OC Inventory

Solution d'inventaire de parc informatique

Dossier de conception technique

Version 1.0

Auteur
David Bouzerar
Développeur

IT Consulting & Development

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
David Bouzerar	22/10/2022	Création du document	0.1
David Bouzerar	07/11/2022	Ajout des sections	0.2
David Bouzerar	08/04/2023	Mise à jour du document en cohérence avec le développement	0.6
David Bouzerar	09/04/2023	Mise à jour de l'architecture de déploiement	0.8
David Bouzerar	16/04/2023	Ajout de la structure des sources	1.0
David Bouzerar	16/04/2023	Finalisation du document	1.0

2 - Introduction

2.1 - Objet du document

Le présent document constitue le dossier de conception technique de l'application OC Inventory, basé sur les informations présentent dans le dossier de conception fonctionnelle.

L'objectif de ce document est de présenter la solution technique que nous proposons pour répondre aux différents besoins exprimés par le demandeur.

2.2 - Références

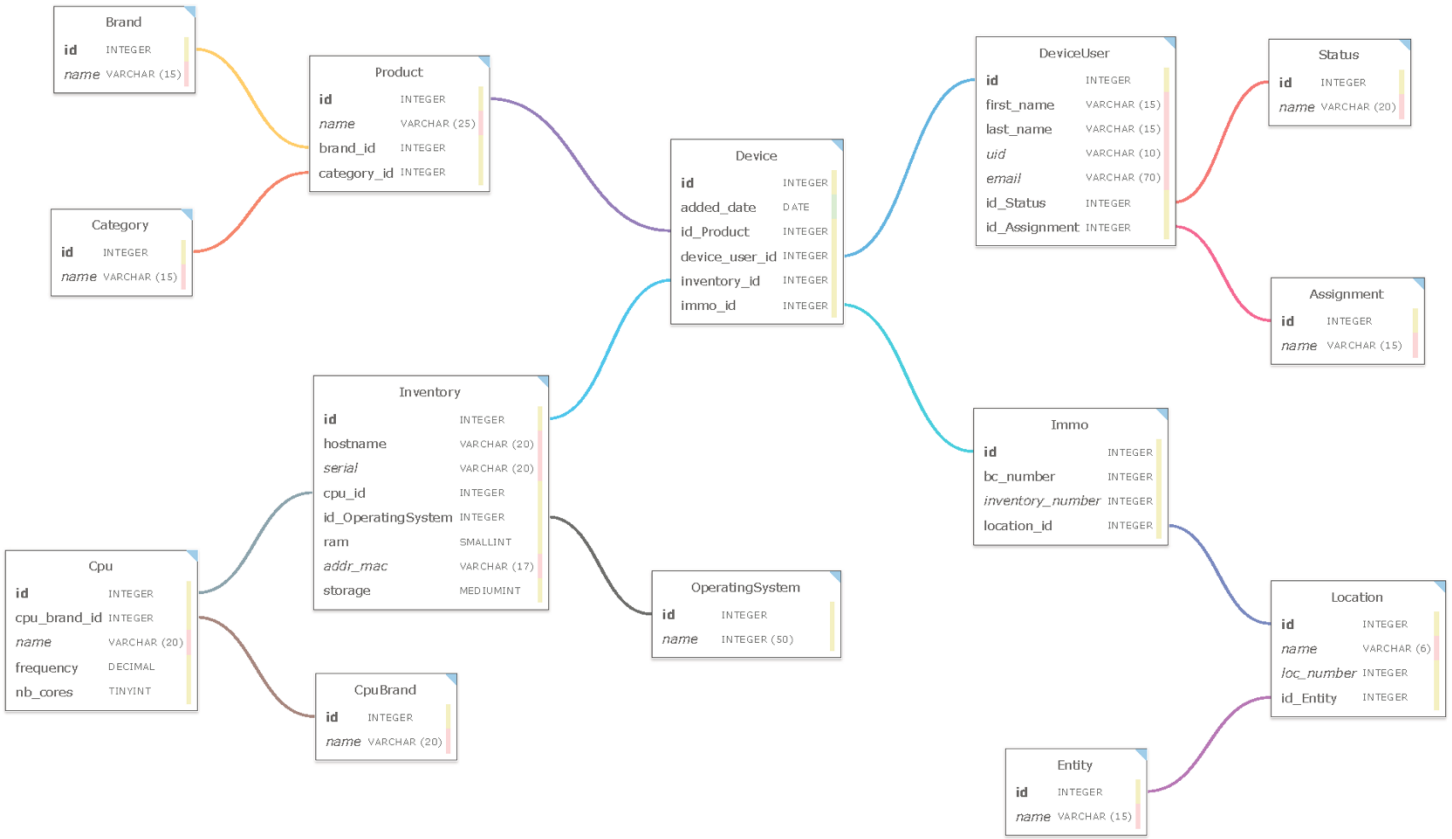
Pour de plus amples informations, se référer également aux éléments suivants :

