

RAPPORT DE STAGE

DEVELOPPEMENT D'UN
SYSTEME DE GESTION DE
FICHIER



Présenté par: Oudaoud El Mehdi
Responsable entreprise: Hamza AIT ES SAID
Durée: 30 jour

REMERCIEMENT

Mes remerciements s'adressent principalement à mon maître de stage, Hamza AIT ES SAID, pour m'avoir accordé sa confiance totale pour ce projet et pour toute l'aide qu'il m'a apportée du côté matériel ainsi son assistance technique au long du projet. Ainsi que mes collègues pendant la période de travail qui m'ont aidé à intégrer

RÉSUMÉ

Ce Projet propose une solution pour stocker les informations des clients sous forme d'un système fichier en utilisant une application web, tout en prenant en considération les limites matérielles et temporelles ainsi que l'aspect sécurité.

ABSTRACT

This project provides a solution to a data-sharing problem with users having read-only privileges in the form of a web app. In this report, I'll be detailing the train of thought behind its conception and the tools I used to optimize it for the available time and the hardware as well as my thoughts about its security.

TABLE DES MATIÈRES

RESUME	3
ABSTRACT	3
TABLE DES FIGURES.....	5
INTRODUCTION	6
MISSION DU STAGE	7
1- PRESENTATION DE L'ENTREPRISE	7
2- MISSION	7
3- CAHIER DE CHARGES	7
ANALYSE DU PROJECT	8
1- BESOIN FONCTIONNEL.....	8
2- BESOINS NON FONCTIONNEL	8
CONCEPTION ET IMPLEMENTATION	9
1- CHOIX DE TECHNOLOGIE	9
2- IMPLEMENTATION	9
1- <i>Arborescence</i>	9
1- Liste adjacente (adjacent list).....	10
2- Chemin matérialisé (materialized path)	12
3- Conclusion partielle :.....	13
2- <i>Architecture de l'application – Coté Serveur</i>	15
1- Nginx	15
2- Unicorn.....	16
3- <i>Architecture de application – Backend</i>	17
1- Django	17
2- Généralités sur l'application.....	18
3- ERD de la base de données	19
4- <i>Diagramme de classes - Front-End</i>	20
1- Partie fonctionnelle -SDK pour développeur front-end	20
2- Partie style – choix des couleurs et de la bibliothèque CSS	22
5- <i>Présentation de l'application</i>	23
1- Partie client	23
2- Partie Admin	24
3- API	25
6- <i>Problèmes rencontré</i>	26
CONCLUSION	30

TABLE DES FIGURES

Figure 1 : arbre pour une liste adjacente	9
Figure 2: Shema SQL pour AL:	10
Figure 3:Django Model pour AL.....	10
Figure 4: Méthode - parcours en profondeur pour AL.....	11
Figure 5: Méthode- parcours ascendant pour AL.....	11
Figure 6: Representation graphique de MP	12
Figure 7: Django Model pour MP	12
Figure 8: Méthode - parcours en profondeur MP	12
Figure 9: résumé des table utilisé pour la base de donnés.....	14
Figure 10: représentation en OOP du fichier JSON	14
Figure 11: Shéma représentant l'architecture de l'application.....	15
Figure 12: représentation du modèle MTV de django	17
Figure 13: architecture fonctionnel (non détaillé).....	18
Figure 14: ERD de la base données	19
Figure 15: petite référence sur la forme de l'inclusion du script	20
Figure 16: Diagramme de Classe (front-end)	20
Figure 17: palette utilisé.....	22
Figure 18: Authentification.....	23
Figure 19: représentation des statistique du client	23
Figure 20:Représentation des fichiers du client.....	23
Figure 21: Homepage de l'admin	24
Figure 22: représentation avec filtres des utilisateurs.....	24
Figure 23: Interface pour manipulation des fichiers des clients	24
Figure 24: les URL disponible sur l'api.....	25
Figure 25: Configuration des templates admin	27

INTRODUCTION

La qualité, la précision et la fiabilité des données présentées au client est primordiale pour le succès d'un cabinet comptable.

Néanmoins c'est le côté logistique et la bonne expérience qu'un cabinet fournit pour ses clients qui fait la grande différence. Et lorsqu'on parle logistique on incite toujours l'importance de la méthode de partage que ce soit des matériaux ou des données, ainsi que la façon avec lesquelles ils sont organisés.

Ce projet est une initiative proposée par Mr Hamza qui vise l'automatisation de la tâche de partage des données relatives aux clients d'une façon sécurisée via une application web simple à utiliser.

MISSION DU STAGE

1- Présentation de l'entreprise

ADVISORIS est un cabinet d'expertise comptable situé à Casablanca qui évolue principalement sur des activités :

- Assistance à la tenue de comptabilité et respect des obligations administratives, fiscales et sociales mensuelles ;
- Supervision, surveillance et conseil comptable ;
- Assistance à la préparation des reportings ;
- Assistance à l'établissement des déclarations fiscales et sociales (TVA, IR, IS, CNSS, etc..) ;
- Suivi de situations intermédiaires (mensuelles, trimestrielles ou semestrielles) ;
- Assistance à la préparation des comptes annuels ;
- Externalisation de la paie
- L'évaluation d'entreprise, l'assistance aux opérations de restructuration et Diagnostic Stratégique ;
- La gestion et la valorisation des stocks, des immobilisations et de l'ensemble du patrimoine de la société ;
- La sécurité et la fiabilité du système d'information ;
- La conformité du système d'information par rapport aux obligations légales et réglementaires ;
- La révision des comptes sociaux...

Voir le site web <http://advisoris.ma/>

2- Mission

Implémenter une application web capable de :

- Manipuler les fichiers des utilisateurs via une interface accessible aux administrateurs
- Afficher les données relatives à un utilisateur
- Assurer que ces données ne sont pas accessibles à d'autre utilisateur

3- Cahier de charges

Vu qu'il n'avait pas vraiment de cahier de charges explicite à respecter car cette application est un simple prototype, il fallait que je précise ce que l'application doit faire afin d'améliorer son architecture. Voici les points principaux qu'on a fini par adopter :

- Un système d'authentification complet (profil, récupération de mot de passe ...)
- Un système qui gère les permissions d'accès aux fichiers.
- Une interface où l'administrateur manipule les dossiers et fichiers.
- Une interface où le client peut accéder à son profil ainsi qu'au fichier que l'administrateur à partager

ANALYSE DU PROJET

Avant d'entrer dans la phase de conceptualisation il faut qu'on aie une claire et précise idée sur ce qu'on doit implémenter et son ordre d'importance sans inclure aucune contrainte sur le matériel où le budget du project.

