

2^e Année de Licence Informatique
Université Paris 8
Institut d'Enseignement à Distance

Réalisation de Programme

Projet : MoodRSS

Documentation Technique

Le 6 avril 2024

Table des matières

1	Introduction	3
2	Architecture du Projet	4
2.1	Structure du Projet : Organisation des Dossiers et Fichiers	4
2.2	Composants Utilisés	5
3	Configuration de l'Environnement de Développement	7
3.1	Configuration de l'Environnement Virtuel	7
4	Paramètres du Projet Django	9
4.1	Paramètres de Configuration (<code>settings.py</code>)	9
4.2	Gestion des URL (<code>urls.py</code>)	10
5	Développement de l'Application	11
5.1	Analyseur de Flux RSS (<code>parser.py</code>)	11
5.2	Analyseur de sentiment (<code>prediction.py</code>)	13
5.3	Vues (<code>views.py</code>)	15
5.4	Tests globaux (<code>tests.py</code>)	16
5.5	Page HTML (<code>dashboard.html</code>)	18
5.6	Feuille de style (<code>style.css</code>)	21
6	Packaging	24
6.1	Packaging Version de Développement	24
6.1.1	Collection des fichiers statiques	24
6.1.2	Gestion des Variables d'Environnement	24
6.1.3	Utilisation de Docker et Docker Compose	24
6.1.4	Makefile	26
6.2	Packaging Version de Production	27
6.2.1	Configuration de Nginx comme Proxy Inverse	27
6.2.2	Gestion des Variables d'Environnement	28
6.2.3	Utilisation de Docker et Docker Compose	29
7	Déploiement	31
7.1	Configuration du Routeur	31
7.1.1	Adresse IP Statique	31
7.1.2	Redirection des ports	31
7.1.3	Pare-feu	32
7.2	Installation du conteneur Docker	32
7.3	Mise en Place de DDNS	34
7.4	Mise en place de proxy inversé	35

A	Annexes	37
A.1	Annexes "3. Configuration de l'Environnement de Développement"	37
A.1.1	moodrss/requirements.txt	37
A.2	Annexes "4. Paramètres du Projet Django"	37
A.2.1	/moodrss/config/settings.py	37
A.2.2	/moodrss/config/urls.py	39
A.3	Annexes "5. Développement de l'Application"	40
A.3.1	/moodrss/rss/parser.py	40
A.3.2	/moodrss/rss/prediction.py	41
A.3.3	/moodrss/rss/views.py	43
A.3.4	/moodrss/rss/tests.py	44
A.3.5	/moodrss/rss/templates/dashboard.html	45
A.3.6	/moodrss/rss/static/style.css	48
A.4	Annexes "6. Packaging"	51
A.4.1	/.env.dev	51
A.4.2	/moodrss/Dockefile	51
A.4.3	/docker-compose.dev.yml	52
A.4.4	/Makefile	52
A.4.5	/nginx/nginx.conf	52
A.4.6	/nginx/Dockerfile	53
A.4.7	/.env	53
A.4.8	/docker-compose.yml	53
B	Ressources	55

1 Introduction

Le projet MoodRSS a pour objectif de développer une application web basée sur le framework Django. Cette application est conçue pour récupérer et analyser les sentiments exprimés dans les articles provenant de divers flux RSS. Ces sentiments peuvent être classés en trois catégories distinctes : positifs, négatifs ou neutres. Les articles sont ensuite présentés sur un tableau de bord interactif, où il est possible de filtrer par sentiment.

La présente documentation couvre l'ensemble des aspects liés au développement et à la mise en œuvre de cette application. Elle détaille l'architecture du projet, présente l'environnement de développement nécessaire à la réalisation de l'application et explique les différents éléments de développement du projet. Elle aborde également le processus de packaging de l'application.

De plus, cette documentation fournit des informations sur les différentes étapes requises pour déployer l'application sur un serveur. Nottament la configuration d'un routeur, la mise en place d'un système de noms de domaine dynamique (DDNS) et d'un proxy inversé.

2 Architecture du Projet

Cette section présente l'architecture globale du projet. Elle décrit en détail l'organisation des dossiers et des fichiers du projet. De plus, les principales technologies du projet sont présentées. Elles sont divisées en quatre catégories : le Frontend, le Backend, la Base de Données, et le Déploiement et l'Hébergement.

2.1 Structure du Projet : Organisation des Dossiers et Fichiers

Le projet est organisé en deux dossiers principaux, le premier contenant l'application Django pour MoodRSS et le second comprenant le serveur Nginx. Parallèlement à cela, on trouve également divers fichiers dédiés au déploiement du projet. Parmi ces fichiers, se trouve la présence de variables d'environnement et d'un Makefile.

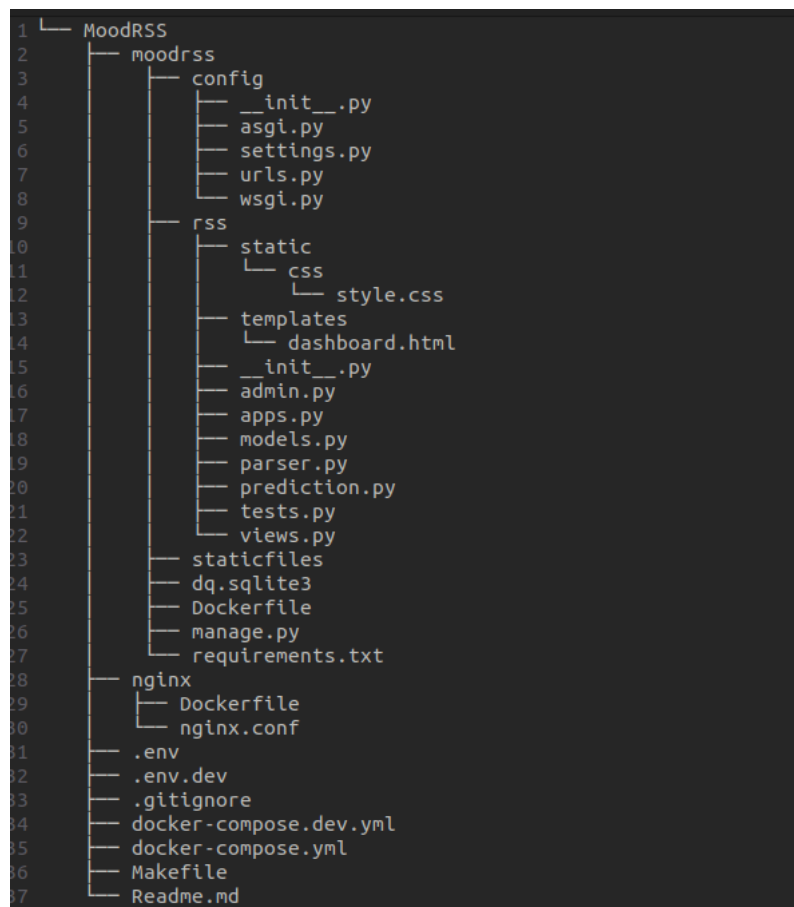


FIG. 1 : Organisation des dossiers et fichiers.

- `moodrss/` : Ce répertoire contient le code principal de l'application Django.
 - `config/` : Contient les fichiers de configuration du projet Django.
 - `asgi.py` : Configuration pour le serveur ASGI (Asynchronous Server Gateway Interface).
 - `settings.py` : Configuration principale du projet Django.
 - `urls.py` : Définit les URLS du projet.
 - `wsgi.py` : Configuration pour le serveur WSGI (Web Server Gateway Interface).

- `static/css/style.css` : Contient la Feuille de style CSS personnalisée pour l'application.
- `templates/dashboard.html` : Contient le fichier HTML de l'application.
- `admin.py` : Configuration de l'interface d'administration Django.
- `apps.py` : Configuration de l'application RSS.
- `models.py` : Définition des modèles de données.
- `parser.py` : Contient le code pour analyser et nettoyer les flux RSS.
- `prediction.py` : Contient le code pour analyser le sentiment des articles.
- `tests.py` : Contient les tests pour l'application.
- `views.py` : Contient les vues de l'application pour gérer les requêtes HTTP.
- `staticfiles` : Dossier pour collecter les fichiers statiques lors du déploiement.
- `db.sqlite3` : Base de données SQLite.
- `Dockerfile` : Fichier de configuration Docker pour containeriser l'application.
- `manage.py` : Utilitaire en ligne de commande pour les tâches de Django.
- `requirements.txt` : Liste des dépendances Python nécessaires pour le projet.
- `nginx/` : Ce répertoire contient la configuration du serveur Nginx utilisé pour servir l'application.
 - `Dockerfile` : Définit l'image Docker pour le serveur Nginx.
 - `nginx.conf` : Configuration du serveur Nginx.
- `.env` : Fichier pour stocker les variables d'environnement de production.
- `.env.dev` : Fichier pour stocker les variables d'environnement de développement.
- `.gitignore` : Fichier spécifiant les fichiers et dossiers à ignorer par Git.
- `docker-compose.dev.yml` : Fichier de configuration Docker Compose pour l'environnement de développement.
- `docker-compose.yml` : Fichier de configuration Docker Compose pour l'environnement de production.
- `Makefile` : Contient des commandes Make pour les tâches courantes telles que la construction des images Docker, le démarrage et l'arrêt des services, et l'exécution des tests.
- `Readme.md` : Documentation du projet, contenant des informations sur l'installation, la configuration et l'utilisation de l'application.

2.2 Composants Utilisés

La section des composants du projet décrit les technologies utilisées dans le développement du projet ainsi que dans son déploiement. Elle est divisée en quatre parties : le Frontend, le Backend, la Base de Données, et le Déploiement et l'Hébergement.

- **Frontend**
 - *HTML5* : Définit la structure de la page web.
 - *CSS3* : Définit le style.

- *JavaScript & Bootstrap* : Ajoute l'interactivité côté client.

Le but est de fournir une interface utilisateur intuitive pour la saisie des URLs des flux RSS, l'affichage des articles et des sentiments, ainsi que les fonctionnalités de filtrage.

- **Backend**

- *Django* : Framework pour le langage Python.
- *VADER* : Bibliothèque utilisée pour analyser les sentiments des textes.

Le backend est construit avec le framework Django, qui gère la logique métier, les interactions avec la base de données, et l'analyse des sentiments. Concernant cette analyse elle s'appuie sur la bibliothèque python VADER (Hutto & Gilbert, 2014). Django fournit également les éléments nécessaires pour fournir le contenu des pages HTML dynamiquement.

- **Base de Données**

- *SQLite* : Utilisé pour le développement et la production.

La base de données stocke les informations des flux RSS et des articles. Etant donné qu'il n'y a pas de gestion de profil utilisateur, SQLite est suffisant pour être utilisé pour le développement et la production car il est léger et facile à configurer.

- **Déploiement et Hébergement**

- *Docker* : Conteneurisation de l'application pour une gestion simplifiée des dépendances et des environnements.
- *Nginx* : Serveur web et proxy inverse.
- *Serveur Synology* : Héberge l'application, avec configuration de DDNS et redirection de ports pour l'accès externe.

L'application est conteneurisée avec Docker et déployée sur un serveur Synology. Nginx est configuré comme proxy inverse pour gérer les requêtes HTTP/HTTPS, assurant une communication sécurisée avec l'application.

3 Configuration de l'Environnement de Développement

Cette section fournit des informations détaillées sur la configuration requise pour le développement du projet MoodRSS.

Elle aborde d'abord la mise en place de l'environnement virtuel, une étape essentielle pour isoler les dépendances du projet et garantir sa portabilité.

Ensuite, la section se termine par la création du projet et de l'application Django, qui forme la base du projet.

3.1 Configuration de l'Environnement Virtuel

Il est recommandé d'utiliser un environnement virtuel pour isoler les dépendances du projet durant son développement. Pour créer cet environnement virtuel il faut dans un premier temps créer le répertoire principal qui héberge le projet dans sa globalité.

Création du dossier contenant le projet :

```
& mkdir MoodRSS
```

Ensuite, la création du dossier contenant l'environnement virtuel.

Création et activation de l'environnement virtuel :

```
& cd MoodRSS
& python -m venv .venv
& source .venv/bin/activate
```

L'environnement, une fois créé et activé, nécessite l'installation de dépendances. Ces dépendances apportent des modules et des bibliothèques complémentaires à Python, qui sont utilisés dans le développement du projet.

Ces dépendances sont contenues dans un fichier appelé `requirements.txt`. Ce fichier énumère les bibliothèques à ajouter, ainsi que leurs versions respectives. L'utilisation de ce fichier facilite la gestion des dépendances du projet, car il permet l'installation simultanée de toutes les dépendances requises avec une seule commande. De plus, en spécifiant les versions des bibliothèques, on évite les problèmes de compatibilité qui pourraient survenir avec des versions différentes.

/requirements.txt (annexe [A.1.1](#)) :

```
1 Django==5.0.6
2 asgiref==3.8.1
3 click==8.1.7
4 feedparser==6.0.11
5 joblib==1.4.2
6 nltk==3.8.1
7 python-dateutil==2.9.0.post0
8 regex==2024.5.15
9 sgmlib3k==1.0.0
10 six==1.16.0
11 sqlparse==0.5.0
12 tqdm==4.66.4
13 typing_extensions==4.12.2
14 gunicorn==22.0.0
```

Installation des dépendances :

```
& pip install --upgrade pip
```



```
& pip install -r requirements.txt
```

Une fois les dépendances installées dans l'environnement virtuel, il faut créer le projet Django. Un projet Django est une collection de paramètres et de configurations, il contient automatiquement une structure de dossiers et de fichiers qui constitue la base du projet. Celui-ci contient un répertoire contenant les fichiers de configuration ainsi qu'un fichier `manage.py` qui est un utilitaire en ligne de commande permettant d'interagir avec le projet.

Par défaut, le dossier principal du projet porte le nom spécifié dans la commande, ici "moodrss". Cependant, pour des raisons de clarté, il est recommandé de renommer le sous-dossier généré automatiquement avec le même nom que le projet. Cette étape de renommage facilite la compréhension de la structure du projet et permet d'avoir un nom de dossier plus significatif.

```
& django-admin startproject moodrss  
& mv moodrss/moodrss/ moodrss/config  
& cd moodrss
```

Après avoir créé le projet Django, il faut à présent créer l'application Django contenue dans ce projet. Une application dans Django est un sous-composant du projet, conçu pour développer les fonctionnalités de celui-ci. Sa création génère également des fichiers nécessaires au fonctionnement du site.

Dans le contexte de ce projet, l'application ici sera nommée "rss". En plus des éléments créés automatiquement, deux dossiers seront également créés : `templates`, pour héberger le fichier HTML qui définit la structure de la page web de l'application, et `static/css` pour la feuille de style qui définit le style et la mise en page de la page web.

```
& python manage.py startapp rss  
& mkdir -p rss/static/css  
& mkdir -p rss/templates
```

4 Paramètres du Projet Django

Cette section se concentrera sur la configuration du projet Django précédemment créé. Plus précisément, elle portera sur la modification de certains paramètres présents dans les fichiers créés automatiquement par Django, `settings.py` et `urls.py`.

