

DATASCIENTEST - JOB MARKET

Projet Data engineer - Emmanuelle LASTRUCCI- Célestine SAUVAGE-
Fabien WAY

Présentation

Afin de nous concentrer sur l'essentiel et répondre au mieux à l'attendu tel qu'énoncé dans la fiche du projet "Job Market", nous avons décidé de restreindre le périmètre de notre travail sur les offres d'emploi situées en France uniquement.

Mise en place de l'infrastructure

En parallèle de la partie "choix et récupération des données", nous avons souhaité mettre en place le plus rapidement possible une pratique de développement commune. Voici les différents choix qui ont été faits:

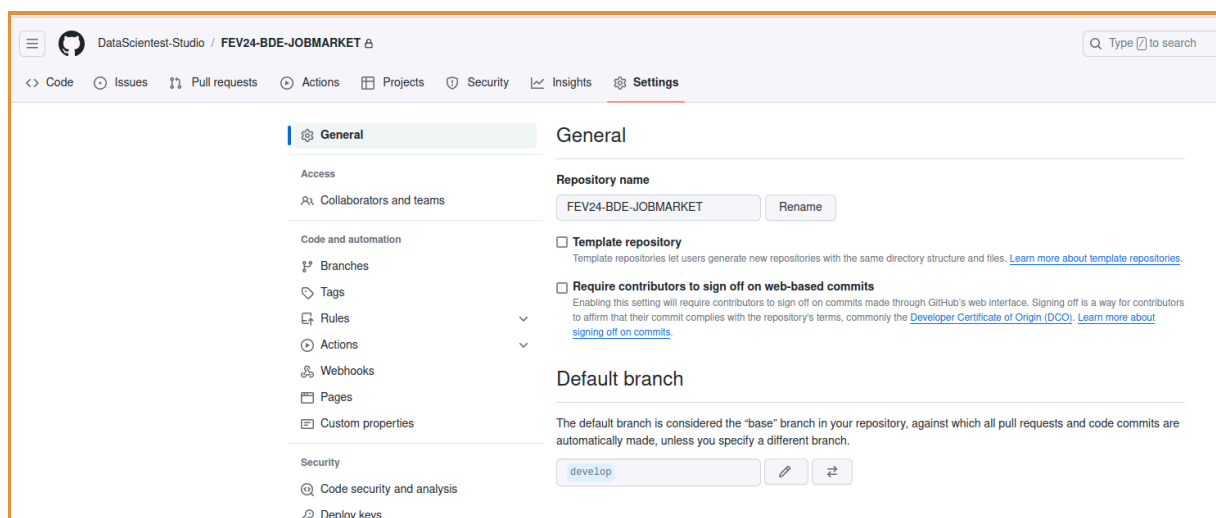
Environnement virtuel

Nous avons décidé d'utiliser un environnement virtuel python avec la version 3.12. En effet, d'après ce [site](#), cette version est la dernière version stable python et dont la fin de vie est prévue pour fin 2028.

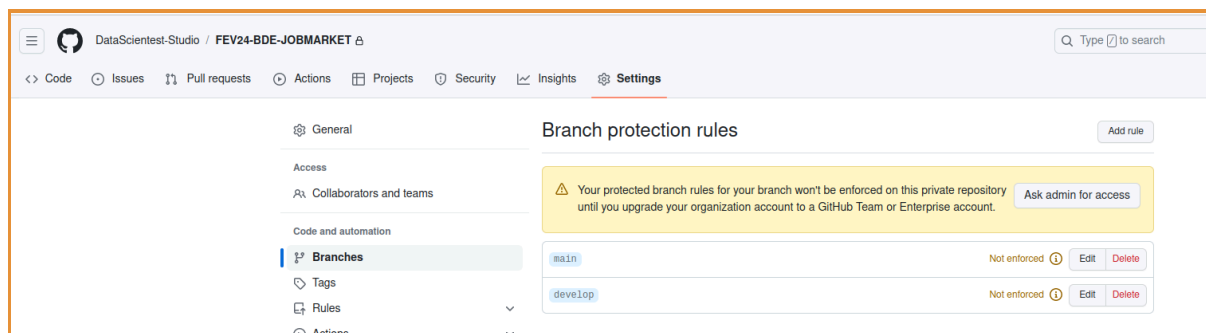
Stratégie de gestion des branches git

La stratégie de gestion des branches git utilisée sera une version simplifiée du [gitflow](#). Nous aurons donc une branche *main*, une branche *develop* ainsi que de multiples branches *feature* pour le développement de nouvelles fonctionnalités et qui dérivent obligatoirement de la branche *develop*. L'idée est de protéger au maximum la branche *main* qui est considérée comme la branche de production et de déploiement.