1- Besoin fonctionnel

- Un systeme d'authentification
- Une manière de sauvegarder les profiles utilisateurs, administrateur
- Une couche de sécurité pour protéger les données de l'utilisateur (leaks, corruption de données ...)
- Interface présentative des données analytique de l'entreprise (IF , résultats des années précédentes en graphs...)
- Interface présentative des fichiers partagé par Advisoris avec des fonctionnalité de recherche
- Interface pour l'administrateur où il peut manipuler les fichiers/dossiers partagé avec le client
- Interface où l'administrateur peut manipuler les données des utilisateurs et leur accorder des permission
- Choisir la meilleur façon avec la quelle on sauvegarde arborescence du client.

2- Besoins non fonctionnel

- Une interface familière et facile à naviguer
- Optimiser le temps de chargement de la page web en utilisant un système cache.
- Un API accessible par le client via authentification pour que les clients puisse automatiser l'accès au fichier sans passer par l'interface graphique
- Un mécanisme sécurisé qui gère la récupération de mot de passe via email
- Utilisation d'un basculement (fail-over) pour garantir une haute disponibilité
- Utilisation d'un load balancer pour améliorer les performances de l'application
- Un ensemble d'animations et un style uniforme pour donner un bon look à l'application
- Sécuriser le serveur web

CONCEPTION ET IMPLEMENTATION

1- Choix de technologie

Mon choix était basé sur quatre critères :

- La durée du stage (1 mois)
- Le budget du Project (0dh)
- La technologie utilisée soit toujours vivante (reçois des mises à jour régulière)
- Une bonne documentation

Ainsi mon choix était des technologies open source et rapide à développer avec :

- Django pour le backend
- HTML , Materialize CSS , JS pour le front-end
- Ngnix come serveur web (open source)
- Gunicorn come middleware entre Django et Ngnix
- PostgreSQL come base de données
- Stay-alived pour le fail-over (si possible)

2- Implementation

1- Arborescence

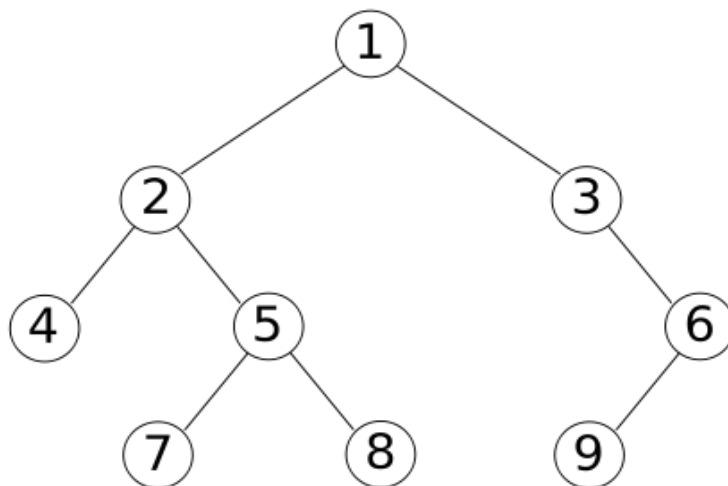


Figure 1 : arbre pour une liste adjacente

La plus grande impasse du projet est comment sauvegarder une arborescence spécifique à chaque utilisateur. Voici les méthodes que j'ai trouver

1- Liste adjacente (adjacent list)

Si je devais représenter la structure ci-dessus dans un schéma SQL pour le même serait de stocker une clé étrangère auto-référencée. Un attribut supplémentaire peut être ajouté qui peut décrire l'ordre du frère à un niveau particulier.

id	title	sib_order	desc	parent_id
1	N1	1		
2	N2	2		1
3	N3	3		1
4	N4	4		2
5	N5	5		2
6	N6	6		2
7	N7	7		3
8	N8	8		3
9	N9	9		7
10	N10	10		7

(10 rows)

Figure 2: Shema SQL pour AL:

Code :

```
from django.db import models
from django.db.models import fields

class Node(models.Model):
    parent = models.ForeignKey(
        "self",
        on_delete=models.CASCADE,
    )

    some_payload = ...

CREATE TABLE "node" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "parent_id" integer NULL REFERENCES "node" ("id"),
    "some_payload" varchar(1000)
);
```

Figure 3:Django Model pour AL

Avantages :

- Facile à implémenter
- Les opérations écriture sont rapide

Inconvénient :

- On a besoin une nouvelle requête SQL pour chaque niveau de profondeur ce qui introduit un va et vient entre l'application et la base de données.
- Chercher les descendants dans cette implémentation est simple mais ça retourne un itérateur donc on perd les bénéfices d'un Queryset.

```

from itertools import chain
from typing import Iterable

def get_descendants(node: Node) -> Iterable[Node]:
    queryset = Node.objects.filter(parent=node)
    results = chain(queryset)
    for child in queryset:
        results = chain(results, get_descendants(child))
    return results

list(get_descendants(Node.objects.get(pk=4)))

```

Figure 4: Méthode - parcours en profondeur pour AL

```

from itertools import chain
from typing import Iterable

def get_ancestors(node: Node) -> Iterable[Node]:
    if node.parent:
        return chain([node.parent], get_ancestors(node.parent))
    return chain()

list(get_ancestors(Node.objects.get(pk=16)))

```

Figure 5: Méthode- parcours ascendant pour AL

- Filtrer les données à une complexité de $O(n)$ en temps et en mémoire

2- Chemin matérialisé (materialized path)

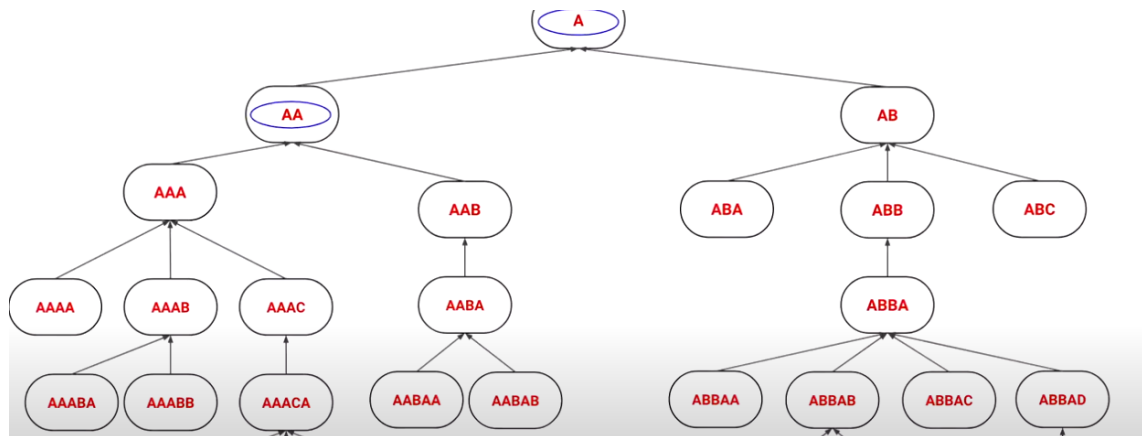


Figure 6: Représentation graphique de MP

Dans une approche de chemin matérialisé, chaque nœud de l'arborescence aura un attribut de chemin, où le chemin complet de la racine au nœud sera stocké.