4.1 Paramètres de Configuration (`settings.py`)

Le fichier `/moodrss/config/settings.py` de Django contient tous les paramètres de configuration essentiels pour le fonctionnement et la sécurité du projet. Cela inclut les paramètres de sécurité, de base de données, d'applications installées et de gestion des hôtes. Les valeurs des éléments modifiés ci-dessous proviendront des fichiers d'environnement `.env` situés à la racine du projet.

`/moodrss/config/settings.py`, `SECRET_KEY` (annexe [A.2.1](#)) :

```
1 SECRET_KEY = os.environ.get("SECRET_KEY", secrets.token_urlsafe(50))
```

La valeur `SECRET_KEY` dans les paramètres de Django est utilisée entre autre pour signer les cookies, générer des tokens CSRF et hacher les mots de passe. Si cette variable n'est pas définie, une nouvelle clé secrète est générée à l'aide de la fonction `secrets.token_urlsafe(50)`.

`/moodrss/config/settings.py`, `DEBUG` (annexe [A.2.1](#)) :

```
1 DEBUG = bool(os.environ.get("DEBUG", True))
```

La variable `DEBUG` en Django active le mode de débogage avec des messages d'erreur détaillés et des informations sensibles, ce qui ne doit pas être activé en production pour éviter l'exposition de vulnérabilités et d'informations sensibles. Le mode de débogage est ici activé par défaut.

`/moodrss/config/settings.py`, `ALLOWED_HOSTS` (annexe [A.2.1](#)) :

```
1 ALLOWED_HOSTS = os.environ.get("DJANGO_ALLOWED_HOSTS", "localhost").split(" ")
```

La variable `DJANGO_ALLOWED_HOSTS` spécifie les noms de domaine ou adresses IP autorisés à accéder à l'application Django, empêchant ainsi notamment les requêtes HTTP malveillantes, elle doit être correctement configurée en production pour éviter les risques de sécurité. Les hôtes autorisés à se connecter au serveur Django sont définis à partir de la variable d'environnement du même nom. La valeur "localhost" est défini comme hôte autorisé par défaut.

`/moodrss/config/settings.py`, `INSTALLED_APPS` (annexe [A.2.1](#)) :

```
1 INSTALLED_APPS = [  
2     'django.contrib.admin',  
3     'django.contrib.auth',  
4     'django.contrib.contenttypes',  
5     'django.contrib.sessions',  
6     'django.contrib.messages',  
7     'django.contrib.staticfiles',  
8     'rss' # ajout de l'application rss  
9 ]
```

L'application Django nommé "rss" créé précédemment est inclut dans la liste des applications installées en plus des applications standard de Django.

`/moodrss/config/settings.py`, `CSRF_TRUSTED_ORIGINS` (annexe [A.2.1](#)) :

```
1 CSRF_TRUSTED_ORIGINS = os.environ.get("CSRF_TRUSTED_ORIGINS", "http://localhost:8000").split(" ")
```

La variable `CSRF_TRUSTED_ORIGINS` dans Django spécifie une liste de domaines de confiance à partir desquels les requêtes peuvent inclure des tokens CSRF, aidant à prévenir les attaques de type Cross-Site Request Forgery en autorisant seulement les requêtes provenant de ces sources approuvées. Si cette variable n'est pas définie, `"http://localhost:8000/"` est défini comme origine de confiance par défaut.

4.2 Gestion des URL (`urls.py`)

Le fichier `/moodrss/config/urls.py` de Django définit les routes URL de l'application et associe chaque route à une vue correspondante.

`/moodrss/config/urls.py` (annexe [A.2.2](#)) :

```
1 """URL Configuration
2
3 The `urlpatterns` list routes URLs to views. For more information please see:
4     https://docs.djangoproject.com/en/1.10/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import:  from my_app import views
8     2. Add a URL to urlpatterns:  url(r'^$', views.home, name='home')
9 Class-based views
10    1. Add an import:  from other_app.views import Home
11    2. Add a URL to urlpatterns:  url(r'^$', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.conf.urls import url, include
14    2. Add a URL to urlpatterns:  url(r'^blog/', include('blog.urls'))
15 """
16
17 # from django.contrib import admin
18 from django.urls import path
19 from rss.views import dashboard
20
21 urlpatterns = [
22     # désactivation de la page d'administration
23     # path('admin/', admin.site.urls),
24     path('', dashboard, name="dashboard") # ajout de la vue contenant la page
     dashboard
25 ]
```

Ici, la page d'administration est désactivée en commentant la ligne correspondante. Une nouvelle route est ajoutée, elle associe la racine du site (") à la vue "dashboard". Cette configuration affiche la page "dashboard" lorsque l'utilisateur visite la page d'accueil de l'application.

5 Développement de l'Application

Cette section abordera tout le côté purement développement de l'application, allant de l'intégration du module d'analyse de sentiment jusqu'à la page web en HTML.

5.1 Analyseur de Flux RSS (parser.py)

/moodrss/rss/parser.py, clean_html() (annexe [A.3.1](#)) :

```
1 # Nettoie le contenu HTML brut en supprimant les balises HTML et en
  # convertissant les caractères spéciaux.
2 # @param raw_html -> le contenu HTML brut ;
3 # @return -> le texte nettoyé sans balises HTML.
4 def clean_html(html_text):
5     # Suppression des balises HTML
6     clean_text = re.sub('<.*?>', '', html_text)
7     # Conversion des caractères spéciaux
8     return unescape(clean_text)
```

La fonction `clean_html(html_text)` dans le fichier 'parser.py' est utilisée pour nettoyer le contenu HTML brut provenant des flux RSS. Cette fonctionnalité permet que le texte analysé par l'analyseur de sentiment est exempt de balises HTML.

Le premier argument de la fonction est `html_text`, qui représente le contenu HTML brut à nettoyer. Ce contenu est généralement extrait directement des flux RSS et peut contenir diverses balises HTML qui doivent être supprimées avant l'analyse du sentiment.

La fonction utilise une expression régulière pour supprimer toutes les balises HTML du texte. La fonction `re.sub` remplace toutes les occurrences de cette expression régulière dans `html_text` par une chaîne vide, ce qui a pour effet de supprimer toutes les balises HTML.

Après avoir supprimé les balises HTML, la fonction utilise la méthode `unescape` pour convertir les caractères spéciaux HTML en leur équivalent au standard ASCII. Cette étape est nécessaire car les flux RSS peuvent contenir des caractères spéciaux HTML qui ne sont pas directement lisibles.

Enfin, la fonction renvoie le texte nettoyé. Ce texte est maintenant prêt à être analysé par l'analyseur de sentiment.

/moodrss/rss/parser.py, prepare_rss_feed() (annexe [A.3.1](#)) :

```
1 # Récupère un flux RSS depuis une URL, nettoie le contenu HTML des résumés et
  # récupère les dates de publication.
2 # @param url -> l'URL du flux RSS ;
3 # @return -> une liste d'entrées de flux nettoyées, ou "Not found" si aucune
  # entrée n'est trouvée.
4 def prepare_rss_feed(url):
5     # Parcourt le flux RSS
6     feed = feedparser.parse(url)
7     # Vérifie s'il y a des entrées dans le flux
8     if feed.entries:
9         for entry in feed.entries:
10             # Nettoie le résumé
11             entry.summary = clean_html(entry.summary)
12             # Récupère la date de publication
13             entry.published = dateparse(entry.published)
14         return feed.entries
15     else:
16         return "Not found"
```

La fonction `prepare_rss_feed(url)` récupère un flux RSS à partir d'une URL donnée, nettoie le contenu HTML des résumés et récupère les dates de publication.

Cette fonction prend en entrée une URL qui pointe vers le flux RSS à analyser. Elle utilise ensuite la bibliothèque `feedparser` (McKee & Pilgrim, s.d.) pour parcourir et analyser le flux.

Une vérification est effectuée pour s'assurer que le flux RSS contient des entrées. Si des entrées sont présentes, un nettoyage du texte est réalisé sur le résumé de chaque entrée en utilisant la fonction `clean_html()`. La date de publication de chaque entrée est ensuite récupérée en utilisant la fonction `dateparse()` de la bibliothèque `dateutil` (Niemeyer et al., 2003).

Si aucune entrée n'est trouvée dans le flux RSS, la fonction retourne simplement la chaîne de caractères "Not found".

Au final, cette fonction retourne une liste d'entrées de flux nettoyées.

`/moodrss/rss/parser.py`, `TestPrepareRssFeed()` (annexe A.2.1) :

```
1 class TestParser(unittest.TestCase):
2
3     # Test avec une entrée
4     def test_with_entries(self):
5         rss_content = """
6         <rss version="2.0">
7             <channel>
8                 <title>Example RSS Feed</title>
9                 <description>This is an example RSS feed</description>
10                <item>
11                    <title>Exemple</title>
12                    <description><![CDATA[Test résumé avec du contenu <b>HTML</b>
13>.]></description>
14                    <pubDate>Tue, 21 May 2024 16:36:44 +0000</pubDate>
15                </item>
16            </channel>
17        </rss>
18        """
19        result = prepare_rss_feed(rss_content)
20        # Vérifie que la fonction retourne une entrée
21        self.assertEqual(len(result), 1)
22        # Vérifie que le résumé de l'entrée est nettoyé correctement
23        self.assertEqual(result[0].summary, "Test summary with HTML content.")
24
25    # Test sans une entrée
26    def test_without_entries(self):
27        rss_content = """
28        <rss version="2.0">
29            <channel>
30                <title>Exemple RSS</title>
31                <description>RSS</description>
32            </channel>
33        </rss>
34        """
35        result = prepare_rss_feed(rss_content)
36        # Vérifie que la fonction retourne "Not found" quand il n'y a pas d'
37        entrées
38        self.assertEqual(result, "Not found")
39
40    # Test de la fonction clen_html()
41    def test_clean_html(self):
42        # HTML brut
43        raw_html = "<p>Test summary with <b>HTML</b> content.</p>"
```

```

42     # Résultat attendu après nettoyage du HTML
43     expected_output = "Test summary with HTML content."
44     # Vérifier que la fonction clean_html nettoie le HTML correctement
45     self.assertEqual(clean_html(raw_html), expected_output)

```

Pour tester le fichier dans son intégralité, des tests unitaires sont mis en place.

La classe `TestParser` contient trois tests : `test_with_entries()`, `test_without_entries()` et `test_clean_html()`. Chaque fonction teste un scénario spécifique pour la fonction `prepare_rss_feed()`.

La fonction `test_with_entries()` teste le scénario où le flux RSS contient une entrée. Elle crée un exemple de contenu RSS avec une entrée et appelle la fonction `prepare_rss_feed()` avec ce contenu. Ensuite, elle vérifie si la fonction renvoie une entrée et si le résumé de l'entrée est correctement nettoyé.

La seconde fonction `test_without_entries()` teste si le flux RSS ne contient pas d'entrées. Elle utilise un exemple sans entrées avec la fonction `prepare_rss_feed()`. Puis, comme précédemment vérifie si la fonction renvoie "Not found", ce qui est le comportement attendu lorsque le flux RSS ne contient pas d'entrées.

La dernière fonction, `test_clean_html()` teste la fonction `clean_html()`, qui est utilisée par `prepare_rss_feed()` pour nettoyer le HTML du résumé de l'entrée. Une chaîne de caractères contenant du HTML brut est affecté a une variable nommée `raw_html`, appelle ensuite `clean_html()` avec cette variable et vérifie si le contenu HTML est correctement nettoyé.

En somme, ces tests unitaires permettent de valider le bon fonctionnement des fonctions `prepare_rss_feed()` et `clean_html()`, qui sont essentielles pour préparer et nettoyer le flux RSS.

5.2 Analyseur de sentiment (prediction.py)

/moodrss/rss/prediction.py, get_prediction() (annexe [A.3.2](#)) :

```

1  # Analyse le sentiment d'un titre et d'un résumé pour évaluer le sentiment
   global.
2  # @param title -> le titre de l'article ;
3  # @param summary -> le résumé de l'article ;
4  # @return -> chaîne de caractères, 'Positive', 'Negative' ou 'Neutral'.
5  def get_prediction(title, summary):
6      try:
7          nltk.data.find('sentiment/vader_lexicon.zip')
8      except LookupError:
9          nltk.download('vader_lexicon')
10
11     text = title + " " + summary
12
13     # Analyse des scores de sentiment pour le texte
14     sia = SentimentIntensityAnalyzer()
15     sentiment_scores = sia.polarity_scores(text)
16
17     # Extraction du score compound (global) des sentiments
18     compound_score = sentiment_scores['compound']
19
20     # Sélection du sentiment
21     if compound_score >= 0.1:
22         return 'Positive'
23     elif compound_score <= -0.1:
24         return 'Negative'

```

```

25     else:
26         return 'Neutral'

```

La méthode `get_prediction()` dans le fichier `prediction.py` est destinée à évaluer le sentiment global d'un article en analysant le ton exprimé dans le titre et le résumé de l'article. Elle prend en paramètres le titre et le résumé de l'article et retourne une chaîne de caractères indiquant si le sentiment est 'Positive', 'Negative' ou 'Neutral'.

Au début de la fonction, une vérification est effectuée pour voir si le module VADER (Hutto, C.J. & Gilbert, E.E., 2014). est disponible. Si ce n'est pas le cas, le lexique est téléchargé à l'aide de la fonction `nltk.download('vader_lexicon')`. Le module VADER (Valence Aware Dictionary and sEntiment Reasoner) est un outil d'analyse de sentiment développé dans le cadre du projet open-source NLTK.

L'analyseur de sentiment, `SentimentIntensityAnalyzer()`, est ensuite utilisé pour obtenir les scores de sentiment pour le texte. Ces scores sont stockés dans le dictionnaire `sentence_scores`.

Le score `compound` est extrait de `sentence_scores`. Il s'agit d'un score qui mesure le ton global du texte, allant de -1 (extrêmement négatif) à +1 (extrêmement positif).

Enfin, en fonction de la valeur du score `compound`, la fonction retourne 'Positive', 'Negative' ou 'Neutral'. Si le score `compound` est supérieur ou égal à 0.1, le sentiment est considéré comme positif. Si le score `compound` est inférieur ou égal à -0.1, le sentiment est considéré comme négatif. Sinon, le sentiment est considéré comme neutre.