1. **DCF-01** : Dossier de conception fonctionnelle de l'application
2. **DEXP-03** : Dossier d'exploitation de l'application

3 - Architecture Technique

3.1 - Modèle physique de données

Modèle physique de données - OC Inventory



3.2 - Description du modèle physique de données

Le diagramme ci-dessus (**MPD**) illustre l'architecture de la base de données ainsi que les relations existantes entre les différentes tables.

Nous pouvons, ici, considérer qu'il existe 4 parties distinctes (en dehors de la table principale nommée « **Device** ». Les parties « **Product** », « **Inventory** », « **Immo** » ainsi que « **DeviceUser** » dont voici la description :

3.2.1 - Gestion des produits :

- **Brand** : La table « **Brand** » est utilisée pour stocker les différentes marques des périphériques de notre inventaire. Elle contient un champ « *name* » sous contrainte d'unicité.

- **Category** : La table « **Category** » contient, elle, les différents types de périphériques (ex. : « *Workstation* », « *Notebook* », etc...). Elle contient également un champ « *name* » sous contrainte d'unicité.

- **Product** : La table « **Product** » rassemble les informations précédemment citées ainsi que le modèle du périphérique (champ « *name* »). Deux clés étrangères permettent de lier les tables « *Category* » (« *category_id* ») et « Brand » (« *brand_id* »).

3.2.2 - Gestion de l'inventaire (matériel) :

- **CpuBrand** : La table « **cpuBrand** » contient les différentes marques des CPU remontés en base. Un champ « *name* », sous contrainte d'unicité, le permet.

- **Cpu** : Cette table « **CPU** » contient les informations techniques des processeurs. Nous y retrouvons notamment la fréquence du CPU, le nom du modèle, le nombre de cores ainsi qu'une clé étrangère (« *cpu_brand_id* ») qui pointe vers la table « **CpuBrand** » (marque du CPU).

- **OperatingSystem** : La table « **OperatingSystem** » contient, comme son nom l'indique, les différents systèmes d'exploitation installés sur les périphériques.

- **Inventory** : La table « **Inventory** » réunit les informations fournis par les tables précédemment citées ainsi que, d'autres, complémentaires. On y retrouve notamment le « *hostname* » qui est le nom du périphérique sous son système d'exploitation, le « *serial* » qui est un élément unique permettant d'identifier précisément le périphérique, « *ram* » soit la quantité de mémoire vive embarquée, le champ « *storage* » qui correspond à la quantité de stockage présente sur le périphérique. On y retrouve également l'adresse MAC de la carte-réseau mais également deux clés étrangères vers les tables « **Cpu** » et « **OperatingSystem** ».

3.2.3 - Gestion de l'inventaire (administratif) :

- **Entity** : La table « **Entity** » permet d'identifier l'entité, propriétaire, du périphérique

- **Location** : La table « **Location** » contient toutes les informations permettant d'identifier la localisation du périphérique au sein de l'établissement. On y retrouve notamment les champs « *name* » (contenant le nom du bureau), « *loc_number* » contenant l'identifiant numérique du bureau ainsi qu'une clé étrangère vers la table « **Entity** ».