Code :

```
from django.db import models
from django.db.models import fields

class Node(models.Model):
    steplen = 4 # in this presentation: 1
    alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

    path = fields.CharField(
        "Path",
        max_length=1000,
        db_index=True,
    )

    some_payload = ...
```

Figure 7: Django Model pour MP

```
def get_descendants(node: Node) -> QuerySet:
    qs = Node.objects.filter(path__startswith=node.path)
    qs = qs.exclude(path=node.path)
    return qs

list(get_descendants(Node.objects.get(pk=4)))
```

Figure 8: Méthode - parcours en profondeur MP

Avantages :

- Recherche de parent et de descendants est super rapide
- Les opérations écriture sont rapide

Inconvénient :

- Désinfecter et vérifier la validité des données est très difficile à implémenter et consomme énormément du temps.

3- Conclusion partielle :

J'ai aussi considéré deux autres méthodes (nested sets, AL with common table Expressions). Cependant je me suis concentré sur les solutions les plus réalistes en terme de temps d'implémentation ainsi que la magnitude du projet (prototype). Autrement dit utiliser une solution un peu moins sophistiquée.

L'idée derrière l'implémentation finale est d'utiliser **l'espace de stockage disponible pour sauvegarder l'arborescence sur un fichier JSON (dupliqué automatiquement pour rendre la base de données plus robuste contre les erreurs et corruptions)** ainsi on utilisera le matériel de l'utilisateur pour procéder à l'arbre au lieu d'inonder le serveur par des requêtes à la base de données.

Pour mieux expliquer pourquoi cette méthode est fiable voici quelques points importants :

- 1- chaque utilisateur a sa propre arborescence de fichiers
- 2- chaque fichier est spécifique à un utilisateur et seul cet utilisateur peut y accéder
- 3- les fichiers ne sont pas dupliqués dans l'arborescence (fichier1 est à l'intérieur de dossier1 uniquement !)
- 4- l'arbre n'est pas trop profonde donc le fichier JSON ne dépassera pas les 1-3 ko

Ceci signifie que chaque répertoire a des fichiers uniques ainsi tous les répertoires sont uniques.

L'ERD des tables en relation avec l'arborescence est la suivante :

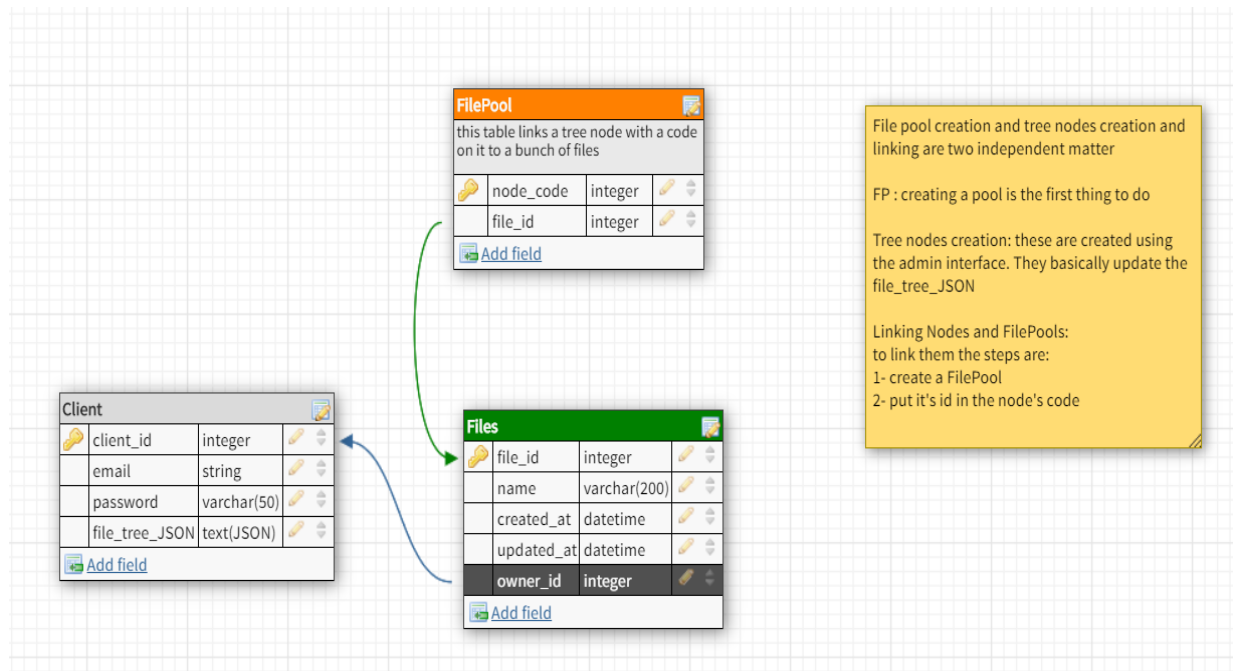


Figure 9: résumé des table utilisé pour la base de donnés

La forme de l'arborescence (file_tree_JSON) peut être représenté en tant que classe

```
class Dossier
{
    display:String;
    filepool:number;
    subtrees:Array<Node>;
    constructor(NomDuDossier:String, RepertoireDuDossier:number, SousDossiers:Array<Node>){
        this.display = NomDuDossier;
        this.filepool = RepertoireDuDossier;
        this.subtrees = SousDossiers;
    }
}
```

Figure 10: représentation en OOP du fichier JSON

- NomDuDossier : comme son nom l'indique elle présente le nom du dossier
- RepertoireDuDossier : ou filepool comme présenté sur la base de données
 - Si = 0 : ce dossier ne contiens aucun fichier
 - Sinon ce dossier contiens un ensemble de fichiers qui sont enregistré
- SousDossiers : une liste de dossier qui existe dedans le dossier couramment sélectionné

2- Architecture de l'application – Coté Serveur

Idéalement il faut utiliser docker pour les copies de l'application et une machine virtuelle pour la base de données afin d'isoler les différentes copies de l'application qui marche en parallèle sur un même serveur (ou un cluster de serveur) pour assurer une meilleure performance et disponibilité ainsi que faciliter le diagnostic des anomalies (lié à l'application non au transaction avec les clients).

L'application elle-même est sans état (stateless) tout état est soit dans passé par les cookies (information sur la session) ou bien sauvegardé dans la base de données (son arborescence).