[/moodrss/rss/prediction.py](#), `TestPrediction()` (annexe A.3.2) :

```

1 class TestPrediction(unittest.TestCase):
2
3     def test_positive(self):
4         title = "Paddington In Peru Trailer Sends The Adorable Bear On A New
5         Adventure, And I'm Ready To Follow Him Anywhere"
6         summary = "The Paddington series of movies ranks as the most delightful,
7         and Paddington in Peru looks to extend the winning streak."
8         self.assertEqual(get_prediction(title, summary), 'Positive')
9
10    def test_negative(self):
11        title = "Someone Finally Asked Austin Butler About Those Pirates Of The
12        Caribbean Rumors"
13        summary = "Austin Butler responds to rumors he could star in the
14        next Pirates of the Caribbean movie."
15        self.assertEqual(get_prediction(title, summary), 'Negative')
16
17    def test_neutral(self):
18        title = "Late Night With Seth Meyers Staff Losing Jobs Amidst Budget
19        Cuts: 'Nobody Wants To Pay'"
20        summary = "As part of budget cuts at NBC, Late Night with Seth Meyers is
21        cutting a notable portion of its staff."
22        self.assertEqual(get_prediction(title, summary), 'Neutral')

```

Le fichier `prediction.py` contient également une classe de test unitaire nommée `Test Prediction`. Cette classe assure la validité des fonctionnalités de prédiction de sentiment. Elle comprend trois fonctions test : `test_positive()`, `test_negative()` et `test_neutral()`.

Ces fonctions utilisent le même principe, elle teste le sentiment indiqué est attendu en fonction du titre et du résumé donnés. Le titre et le résumé utilisés dans ces tests sont délibérément optimistes, négatifs ou neutres. Les fonctions vérifient ensuite si la fonction `get_prediction()` retourne bien la chaîne de caractères attendu.

5.3 Vues (views.py)

/moodrss/rss/views.py, dashboard() (annexe [A.3.3](#)) :

```
1 # Traite les requêtes pour afficher un tableau de bord des flux RSS avec analyse
  de sentiment.
2 # @param request -> l'objet de requête HTTP ;
3 # @return -> rend la page HTML du tableau de bord avec les flux RSS analysés et
  filtrés par sentiment.
4 def dashboard(request):
5     if request.method == 'POST':
6         # Récupère les données soumises par l'utilisateur
7         data = request.POST
8         url = data['url'] # URL du flux RSS entré par l'utilisateur
9         feeds = prepare_rss_feed(url) # Prépare le flux RSS
10
11        # Vérifie si un filtre de sentiment a été sélectionné
12        if data['sentiment']:
13            selected = data['sentiment']
14
15        # Vérifie si des flux ont été trouvés
16        if feeds != "Not found":
17            found = 'true'
18
19            unique_feeds = []
20            seen_titles = set()
21
22            # Élimine les doublons
23            for feed in feeds:
24                if feed.title not in seen_titles:
25                    unique_feeds.append(feed)
26                    seen_titles.add(feed.title)
27
28            for feed in unique_feeds:
29                # Analyse le sentiment du titre et du résumé
30                feed['sentiment'] = get_prediction(feed.title, feed.summary)
31                # Ajoute du texte secondaire en fonction du sentiment analysé
32                if feed['sentiment'] == 'Positive':
33                    feed['secondary_text'] = 'This feed brings positive content.
34                ,
35                elif feed['sentiment'] == 'Negative':
36                    feed['secondary_text'] = 'The tone of this feed is more
37                negative.'
38                else:
39                    feed['secondary_text'] = 'This feed provides articles with a
40                neutral tone.'
41
42            # Affecte le nom de l'auteur s'il est disponible
43            if 'authors' in feed:
44                feed['author_name'] = feed.authors[0]['name']
45            else:
46                feed['author_name'] = None
47
48            # Affecte le lien du flux s'il est disponible
49            if 'link' in feed:
50                feed['link'] = feed.link
51            else:
52                feed['link'] = None
53
54            # Affecte la date de publication si elle est disponible
55            if 'published' in feed:
```



```

53         feed['published'] = feed.published
54     else:
55         feed['published'] = None
56
57     # Filtre les flux par sentiment sélectionné
58     selected_sentiment = data.get('sentiment', 'All') # Par défaut
59     if selected_sentiment != 'All':
60         unique_feeds = [feed for feed in unique_feeds if feed['sentiment'] == selected_sentiment]
61
62     # Vérifie s'il y a des flux à afficher et rend la page avec les flux
63     if len(unique_feeds) > 0:
64         return render(request, "dashboard.html", {'feeds': unique_feeds, 'found': found, 'selected': selected, 'url': url, 'empty': 'no', 'feed': unique_feeds[0]})
65     else:
66         # Rend la page indiquant qu'il n'y a aucun flux à afficher
67         return render(request, "dashboard.html", {'feeds': unique_feeds, 'found': found, 'selected': selected, 'url': url, 'empty': 'yes'})
68
69     else:
70         # Indique que le flux RSS n'a pas été trouvé
71         found = 'false'
72         return render(request, "dashboard.html", {'feeds': None, 'found': found, 'selected': selected})
73
74     else:
75         # Rend la page du tableau de bord vide pour une requête GET
76         return render(request, "dashboard.html")

```

La fonction `dashboard()` est définie pour traiter les requêtes HTTP et retourner les éléments attendu dans le code HTML pour afficher la page. Elle accepte un argument `request` qui est l'objet de requête HTTP.

Si la méthode de la requête est `POST`, la fonction commence par récupérer ces données. L'URL du flux RSS, entrée par l'utilisateur, est obtenue à partir de ces données.

Ensuite, la fonction prépare le flux RSS avec l'URL récupérée. Elle vérifie également si un filtre de sentiment a été sélectionné par l'utilisateur. Dans le cas où des articles sont trouvés, la fonction élimine les doublons en créant une nouvelle liste d'articles uniques et en la remplissant avec des articles dont les titres n'ont pas été vus auparavant.

Pour chaque article unique, la fonction analyse le sentiment du titre et du résumé. Elle attribue ensuite différentes propriétés lié à article (comme le nom de l'auteur, le lien de l'article et la date de publication) si elles sont disponibles.

Les articles uniques sont ensuite filtrés par le sentiment sélectionné (si un sentiment a été sélectionné). Si, après ce filtrage, des articles sont à afficher, la fonction rend la page avec ces articles. Sinon, elle rend une page indiquant qu'il n'y a aucun article à afficher.

Si aucun article n'a été trouvé initialement, la fonction indique que le flux RSS n'a pas été trouvé et rend une page appropriée.

5.4 Tests globaux (tests.py)

`/moodrss/rss/tests.py`, `MoodRSSTests()` (annexe [A.3.4](#)) :

```

1 class MoodRSSTests(TestCase):
2

```

```

3     # Prépare l'environnement de test
4     def setUp(self):
5         self.client = Client()
6         self.url = reverse('dashboard')
7
8     # Test de la requête GET pour vérifier que la page se charge correctement
9     # sans flux RSS
10    def test_get(self):
11        response = self.client.get(self.url)
12        self.assertEqual(response.status_code, 200)
13        self.assertTemplateUsed(response, 'dashboard.html')
14        self.assertContains(response, 'MoodRSS')
15
16    # Test de la requête POST avec une URL de flux RSS valide
17    def test_post_valid_rss(self):
18        response = self.client.post(self.url, {'url': 'https://www.cinemablend.
19        com/feeds.xml', 'sentiment': 'All'})
20        self.assertEqual(response.status_code, 200)
21        # Vérifie qu'il ne contient pas le message "Feed not found"
22        self.assertNotContains(response, 'Feed not found')
23        # Vérifie que des flux sont retournés
24        self.assertGreater(len(response.context['feeds']), 0)
25
26    # Test de la requête POST avec une URL de flux RSS invalide
27    def test_post_invalid_rss(self):
28        response = self.client.post(self.url, {'url': 'http://invalid-rss-feed.
29        com/rss', 'sentiment': 'All'})
30        self.assertEqual(response.status_code, 200)
31        self.assertContains(response, 'Feed not found')
32
33    # Test de la requête POST avec un filtre de sentiment
34    def test_post_sentiment_filter(self):
35        # Effectue une requête avec un filtre de sentiment
36        response = self.client.post(self.url, {'url': 'https://www.cinemablend.
37        com/feeds.xml', 'sentiment': 'Positive'})
38        self.assertEqual(response.status_code, 200)
39        self.assertNotContains(response, 'Feed not found')
40        # Vérifie que tous les flux retournés ont un sentiment positif
41        for feed in response.context['feeds']:
42            self.assertEqual(feed['sentiment'], 'Positive')

```

La classe `MoodRSSTests` est utilisée pour effectuer différents tests unitaires sur les fonctionnalités de l'application `MoodRSS` dans son ensemble. Ces tests sont essentiels pour s'assurer que l'application fonctionne correctement avant de tester l'interface du tableau de bord dans un navigateur web.

La fonction `setUp()` qui est appelée avant les tests, est utilisée pour configurer l'environnement de test. Dans ce cas, un client de test Django est créé pour simuler un navigateur Web et l'URL du tableau de bord est générée.

Le premier test `test_get()` effectue une requête GET sur l'URL du tableau de bord pour vérifier que la page se charge correctement même sans flux RSS. L'état de la réponse, le modèle utilisé pour la réponse et le contenu de la réponse sont vérifiés.

Le second test `test_post_valid_rss()` effectue une requête POST sur l'URL du tableau de bord en envoyant un flux RSS valide. Il vérifie que la page se charge correctement, qu'aucun message d'erreur n'est renvoyé et qu'au moins un flux est retourné.

Le troisième test `test_post_invalid_rss()` effectue également une requête POST sur

l'URL du tableau de bord, mais envoie cette fois-ci un flux RSS invalide. Il vérifie que la page se charge correctement, mais s'attend à un message d'erreur indiquant que le flux n'a pas été trouvé.

Enfin, le dernier test `test_post_sentiment_filter()` effectue une requête POST en envoyant un flux RSS valide et un filtre de sentiment. Il vérifie que la page se charge correctement, qu'aucun message d'erreur n'est renvoyé et que tous les flux renvoyés ont le sentiment correspondant au filtre.

5.5 Page HTML (dashboard.html)

/moodrss/rss/templates/dashboard.html (annexe A.3.5) :

```
1 {% load static %}
2 <!DOCTYPE html>
3 <html lang="en">
4     <head>
5         <meta charset="UTF-8">
6         <meta name="viewport" content="width=device-width, initial-scale=1,
7             shrink-to-fit=no">
8
9         <!-- Fichier CSS de Bootstrap -->
10        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/
11            bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/
12            azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3yD65VohhpuuC0mLASjC" crossorigin="
13            anonymous">
14        <!-- Fichier CSS personnalisé -->
15        <link rel="stylesheet" href="{% static 'css/style.css' %}">
16    </head>
17
18    <body>
19        <div id="content" >
20            <!-- En-tête de la page -->
21            <div class="d-flex align-items-center justify-content-center my-3"
22                style="border-bottom: 1px solid rgb(221, 218, 218);">
23                <h1 style="font-family: 'Trebuchet MS', 'Lucida Sans Unicode', '
24                    Lucida Grande', 'Lucida Sans', Arial, sans-serif; color: #3068ff;"><b>MoodRSS
25                </b></h1>
26            </div>
27            <!-- Formulaire de recherche pour les flux RSS -->
28            <div id="search-container" class="d-flex flex-column align-items-
29                center justify-content-center">
30                <p class="search-text"><b>Search for Feeds</b></p>
31                <form class="input-group w-50" action="/" method="post">
32                    {% csrf_token %}
33                    <!-- Affiche l'URL entré s'il existe, sinon un champ vide --
34                >
35
36                    {% if url %}
37                        <input type="text" class="form-control my-input"
38                            placeholder="Search for RSS Feeds..." name="url" value="{{url}}">
39                    {% else %}
40                        <input type="text" class="form-control my-input"
41                            placeholder="Search for RSS Feeds..." name="url">
42                    {% endif %}
43
44                    <!-- Valeur du sentiment par défaut sur 'All' -->
45                    <input type="hidden" name="sentiment" value="All">
46                    <div class="input-group-append" style="margin-left: -2px;">
47                        <!-- Bouton de soumission du formulaire -->
```

```

35         <button class="btn my-btn" type="submit" style="border-
top-left-radius: 0px; border-bottom-left-radius: 0px;">Search</button>
36     </div>
37 </div>
38
39 <div class="row">
40     <!-- Affiche un message si aucun flux n'est trouvé -->
41     {% if found == 'false' %}
42         <div class="col-md-12 px-5 py-5" style="min-height: 180px;">
43             <div class="h-100 d-flex justify-content-center align-
items-center error-div" style="border-radius: 10px;">
44                 <h3 style="font-family: 'Trebuchet MS', 'Lucida Sans
Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif; color: #EF2917;
">Feed not found!</h3>
45             </div>
46         </div>
47     {% else %}
48         <!-- Affiche le menu déroulant si un URL est présent -->
49         {% if url %}
50             <div class="col-md-12 d-flex justify-content-center
align-items-center" style="min-height: 50px;">
51                 <div class="dropdown">
52                     <button class="btn dropdown-toggle filter-button
" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" aria-
expanded="false">
53                         Filter by Mood
54                     </button>
55                     <ul class="dropdown-menu my-dropdown" aria-
labelledby="dropdownMenuButton1">
56                         <!-- Boutons pour filtrer les flux par
sentiment -->
57                         <li><button class="dropdown-item" type="
submit" name="sentiment" value="All">All</button></li>
58                         <li><button class="dropdown-item" type="
submit" name="sentiment" value="Positive">Positive</button></li>
59                         <li><button class="dropdown-item" type="
submit" name="sentiment" value="Negative">Negative</button></li>
60                         <li><button class="dropdown-item" type="
submit" name="sentiment" value="Neutral">Neutral</button></li>
61                     </ul>
62                 </div>
63             </form>
64         </div>
65     {% endif %}
66
67     <!-- Affiche le sentiment des flux filtrés si un filtre
autre que 'All' est sélectionné -->
68     {% if selected != 'All' and empty == 'no' %}
69         <div class="feeds-div-sentiment" id="feeds-div-sentiment
">
70             <div class="d-flex flex-column align-items-
center justify-content-center h-100 search-content py-3 px-5" style="text-
align: center;">
71                 <!-- Affiche le sentiment avec des
couleurs spécifiques -->
72                 {% if feed.sentiment == 'Positive' %}
73                     <p class="sentiment" style="color
:#008148"><b>{{ feed.sentiment }}</b></p>
74                 {% elif feed.sentiment == 'Negative' %}

