UNIVERSITÉ
DE MONTPELLIER

Projet HMSN204

Master Sciences et Numérique pour la Santé
parcours Bioinformatique, Connaissances, Données

[Étudiant] Xavier GRAND
[Étudiante] Marine SALSON
[Étudiante] Daniela Jessica VARGAS ANDRADE
[Enseignant] Isabelle MOUGENOT

Mars 2020 - Mai 2020

Table des Matières

Liste des Figures

1 Introduction

Ce rapport présente le projet que nous avons dû mener à bien dans le cadre de l'UE HMSN204 Info Biologique et Outils Bioinformatiques, du deuxième semestre du Master Sciences et Numérique pour la Santé, parcours Bioinformatique, Connaissances, Données.

Ce projet a consisté à modéliser et à rassembler au sein d'une **base de données** de notre choix des informations obtenues par des élèves du Master Biologie Fonctionnelle des Plantes. Ces données concernent la description de différents traits morphologiques, chez plusieurs génotypes de plantules d'*Arabidopsis thaliana*. La construction d'une base de données permet de conserver et d'avoir accès aux diverses information plus facilement, et, ces dernières peuvent être manipulées à la convenance de chaque utilisateur à travers différentes requêtes. En effet, une base de données peut être partagée avec la communauté scientifique, et il est important qu'elle puisse être suffisamment générique. Nous avons également ajouté à ce projet des **traitements métiers bioinformatiques**, portant notamment sur l'analyse de séquences biologiques et l'utilisation de l'outil Blast (Basic Local Alignment Searching Tool), ainsi que sur la mise en place d'un algorithme permettant l'alignement par paire global de deux séquences.

La première partie de ce projet a consisté à modéliser la structure au niveau de laquelle nous allons pouvoir représenter et stocker les données à l'étude. Pour cette première étape d'analyse et de conceptualisation, nous nous sommes appuyés sur le **langage UML** (Unified Modeling Langage). Ce langage nous a permis de visualiser l'architecture de notre base, ainsi que les différentes utilisations que nous pourrons faire de cette dernière et des traitements bioinformatiques développés. Nous présentons les différents diagrammes UML réalisés, statiques et dynamiques, au sein de la partie **Modélisation conceptuelle** de ce rapport.

Nous avons ensuite dû choisir un **système de gestion de base de données**, permettant d'organiser les informations obtenues par les élèves du Master Biologie des Plantes, et rassemblées au sein d'un fichier CSV. Nous expliquons notre choix au sein d'une deuxième partie. Ce choix, qui était initialement d'exploiter les connaissances acquises au cours de cette UE afin de réaliser une base de données graphes avec le système de gestion Neo4J, s'est finalement porté vers une **base de données relationnelle**. Nous présentons les résultats de l'implémentation de cette dernière au sein de la dernière partie du rapport Résultats.

Les traitements bioinformatiques que nous avons développés grâce à l'utilisation du langage de programmation Python, et notamment de la librairie Biopython, sont expliqués et présentés au sein de la troisième partie de ce rapport. Nous présentons notamment le programme qui permet l'alignement global de deux séquences, et un exemple de résultat qu'il est sensé fournir.

2 Modélisation conceptuelle

La première partie du projet a consisté à schématiser les structures qui permettront de stocker nos données, sous forme de différents types de diagrammes, à l'aide du langage UML. Ce premier travail, qui s'inscrit au cours des étapes d'analyse et de conception lors d'une conduite de projet habituelle, nous a permis d'avoir un premier aperçu de la manière avec laquelle les données fournies allaient pouvoir être structurées.

2.1 Diagramme de classes et d'instances

Afin de réaliser le diagramme de classes devant présenter les différentes classes de notre système, nous utilisons le logiciel Modelio (Figure 1).