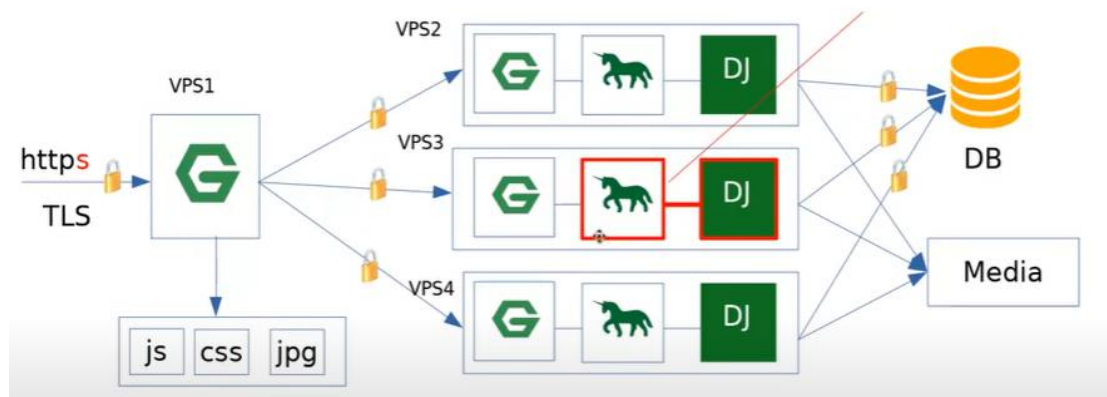


Figure 11: Schéma représentant l'architecture de l'application

L'application Django sera écrite comme une seule unité avec une Communication indirecte avec la base de données via ORM ou une communication avec le Rest API implémenter pour supporter les transaction asynchrone avec les clients (3 Layer architecture)

1- Nginx

Nginx est un serveur Web et un proxy inverse et c'est l'endroit où les demandes d'Internet arrivent en premier. Il peut les traiter très rapidement et est généralement configuré pour ne laisser passer que les demandes, qui doivent vraiment arriver à votre application Web. Il est hautement optimisé pour tout ce qu'un serveur Web doit faire. Voici quelques points pour lesquels il est excellent:

- Prenez soin du routage du nom de domaine (décide de la destination des requêtes ou si une réponse d'erreur est en ordre)
- Servir des fichiers statiques
- Gérez de nombreuses demandes en même temps
- Gérer les clients lents (utile contre les attaques Slow loris)
- Transmet les demandes qui doivent être dynamiques à Gunicorn
- Terminer SSL (https se produit ici)
- Économisez des ressources informatiques (CPU et mémoire) par rapport à votre code Python
- Équilibrage de charge, mise en cache, ...

Cependant Nginx ne peut pas :

- Exécuter des applications Web Python pour vous
- Traduire les demandes en WSGI

Donc on a besoin d'un homme au milieu pour faire la traduction on choisit pour ce projet le serveur WSGI Gunicorn

2- Gunicorn

Une fois que Nginx décide qu'une demande particulière doit être transmise à Gunicorn (en raison des règles avec lesquelles vous l'avez configurée), il est temps pour Gunicorn de briller. Il est hautement optimisé et possède de nombreuses fonctionnalités pratiques. Principalement, ses emplois consistent en :

- Exécution d'un pool de processus de travail / threads (exécution de votre code !)
- Traduit les demandes provenant de Nginx pour qu'elles soient compatibles WSGI
- Traduire les réponses WSGI de votre application en réponses http appropriées
- Appelle en fait votre code Python lorsqu'une requête arrive
- Gunicorn peut parler à de nombreux serveurs Web différents

Ce que Gunicorn ne peut pas faire pour vous :

- Pas prêt à être face à face : facile à DOS et submergé
- Impossible de mettre fin à SSL (pas de gestion https)
- Faites le travail d'un serveur Web comme Nginx, ils sont meilleurs dans ce domaine

3- Architecture de application – Backend

1- Django

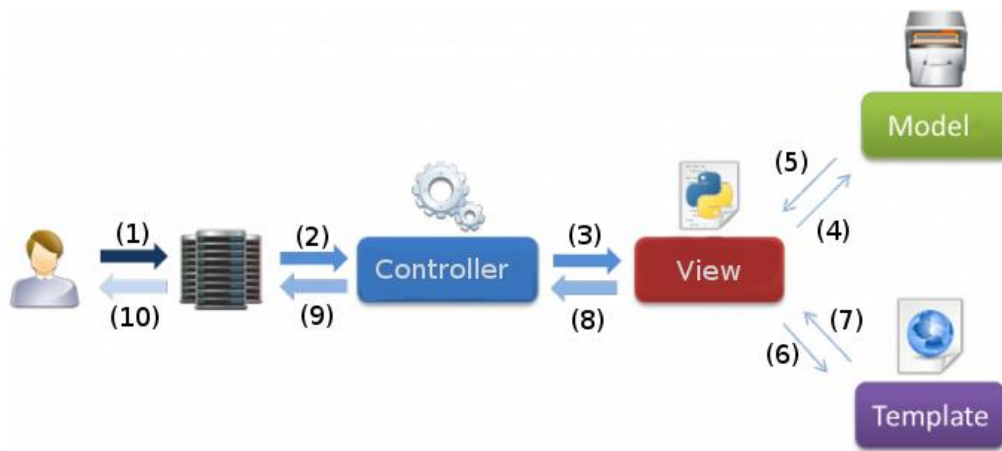


Figure 12: représentation du modèle MTV de django

Un modèle de conception de cadre Web est l'architecture (c'est-à-dire le processus) que le cadre adopte pour traiter une demande client et renvoyer la réponse. Django suit le modèle MTV (Model, View, Template). Comme décrit dans la figure ci-dessus, la requête envoyée par le client est traitée par le Controller qui est un composant intégré de Django, cette procédure est transparente pour le développeur. Ensuite, le contrôleur transmettra la requête à la vue correspondante, dont le rôle est d'interroger le composant modèle pour extraire les données de la base de données. Après avoir extrait les données recherchées par le client, les données sont formatées par le modèle qui rend un fichier HTML ou XML. Lorsque le processus est terminé, le composant Template retournera le fichier rendu à la vue qui à son tour envoie ce fichier au contrôleur. Enfin, le contrôleur envoie la page Web (HTML ou XML) au client.

