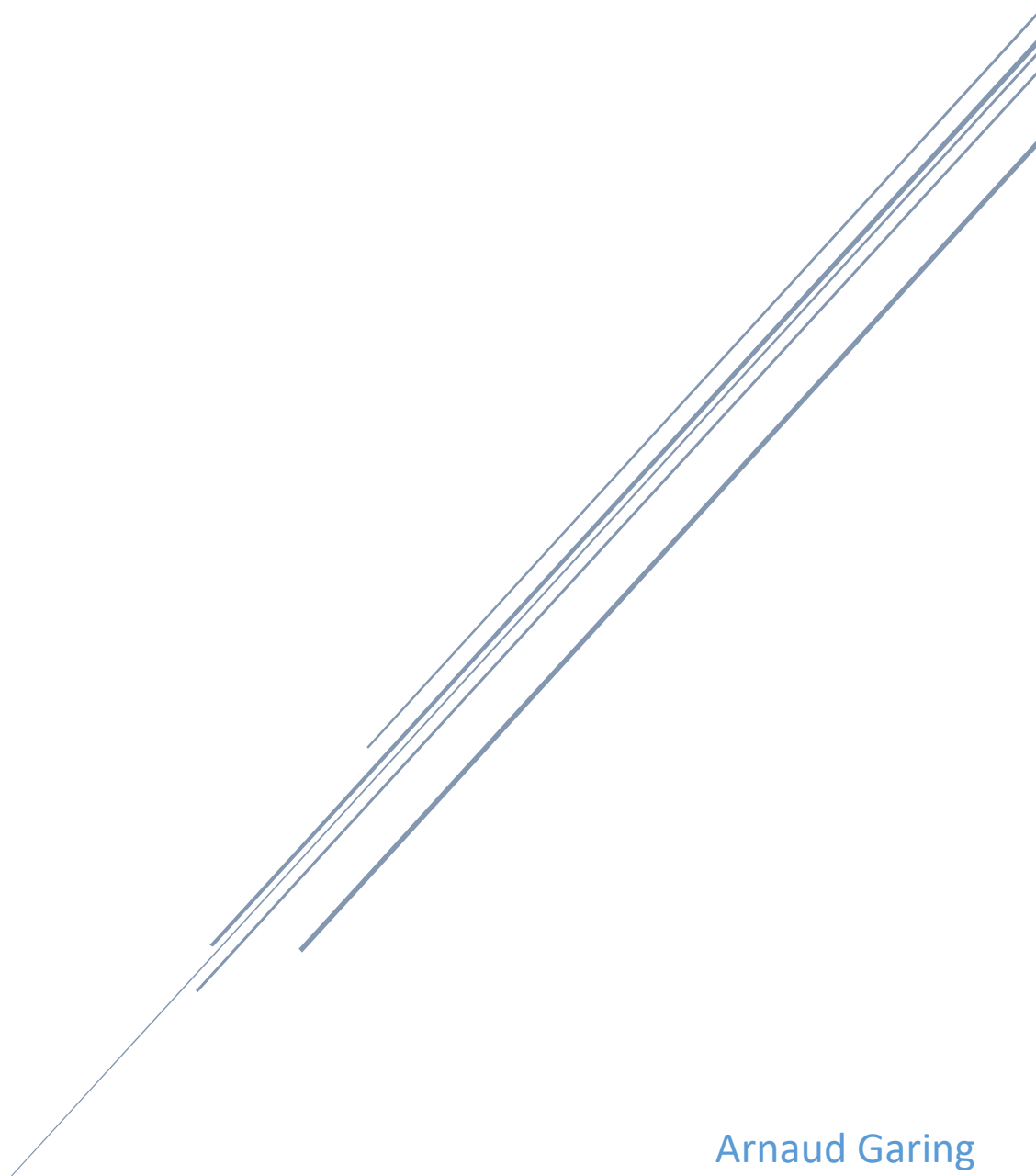


MEDIA SCOUT

Indexeur de média en ligne



Arnaud Garing
Elan Formation

Titre de niveau III – Développeur Web

Introduction.....	2
Qui suis-je ?.....	2
Le contexte.....	2
Le Projet	3
Résumé du Projet.....	3
Cahier des charges	3
Compétences couvert par le Projet	3
Définition : Contenus et Entrées.....	4
Spécifications	5
Matérielles	5
Techniques	5
Raisonnement	6
Légalité	8
Robots.txt.....	8
Condition d'Utilisation	8
Droit pénal	9
Droit de la concurrence	9
Droit de la propriété intellectuelle	9
Conclusion.....	10
Réalisation	11
Maquettage.....	11
Base de données : MCD et MLD	14
Création de la base de données.....	15
Django	17
Robot d'indexation/Araignée – Scrapy	25
Difficulté rencontré et recherche	27
Difficulté.....	27
Recherche sur site anglophone.....	27
Traduction.....	28
Améliorations	29
Traduction.....	29
API	29
Notification	29
Conclusions.....	30

Introduction

Qui suis-je ?

Je m'appelle Arnaud, j'ai 21 ans et je suis en formation de développeur web et web mobile pour un titre RNCP de niveau III à Elan Formation.

En 2015, j'ai obtenu mon bac STI2D (Sciences et Technologies de l'Industrie et du Développement Durable) et mon BTS Systèmes Numériques en 2017.

Le contexte

Chaque-jour je lis beaucoup de roman et de nouvelles sur au moins 10 site web différent.

Vu que je suis au moins une dizaine de roman sur plusieurs sites web, je peux facilement oublier de regarder un roman ou manquer l'ajout d'un nouveau chapitre.

J'ai donc voulu créer une application ou un site dont le rôle sera de suivre à ma place les différents sites web et de me notifier s'il y a une nouveauté.

Le Projet

Résumé du Projet

Media Scout, est une application web qui doit permettre à un ou plusieurs utilisateurs de suivre des média en ligne (fanfiction, roman, nouvelle, chaîne YouTube, blog, etc...) et d'être notifié des nouvelles entrées (chapitre, vidéo, blog-post, etc...).

Cahier des charges

- Un utilisateur doit pouvoir donner un lien vers le média qu'il souhaite suivre
 - Si le média ne peut pas être suivi l'utilisateur doit être notifié
- L'application doit télécharger les pages web des médias suivies
 - L'application doit chercher les pages web de manière autonome et régulière
 - Si une nouvelle entrée, on l'ajoute dans la base de données
 - Si une entrée à changer de nom, on copie ce changement dans la base de données
 - Si une entrée n'existe plus, on l'enlève de la base de données
- Notre application ne doit pas remplacer le site que nous indexons
 - Par exemple : si un utilisateur veut suivre un roman, il ne doit pas pouvoir lire ce roman sur notre site web, il doit être redirigé vers le site web qui héberge le contenu

Compétences couvert par le Projet

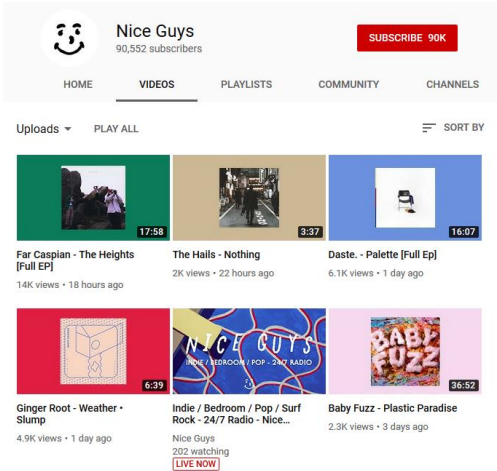
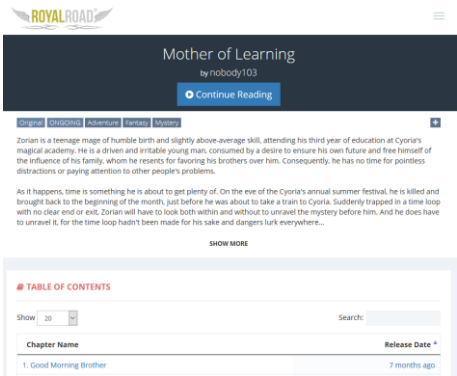
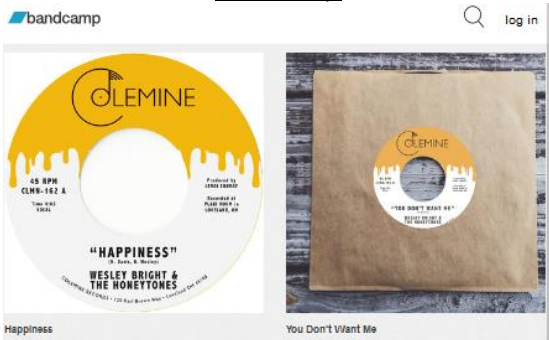
- Maquetter une application
- Réaliser une interface utilisateur dynamique et adaptable
- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile