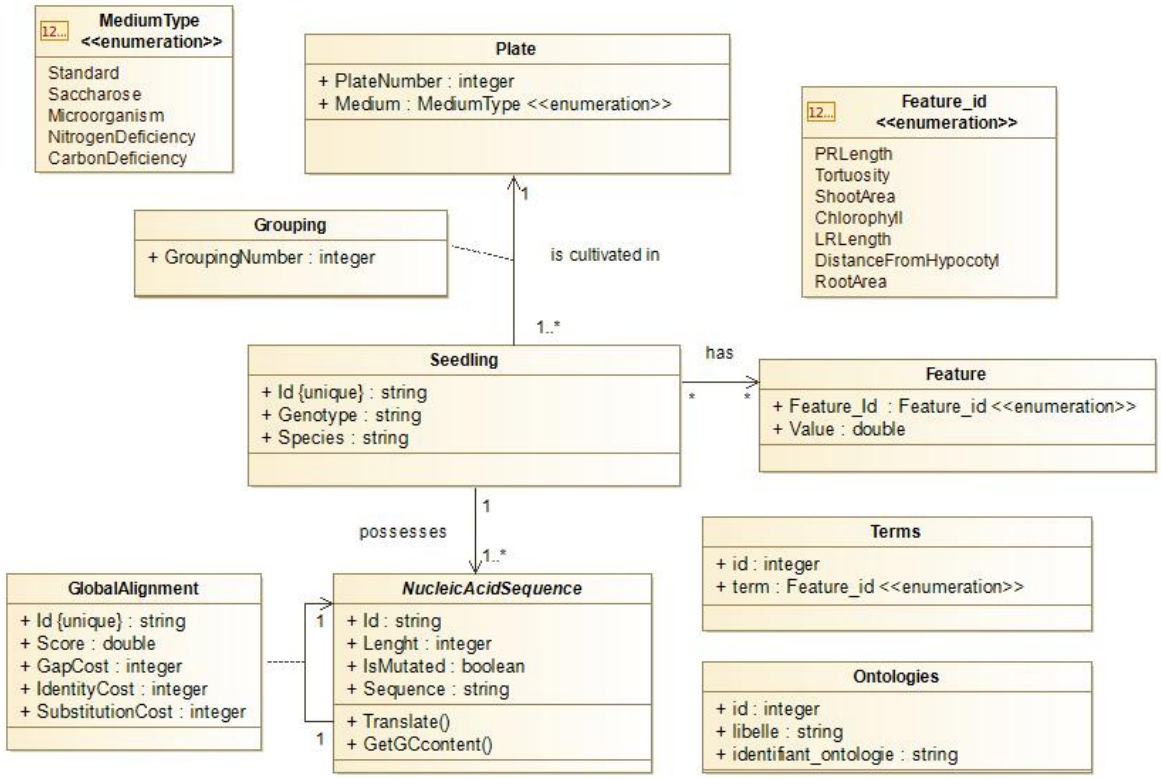

Figure 1: Diagramme de classes permettant de présenter la structuration de notre système de stockage des données.

La classe principale de notre diagramme est la classe **Seedling**, dont les instances correspondent aux différentes plantules qui ont été phénotypées. Les objets de cette classe possèdent ainsi l'attribut **Id**, qui correspond à l'identifiant de chaque individu, et qui est unique. Cet attribut est donc la **clé primaire** de cette classe : il permet de désigner de

façon unique chaque objet, c'est à dire chaque plantule. Celui-ci correspond par exemples aux valeurs "A1_1", ou "B2_9" du jeu de données fournies. Cette classe Seedling comporte également les attributs **Genotype** et **Species**. Si l'attribut Species correspond uniquement à l'espèce *Arabidopsis thaliana* au sein des données qui nous ont été données, il permet cependant à notre modèle de pouvoir intégrer d'autres espèces, et d'avoir ainsi une base de données plus générique. De même, l'attribut Genotype correspond aux valeurs "Col_0", "nrt", "sex1" et "pgm" des données se trouvant au sein du fichier CSV fourni, mais pourrait permettre d'identifier d'autres types de mutants.
La classe **Feature** correspond aux différentes caractéristiques rattachées à une plantule. Ainsi, l'assocation binaire "has" est établie de la classe Seedling vers la classe Feature. En outre, les objets de cette classe possèdent les attributs **Feature_id** et **Value**. Le premier correspond à un trait phénotypique rattaché à une plantule, et le deuxième attribut Value à sa valeur associée. Ainsi l'objet Seedling d'identifiant "A1_1" pourrait être rattaché à l'objet Feature possédant le Feature_id "PRLength" et la Value 3.176. Dans notre cas, nous décidons d'établir une **énumération** des différents traits morphologiques qui ont été étudiés par les étudiants du Master Biologie Fonctionnelle des Plantes, tels que la longueur de la racine primaire ("PRLength"), ou la surface des feuilles ("ShootArea"), par exemples. Aussi, cette énumération pourrait être complétée, dans le cas où d'autres caractères morphologiques, appartenant ou non à *Arabidopsis thaliana*, seraient étudiés et donc ajoutés à la base de données.
Nous réalisons également la classe **Plate**, dont les instances correspondent aux différentes boîtes au niveau desquelles les plantules ont été cultivées. Ainsi une association binaire "is cultivated in" existe de la classe Seedling vers la classe Plate. Les objets de cette classe comportent en outre les attributs **PlateNumber**, pour le numéro de la boîte en question, et **Medium**, qui correspond au milieu de culture contenu dans cette dernière. A nouveau, nous décidons pour les différentes valeurs pouvant être prises par l'attribut Medium d'établir une **énumération**, adaptée aux protocoles expérimentaux utilisés par les élèves du Master Biologie Fonctionnelle des Plantes. Ainsi, les objets Plate peuvent par exemple avoir pour valeur "Standard" ou "Saccharose", en ce qui concerne l'attribut Medium. Cette énumération pourrait également être complétée, dans le cas où un nouveau schéma expérimental serait étudié et ajouté à la base de données.
Nous modélisons également la **classe association Grouping**, entre les classes Plate et Seedling, devant représenter le groupe auquel les plantules appartiennent au sein de la boîte dans laquelle elles sont cultivées. Ainsi, cette classe association Grouping ne possède que l'attribut **GroupingNumber** qui lui est spécifique.
Nous décidons également d'ajouter la classe **NucleicAcidSequence**, dont les instances correspondent aux séquences nucléiques, d'ADN ou d'ARN, que peuvent posséder les plantules. Ces séquences possèdent un attribut **Id**, qui permet de désigner le nom du gène en question, ainsi que les attributs **Sequence** qui correspond à la séquences de nucléotides en elle même, et Length, pour la longueur de la séquence. L'attribut **IsMutated**, qui est de type booléen, permet de préciser si la séquence du gène en question est mutée ou non. Ainsi, les objets de cette classe peuvent correspondre au gène muté *pgm* par exemple, ainsi qu'à sa version non mutée. Il serait de ce fait possible de générer des alignements de ces séquences, en manipulant les attributs Sequence (string). Nous présentons au niveau de la partie 4, sur les traitements Biopython implémentés, un script qui permet l'alignement global de deux séquences, en

