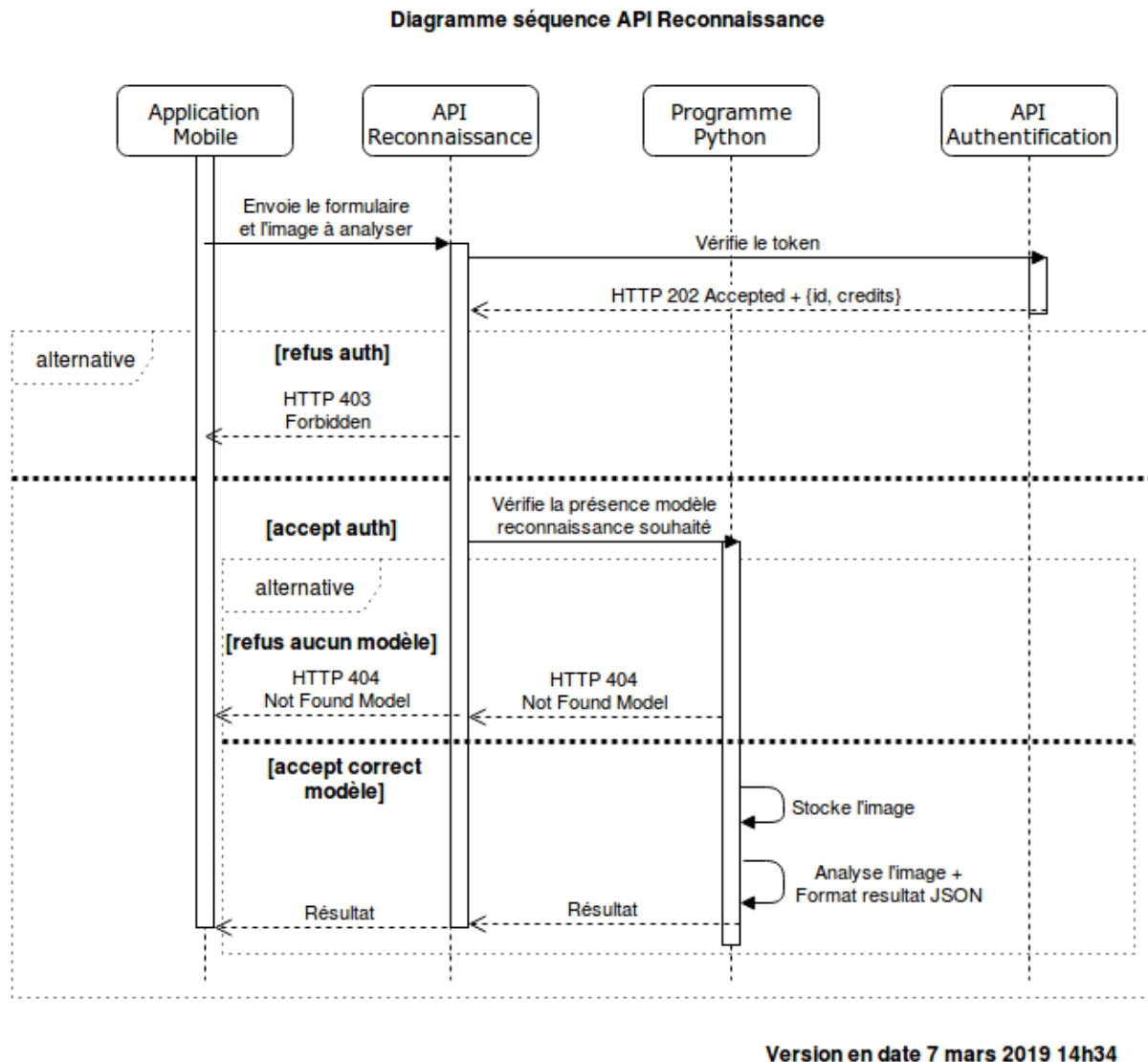
Une interface web est également disponible grâce à la librairie : Django API REST **from rest\_framework.views import APIView**. Cette interface permet d'effectuer les requêtes API REST directement en ligne.

### 3.2 Implémentations

Lors de la phase de veille technologique, la solution retenue était API FLASK. Cependant lors de la phase de conception, la solution finalement utilisée est Django API REST. Ce changement de solution est survenu lorsque qu'un premier prototype de requête API REST a été mis en oeuvre. Ce prototype qui embarquait la librairie API Flask était détaché du contexte Django. De ce fait, Django embarque une base de données MySQL qui contient les réseaux de neurones. Il n'était pas pertinent d'ouvrir une seconde connexion à la base de données en plus de l'application web initial. De plus, ce prototype isolait complètement les deux solutions ce qui provoquait l'utilisation de deux ports réseaux afin d'accéder à ces services.

Ce prototype compliquait la mise en place d'un API REST couplé à l'application Django. Une veille technologique a été approfondie sur l'application Django. Ce qui a mené à des résultats conformes aux attentes.

La figure suivante est un diagramme de séquence illustrant l'implémentation de l'API de reconnaissance :



**Figure 2 –** Diagramme de séquence API Reconnaissance

L'application mobile vient interroger l'API de reconnaissance en envoyant une image et une catégorie de reconnaissance. Ensuite, l'API de reconnaissance doit vérifier le token d'authentification dans son entête qu'elle a reçu de l'application mobile. La vérification se fait par l'appel d'un API d'authentification qui approuve ou non la validité du token. En cas d'échec, l'API d'authentification informe l'API de reconnaissance, qui elle même informe l'application mobile en retournant une erreur.

Dans le cas d'une bonne validité du token, l'API reconnaissance continue son cheminement. Il va vérifier la bonne présence du modèle de reconnaissance souhaité dans sa base de données. Auquel cas il retourne une erreur. Puis il stocke l'image qu'il a reçu dans sa base de données. Analyse l'image et retourne le résultat à l'application sous forme de JSON.

### 3.3 Qualité/Performance

La qualité de code est justifié par l'utilisation de Framework Django API REST, ainsi que d'autres bibliothèques permettant d'effectuer des requêtes API. Cette fonctionnalité a été vérifiée sur plusieurs jeux de tests.

Le premier jeu de tests concerne une utilisation "normale". L'utilisateur utilise un client API, par exemple Postman. Il souhaite avoir la liste des algorithmes de reconnaissance disponible :

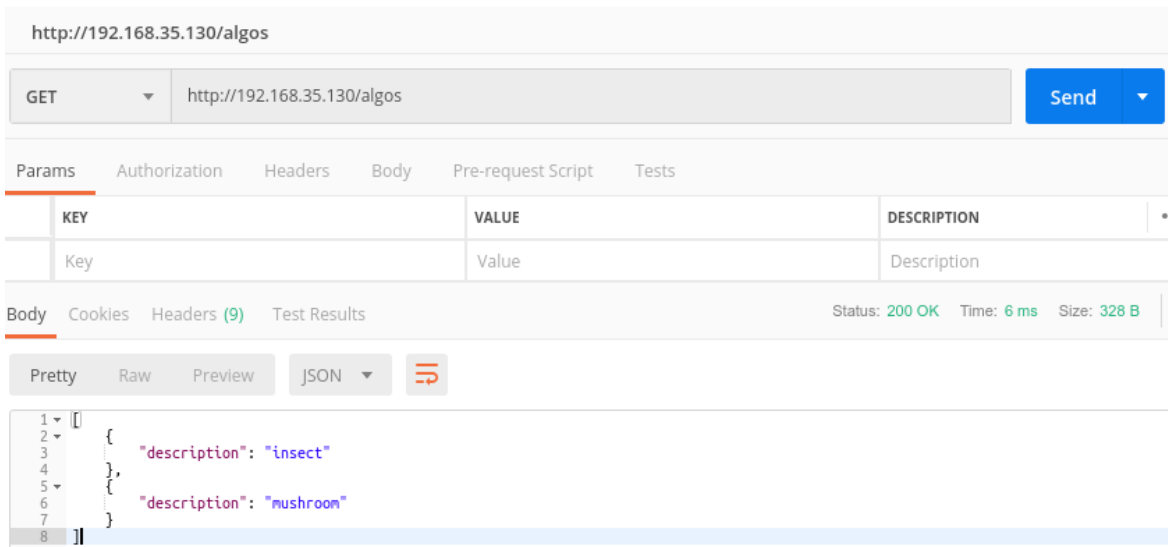


Figure 3 – Requête API - liste des algorithmes

Le second jeu de tests concerne une utilisation "normale" d'une analyse d'image. Les paramètres d'entrée de la requête sont les suivants :

- Entête : Authorization avec la valeur du token de type Bearer
- Body : Argument avec une clé "algo" et sa valeur "insect" correspondant à la catégorie de reconnaissance souhaitée.
- Body : Argument avec une clé "image" et sa valeur "image encodée en base64". L'image pour ce test est un parasitoïde.

Le résultat en sortie est conforme aux attentes. Il est représenté sous format JSON avec à 99% une reconnaissance de l'insecte Parasitoïde.

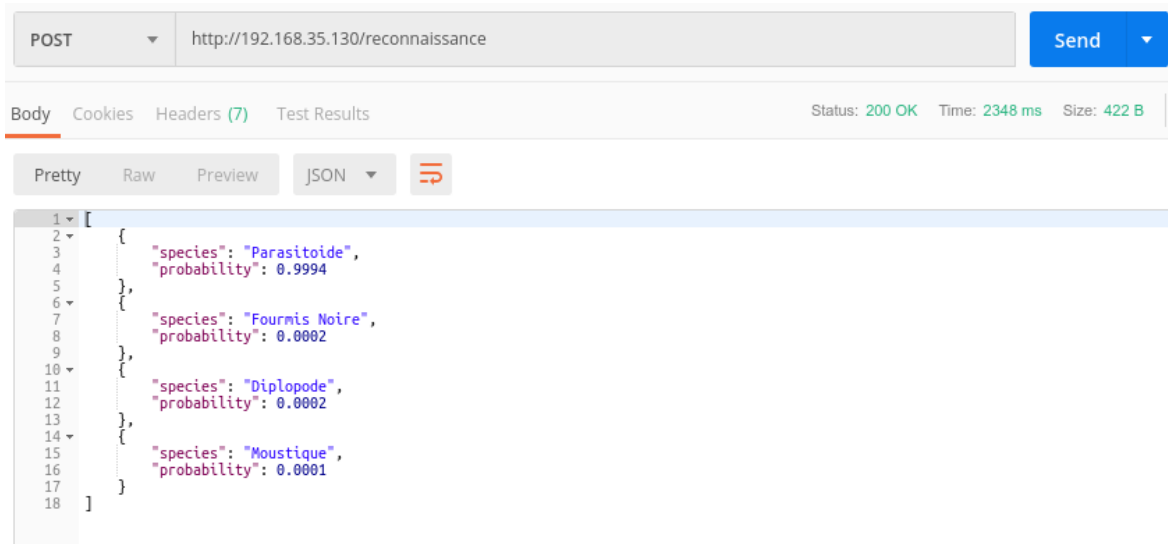


Figure 4 – Requête API - résultat analyse image

Le troisième jeu de tests concerne une utilisation "anormale" d'une analyse d'image. Les paramètres d'entrée de la requête sont les suivants :

- Entête : Authorization avec la valeur d'un token **non autorisé** de type Bearer
- Body : Argument avec une clé "algo" et sa valeur "insect" correspondant à la catégorie de reconnaissance souhaitée.
- Body : Argument avec une clé "image" et sa valeur "image encodée en base64". L'image pour ce test est un parasitoïde.

Le résultat en sortie est conforme aux attentes. Il permet de ressortir une erreur 500 sans exécuter l'algorithme. En effet, le token ne permet pas de lancer l'analyse de reconnaissance.

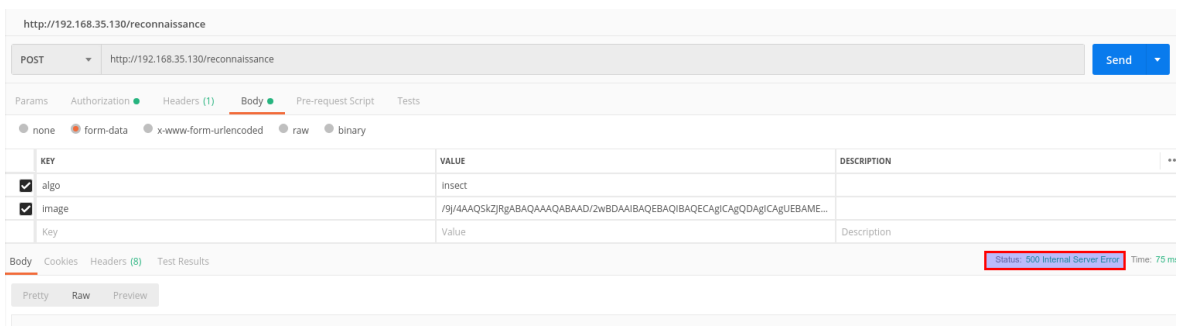


Figure 5 – Requête API - résultat analyse image avec erreur auth

### 3.4 Mise en place de tests

Cette fonctionnalité a été testée de manière plus approfondi. Des tests unitaires et fonctionnels ont été mis en place en utilisant les outils de test du Framework Django.