Pour plus de détails sur MTV, les modèles (ORM) , les vues et les contrôleur de routage les middlewares ... voir <https://docs.djangoproject.com/fr/3.0/>

2- Généralités sur l'application

En somme voici l'architecture Totale de l'application

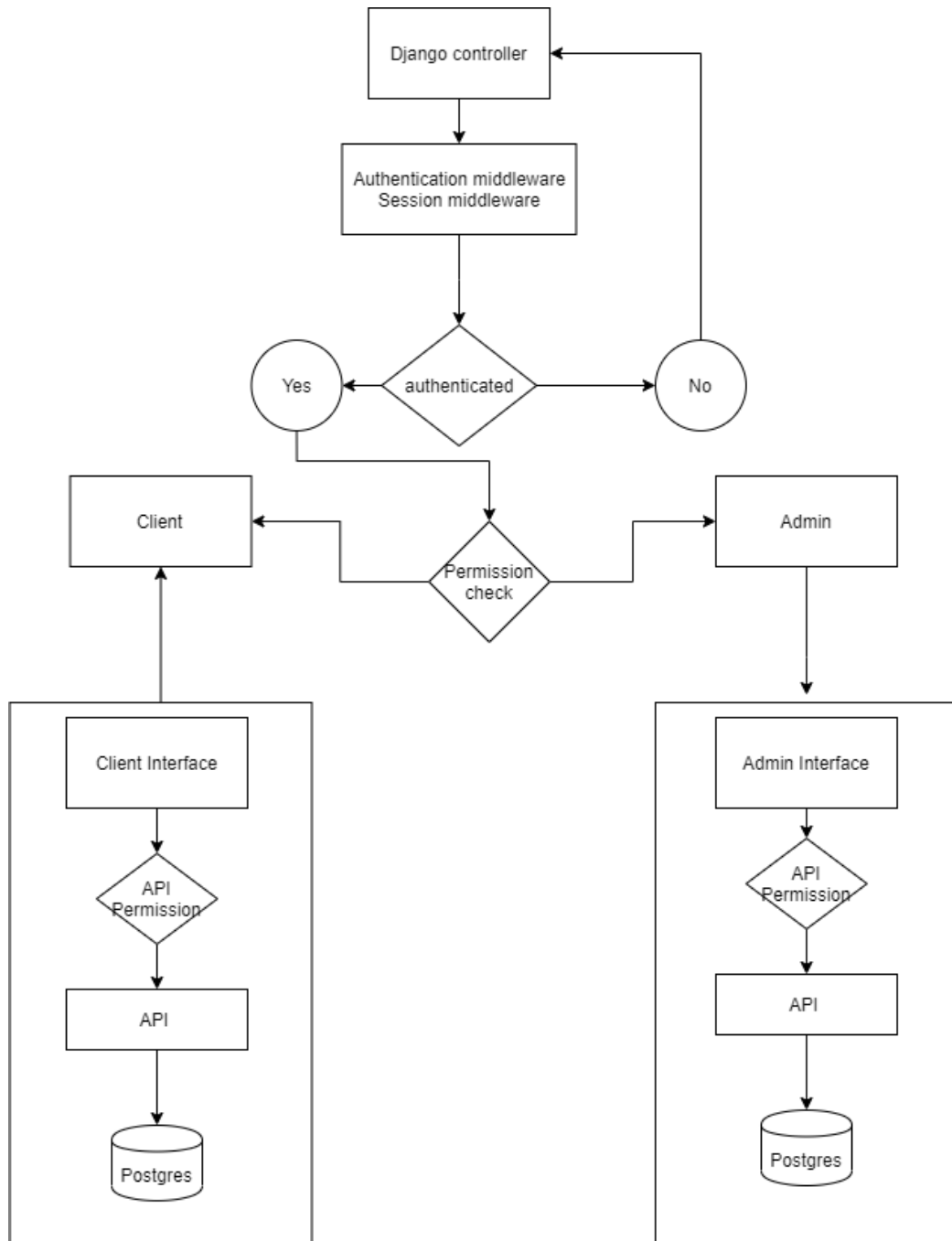


Figure 13: architecture fonctionnel (non détaillé)