utilisant le principe de la programmation dynamique. Le script en question prend notamment pour paramètres, deux séquences de type String. Il pourrait dans une version du projet plus avancée, prendre des objets **NucleicAcidSequence** comme paramètres et manipuler les attributs Sequence.
Par ailleurs, nous avons également réalisé les classes **Terms** et **Ontologies**, qui permettent d'associer à chaque terme utilisé pour désigner un caractère morphologique, un identifiant spécifique et générique. Nous avons ainsi modélisé les référentiels de valeurs afin que nos données soient adaptées aux conventions d'annotations utilisées actuellement au sein d'autres bases de données. Il s'agit ainsi d'associer à tous les termes que nous avons utilisés dans le cadre de notre modélisation, des termes appartenant à des **ontologies** de références (**Plant Ontology** (PO) et **the Phenotype And Trait Ontology** (PATO), principalement). Ceci a pour objectif de faciliter la compréhension des termes utilisés par quelqu'un d'extérieur en se référant aux ontologies, mais surtout cela permet de pouvoir potentiellement regrouper plusieurs bases entres elles. En effet si les termes sont hétérogènes il est très complexe de faire le lien entre des bases distinctes, mais dans le cas où les termes utilisés sont associés de part et d'autre à une même ontologie le lien devient beaucoup plus simple. Le lien permettant d'associer les termes utilisés avec les ontologies se trouve au niveau de la Figure 2.

Ontologies		
id	libelle	identifiant_ontologie
1	root	PO_0009005
2	primary root	PO_0020127
3	lateral root	PO_0020121
4	shoot	PO_0004545
5	leaf	PO_0025034
6	length	PATO_0000122
7	area	PATO_0001323
8	distance	PATO_0000040
9	hypocotyl	PO_0020100
10	Chlorophyll	OMIT_0004055

Terms	
id	term
1	PR length
2	RootArea
3	LR length
4	distanceFromHypocotyl std (cm)
5	ShootArea
6	Chlorophyll (a.u)

Liens	
idTerm	idOntologie
1	2
1	6
2	1
2	7
3	3
3	6
4	8
4	9
5	4
5	7
6	10

Figure 2: Tables permettant de mettre en lien les identifiants des ontologies PATO (the Phenotype And Trait Ontology) et PO (Plant Ontology), avec les termes utilisés au sein des données à l'étude pour désigner les caractères morphologiques des plantules.

Enfin, nous avons également élaboré un diagramme d'objets, afin d'illustrer un exemple d'instanciation des classes que nous avons choisies pour structurer les données à l'étude (Figure 3).