Pour cela, nous avons fixé *develop* comme la branche par défaut :



De plus, nous obligeons la création d'une pull-request pour tout commit-push sur les branches *main* et *develop*. Cette dernière contrainte n'est pas activable sur *github* par manque de droit (voir ci-dessous), mais nous appliquerons les principes de protection des branches :



Bonnes pratiques de code

Certaines conventions pour le développement Python seront mises en place. Nous suivrons le guide [pep8](#) ainsi que [flake8](#). Nous nous baserons également [sur ces bonnes pratiques](#) afin d'homogénéiser notre code.

Architecture du projet

Dans cette partie, nous allons nous concentrer sur la manière dont nous avons agencé les différents fichiers, répertoires et modules Python de notre projet, en nous inspirant de [ce guide](#).

Voici l'arborescence actuelle de notre projet :

```
.
├── CONTRIBUTING.md
├── pyvenv.cfg
├── README.md
├── requirements.txt
├── docs
│   ├── deliverables
│   └── reports
│       ├── compte_rendu_01.03.2024.txt
│       └── compte_rendu_05_03_2024.txt
├── src
│   └── api_providers_credentials.yml
├── FranceEmploi
│   └── FranceEmploiApiCaller.py
├── helpers
│   ├── DataDump.py
│   ├── HttpCaller.py
│   ├── Logger.py
│   └── OAuth2Helper.py
├── Muse
│   └── muse_data_extraction.py
├── francetravail_download.py
└── __main__.py
```

./requirements.txt et pyvenv.cfg

Le fichier `requirements.txt` est utilisé pour `pip`, `anaconda` et `pyvenv.cfg` pour `pyvenv`. `requirements.txt` spécifie les dépendances requises de librairies python pour l'exécution du projet et `pyvenv.cfg` va permettre de charger la configuration de l'environnement virtuel.

README.md et CONTRIBUTING.md

Ces fichiers contiennent la présentation générale du projet ainsi que les commandes pour lancer ou participer au projet.

docs

Ce dossier va contenir les documentations à la fois Python de nos modules (pour l'instant non générées), ainsi que les rapports de réunions et des livrables Datascientest.

src

Enfin, ce dossier va contenir l'ensemble de nos modules python liés au projet. Le fichier `api_providers_credentials.yml` est amené à disparaître. Voici en détail l'utilité de nos différents modules.

helpers

Ce module contient l'ensemble des fonctions utilitaires de notre projet.

Fichier	Fonctionnalité
<code>HttpCaller.py</code>	Simplifie l'utilisation du module <code>requests</code>
<code>Oauth2Helper.py</code>	Permet de récupérer un token d'accès via OAuth
<code>Logger.py</code>	logger pour retracer les appels de fonction et les différentes informations / warning liés
<code>DataDump.py</code>	transforme les résultats web en des données nettoyées pour l'enregistrement dans une BDD ou un fichier CSV

Muse et France Emploi

Ces 2 modules contiennent les fonctions pour télécharger les données. Ils seront détaillés dans la partie suivantes

Diagrammes UML

+ `.gitignore` + `credentials` + `helpers` (factorisation du code)

Les données

Voici les différentes sources de données explorées pour ce projet.

L'obtention des données est réalisée par requêtes HTTP via Python et sa bibliothèque `requests`.

Les sources retenues

France Travail

Présentation générale

France Travail est le service public dédié au marché du travail, composé de 900 agences sur le territoire, et d'un réseau de partenaires.

Les principales missions de France Travail consistent à :

- accueillir et orienter les demandeurs d'emploi,
- mettre en relation les demandeurs et les recruteurs,
- contrôler la recherche d'emploi,
- indemniser les demandeurs d'emploi,
- mettre à disposition les données relatives à l'emploi.

Dans le cadre de sa mission France Travail a notamment la charge des données suivantes :

Donnée	Description
Indemnisation	Montant des allocations, part des demandeurs indemnisables, évolution dans le temps
Statistiques	Statistiques relatives au marché du travail, nombre de demandeurs...
Offres	Offres d'emploi diffusées
Formation	Données sur les formations des demandeurs d'emploi (840 000 demandeurs ont suivi une formation en 2019)

Actuellement France Travail publie et maintient **12 API**.