- **Immo** : Comme dans les groupes précédents, cette table rassemble les informations des tables précédentes et contient des informations complémentaires. On y retrouve notamment le champ « *bc_number* » contenant le numéro du bon de commande lié à l'acquisition du matériel, « *inventory_number* » qui contient le numéro d'inventaire du périphérique ainsi qu'une clé étrangère « *location_id* » permettant une liaison à la table « **Location** ».

3.2.4 - Gestion des utilisateurs des périphériques :

- **Status** : Cette table permet d'identifier le statut d'un utilisateur au sein de l'établissement. Nous pouvons ainsi savoir si l'utilisateur et un personnel technique, administratif, pédagogique etc… Un seul champ « *name* » dans cette table contient le nom du statut.

- **Assignment** : La table « **Assignment** » permet de connaître le lieu d'affectation de l'utilisateur. Il peut par exemple faire partie d'un laboratoire, d'une UFR, une coordination etc… Cette table contient également un champ unique contenant le nom de l'affectation.

- **DeviceUser** : Cette table contient les différentes informations permettant d'identifier un utilisateur. On y retrouve notamment les champs « *first_name* » et « *last_name* » contenant le prénom et nom de l'utilisateur, « *uid* » contenant l'uid LDAP de l'utilisateur, « *email* » contenant donc l'adresse email professionnelle, ainsi que deux clés étrangères « *id_status* » et « *id_assignment* » pointant respectivement vers les tables « **Status** » et « **Assignment** ».

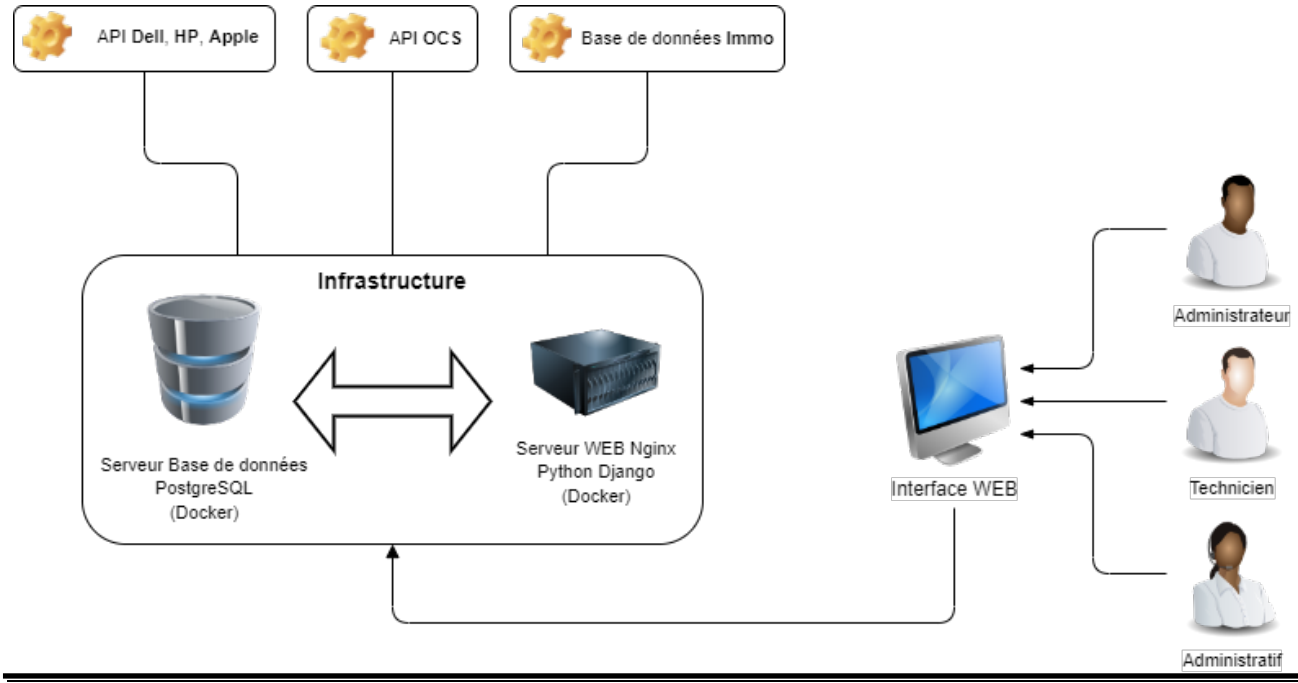
3.2.5 - Table principale de la base :