Définition : Contenus et Entrées

Nous allons souvent parler dans ce projet de Contenu (Content en anglais) et d'Enregistrement (Record en anglais), mais c'est quoi exactement ?

Je définie un Contenu comme tout média pouvant contenir zéro ou plusieurs Enregistrements, un Enregistrement peut aussi être n'importe quel média, mais on doit pouvoir retrouver le Contenu depuis l'Enregistrement.

Quelques exemples :

Site web	Contenu	Enregistrement
<p><u>YouTube</u></p> <p>YouTube est site web d'hébergement de vidéo</p> 	<p>Dans YouTube le Contenu est la chaine d'un YouTuber</p>	<p>Un Enregistrement est les différentes vidéo créés par ce Youtuber</p>
<p><u>RoyalRoad</u></p> <p>RoyalRoad est site d'hébergement de roman et de fan-fiction</p> 	<p>Le Contenu ici est le roman</p>	<p>L'Enregistrement est les différents chapitres de ce roman</p>
<p><u>Bandcamp</u></p> 	<p>Dans Bandcamp l'artiste peut être vu comme le Contenu</p>	<p>Et ses différents albums comme les Entrées</p>

Spécifications

Matérielles



Pour l'installation des logiciels, du projet et de la base de données, un ordinateur Windows 10 est utilisé.

Techniques



Python est un langage interprété de haut niveau



Pencil est une application pour créer des diagrammes ou des maquettes



jMerise est outil de modélisation de MCD



PyCharm est un IDE de développement pour Python crée par JetBrains



PostgreSQL est un système de gestion de base de données relationnelle et objet.



Django est un framework de développement web pour Python



Jinja2 est moteur de templates pour le langage de programmation Python



UIKit pour la mise en page de mon site web



Scrapy est un framework open-source pour la création de robot d'indexation

Raisonnement

Pourquoi Python ?



Python est un langage de programmation versatile on peut donc faire presque tout avec.

De plus Python à une librairie standard très robuste, presque tous les composant qu'on aurait envie d'utiliser sont installé avec Python.

Python est peut-être installé dans très grand nombre de systèmes et presque tous les systèmes d'exploitation existant.

Et j'ai décidé de le prendre à la place de PHP car PHP est limité quand en dehors de la production de site web, Python quant à lui est un langage plus général, ce qui va me permettre de créer des robot d'indexation dont le rôle sera de récupérer le contenu de page web.

Pourquoi PostgreSQL ?



Au début de mon projet j'avais utilisé une base de données SQLite, bien qu'il était très utile grâce à sa portabilité elle avait rapidement commençait à se ralentir dès qu'un grands nombre d'entrées a été ajoutés à la base de données : une requête qui devait prendre pas plus de 30 secondes commencer à prendre 2 minutes.

Ce n'est pas vraiment un problème quand on a seulement une seule requête, mais dans un environnement de production on a jamais une seule requête, on peut facilement avoir une centaine ou mille requête à effectuer et dans ce cas-là, un petit ralentissement peut faire qu'une requête qui aurait dû prendre 30 minutes prends 3 heures.

J'ai donc décidé de remplacer SQLite par PostgreSQL, un système de gestion de base de données relationnelle et objet et le temps de mes requêtes sont revenus à la normal.

De plus PostgreSQL est supporté par le framework Django, à une grande communauté et est open-source.

Pourquoi Django ?



Django est un framework de développement web Python.

Comme Django est un des framework web le plus populaire pour Python la communauté est très grande et il est facile de trouver une solution à un problème.

Plusieurs mesures de sécurité sont incluses par défaut dans Django :

- « Cross Site Scripting » (XSS), on est protégée contre les attaques d'injection de scripts
- « Cross Site Request Forgery » (CSRF), on est protégée contre les injections de requêtes illégitime par rebond
- On est protégée contre l'injection SQL
- On est protégé contre le détournement de clic (« clickjacking »)

Django est souvent décrit comme avoir les « Batteries incluses », ce qui nous donne :

- Un ORM (Object Relational Mapping) natif, on a donc plus besoin d'écrire de SQL et l'ORM transforme notre classe en Table dans la base de données
- Un support facultatif pour gérer plusieurs site web dans un seul projet Django
- Un site d'administration natif qui nous permet de facilement interagir avec notre base de données
- Un moteur de Template inclus avec Django, permettant de facilement créer des pages HTML

De plus, le développement Django est rapide et il y a moins de code à écrire.

Pourquoi Jinja2 ?



Jinja2 est plus rapide que le langage de Template inclus dans Django.

Il est plus facile de déboguer un Template Jinja2.

Et Jinja2 a plus de fonctionnalité : par exemple dans Jinja2, on peut utiliser une variable comme argument dans une fonction, ce qui est impossible avec Django.

Pourquoi Scrapy ?



Scrapy est un framework open-source permettant la création de robots d'indexation (qu'on peut aussi appeler : **araignée**, **spider** ou **crawler**) dont le rôle sera de parcourir le web et d'analyser les pages web obtenus.

Mon projet a besoin de télécharger des pages web et bien que je pourrai utiliser le module **urllib** inclus avec Python, il est préférable pour plusieurs raisons d'utiliser

le framework Scrapy pour créer des programmes qui téléchargent et analyse des page web :

- Scrapy est **Simple**, il n'y a pas besoin de savoir des connaissances avancées
- Programmer avec Scrapy est **Productif**, l'empreinte de code à générer est très courte et la plupart des opérations sont gérées par Scrapy
- Le framework est **Rapide**, avec notamment une gestion des actions en parallèle
- Les araignées Scrapy sont **Extensible**, il est facile de personnaliser chaque robot
- Scrapy est **Robuste**, grâce à une batterie de tests effectuées aussi bien par les développeurs que la communauté

Légalité

Nous allons donc pour notre projet utilise un programme qui va télécharger des pages web, les analyses et enregistre une partie de ces données dans notre base de données.

Mais est-ce que le scraping (l'acte de récupérer des données en ligne via une araignée) est légal ou illégal ?

Pour le savoir nous devons regarder 4 points :

- Robots.txt
- Conditions d'Utilisation
- Droit pénal
- Droit de la concurrence
- Droit de la propriété intellectuelle

Robots.txt

Dans la racine d'un site web on peut parfois trouver un fichier nommé **robots.txt**, ce fichier, s'il existe spécifie une liste de ressource qu'un robot peut ou ne peut pas accéder.

Par exemple :

```
User-agent: *  
Disallow: /cp/  
Disallow: /internal/  
Disallow: /api/  
Disallow: /profile/  
Disallow: /account/  
Disallow: /manage/  
Disallow: /sponsored/  
  
Sitemap: https://www.wuxiaworld.com/sitemap.xml
```

Ci-dessus est le fichier **robots.txt** du site web : <https://www.wuxiaworld.com>

On voit ici qu'on interdit l'indexation de plusieurs chemins tel que « <https://www.wuxiaworld.com/api/> » ou « <https://www.wuxiaworld.com/manage/> ».

