Documentation Technique MR-biosoft Annotation Platform

Théo Roncalli Gustavo Magaña López

Contents

Introduction

Le projet repose sur deux installations principalement: PostgreSQL et Python. Le site a été conçu en utilisant *django*, un *framework* pour le développement web écrit en langage Python. Toute la partie *front-end* du projet a été conçue dès zéro, en définissant les styles CSS sans s'appuyer sur des *frameworks* tels que bootstrap. Le système de gestion de bases de données relationnelles choisi est *PostgreSQL*.

Toutes les explications et instructions trouvées dans ce document partent de l'hypothèse que `Python3.8+` et `psql (PostgreSQL) 12.7+` sont installés dans votre système. Si vous n'avez pas PostgreSQL, je vous recommande de suivre cette guide. Python est installé par défaut en la plupart de distributions GNU/Linux. Tout le développement de l'application a été fait en Ubuntu et Pop!_OS, toutes les deux basées sur Debian. Aucune garantie n'est donnée pour d'autres distributions.

Gestion des Dépendances

- Poetry for Python dependencies management.
- PostgreSQL for database management.

La gestion des dépendances du projet a été faite grâce à *Poetry*, un gestionnaire de paquets pour Python qui offre l'avantage d'être 100% reproductible, contrairement à d'autres plateformes de gestion d'environnements très populaires telles que *conda*. Le manque de reproductibilité de conda vient du fait que son *channel* principal n'est pas un miroir de l'index de paquets python (PyPI). Pour installer différents paquets, plusieurs *channels* doivent être ajoutés. Parfois on ne trouve pas le paquet et on finit par l'installer directement via `pip`. Or, les paquets installés via `pip` ne seront pas suivis par conda. Si l'on essaie de partager un fichier `env.yml` produit par conda, on n'a aucune garantie que tous les paquets seront inclus dans le fichier.

Par contre, poetry a un *dependency-solver* très robuste qui permet de trouver des possibles conflits de dépendances lors de l'ajout via la commande `poetry add <dependency>`. En étant stricte et informant l'utilisateur des conflits lors

de l'ajout, en suggérant comment les résoudre (si possible), poetry garantit la Par défaut, poetry installe les paquets depuis le PyPI, ce qui veut dire qu'il peut trouver tout paquet de l'index officiel de python.

Pour installer poetry et configurer les variables d'environnement nécessaires afin de lancer le serveur, il est recommandé de suivre le guide README du projet.

Un script pour faciliter la configuration se trouve dans le livrable (pas dans le repo car l'inclure dans un répertoire publique représenterait une grande vulnerabilité).

Une fois que script `bootstrap_installation.bash` exécuté, une nouvelle session `bash` doit être démarrée afin d'avoir accès à la commande poetry.

À l'intérieur du répertoire où se trouvent `pyproject.toml` et `poetry.lock`, exécuter `poetry install` pour installer toutes les dépendances du projet. Puis, exécuter `poetry shell` pour activer l'environnement virtuel et lancer l'application.

Organisation du code serveur

```
poetry.lock
pyproject.toml
prokaryote
 annotation # Application principale
 bootstrap_installation.bash # script d'aide à l'installation
 createdb.bash # idem mais pour la création des tables de la base de données
 dbsetup # Application django pour gestion de la base de données
 debug.py # Script pour analyser la sortie des error-logs
 gene_importation_error_log.jsonl # logs d'erreur liés à problèmes de parsing
 home # Application qui gère la page d'accueil
 manage.py # Couteau Suisse de django.
 prokaryote # app. de base qui fait le lien entre toutes les autres
 runall.bash # Script qui permet d'effacer et recréer la base et importer
             # toutes les données
 static # répertoire contenant le css de tout le site
 upload # Application pour importer des nouvelles séquences
```

Introduction à django

django permet de construire des sites web modulaires, composés de plusieurs applications indépendantes. Toute commande liée à un projet django passe par le script `manage.py`. Celui-ci permet de lancer le serveur, se connecter à la base de données de l'application, lancer une session Python interactive qui permet de tester le code développé.

Guide de commandes de base

- Créer un nouveau site : `django-admin startproject <project-name>`

À l'intérieur du dossier du projet, les commandes suivantes sont disponibles :

- Créer une nouvelle application : `./manage.py startapp <app-name>`
- Se connecter à la base de données : `./manage.py dbshell`
- Pour lancer une session python interactive : `./manage.py shell`

Commandes développées par l'équipe

- Exécuter un script sql à l'intérieur de la base de données du site : `./manage.py dbexec script.sql`
- Importer des données (en format FASTA) :
  - Des génomes `./manage.py importgenome fichier_genome.fa`
  - Des gènes `./manage.py importgenes fichier_genes.fa`
  - Des protéines `./manage.py importproteins fichier_proteines.fa`

Le fichier `runall.bash` permet de tout exécuter: créer les tables de la base, importer tous les génomes, tous les gènes et toutes les protéines (attention, un dossier appelé `Data` doit contenir tous les fichier et doit se trouver à la racine du projet).