- **Device** : Il reste, tout de même, une dernière table que je n'ai volontairement pas classée dans une catégorie puisque celle-ci est le point d'entrée dans notre base et elle est seule à jouer ce rôle. Elle est liée à toutes les catégories que nous venons d'énumérer. En dehors du champ « *added_date* » (qui contient, par défaut, un paramètre **NOW**), elle ne contient que des clés étrangères vers les différentes catégories. On y retrouve notamment les champs « *id_product* » permettant de joindre la table « **Product** », « *device_user_id* » qui lie la table « **DeviceUser** », « *inventory_id* » et « *immo_id* » qui lient, respectivement, les tables « **Inventory** » et « **Immo** ».

3.3 - Application Web

La pile logicielle est la suivante :

- Serveurs **Ubuntu Server 22.04 LTS**
- Application **Django** (*version 4.2 LTS*) / **Python 3.9** / **PostgreSQL 14.7** / **HTML 5** / **CSS 3** / **Bootstrap 4**
- Serveur *HTTP* **NGINX 1.23**



*Solution **OC Inventory***

3.3.1 - Description

L'interface *WEB* étant accessible depuis n'importe quel périphérique (**PC**/**Mac**, tablette **iOS**/**Android**, Smartphone **iOS**/**Android**), il est nécessaire que celle-ci soit « **Responsive** ». C'est-à-dire qu'elle pourra s'adapter à n'importe quel périphérique et quelle que soit la résolution.

Cela permettra, dans un premier temps, de s'affranchir du développement d'application pour **iOS** et **Android**. Il sera toujours possible de le faire, si nécessaire, par la suite. Même si, l'utilisation du mobile n'est pas quelque chose d'indispensable pour ce type d'application.

Si jamais le mobile est un besoin, il sera tout de même judicieux de proposer une version « mobile » du site *web*, plus adaptée aux petites diagonales sur certains smartphones notamment.

L'application sera hébergée sur un serveur (**Linux**) Web **NGINX** où seront installés **Python** et **Django**. Un second serveur (**Linux**) hébergera la base de données **PostgreSQL**. Le serveur *Web* pourra communiquer avec la base de données lorsque cela sera nécessaire.