API offres d'emploi

L'[API Offres d'emploi](#) restitue en temps réel les offres d'emploi actives collectés par Pôle emploi ou que Pôle emploi a reçues de ses partenaires à la condition que ceux-ci aient consenti à leur mise à disposition dans l'API. L'API s'adresse aux particuliers, entreprises & startups et collectivités territoriales et permet de développer des solutions personnalisées de recherche d'emploi pour un site ou une application.

Les ressources de l'API permettent de :

- Réaliser une recherche d'offres à partir de critères de sélection et de récupérer une liste de résultats paginée,
- Consulter le détail d'une offre (intitulé, lieu, entreprise, contrat...),
- Restituer les référentiels utilisés par l'API Offres d'emploi (Lieu, Secteurs d'activités, contrats, formations, métiers...),
- Filtrer sur plusieurs métiers, communes, départements, contrats.

Requête

L'utilisation de L'API est libre. Il suffit de créer un compte sur francetravail.io, de spécifier quels usages seront faits des données, puis créer une application et y déclarer les API France Travail qu'on souhaite utiliser. Une clé secrète rattachée à l'application est générée, et couplée à l'identifiant client du compte France Travail, le couple sert à l'authentification préalable à l'appel des API.

Cette authentification est en OAuth2 avec le grant type "*client_credentials*" donc délivre un jeton d'accès - valide durant 30 minutes chez France Travail.

Cette API a cependant une limitation : **3 appels / seconde**. Cela reste largement suffisant pour nos besoins.

Requête d'obtention du jeton d'accès OAuth2

En ligne de commande avec l'utilitaire xh pour la découverte de l'API :

```
xh post https://entreprise.pole-emploi.fr/connexion/oauth2/access_token\?realm\=%2Fpartenaire
--form grant_type=client_credentials client_id=$CLIENT_ID client_secret=$CLIENT_SECRET
scope='api_offresdemploi_v2 o2dsoffre'
```

En Python pour le projet :

```
def get_access_token_by_client_credential(access_token_url: str, scope: str, client_id: str,
client_secret: str, params: dict = {}) -> str

def authenticate(self, scope: str, params: dict = {}):
    self.access_token = OAuth2Helper.get_access_token_by_client_credential(
        access_token_url = self.access_token_url, scope = scope, client_id = self.client_id,
        client_secret = self.client_secret, params = params
    )
```


Requête d'obtention des données

Une fois le jeton récupéré, nous pouvons exécuter nos requêtes auprès de l'API concernée.

Voici un exemple de requête possible :

```
url = "https://api.pole-emploi.io/partenaire/offresdemploi/v2/offres/search"
querystring = {"range": "100-200", "departement": "59", "publieeDepuis": "3"}
headers = {
    "Authorization": "Bearer ACCESS_TOKEN",
    "Accept": "application/json"
}
response = requests.get(url, headers=headers, params=querystring)
```

Les paramètres de notre requête sont :

- range:  Il n'est pas possible de récupérer plus de 150 résultats à la fois, et maximum à 3000, il est nécessaire de spécifier quel ensemble de résultats nous voulons.
- departement: le département français où sont situées les offres d'emploi

- `publieeDepuis`: les offres publiées depuis maximum « X » jours, les valeurs acceptées sont 1, 3, 7, 14 ou 31

Résultat

Le résultat est renvoyé au format JSON. Voici une partie de la réponse envoyée de la requête précédente avec les attributs qui nous intéressent :

```
{
  "id": "170VJRB",
  "intitule": "Assistant Commercial Bilingue Anglais (F/H)",
  "description": "...",
  "dateCreation": "2024-03-09T17:27:14.000Z",
  "dateActualisation": "2024-03-09T17:27:16.000Z",
  "lieuTravail": {
    "libelle": "59 - MAUBEUGE",
    "codePostal": "59600",
    "commune": "59392"
  },
  "romeLibelle": "Gestion administrative des ventes",
  "appellationLibelle": "Assistant / Assistante commerce international",
  "entreprise": {
    "nom": "SYNERGIE"
  },
  "typeContrat": "CDI",
  "experienceLibelle": "5",
  "salaire": {
    "libelle": "Annuel de 30000,00 Euros à 35000,00 Euros sur 12 mois"
  },
  "dureeTravailLibelleConverti": "Temps plein",
  "qualificationCode": "6",
  "qualificationLibelle": "Employé qualifié",
  "secteurActivite": "78",
  "secteurActiviteLibelle": "Activités des agences de travail temporaire",
}
```

L'API nous renvoie 100 offres, que l'on peut stocker temporairement dans un fichier JSON dont les données seront ensuite traitées puis stockées en base de données.