```

```

75         <p class="sentiment" style="color:#
EF2917"><b>{{ feed.sentiment }}</b></p>
76         {% else %}
77         <p class="sentiment" style="color
:#46494C"><b>{{ feed.sentiment }}</b></p>
78         {% endif %}
79         <p class="sentiment-text">{{feed.
secondary_text}}</p>
80     </div>
81 </div>
82 {% endif %}
83 <!-- Boucle sur les flux pour les afficher -->
84 {% for feed in feeds %}
85     <!-- Affiche les détails du flux si aucun filtre ou
filtre 'All' est sélectionné -->
86     {% if selected == 'All' %}
87     <div class="col-md-12 feeds-div-content">
88         <div class="search-content h-100 py-3 px-4">
89             <div class="d-flex flex-row">
90                 <p class="feed-title" style="flex: 1"><b>{{
feed.title}}</b></p>
91                 <!-- Lien pour ouvrir le flux dans un nouvel
onglet -->
92                 <a class="btn btn-primary my-open-btn ml-
auto" href="{{ feed.link }}" target="_blank"><span>Open</span></a>
93             </div>
94             <p style="width: 70%;">{{feed.summary}}</p>
95             <!-- Affiche l'auteur du flux si disponible -->
96             {% if feed.author_name %}
97             <p>{{feed.author_name}}</p>
98             {% endif %}
99             <!-- Affiche la date de publication formatée -->
100             <p>{{feed.published|date:"d F Y, H:i"}}</p>
101         </div>
102     </div>
103     {% else %}
104     <!-- Affiche les détails du flux filtré par sentiment --
>
105     <div class="feeds-div-content">
106         <div class="search-content h-100 py-3 px-4">
107             <div class="d-flex flex-row">
108                 <p class="feed-title" style="flex: 1;"><b>{{
feed.title}}</b></p>
109                 <a class="btn btn-primary my-open-btn ml-
auto" href="{{ feed.link }}" target="_blank"><span>Open</span></a>
110             </div>
111             <p style="width: 70%;">{{feed.summary}}</p>
112             {% if feed.author_name %}
113             <p>{{feed.author_name}}</p>
114             {% endif %}
115             <p>{{feed.published|date:"d F Y, H:i"}}</p>
116         </div>
117     </div>
118     {% endif %}
119
120     <!-- Affiche une ligne de séparation entre les flux si
un sentiment est défini -->
121     <div class="col-md-12 px-5" style="padding-top: 2rem;
padding-bottom: 0.7rem;">

```

```

122         <div class="" style="border-top: 1px solid rgb(221,
123         218, 218);">
124             </div>
125         </div>
126         {% endfor %}
127     {% endif %}
128 </div>
129
130 <!-- Scripts pour le menu déroulant -->
131 <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/
132 popper.min.js" integrity="sha384-IQsoLX15PILFhosVNubq5LC7Qb9DXgDA9i+
133 tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p" crossorigin="anonymous"></script>
134 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/
135 bootstrap.min.js" integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTR1cfu0JTxR+EQDz/
136 bgldoEyl4H0zUF0QKbrJ0EcQF" crossorigin="anonymous"></script>
137
138 </body>
139 </html>

```

Le fichier `dashboard.html` est la page principale de l'application MoodRSS.

La balise "body" contient le contenu principal de la page web. Il y a un formulaire de recherche pour les flux RSS, un menu déroulant pour filtrer les flux par humeur, et une section pour afficher les flux. Si aucun flux n'est trouvé, un message est affiché à l'utilisateur. Sinon, chaque flux est affiché avec son titre, son résumé, le nom de l'auteur (si disponible), et la date de publication. Si un filtre autre que 'All' est sélectionné, le sentiment des flux filtrés est également affiché.

Enfin, des scripts pour le menu déroulant sont inclus à la fin de la balise "body". Ces scripts sont liés à des fichiers JavaScript hébergés sur un CDN (Content Delivery Network) et sont nécessaires pour la fonctionnalité du menu déroulant de Bootstrap.

5.6 Feuille de style (style.css)

`/moodrss/rss/static/css/style.css` (annexe [A.3.6](#)) :

```

1 body {
2     /* Cache les débordements horizontaux */
3     overflow-x: hidden;
4 }
5
6 /* Conteneur de la barre de recherche*/
7 #search-container {
8     margin-bottom: 20px;
9     min-height: 180px;
10    border-bottom: 1px solid rgb(221, 218, 218);
11 }
12
13 /* Texte du nom de la recherche */
14 .search-text {
15     font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
16     Sans', Arial, sans-serif;
17     font-size: x-large;
18 }
19
20 /*Bouton Search*/
21 .my-btn {
22     background-color: #3068ff;

```

```

22     color: #F8F8F8;
23 }
24
25 /*Barre de recherche*/
26 .my-input {
27     border: 1px solid #3068ff;
28     border-top-left-radius: 5px !important;
29     border-bottom-left-radius: 5px !important;
30 }
31
32 /*Contenu de la recherche*/
33 .search-content {
34     border: 1px solid rgb(221, 218, 218);
35     border-radius: 10px;
36 }
37
38 /*Survol du contenu de la recherche*/
39 .search-content:hover {
40     border: 1px solid #3068ff;
41 }
42
43 /*Titre du sentiment*/
44 .sentiment {
45     font-size: xx-large;
46     font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
47     Sans', Arial, sans-serif;
48 }
49
50 /*Description du sentiment*/
51 .sentiment-text {
52     font-size: x-large;
53     font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
54     Sans', Arial, sans-serif;
55 }
56
57 /*Titre d'article*/
58 .feed-title {
59     font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
60     Sans', Arial, sans-serif;
61     font-size: 20px;
62 }
63
64 /*Bouton d'ouverture de lien*/
65 .my-open-btn {
66     background-color: #3068ff;
67     width: 5rem;
68     height: 2.2rem;
69 }
70
71 /*Ecran de moins de 991px*/
72 @media (max-width: 991px) {
73     /*Barre de recherche*/
74     .input-group {
75         width: 90% !important;
76     }
77
78     /*Marges pour les différents liens*/
79     .feeds-div-content {
80         padding-left: 30px;
81         padding-right: 30px;

```

```

79     padding-top: 20px;
80 }
81
82 /*Marges pour lle sentiment selectionne*/
83 .feeds-div-sentiment {
84     padding-left: 30px;
85     padding-right: 30px;
86     padding-top: 20px;
87 }
88 }
89
90 /*Ecran de plus de 991px*/
91 @media (min-width: 991px) {
92     /*Marges pour les différents liens*/
93     .feeds-div-content {
94         padding-left: 40px;
95         padding-right: 40px;
96         padding-top: 20px;
97     }
98
99     /*Marges pour lle sentiment selectionne*/
100    .feeds-div-sentiment {
101        padding-right: 40px;
102        padding-left: 40px;
103        padding-top: 20px;
104    }
105 }
106
107 /*Bordure de la division si aucun flux n'est trouvé*/
108 .error-div {
109     border: 1px solid rgb(221, 218, 218);
110 }
111
112 /*Change la couleur de la bordure au survol*/
113 .error-div:hover {
114     border: 1px solid #3068ff;
115 }
116
117 /* Supprime l'ombre et le contour des formulaire et des boutons */
118 .form-control, .btn {
119     box-shadow: none !important;
120     outline: none !important;
121 }
122
123 /* Définit la bordure et la couleur du texte pour le bouton de filtre */
124 .filter-button {
125     border: 1px solid #3068ff;
126     color: #3068ff;
127 }

```

Le fichier CSS est structuré en plusieurs sections, chacune dédiée à un élément spécifique de la page web.

6 Packaging

Ce chapitre traite de la préparation de l'application MoodRSS pour son déploiement complet. Il comprend la définition de deux environnements distincts : un pour le développement, où les fonctionnalités peuvent être testées sans risque, et un autre pour la production, où l'application sera utilisée par les utilisateurs finaux. Ce chapitre couvre non seulement la collecte des fichiers statiques, mais aussi la gestion des variables d'environnement. De plus, il explique comment créer les fichiers de configuration pour Docker et Docker Compose pour la conteneurisation. Enfin, il aborde la configuration de Nginx en tant que proxy inverse pour l'environnement de production.

6.1 Packaging Version de Développement

6.1.1 Collection des fichiers statiques

La commande `python manage.py collectstatic` est utilisée pour rassembler tous les fichiers statiques de l'application dans un seul endroit, afin qu'ils puissent être servis rapidement. Les fichiers statiques sont des fichiers qui ne sont pas générés dynamiquement par l'application Django. Ils peuvent inclure des fichiers CSS, JavaScript, des images et tout autre type de fichier qui est référencé dans les templates HTML de l'application. Cette commande est particulièrement utile lors du déploiement de l'application, car elle garantit que tous les fichiers statiques nécessaires sont inclus dans le dossier `staticfiles`, prêts à être servis par le serveur web.

```
$ python manage.py collectstatic
```

6.1.2 Gestion des Variables d'Environnement

/ .env.dev (annexe [A.4.1](#)) :

```
1 DEBUG=True
2 DJANGO_ALLOWED_HOSTS=localhost 0.0.0.0
```

Le fichier `/ .env.dev` est un fichier d'environnement qui est utilisé pour définir les variables d'environnement nécessaires pour le projet dans un contexte de développement.

6.1.3 Utilisation de Docker et Docker Compose

/moodrss/Dockerfile (annexe [A.4.2](#)) :

```
1 #####
2 # BUILDER #
3 #####
4
5 # Image de base du conteneur
6 FROM python:3.11.4-slim-buster as builder
7
8 # Définit le répertoire de travail
9 WORKDIR /usr/src/moodrss
10
11 # Variables d'environnement pour optimiser les performances de Python
12 ENV PYTHONUNBUFFERED 1 \
13     PYTHONDONTWRITEBYTECODE 1
14
15 # Copie le code source de l'application
16 COPY . /usr/src/moodrss/
17
18 # Controle des conventions de code
```

```

19 RUN pip install --upgrade pip
20 RUN pip install flake8==6.0.0
21 RUN flake8 --ignore=E501,F401 .
22
23 # Installe les dépendances
24 COPY ./requirements.txt .
25 RUN pip wheel --no-cache-dir --no-deps --wheel-dir /usr/src/moodrss/wheels -r
    requirements.txt
26
27 #####
28 # FINAL #
29 #####
30
31 FROM python:3.11.4-slim-buster
32
33 # Crée l'utilisateur de l'application
34 RUN addgroup --system moodrss && adduser --system --ingroup moodrss moodrss
35
36 # Crée les répertoires appropriés
37 ENV APP_HOME=/home/moodrss/web
38 RUN mkdir -p $APP_HOME/staticfiles
39 WORKDIR $APP_HOME
40
41 # Copie des dépendances depuis l'étape de construction
42 COPY --from=builder /usr/src/moodrss/wheels /wheels
43 COPY --from=builder /usr/src/moodrss/requirements.txt .
44
45 # Installe les dépendances
46 RUN pip install --no-cache-dir --upgrade pip
47 RUN pip install --no-cache /wheels/*
48
49 # Copie le code source de l'application
50 COPY . $APP_HOME
51
52 # Change l'utilisateur
53 USER moodrss

```

Ce script Dockerfile est divisé en deux phases : la phase de construction **BUILDER** et la phase finale **FINAL**.

Dans la phase **BUILDER**, l'image de base Python 3.11.4 est utilisée pour construire un conteneur. L'environnement de travail est défini dans le répertoire `/usr/src/moodrss`. Des variables d'environnement sont ensuite définies pour optimiser les performances de Python. Le code source de l'application est copié dans le répertoire de travail du conteneur.

Ensuite, le gestionnaire de paquets pip est mis à jour et le module flake8 est installé pour contrôler les conventions de code. Les dépendances sont copiées et installées grâce au module **wheel**, qui crée des fichiers de distribution pour les packages.

La phase **FINAL** commence avec la même image de base Python. Un nouvel utilisateur et groupe nommé moodrss sont créés pour exécuter l'application. Des répertoires appropriés sont créés pour l'application et définis comme le répertoire de travail.

Les dépendances construites au cours de la phase **BUILDER** sont ensuite copiées dans le conteneur final et installées. Après cela, le code source de l'application est également copié dans le conteneur. Enfin, l'utilisateur du conteneur est changé pour l'utilisateur précédemment créé, garantissant que l'application s'exécute en tant qu'utilisateur non privilégié pour des raisons de sécurité.

/docker-compose.dev.yml (annexe [A.4.3](#)) :

```
1 version: '3.8'
2
3 services:
4   web:
5     build: ./moodrss
6     container_name: moodrss_web
7     command: python manage.py runserver 0.0.0.0:8000
8     volumes:
9       - static_volume:/home/moodrss/web/staticfiles
10    ports:
11      - 8000:8000
12    env_file:
13      - ./env.dev
14    security_opt:
15      - no-new-privileges:true
16    restart: always
17
18 volumes:
19   static_volume:
```

Le fichier `docker-compose.dev.yml` sert à définir les services, les configurations et les dépendances nécessaires pour déployer l'application dans un environnement de développement à l'aide de Docker Compose.

Dans la section des services, un service nommé "web" est défini. Ce service représente l'application web MoodRSS. La clé "build" indique le chemin vers le Dockerfile de l'application. Le nom du conteneur est défini par "moodrss_web".

La commande spécifiée est celle qui sera exécutée lorsque le conteneur sera lancé. Ici, il s'agit d'exécuter le serveur de l'application Django sur le port 8000.

Le volume `static_volume` est monté au chemin `/home/moodrss/web/staticfiles` dans le conteneur. Cela permet de persister les fichiers statiques générés par l'application.

Les ports spécifient le mappage des ports entre le conteneur et la machine hôte. Ici, le port 8000 du conteneur est mappé sur le port 8000 de la machine hôte.

Le fichier `.env.dev` est utilisé pour définir les variables d'environnement nécessaires à l'application dans le conteneur.

Ensuite, il y a l'option de sécurité `no-new-privileges`, qui est définie sur `true`. Cela empêche l'escalade des privilèges dans le conteneur, améliorant ainsi la sécurité.

Enfin, la politique de redémarrage du conteneur est définie sur `always`, ce qui signifie que le conteneur sera redémarré automatiquement si une erreur se produit.

6.1.4 Makefile

/Makefile (annexe [A.4.4](#)) :

```
1 # Variables
2 DOCKER_COMPOSE = docker-compose
3 COMPOSE_FILE ?= docker-compose.yml
4
5 # Défini le fichier docker-compose en fonction de l'environnement
6 ifeq ($(ENV),dev)
7   COMPOSE_FILE = docker-compose.dev.yml
8 endif
9
```

```

10
11 build: # Construit les images Docker
12     $(DOCKER_COMPOSE) -f $(COMPOSE_FILE) up -d --build
13
14 down: # Arrête les services
15     $(DOCKER_COMPOSE) -f $(COMPOSE_FILE) down -v
16
17 logs: # Affiche les logs des services
18     $(DOCKER_COMPOSE) -f $(COMPOSE_FILE) logs -f
19
20 clean: # Supprime les conteneurs, images, volumes et réseaux
21     $(DOCKER_COMPOSE) -f $(COMPOSE_FILE) down --rm all --volumes --remove-orphans

```

Le fichier `Makefile` contient des commandes pour gérer les conteneurs Docker, qui sont utilisés pour déployer l'application.

Le fichier commence par définir des variables. `DOCKER_COMPOSE` est défini comme `docker-compose`, qui est la commande utilisée pour gérer les services Docker Compose. `COMPOSE_FILE` est une variable qui détermine le fichier de configuration Docker Compose à utiliser. Par défaut, il est défini sur `docker-compose.yml`, qui est le fichier de configuration pour l'environnement de production.