4 - Architecture de Deploiement

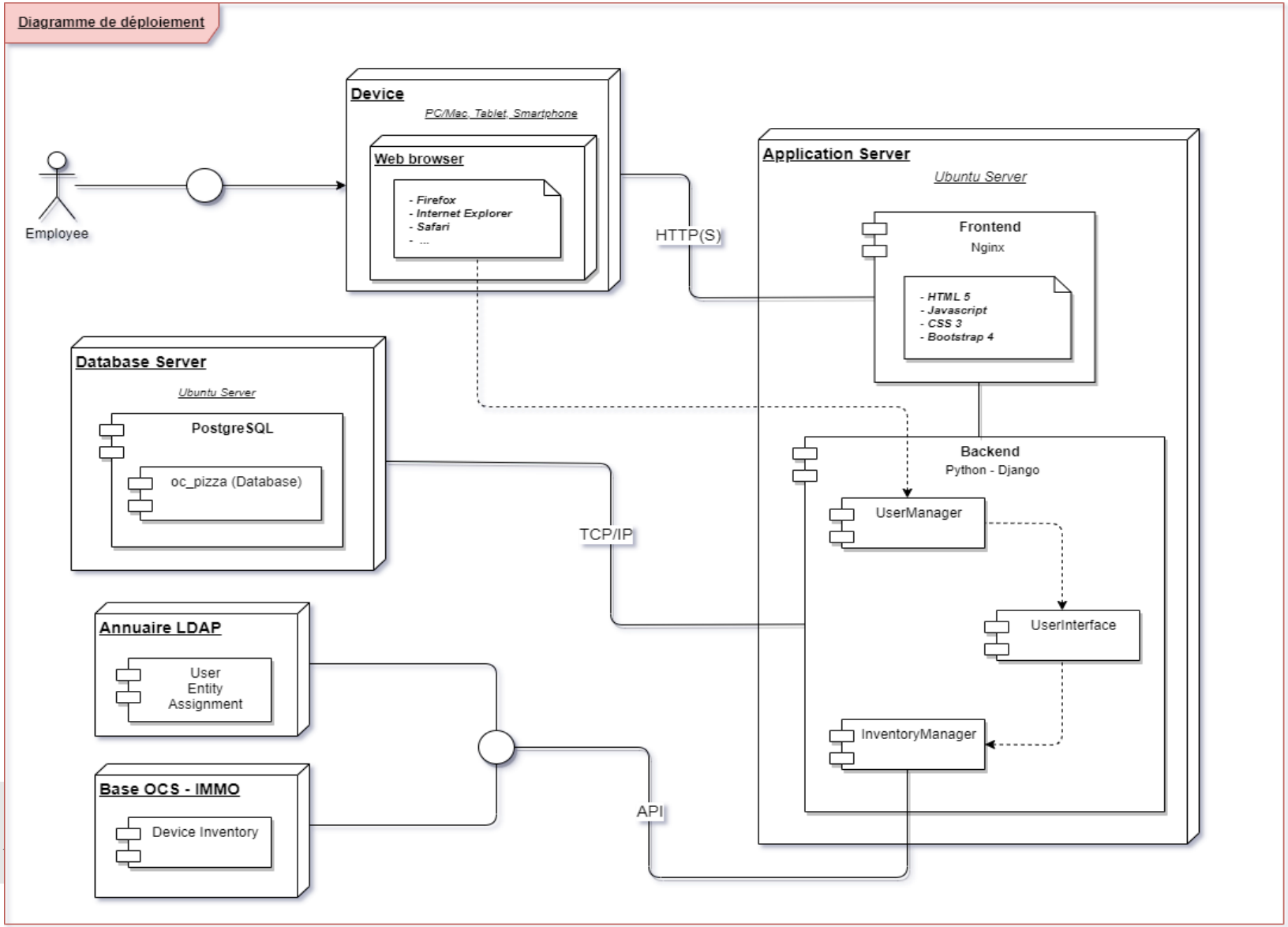


Diagramme de déploiement
Device
PC/Mac, Tablet, Smartphone
Web browser
- Firefox
- Internet Explorer
- Safari
- ...
Employee
HTTP(S)
Application Server
Ubuntu Server
Frontend
Nginx
- HTML 5
- Javascript
- CSS 3
- Bootstrap 4
Database Server
Ubuntu Server
PostgreSQL
oc_pizza (Database)
TCP/IP
Backend
Python - Django
UserManager
UserInterface
InventoryManager
Annuaire LDAP
User
Entity
Assignment
Base OCS - IMMO
Device Inventory
API

Ce diagramme illustre la façon dont les éléments (nécessaires au bon fonctionnement de l'application) sont répartis et communiquent au sein de l'infrastructure.