3- ERD de la base de données

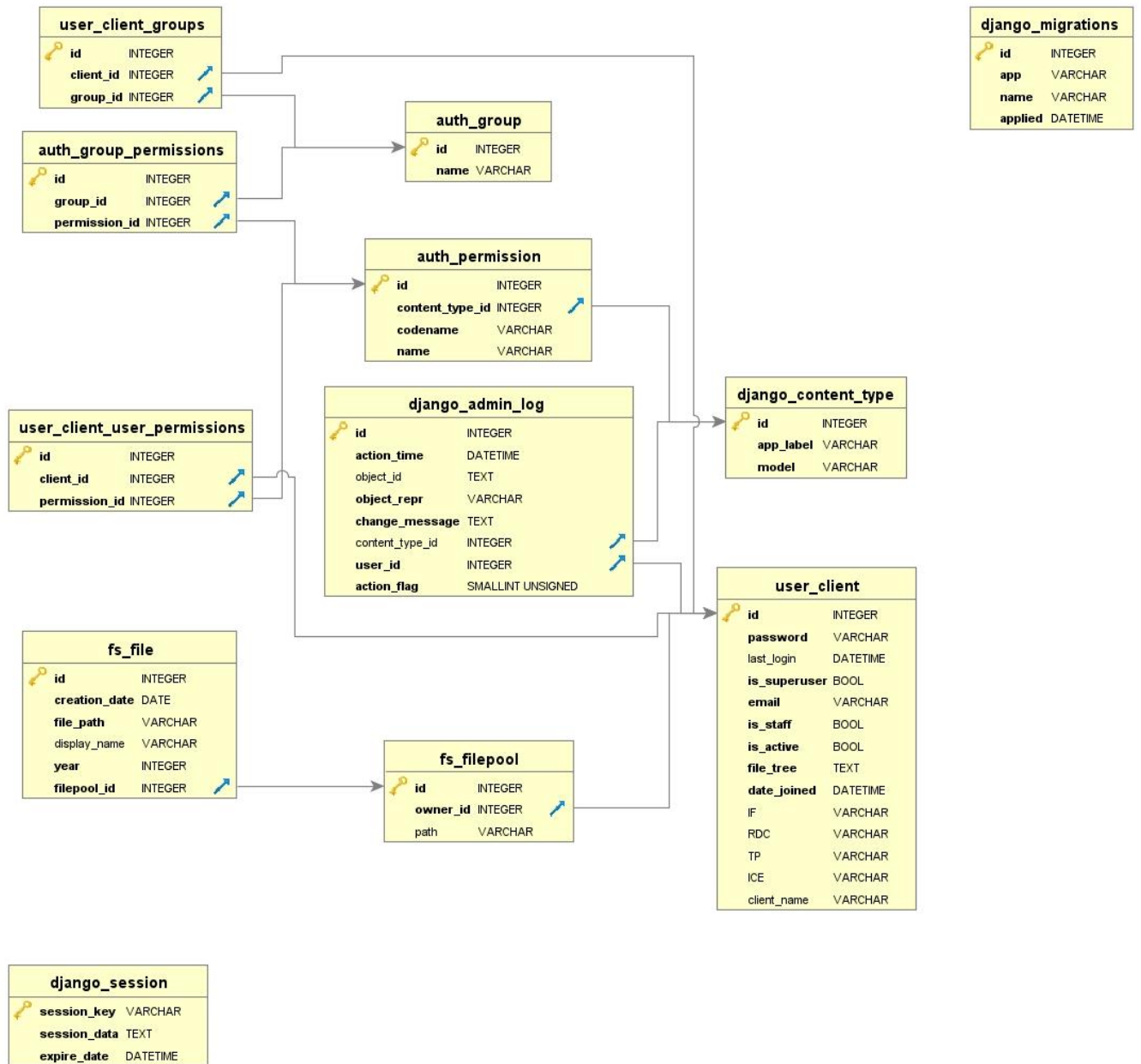


Figure 14: ERD de la base données

4- Diagramme de classes - Front-End

1- Partie fonctionnelle -SDK pour développeur front-end

```
<script>
  const pdf_image_url = "{% static 'img/pdf_icon.jfif' %}";
  const txt_image_url = "{% static 'img/txt_icon.png' %}";
  const xlsx_image_url = "{% static 'img/xlsx_icon.png' %}";
  const docx_image_url = "{% static 'img/docx_icon.png' %}";
  const im_image_url = "{% static 'img/jpg_icon.png' %}";
  const other_image_url = "{% static 'img/other_icon.png' %}";
</script>
<script src="{% static 'js/utlis.js' %}"></script>
<script src="{% static 'js/FSHandler.js' %}"></script>
<script src="{% static 'js/UserUI.js' %}"></script>
<script>
  const h = new InterfaceHandler('{request.user.id}');
</script>
```

Figure 15: petite référence sur la forme de l'inclusion du script

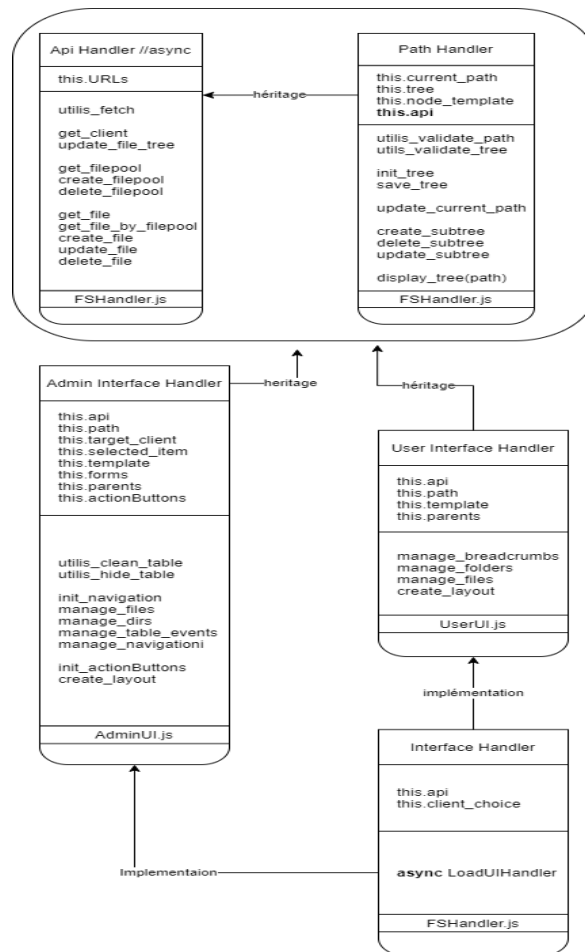


Figure 16: Diagramme de Classe (front-end)

1- Path Handler :

Cette classe permet de simplifier la manipulation et affichage de l'arborescence en abstrayant sa forme. C'est-à-dire on travaille avec des 'path' comme ceux dans un système d'exploitation au lieu de chercher les nœuds dans l'arbre de manière directe.

2- API Handler :

Cette Classe asynchrone permet de faciliter les échanges avec l'API ainsi que permettre la modification des URL de l'API juste en changeant la variable de class 'this.URLS'. Ceci rend cette interface assez flexible pour l'implémentation de nouvelle version de l'API.

Elle permet aussi d'abstraire toutes les opérations dont l'API est capable de gérer.

3- Interface Handler

Cette Interface permet d'abstraire la façon avec laquelle les générateurs de l'interface marchent en spécifiant une instance Api Handler et le client de choix (seul l'admin peut avoir accès aux données des autres clients)

La méthode LoadUIHandler permet de démarrer les générateurs d'interface (en utilisant create_layout) de manière asynchrone de telle sorte que l'utilisateur ne doit pas actualiser la page pour voir les nouveaux fichiers dans son arbre, ainsi que l'administrateur en manipulant les fichiers/dossiers dans l'arborescence d'un utilisateur.

4- Admin Interface Handler

Implémente Interface Handler et permet de générer une interface pour administrateur en choisissant un client

La template utilisée pour représenter cette interface est modifiable depuis la variable de classe 'this.template' facilitant ainsi la tâche de re-design de l'interface sans modifier le code.

5- User Interface Handler

Fonctionne de façon similaire à admin interface handler cependant elle présente les parties dynamiques dans l'interface d'un client non administrateur.

2- Partie style – choix des couleurs et de la bibliothèque CSS

Cette application n'est qu'un prototype pour les clients et l'administrateur pour avoir des feedbacks sur cette ergonomie. Cependant je pense qu'il est nécessaire pour cette application d'avoir une bonne esthétique pour contraster un peu la forme finale du projet. Donc j'ai choisi une palette couleur similaire à celle du logo avec penchant vers un thème 'light' pour que l'application parait accueillante.

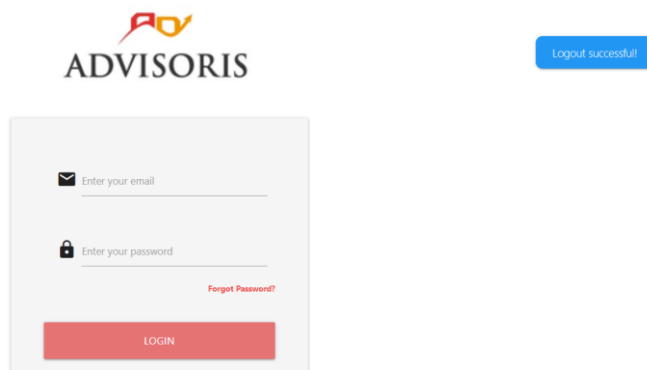


Figure 17: palette utilisé

J'ai utilisé un style du genre « Material Design » ainsi le choix de la bibliothèque CSS était assez simple : Materialise CSS (proposé par Google). Cette Framework est très similaire à Bootstrap4 donc je n'avais pas de problème en l'utilisant.

5- Présentation de l'application

1- Partie client



The login page for Advisoris features the company logo at the top center. Below it is a login form with two input fields: 'Enter your email' and 'Enter your password'. A 'Forgot Password?' link is positioned below the password field. A red 'LOGIN' button is at the bottom of the form. In the top right corner, a blue button indicates 'Logout successful!'.