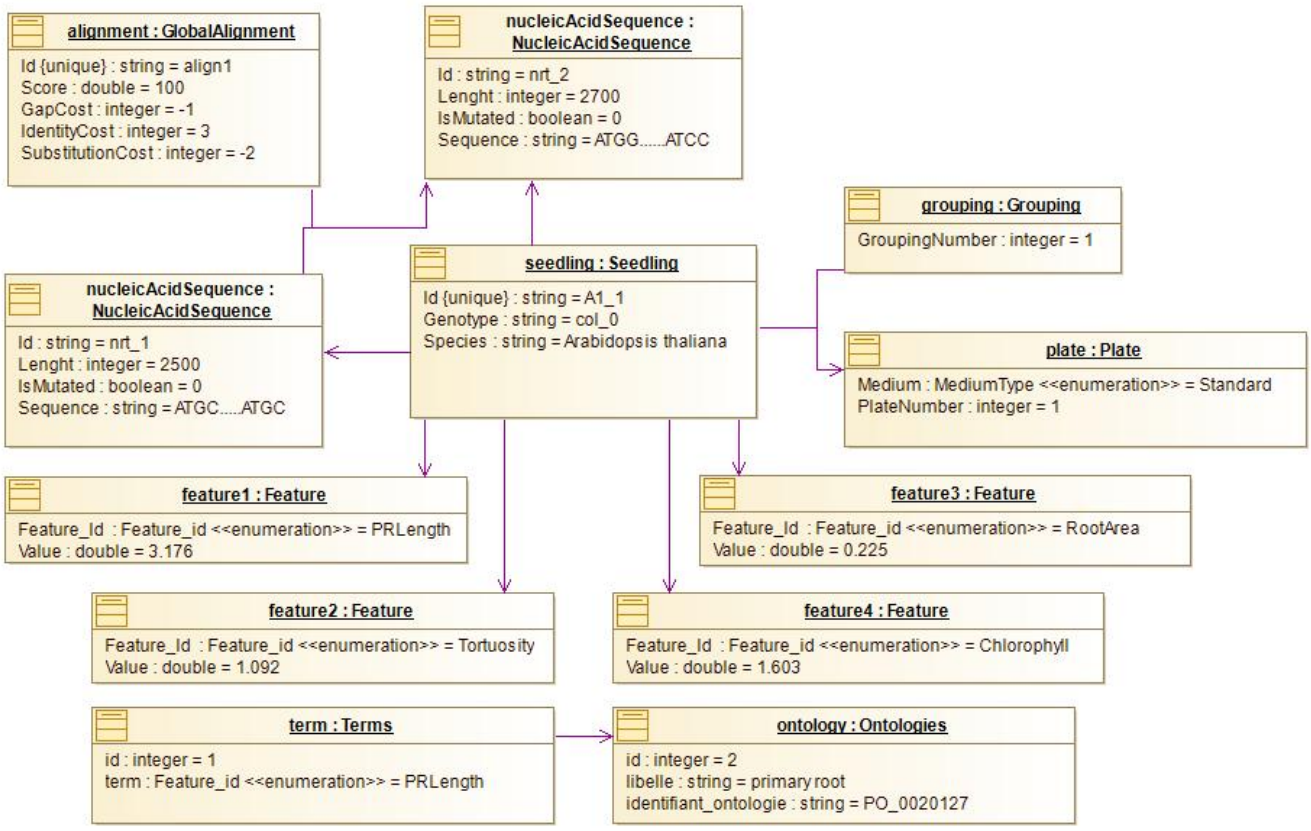

Figure 3: Diagrammes d'instances permettant de présenter des objets des classes permettant de structurer les données à l'étude.

Nous décidons de représenter les informations concernant la première plantule du fichier CSV qui nous est fourni, d'identifiant "A1_1". Pour celle-ci, nous représentons tout d'abord une première instance de la classe **Seedling**, avec pour valeur de clé primaire "A1_1", et pour valeurs des attributs Genotype et Species,"Col_0" et *Arabidopsis thaliana*. Nous modélisons une instance pour chacune des classes Grouping et **Plate**, afin de désigner au niveau de quelle boîte la plantule a été cultivée, et au niveau de quel groupe elle se trouve au sein de cette dernière. Nous décidons également de représenter quatre instances de la classe **Feature**, qui nous permettent de représenter quatre des sept traits morphologiques à l'étude associés aux plantules : la tortuosité ("Tortuosity"), la longueur de la racine primaire ("PRLength"), la surface des racines ("RootArea"), et la teneur en chlorophylle ("Chlorophyll"), et, de

leur attribuer les valeurs de la plantule A1_1. Nous schématisons également un exemple de deux objets **NucleicAcidSequence**, représentant deux gènes de la famille nrt, présents chez *Arabidopsis thaliana* et appartenant à la plantule "A1_1". Nous modélisons pour finir la création d'un objet **GlobalAlignment** possédant les différents caractéristiques d'un alignement réalisé entre ces deux séquences.

2.2 Cas d'usage

Enfin, nous réalisons un dernier diagramme statique : un diagramme de cas d'utilisation, devant permettre d'illustrer les différentes utilisations possibles de notre système de base de données et des outils implémentés avec Python (Figure 4).