Ensuite, une condition est utilisée pour vérifier si l'environnement est défini sur "dev" (développement). Si c'est le cas, `COMPOSE_FILE` est réassigné à `docker-compose.dev.yml`, qui est le fichier de configuration pour l'environnement de développement.

Le fichier `Makefile` contient ensuite une série de commandes :

- **build** : Cette commande construit les images Docker et démarre les services en utilisant les configurations définies dans le fichier Docker Compose spécifié.
- **down** : Cette commande arrête les services Docker et supprime les conteneurs, les réseaux, les volumes et les images définis dans le fichier Docker Compose.
- **logs** : Cette commande affiche les journaux des services Docker.
- **clean** : Cette commande supprime tous les conteneurs, images, volumes et réseaux liés au fichier Docker Compose spécifié.

6.2 Packaging Version de Production

6.2.1 Configuration de Nginx comme Proxy Inverse

Dans le cadre de la version de production du projet MoodRSS, Nginx est utilisé comme un proxy inverse. Il est configuré à l'aide du fichier `nginx.conf`.

/nginx/nginx.conf (annexe [A.4.5](#)) :

```

1 upstream moodrss {
2     server web:8000;
3 }
4
5 server {
6
7     listen 80;
8
9     location / {
10         proxy_pass http://moodrss;
11         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

```

```

12     proxy_set_header Host $host;
13     proxy_redirect off;
14 }
15
16     location /static/ {
17         alias /home/moodrss/web/staticfiles/;
18     }
19
20 }

```

Le fichier `nginx.conf` définit une configuration pour le serveur Nginx. Il est structuré en deux sections principales. La première section, `upstream`, est utilisée pour définir un groupe de serveurs. Dans ce cas, le groupe `moodrss` est créé, qui inclut le serveur web sur le port 8000.

La deuxième section, `server`, est utilisée pour définir un serveur virtuel. Le serveur écoute sur le port 80 correspondant au port HTTP. Dans cette configuration, deux emplacements sont définis : `/` et `/static/`.

L'emplacement `/` est configuré pour passer les requêtes au groupe de serveurs `moodrss` défini précédemment. Trois directives de proxy sont utilisées ici pour gérer les en-têtes HTTP. La directive `proxy_pass` est utilisée pour rediriger les requêtes, `proxy_set_header X-Forwarded-For` pour passer l'adresse IP du client au serveur et `proxy_set_header Host` pour passer l'hôte de la requête au serveur. Enfin, `proxy_redirect off` est utilisé pour désactiver le réacheminement des réponses HTTP.

L'emplacement `/static/` est configuré pour servir les fichiers statiques de l'application. La directive `'alias'` est utilisée pour spécifier le chemin vers le répertoire contenant les fichiers statiques.

/nginx/Dockerfile (annexe [A.4.6](#)) :

```

1 # Image officielle
2 FROM nginx:1.25
3
4 # Supprime la configuration par défaut de NGINX
5 RUN rm /etc/nginx/conf.d/default.conf
6
7 # Copie le fichier de configuration personnalisé dans le répertoire de
  configuration
8 COPY nginx.conf /etc/nginx/conf.d

```

Ce fichier Dockerfile sert à déployé dans un conteneur le serveur Nginx dédié à l'application MoodRSS. La première ligne indique que l'image officielle de Nginx version 1.25 est utilisée comme base pour la construction.

Dans la ligne suivante, la commande `RUN rm /etc/nginx/conf.d/default.conf` est exécutée. C'est une instruction qui supprime la configuration par défaut de Nginx. Cette opération est nécessaire pour permettre l'utilisation d'une configuration personnalisée de Nginx, qui sera plus adaptée aux besoins spécifiques de l'application.

Enfin, la dernière ligne du fichier Docker copie le fichier de configuration personnalisé de Nginx dans le répertoire de configuration du serveur Nginx dans le conteneur Docker. Cette opération est réalisée grâce à la commande `COPY nginx.conf /etc/nginx/conf.d`.

6.2.2 Gestion des Variables d'Environnement

`/.env` (annexe [A.4.7](#)) :

```

1 SECRET_KEY=secrets.token_urlsafe(50)
2 DEBUG=False
3 DJANGO_ALLOWED_HOSTS=moodrss.seizh.synology.me localhost
4 CSRF_TRUSTED_ORIGINS=https://moodrss.seizh.synology.me http://localhost:1009

```

Cette fois-ci, pour la version de production un autre fichier contenant les variables d'environnement est utilisé. Les variables `SECRET_KEY`, `DEBUG`, `DJANGO_ALLOWED_HOSTS`, `CSRF_TRUSTED_ORIGINS`, contiennent alors une valeur différentes de la version de développement.

6.2.3 Utilisation de Docker et Docker Compose

/docker-compose.yml (annexe [A.4.8](#)) :

```

1 version: '3.8'
2
3 services:
4   web:
5     build: ./moodrss
6     container_name: moodrss_web
7     command: gunicorn config.wsgi:application --bind 0.0.0.0:8000
8     volumes:
9       - static_volume:/home/moodrss/web/staticfiles
10    env_file:
11      - ./env
12    ports:
13      - 8000
14    security_opt:
15      - no-new-privileges:true
16    restart: always
17
18    nginx:
19      build: ./nginx
20      container_name: moodrss_nginx
21      volumes:
22        - static_volume:/home/moodrss/web/staticfiles
23      ports:
24        - 1009:80
25      depends_on:
26        - web
27      security_opt:
28        - no-new-privileges:true
29      restart: always
30
31 volumes:
32   static_volume:

```

En déployant la version de production, un nouveau fichier `docker-compose` est également créé afin d'inclure plus de paramètres ainsi que la serveur Nginx.

Au service `web`, est remplacé la commande d'exécution `python manage.py runserver 0.0.0.0:8000`, par `gunicorn config.wsgi application bind 0.0.0.0:8000`. Celle-ci démarre le fichier WSGI (Web Server Gateway Interface) de l'application Django.

Ce type de fichier dans une application Django sert de point d'entrée pour les serveurs compatibles WSGI afin de déployer l'application. Lorsqu'il est utilisé avec Gunicorn, ce fichier permet de charger et de servir l'application Django en production, en facilitant la gestion des requêtes HTTP entre le serveur web et l'application Django.

Les variables d'environnement sont lues à partir du fichier `.env` cette fois-ci.

Le service `nginx` est similaire. Il est construit à partir du répertoire du même nom, qui contient le Dockerfile vu précédemment. Le nom du conteneur est défini comme `moodrss_nginx`. Le même volume `static_volume` est également monté sur le chemin `/home/moodrss/web/staticfiles` du conteneur. Le service est configuré pour dépendre du service `web`, ce qui signifie que ce dernier sera démarré avant le service `nginx`. Le port 80 du conteneur Nginx est mappé sur le port 1009 de l'hôte pour les connexions entrantes.

Enfin, un volume Docker nommé `static_volume` est défini à la fin du fichier. Ce volume est utilisé pour partager les fichiers statiques entre les services `web` et `nginx`.

7 Déploiement

Cet ultime section aborde le déploiement final de l'application. Celui se fera alors dans un serveur. Les éléments abordés iront de la configuration du routeur du réseau interne contenant le serveur jusqu'à la mise en place d'un proxy inversé présent sur le serveur.

7.1 Configuration du Routeur

La première étape de déploiement consiste à configurer le routeur présent dans le réseau interne du serveur. Cette étape est cruciale pour s'assurer que le serveur et l'application soit accessible à partir de l'extérieur.

7.1.1 Adresse IP Statique

Afin d'assurer une connectivité constante et fiable, il est nécessaire d'attribuer une adresse IP statique au serveur. Une adresse IP statique est une adresse IP qui est assignée manuellement à un dispositif dans un réseau et qui reste constante, contrairement à une adresse IP dynamique qui peut changer au fil du temps avec le renouvellement du bail DHCP. Cette adresse IP fixe facilite l'accès au serveur, notamment pour les connexions entrantes, car elle ne change pas. Cela est particulièrement important pour les serveurs qui hébergent des sites web ou d'autres services qui doivent être constamment accessibles.

Baux DHCP statiques

Attribuez vous-même une adresse IP à votre équipement.

Android 192.168.1.12 D4:3A:2E:44:10:94 [Ajouter](#)

Équipement	Adresse IP statique	Adresse MAC	
Tesseract	192.168.1.24	00:11:32:E2:E5:5E	

FIG. 2 : Allocation d'adresse IP statique.

7.1.2 Redirection des ports

Ensuite, il est nécessaire d'effectuer la redirection des ports numéro 80 et 443.. Le protocole HTTP (HyperText Transfer Protocol) utilise le port 80 par défaut pour la transmission de données non sécurisées, tandis que le protocole HTTPS (HyperText Transfer Protocol Secure) utilise le port 443 pour la transmission de données sécurisées via SSL/TLS. En redirigeant ces ports, nous nous assurons que toutes les demandes HTTP et HTTPS entrant sur ces ports spécifiques sont correctement acheminées vers le serveur.

DHCP

NAT/PAT

DNS

UPnP

DynDNS

DMZ

NTP

IPv6

Vos règles personnalisées

Choisissez des ports qui ne sont pas bloqués par le pare-feu.

Nous vous déconseillons la création d'une règle sur le port 53 (service DNS).

Les équipements doivent être configurés avec une adresse IP statique pour être disponibles.

FTP Server

21

21

TCP

Android

Toutes

Créer

ex. : 1000

ex. : 1000-2000

IP externes autorisées

Activer	Application/Service	Port interne	Port externe	Protocole	Équipement	IP externe	
<input checked="" type="checkbox"/>	Web Server _HTTP_	80	80	TCP	Tesseract	Toutes	
<input checked="" type="checkbox"/>	Secure Web Server _HTTPS_	443	443	TCP	Tesseract	Toutes	

FIG. 3 : Redirection des ports n°80 & 443.

7.1.3 Pare-feu

Enfin, il est nécessaire de configurer le pare-feu du routeur pour autoriser le trafic entrant et sortant sur les ports 80 et 443. Le pare-feu, qui agit comme une barrière de sécurité entre le réseau interne et le réseau externe (Internet), utilise des règles pour autoriser ou bloquer le trafic.

FTP Server

21

TCP

Android

Toutes

Créer

ex. : 1000-2000

IP externes autorisées

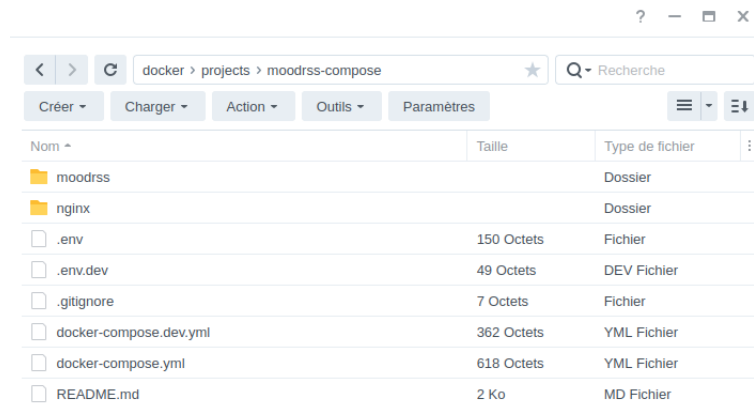
Activer	Application/Service	Port	Protocole	Équipement	Adresse IP externe	
<input checked="" type="checkbox"/>	Web Server (HTTP)	80	TCP	Tesseract	Toutes	
<input checked="" type="checkbox"/>	Secure Web Server (HTTPS)	443	TCP	Tesseract	Toutes	

FIG. 4 : Configuration du pare-feu.

7.2 Installation du conteneur Docker

L'installation des conteneurs Docker sur le serveur s'effectue grâce à une application prenant en charge la gestion de Docker. Ici, l'application proposée par Synology est "Container Manager".

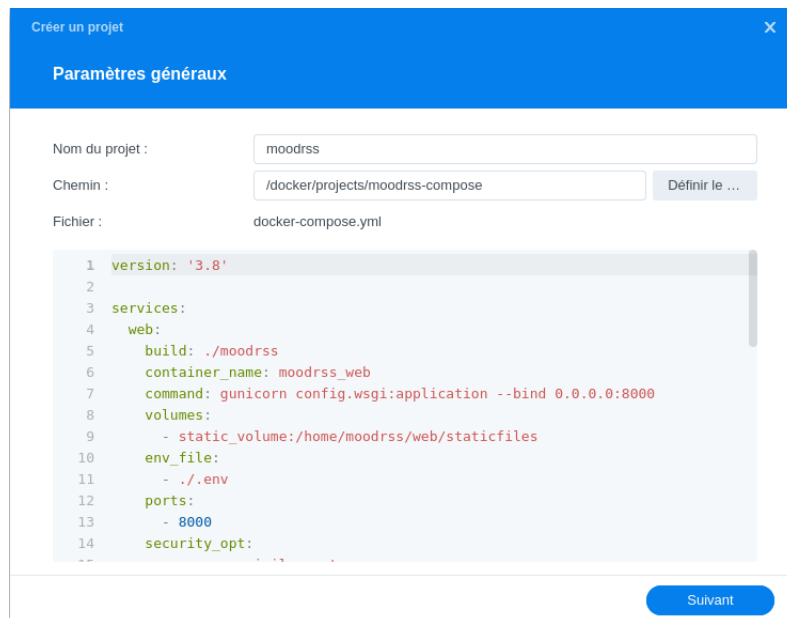
Le projet comprend deux conteneurs : "moodrss_web" pour l'application principale, et "moodrss_nginx" pour la configuration du serveur proxy Nginx. Une fois le fichier `docker-compose.yml` disponible suite à l'import des fichiers du projet dans le serveur (figure 5), l'installation des conteneurs se fait via le menu "Projet". Cette approche permet de monitorer conjointement l'installation des deux conteneurs.



Nom	Taille	Type de fichier
moodrss		Dossier
nginx		Dossier
.env	150 Octets	Fichier
.env.dev	49 Octets	DEV Fichier
.gitignore	7 Octets	Fichier
docker-compose.dev.yml	362 Octets	YML Fichier
docker-compose.yml	618 Octets	YML Fichier
README.md	2 Ko	MD Fichier

FIG. 5 : Import des fichiers dans le serveur.

Pour lancer l'installation, le nom du projet et le chemin d'accès vers les fichiers de celui-ci doivent être précisés. Le fichier YML est alors automatiquement détecté et l'installation commence (figure 6).



Créer un projet

Paramètres généraux

Nom du projet : moodrss

Chemin : /docker/projects/moodrss-compose Définir le ...