Figure 18: Authentification



Figure 19: représentation des statistique du client

root > dir4

Directories

dir4_subdir 3

dir4_subdir4

Files





 pep 8 pdf edit Size: 190.98 kb Year: 2010 DOWNLOAD	 image Size: 19.36 kb Year: 2012 DOWNLOAD	 excel sheet Size: 206.01 kb Year: 2010 DOWNLOAD	 document Size: 15.87 kb Year: 2030 DOWNLOAD
--	--	---	---

Figure 20: Représentation des fichiers du client

2- Partie Admin

welcome to Advisoris FS

FS

Files

+ Add

Change

USER

Clients

+ Add

Change

Recent actions

My actions

Y company

Client

client 20

Client

+ client 20

Client

admin 1

Client

admin 1

Client

admin 1

Client

admin 1

Client

admin 1

Client

admin 1

Client

Client XXX

Client

+ Client XX

Client

Figure 21: Homepage de l'admin

Select client to change

Q

Search

Actions: 0 of 5 selected

<input type="checkbox"/>	CLIENT NAME	EMAIL ADDRESS	DATE JOINED	STAFF STATUS
<input type="checkbox"/>	admin 1	02.oudaoud@gmail.com	Aug 1, 2020, 12:24 p.m.	<div></div>
<input type="checkbox"/>	client 20	69.oudaoud@gmail.com	Aug 16, 2020, 1:53 p.m.	<div></div>
<input type="checkbox"/>	Client XXX	user12@gmail.com	Aug 15, 2020, 8:21 p.m.	<div></div>
<input type="checkbox"/>	Client XX	user1@gmail.com	Aug 15, 2020, 8:16 p.m.	<div></div>
<input type="checkbox"/>	Y company	ww.oudaoud@gmail.com	Aug 1, 2020, 12:25 p.m.	<div></div>

5 clients

FILTER

By staff status

All

Yes

No

By superuser status

All

Yes

No

By active

All

Yes

No

Figure 22: représentation avec filtres des utilisateurs

Add files

Chose Client: Y company

←

root/dir4/

✓

+

Name	Type	Size	Year
dir4 subdir 3	Folder	-	-
dir4_subdir4	Folder	-	-
pep 8 pdf edit	pdf	190.98 kb	2010
image	jpg	19.36 kb	2012
excel sheet	xlsx	206.01 kb	2010
document	docx	15.87 kb	2030

Figure 23: Interface pour manipulation des fichiers des clients

3- API

J'aimerais bien vous montrer quelque exemple du système de permission sur l'API :

Les url pour la version 1.0 de l'API :

```
GET /api/

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

{
  "clients": "http://localhost:8000/api/clients/",
  "filepools": "http://localhost:8000/api/filepools/",
  "files": "http://localhost:8000/api/files/"
}
```

Figure 24: les URL disponible sur l'api

Pour un compte admin (GET api/clients) :

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

[
  {
    "id": 1,
    "url": "http://localhost:8000/api/clients/1/",
    "client_name": "admin 1",
    "file_tree": "{\\display\\": \"root\\", \"filepool\\": 0, \"subtrees\\": []}",
    "filepools": []
  },
  {
    "id": 2,
    "url": "http://localhost:8000/api/clients/2/",
    "client_name": "Y company",
    "file_tree": "{\\display\\": \"root\\", \"filepool\\": 0, \"subtrees\\": [{\\display\\": \"dir4\\", \"filepool\\": 4, \"subtrees\\": [{\\display\\": \"dir4_subdir2\\", \"filepool\\": 0, \"subtrees\\": []}]}]",
    "filepools": [
      3,
      4,
      38,
      33,
      35
    ]
  }
]
```

L'Api affiche les informations de tous les clients

Pour un compte client :

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

[
  {
    "id": 2,
    "url": "http://localhost:8000/api/clients/2/",
    "client_name": "Y company",
    "file_tree": "{\\display\\": \"root\\", \"filepool\\": 0, \"subtrees\\": [{\\display\\": \"dir4\\", \"filepool\\": 4, \"subtrees\\": [{\\display\\": \"dir4_subdir2\\", \"filepool\\": 0, \"subtrees\\": []}]}]",
    "filepools": [
      3,
      4,
      38,
      33,
      35
    ]
  }
]
```

L'Api affiche les informations du client en question

6- Problèmes rencontré

1- Personnaliser l'interface admin - Django

Django possède une interface administrateur qui se génère automatiquement selon les modèles enregistrés. Cependant pour ce projet il faut faire des modifications pour assurer que l'interface soit adéquate au besoin de l'employant. Cependant j'ai pas trouver des articles détaillant cet aspect car il est préférable de créer sa propre interface admin en utilisant l'API que Django propose, mais j'avais pas vraiment assez de temps pour tout apprendre.

Heureusement Django est open-source, donc c'est possible de lire le code que Django propose pour créer automatiquement l'interface admin et le manipuler à notre fin.

Dans django 'admin.ModelAdmin', vous avez le choix entre plusieurs options: (dans options.py)

- 1- personnaliser les filtres, afficher
- 2- Personnalisez les modèles html utilisés
- 3- Ajouter des URL à l'interface d'administration
- 4- Ajouter des fichiers MEDIA à l'interface d'administration

```
@property
def media(self):
    extra = '' if settings.DEBUG else '.min'
    js = [
        'vendor/jquery/jquery%s.js' % extra,
        'jquery.init.js',
        'core.js',
        'admin/RelatedObjectLookups.js',
        'actions%s.js' % extra,
        'urlify.js',
        'prepopulate%s.js' % extra,
        'vendor/xregexp/xregexp%s.js' % extra,
    ]
    return forms.Media(js=['admin/js/%s' % url for url in js])
```

Media est un widget dans le django.forms voici une description rapide de la classe.

```

@html_safe
class Media:
    def __init__(self, media=None, css=None, js=None):
        if media is not None:
            css = getattr(media, 'css', {})
            js = getattr(media, 'js', [])
        else:
            if css is None:
                css = {}
            if js is None:
                js = []
        self._css_lists = [css]
        self._js_lists = [js]

```

Alors, comment puis-je passer outre la propriété des médias

1- les fichiers js sont constanst

- nous pouvons créer une autre propriété avec le même nom dans notre AdminModel personnalisé
- nous créons un objet ModelAdmin factice uniquement pour obtenir les valeurs constantes renvoyées par le média

Conclusion: cette façon de faire marche cependant c'est moins sécurisé et surtout 'overkill' pour notre cas

Après avoir lire le code de la classe AbstractModelAdmin j'ai trouvé une façon moins flexible mais suffisante pour modifier la template du site administrateur.