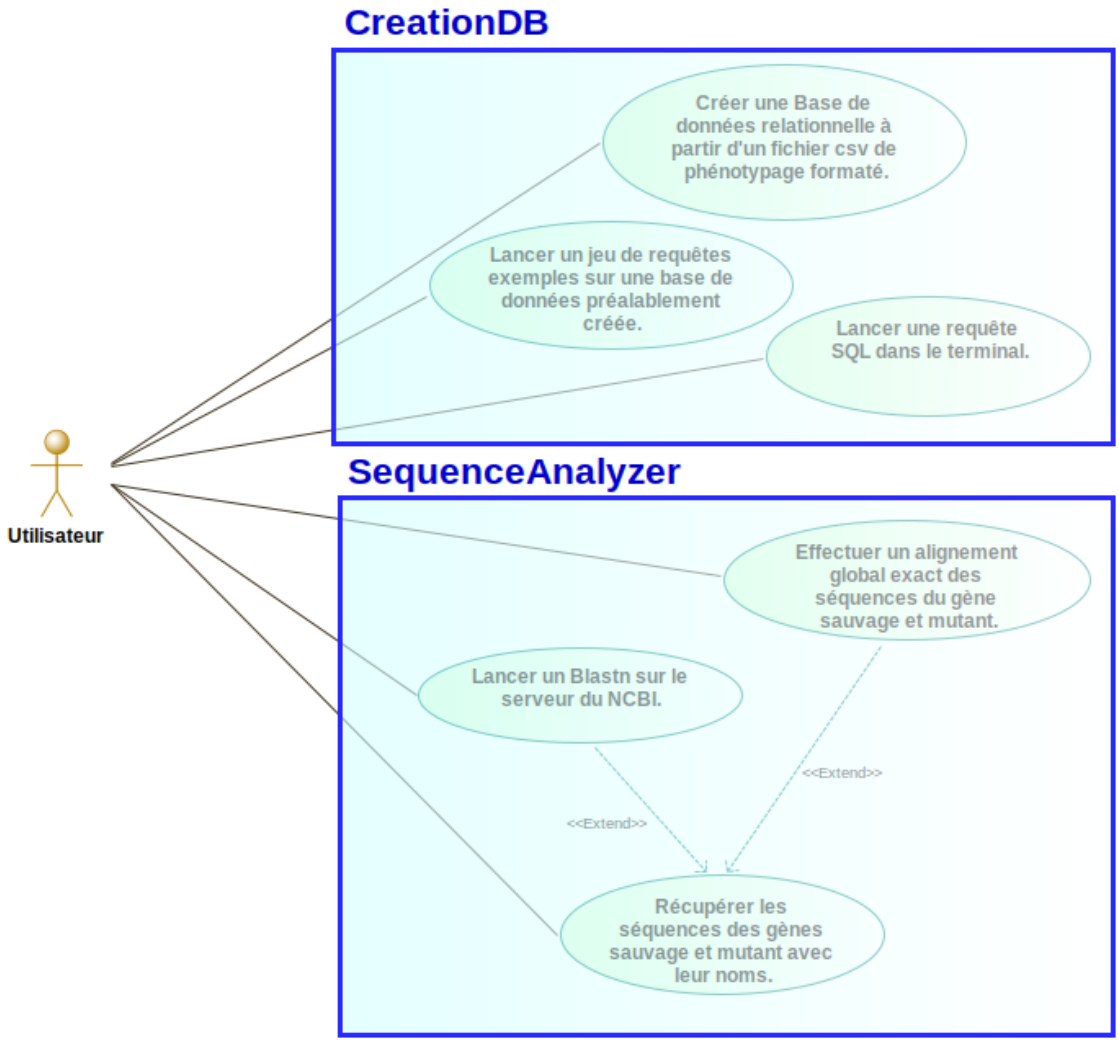

Figure 4: Diagrammes de cas d'utilisation des applications de création de la base de données et de l'analyse des séquences des gènes sauvage et mutant.

A ce stade du projet, nous n'avons pas lié les deux parties qui sont la création/interrogation de la base de données d'une part, et l'analyse des séquences des gènes d'autre part tel que présenté dans le diagramme de classe précédent. Cela ferait l'objet de quelques développements supplémentaires. Ainsi, nous proposons que l'utilisateur, par le biais de deux API puisse réaliser différentes actions :
L'utilisateur, via l'API **MainCreationDB**, peut effectuer :

- La création d'une base de données relationnelle à partir d'un fichier CSV formaté.
- L'interrogation d'une base de données préalablement créée avec un jeu de requêtes d'exemple.
- L'interrogation d'une base de données préalablement créée par une requête en langage SQL en la tapant dans le terminal. L'API invite cette requête en fonction des paramètres choisis.
- Et enfin la classe Blast qui réalisera un Blast sur la séquence non mutée.

Par ailleurs, via l'API **MainSequenceAnalyzer**, l'utilisateur peut :

- Récupérer les séquences des gènes sauvage et mutant via leurs noms sur le serveur du NCBI via le portail Entrez.
- Aligner les séquences des gènes sauvage et mutant précédemment récupérés.
- Lancer un Blastn sur le serveur du NCBI avec la séquence du gène mutant.

Les actions de construction de la base de données sont disponibles pour l'utilisateur. C'est un choix qui est rendu possible par l'empaquetage complet dans une API codée en python. En effet, la création de la base est réalisée de manière automatique à condition que le fichier de phénotypage en entrée, un fichier CSV, soit formaté de la même façon que celui fourni pour le projet.