Fichier : docker-compose.yml

```

1 version: '3.8'
2
3 services:
4   web:
5     build: ./moodrss
6     container_name: moodrss_web
7     command: gunicorn config.wsgi:application --bind 0.0.0.0:8000
8     volumes:
9       - static_volume:/home/moodrss/web/staticfiles
10    env_file:
11      - ./env
12    ports:
13      - 8000
14    security_opt:

```

Suivant

FIG. 6 : Installation des conteneurs.

A l'issue de l'installation (figure 7), la présence et le fonctionnement des deux conteneurs "moodrss_nginx" et "moodrss_web" sont confirmés sur la page du projet "moodrss" (figure 8).

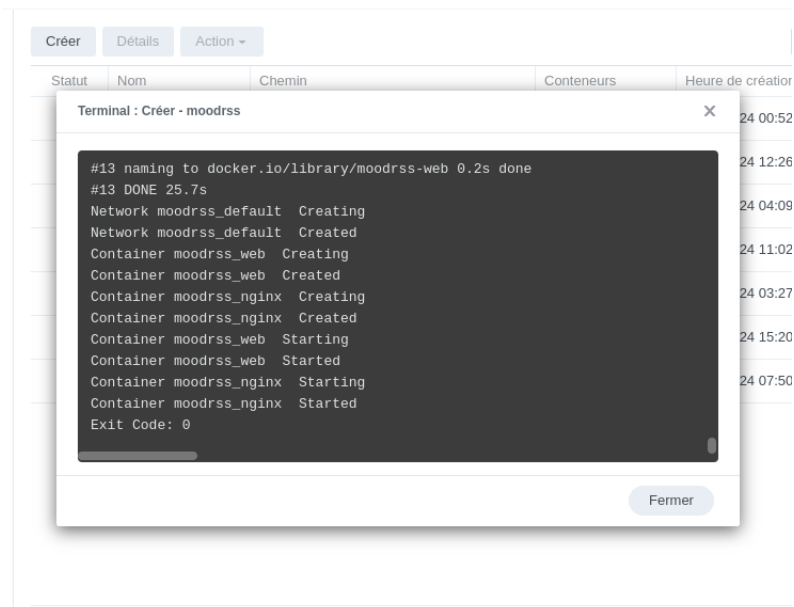


FIG. 7 : Fin de la création des conteneurs.

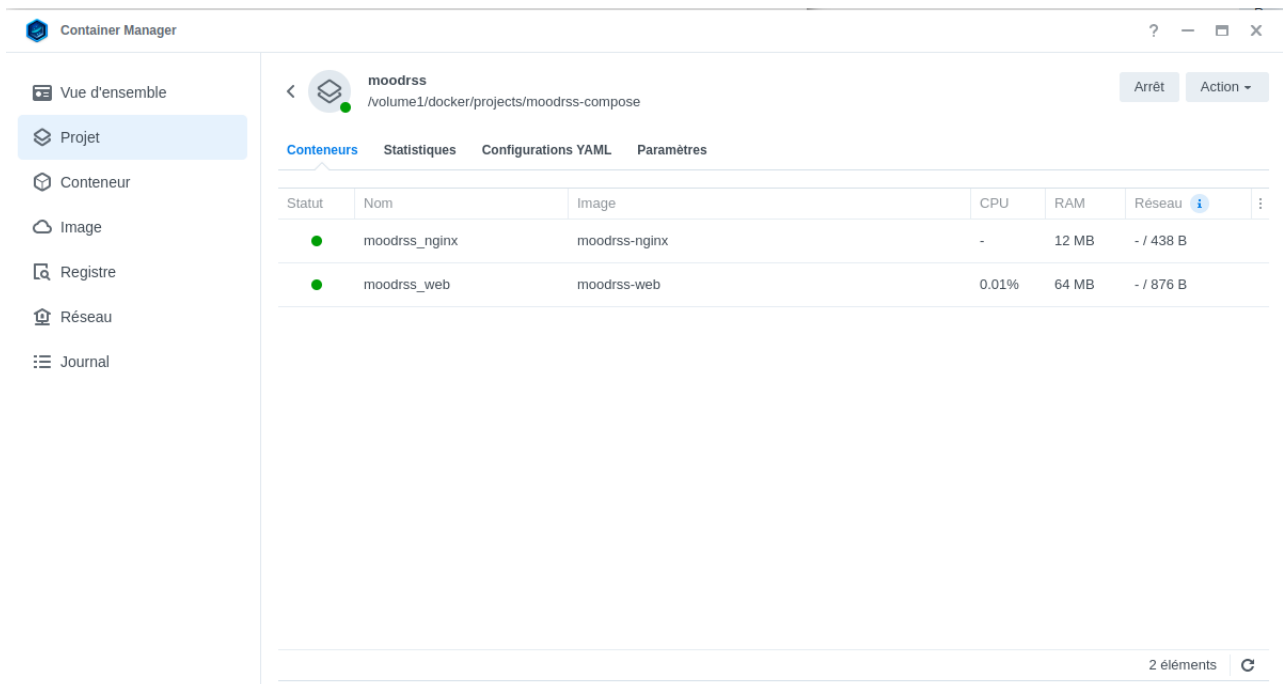


FIG. 8 : Visualisations des conteneurs exécutés.

7.3 Mise en Place de DDNS

Lorsque l'application MoodRSS est mise en place sur le serveur, l'activation du service DDNS (Dynamic Domain Name System) sur le serveur est la prochaine étape. Cela peut être réalisé en accédant aux paramètres "Accès Externe" du serveur (figure 9).

Modifier le DDNS

☒ Activer la prise en charge DDNS pour permettre aux utilisateurs d'accéder au serveur sous un nom d'hôte enregistré.

Fournisseur de service : Synology

Nom d'hôte : seizh . synology.me

Compte Synology : rlossouarn@live.fr

Adresse externe(IPv4) : Auto (90.21.158.71)

Adresse externe(IPv6) : Auto (-)

Statut : Normal Test de connexion

☐ Obtenez un certificat auprès de Let's Encrypt et définissez-le comme certificat par défaut

Remarque : nous transmettons votre domaine à Let's Encrypt afin qu'il soit enregistré. Pour plus d'informations, veuillez consulter notre [Déclaration de confidentialité](#).

☒ Activer Heartbeat ?

[Allez sur le site web du fournisseur DDNS](#)

Annuler OK

FIG. 9 : Activation du DDNS pour le serveur.

La spécification d'un nom d'hôte principal pour le serveur est nécessaire, étant donné qu'un sous-domaine sera attribué à l'application ultérieurement. Une adresse IPv4 est automatiquement assignée par le système.

Pour utiliser le protocole HTTPS de manière sécurisée, un certificat, délivré par Let's Encrypt sera délivré par défaut. Il convient de noter que cette option n'est pas visible sur la capture d'écran puisqu'elle est liée à une modification de paramètre.

Une fois ces modifications appliquées, le système indique que le nom d'hôte est activé et affiche l'adresse IP externe correspondante (figure 10).

Panneau de configuration

QuickConnect **DDNS** Configuration du routeur Avancé

Ajouter Modifier Supprimer Mettre à jour maintenant Personnaliser le fournisseur

Fournisseur de ser...	Nom d'hôte	Adresse externe	Statut
Synology	seizh.synology.me	90.21.158.71	Normal

1 élément

FIG. 10 : Visualisations du DDNS activé.

7.4 Mise en place de proxy inversé

L'accès à l'application depuis l'extérieur se fera via un sous-domaine qui redirige les requêtes vers le port du serveur Nginx de l'application, le conteneur moodrss_nginx. Pour réaliser cet accès, l'installation d'un proxy inversé est indispensable. En effet, dans ce contexte, le proxy inversé joue le rôle d'intermédiaire pour les demandes émanant des clients souhaitant accéder à l'application MoodRSS.

Les paramètres de configuration du proxy sont les suivants (figure 11) : la source est définie par le protocole HTTPS, le nom d'hôte et le port source 443. Le nom d'hôte est déterminé en fonction du sous-domaine attribué à l'application (moodrss.seizh.synology.me) et le port 443 correspond au protocole HTTPS.

Un élément supplémentaire à prendre en compte lors de la mise en place du proxy inversé est l'activation de HSTS (HTTP Strict Transport Security). HSTS est un mécanisme de sécurité web qui permet aux sites web de se déclarer comme accessibles uniquement via des connexions sécurisées. De plus, il informe les navigateurs web qu'ils doivent interagir avec eux uniquement en utilisant des connexions sécurisées HTTPS.

En ce qui concerne la destination, elle est basée sur le protocole HTTP car il s'agit à présent de communication interne au serveur. Le nom d'hôte spécifié est dans ce contexte "localhost" et le port 1009. Il convient de noter que ce port est celui indiqué dans le proxy Nginx du conteneur "mood_nginx".

Règles de proxy inversé

Général En-tête personnalisé Paramètres avancés

Nom du proxy inversé : moodrss

Source

Protocole : HTTPS

Nom d'hôte : moodrss.seizh.synology.me

Port : 443

☒ Activer HSTS

Profil de contrôle d'accès : Non configuré

Destination

Protocole : HTTP

Nom d'hôte : localhost

Port : 1009

Annuler Sauvegarder

FIG. 11 : Paramétrage du proxy inversé.

A Annexes

A.1 Annexes "3. Configuration de l'Environnement de Développement"

A.1.1 moodrss/requirements.txt

```
1 Django==5.0.6
2 asgiref==3.8.1
3 click==8.1.7
4 feedparser==6.0.11
5 joblib==1.4.2
6 nltk==3.8.1
7 python-dateutil==2.9.0.post0
8 regex==2024.5.15
9 sgmlib3k==1.0.0
10 six==1.16.0
11 sqlparse==0.5.0
12 tqdm==4.66.4
13 typing_extensions==4.12.2
14 gunicorn==22.0.0
```

A.2 Annexes "4. Paramètres du Projet Django"

A.2.1 /moodrss/config/settings.py

```
1 """
2 Django settings for helloworld project.
3
4 Generated by 'django-admin startproject' using Django 1.10.
5
6 For more information on this file, see
7 https://docs.djangoproject.com/en/1.10/topics/settings/
8
9 For the full list of settings and their values, see
10 https://docs.djangoproject.com/en/1.10/ref/settings/
11 """
12
13 from pathlib import Path
14 import os
15 import secrets
16
17
18 # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
19 BASE_DIR = Path(__file__).resolve().parent.parent
20
21 # Quick-start development settings - unsuitable for production
22 # See https://docs.djangoproject.com/en/1.10/howto/deployment/checklist/
23
24 # SECURITY WARNING: keep the secret key used in production secret!
25 SECRET_KEY = os.environ.get("SECRET_KEY", secrets.token_urlsafe(50))
26
27 # SECURITY WARNING: don't run with debug turned on in production!
28 DEBUG = bool(os.environ.get("DEBUG", True))
29
30 # Définit les domaines autorisés pour l'application
31 ALLOWED_HOSTS = os.environ.get("DJANGO_ALLOWED_HOSTS", "localhost").split(" ")
32
33 # Application definition
34
```

```

35 INSTALLED_APPS = [
36     'django.contrib.admin',
37     'django.contrib.auth',
38     'django.contrib.contenttypes',
39     'django.contrib.sessions',
40     'django.contrib.messages',
41     'django.contrib.staticfiles',
42     'rss' # ajout de l'application rss
43 ]
44
45 MIDDLEWARE = [
46     'django.middleware.security.SecurityMiddleware',
47     'django.contrib.sessions.middleware.SessionMiddleware',
48     'django.middleware.common.CommonMiddleware',
49     'django.middleware.csrf.CsrfViewMiddleware',
50     'django.contrib.auth.middleware.AuthenticationMiddleware',
51     'django.contrib.messages.middleware.MessageMiddleware',
52     'django.middleware.clickjacking.XFrameOptionsMiddleware',
53 ]
54
55 ROOT_URLCONF = 'config.urls'
56
57 TEMPLATES = [
58     {
59         'BACKEND': 'django.template.backends.django.DjangoTemplates',
60         'DIRS': [],
61         'APP_DIRS': True,
62         'OPTIONS': {
63             'context_processors': [
64                 'django.template.context_processors.debug',
65                 'django.template.context_processors.request',
66                 'django.contrib.auth.context_processors.auth',
67                 'django.contrib.messages.context_processors.messages',
68             ],
69         },
70     },
71 ]
72
73 WSGI_APPLICATION = 'config.wsgi.application'
74
75 # Database
76 # https://docs.djangoproject.com/en/5.0/ref/settings/#databases
77
78 DATABASES = {
79     'default': {
80         'ENGINE': 'django.db.backends.sqlite3',
81         'NAME': BASE_DIR / 'db.sqlite3',
82     }
83 }
84
85 # Password validation
86 # https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators
87
88 AUTH_PASSWORD_VALIDATORS = [
89     {
90         'NAME': 'django.contrib.auth.password_validation.
UserAttributeSimilarityValidator',
91     },
92     {
93         'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator'

```

```

94     },
95     {
96         'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
97     },
98     {
99         'NAME': 'django.contrib.auth.password_validation.
NumericPasswordValidator',
100     },
101 ]
102
103 # Internationalization
104 # https://docs.djangoproject.com/en/5.0/topics/i18n/
105
106 LANGUAGE_CODE = 'en-us'
107
108 TIME_ZONE = 'UTC'
109
110 USE_I18N = True
111
112 USE_TZ = True
113
114 # Static files (CSS, JavaScript, Images)
115 # https://docs.djangoproject.com/en/5.0/howto/static-files/
116
117 STATIC_URL = '/static/'
118 STATIC_ROOT = BASE_DIR / 'staticfiles'
119
120 # Default primary key field type
121 # https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field
122
123 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
124
125 # Définit les URL de confiance pour la protection CSRF
126
127 CSRF_TRUSTED_ORIGINS = os.environ.get("CSRF_TRUSTED_ORIGINS", "http://localhost
:8000").split(" ")

```

A.2.2 /moodrss/config/urls.py

```

1  """URL Configuration
2
3  The `urlpatterns` list routes URLs to views. For more information please see:
4      https://docs.djangoproject.com/en/1.10/topics/http/urls/
5  Examples:
6  Function views
7      1. Add an import:  from my_app import views
8      2. Add a URL to urlpatterns:  url(r'^$', views.home, name='home')
9  Class-based views
10     1. Add an import:  from other_app.views import Home
11     2. Add a URL to urlpatterns:  url(r'^$', Home.as_view(), name='home')
12  Including another URLconf
13     1. Import the include() function: from django.conf.urls import url, include
14     2. Add a URL to urlpatterns:  url(r'^blog/', include('blog.urls'))
15  """
16
17 # from django.contrib import admin
18 from django.urls import path
19 from rss.views import dashboard
20

```



```

21 urlpatterns = [
22     # désactivation de la page d'administration
23     # path('admin/', admin.site.urls),
24     path('', dashboard, name="dashboard") # ajout de la vue contenant la page
    dashboard
25 ]

```

A.3 Annexes "5. Développement de l'Application"