Les tests qui ont été mis en place sont les suivants :

- **def test\_api\_auth\_token(self):** : Il s'agit d'un test unitaire permettant de tester l'API authentification pour la vérification de token.

Attendu : Retourne une bonne exécution avec un code 202

- **def test\_api\_reconnaissance\_exception(self):** : Il s'agit d'un test fonctionnel qui permet de tester un appel à l'API de reconnaissance avec exception et :

- un token d'authentification valide
- un modèle de reconnaissance d'insecte
- une image de Parasitoïde codée en base64
- sans modèle de reconnaissance dans la base de données

Attendu : Retourne une erreur 404

- **def test\_api\_reconnaissance\_exception\_token(self):** : Il s'agit d'un test fonctionnel qui permet de tester un appel à l'API de reconnaissance avec exception de refus d'accès et :

- un token d'authentification invalide
- un modèle de reconnaissance d'insecte
- une image de Parasitoïde codée en base64

Attendu : Retourne une erreur 403 Forbidden

- **def test\_api\_reconnaissance\_get\_result(self):** : Il s'agit d'un test fonctionnel qui permet de tester un appel à l'API de reconnaissance avec :

- un token d'authentification valide
- un modèle de reconnaissance d'insecte
- une image de Parasitoïde codée en base64

Attendu : Retour la liste de résultat avec le Parasitoïde avec une probabilité de 99% et d'autre sous résultat.

- **def test\_api\_reconnaissance\_get\_algos(self):** : Il s'agit d'un test fonctionnel qui permet de tester un appel à l'API de reconnaissance pour récupérer la liste des modèles de reconnaissance avec :

- un token d'authentification valide

Attendu : Retourne la liste des modèles de reconnaissance disponible.



## 4 Fonctionnalité 2 - Déploiement continu

### 4.1 Outils et librairies

La fonctionnalité de déploiement a été mise en place grâce aux librairies et outils suivants :

- Apache2 et `mod_wsgi`
- MySQL Server
- Virtualenv et `pip3`
- Python Fabric3

Du fait que le serveur de production n'est actuellement pas accessible de l'extérieur (internet). Il a fallu revoir la veille technologique. La solution GitLab CI précédemment retenu dans la veille technologique nécessitait que le serveur de production soit disponible via internet. Pour ce faire, différentes solutions permettent de déployer depuis un poste développeur. Il s'agit de script à exécuter pas à pas.

### 4.2 Implémentations

La solution Python Fabric3 a été mise en place car elle était la plus adaptée pour la gestion de la base de données Django, l'installation des pré-requis pour la plate-forme de reconnaissance d'objet.

Il existe des risques pour cette solution. Notamment sur l'aspect sécurité où les identifiants d'accès à la base de données étaient en clair dans le script. Pour y remédier, les identifiants ne sont plus directement dans le script mais sont demandés lors de l'exécution du script. Une autre solution est de stocker les identifiants dans un fichier local sur la machine du développeur.

D'autre part, pour éviter à l'utilisateur de se connecter en SSH avec des identifiants et mot de passe. Il est possible d'utiliser une authentification par clé privé/publique. Ce qui permet d'éviter à l'utilisateur de saisir les identifiants de connexion au serveur de production ainsi que le droit d'administration (`sudo`).

### 4.3 Qualité/Performance

Le déploiement continu a été conçu grâce à une bibliothèque qui couple parfaitement avec Django. Cette bibliothèque est très complète et permet de nombreuses configurations permettant de faire cette tâche. Il s'agit donc d'un script Python3 Fabric3 respectant les conventions de nommage et la structure de la bibliothèque Fabric3.

Cette bibliothèque a permis d'interagir avec la base de données en production et les fichiers par le biais de l'outil `rsync` et `ssh` sur Linux.

## 5 Fonctionnalité 3 - Gestion modèles reconnaissance

### 5.1 Outils et librairies

Les outils et librairies utilisés pour la partie modèle `from django.db import models`, `from django.db.models.signals import post_delete`, `from django.dispatch import receiver`. Puis la librairie `"from django import forms"` pour le formulaire d'ajout/modification/suppression.

Et pour attacher le formulaire à la vue administration : `from django.contrib import admin`.

### 5.2 Implémentations

Cette fonctionnalité a nécessité l'ajout d'un modèle dans Django. Ce modèle contient 1 champs qui permet de nommer la classe de reconnaissance (exemple : "insecte"). Ainsi que trois autres champs contenant les fichiers de reconnaissance `meanstd`, `model` et `training_names`. Puis un dernier champs `date` pour historiser la date d'import de ce modèle. De plus, si l'administrateur souhaite supprimer un modèle, une méthode a été défini.

Un formulaire a été ajouté à Django dans le fichier `"forms.py"`.

Grâce aux fonctionnalités Django, la gestion des modèles de reconnaissance n'est accessible que pour les administrateurs authentifiés. Il a fallu déclarer le formulaire comme étant uniquement accessible à l'interface admin dans le fichier `admin.py`.

### 5.3 Qualité/Performance

La fonctionnalité de généricité de la plate-forme utilise au maximum ce qui est déjà fournis par le Framework Django. C'est-à-dire que la gestion des modèles se fait directement depuis l'interface fournis par Django soit "Django Admin". La plate-forme Django et sa page Admin propose un système de formulaire automatisé. C'est-à-dire qu'il faut déclarer un formulaire avec des champs. Lorsqu'un utilisateur interagit avec le formulaire, la plate-forme Django va représenter les données sous forme d'objet et les stockés dans une base de données.

Afin d'évaluer les performances, des jeux de tests ont été effectués :

- Le premier test consiste à ajouter un modèle via l'interface d'administration.
- Le second test consiste à modifier un modèle existant via l'interface d'administration.
- Le second test consiste à supprimer un modèle existant via l'interface d'administration.

La figure suivante illustre l'interface d'administration lors d'une modification d'un modèle existant.

Administration de Django

BIENVENUE, POLYTECH. VOIR LE SITE / MODIFIER LE MOT DE PASSE / DÉCONNEXION

Accueil › Mainapp › Reconnaissance documents › insect

Modification de reconnaissance document

HISTORIQUE

Description :

insect

Meanstd :

Actuellement: reconnaissance/meanstd/meanstd\_HJjHObI.pkl

Modifier: 

Browse...

 No file selected.

Model :

Actuellement: reconnaissance/model/model\_FT1YNUo.h5

Modifier: 

Browse...

 No file selected.

Training names :

Actuellement: reconnaissance/training\_names/training\_names\_JTw4Czx.pkl

Modifier: 

Browse...

 No file selected.

Supprimer

Enregistrer et ajouter un nouveau

Enregistrer et continuer les modifications

ENREGISTRER

Figure 6 – Django - Ajout/Modification/Suppression modèle de reconnaissance

# 6

## Bilan et conclusion

Ce semestre a été riche en recherches avec des solutions très intéressantes pour combler les besoins annoncés.

La première fonctionnalité de mise en place d'un API REST de reconnaissance d'objet a permis d'échanger avec le groupe de SI travaillant sur un API Authentification. De ce fait, des premiers prototypes de API REST pour la reconnaissance d'objet ont été mis en place. Cela a permis de tester les librairies Django API REST et Python Flask API.

Cette fonctionnalité est dans les temps prévu. Grâce au prototype conçu, le développement ne devrait pas avoir de forte complication.

La seconde fonctionnalité concernant l'intégration/déploiement continu est bien engagé. Ayant déjà une expérience sur GitLab Continus Integration, la mise en place de l'intégration continu devrait bien se dérouler. Le déploiement reste une partie sensible afin d'interagir avec le vrai serveur de déploiement.

Cette fonctionnalité est dans les temps prévu. Également grâce à des premiers essais mis en place. Une petite vigilance sur le déploiement est à noter.

La troisième fonctionnalité abordant la généricité de l'application est plus coûteuse en temps. Tout dépend de la solution retenue. Dans un premier temps, la solution la plus réalisable est l'utilisation d'une librairie permettant de gérer un arbre hiérarchique de fichiers/dossiers. Chaque dossier correspondra à une catégorie de reconnaissance (ex : animal, insecte, fleur...).

La solution plus coûteuse serait d'incorporer dans l'interface administrateur Django un bouton d'importation de fichier compressé et un autre moyen de sélection des catégories de reconnaissance pour pouvoir les supprimer.

Cette fonctionnalité est dans les temps prévu. Une grande vigilance est à porter.

Le planning qui sera suivit le prochain semestre pour la conception sera :

- Mise en place de l'API REST de reconnaissance
- Mise en place de la généricité de l'application
- Mise en place de l'intégration continu / déploiement continu

Chacune des fonctionnalités vont suivre l'ordonnancement suivant :

- Prototype
- Réalisation
- Test de la solution
- Validation de la solution
- Conception de documentation utilisateur et développeur

Le second semestre était la partie de mise en oeuvre de chacune des fonctionnalités spécifiées. L'ensemble des spécificités ont été réalisé avec succès. De nombreux points d'étapes ont été planifiés tout au long du semestre afin d'assurer de la satisfaction des réalisations. Conformément aux réunions, les modifications ont été apportés pour garantir la conformité des spécifications.

### **Gestion de projet**

La planification de gestion de projet a été conformément suivi. La fonctionnalité de "déploiement continu en production" a été cependant surestimé en temps de réalisation. Ce temps supplémentaire a permis d'améliorer les aspects sécurité lors du déploiement continu. De peaufiner et de rendre plus compréhensible les documentations. La documentation de code a également été complétée.

### **Qualité de code**

La qualité de code a été mené par l'utilisation de librairie stable et régulièrement à jour. Le précédent code a été refactorisé notamment sur la partie pour rendre l'application générique. Le code a été documenté à l'intérieur des méthodes ainsi qu'en en-tête de méthode et classe. Ce qui permet de décrire au plus proche chaque utilité du code.

# 7

## Annexes

### 1 Description des interfaces externes du logiciel

#### 1.1 Interfaces matériel/logiciel

Les moyens matériels mis en place sont un serveur de production Debian 9 Apache2 MySQL ayant l'application d'intégrée. Un accès VPN pour communiquer avec ce serveur de production. Ce serveur fait partie du réseau local de Polytech Tours bâtiment informatique.

#### 1.2 Interfaces logiciel/logiciel

La fonctionnalité d'interaction entre l'API de reconnaissance d'objet et le reste de l'application se définit par un API RESTful. C'est un style d'architecture qui permet de créer des applications Web, Web Service. Ainsi qu'un ensemble de conventions et de bonnes pratiques à respecter.

L'architecture REST utilise les spécifications du protocole HTTP, au lieu de réinventer une surcouche (comme SOAP par exemple).

- l'URI comme identifiant des ressources
- les verbes HTTP comme identifiant des opérations
- les réponses HTTP comme représentation des ressources
- les liens comme relation entre ressources
- un paramètre comme jeton d'authentification

### 2 Spécifications fonctionnelles

Ce projet fait suite à plusieurs projets de création d'une plate-forme web de reconnaissance d'insectes. Ces projets ont abouti à une première application qui permet une reconnaissance image d'insectes. Diverses fonctionnalités et modifications restent à faire telles que :

- Concevoir une interface de programmation d'application par requête HTTP. La solution souhaitée par le client est un API REST.

- Mettre en place une intégration continue et le déploiement continu.
- Améliorer en ajoutant des fonctionnalités dans l'interface administration.