*A noter que le système d'exploitation installé sur les différents serveurs sera **Ubuntu Server**, basé sur **Debian**.*

4.1 - Utilisateurs :

Nous pouvons voir que les utilisateurs (**User**), qui, pour interagir avec l'interface web, utilisent un périphérique (*PC/Mac*, tablette ou smartphone, l'application étant *responsive*). Par le biais d'un navigateur *web* (*Firefox*, *Internet Explorer*, *Safari*, *Chrome* ...), ils communiquent avec le serveur *web* nommé ici « **Web Server** » en utilisant le protocole **HTTP(S)**.

4.2 - Serveur d'application (Frontend) :

Le serveur d'application (**Application Server**) héberge donc, comme son nom l'indique, l'application « **OC Inventory** ». **Python**, ainsi que le Framework **Django** sont installés sur ce serveur. Nous pouvons diviser l'application en deux groupes, le « **Frontend** » et le « **Backend** ».

Le « **Frontend** », peut également être défini comme étant le serveur **HTTP**. En tant que serveur **HTTP**, nous utilisons le logiciel « **NGINX** » pour traiter, notamment, les requêtes **HTTP** que le serveur réceptionnera.

Concernant la « partie visible », nous utilisons **HTML 5**, **Javascript**, **CSS 3** et **Bootstrap 4** pour créer l'interface graphique de l'application.

4.3 - Serveur d'application (Backend) :

La partie « **Backend** » est la partie immergée, le cerveau de l'application. Elle permet, notamment, d'interagir avec la base de données par l'intermédiaire de l'**ORM** intégré à **Django** et de renvoyer les informations au « **Frontend** » qui se chargera de les afficher sur le navigateur de l'utilisateur.

Concernant les relations, nous pouvons voir que l'utilisateur interagit avec l'élément « **UserManager** » notamment lors de l'authentification. Lui-même est lié à l'élément « **UserInterface** » qui est chargé d'afficher l'interface de l'utilisateur authentifié.

On peut également noter la liaison entre « **InventoryManager** », « **OCS** - **IMMO**» et l'annuaire « **LDAP** » nécessaire pour alimenter la base de données principale. Ces éléments étant externes à l'infrastructure, il est nécessaire de communiquer par le biais d'*API*.
Ce serveur communique également avec le serveur de base de données « **Database Server** » en utilisant le protocole **TCP/IP**

4.4 - Serveur de Base de données :

Nous terminons avec le serveur hébergeant la base de données. Ce serveur communique uniquement avec le serveur d'application.

PostgreSQL est installé sur ce serveur ainsi que **PgAdmin** pour faciliter l'administration de la base hébergée et nommée « **OC_Inventory** », qui contient elle-même toutes les tables nécessaires au bon fonctionnement de l'application.

5 - ARCHITECTURE LOGICIELLE

5.1 - Principes généraux

Les sources et versions du projet sont gérées par **Git**, alors que les dépendances de l'application seront gérées par le gestionnaire de paquet « **pip »** concernant la partie **Django**.

Django est un **Framework Python** basé sur une architecture **MVT** (**Model** – **View** – **Template**). Il permet notamment d'interagir avec la base de données par l'intermédiaire de son **ORM**, nous parlerons ici du « **Model** ». Il peut également recevoir des requêtes **HTTP** et d'y répondre en renvoyant (**View**), par exemple, un **Template** qui est en fait un fichier **HTML** pouvant recevoir des objets **Python**.

5.1.1 - Les couches

L'architecture applicative est la suivante :

- Une couche **View** : Responsable de la logique de l'application
- Une couche **Model** : Responsable de la représentation des objets métiers
- Une couche **Template** : Responsable des interfaces « utilisateur » de l'application

5.1.2 - Structure des sources

La structuration des répertoires du projet suit la logique suivante :