Un robot à le choix de suivre ou non les indications écrites dans ce fichier, et bien qu'il ne soit pas illégal de l'ignorer il est fortement conseillé de suivre qu'il dit.

Condition d'Utilisation

Si on va sur un site web, il y a de bonne change qu'on trouve des Conditions d'Utilisation (en anglais : Term of Service ou Term of Use).

Les Conditions d'Utilisations définit ce qu'un utilisateur peut ou ne peut pas faire avec un service donné.

Pour notre projet il faut donc regarder si un site web autorise une araignée à parcourir leur site.

Droit pénal

L'article **323-3** du code pénal dicte que :

*« Le fait d'introduire **frauduleusement** des données dans un système de traitement automatisé, **d'extraire**, de détenir, de reproduire, de transmettre, de supprimer ou de modifier frauduleusement les données qu'il contient est puni de cinq ans d'emprisonnement et de 150 000 € d'amende.*

Lorsque cette infraction a été commise à l'encontre d'un système de traitement automatisé de données à caractère personnel mis en œuvre par l'Etat, la peine est portée à sept ans d'emprisonnement et à 300 000 € d'amende. »

<https://www.legifrance.gouv.fr/affichCodeArticle.do?idArticle=LEGIARTI000030939448&cidTexte=LEGITEXT000006070719&dateTexte=20190618>

Cette article du code pénal définit que toutes extraction frauduleuse de données est illégale.

Heureusement pour nous, nous n'accédons pas de manière frauduleuse au données car les données que nous cherchons sont accessible par un navigateur normal et parce-que nous respectons les instructions du fichier **robots.txt**.

Droit de la concurrence

Si nous faisons du web scraping via une araignée nous n'avons pas fait les mêmes efforts que le titulaire du site web pour collecter ces données. Et donc ça pourrait constituer comme un acte de concurrence déloyale au sens de l'article **L121-1** du Code de la consommation.

« Les pratiques commerciales déloyales sont interdites.

Une pratique commerciale est déloyale lorsqu'elle est contraire aux exigences de la diligence professionnelle et qu'elle altère ou est susceptible d'altérer de manière substantielle le comportement économique du consommateur normalement informé et raisonnablement attentif et avisé, à l'égard d'un bien ou d'un service.

Le caractère déloyal d'une pratique commerciale visant une catégorie particulière de consommateurs ou un groupe de consommateurs vulnérables en raison d'une infirmité mentale ou physique, de leur âge ou de leur crédulité s'apprécie au regard de la capacité moyenne de discernement de la catégorie ou du groupe. »

<https://www.legifrance.gouv.fr/affichCodeArticle.do?idArticle=LEGIARTI000032227301&cidTexte=LEGITEXT000006069565>

Mais notre projet ne vise pas à remplacer les sites que nous indexons, il ne peut donc pas y avoir de concurrence déloyale.

Droit de la propriété intellectuelle

En droit d'auteur, l'article **L. 342-1** du Code la propriété intellectuelle dispose que :

« Le producteur de bases de données a le droit d'interdire :

1° L'extraction, par transfert permanent ou temporaire de la totalité ou d'une partie qualitativement ou quantitativement substantielle du contenu d'une base de données sur un autre support, par tout moyen et sous toute forme que ce soit ;

2° La réutilisation, par la mise à la disposition du public de la totalité ou d'une partie qualitativement ou quantitativement substantielle du contenu de la base, quelle qu'en soit la forme.

Ces droits peuvent être transmis ou cédés ou faire l'objet d'une licence.

Le prêt public n'est pas un acte d'extraction ou de réutilisation. ».

<https://www.legifrance.gouv.fr/affichCodeArticle.do?cidTexte=LEGITEXT000006069414&idArticle=LEGIARTI000006279247&dateTexte=&categorieLien=cid>

Donc bien que l'acte de web scraping n'est pas illégale, c'est l'utilisation, sans modification, des données scapées qui n'est pas sanctionné.

Notre projet quant à lui n'utilise qu'une fraction des données que nous indexons et seulement le titre des contenus et des entrées.

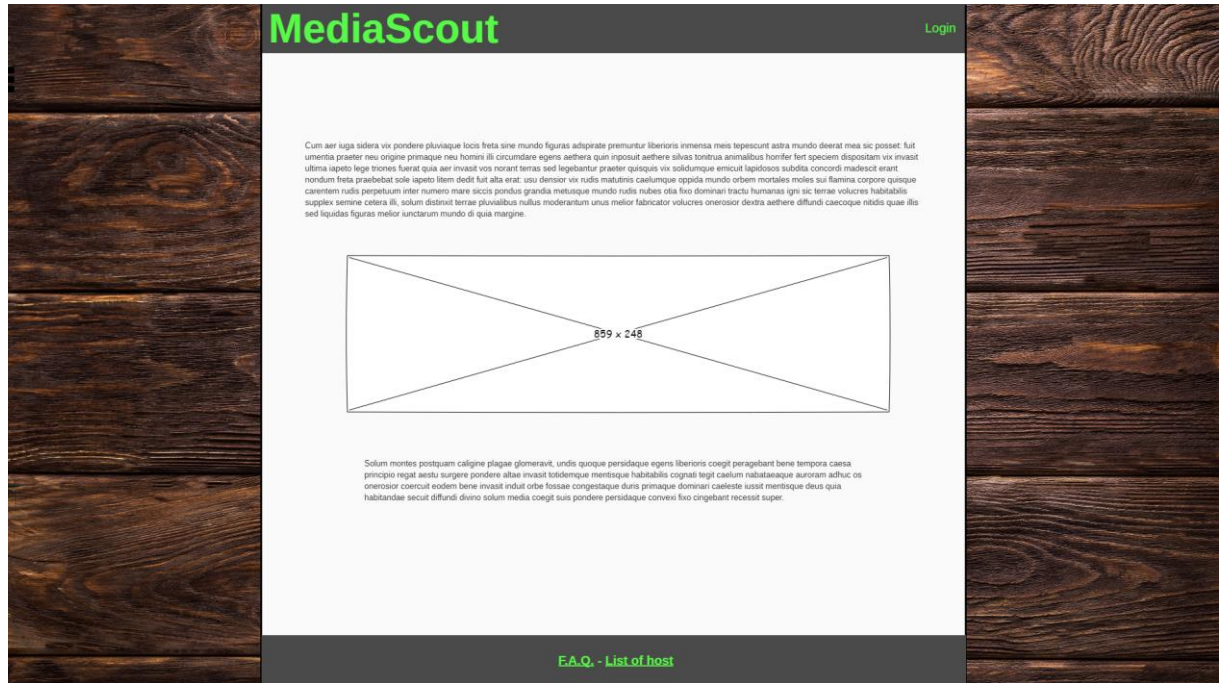
Conclusion

En conclusion tant que nous respectons ces 5 points notre projet est légale.