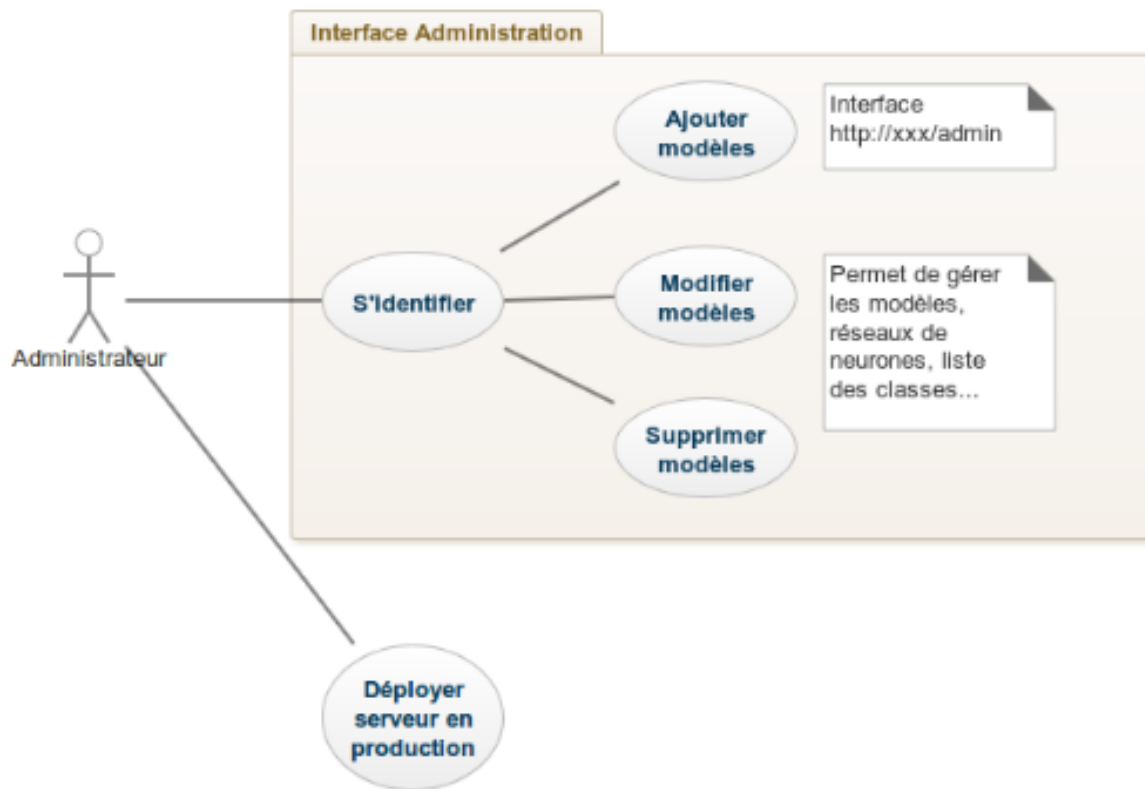


Figure 1 – Diagramme de cas d'utilisation

## 2.1 Définition de la fonction API REST reconnaissance

### Présentation de la fonction

- Cette fonctionnalité est nécessaire pour mettre en place une interaction entre la future application qui sera développée par le groupe d'étudiant DI5 SI. Le besoin est de pouvoir accéder à la partie de l'algorithme de reconnaissance.
- Il faut savoir que cette API interagi avec une application mobile et un API d'authentification. Ce sont donc dans deux environnement séparé. Autrement dit, l'application mobile n'a pas un accès direct à l'algorithme. Pour ce faire, il faut mettre en place cette fonctionnalité pour offrir un accès "extérieur" à l'algorithme de reconnaissance d'objet.
- Son rôle est d'effectuer une transaction qui permet le lancement de l'algorithme de reconnaissance et qui permet de ressortir un tuple de prédictions dans l'ordre des pourcentages les plus élevés.
- Cette fonction peut se résumer de la façon suivante : Fournir une image URL et un ID d'algorithme de reconnaissance -> Transaction API REST -> Retourne tuple de prédictions dans l'ordre des pourcentages les plus élevés.
- La priorité associée à la réalisation de la fonction est primordiale.

### Description de la fonction

#### Préconditions :

- Un jeton (token) d'authentification doit être valable pour effectué une transaction. L'API authentification est appelée au préalable pour contrôler la précondition.

#### Entrées :

- chemin de l'image que l'on va traiter
- identifiant de l'algorithme à utiliser pour la reconnaissance

#### Sorties :

- un tuple représentant les prédictions de reconnaissances de classe dans l'ordre des pourcentages les plus élevés

#### Postcondition :

- l'algorithme fait appel aux différentes fonctions de cette classe au préalable pour assurer le traitement de l'image.

#### Interactions :

- Cette fonction interagit avec les modèles, les réseaux de neurones, les listes des classes. Ce sont des fichiers dans "mainapp/reconnaissance" respectivement les fichiers suivant :
  - "mainapp/reconnaissance/meanstd.pkl",
  - "mainapp/reconnaissance/model.h5",
  - "mainapp/reconnaissance/training\_names.pkl".

La fonction interagit principalement avec la méthode "def get\_class\_list(self,img\_path):" de la classe "mainapp/algorithm.py", cette méthode découle de plusieurs sous méthode de la classe "mainapp/algorithm.py". Cette classe effectue les traitements d'images nécessaires avant de lancer l'algorithme de reconnaissance d'image. Elle charge le modèle de l'algorithme lors de son instantiation pour appeler les fonctions de tensorflow avec l'image et le modèle.

- Le groupe d'étudiant qui développe l'application mobile souhaite un format JSON en sortie de la manière suivante :



```

1  [
2      {
3          "species" : "???",
4          "probability": "0.83"
5      },
6      {
7          "species" : "???",
8          "probability": "0.12"
9      },
10     {
11         "species" : "???",
12         "probability": "0.05"
13     }
14 ]

```

Figure 2 – Page d'accueil Django REST

**Moyen de tests :**

- Mise en place de jeux de tests manuels dans un premier temps. Exécuter une requête par le biais d'un outil comme Postman, vérifier la conformité des résultats.
- Mise en place dans un second temps de test unitaire pour contrôler le résultat.

**Gestion des erreurs :**

- En cas d'erreur où l'API est indisponible ou inaccessible, la requête d'interaction indiquera une ressource introuvable avec le code d'erreur 404.
- En cas de non authentification de l'utilisateur, la requête d'interaction indiquera un problème de permission avec un code d'erreur en 50x.
- En cas d'utilisateur non abonné à une formule payante et/ou ayant un crédit d'utilisation de requête insuffisant dans son forfait, la requête d'interaction indiquera un problème de permission avec un code d'erreur en 50x.
- Concernant l'authentification, le groupe d'étudiant DI5 SI utilisent des JWT (JSON Web Token). Quand on fait une requête à une API, pour s'authentifier, on envoie dans le Header "Authorization" une chaîne qui contient "Bearer" puis un token qui correspond à des données chiffrées, que l'API reconnaissance n'est pas capable de comprendre. Ce Header doit être repris tel quel, transféré par une requête POST à l'API Auth, par exemple POST /auth/check. Cette API valide ou non le Token, et renvoie un status http correspondant. 202 = accepted = authentification OK, 401 = unauthorized (token present mais invalide), 498 = Token absent. Si l'API Auth retourne autre chose que 202 l'API Reconnaissance s'arrête là et réponds simplement le même status. Sinon, l'API Reconnaissance continue son exécution.

### 3 Définition de la fonction déploiement continu

**Présentation de la fonction**

- Déploiement continu en production
- Son rôle est de moderniser les pratiques de développements. La fonction automatise l'ensemble des étapes de déploiement (ou "mise en production"), de sorte qu'après chaque intégration qui se solde par des tests passant avec succès, l'application sur le serveur de production est mise à jour.
- La priorité associée à la réalisation de la fonction est primordiale.

**Description de la fonction**

**Entrée :**

- Gestion de source GitLab

**Sortie :**

- Déploiement du projet en production

**Postcondition :**

- Passage en revue des tests unitaires

**Interactions :**

- La fonction interagi avec le serveur en production ainsi que le projet à intégrer.

**Moyen de tests :**

- Lancer une pipeline intégrale sur GitLab en commençant par lancer l'ensemble des tests unitaires (ou d'autres tests d'intégration/fonctionnel)
- Lancer le mode de déploiement Staging et le mode de déploiement Production depuis la pipeline GitLab
- Contrôler le bon téléversement SFTP sur le serveur de staging ou production en question. Ce contrôle se fait en comparant les différences entre le code du serveur de production et le code en dernière date sur le GitLab qui a été poussé pour déploiement.

**Gestion des erreurs :**

- En cas d'erreur dans quelconques étapes de la pipeline de GitLab, un événement apparaît sur une étape problématique. Une console est accessible afin d'afficher les logs lors de l'erreur.
- En cas d'erreur plus complexe qui se déclencherai pendant ou après le téléversement des fichiers, des logs seront visibles au sein du serveur Linux habituellement situé dans /var/log/... Par exemple /var/log/syslog.

## 4 Définition de la fonction administration modèle - Ajout/modification

**Présentation de la fonction**

- Administration modèle de reconnaissance - Ajout/modification
- Son rôle est d'offrir la possibilité à un administrateur de gérer le modèle de reconnaissance. Il est possible pour lui de modifier le réseau de neurones ou la liste des classes.

**Description de la fonction****Entrée :**

- Les fichiers modèles à ajouter/modifier
- L'identifiant du modèle

**Sortie :**

- Message de réussite ou d'erreur le cas échéant

**Postcondition :**

- Effectuer l'ajout/modification et la liaison des fichiers au sein du projet. Gérer les cas de doublon.

**Interactions :**

- L'administrateur interagit avec l'interface d'administration et peut importer ses fichiers.

## 5 Définition de la fonction administration modèle - Suppression

### Présentation de la fonction

- Administration modèle de reconnaissance - Suppression
- Son rôle est d'offrir la possibilité à un administrateur de gérer le modèle de reconnaissance. Il est possible pour lui de supprimer le réseau de neurones ou la liste des classes.

### Description de la fonction

#### Entrée :

- Les fichiers modèles à supprimer

#### Sortie :

- Message de réussite ou d'erreur le cas échéant

#### Postcondition :

- Effectuer la suppression des fichiers au sein du projet. Gérer les cas de doublon.

#### Interactions :

- L'administrateur interagit avec l'interface d'administration et peut supprimer ses fichiers.

## 6 Spécifications non fonctionnelles

### 6.1 Contraintes de développement et conception

- **Matériels** : Le serveur de production est composé d'un système d'exploitation Linux Ubuntu 16.04 avec comme serveur web Apache2, ainsi qu'une base de données MySQL.
- **Langages de programmation** : Le projet est actuellement une plate-forme développée avec le langage Python, HTML, CSS. La syntaxe des types de fichiers XML, JSON sont imposés dans la réalisation du projet.
- **Bibliothèque** : plate-forme web avec Framework Django.
- **Environnements nécessaires** : accès VPN au serveur de production notamment pour la mise en place du déploiement continu.

### 6.2 Contraintes de fonctionnement et d'exploitation

#### 6.2.1 Sécurité

La fonctionnalité qui concerne la création d'un API REST, pour permettre l'interaction avec l'algorithme de reconnaissance d'objet, nécessite une attention sur la sécurité. L'API devra être accessible uniquement localement car elle interagira uniquement avec un autre API d'authentification ainsi que l'application mobile.

## 7 Mise en place prototype Django REST

La structure Django REST est une boîte à outils puissante et flexible pour la création d'API Web. Il est facile de l'essayer dans un environnement en suivant un démarrage rapide grâce à la documentation officielle [www.django-rest-framework.org/tutorial/quickstart/](http://www.django-rest-framework.org/tutorial/quickstart/) [5].

Les figures suivantes représentent la mise en place d'un API Django REST par l'étudiant Pierrick BOBET.

La figure suivante représente la page d'accueil de l'API, il est possible de s'authentifier pour accéder à des requêtes API sensibles. Les différentes requêtes publiques sont visibles dans le GET. Il est possible de consulter des modèles "users" et "groups". Les liens URL sont "http://localhost:8000/users/" ou "http://localhost:8000/groups"

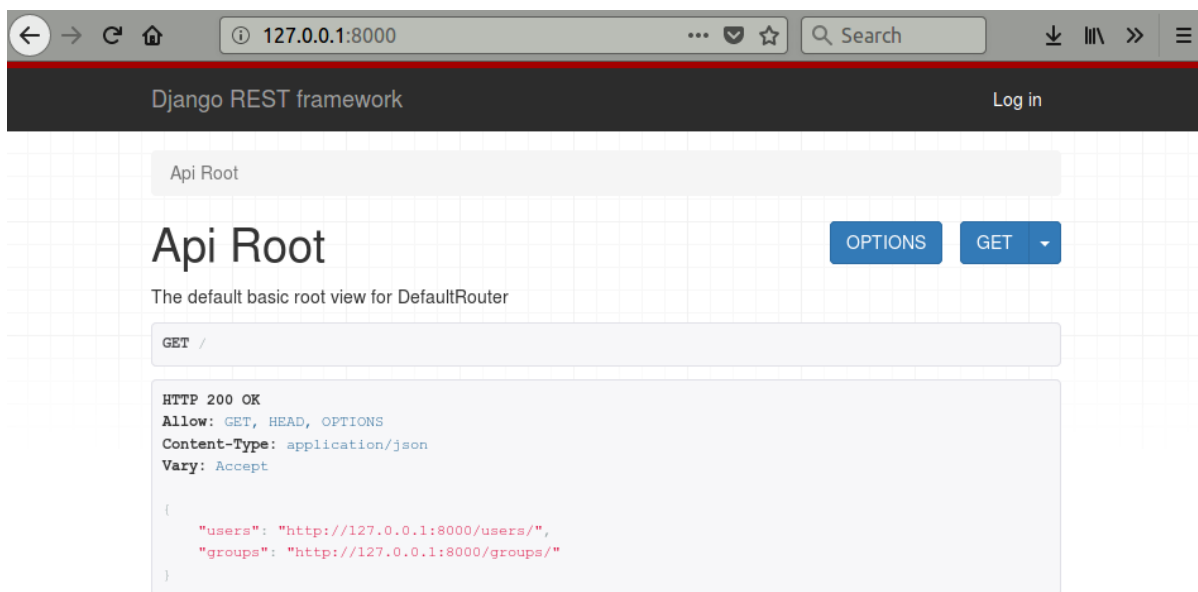


Figure 3 – Page d'accueil Django REST

La figure suivante représente le lien URL "http://localhost:8000/users/" qui permet de consulter (GET) ou ajouter (POST).

Api Root / User List

## User List

API endpoint that allows users to be viewed or edited.

GET /users/

```

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "count": 1,
  "next": null,
  "previous": null,
  "results": [
    {
      "url": "http://127.0.0.1:8000/users/1/",
      "username": "admin",
      "email": "pierrick.bobet@etu.univ-tours.fr",
      "groups": []
    }
  ]
}

```

Raw data HTML form

Username

Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.

Email address

Groups

No items to select.

The groups this user belongs to. A user will get all permissions granted to each of their groups.

POST

Figure 4 – Page API "users" Django REST

La figure suivante représente l'ajout d'un utilisateur par le biais de API Django REST.