Muse

Présentation générale

[Muse](#) est une plateforme afin de mettre en contact chercheurs d'emploi et entreprises. Il est évidemment possible de rechercher des offres d'emploi en fonction d'un type de poste, de lieu et bien d'autres filtres, mais pas seulement. La philosophie de Muse est également d'être capable d'identifier les entreprises qui seraient adaptées à nos besoins. Il est ainsi possible de connaître des informations basiques: lieu, taille, domaine des entreprises, mais également de savoir qu'elle est la culture de l'entreprise et ses valeurs,

notamment via les récits d'anciens employés. Un autre aspect de muse est de mettre à disposition des services et des coach en carrière afin d'aider les utilisateurs à développer leurs plan de carrières en fonction de leurs compétences et leurs besoins spécifiques.

API

[l'API muse](#) est une solution pour récupérer des listes d'emplois et d'entreprises. Cette Api est en accès libre. Il sera nécessaire de créer une clé API afin de ne pas être limité en terme de nombre de requêtes. Cette clé nous permettra donc de faire 3600 requêtes/heure. Le résultat des requêtes sera sous forme de liste paginé avec un maximum de 20 résultats retournés selon la requête.

Les requêtes sur l'API permettent de :

- Réaliser une recherche d'offres d'emplois basée sur les filtres obligatoires suivant: le numéro de la page, l'ordre de filtrage sur la date de publication (croissant, décroissant). Il est possible de filtrer en fonction du nom de l'entreprise, de la catégorie professionnelle, du niveau d'expérience et de la localisation.
- Récupérer les informations associés à une offre d'emploi à partir de son id
- Réaliser une recherche sur les entreprises basée sur les filtres obligatoires suivants: le numéro de la page, afficher par ordre croissant ou décroissant (**VÉRIFIER PAR QUOI : date publication ou nom de l'entreprise????**). Il est possible de filtrer en fonction du secteur d'activité, de la taille et de la localisation de l'entreprise
- Récupérer les informations associés à une entreprise à partir de son id

Requête et résultat

Récupération des offres de la première page, les données récupérés sont au format json :

```
response = requests.get("https://www.themuse.com/api/public/jobs?page=1&descending=true").json()
```

```
>>response
```

```
{
  'page': 1, #Numéro de la page récupéré
  'page_count': 15574, #Nombre total de page
  'items_per_page': 20, #Nombre de résultat pour cette page
  'took': 77,
  'timed_out': False,
  'total': 311479, #Nombre total d'offre pour les filtres demandés
  'results': [ #Liste des offres sur cette page
    {...
  },
  {...
  },
  ...
],
  'aggregations': {}
}
```

Exemple de dictionnaire json pour une offre d'emploi :

```
>> response["results"][0]
{
  "categories": [],
  "company": {
    "id": 15000029,
    "name": "TikTok",
    "short_name": "tiktok"
  },
  "contents": '<p><b>Responsibilities</b><br><br> TikTok is the leading '
    'destination for short-form mobile video. Our mission is to '
    'inspire creativity and bring joy. TikTok has global offices '
    '.....</p>',
  "id": 13113367,
  "levels": [
    {
      "name": "Mid Level",
      "short_name": "mid"
    }
  ],
  "locations": [
    {
      "name": "Singapore"
    }
  ],
  "model_type": "jobs",
  "name": "Software Engineer - Global CRM Engineering",
  "publication_date": "2023-11-08T01: 33: 08Z",
  "refs": {
    "landing_page": "https:
//www.themuse.com/jobs/tiktok/software-engineer-global-crm-engineering-e77e2d"
  },
  "short_name": "software-engineer-global-crm-engineering-e77e2d",
  "tags": [],
  "type": "external"
}
```