A.3.1 /moodrss/rss/parser.py

```

1  ''' Nom : parser.py
2  Rôle : Programme qui récupère et analyse les flux RSS et nettoie le contenu HTML
   .
3  Auteur : Ronan LOSSOUARN
4  Date : 03/04/2024
5  Licence : L3 Réalisation de programme
6
7  DESCRIPTION
8  Le programme récupère un flux RSS depuis une URL et nettoie le contenu HTML des
   résumés des entrées.
9  '''
10
11 import feedparser
12 import re
13 from html import unescape
14 from dateutil.parser import parse as dateparse
15 import unittest
16
17
18 # Nettoie le contenu HTML brut en supprimant les balises HTML et en
   convertissant les caractères spéciaux.
19 # @param raw_html -> le contenu HTML brut ;
20 # @return -> le texte nettoyé sans balises HTML.
21 def clean_html(html_text):
22     # Suppression des balises HTML
23     clean_text = re.sub('<.*?>', '', html_text)
24     # Conversion des caractères spéciaux
25     return unescape(clean_text)
26
27
28 # Récupère un flux RSS depuis une URL, nettoie le contenu HTML des résumés et
   récupère les dates de publication.
29 # @param url -> l'URL du flux RSS ;
30 # @return -> une liste d'entrées de flux nettoyées, ou "Not found" si aucune
   entrée n'est trouvée.
31 def prepare_rss_feed(url):
32     # Parcourt le flux RSS
33     feed = feedparser.parse(url)
34     # Vérifie s'il y a des entrées dans le flux
35     if feed.entries:
36         for entry in feed.entries:
37             # Nettoie le résumé
38             entry.summary = clean_html(entry.summary)
39             # Récupère la date de publication
40             entry.published = dateparse(entry.published)
41         return feed.entries
42     else:
43         return "Not found"
44
45

```

```

46 class TestParser(unittest.TestCase):
47     # Test avec une entrée
48     def test_with_entries(self):
49         rss_content = """
50         <rss version="2.0">
51             <channel>
52                 <title>Example RSS Feed</title>
53                 <description>This is an example RSS feed</description>
54                 <item>
55                     <title>Exemple</title>
56                     <description><![CDATA[Test résumé avec du contenu <b>HTML</b>
57 >.]></description>
58                     <pubDate>Tue, 21 May 2024 16:36:44 +0000</pubDate>
59                 </item>
60             </channel>
61         </rss>
62         """
63         result = prepare_rss_feed(rss_content)
64         # Vérifie que la fonction retourne une entrée
65         self.assertEqual(len(result), 1)
66         # Vérifie que le résumé de l'entrée est nettoyé correctement
67         self.assertEqual(result[0].summary, "Test summary with HTML content.")
68
69     # Test sans une entrée
70     def test_without_entries(self):
71         rss_content = """
72         <rss version="2.0">
73             <channel>
74                 <title>Exemple RSS</title>
75                 <description>RSS</description>
76             </channel>
77         </rss>
78         """
79         result = prepare_rss_feed(rss_content)
80         # Vérifie que la fonction retourne "Not found" quand il n'y a pas d'
entrées
81         self.assertEqual(result, "Not found")
82
83     # Test de la fonction clen_html()
84     def test_clean_html(self):
85         # HTML brut
86         raw_html = "<p>Test summary with <b>HTML</b> content.</p>"
87         # Résultat attendu après nettoyage du HTML
88         expected_output = "Test summary with HTML content."
89         # Vérifier que la fonction clean_html nettoie le HTML correctement
90         self.assertEqual(clean_html(raw_html), expected_output)
91
92 if __name__ == "__main__":
93     unittest.main()

```

A.3.2 /moodrss/rss/prediction.py

```

1  ''' Nom : prediction.py
2  Rôle : Programme qui analyse le sentiment des titres et résumés d'articles en
    utilisant VADER.
3  Auteur : Ronan LOSSOUARN
4  Date : 03/04/2024
5  Licence : L3 Réalisation de programme
6
7  DESCRIPTION

```

```

8 Le programme combine un titre et un résumé d'article en un seul texte et utilise
  l'analyseur de sentiment VADER pour évaluer le sentiment global de ce texte.
  En fonction du score de sentiment global (score compound), le programme
  retourne une prédiction de sentiment sous forme de chaîne de caractères : '
  Positive' si le score est positif, 'Negative' si le score est négatif, '
  Neutral' si le score est neutre.
9 '''
10
11 import nltk
12 from nltk.sentiment.vader import SentimentIntensityAnalyzer
13 import unittest
14
15
16 # Analyse le sentiment d'un titre et d'un résumé pour évaluer le sentiment
  global.
17 # @param title -> le titre de l'article ;
18 # @param summary -> le résumé de l'article ;
19 # @return -> chaîne de caractères, 'Positive', 'Negative' ou 'Neutral'.
20 def get_prediction(title, summary):
21     try:
22         nltk.data.find('sentiment/vader_lexicon.zip')
23     except LookupError:
24         nltk.download('vader_lexicon')
25
26     text = title + " " + summary
27
28     # Analyse des scores de sentiment pour le texte
29     sia = SentimentIntensityAnalyzer()
30     sentiment_scores = sia.polarity_scores(text)
31
32     # Extraction du score compound (global) des sentiments
33     compound_score = sentiment_scores['compound']
34
35     # Sélection du sentiment
36     if compound_score >= 0.1:
37         return 'Positive'
38     elif compound_score <= -0.1:
39         return 'Negative'
40     else:
41         return 'Neutral'
42
43
44 class TestPrediction(unittest.TestCase):
45
46     def test_positive(self):
47         title = "Paddington In Peru Trailer Sends The Adorable Bear On A New
  Adventure, And I'm Ready To Follow Him Anywhere"
48         summary = "The Paddington series of movies ranks as the most delightful,
  and Paddington in Peru looks to extend the winning streak."
49         self.assertEqual(get_prediction(title, summary), 'Positive')
50
51     def test_negative(self):
52         title = "Someone Finally Asked Austin Butler About Those Pirates Of The
  Caribbean Rumors"
53         summary = "Austin Butler responds to rumors he could star in the
  next Pirates of the Caribbean movie."
54         self.assertEqual(get_prediction(title, summary), 'Negative')
55
56     def test_neutral(self):
57         title = "Late Night With Seth Meyers Staff Losing Jobs Amidst Budget

```

```

    Cuts: 'Nobody Wants To Pay"
58     summary = "As part of budget cuts at NBC, Late Night with Seth Meyers is
        cutting a notable portion of its staff."
59     self.assertEqual(get_prediction(title, summary), 'Neutral')
60
61
62 if __name__ == "__main__":
63     unittest.main()

```

A.3.3 /moodrss/rss/views.py

```

1  ''' Nom : views.py
2  Rôle : Vue Django pour analyser les flux RSS, déterminer le sentiment des
        articles et les afficher sur un tableau de bord.
3  Auteur : Ronan LOSSOUARN
4  Date : 04/04/2024
5  Licence : L3 Réalisation de programme
6
7  DESCRIPTION
8  Cette vue Django récupère un flux RSS depuis une URL fournie par l'utilisateur,
        nettoie le contenu HTML des résumés, analyse le sentiment des articles en
        utilisant VADER, et affiche les articles. Les articles peuvent être filtrés
        par sentiment. La vue gère les requêtes POST pour traiter les données
        soumises par l'utilisateur. '''
9
10 from django.shortcuts import render
11 from rss.parser import prepare_rss_feed
12 from rss.prediction import get_prediction
13
14
15 # Traite les requêtes pour afficher un tableau de bord des flux RSS avec analyse
    de sentiment.
16 # @param request -> l'objet de requête HTTP ;
17 # @return -> rend la page HTML du tableau de bord avec les flux RSS analysés et
    filtrés par sentiment.
18 def dashboard(request):
19     if request.method == 'POST':
20         # Récupère les données soumises par l'utilisateur
21         data = request.POST
22         url = data['url'] # URL du flux RSS entré par l'utilisateur
23         feeds = prepare_rss_feed(url) # Prépare le flux RSS
24
25         # Vérifie si un filtre de sentiment a été sélectionné
26         if data['sentiment']:
27             selected = data['sentiment']
28
29         # Vérifie si des flux ont été trouvés
30         if feeds != "Not found":
31             found = 'true'
32
33         unique_feeds = []
34         seen_titles = set()
35
36         # Élimine les doublons
37         for feed in feeds:
38             if feed.title not in seen_titles:
39                 unique_feeds.append(feed)
40                 seen_titles.add(feed.title)
41
42         for feed in unique_feeds:
43             # Analyse le sentiment du titre et du résumé

```

```

44         feed['sentiment'] = get_prediction(feed.title, feed.summary)
45         # Ajoute du texte secondaire en fonction du sentiment analysé
46         if feed['sentiment'] == 'Positive':
47             feed['secondary_text'] = 'This feed brings positive content.
48
49             elif feed['sentiment'] == 'Negative':
50                 feed['secondary_text'] = 'The tone of this feed is more
51                 negative.'
52
53                 else:
54                     feed['secondary_text'] = 'This feed provides articles with a
55                     neutral tone.'
56
57             # Affecte le nom de l'auteur s'il est disponible
58             if 'authors' in feed:
59                 feed['author_name'] = feed.authors[0]['name']
60             else:
61                 feed['author_name'] = None
62
63             # Affecte le lien du flux s'il est disponible
64             if 'link' in feed:
65                 feed['link'] = feed.link
66             else:
67                 feed['link'] = None
68
69             # Affecte la date de publication si elle est disponible
70             if 'published' in feed:
71                 feed['published'] = feed.published
72             else:
73                 feed['published'] = None
74
75             # Filtre les flux par sentiment sélectionné
76             selected_sentiment = data.get('sentiment', 'All') # Par défaut
77             if selected_sentiment != 'All':
78                 unique_feeds = [feed for feed in unique_feeds if feed['sentiment'] == selected_sentiment]
79
80             # Vérifie s'il y a des flux à afficher et rend la page avec les flux
81             if len(unique_feeds) > 0:
82                 return render(request, "dashboard.html", {'feeds': unique_feeds,
83                 'found': found, 'selected': selected, 'url': url, 'empty': 'no', 'feed':
84                 unique_feeds[0]})
85             else:
86                 # Rend la page indiquant qu'il n'y a aucun flux à afficher
87                 return render(request, "dashboard.html", {'feeds': unique_feeds,
88                 'found': found, 'selected': selected, 'url': url, 'empty': 'yes'})
89
90             else:
91                 # Indique que le flux RSS n'a pas été trouvé
92                 found = 'false'
93                 return render(request, "dashboard.html", {'feeds': None, 'found':
94                 found, 'selected': selected})
95
96             else:
97                 # Rend la page du tableau de bord vide pour une requête GET
98                 return render(request, "dashboard.html")

```

A.3.4 /moodrss/rss/tests.py

```

1 from django.test import TestCase, Client
2 from django.urls import reverse
3

```

```

4
5 class MoodRSSTests(TestCase):
6
7     # Prépare l'environnement de test
8     def setUp(self):
9         self.client = Client()
10        self.url = reverse('dashboard')
11
12    # Test de la requête GET pour vérifier que la page se charge correctement
    sans flux RSS
13    def test_get(self):
14        response = self.client.get(self.url)
15        self.assertEqual(response.status_code, 200)
16        self.assertTemplateUsed(response, 'dashboard.html')
17        self.assertContains(response, 'MoodRSS')
18
19    # Test de la requête POST avec une URL de flux RSS valide
20    def test_post_valid_rss(self):
21        response = self.client.post(self.url, {'url': 'https://www.cinemablend.
    com/feeds.xml', 'sentiment': 'All'})
22        self.assertEqual(response.status_code, 200)
23        # Vérifie qu'il ne contient pas le message "Feed not found"
24        self.assertNotContains(response, 'Feed not found')
25        # Vérifie que des flux sont retournés
26        self.assertGreater(len(response.context['feeds']), 0)
27
28    # Test de la requête POST avec une URL de flux RSS invalide
29    def test_post_invalid_rss(self):
30        response = self.client.post(self.url, {'url': 'http://invalid-rss-feed.
    com/rss', 'sentiment': 'All'})
31        self.assertEqual(response.status_code, 200)
32        self.assertContains(response, 'Feed not found')
33
34    # Test de la requête POST avec un filtre de sentiment
35    def test_post_sentiment_filter(self):
36        # Effectue une requête avec un filtre de sentiment
37        response = self.client.post(self.url, {'url': 'https://www.cinemablend.
    com/feeds.xml', 'sentiment': 'Positive'})
38        self.assertEqual(response.status_code, 200)
39        self.assertNotContains(response, 'Feed not found')
40        # Vérifie que tous les flux retournés ont un sentiment positif
41        for feed in response.context['feeds']:
42            self.assertEqual(feed['sentiment'], 'Positive')

```

A.3.5 /moodrss/rss/templates/dashboard.html

```

1 {% load static %}
2 <!DOCTYPE html>
3 <html lang="en">
4     <head>
5         <meta charset="UTF-8">
6         <meta name="viewport" content="width=device-width, initial-scale=1,
    shrink-to-fit=no">
7
8         <!-- Fichier CSS de Bootstrap -->
9         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/
    bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/
    azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3yD65VohhpuaC0mLASjC" crossorigin="
    anonymous">
10
11        <!-- Fichier CSS personnalisé -->
12        <link rel="stylesheet" href="{% static 'css/style.css' %}">

```

```

12     </head>
13
14     <body>
15         <div id="content" >
16             <!-- En-tête de la page -->
17             <div class="d-flex align-items-center justify-content-center my-3"
18 style="border-bottom: 1px solid rgb(221, 218, 218);">
19                 <h1 style="font-family: 'Trebuchet MS', 'Lucida Sans Unicode', '
20 Lucida Grande', 'Lucida Sans', Arial, sans-serif; color: #3068ff;"><b>MoodRSS
21 </b></h1>
22             </div>
23             <!-- Formulaire de recherche pour les flux RSS -->
24             <div id="search-container" class="d-flex flex-column align-items-
25 center justify-content-center">
26                 <p class="search-text"><b>Search for Feeds</b></p>
27                 <form class="input-group w-50" action="/" method="post">
28                     {% csrf_token %}
29                     <!-- Affiche l'URL entré s'il existe, sinon un champ vide --
30 >
31                     {% if url %}
32                         <input type="text" class="form-control my-input"
33 placeholder="Search for RSS Feeds..." name="url" value="{{url}}">
34                     {% else %}
35                         <input type="text" class="form-control my-input"
36 placeholder="Search for RSS Feeds..." name="url">
37                     {% endif %}
38                     <!-- Valeur du sentiment par défaut sur 'All' -->
39                     <input type="hidden" name="sentiment" value="All">
40                     <div class="input-group-append" style="margin-left: -2px;">
41                         <!-- Bouton de soumission du formulaire -->
42                         <button class="btn my-btn" type="submit" style="border-
43 top-left-radius: 0px; border-bottom-left-radius: 0px;">Search</button>
44                     </div>
45                 </div>
46
47                 <div class="row">
48                     <!-- Affiche un message si aucun flux n'est trouvé -->
49                     {% if found == 'false' %}
50                         <div class="col-md-12 px-5 py-5" style="min-height: 180px;">
51                             <div class="h-100 d-flex justify-content-center align-
52 items-center error-div" style="border-radius: 10px;">
53                                 <h3 style="font-family: 'Trebuchet MS', 'Lucida Sans
54 Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif; color: #EF2917;
55 ">Feed not found!</h3>
56                             </div>
57                         </div>
58                     {% else %}
59                         <!-- Affiche le menu déroulant si un URL est présent -->
60                         {% if url %}
61                             <div class="col-md-12 d-flex justify-content-center
62 align-items-center" style="min-height: 50px;">
63                                 <div class="dropdown">
64                                     <button class="btn dropdown-toggle filter-button
65 " type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" aria-
66 expanded="false">
67                                         Filter by Mood
68                                     </button>
69                                     <ul class="dropdown-menu my-dropdown" aria-
70 labelledby="dropdownMenuButton1">
71                                         <!-- Boutons pour filtrer les flux par