2.3 Diagrammes dynamiques : diagramme de séquences

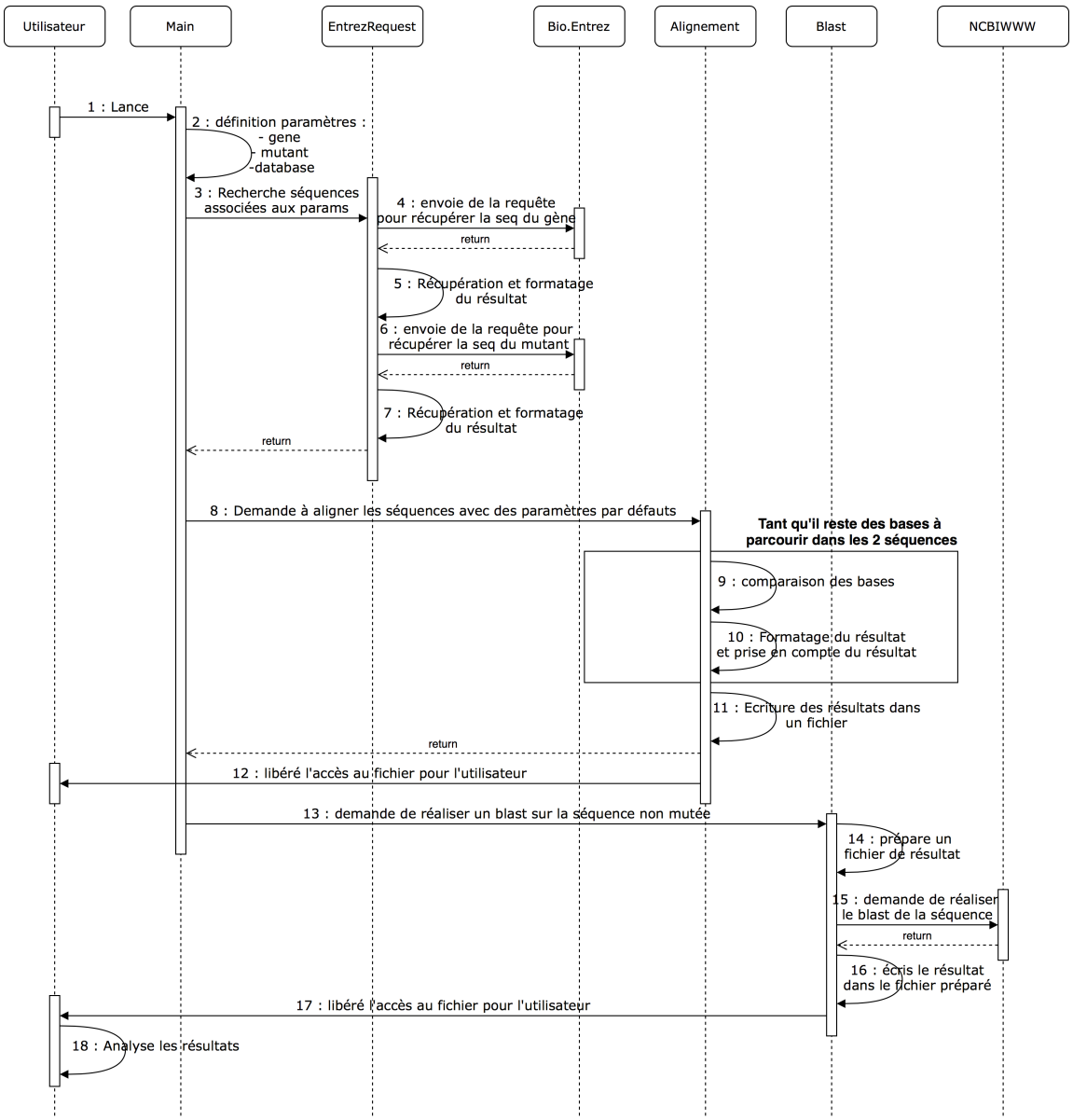

Figure 5: Diagrammes de séquences sur l'alignement de séquences.

Sur ce premier diagramme de séquences (Figure 5), nous constatons plusieurs classes afin de permettre de réaliser un alignement complet, comme nous pouvons le voir sur le diagramme de séquences nous avons :

- Le Main qui va "gérer" les différentes étapes, c'est cette classe qui va faire appel aux autres et passer les différents paramètres. Il s'agit de la classe appelée par l'utilisateur.
- La classe EntrezRequest qui va nous permettre de récupérer les séquences, qui nous permettront de réaliser les étapes suivantes.

- La classe Alignement, qui comme sont nom l'indique va permettre de réaliser un alignement entre les deux séquences précédemment récupérer.
- Et enfin la classe Blast qui réalisera un Blast sur la séquence non mutée.

Ces classes font potentiellement appel à des librairies déjà existante pour réaliser leurs tâches, comme on le voit sur le diagramme avec la classe Bio.Entrez par exemple.