– Les répertoires sources sont créés de façon à respecter la philosophie de **Django** (à savoir : « **Model** – **View** – **Template** »)

```
OC_INVENTORY_APP
├── manage.py
├── requirements.txt
├── api
│   ├── immo_db.py
│   └── ocs_db.py
├── controllers
│   ├── controller.py
│   ├── controller_immo.py
│   └── controller_ocs.py
├── dashboard
│   ├── apps.py
│   ├── forms.py
│   ├── templates
│   │   └── dashboard
│   │       ├── advanced_search.html
│   │       ├── dashboard.html
│   │       └── show_device_table.html
│   ├── tests
│   │   ├── test_functional.py
│   │   └── test_views.py
│   ├── urls.py
│   └── views.py
├── databases
│   └── database_manager.py
├── inventory
│   ├── asgi.py
│   ├── settings
│   │   ├── production.py
│   │   └── travis.py
│   ├── staticfiles
│   │   └── ...
│   ├── urls.py
│   └── wsgi.py
├── product
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── management
│   │   └── commands
│   │       └── fill_main_db.py
│   ├── migrations
│   │   └── ...
│   ├── models.py
```

```
|   ├── static
|   |   ├── product
|   |   |   ├── css
|   |   |   |   ├── custom.css
|   |   |   |   └── styles.css
|   |   |   ├── img
|   |   |   |   ├── 404.jpg
|   |   |   |   ├── bg_blur.PNG
|   |   |   |   └── favicon.ico
|   |   |   └── js
|   |   |       ├── datatables-simple-demo.js
|   |   |       └── scripts.js
|   |   └── tests
|   |       └── test.css
|   ├── templates
|   |   ├── 404.html
|   |   └── product
|   |       ├── base.html
|   |       ├── base_body.html
|   |       ├── contact_us
|   |       |   ├── contact_us.html
|   |       |   ├── request_contact_us.html
|   |       |   └── request_contact_us_fulltext.html
|   |       ├── index.html
|   |       └── menu_user.html
|   ├── templatetags
|   |   └── product_extras.py
|   ├── tests
|   |   ├── test_functional.py
|   |   ├── test_models.py
|   |   └── test_views.py
|   ├── urls.py
|   └── views.py
├── product_user
|   ├── apps.py
|   ├── forms.py
|   ├── templates
|   |   ├── device_users
|   |   |   ├── device_users.html
|   |   |   ├── device_users_info.html
|   |   |   └── last_device_users.html
|   |   └── devices
|   |       ├── add_new_device.html
|   |       ├── all_devices.html
|   |       ├── device_info.html
|   |       └── last_devices.html
```

```
|   ├── tests
|   |   ├── test_functional.py
|   |   └── test_views.py
|   ├── urls.py
|   └── views.py
├── tools
|   ├── const.py
|   ├── func_db.py
|   └── func_for_tests.py
└── users
    ├── apps.py
    ├── forms.py
    ├── templates
    |   ├── registration
    |   |   ├── login.html
    |   |   ├── password_reset_confirm.html
    |   |   ├── password_reset_form.html
    |   |   └── signup.html
    |   └── users
    |       ├── account
    |       |   ├── account.html
    |       |   ├── change_fullname.html
    |       |   └── change_password.html
    |       ├── mails
    |       |   ├── mail_activate_account.html
    |       |   ├── mail_activate_account_fulltext.html
    |       |   ├── mail_reset_password.html
    |       |   └── mail_reset_password_fulltext.html
    |       └── notifications
    |           └── account_messages.html
    ├── tests
    |   ├── test_functional.py
    |   └── test_views.py
    ├── token.py
    ├── urls.py
    └── views.py
```

6 - POINTS PARTICULIERS

6.1 - Gestion des logs

La gestion des *erreurs* (et des *exceptions*) sera assurée par **Sentry**. Nous l'utiliserons également en tant que gestionnaire des événements pour l'application **OC Inventory**. En plus de la gestion des erreurs et des logs, il sera relié au **repository GitHub** de l'application afin de pouvoir suivre les issues éventuelles.

Nous utiliserons **NewRelic** pour surveiller les performances de l'application et, éventuellement l'optimiser. Le monitoring fourni par l'hébergeur (**DigitalOcean**) permettra également de surveiller les performances « matériels » du serveur. Afin, par exemple, de nous assurer que le serveur soit correctement dimensionné pour l'application ou encore d'être averti si l'utilisation des ressources devaient atteindre un certain seuil.