```

```

sentiment -->
57         <li><button class="dropdown-item" type="
submit" name="sentiment" value="All">All</button></li>
58         <li><button class="dropdown-item" type="
submit" name="sentiment" value="Positive">Positive</button></li>
59         <li><button class="dropdown-item" type="
submit" name="sentiment" value="Negative">Negative</button></li>
60         <li><button class="dropdown-item" type="
submit" name="sentiment" value="Neutral">Neutral</button></li>
61     </ul>
62 </div>
63 </form>
64 </div>
65 {% endif %}
66
67 <!-- Affiche le sentiment des flux filtrés si un filtre
autre que 'All' est sélectionné -->
68 {% if selected != 'All' and empty == 'no' %}
69     <div class="feeds-div-sentiment" id="feeds-div-sentiment
">
70         <div class="d-flex flex-column align-items-
center justify-content-center h-100 search-content py-3 px-5" style="text-
align: center;">
71             <!-- Affiche le sentiment avec des
couleurs spécifiques -->
72             {% if feed.sentiment == 'Positive' %}
73                 <p class="sentiment" style="color
:#008148"><b>{{ feed.sentiment }}</b></p>
74             {% elif feed.sentiment == 'Negative' %}
75                 <p class="sentiment" style="color:#
EF2917"><b>{{ feed.sentiment }}</b></p>
76             {% else %}
77                 <p class="sentiment" style="color
:#46494C"><b>{{ feed.sentiment }}</b></p>
78             {% endif %}
79             <p class="sentiment-text">{{feed.
secondary_text}}</p>
80         </div>
81     </div>
82 {% endif %}
83 <!-- Boucle sur les flux pour les afficher -->
84 {% for feed in feeds %}
85     <!-- Affiche les détails du flux si aucun filtre ou
filtre 'All' est sélectionné -->
86     {% if selected == 'All' %}
87         <div class="col-md-12 feeds-div-content">
88             <div class="search-content h-100 py-3 px-4">
89                 <div class="d-flex flex-row">
90                     <p class="feed-title" style="flex: 1"><b>{{
feed.title}}</b></p>
91                     <!-- Lien pour ouvrir le flux dans un nouvel
onglet -->
92                     <a class="btn btn-primary my-open-btn ml-
auto" href="{{ feed.link }}" target="_blank"><span>Open</span></a>
93                 </div>
94                 <p style="width: 70%;">{{feed.summary}}</p>
95                 <!-- Affiche l'auteur du flux si disponible -->
96                 {% if feed.author_name %}
97                     <p>{{feed.author_name}}</p>
98                 {% endif %}

```



```

99         <!-- Affiche la date de publication formatée -->
100         <p>{{feed.published|date:"d F Y, H:i"}}</p>
101     </div>
102 </div>
103 {% else %}
104 <!-- Affiche les détails du flux filtré par sentiment --
105 >
106     <div class="feeds-div-content">
107         <div class="search-content h-100 py-3 px-4">
108             <div class="d-flex flex-row">
109                 <p class="feed-title" style="flex: 1;"><b>{{
110 feed.title}}</b></p>
111                 <a class="btn btn-primary my-open-btn ml-
112 auto" href="{{ feed.link }}" target="_blank"><span>Open</span></a>
113             </div>
114             <p style="width: 70%;">{{feed.summary}}</p>
115             {% if feed.author_name %}
116                 <p>{{feed.author_name}}</p>
117             {% endif %}
118             <p>{{feed.published|date:"d F Y, H:i"}}</p>
119         </div>
120     </div>
121     {% endif %}
122
123     <!-- Affiche une ligne de séparation entre les flux si
124 un sentiment est défini -->
125     <div class="col-md-12 px-5" style="padding-top: 2rem;
126 padding-bottom: 0.7rem;">
127         <div class="" style="border-top: 1px solid rgb(221,
128 218, 218);">
129     </div>
130 </div>
131     {% endfor %}
132     {% endif %}
133 </div>
134 </div>
135
136 <!-- Scripts pour le menu déroulant -->
137 <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/
138 popper.min.js" integrity="sha384-IQsoLX15PILFhosVNubq5LC7Qb9DXgDA9i+
139 tQ8Zj3iWAwPtgFTxbJ8NT4GN1R8p" crossorigin="anonymous"></script>
140 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/
141 bootstrap.min.js" integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/
142 bglldoEyl4H0zUF0QKbrJ0EcQF" crossorigin="anonymous"></script>
143
144 </body>
145 </html>

```

A.3.6 /moodrss/rss/static/style.css

```

1 body {
2     /* Cache les débordements horizontaux */
3     overflow-x: hidden;
4 }
5
6 /* Conteneur de la barre de recherche*/
7 #search-container {
8     margin-bottom: 20px;
9     min-height: 180px;
10    border-bottom: 1px solid rgb(221, 218, 218);
11 }

```

```

12
13 /* Texte du nom de la recherche */
14 .search-text {
15     font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
16     Sans', Arial, sans-serif;
17     font-size: x-large;
18 }
19 /*Bouton Search*/
20 .my-btn {
21     background-color: #3068ff;
22     color: #F8F8F8;
23 }
24
25 /*Barre de recherche*/
26 .my-input {
27     border: 1px solid #3068ff;
28     border-top-left-radius: 5px !important;
29     border-bottom-left-radius: 5px !important;
30 }
31
32 /*Contenu de la recherche*/
33 .search-content {
34     border: 1px solid rgb(221, 218, 218);
35     border-radius: 10px;
36 }
37
38 /*Survol du contenu de la recherche*/
39 .search-content:hover {
40     border: 1px solid #3068ff;
41 }
42
43 /*Titre du sentiment*/
44 .sentiment {
45     font-size: xx-large;
46     font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
47     Sans', Arial, sans-serif;
48 }
49 /*Description du sentiment*/
50 .sentiment-text {
51     font-size: x-large;
52     font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
53     Sans', Arial, sans-serif;
54 }
55 /*Titre d'article*/
56 .feed-title {
57     font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
58     Sans', Arial, sans-serif;
59     font-size: 20px;
60 }
61 /*Bouton d'ouverture de lien*/
62 .my-open-btn {
63     background-color: #3068ff;
64     width: 5rem;
65     height: 2.2rem;
66 }
67

```

```

68 /*Ecran de moins de 991px*/
69 @media (max-width: 991px) {
70     /*Barre de recherche*/
71     .input-group {
72         width: 90% !important;
73     }
74
75     /*Marges pour les différents liens*/
76     .feeds-div-content {
77         padding-left: 30px;
78         padding-right: 30px;
79         padding-top: 20px;
80     }
81
82     /*Marges pour lle sentiment selectionne*/
83     .feeds-div-sentiment {
84         padding-left: 30px;
85         padding-right: 30px;
86         padding-top: 20px;
87     }
88 }
89
90 /*Ecran de plus de 991px*/
91 @media (min-width: 991px) {
92     /*Marges pour les différents liens*/
93     .feeds-div-content {
94         padding-left: 40px;
95         padding-right: 40px;
96         padding-top: 20px;
97     }
98
99     /*Marges pour lle sentiment selectionne*/
100    .feeds-div-sentiment {
101        padding-right: 40px;
102        padding-left: 40px;
103        padding-top: 20px;
104    }
105 }
106
107 /*Bordure de la division si aucun flux n'est trouvé*/
108 .error-div {
109     border: 1px solid rgb(221, 218, 218);
110 }
111
112 /*Change la couleur de la bordure au survol*/
113 .error-div:hover {
114     border: 1px solid #3068ff;
115 }
116
117 /* Supprime l'ombre et le contour des formulaire et des boutons */
118 .form-control, .btn {
119     box-shadow: none !important;
120     outline: none !important;
121 }
122
123 /* Définit la bordure et la couleur du texte pour le bouton de filtre */
124 .filter-button {
125     border: 1px solid #3068ff;
126     color: #3068ff;
127 }

```

A.4 Annexes "6. Packaging"

A.4.1 /.env.dev

```
1 DEBUG=True
2 DJANGO_ALLOWED_HOSTS=localhost 0.0.0.0
```

A.4.2 /moodrss/Dockefile

```
1 #####
2 # BUILDER #
3 #####
4
5 # Image de base du conteneur
6 FROM python:3.11.4-slim-buster as builder
7
8 # Définit le répertoire de travail
9 WORKDIR /usr/src/moodrss
10
11 # Variables d'environnement pour optimiser les performances de Python
12 ENV PYTHONUNBUFFERED 1 \
13     PYTHONDONTWRITEBYTECODE 1
14
15 # Copie le code source de l'application
16 COPY . /usr/src/moodrss/
17
18 # Controle des conventions de code
19 RUN pip install --upgrade pip
20 RUN pip install flake8==6.0.0
21 RUN flake8 --ignore=E501,F401 .
22
23 # Installe les dépendances
24 COPY ./requirements.txt .
25 RUN pip wheel --no-cache-dir --no-deps --wheel-dir /usr/src/moodrss/wheels -r
    requirements.txt
26
27 #####
28 # FINAL #
29 #####
30
31 FROM python:3.11.4-slim-buster
32
33 # Crée l'utilisateur de l'application
34 RUN addgroup --system moodrss && adduser --system --ingroup moodrss moodrss
35
36 # Crée les répertoires appropriés
37 ENV APP_HOME=/home/moodrss/web
38 RUN mkdir -p $APP_HOME/staticfiles
39 WORKDIR $APP_HOME
40
41 # Copie des dépendances depuis l'étape de construction
42 COPY --from=builder /usr/src/moodrss/wheels /wheels
43 COPY --from=builder /usr/src/moodrss/requirements.txt .
44
45 # Installe les dépendances
46 RUN pip install --no-cache-dir --upgrade pip
47 RUN pip install --no-cache /wheels/*
48
49 # Copie le code source de l'application
50 COPY . $APP_HOME
51
```

```
52 # Change l'utilisateur
53 USER moodrss
```

A.4.3 /docker-compose.dev.yml

```
1 version: '3.8'
2
3 services:
4   web:
5     build: ./moodrss
6     container_name: moodrss_web
7     command: python manage.py runserver 0.0.0.0:8000
8     volumes:
9       - static_volume:/home/moodrss/web/staticfiles
10    ports:
11      - 8000:8000
12    env_file:
13      - ./env.dev
14    security_opt:
15      - no-new-privileges:true
16    restart: always
17
18 volumes:
19   static_volume:
```

A.4.4 /Makefile

```
1 # Variables
2 DOCKER_COMPOSE = docker-compose
3 COMPOSE_FILE ?= docker-compose.yml
4
5 # Défini le fichier docker-compose en fonction de l'environnement
6 ifeq ($(ENV),dev)
7   COMPOSE_FILE = docker-compose.dev.yml
8 endif
9
10
11 build: # Construit les images Docker
12   $(DOCKER_COMPOSE) -f $(COMPOSE_FILE) up -d --build
13
14 down: # Arrête les services
15   $(DOCKER_COMPOSE) -f $(COMPOSE_FILE) down -v
16
17 logs: # Affiche les logs des services
18   $(DOCKER_COMPOSE) -f $(COMPOSE_FILE) logs -f
19
20 clean: # Supprime les conteneurs, images, volumes et réseaux
21   $(DOCKER_COMPOSE) -f $(COMPOSE_FILE) down --rmi all --volumes --remove-orphans
```

A.4.5 /nginx/nginx.conf

```
1 upstream moodrss {
2     server web:8000;
3 }
4
5 server {
6
7     listen 80;
8
9     location / {
```

```

10     proxy_pass http://moodrss;
11     proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
12     proxy_set_header Host $host;
13     proxy_redirect off;
14 }
15
16 location /static/ {
17     alias /home/moodrss/web/staticfiles/;
18 }
19
20 }

```

A.4.6 /nginx/Dockerfile

```

1 # Image officielle
2 FROM nginx:1.25
3
4 # Supprime la configuration par défaut de NGINX
5 RUN rm /etc/nginx/conf.d/default.conf
6
7 # Copie le fichier de configuration personnalisé dans le répertoire de
  configuration
8 COPY nginx.conf /etc/nginx/conf.d

```

A.4.7 /.env

```

1 SECRET_KEY=secrets.token_urlsafe(50)
2 DEBUG=False
3 DJANGO_ALLOWED_HOSTS=moodrss.seizh.synology.me localhost
4 CSRF_TRUSTED_ORIGINS=https://moodrss.seizh.synology.me http://localhost:1009

```

A.4.8 /docker-compose.yml

```

1 version: '3.8'
2
3 services:
4   web:
5     build: ./moodrss
6     container_name: moodrss_web
7     command: gunicorn config.wsgi:application --bind 0.0.0.0:8000
8     volumes:
9       - static_volume:/home/moodrss/web/staticfiles
10    env_file:
11      - ../.env
12    ports:
13      - 8000
14    security_opt:
15      - no-new-privileges:true
16    restart: always
17
18    nginx:
19      build: ./nginx
20      container_name: moodrss_nginx
21      volumes:
22        - static_volume:/home/moodrss/web/staticfiles
23      ports:
24        - 1009:80
25      depends_on:
26        - web
27      security_opt:
28        - no-new-privileges:true

```

```
29     restart: always
30
31 volumes:
32     static_volume:
```

B Ressources

McKee, K., & Pilgrim, M. (s.d.). Common RSS elements — FeedParser 6.0.11 documentation. Feedparser. <https://feedparser.readthedocs.io/>

Niemeyer, G., Pieviläinen, T., De Leeuw, Y., & Ganssle, P. (2003). DateUtil - Powerful extensions to DateTime — DateUtil 3.9.0 documentation. Dateutil. <https://dateutil.readthedocs.io/>

Hutto, C.J. & Gilbert, E.E. (2014). VADER : A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014 <https://www.nltk.org/>