Exemple de données que l'on obtient post-processing (celles-ci sont amenées à évoluer avec le temps) :

```
{
  "category": "Management",
  "description": '<p><strong>Siemens Digital Industries Software</strong> is a '
    'leading provider of solutions for the design, simulation, '
    'and manufacture of products across many different .....'',
  "fonctionnal_name": "senior-solution-architect-hf-577035",
}
```



```
"level":"senior",
"link":"https:
//www.themuse.com/jobs/siemensdigitalindustriessoftware/senior-solution-architect-hf-577035",
"name":"Senior Solution Architect h/f",
"place":[
  {
    "name":"Paris, France"
  }
],
"publication_date":"2024-02-28T10: 55: 31Z",
"qualification_level":"None",
"salary":"None",
"technical_id":12550114,
"type":"external"
}
```

Les sources non-retenues

Plusieurs autres sources d'offres d'emploi nous ont intéressé pour ce projet, cependant elles ne disposent pas d'API d'obtention d'offres.

C'est pourquoi nous avons tenté d'obtenir ces offres par Scraping à l'aide de Python et sa bibliothèque BeautifulSoup4.

Malgré la présence des entêtes HTTP, d'un User Agent et des éventuels cookies utiles à la simulation d'une navigation par navigateur Web, ces sources ne sont pas interrogeables par des techniques de Scraping simples.

Les sociétés de ces sources nous semblent - et sans réelle surprise - bien au fait des techniques de Scraping et ont mis en place diverses contre-mesure efficaces pour les contrer :

Indeed :

Chez eux, toute requête HTTP produite par scraping à l'aide de BeautifulSoup4 se solde par une réponse HTTP accompagnée du code 404 "*page non-trouvée*", nos machines virtuelles hébergées sur Amazon semble faire partie d'une black-list d'adresses IP.

Nous avons découvert que le marketplace d'API *RapidAPI* (anciennement nommé *Paw*) - grâce à ses techniques bien plus avancées de scraping (rotating proxy notamment) - propose une API retournant les offres Indeed. Cette API, même si relativement lente, est fonctionnelle mais nécessite pour nos besoins de souscrire à une offre payante de RapidAPI pour nous permettre un nombre suffisant d'appels ; nous avons donc pour cette raison été contraints d'abandonner cette source.

Welcome To The Jungle :

Welcome To The Jungle, eux, usent d'une autre contre-mesure qu'on retrouve chez LeBonCoin : l'anonymisation et randomisation des classes des balises de la page HTML.

L'anonymisation des classes de ces balises rend le code du scraping peu intuitif et sa maintenabilité pénible. Couplé à la randomisation, le scraping est de plus rendu non-pérenne et nécessite d'être alors régulièrement mis-à-jour pour demeurer fonctionnel.

Yaniv Benichou de DataScientest a d'ailleurs un projet GitHub sur le scraping de cette source de données et dont le code (plus fonctionnel) dénote de la difficile maintenabilité : <https://github.com/benech17/webscraping-welcome-to-the-jungle/>

En exemple ci-dessous le bloc HTML anonymisé et randomisé d'une offre d'emploi présente dans une page d'offres retournée comme résultats d'une recherche par mot-clés :

```
<div class="sc-bXCLTC iiwBSR sc-ezreuY kFVkJDa">
  <div class="sc-bXCLTC jYvPbE">
    <div class="sc-6i2fyx-2 hqLPMA">{...}</div>
    <span class="sc-ERObt ldmfCZ sc-6i2fyx-3 eijbZE wui-text">Computacenter</span>
  </div>
  <div class="sc-bXCLTC bAjxyY">
    <a href="/fr/companies/computacenter/jobs/ingenieur-workplace-microsoft-print-h-{...}">
      <h4 class="sc-ERObt neOJH sc-6i2fyx-1 eKEFKi wui-text">
        <div role="mark" class="sc-bXCLTC hlqow9-0 helNZg">Ingénieur Workplace Microsoft /
      </div>
    </a>
  </div>
</div>
```

```

    </h4>
  </a>
  <div class="sc-bXCLTC heipUY"><i class="sc-fUBkdm hFJXgf wui-icon-font"
name="location"></i>
    <p class="sc-ERObt hozLPp wui-text"><span class="sc-68sumg-2 hCLwRn">
      <span class="sc-68sumg-0 gvkFZv">La Chapelle-du-Mont-de-France</span>
    </span></p>
  </div>
  <div class="sc-bXCLTC jNbJEG">
    <div variant="default" w="fit-content" class="sc-dQEtJz cJTvEr"><i class="sc-fUBkdm
bizICT wui-icon-font" name="contract"></i>
      <span>CDI</span>
    </div>
    <div variant="default" w="fit-content" class="sc-dQEtJz cJTvEr"><i class="sc-fUBkdm
cJYLET wui-icon-font" name="remote"></i>
      <span>Télétravail fréquent</span>
    </div>
  </div>
</div>
</div>
</div>

```

Monster :

Monster est similaire à Welcome To the Jungle à la différence que les balises sont partiellement anonymisées donc qu'il semble possible de s'y retrouver mais que le contenu de la page HTML retournée s'avère dépourvu du texte ! C'est-à-dire que seule la structure de la page est retournée ; donc aucune donnée relative aux offres n'est accessible.