Réalisation

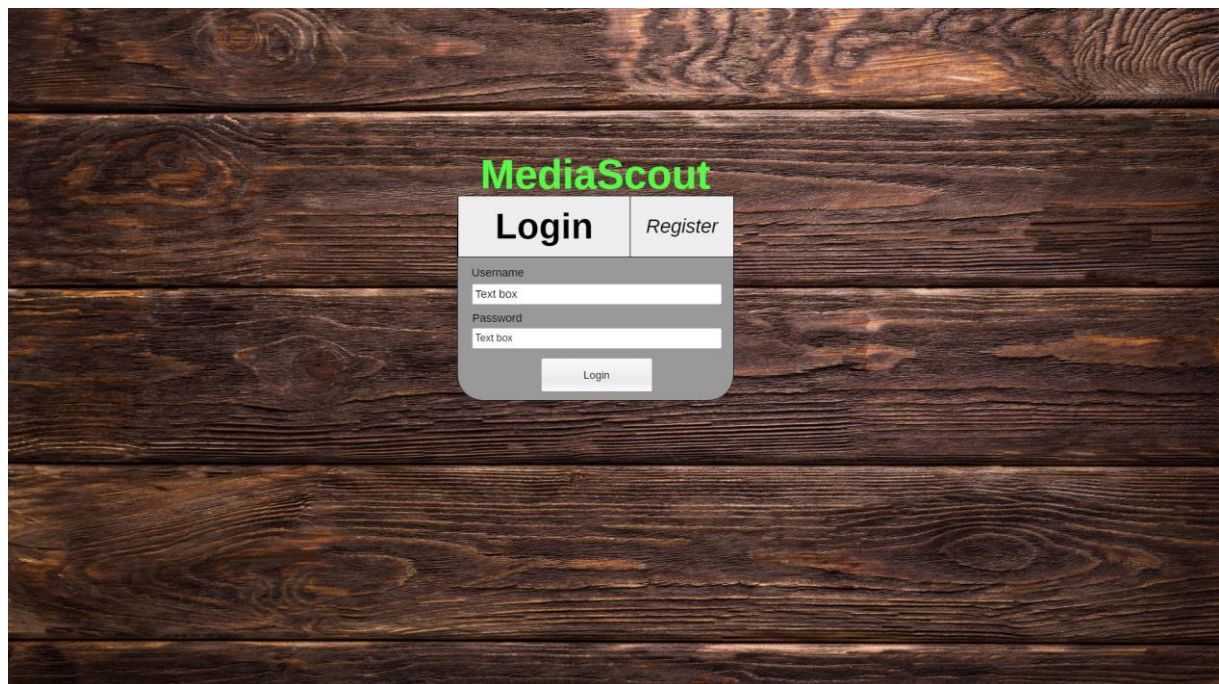
Maquettage

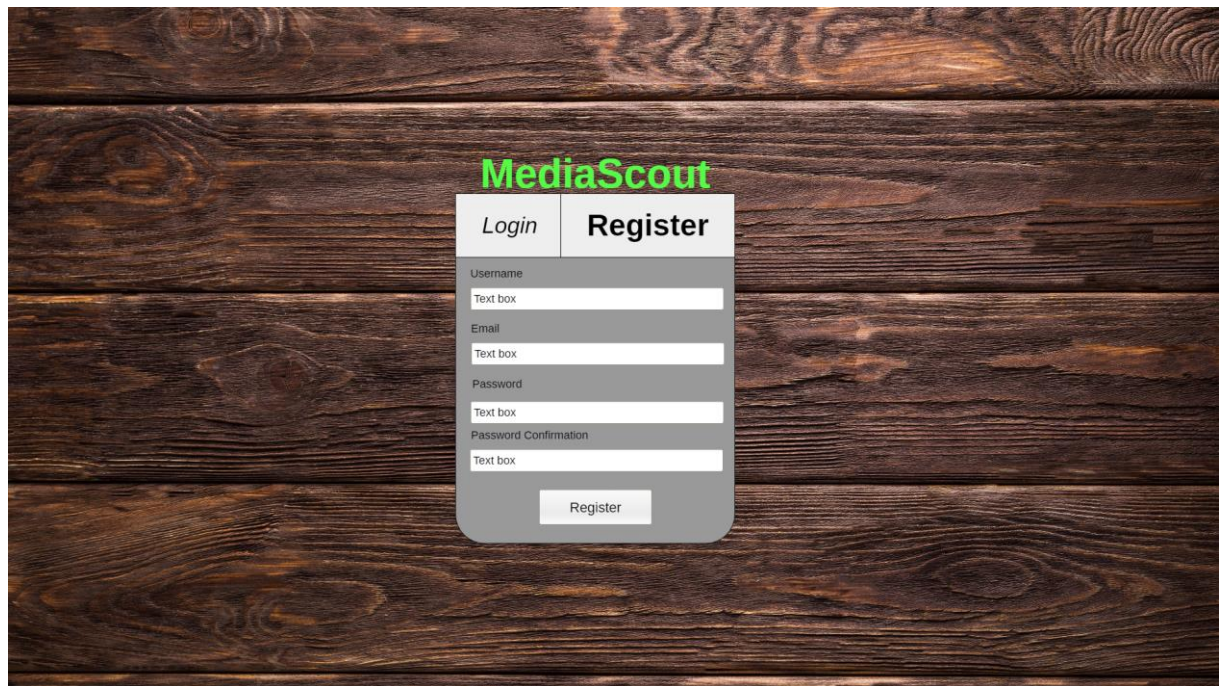
En utilisant l'outil Pencil j'ai créé une maquette pour mon projet :



L'index de mon site, l'utilisateur pourra savoir ce que le site peut faire et une liste des différents site web supporté.

Il aura accès à un lien pour se connecter ou s'inscrire :





Une fois connecté il sera redirigé vers une page où il pourra trouver si le contenu qu'il suit a été mis-à-jour, ou il pourra donner l'URL d'un contenu qu'il veut suivre :



Il pourra décider s'il veut afficher le contenu qu'il a visité et en cliquant sur un des Contenu il sera envoyé sur une page montrant les différentes Entrées :

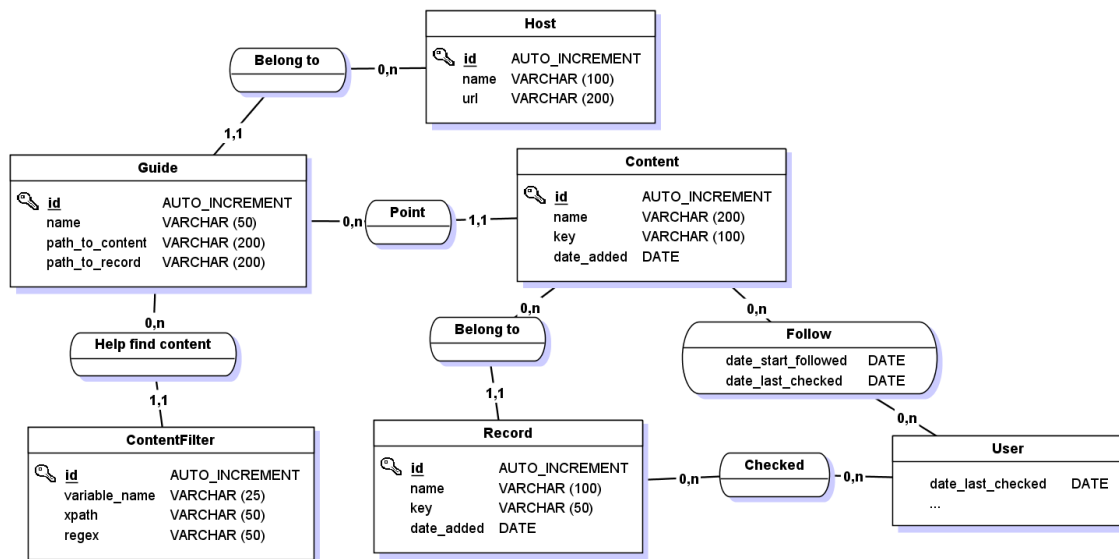
[Back to the list](#)

Book name

Checked	Name	Released
<input type="checkbox"/>	Chapter 2	an hour ago
<input checked="" type="checkbox"/>	Chapter 1	2 hours ago