Django REST framework Log in

Api Root / User List

## User List OPTIONS GET

API endpoint that allows users to be viewed or edited.

POST /users/

```

HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Location: http://127.0.0.1:8000/users/2/
Vary: Accept

{
  "url": "http://127.0.0.1:8000/users/2/",
  "username": "Toto",
  "email": "toto@tata.fr",
  "groups": []
}

```

Raw data HTML form

**Username**

Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.

**Email address**

**Groups**

No items to select.

The groups this user belongs to. A user will get all permissions granted to each of their groups.

POST

Figure 5 – Page API ajout "users" Django REST

La figure suivante représente l’affichage de l’utilisateur ajouté précédemment.

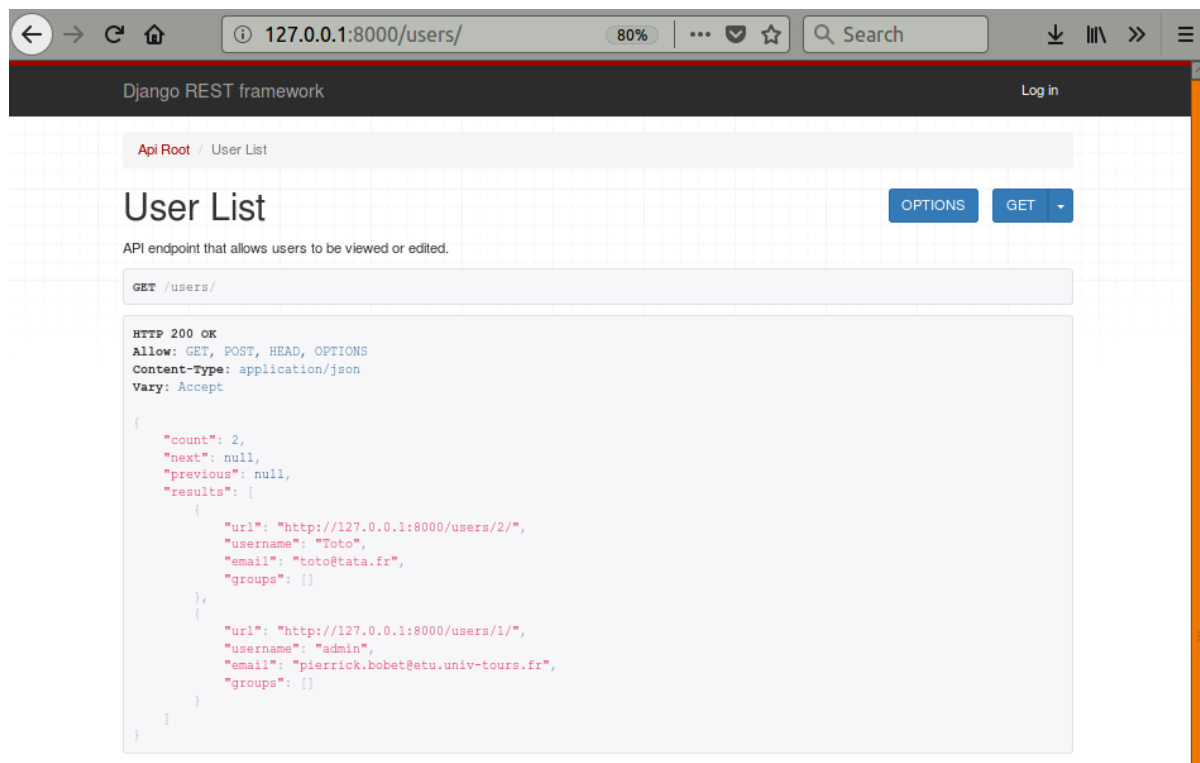


Figure 6 – Page API après ajout "users" Django REST

La figure suivante représente l’ajout d’un utilisateur par le biais de API Django REST. La figure suivante représente la récupération de données d’utilisateur par le biais de la commande CURL.

```
➔ ~ curl -H 'Accept: application/json; indent=4' -u admin:polytech http://127.0.0.1:8000/users/
{
  "count": 1,
  "next": null,
  "previous": null,
  "results": [
    {
      "url": "http://127.0.0.1:8000/users/1/",
      "username": "admin",
      "email": "pierrick.bobet@etu.univ-tours.fr",
      "groups": []
    }
  ]
}
```

Figure 7 – Récupération info avec CURL pour "users" Django REST

## 8 Mise en place prototype FileBrowser

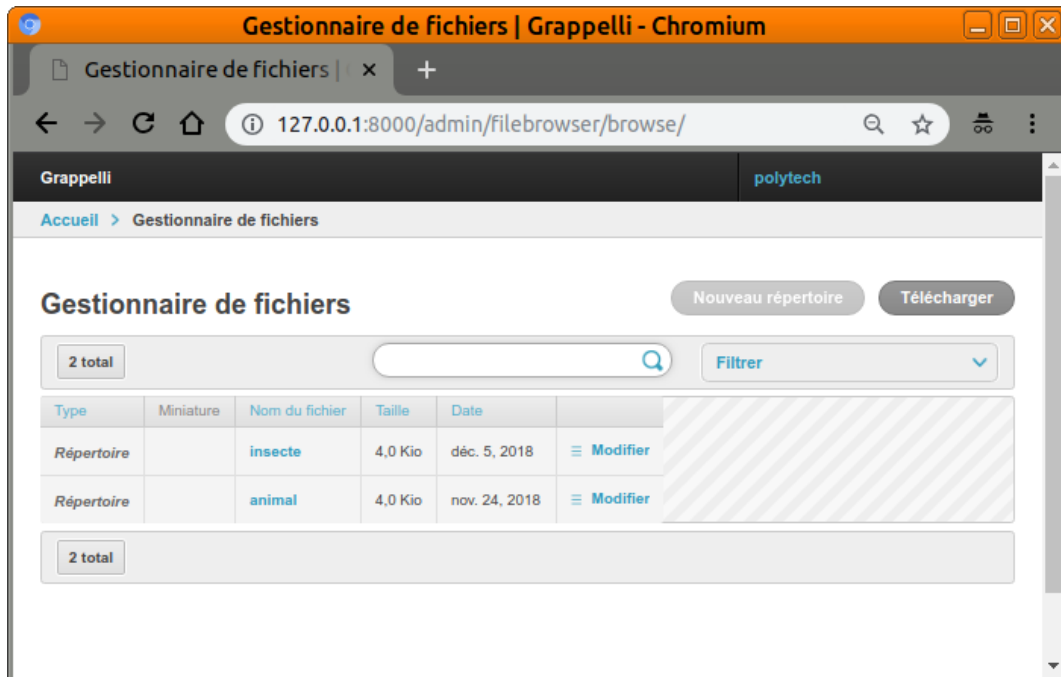


Figure 8 – Exemple d'interface Django FileBrowser

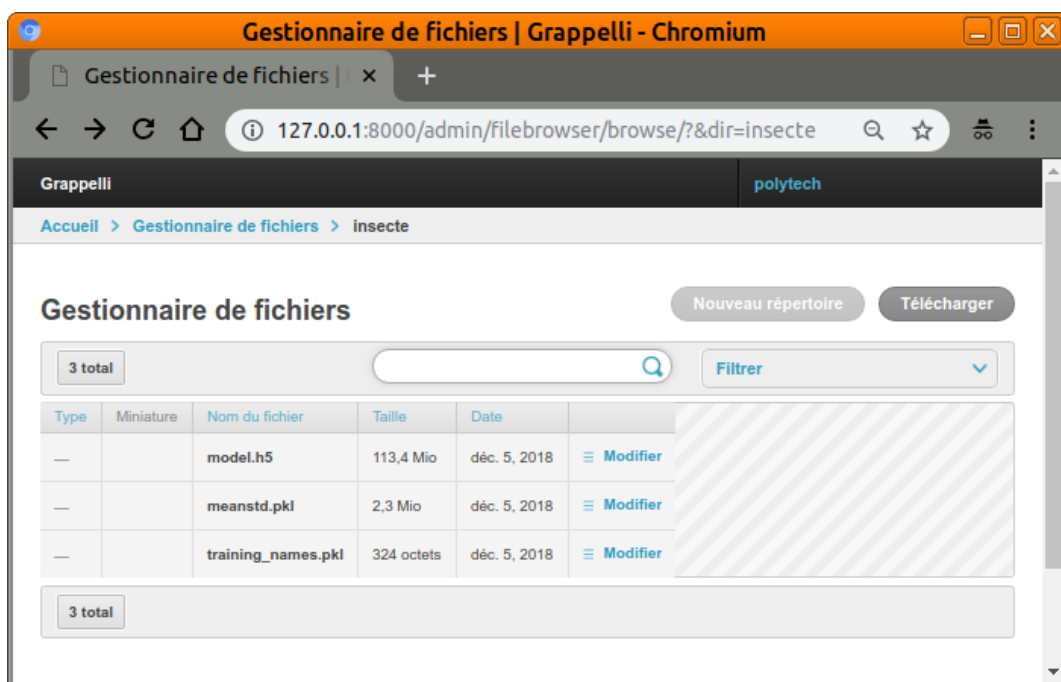


Figure 9 – Exemple d'interface Django FileBrowser



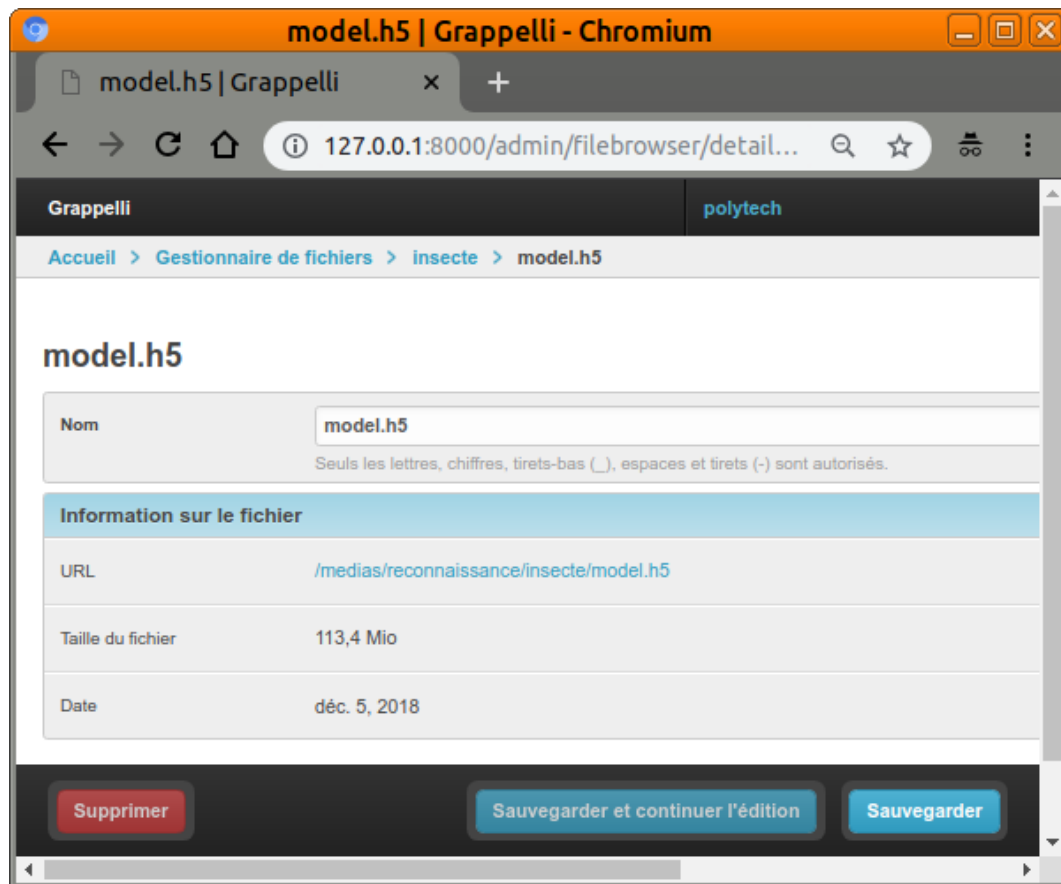


Figure 10 – Exemple d'interface Django FileBrowser

## 9 Ajustement veille techno API REST

Je n'ai finalement pas utilisé API Flask comme convenu dans les recherches.

J'ai fait mon premier prototype de gestion des modèles ainsi qu'un prototype API avec Python Flask.

La gestion des modèles se trouve sur l'application web de contexte (Django), de ce fait, les modèles qu'on importe depuis l'interface web se regroupent dans un objet Django Model. Ce qui permet d'avoir une liaison entre une catégorie de reconnaissance avec son algorithme de reconnaissance ainsi que ces fichiers réseau neurones.

Ces objets Django sont alors stockés dans la base de données SQLite de Django.

Sur ma première version de prototype API Flask, je n'avais pas encore pris en compte cette gestion de modèle, je partais sur quelque chose de simpliste. C'est-à-dire qu'en entrée de ma requête POST API REST, je prenais un nom de catégorie de reconnaissance (exemple : insecte). Puis je définissais un endroit "/dossierDeReconnaissance" et dans ce dossier on avait des sous-dossiers qui correspondaient à chaque catégorie de reconnaissance. Par exemple /dossierDeReconnaissance/insecte/.../dossierDeReconnaissance/plante/...et l'on retrouvait les 3 fichiers (meanstd.pkl, training\_names.pkl, model.h5).