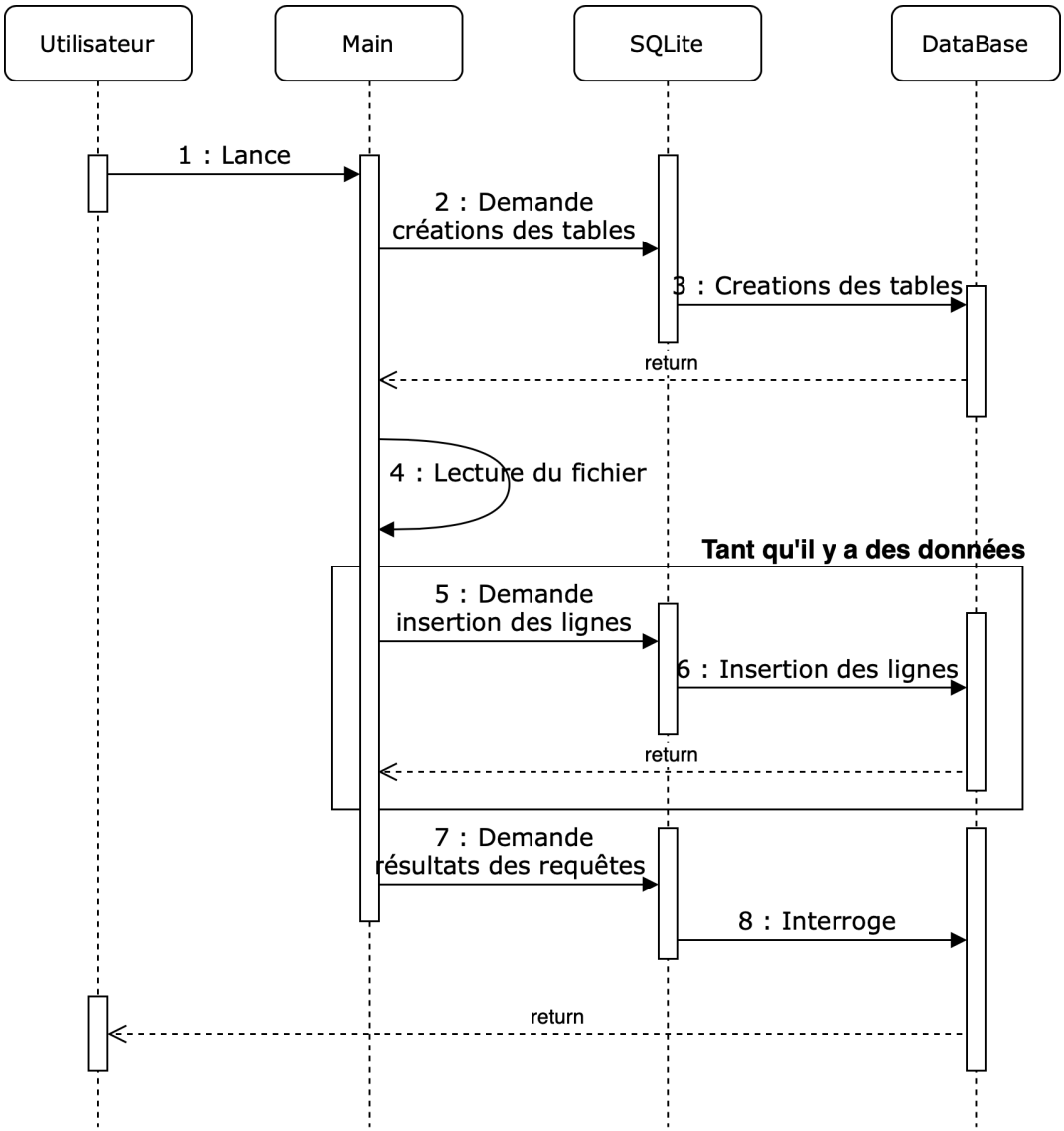

Figure 6: Diagrammes de séquences sur la création et requêtage de la base de données.

Pour ce deuxième diagramme de séquences (Figure 6), nous montrent les différentes étapes pour la création et requêtage de la base de données. Le Main réalisera la demande de création de tables et d'insertion des lignes à travers SQLite qui effectuera les différentes requêtes dans la base de données. Pour les requêtes il existe deux possibilités : soit avec le jeu de requêtes que nous avons créé et qui se fait automatiquement avec le Main, comme illustre dans le diagramme, soit de faire ses propres requêtes sur le terminal.

3 Choix du Système de Gestion de Bases de Données

L'un des choix possible afin de mettre en place une base de données permettant de stocker les informations fournies en exemple et de pouvoir effectuer rapidement des requêtes dessus, est l'utilisation du **système de gestion de base de données embarqué SQLite**. **SQLite** est une librairie qui permet de gérer des bases de données **relationnelles**, et nous permettra aisément d'organiser nos données dans des tables, liées les unes aux autres. Il sera également plus aisé de lier dans le même script Python la récupération des données contenues au sein du fichier CSV fourni, avec l'élaboration et l'enrichissement de la base de données. De plus, cela nous permet de réinvestir les notions que nous avons pu voir au premier semestre de ce Master.

Une autre possibilité, que nous avons commencé à essayer de mettre en place, est l'élaboration d'une **base de données orientée graphe**, avec le système de gestion de base de données **Neo4J**. Cependant, du fait que nous disposions d'un temps et d'une possibilité de communiquer entre nous relativement limités dans le contexte actuel, nous avons préféré nous investir dans la prise en main de la librairie SQLite et dans l'élaboration d'une base de données relationnelle. En effet, nous avions déjà des bases d'utilisation du langage SQL, et cette librairie nous a paru relativement aisée à prendre en main dans un court laps de temps. Nous aurions cependant trouvé cela plus intéressant dans le cadre de cette UE de pouvoir manipuler le système de gestion Neo4J, et ainsi de mieux appréhender la construction et l'organisation des bases de données de type graphes. Nous aurions également pu nous entraîner à élaborer des requêtes sur notre base de données graphe avec le **langage Cypher**, que nous avions commencé à utiliser en Travaux Pratiques. Par appréhension de ne pas réussir à obtenir une base de données fonctionnelle dans le temps imparti et de ne pas réussir à maîtriser suffisamment ces outils, nous avons finalement opté pour l'élaboration d'une base de données relationnelle, et l'exploitation de la librairie SQLite. Nous présentons les résultats obtenus pour cette dernière dans la dernière partie de ce rapport, et, le script permettant la création des tables et leur enrichissement se trouve au sein du fichier **creationBDRelationnelle.py** dans le dossier **lib**.

4 Traitements Biopython Implémentés

Nous avons réalisé différents traitements bioinformatiques avec Python. Nous avons d'une part exploité la librairie **Biopython** afin de récupérer des séquences au sein de bases de données du NCBI, et de réaliser des Blast avec ces dernières. Nous avons d'autre part réalisé un script permettant l'alignement global de deux séquences. L'ensemble des traitements sont accessible via l'API **MainSequenceAnalyzer**. La première étape à tous traitements est la récupération des séquences, puis, deux possibilités sont offertes à l'utilisateur, l'alignement par paire global des séquences et/ou un Blastn sur le serveur du NCBI de la séquence du gène mutant. L'ensemble des traitements se lance avec le fichier **MainSequenceAnalyzer.py**, la liste des paramètre est accessible en ajoutant le paramètre **'-h'** ou **'- -help'**.