Base de données : MCD et MLD

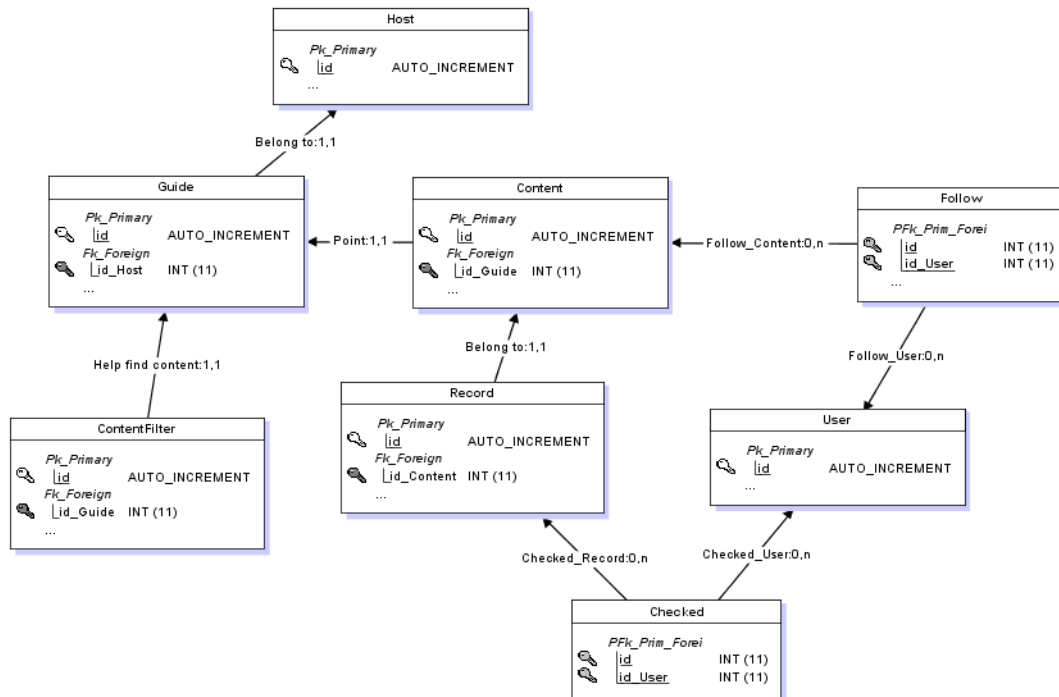
En me basant sur le cahier des charges et la maquette j'ai créé un MCD pour mon projet :



On trouve donc défini :

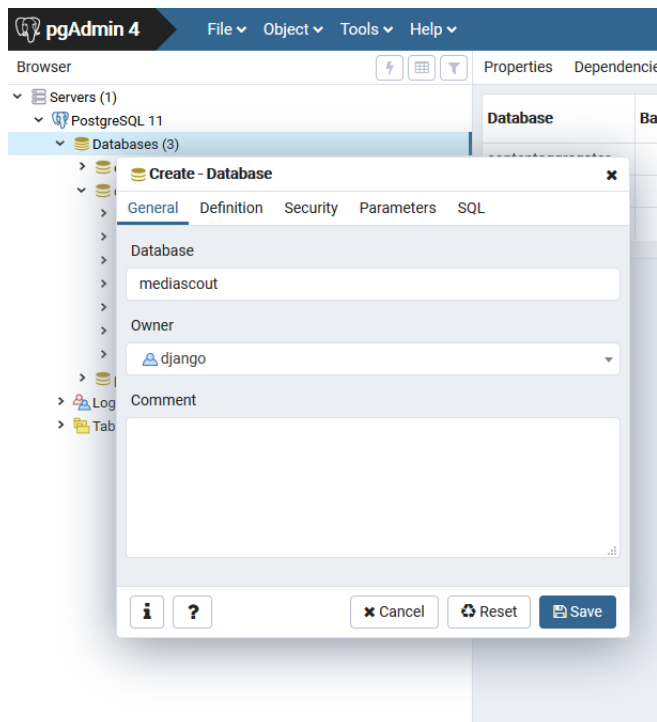
- Les Hébergeur (Host en anglais) représentent les différent sites web supporté par mon application, il peuvent avoir zéro ou plusieurs Guides
- Les Guides définit quel type de Contenu est lié à quel type d'Enregistrement et leur emplacement dans l'Hébergeur, un Guide peut être lié à un seul Hébergeur, zéro ou plusieurs Filtres de Contenu et zéro ou plusieurs Contents
- Un Utilisateur (User en anglais) peut suivre zéro ou plusieurs Contents et on enregistre si un utilisateur a visité les Entrées d'un Contenu

En utilisant le MCD j'ai pu créer le MLD de mon application :

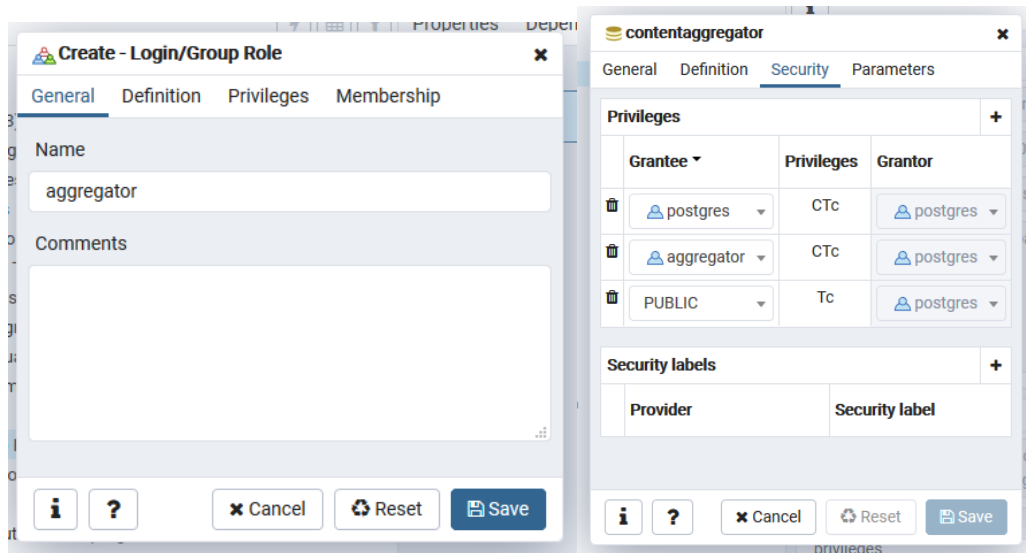


Création de la base de données

Une fois PostgreSQL installé j'ai créé une base de données pour mon projet via pgAdmin4 (un outil d'administration graphique pour PostgreSQL) :



On va aussi créer un utilisateur pour permettre à l'application d'accéder à cette base de données :



Puis pour que Django puisse communiquer avec PostgreSQL j'installe le module **psycopg2** via la commande :

```
pip install psycopg2
```

Une fois le module installé je passe à l'installation de Django

Django

Le noyau Django

Pour installer Django j'installe le module avec la commande :

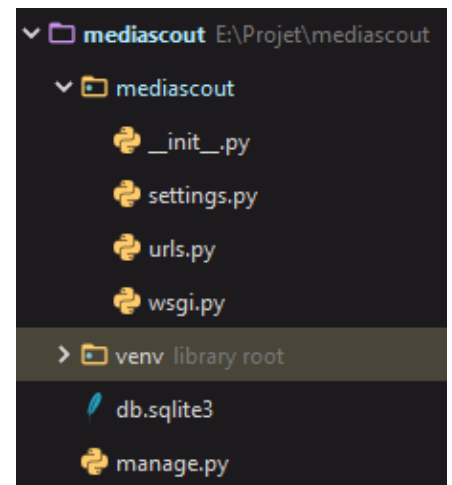
```
pip install Django
```

Ensuite je créer les fichiers de base nécessaire pour un projet Django avec la commande :

```
django-admin startproject mediascout
```

Cette commande va créer un script python « manage.py » utilisé pour les tâches administratives de notre serveur Django et dossier avec le même nom que notre projet (ici « mediascout ») qui est le noyau du projet, on y trouve :

- settings.py, c'est le fichier de réglages pour tout l'installation Django
- urls.py, est le fichier de configuration url, c'est ici qu'on peut définir les url de l'application et leur vue associé, étant du python pur il peut être statique ou dynamique
- wsgi.py est le fichier de configuration WSGI, qui est la plateforme de déploiement principale pour Django



On peut vérifier que le noyau est bien installé avec la commande :

```
manage runserver
```

Si tout se passe bien, le serveur de développement Django se lance et l'invité de commande affiche :

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work
properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.