Mais je n'ai pas pu continuer sur cette voie, car lorsque j'ai fait mon prototype de gestion des modèles avec l'import par l'interface. J'ai trouvé uniquement la solution d'importer avec un nom aléatoire en suffixe pour éviter les doublons d'import. C'est une façon de faire sur Django pour simplifier avec des modèles par le biais de formulaire.

Je devais alors réadapter le code de mon premier prototype d'API REST. Je n'ai pas pu continuer avec Python Flask car j'avais le besoin d'accéder aux objets de modèle qui étaient présent sur la base de données Django. De ce fait je me suis rapproché sur une autre veille technologique qui m'a permis de faire ce que je souhaitais et directement avec le framework Django API REST.

Grâce à ça je suis directement lié au projet Django, ce qui me permet de profiter de la base de données et des objets modèles de reconnaissance que j'ai pu importer par l'interface web.

## 10 Mode opératoire API reconnaissance

---

# Mode opératoire

Utilisation API Reconnaissance

Pierrick BOBET

Version 1.0

---

## Introduction

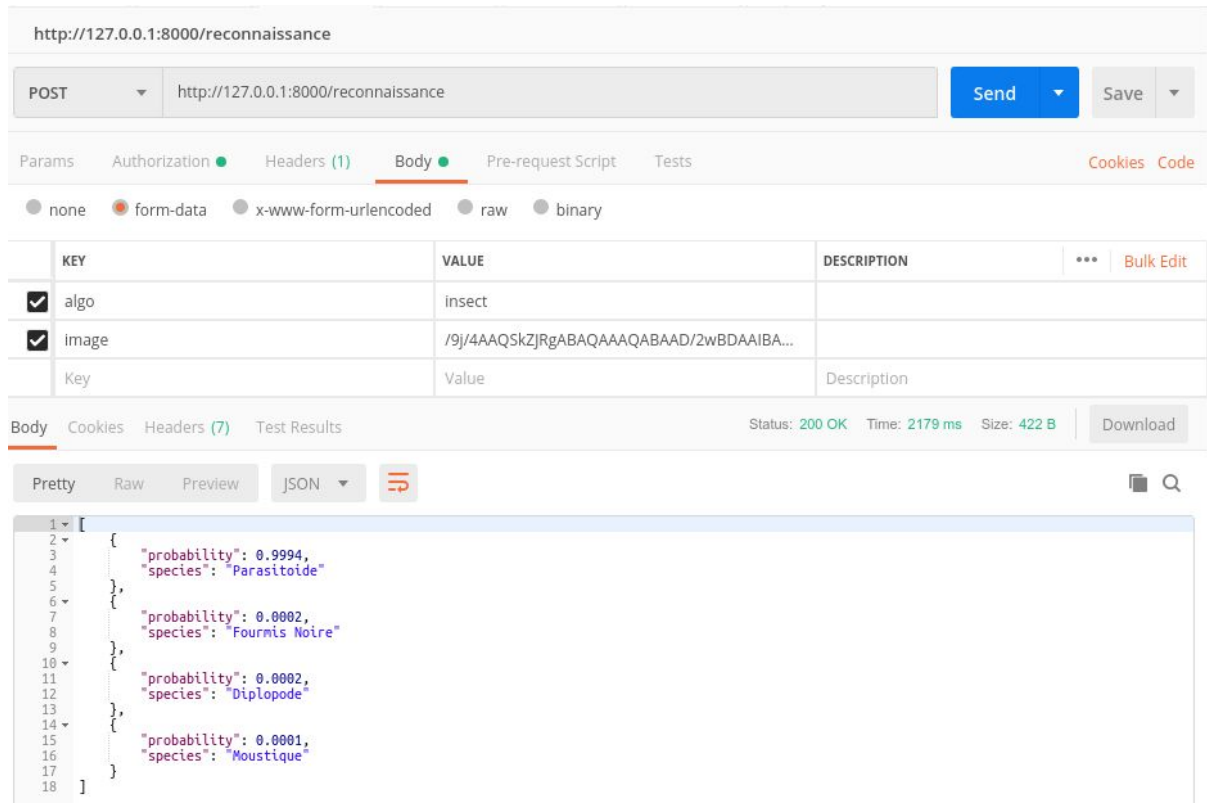
Ce mode opératoire a pour but d'expliquer comment utiliser l'API de reconnaissance.

## Commandes

	Requête API REST Reconnaissance	
Type	GET	POST
URI	http://.../algos	http://.../reconnaissance
Header	"Authorization" : "Bearer mon_token"	
Body		"algo" : "mon_algo"
		"image" : "mon_img_base64"

# Exemple d'utilisation avec Postman

Requête de reconnaissance avec une image en base64 :

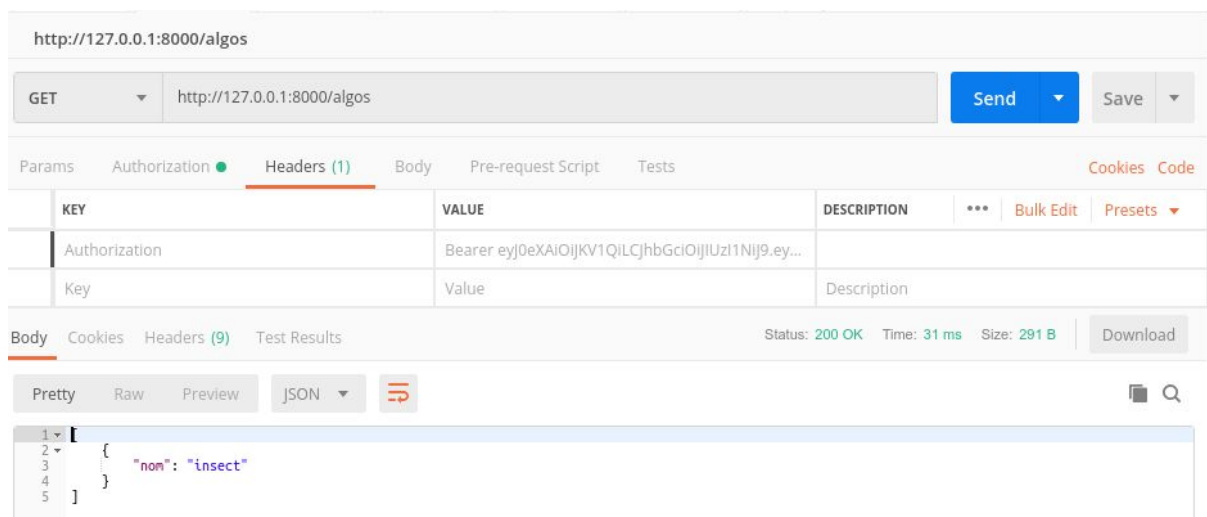


Postman interface showing a POST request to `http://127.0.0.1:8000/reconnaissance`. The request body is a JSON array of objects, each containing a probability and a species name. The response is a JSON array of objects, each containing a probability and a species name.

KEY	VALUE	DESCRIPTION
algo	insect	
image	/9j/4AAQSkZJRgABAQAAQABAAAD/ZwBDAAI...	
Key	Value	Description

```
[{"probability": 0.9994, "species": "Parasitoide"}, {"probability": 0.0002, "species": "Fourmis Noire"}, {"probability": 0.0002, "species": "Diplopode"}, {"probability": 0.0001, "species": "Moustique"}]
```

Requête de récupération des modèles de reconnaissance disponible :



Postman interface showing a GET request to `http://127.0.0.1:8000/algos`. The request headers include `Authorization` and `Key`. The response is a JSON array of objects, each containing a `nom` and a `species` name.

KEY	VALUE	DESCRIPTION
Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ...	
Key	Value	Description

```
[{"nom": "insect"}]
```

## 11 Mode opératoire déploiement continu

---

# Mode opératoire

Mise en place déploiement Django

Pierrick BOBET

Version 1.0

---

## Introduction

Ce mode opératoire a pour but d'expliquer comment configurer Django dans un environnement virtuel Python pour le rendre prêt à déployer. Il contient également la configuration Apache2 avec l'application afin qu'il puisse gérer les demandes des clients directement avant de transmettre les demandes vers Django. Pour ce faire, ce mode opératoire utilise le module Apache2 `mod_wsgi` qui peut communiquer avec Django via la spécification d'interface WSGI.

## Pré-requis

Avant de commencer, ce mode opératoire se base sur une version Ubuntu 18.04 avec un serveur Apache2 et MySQL. Un utilisateur affecté au groupe `sudo` doit être créé et il sera utilisé dans le reste de ce document.

Une fois que l'application sera opérationnelle, nous configurons Apache pour l'interface avec l'application Django. Cela se fera avec le module Apache `mod_wsgi`, qui peut traduire les requêtes HTTP dans un format d'application défini par une spécification appelée WSGI.



## Installation des pré-requis

Pour commencer, nous téléchargeons et installerons tous les éléments nécessaires dans les dépôts référentiels Ubuntu. Cela inclura le serveur Web Apache, le module `mod_wsgi` utilisé pour l'interface avec notre application Django, et `pip`, le gestionnaire de paquets Python qui peut être utilisé pour télécharger nos outils liés à Python.

Pour obtenir tout ce dont nous avons besoin, mettez à jour l'index de paquet local de votre serveur, puis installez les paquetages appropriés.

Les commandes suivantes permettent d'installer l'ensemble des pré-requis :

- Apache2
- MySQL
- Python PIP
- WSGI

```
sudo apt-get update
```

```
sudo apt-get -y install python-pip python3-pip apache2  
libapache2-mod-wsgi-py3 apache2 php libapache2-mod-php  
mysql-server php-mysql python-dev python3-dev  
libmysqlclient-dev virtualenv
```

Maintenant que nous avons les composants des référentiels Ubuntu, nous pouvons commencer à travailler sur notre projet Django.

## [Installation automatique]

Cette section s'effectue directement depuis un poste développeur. Il suffit de cloner le projet sur le gestionnaire de sources. Puis de se rendre dans le répertoire **.../InsectPlatform/automated-deployments/prod**

Installation de l'outil Fabric3 Python :

```
pip3 install fabric3  
pip3 install fabric3-virtualenv
```

Et ainsi d'exécuter le script de la façon suivante :

```
fab deploy
```

Le script permet d'effectuer le déploiement continu sur le serveur pas à pas.

## [Installation manuelle]

### Configurer environnement virtuel Python

La première étape consiste à créer un environnement virtuel Python afin que notre projet Django soit séparé des outils système et de tout autre projet Python sur lequel nous pourrions travailler. Nous devons installer la commande virtualenv pour créer ces environnements. Nous pouvons obtenir ce paquet en utilisant pip :

```
sudo pip3 install virtualenv
```

Avec virtualenv installé, nous pouvons commencer à former notre projet. Créez un répertoire dans lequel vous souhaitez conserver votre projet et accédez au répertoire suivant:

```
mkdir ~/reconnaissance
```

Maintenant, nous pouvons importer l'ensemble des fichiers du projet dans ce dossier.

```
cd ~/reconnaissance
```

Dans le répertoire du projet, créez un environnement virtuel Python en tapant:

```
virtualenv reconnaissanceenv
```

Cela créera un répertoire appelé **reconnaissanceenv** dans votre répertoire **reconnaissance**. À l'intérieur, il installera une version locale de Python et une version locale de pip. Nous pouvons l'utiliser pour installer et configurer un environnement Python isolé pour notre projet.

Avant d'installer les exigences Python de notre projet, nous devons activer l'environnement virtuel. Vous pouvez le faire en tapant:

```
source ~/reconnaissance/reconnaissanceenv/bin/activate
```

Votre invite devrait changer pour indiquer que vous travaillez maintenant dans un environnement virtuel Python. Cela ressemblera à quelque chose comme ceci: utilisateur

```
(reconnaissanceenv)user@host:~/reconnaissance$
```

Avec votre environnement virtuel actif, installez Django avec l'instance locale de pip:

Remarque: les environnements virtuels utilisent leur propre version de Python et des outils associés. Lorsque l'environnement virtuel est activé, vous devez utiliser la commande pip3 :

```
(reconnaissanceenv) $ pip3 install -r requirements.txt
```

Cela installera les pré-requis du projet Django dans votre environnement virtuel Python.

Nous allons nous concentrer sur la configuration des hôtes autorisés afin de restreindre les domaines auxquels nous répondons et sur la configuration du répertoire de fichiers statiques, où Django placera les fichiers statiques afin que le serveur Web puisse les gérer facilement.