En exemple ci-dessous le bloc HTML d'une offre d'emploi présente dans une page d'offres retournée comme résultats d'une recherche par mot-clés :

```

<li class="sc-blKGMR etPslv">
  <div class="splitviewstyle__SkeletonJobCardWrap-sc-zpzpmg-5 dVweKQ">
    <div class="sc-dAEZTx iZckGP">
      <div class="sc-hqpNSm dTGORb">
        <div class="sc-etVdmn kGzGlx logo"></div>
        <div class="sc-fzQBhs IvpXN">
          <div class="sc-cDvQBt kRwHob">
            <div class="sc-etVdmn kGzGlx title">
            </div>
            <div class="sc-etVdmn kGzGlx company">
            </div>
            <div class="sc-etVdmn kGzGlx details">
            </div>
          </div>
          <div class="sc-hpGnlu eVeKyx">
            <div class="sc-etVdmn kGzGlx tag">
            </div>
          </div>
        </div>
      </div>
    </li>
  </div>

```

```

        <div class="sc-etVdmn kGzGlX tag">
        </div>
    </div>
    <div class="sc-etVdmn kGzGlX description">
    </div>
</div>
<div class="sc-etVdmn kGzGlX contextDetails">
</div>
<div class="sc-etVdmn kGzGlX button"></div>
</div>
</div>
</div>
</li>

```

Perspective et amélioration pour le livrable 2

Choix et format des données

Voici les données que nous allons stocker dans notre future base de données :

- ID technique (auto-généré)
- Source
 - Nom
 - ID (ID de l'offre sur *France Travail*, sur *Muse*)
- Libellé
- Description (=corps du job, parsing à effectuer)
- Lieu
- Date de publication
- Niveau de qualification : liste finie a identifier (Technicien/ cadre/docteur)
- Catégorie professionnelle : liste finie a identifier (Data scientist, RH ...)
- Niveau d'expérience liste finie a identifier
- Type de contrat : (CDD, CDI, Freelance...)
- Remote si précisé
- Salaire si précisé

Nous partons sur une base de données NoSQL : **MongoDB** car beaucoup de chaînes de caractères vont être stockées ainsi que des données non-structurées.

TODO : lister pour les 2 API les valeurs possibles pour certains attributs et se mettre d'accord.

Mise à jour des données

Modification d'une offre : comment actualiser notre BDD ? A réfléchir

Création de la BDD

A réfléchir

Amélioration du code

Refactorisation

Nous devons refactoriser nos 2 classes Muse & France Travail afin de créer une classe parente qui permettra d'avoir des fonctions génériques peu importe les données présentes en entrée, ainsi que pour faciliter l'insertion dans le BDD.

Amélioration de code

France Travail

France Travail a une limite de 3000 résultats pour un type de requête, même si plus de résultats sont disponibles. Il faudra affiner la requête pour arriver à un nombre de résultats inférieur à 3000 afin d'être sûr de récupérer tous les résultats possibles, par exemple en spécifiant la dernière date d'actualisation. Grâce à l'attribut "publieeDepuis", on pourra également ajouter facilement de nouvelles données, si on fait une requête à heure fixe chaque jour par exemple.

Muse

Afin d'homogénéiser nos données et d'obtenir des données de qualité pour notre future base de données, il reste à effectuer plusieurs tâches de post-processing sur les données de *Muse* :

- Filtrer exclusivement les offres géographiquement situées en France.
- Inclure ou non les offres ouvertes au télétravail.
- Intégrer en base de données les offres de manière incrémentale, de façon à ne pas intégrer celles déjà intégrées, car il n'est pas possible de filtrer les offres en fonction de leur date de publication.

Tests unitaires

- Tests unitaires à réaliser sur les helpers + des API
- Tests fonctionnels ? sur le téléchargement des données notamment