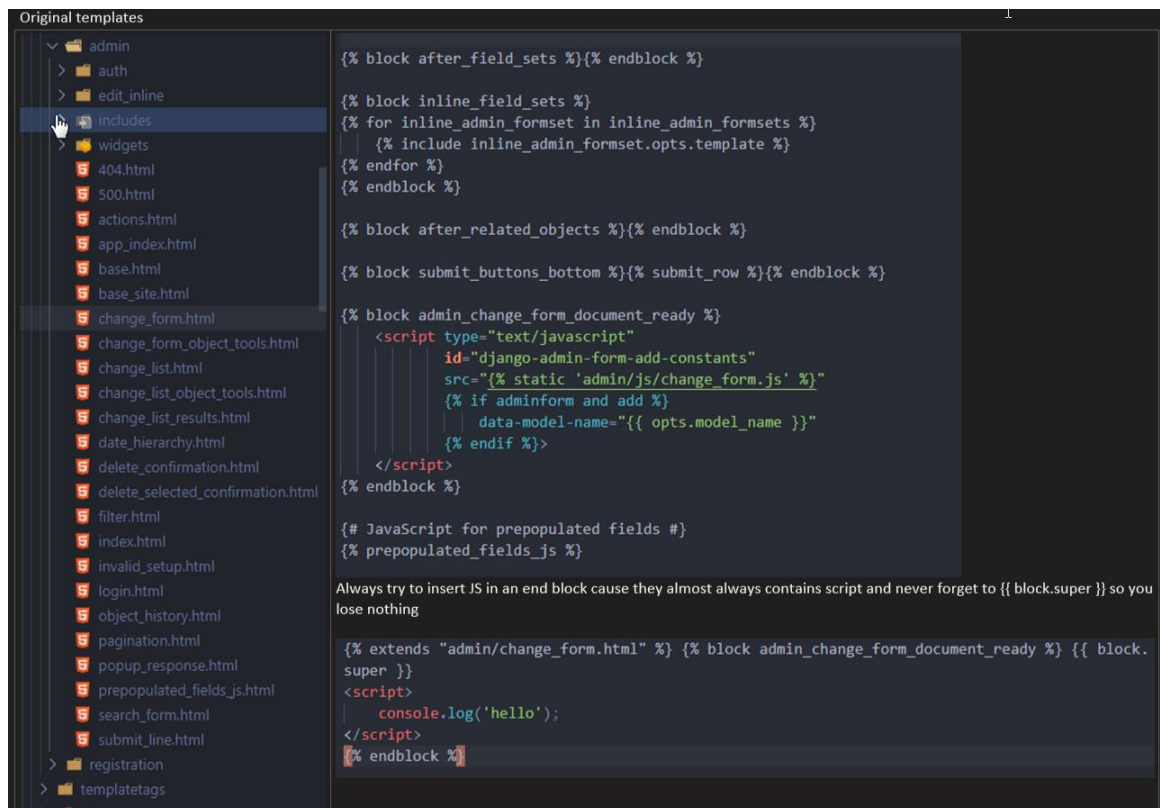
```

#? Custom templates (designed to be over-ridden in subclasses)
#     add_form_template = None
#     change_form_template = None
#     change_list_template = None
#     delete_confirmation_template = None
#     delete_selected_confirmation_template = None
#     object_history_template = None
#     popup_response_template = None
#? add to urls --> override get_url()

```

Figure 25: Configuration des templates admin

Voici une figure qui montre les templates existante si on cherchera encore plus de changement dans l'interface administrateur



2- Passer/Analyser les fichiers(BLOB) sur JS puis vers REST API

Un autre problème se pose lors de la modification d'un fichier c'est que REST API ne peut pas accéder au fichier media donc à chaque fois qu'on fait une modification à un fichier il faut faire une requête GET pour avoir ce fichier puis le transformer en format lisible par JS puis l'envoyer à l'API via une requête PATCH pour qu'il puisse modifier les données. Cependant cette méthode ne marche pas car un BLOB(binary large object) n'est pas lisible sur REST API. Donc la modification des fichiers se fait sur une autre fenêtre ce qui n'est pas optimal pour l'expérience de l'administrateur.

Ainsi on revient au plus grand problème que j'ai rencontré lors de la conception du projet : Mon inexpérience

Puisque j'avais besoin de fournir un API puissant sur le front end notamment les scripts que j'ai écrit en JavaScript pour manipuler l'interface utilisateur et administrateur, normaliser les échanges entre l'interface et l'API via AJAX sur le backend j'avais besoin de plus de liberté avec la manipulation des données ou bien un Framework plus exhaustive que Rest (j'ai peut-être tort et juste pas su comment utiliser la technologie de manière efficace) **la meilleure technologie pour l'API devait être Express JS au lieu de Django REST API**.

Solution : utiliser une extension pour REST API qui permet de générer les formes HTML. Malheureusement j'avais pas assez de temps pour faire ce changement.

3- Sécuriser les fichiers media en utilisant X-Accel (Nginx)

L'un des aspects important en travaillant avec Django est de différencier entre les méthodes utilisées lors du développement et ceux utilisé en production. Notamment la façon avec laquelle on sert les fichiers media et les fichiers statiques : en développement on utilise Django cependant en production cette manière est non sécurisée et surtout non optimisée donc on laisse le serveur web faire. Pour le fichier statique le serveur web n'a pas besoin de passer par l'application pour les servir mais pour les fichiers media il faut passer par Django.

Ainsi afin de sécuriser les fichiers média contre accès non autorisés on utilise la méthode suivante

1- Redirection interne sur NGINX

```
location /protected/ {
    internal;
    alias /path/to/my/django/project/;
}
```

2- Vérification de l'autorisation sur Django et modification de la requête pour que NGINX puisse servir les fichiers media

```
def media_access(request, path):
    """
    When trying to access :
    myproject.com/media/uploads/image.png

    If access is authorized, the request will be redirected to
    myproject.com/protected/media/uploads/image.png

    This special URL will be handled by nginx with the help of X-Accel
    """

    access_granted = False

    user = request.user
    if user.is_authenticated():
        if user.is_staff:
            # If admin, everything is granted
            access_granted = True
        else:
            # For simple user, only their documents can be accessed
            user_documents = [
                user.identity_document,
                # add here more allowed documents
            ]

            for doc in user_documents:
                if path == doc.name:
                    access_granted = True

    if access_granted:
        response = HttpResponseRedirect()
        # Content-type will be detected by nginx
        del response['Content-Type']
        response['X-Accel-Redirect'] = '/protected/media/' + path
        return response
    else:
        return HttpResponseForbidden('Not authorized to access this media.')
```

Bien sûr il faut ajouter l'URL pour les fichiers media en utilisant une expression régulière :

```
urlpatterns = [..., url(r'^media/(?P<path>.*)', media_access, name='media'),]
```

CONCLUSION

Ce stage m'a permis de mettre en œuvre des compétences scolaire, humaines et professionnelle pour un sujet intéressant. De plus j'avais acquis de nouvelle compétence dans le domaine du développement web et DevOps.

À l'heure actuelle, l'application est prête à être utilisée. Ainsi je peux affirmer que le but qui m'avait été fixé a été atteint.

Cependant pour des raisons de sécurité des données des clients dû principalement à mon inexpérience j'aimerais que l'application reste comme prototype que d'autres développeurs avec plus expérience peuvent s'y référer.