Commencez par trouver la ligne `ALLOWED_HOSTS`. Entre crochets, entrez l'adresse IP publique de votre serveur, son nom de domaine ou les deux. Chaque valeur doit être entourée de guillemets et séparée par une virgule, comme une liste Python normale. C'est une bonne idée d'ajouter des adresses locales comme `127.0.0.1` et `127.0.1.1` :

```
vi
~/reconnaissance/prd_di5_devappreconnobj/InsectPlatform/I
nsectPlatform/settings.py
```

```
...
ALLOWED_HOSTS = ["adresse ip serveur", "127.0.0.1",
"127.0.1.1"]
...
```

## Configuration base de données

Maintenant, nous pouvons migrer le schéma de base de données initial vers notre base de données SQLite en utilisant le script de gestion:

```
cd ~/reconnaissance/InsectPlatform
```

```
chmod +x manage.py
```

```
chown -R :web ~/reconnaissance/
```

```
chmod -R 77  
~/reconnaissance/InsectPlatform/mainapp/medias
```

La prochaine étape consiste à configurer les identifiants de la base de données MySQL :

```
vi  
~/reconnaissance/prd_di5_devappreconnobj/InsectPlatform/I  
nsectPlatform/settings.py
```

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'changer BD ex : django',  
        'USER': 'changer ID',  
        'PASSWORD': 'changer mot de passe',  
        'HOST': 'localhost',  
    }  
}
```

Le serveur va être prochainement en production, il faut donc retirer toute information sensible. Pour ce faire, les logs sont des données sensibles, il faut donc les désactiver pour l'utilisateur (page HTML). Dans le fichier settings.py, nous pouvons passer l'attribut DEBUG à false :

```
...  
DEBUG = False  
...
```

Il faut également placer la **SECRET\_KEY** dans un fichier situé dans : /etc/secret\_key.txt

Ensuite il est nécessaire de convertir le modèle de données Django sous un format MySQL, de ce fait les deux commandes suivantes permettent de migrer le modèle Django :

```
python3 manage.py makemigrations
```

```
python3 manage.py migrate
```

Il est nécessaire de récupérer l'ensemble des fichiers statiques permettant le bon fonctionnement de Django. La commande suivante permet de collecter l'ensemble de ces fichiers :

```
python3 manage.py collectstatic
```

D'autre part, il est nécessaire de configurer l'interaction Apache WSGI avec le fichier wsgi.py qui a été généré par Django :

```
import os
import sys

from django.core.wsgi import get_wsgi_application

from InsectPlatform import settings

path = settings.BASE_DIR #
'/var/www/prd_di5_devappreconnobj/InsectPlatform'
if path not in sys.path:
    sys.path.append(path)

os.environ.setdefault("DJANGO_SETTINGS_MODULE",
"InsectPlatform.settings")

application = get_wsgi_application()
```

La dernière étape consiste à configurer la partie Apache2 du serveur. Il s'agit de configurer le VirtualHost du port 80 de la manière suivante :

```
<VirtualHost *:80>
...

    # Django Part
```

```

Alias /static /home/pbobet/reconnaissance/InsectPlatform/mainapp/static
Alias /medias /home/pbobet/reconnaissance/InsectPlatform/mainapp/medias
Alias /mainapp /home/pbobet/reconnaissance/InsectPlatform/mainapp
Alias /image_download
/home/pbobet/reconnaissance/InsectPlatform/mainapp/medias/image_download

<Directory /home/pbobet/reconnaissance/InsectPlatform/mainapp/static>
    Options Indexes FollowSymLinks Includes ExecCGI
    AllowOverride All
    Require all granted
</Directory>

<Directory /home/pbobet/reconnaissance/InsectPlatform/mainapp/medias>
    Options Indexes FollowSymLinks Includes ExecCGI
    AllowOverride All
    Require all granted
</Directory>

<Directory /home/pbobet/reconnaissance/InsectPlatform/InsectPlatform>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>

WSGIDaemonProcess myproject
python-home=/home/pbobet/reconnaissance/reconnaissanceenv
python-path=/home/pbobet/reconnaissance/InsectPlatform
WSGIProcessGroup myproject
WSGIScriptAlias /
/home/pbobet/reconnaissance/InsectPlatform/InsectPlatform/wsgi.py

SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1

</VirtualHost>

```

## Sources

- <https://docs.djangoproject.com/fr/2.1/howto/deployment/wsgi/>
- <https://openclassrooms.com/fr/courses/1871271-developpez-votre-site-web-avec-le-framework-django/1874623-deployer-votre-application-en-production>
- <http://www.formation-django.fr/framework-django/zoom-sur/deploiement-application-django.html>
- [https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod\\_wsgi-on-debian-8](https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod_wsgi-on-debian-8)



## 12 Plan de tests

---

# Plan des tests

Plate-forme de reconnaissance

Pierrick BOBET

Version 1.0

---

## Introduction

Ce document est un descriptif des plans de tests mis en oeuvre pour la plate-forme de reconnaissance. Il s'agit du document rédigé en partie par l'ancien étudiant de ce projet Pierre JUILLE ainsi que Pierrick BOBET. Les anciens tests ainsi que les nouveaux réalisés lors de ce projet de fin d'étude y sont présent.

## Avant Propos

Il est fortement conseillé de lire attentivement la notice de Django concernant les tests et d'avoir lu de manière générale le fonctionnement de l'application ainsi que la documentation d'installation et le cahier du développeur.

Certains tests sont déjà implémentés par le framework et ne sont pas refaits explicitement dans les fichiers de tests.

Les tests peuvent nécessiter au préalable une migration de la base de données. Il suffit de saisir les deux commandes suivantes :

```
python3 manage.py makemigrations
```

```
python3 manage.py migrate
```

## Où trouver les tests ?

Tous les tests sont répartis par type de tests dans des fichiers stockés au niveau du dossier Tests de chaque application.

Concernant notre application mainapp, le dossier Tests contient 2 fichiers:

- test\_api.py : permet de tester l'api de reconnaissance mis en place ainsi que l'authentification par Token
- test\_form.py : permet de tester les formulaires de l'application mainapp .
- test\_pages.py : permet de tester les pages web et notamment les redirections au niveau des contrôleurs.

## Fonctionnalités testées explicitement :

- test\_api.py :
  - test\_api\_auth\_token : Fonctionnement d'une requête API GET vers l'API d'authentification qui a été conçu par un autre groupe d'étudiant. Permet de tester l'API authentification pour la vérification de token.
    - requête : GET, chemin : "/check"
    - réponse attendu : Retourne une bonne exécution avec un code 202

Attention ce test nécessite un token qui soit au préalable valide.
  - test\_api\_reconnaissance\_exception : Tester un appel à l'API de reconnaissance avec exception et :
    - paramètres :
      - un token d'authentification valide
      - un modèle de reconnaissance d'insecte
      - une image de Parasitoïde codée en base64
      - sans modèle de reconnaissance dans la base de données
    - requête : POST, chemin : "/reconnaissance"
    - réponse attendu : Retourne une erreur 404

Attention ce test nécessite un token qui soit au préalable valide.
  - test\_api\_reconnaissance\_get\_result : tester un appel à l'API de reconnaissance avec :
    - paramètres :
      - un token d'authentification valide
      - un modèle de reconnaissance d'insecte
      - une image de Parasitoïde codée en base64
    - requête : POST, chemin : "/reconnaissance"
    - réponse attendu : Retour la liste de résultat avec le Parasitoïde avec une probabilité de 99% et d'autre sous résultat.

Attention ce test nécessite un token qui soit au préalable valide.
  - test\_api\_reconnaissance\_get\_algos : Permet de tester un appel à l'API de reconnaissance pour récupérer la liste des modèles de reconnaissance avec :
    - paramètre :

- un token d'authentification valide
  - requête : GET, chemin : “/algos”
  - réponse attendu : Retourne la liste des modèles de reconnaissance disponible.

Attention ce test nécessite un token qui soit au préalable valide.

- test\_forms.py : Fonctionnement du formulaire UploadImageForm dans le cas d'une image classique, pour vérifier sa cohérence avec le modèle. Le test vérifie que `form.is_valid == true` .
- test\_pages.py : Redirection correcte dans les cas :
  - requête : GET, chemin : “/”
  - requête : GET, chemin : “/image”
  - requête : POST, chemin : “/image”, contenu : {‘test’: ‘test’}

La réponse générée doit être 200.

Attention cependant, le dernier test n'est possible que dans ces conditions. D'une api extérieure à l'environnement de test, la requête est rejetée notamment pour un problème de token CSRF (comportement normal, voulu, et testable grâce à des clients comme POSTMAN)







































## Lancer les tests :

Si on souhaite lancer ces tests, assurez vous d'être placé dans le répertoire de base du projet **InsectPlatform/** au niveau du fichier `manage.py`. Puis lancer cette commande :