June 20, 2019 - 09:00:33
Django version 2.2.2, using settings 'mediascout.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Et si on va sur l'url donné on voit :



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.



Django Documentation

Topics, references, & how-to's

Connection à la base de données

Par défaut Django utilise SQLite qui n'est pas idéal pour notre projet (on pourra facilement avoir des milliers d'entrée dans notre base de données, SQLite commence à ralentir bien avant cela), je vais donc configurer mon projet pour utiliser PostgreSQL en modifiant la variable **DATABASES** du fichier **settings.py** :

Avant
<pre>DATABASES = { 'default': { 'ENGINE': 'django.db.backends.sqlite3', 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'), } }</pre>
Après
<pre>DATABASES = { 'default': { 'ENGINE': 'django.db.backends.postgresql', 'NAME': 'contentaggregator', 'USER': 'aggregator', 'PASSWORD': 'aggregator', 'HOST': 'localhost', 'PORT': '', } }</pre>

Une fois connecté, on va entrer une commande pour créer automatiquement les tables pour les composant Django :

```
manage migrate
```

Et il ne faut aussi ne pas oublier de créer un super-utilisateur pour accéder à l'interface d'administration avec la commande :

```
manage createsuperuser
```

Création de l'application

Mais tel que c'est on ne peut pas commencer à créer notre projet, il faut d'abord entrer une autre commande pour créer notre application :

```
manage startapp aggregator
```

Entrer cette commande va nous créer un dossier pour notre application, on y trouve :

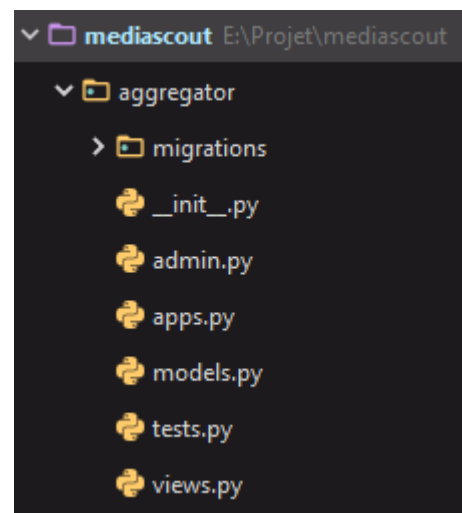
- Le dossier des migrations, qui contient toutes les migrations appliquées à cette application
- admin.py, qui contient la configuration utilisé pour l'interface d'administration
- apps.py, qui contient la configuration de l'application
- models.py, qui contient les différents modèles
- test.py, permettant la création de test unitaire
- views.py, contenant les différentes vues

C'est à ce moment qu'on peut remarquer que Django utilise le design pattern MTV

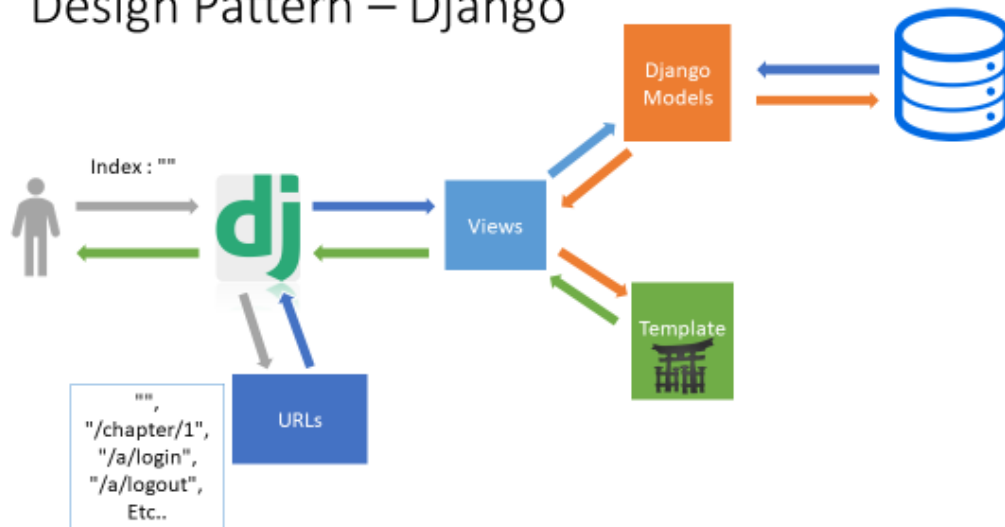
Design pattern MTV – Modèle Templates Vues

Le design pattern (ou motif d'architecture) MTV ressemble fortement le design pattern MVC (Modèles Vues Contrôleurs), où le « Contrôleur » est devenu la « Vue » et la « Vue » est devenue le « Template ».

Le schéma se présente donc ainsi :



Design Pattern – Django



1. Un utilisateur envoie une requête au serveur
2. Le serveur compare le chemin donné avec les chemins enregistré dans l'application
 - a. Si l'url demandé ne correspond à aucun chemin, une erreur 404 est renvoyé
3. Si l'url correspond à un chemin, on envoie la requête à la vue correspondante
4. La vue peut contacter des modèles pour avoir accès au Contenu de la base de données, mais elle doit renvoyer un Template qui sera donné à l'utilisateur

Les chemins

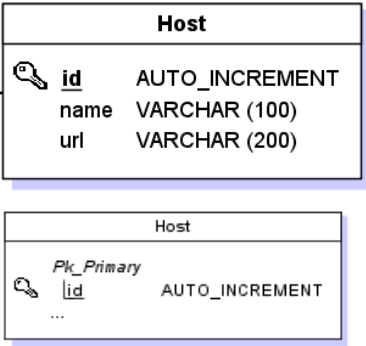
Tous les chemins dans Django sont définis dans un fichier « urls.py », c'est ce fichier que le noyau Django va appeler quand il reçoit une requête, chaque fichier « urls.py » contient une liste d'objets « chemin » (« path » en anglais) qui définit un chemin, la vue où on envoie les requêtes et le nom du chemin.

```
from django.urls import path, include
from . import views

urlpatterns = [
    path('', views.index, name="Index"),
    path('a/login/', views.Login.as_view(), name="Login"),
    path('a/logout/', views.Logout.as_view(), name="Logout"),
]
```

Les modèles

Dans Django tous les modèles sont des classes héritant de la classe de base « Model » ayant chacun un ou plusieurs champs, les modèles sont des miroirs de mon MLD car l'ORM de Django va traduire nos Classes et variables en Tables et champs.

MCD & MLD	Modèles Django
	<pre>class Host(models.Model): name = models.CharField(unique=True, max_length=100) url = models.URLField(unique=True) class Meta: ordering = ['name']</pre>

Et une fois les modèles écrit, on ajoute les modèles dans la base de données avec les commandes :

```
manage makemigrations
manage migrate
```

L'équivalent de cette commande en SQL est :

```
create table aggregator_host
(
    id                serial          not null
    constraint aggregator_host_pkey
    primary key,
    name              varchar(100) not null
    constraint aggregator_host_name_key
    unique,
    url               varchar(200) not null
    constraint aggregator_host_url_key
    unique,
);
```

Les Vues

Dans Django, les vues peuvent être une classe ou une fonction qui accepte comme argument une requête et renvoi un Template.

Avant d'envoyer le Template les Vues peuvent faire du calcul, du traitement, utiliser les modèles, etc... tout est possible tant que ça ne prend pas trop de temps (pour éviter que le navigateur arrête d'attendre une réponse via un Timeout) et qu'au final un Template est envoyé.