6.2 - Fichiers de configuration

6.2.1 - Application OC Inventory

```
OC_INVENTORY_APP
├── manage.py
├── inventory
│   ├── asgi.py
│   ├── settings
│   │   ├── production.py
│   │   └── travis.py
```

Le fichier « *production.py* » contient la configuration (finale) de l'application. Nous retrouvons dans ce fichier :

- La configuration de **Sentry** et notamment le « *niveau* » des alertes qui seront envoyées.
- Les hôtes autorisés (*ALLOWED_HOSTS*), il s'agira ici de saisir le nom de domaine attribué à l'application.
- Les informations concernant le serveur de base de données **Postgres** ainsi que la base de données elle-même.

Un autre fichier nommé « *travis.py* » contient lui la configuration vers l'outil d'intégration continue nommée « **Travis CI** ». Celui-ci contiendra un fichier de configuration simpliste de l'application nécessaire à la bonne exécution de l'outil, notamment concernant la base de données de données **Postgres**.

6.2.2 - Supervisor

```
/etc/supervisor
├── conf.d
|   └── oc_inventory_supervisor.conf
```

Supervisor est un gestionnaire de services, il permet donc de lancer des services au démarrage du serveur et de les surveiller ensuite. S'ils échouent pour une raison ou une autre, il se chargera de les redémarrer.

6.2.3 - NGINX

```
/etc/nginx
├── sites-available/
|   ├── default
|   └── oc_inventory
```

NGINX est un serveur **HTTP** (ainsi qu'un « *reverse proxy* ») couramment utilisé, il est même, à priori, le plus utilisé au monde depuis 2019. Son rôle est d'interpréter les requêtes émises par les navigateurs et de les renvoyer vers l'application.

Le fichier de configuration de l'application contient le nom de domaine, la localisation des fichiers statiques, ou encore, la configuration du *proxy*.

6.3 - Environnement de développement

Concernant le développement de l'application ou l'ajout de nouvelles fonctionnalités, il est recommandé de le faire au sein d'un « *environnement virtuel* ». Cela permettra alors d'intégrer les dépendances de cette application (et d'en ajouter si nécessaire) sans pour autant le polluer par des autres éléments utilisés dans cadre différent. Les dépendances actuelles sont répertoriées dans un fichier « *requirements.txt* » présent à la racine du projet.

6.4 - Procédure de packaging / livraison

Le développement de l'application respecte les valeurs de la méthode **AGILE**. Dans cette optique, les fonctionnalités sont ajoutées de façon continue et incrémentielle.

Pour cela, et comme indiqué plus haut, nous utilisons un outil d'intégration continue nommé **Circle CI**. Une fois l'application développée, le déroulement se fait de la façon suivante :

- Création d'un « *pull request* »
- **Circle CI** prépare son environnement et se charge ensuite d'exécuter les différents tests unitaires, d'intégration et fonctionnels
- Si un ou plusieurs tests échouent, une notification indique aux développeurs que la fonctionnalité ne peut pas être intégrée.
- Si les tests passent, le service indique que la fonctionnalité peut être intégrée à l'application

7 - GLOSSAIRE

B

Backend
Partie de l'application en arrière plan, pas directement accessible par un utilisateur 15

E

environnement virtuel
Un environnement virtuel est un répertoire contenant une installation autonome (indépendante de celle du système d'exploitation). 22

F

Framework
Ensemble d'outils et de composants logiciels à la base d'une application 15, 17
Frontend
Partie d'une application responsable de l'interface utilisateur 15

H

HTTP
Protocole de transmission permettant d'accéder à des pages web 11, 15, 17, 21

I

intégration continue
Ensemble de pratiques utilisées consistant à vérifier à chaque modification du code source que le résultat des modifications ne produit pas de régression dans l'application développée 21, 22

L

Logs
Type de fichier dont la mission consiste à stocker un historique des événements 20

M

monitoring

O

P

R

T