```
> python manage.py test mainapp.tests
```

Ceci devrait lancer tous vos tests concernant l'application `mainapp`. Si vous avez ajouté votre propre application avec la même structure, la commande restera la même en remplaçant `mainapp.tests` par `LeNomDeVotreApp.tests`.

## 13 Organisation et Diagramme GanTT

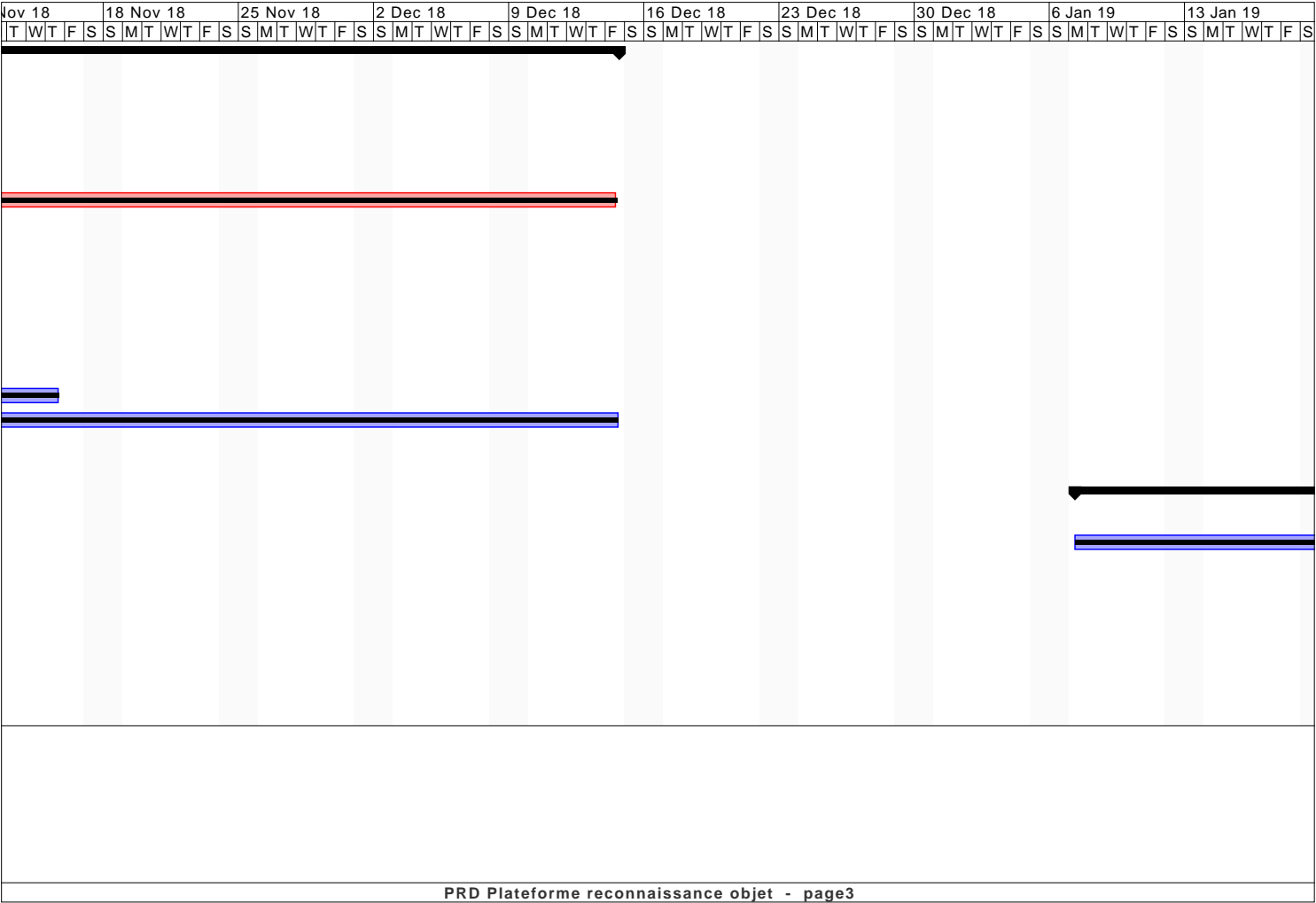
		Name	Duration	Start	Finish	Predecessors
1		<b>Phase d'analyse</b>	<b>63 days?</b>	<b>9/19/18 8:00 AM</b>	<b>12/14/18 5:00 PM</b>	
2		<b>Appropriation du projet</b>	<b>18 days</b>	<b>9/19/18 8:00 AM</b>	<b>10/12/18 5:00 PM</b>	
3		Gestion de projet	2 days	9/19/18 8:00 AM	9/20/18 5:00 PM	
4		Découverte de l'existant + test	13 days	9/26/18 8:00 AM	10/12/18 5:00 PM	3
5	 	Veille - formation API REST	8 days?	10/9/18 8:00 AM	10/18/18 5:00 PM	4
6	 	Intégration continue	7.125 days?	10/15/18 8:00 AM	10/24/18 9:00 AM	5
7	 	Cahier de spécification + rapport PRD	62.75 days?	9/19/18 8:00 AM	12/14/18 3:00 PM	
8		<b>Veille Framework API</b>	<b>6.125 days?</b>	<b>10/18/18 2:00 PM</b>	<b>10/26/18 3:00 PM</b>	
9	 	Veille Framework TastyPie	6 days?	10/18/18 2:00 PM	10/26/18 2:00 PM	
10	 	Veille Framework Django API REST	6.125 days?	10/18/18 2:00 PM	10/26/18 3:00 PM	
11		Veille déploiement continu	6 days?	10/29/18 8:00 AM	11/5/18 5:00 PM	
12		Veille gestion modèles admin	7 days?	11/7/18 8:00 AM	11/15/18 5:00 PM	
13		Rapport PRD partie 1	63 days?	9/19/18 8:00 AM	12/14/18 5:00 PM	
14		<b>Phase développement</b>	<b>66.625 days?</b>	<b>1/7/19 8:00 AM</b>	<b>4/9/19 2:00 PM</b>	
15	 	Rapport PRD partie 2	41.25 days?	2/11/19 8:00 AM	4/9/19 10:00 AM	
16	 	Implémentation API reconnaissance	24.375 days?	1/7/19 8:00 AM	2/8/19 11:00 AM	
17	 	Livable prototype API reconnaissance	0 days	2/11/19 8:00 AM	2/11/19 8:00 AM	
18	 	Implémentation gestion modèles admin	12 days?	2/11/19 8:00 AM	2/26/19 5:00 PM	
19	 	Livable prototype gestion modèles	0 days	2/27/19 8:00 AM	2/27/19 8:00 AM	
20	 	Implémentation déploiement continu	20 days?	2/27/19 8:00 AM	3/26/19 5:00 PM	
21	 	Livable prototype déploiement continu	0 days	3/27/19 8:00 AM	3/27/19 8:00 AM	
22	 	Rédaction documentations	10 days?	3/26/19 2:00 PM	4/9/19 2:00 PM	
23	 	Livable final documentation + 3 fonctionn...	0 days	4/9/19 11:00 AM	4/9/19 11:00 AM	

PRD Plateforme reconnaissance obiet - page1

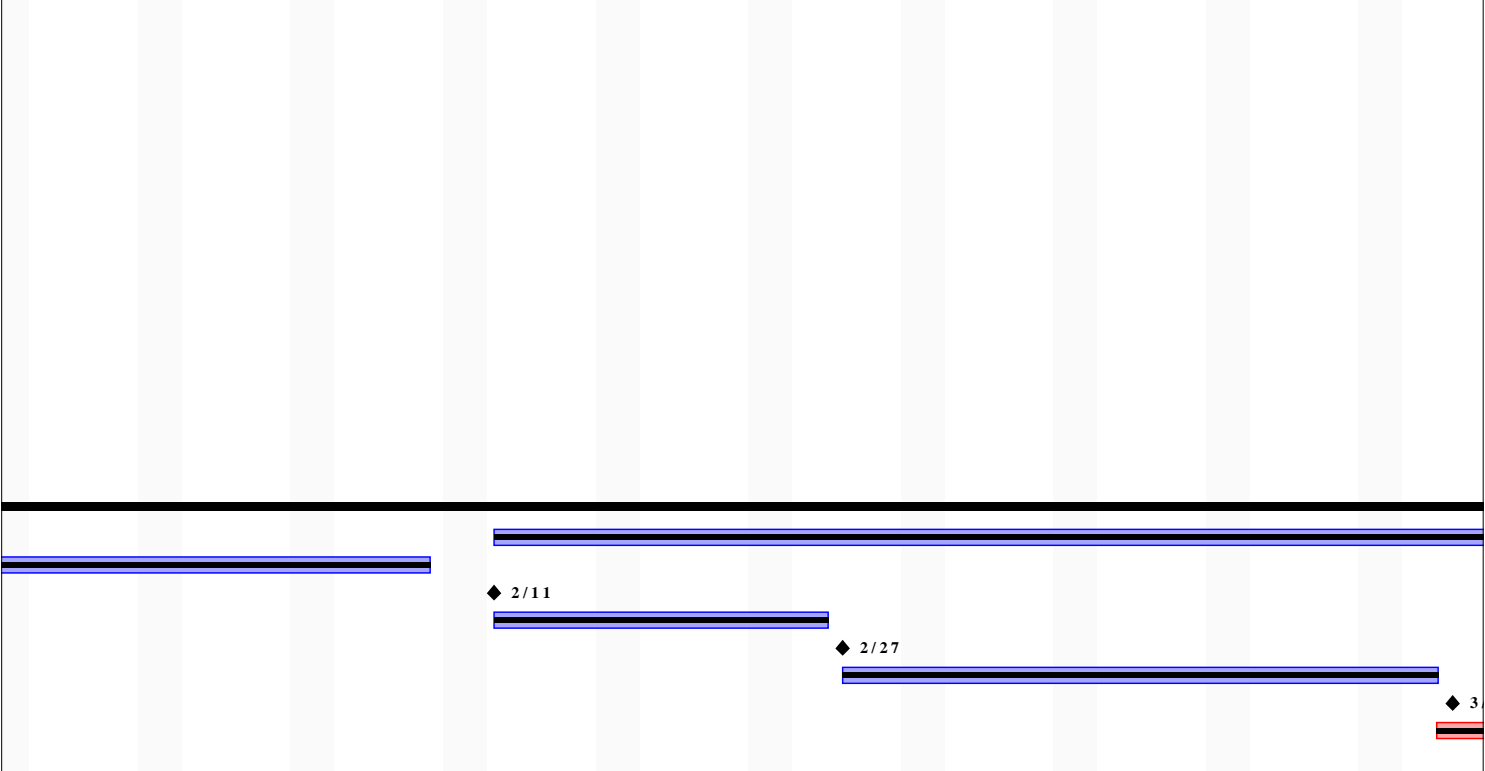
Resource Names	16 Sep 18					23 Sep 18					30 Sep 18					7 Oct 18					14 Oct 18					21 Oct 18					28 Oct 18					4 Nov 18					11 N		
	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M						

PRD Plateforme reconnaissance objet - page2





20 Jan 19	27 Jan 19	3 Feb 19	10 Feb 19	17 Feb 19	24 Feb 19	3 Mar 19	10 Mar 19	17 Mar 19	24 Mar 19
S   M   T   W   T   F   S	S   M   T   W   T   F   S	S   M   T   W   T   F   S	S   M   T   W   T   F   S	S   M   T   W   T   F   S	S   M   T   W   T   F   S	S   M   T   W   T   F   S	S   M   T   W   T   F   S	S   M   T   W   T   F   S	S   M   T   W   T   F   S



[illegible]

# Comptes rendus hebdomadaires

## Compte rendu n°1 du 23/09/2018

Il s'agit de la semaine d'introduction au PRD.

- Prise en main, installation et configuration de l'application existante
- Mise en place d'une gestion de projet centralisé sur GitLab avec des issues, temps estimé, temps réel passé, gestion sous forme tableau Kanban
- Début de mise en place d'intégration continue. Il faut se pencher sur le point du déploiement Staging et Production
- Phase de compréhension du code et d'analyse des besoins.
- Recensement des besoins dans le cahier de spécification + rédaction du contexte

## Compte rendu n°2 du 30/09/2018

Rédaction cahier des spécifications :

- Introduction : Acteurs, en-jeux et contexte, Objectifs, Bases méthodologiques
- Description générale : Environnement du projet, Fonctionnalités du système

Veille technologique :

- Sur les possibilités d'un système d'authentification avec le Framework Django. Expérimentation des solutions trouvées et ajout dans le rapport PRD. Création d'une branche!1 avec une première solution expérimentale
- Sur les possibilités d'un système de paiement avec Paypal. Expérimentation des solutions trouvées. Création d'une branche!2 avec une première solution expérimentale

## Compte rendu n°3 du 07/10/2018

Restructuration du PRD par l'encadrant.

Les besoins énoncés ne sont plus des tâches à accomplir. Ces tâches seront accomplies par le groupe de projet SI DI5.

Description du nouveau besoin, soit la création d'un API REST concernant l'application de reconnaissance d'objet.

Premier pas de veille technologique :

- Suivi du cours OpenClassRoom "Utilisez des API REST dans vos projets web"

Mise en place d'une intégration continue :

- Build du projet (installation dépendances)
- Prise en charge des tests précédemment développés

#### Compte rendu n°4 du 14/10/2018

Rédaction cahier de spécification :

- Objectifs
- Base méthodologiques
- Gestion de projet
- Environnement du projet
- Outils à disposition
- Schémas de l'environnement
- Caractéristiques des utilisateurs
- État de l'art / veille
- Formations
- Découverte de API REST

Début d'une veille technologique sur le Framework Django REST API. Premier pas de mise en place, installation avec exemple de la documentation officielle.

Source : <https://www.django-rest-framework.org>

#### Compte rendu n°5 du 21/10/2018

Rédaction des fonctionnalités du système, création d'un diagramme de cas d'utilisation.

Mise en place d'un environnement de test pour l'installation d'un API Django REST. Suivi de la documentation concernant un exemple de Django REST : <https://www.django-rest-framework.org/tutorial/quickstart/>

Rédaction cahier de spécification : État de l'art / veille

Phase de recherche et début analyse d'un autre Framework Tastypie Python API

#### Compte rendu n°6 du 28/10/2018

Veille technologique sur la solution API TastyPie

Rédaction cahier de spécifications :

Description des interfaces externes du logiciel

Rencontre avec Maxime MARTINEAU pour éclaircir des spécifications :

Concevoir un API permettant de faire la reconnaissance d'objet. 1 méthode POST qui prendra en entrée une image et en sortie le résultat des probabilités de reconnaissance. Prévoir une authentification pour l'accès API.

Ajouter des fonctionnalités pour la partie administration : importer 3 fichiers qui représentent le réseau de neurone, la liste des classes d'objet (exemple : moustique, abeille...), l'ordre statistique des images.

Restreindre l'utilisation de l'API à l'environnement du projet, le monde extérieur (internet) ne peut pas interagir avec l'application dans un premier temps.

Rédaction des contraintes : langage de développement, accès réseaux API, clé privé/public

Premier pas dans la veille technologique concernant le déploiement continue en production.  
Premier test dans un environnement virtuel.

#### **Compte rendu n°7 du 04/11/2018**

Rédaction cahier de spécification partie :

- Environnement du projet
- Caractéristiques des utilisateurs
- Utilisateurs de l'application en général
- spécifications fonctionnelles et diagramme Cas d'utilisation

Rédaction rapport PRD partie :

- Etat de l'art
- Analyse API
- Analyse Déploiement

Veille sur le déploiement, mise en place d'un docker pour simuler le serveur de production.

Tentative de mise en place du déploiement continu par le biais du module WSGI avec Apache2. La tentative n'a pas eu de réussite, des erreurs apparaissent au moment de se rendre sur la page d'accueil du site (Erreur 500).

Création d'un GanTT avec toutes les tâches affectées et estimées.

Veille technologique sur l'ajout de fonctionnalités dans la partie administration de la plateforme.

#### **Compte rendu n°8 du 11/11/2018**

Veille technologique sur l'import/export concernant les modèles pour la reconnaissance.

Mise en place dans une branche pour tester la bibliothèque Django FileBrowser. Cette bibliothèque a permis de consulter/modifier/supprimer des fichiers dans un répertoire défini à l'avance. L'interface est représentée par une page qui permet de parcourir et faire les actions précédemment citées.

Mise en place dans une branche pour tester la bibliothèque FileSystemStorage qui permet simplement de mettre en place un import de fichier en l'intégrant à une page HTML au choix.

Rédaction de la partie veille concernant l'import/export concernant les modèles pour la reconnaissance.

Recherche veille technologique pour la partie déploiement en production. Mise en place d'un déploiement continu dans un environnement virtuel Docker afin d'essayer un didacticiel. La solution utilise le module WGI de Apache2.

Rédaction de la partie veille concernant la partie déploiement.

Début rédaction mode opératoire déploiement continu.

**Compte rendu n°9 du 18/11/2018**

Conception de diagramme de cas d'utilisation.

Rédaction du cahier de spécifications.

Veille technologique sur import/export fichier Django.

Discussion avec les étudiants SI pour comprendre l'interaction entre les API de chacun.

Conception diaporama présentation PRD.

Prototypage en mettant en place une première solution avec API REST. Récupère une image en entrée et la stocke. Analyse l'image en appelant un algorithme choisi en entrée. Ressort sous forme de JSON le résultat.

**Compte rendu n°10 du 25/11/2018**

Réunion avec Robin Maréchal pour discuter de l'interaction entre nos composants (API).

Réunion bilan cahier de spécification avec Maxime MARTINEAU.

Veille technologique sur l'import d'un fichier encapsulé par le biais de l'API REST.

Veille technologique sur API Python Flask avec liaison de l'algorithme de reconnaissance d'objet.

Rédaction de cahier de spécifications.

**Compte rendu n°11 du 02/12/2018**

Veille technologique sur l'import d'un fichier encapsulé par le biais de l'API REST.

Veille technologique sur API Python Flask avec liaison de l'algorithme de reconnaissance d'objet.

Rédaction de cahier de spécifications.

**Compte rendu n°12 du 09/12/2018**

Continuité du prototype import/export

Continuité du prototype déploiement continu

**Compte rendu n°13 du 06/01/2019**

Conception d'un prototype d'importation de nouveau modèle ainsi que la gestion de suppression des modèles déjà existant.

Merge du premier prototype API REST avec le premier prototype de gestion de modèle.

Je n'ai finalement pas utilisé API Flask comme convenu dans les recherches.

J'ai fait mon premier prototype de gestion des modèles ainsi qu'un prototype API avec Python Flask. La gestion des modèles se trouve sur l'application web de contexte (Django), de ce fait, les modèles qu'on importe depuis l'interface web se regroupent dans un objet Django Model. Ce qui permet d'avoir une liaison entre une catégorie de reconnaissance avec son algorithme de reconnaissance ainsi que ces fichiers réseau neurones.

Ces objets Django sont alors stockés dans la base de données SQLite de Django. Sur ma première version de prototype API Flask, je n'avais pas encore pris en compte cette gestion de modèle, je partais sur quelque chose de simpliste. C'est-à-dire qu'en entrée de ma requête POST API REST, je prenais un nom de catégorie de reconnaissance (exemple : insecte). Puis je définissais un endroit /dossierDeReconnaissance et dans ce dossier on avait des sous-dossiers qui correspondaient à chaque catégorie de reconnaissance. Par exemple /dossierDeReconnaissance/insecte/.../dossierDeReconnaissance/plante/... et on retrouvait les 3 fichiers (meanstd.pkl, training\_names.pkl, model.h5).