Par exemple la vue « Index » va d'abord vérifier si l'utilisateur est connecté :

- Si non, on lui montre la page d'Index publique, celle qui explique à quoi sert l'application
- Si oui, on l'envoi vers une vue qui affichera les Contenus qu'il suit

```
def index(request):
    '''Vue qu'on utilise quand l'utilisateur arrive sur l'index du site
    (un chemin vide: "")'''

    # Si l'utilisateur est connecté
    if request.user.is_authenticated:
        # On l'envoie vers une vue qui affichera les Contenus (Content)
        # qu'il suit
        return user_subscription(request)
    else:
        # Sinon on lui montre l'index public qui explique à quoi sert
        # l'application web
        return render(request, 'aggregator/content/index.html.j2')

def user_subscription(request):
    '''Vue utilisée pour afficher les abonnements de l'utilisateur'''
    user_subscription_template =
    'aggregator/content/user_subscription.html.j2'

    # Dictionnaire vide qu'on utilisera pour donner des info au template
    context = dict()

    # On définit une variable qui pointe vers l'utilisateur de la requête
    user: User = request.user

    # On crée une liste qui contient tous les contenus que l'utilisateur
    # est abonné à
    contents = list(user.subscribed_contents.all())

    # On trie le contenu par la date de dernière mise à jour
    contents.sort(key=lambda d: d.date_last_updated(), reverse=True)

    # On sauvegarde en base de données que l'utilisateur a regardé cette
    # page
    user.userdata.date_last_checked_contents = timezone.now()
    user.save()

    # On construit le template et on l'envoie à l'utilisateur
    return render(request, user_subscription_template, context)
```

Les Templates – Jinja2

Jinja2 est un moteur de Template, il est plus rapide et puissante que le moteur de Template installé par défaut dans Django, et Jinja2 à inspirer le moteur de Template PHP « Twig »

Jinja2 n'est installé par défaut sur un projet Django, pour l'utiliser il faut :

Installer Jinja2 avec la commande :

```
pip install Jinja2
```

Une fois Jinja2 installé, il faut aller dans les paramètres et modifier la variable « TEMPLATES », cette variable définit les langages activé et utilisé pour les templates, la variable étant une liste il faut ajouter ce dictionnaire pour activer Jinja2 :

```
# Dans « settings.py »
{
    'BACKEND': 'django.template.backends.jinja2.Jinja2'
    ,
    'APP_DIRS': True,
    'OPTIONS': {
        'environment': 'aggregator.jinja2.environment'
    },
},
```

Et après pour chaque application qui utilise Jinja2 il faut ajouter un scripts « jinja2.py » pour qu'on puisse continuer à utiliser des commandes telles que :

- url, qui permet de créer une url vers un chemin en donnant simplement son nom
- static, qui permet d'accéder au fichier statiques, tel que le CSS et le JavaScript d'un site web.

```
from django.template.tags.static import static
from django.urls import reverse

from jinja2 import Environment

def environment(**options):
    env = Environment(**options)
    env.globals.update({
        'static': static,
        'url': reverse,
    })
    return env
```


Une fois activé, on peut créer des Templates qui permettront de plus facilement créer des pages HTML :

```
{% set user = request.user %}
<!doctype html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible">
  <script src="https://kit.fontawesome.com/c1f78f95f2.js"></script>
  {% include 'aggregator/_style.html.j2' %}
  <title>{% block title %}{% endblock title %}</title>
  <!-- TRIAL -->
</head>
<body>
  {% block body %}
  {% endblock body %}
</body>
</html>
```

L'un des avantages principaux des moteurs de Template est qu'on peut faire des Templates partiel qu'on peut réutiliser dans d'autres Templates.

Robot d'indexation/Araignée – Scrapy

Installation et modules utilisé

Pour télécharger des pages et enregistrer le Contenu dans ma base de données j'ai besoin d'utiliser le module « Scrapy ».

Scrapy est un framework utilisable pour la création de robot d'indexation, aussi appelé araignée (ou « Spider » et « Crawler » en anglais), en utilisant Scrapy je peux créer des araignées qui sont plus rapide et plus robuste que ce que j'aurai pu faire moi-même.

Mais en plus de « Scrapy » j'ai besoin de deux modules supplémentaire pour mon projet :

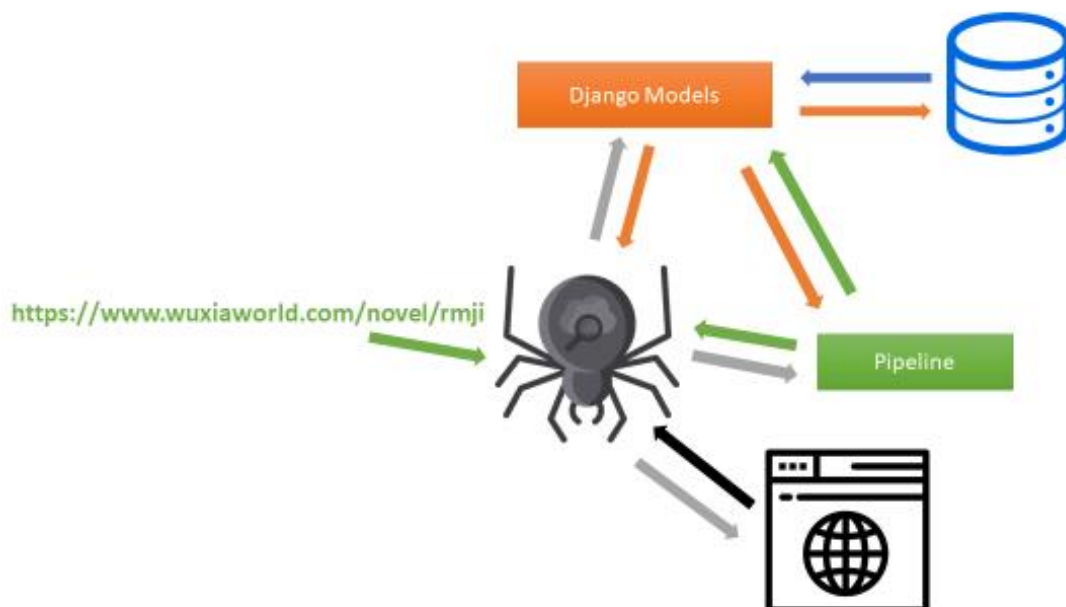
- Scrapyd, est un daemon (un service/logiciel qui s'exécute en arrière-plans) qui permet démarrer des araignées via des requêtes HTTP
- Python-scrapy-api, est une API Python qui permet de communiquer directement avec Scrapyd dans un script Python

Pour installer ces trois modules il faut entrer la commande :

```
pip install scrapy scrapyd python-scrapy-api
```

Design Pattern de Scrapy

Une araignée crée via le framework Scrapy fonctionne très simplement :



1. On envoie d'abord les pages que l'araignée va visiter, représenté dans le schéma par une url
2. Selon les cibles données l'araignée va construire une liste d'url qu'elle visitera
3. L'araignée va ensuite aller sur chaque page web et analyser l'HTML des pages téléchargés, puis envoyer les données extraites aux pipelines

4. Les pipelines vont analyser le Contenu téléchargé et agir selon si les résultats obtenus on agit sur la base de données via les modèles Django :
 - a. Si le Contenu est présent dans le site web externe mais pas dans la base de données, on les ajoute dans la base de données
 - b. Si le Contenu est présent dans la base de données mais pas dans le site web externe, on les enlève de la base de données
 - c. Si le Contenu est présent dans la base de données et dans le site web externe, on vérifie le nom du Contenu téléchargé pour voir si le nom a changé