Mais je n'ai pas pu continuer sur cette voie, car lorsque j'ai fait mon prototype de gestion des modèles avec l'import par l'interface. J'ai trouvé uniquement la solution d'importer avec un nom aléatoire en suffixe pour éviter les doublons d'import. C'est une façon de faire sur Django pour simplifier avec des modèles par le biais de formulaire.

Je devais alors réadapter le code de mon premier prototype d'API REST. Je n'ai pas pu continuer avec Python Flask car j'avais le besoin d'accéder aux objets de modèle qui étaient présents sur la base de données Django. De ce fait je me suis rapproché sur une autre veille technologique qui m'a permis de faire ce que je souhaitais et directement avec le framework Django API REST.

Grâce à ça je suis directement lié au projet Django, ce qui me permet de profiter de la base de données et des objets modèles de reconnaissance que j'ai pu importer par l'interface web.

#### Compte rendu n°14 du 13/01/2019

Travail sur :

- Rendre l'interface web générique (ajout d'un champ de sélection choix d'algo de reconnaissance).
- Connecter l'API REST avec les données de la base de données Django (algo de reconnaissance).
- Permettre de récupérer un JSON par le biais d'un API REST contenant la liste des probas.
- Correction d'un bug lors de l'intégration continue.
- Ajustement des tests unitaire par rapport aux modifications précédentes.

#### Compte rendu n°15 du 20/01/2019

- Réunion bilan de gestion de projet avec un intervenant de Sopra Steria. Cette réunion a permis de faire un point sur la gestion de projet. Cela a permis de confirmer les étapes du GanTT. De plus, des conseils m'ont été donnés afin de rendre des livrables pour ma méthodologie Agile.
- J'ai également rencontré Robin Marechal élève de DI5 groupe SI. Cette étudiant est responsable de la partie interaction entre son environnement de projet SI et mon environnement. Nous avons donc échangé pour contrôler la conformité de nos interactions. Réunion bilan avec l'encadrant Maxime Martineau concernant l'avancement du projet. Des ajustements sont à faire sur la modification des modèles de reconnaissance.
- Ajustement de mon code pour l'interaction entre mon API Reconnaissance et la partie SI. (format JSON de sortie)



**Compte rendu n°16 du 27/01/2019**

- Merge des adaptations URI + sortie JSON
- Connecter la partie Authentification du groupe SI avec API de reconnaissance
- Vérifier le token d'authentification et retourner en cas d'interdiction d'accès à l'API Reconnaissance
- Début d'une possibilité de modification des modèles de reconnaissance
- Ajustement de certains tests unitaires
- Mise en place d'un API de reconnaissance qui retourne le résultat en JSON.
- En entrée de l'API reconnaissance : id de l'algorithme à utiliser pour la reconnaissance, image encodée en base64, header Authorization avec un token à vérifier.

**Compte rendu n°17 du 03/02/2019**

- Adaptation de l'API reconnaissance par rapport à son résultat en sortie. format 0.9 pour 90
- Génération de chemin absolu de manière automatique afin de pouvoir mettre en place le déploiement continu. Permet une concordance lors de l'utilisation d'Apache 2 par rapport à l'environnement virtuel Django.
- Ajout d'une interface Django admin pour permettre la modification des modèles de reconnaissance

**Compte rendu n°18 du 10/02/2019**

- Correction bug configuration Apache2. La phase d'authentification et \* contrôle du token ne fonctionnait pas à cause d'une autorisation manquante dans la configuration Apache2 du serveur de production.
- Rédaction mode opératoire de la mise en place en déploiement continu.
- Réunion bilan avancement avec Maxime Martineau
- Ajustement de nommage dans les objets modèles Django

**Compte rendu n°19 du 17/02/2019**

- Correction bugs : associé le résultat sur l'objet image dans la base de données Django. Que ce soit en entrée par requête API REST ou par interface Web.
- Veille techno sur les solutions de déploiement continu
- Mise en place d'un script Python Fabric3. Ce qui permet de mettre en place le déploiement continu

**Compte rendu n°20 du 24/02/2019**

- Correction de bug sur le script de déploiement continu.
- Optimisation et sécurisation du script de déploiement qui cache les données sensibles (mot de passe, identifiant...).
- Début compte-rendu pour la partie gestion de projet avec les intervenants Sopra Steria.
- Contrôle l'accès API requete get algo
- Object imageDownload Ajout champs résultat image
- Restructuration de la page admin

**Compte rendu n°21 du 03/03/2019**

- Ajout d'une couverture de test sur les fonctionnalités développées pendant ce semestre.

- Ajustement de la réponse JSON de l'api de reconnaissance. Permet d'avoir une réponse toujours dans le même ordre et dans la même forme.
- Rédaction rapport de PRD
- Rédaction mode opératoire déploiement continu
- Rédaction mode opératoire API REST reconnaissance

**Compte rendu n°22 du 10/03/2019**

- Conception posterblock
- Conception résumé FR et EN
- Rédaction d'un document plan de tests
- Ajout de documentation de code
- Rédaction d'un document mode opératoire déploiement continu
- Conception et rédaction rapport gestion projet Sopra Steria
- Conception diagramme cas d'utilisation pour résumer le travail réalisé
- Conception diagramme de séquence pour résumer le travail réalisé

**Compte rendu n°23 du 17/03/2019**

- Ajout de nouveaux tests fonctionnels
- Génération d'une documentation sous format docstring Sphinx avec prise en charge HTML
- Rédaction de documentation de code
- Conception diagramme UML de séquence pour illustrer la réalisation
- Rédaction de la partie mise en oeuvre du rapport PRD
- Ajout de l'ensemble des documentations dans le rapport de PRD

**Compte rendu n°24 du 24/03/2019**

- Finalisation documentation interne. Génération de documentation HTML grâce à Django Admindocs et Sphinx.
- Rédaction rapport PRD.
- Correctif API REST requête get Algo
- Conception présentation orale.

# Webographie

- [1] *Django avec Apache et mod\_wsgi | Django documentation | Django*. URL : <https://docs.djangoproject.com/fr/2.1/howto/deployment/wsgi/modwsgi/> (visité le 08/11/2018).
- [2] *Django REST framework*. URL : <https://www.django-rest-framework.org/> (visité le 11/10/2018).
- [3] Maxime LORANT et Mathieu XHONNEUX. *Déployer votre application en production*. OpenClassrooms. 31 oct. 2018. URL : <https://openclassrooms.com/fr/courses/1871271-developpez-votre-site-web-avec-le-framework-django/1874623-deployer-votre-application-en-production> (visité le 31/10/2018).
- [4] *Que signifie Déploiement continu? - Definition IT de Whatis.fr*. LeMagIT. URL : <https://www.lemagit.fr/definition/Deploiement-continu> (visité le 31/10/2018).
- [5] *Quickstart - Django REST framework*. URL : <https://www.django-rest-framework.org/tutorial/quickstart/> (visité le 18/10/2018).
- [6] Emily REESE. *Utilisez des API REST dans vos projets web*. OpenClassrooms. URL : <https://openclassrooms.com/fr/courses/3449001-utilisez-des-api-rest-dans-vos-projets-web> (visité le 11/10/2018).
- [7] *Tastypie 0.14.0 documentation*. URL : <https://django-tastypie.readthedocs.io/en/stable/toc.html> (visité le 18/10/2018).
- [8] *What is REST – Learn to create timeless RESTful APIs*. URL: <https://restfulapi.net/> (visited on 11/28/2018).

# Développement d'une application reconnaissance d'objets : Ajout de fonctionnalités

Pierrick Bobet

Encadrement : Maxime Martineau

## Contexte

Une plate-forme web Django a été développée lors d'un précédent projet. Cette application est une plate-forme permettant la reconnaissance d'insecte à partir d'une image importée. Polytech souhaite élargir les fonctionnalités de cette application.

## Attentes

- Mettre en place un API REST pour analyse reconnaissance
- Automatisation de l'intégration et du déploiement continu
- Rendre l'application générique avec une gestion des algorithmes de reconnaissance

## Réalisation

La plate-forme offre un pont d'accès par le biais de Django API REST. Ce qui permet l'interaction avec une future appli mobile de communiquer. Le déploiement continu a été mis en place grâce à Python Fabric3 afin de faciliter et automatiser le déploiement en production. Puis la plate-forme permet à présent d'accueillir de nouveau modèle de reconnaissance (plante, animaux...)



Résultat de reconnaissance d'insecte

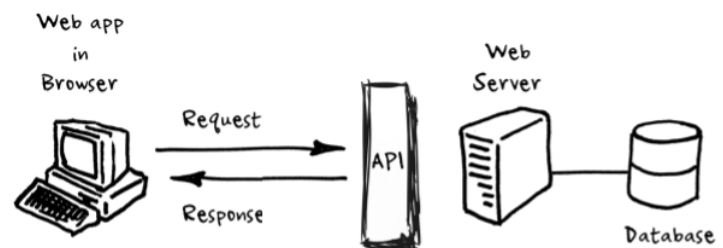


Schéma explicatif API REST

django  
REST  
framework

# Développement d'une application reconnaissance d'objets : Ajout de fonctionnalités

Pierrick Bobet

Encadrement : Maxime Martineau

## Contexte

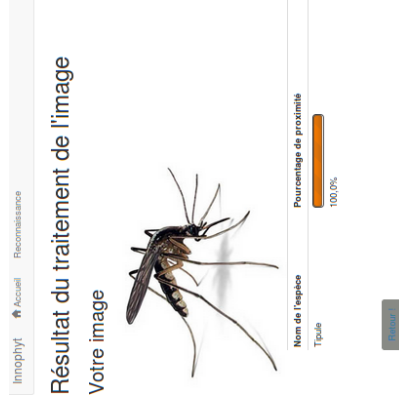
Une plate-forme web Django a été développée lors d'un précédent projet. Cette application est une plate-forme permettant la reconnaissance d'insecte à partir d'une image importée. Polytech souhaite élargir les fonctionnalités de cette application.

## Attentes

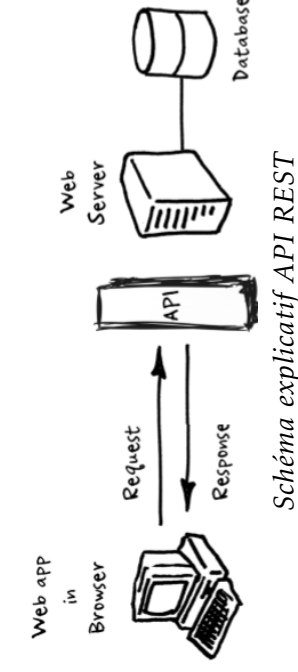
- Mettre en place un API REST pour analyse reconnaissance
- Automatiser de l'intégration et du déploiement continu
- Rendre l'application générique avec une gestion des algorithmes de reconnaissance

## Réalisation

La plate-forme offre un pont d'accès par le biais de Django API REST. Ce qui permet l'interaction avec une future application mobile de communiquer. Le déploiement continu a été mis en place grâce à Python Fabric3 afin de faciliter et automatiser le déploiement en production. Puis la plate-forme permet à présent d'accueillir de nouveau modèle de reconnaissance (plante, animaux...)



Résultat de reconnaissance d'insecte



*Schéma explicatif API REST*

django  
**REST**  
framework

# Développement d'une application reconnaissance d'objets

## Ajout de fonctionnalités

### Résumé

Dans son contexte, ce projet est une application web Python Django de reconnaissance pour les insectes. Les objectifs suivants sont de mettre en place un API REST permettant de faire un pont entre l'application existante et des futures interactions extérieurs (ex: une appli Android). En second, il s'agit de rendre la plateforme de reconnaissance générique à de multiple type. (ex: plante, animaux...). En troisième, la mise en place d'une automatisation afin de simplifier le déploiement en production (déploiement continu).

### Mots-clés

Django, API REST, Deep Learning, Déploiement continu, Généricité

### Abstract

This project is a Python Django web application for insect recognition. The following objectives are to set up a REST API to bridge the existing application and future external interactions (eg: an Android app). Second, it is to make the generic recognition platform to multiple type. (ex: plant, animals ...). Third, the introduction of automation to simplify deployment in production (continuous deployment)

### Keywords

Django, API REST, Deep Learning, Continuous deployment, Generic