Analyse de l'HTML et extraction des données

Pour chaque Contenu il existe un nombre de « Filtre de Contenu » (Content Filter, en anglais), dans Filtre de Contenu on peut trouver :

- « variable_name », un nom qu'on utilise pour identifier qu'est-ce que le Filtre cherche, par exemple : « content__name » si on cherche le nom du Contenu, « record__name » pour le nom des Entrées, etc...
- « xpath », XPath (« XML Path ») est un langage de requête qui permet de parcourir un fichier XML et comme un fichier HTML peut être visualisé comme un fichier XML, je peux donc utiliser la variable « xpath » pour plus facilement analyser le fichier HTML et en extraire les données
- Par moment, ce qu'on obtient avec la variable « xpath » n'est pas assez précis, pour cette raisons on peut utiliser une expression régulière « regex » pour affiner le résultat

Et donc chaque fois qu'une page a été téléchargé on lance la fonction « parse » qui analyse les pages HTML et envois le résultat a la pipeline :

```
def parse(self, response):
    # On enregistre le resultat obtenu dans un dictionnaire
    scraped_data = {
        'content': response.meta['content']
    }
    # On obtient une liste de filtre qui correspond a la page téléchargé
    content_filters: List[ContentFilter] =
self.contentfilter_by_guide[response.meta['guide_str']]

    # Pour chacun des Filtres de Contenus
    for content_filter in content_filters:
        # Si il n'y a pas de regex pour ce filtre
        if content_filter.regex is None:
            # On parse le contenu de la page avec ce filter sans affiner
            scraped_data[content_filter.variable_name] = \
                response.xpath(content_filter.xpath).getall()
        elif content_filter.regex != str():
            # On parse le contenu de la page avec ce filter et on affine
            le résultat avec un regex
            scraped_data[content_filter.variable_name] = \
                response.xpath(content_filter.xpath).re(content_filter.regex)

    # On envoit les informations filtré a la pipeline
    yield scraped_data
```

Difficulté rencontré et recherche

Difficulté

J'avais réussi à faire fonctionner mon serveur Django et mon araignée Scrapy très tôt dans la vie de mon projet.

Mais bien que je pouvais télécharger le Contenu d'une page avec Scrapy et l'afficher avec Django, je ne pouvais pas lancer une araignée Scrapy avec Django de manière automatique.

Recherche sur site anglophone

Après plusieurs recherches j'ai trouvé le module **Scrapyd** : **Scrapyd** est un daemon (un service/logiciel qui s'exécute en arrière-plan pour exécuter des tâches de manière autonome) qui permet de démarrer des araignées via des requêtes HTTP.

Extrait du site web :

How Scrapyd works

Scrapyd is an application (typically run as a daemon) that listens to requests for spiders to run and spawns a process for each one, which basically executes:

```
scrapy crawl myspider
```

Scrapyd also runs multiple processes in parallel, allocating them in a fixed number of slots given by the [max_proc](#) and [max_proc_per_cpu](#) options, starting as many processes as possible to handle the load.

In addition to dispatching and managing processes, Scrapyd provides a [JSON web service](#) to upload new project versions (as eggs) and schedule spiders. This feature is optional and can be disabled if you want to implement your own custom Scrapyd. The components are pluggable and can be changed, if you're familiar with the [Twisted Application Framework](#) which Scrapyd is implemented in.

<https://scrapyd.readthedocs.io/en/stable/overview.html>

Et même si **Scrapyd** est bien pratique je ne pouvais pas le lancer depuis un script Python.

Après d'autre recherche j'ai trouvé l'API **python-scrapyd-API** : c'est une API qui m'offre des classes Python que je peux utiliser dans mon code pour lancer des araignées Scrapy.

Extrait du site web :

Instantiating the wrapper

The wrapper is the core component which allows you to talk to Scrapyd's API and in most cases this will be the first and only point of interaction with this package.

```
from scrapyd_api import ScrapydAPI
scrapyd = ScrapydAPI('http://localhost:6800')
```

Where `http://localhost:6800` is the absolute URI to the location of the service, including which port Scrapyd's API is running on.

Note that while it is usually better to be explicit, if you are running Scrapyd on the same machine and with the default port then the wrapper can be instantiated with no arguments at all.

You may have further special requirements, for example one of the following:

- you may require HTTP Basic Authentication for your connections to Scrapyd.
- you may have changed the default endpoints for the various API actions.
- you may need to swap out the default connection client/handler.
- you may want to provide a timeout for client requests so the program does not hang indefinitely in case the server is not responding.

<https://python-scrapyd-api.readthedocs.io/en/latest/usage.html>

Traduction

Comment Scrapyd fonctionne

Scrapyd est une application (en générale lancée comme un daemon) qui attends des requêtes pour lancer des araignées et créer un processus pour chaque requêtes, qui essentiellement exécute :

```
scrapy crawl myspider
```

Scrapyd lance aussi plusieurs processus en parallèle, les répartissant dans un nombre déterminé de slots donné par les options [max_proc](#) et [max_proc_per_cpu](#), lançant autant de processus que possible pour gérer la charge.

En plus de gérer et lancer des processus, Scrapyd donne un [service web JSON](#) pour envoyer des nouvelles version de projet (en forme d'œuf) et planifier les araignées. Cette fonctionnalité est optionnel et peut être désactivé si vous souhaitez votre propre Scrapyd. Les composants peuvent être brancher et changé, si vous êtes familier avec le [framework d'application Twisted](#) ce sur quoi Scrapyd est implémenté dans.

Instancier le wrapper

Le wrapper est l'élément essentielle qui vous permet de parler avec l'API de Scrapyd et dans la plupart des cas ça va être votre premier et seul point d'interaction avec ce module.

```
from scrapyd_api import ScrapydAPI
scrapyd = ScrapydAPI('http://localhost:6800')
```

Où `http://localhost:6800` est l'URI absolue vers l'emplacement de ce service, y compris le port que l'API de Scrapyd utilise.

Notez qu'en générale il est préférable d'être explicite, si lancer Scrapyd sur la même machine et avec le port par défaut alors le wrapper peut être instanciée avec aucuns arguments.

Vous avez peut-être des besoins spéciaux, par exemple l'un des suivants :

- Vous avez peut-être besoin d'une Authentification HTTP Basique pour votre connexion à Scrapyd
- Vous avez peut-être changer les points de sortie par des défauts des différentes actions de l'API
- Vous avez peut-être besoin de changer le connecteur client/gestionnaire
- Vous souhaitez peut-être donner un délai d'attente au requête client pour que le programme n'attende pas indéfiniment dans le cas où le server ne répond pas

Améliorations

Traduction

L'application telle qu'elle est actuellement est en anglais, dans le future il faudrait pouvoir traduire le site web dans plusieurs langages.

API

Dans l'état actuel du projet, les données récupérées sont obtenues seulement en indexant des pages web.

Mais certains site web offre une API, utiliser une API serait une meilleure méthode pour récupérer les données car une API est créée pour permettre à des développeurs d'accéder des informations plus rapidement et avec moins de trafic web et une API change moins souvent qu'un site web.

Notification

En ce moment pour savoir si un média a de nouvelle entrée les utilisateurs doivent aller sur la page web. Il faudrait pouvoir dans le futur envoyer des notifications à l'utilisateur pour qu'il puisse savoir que son média favori a une nouvelle entrée sans aller sur la page web.

Conclusions

J'ai le sentiment d'avoir énormément appris grâce a cette formation

Il m'a aidé apprendre le langage Python et mieux comprendre comment créer une application web. Et je pense que Python va continuer à être mon langage favori pour la programmation.

Je compte continuer à travailler et améliorer cette application dans le future car ce je voie beaucoup d'utilité dans ce projet et je veux que cette application puisse réaliser tout son potentielle.

Grâce a ce projet et la formation j'ai pu mieux apprendre comment être un développeur et ça confirme donc pour moi que c'est dans ce métier que je veux travailler.