4.1 Récupération des séquences sur le portail Entrez du NCBI

Afin de pouvoir travailler sur les séquences des gènes sauvage et mutant, il est nécessaire de les récupérer. Pour ce faire, nous avons utilisé la fonction **Entrez** de la librairie **Biopython** qui permet d'effectuer des requête sur la base de données du NCBI. En l'occurence, nous interrogeons la base **nucleotide** pour récupérer les séquences des ARN messagers correspondants aux gènes sauvage et mutant considérés. Deux requêtes sont construites, une première pour le gène sauvage et une seconde pour le gène mutant. Les différents paramètres de ces requêtes sont le nom du gène, l'organisme, la propriété ARN messager, et la base de données (nucléotidique ou protéique).
Les résultats sont sauvés dans un fichier **fasta** dans le dossier **Résultats**, et les séquences sont stockées en mémoire sous la forme d'objet **SeqRecord** afin d'être manipulées par l'API pour leur alignement ou pour lancer un Blast. L'implémentation de la récupération des séquence se trouve dans le fichier **EntrezRequest.py** dans le dossier **lib**.

4.2 Alignement par paire global de deux séquences

L'un des traitements Python que nous avons pu réaliser est l'implémentation d'un programme permettant l'obtention de **l'alignement global** de deux séquences de meilleur score en utilisant le principe de la **programmation dynamique**, et plus particulièrement la méthode de Needleman et Wunsch. L'algorithme permettant la réalisation de ce type d'alignement a en effet été abordé dans le cade de l'UE HMSN206 Partie Alignement. Ce type d'alignement permet la comparaison de deux séquences, et pourrait par exemple, permettre d'analyser les différences entre la séquence d'un gène sauvage, et la séquence mutée du même gène. Cela permettrait de repérer les différences entre les deux séquences, et de repérer par exemple les position des différents évènements mutationnels, que ce soient des substitutions, ou des insertions/délétions, étant à l'origine des différences de phénotype chez les individus portant ces gènes.
Dans le cadre de la recherche de l'alignement de **meilleur score** entre deux séquences, la programmation dynamique consiste à se servir des scores des sous-alignements entre les préfixes des séquences, calculés au fur et à mesure, afin d'obtenir le meilleur score d'alignement entre ces deux dernières. Autrement dit, la résolution du problème global, ici l'obtention de l'alignement de meilleur score entre les deux séquences à l'étude, se fait en combinant les

solutions des sous problèmes, qui sont ici les calculs des scores d'alignement de l'ensemble des préfixes des séquences.
Concrètement, la première partie de la résolution de ce problème est l'élaboration d'une **matrice de scores**, au niveau de laquelle se trouvent l'ensemble des scores des sous alignements. Cette matrice est ainsi composée d'un premier axe qui correspond à l'une des séquences, et d'un deuxième axe correspondant à l'autre séquence. Cette dernière est remplie au fur et à mesure, en commençant à calculer les scores d'alignement des premières bases des séquences. L'algorithme implémentant cette méthode et permettant la création de la matrice se trouve au niveau du script **alignement.py** des lignes 51 à 87,
La deuxième partie, une fois la matrice de score établit, est appelée l'étape de **backtracking**, et permet de récupérer et de produire l'alignement de meilleur score en parcourant la matrice. L'algorithme permettant de réaliser cette étape se trouve des lignes 93 à 126 du script **alignement.py**.
Nous décidons de permettre à l'utilisateur de choisir la méthode utilisée : méthode de score ou méthode de distances, ainsi que les différentes pénalités appliquées aux évènements d'identité, de substitutions, et d'insertions/délétions, apparaissant entre les deux séquences. Nous décidons d'afficher le résultats de l'alignement au sein d'un fichier texte, dont un exemple peut être visualisé au niveau du fichier **resultats_alignement.txt**, normalement généré par le script, ainsi que grâce aux Figures 7 et 8. Cet exemple concerne une portion du gène NRT1 que nous avons modifiée à la main et alignée sur elle même, pour tester le script.

```
2020-03-24 19:11:13.934221

********Resultats de l'alignement********


Sequence 1 : ACGGCTATCGACTCGTGAGTTTTCCCCCT
Longueur sequence 1 : 489 nucleotides

Sequence 2 : ACGGCTATCGACTCGTGAGTTTTCCCCCTC
Longueur sequence 2 : 517 nucleotides

Penalite d'identite : 2
Penalite de substitution : -1
Penalite de gap : -1

Methode par similarite

****************************************

Score de l'alignement : 830

Pourcentage d'identite : 84 %
Pourcentage de substitution : 2 %
Pourcentage de gap : 13 %

****************************************
```

Figure 7: Affichage des caractéristiques de l'alignement global réalisé entre deux séquences et présentes au niveau de l'entête du fichier résultats.

Nous décidons notamment d'inclure au niveau de ce fichier résultat les longueurs des séquences, les paramètres utilisés pour les méthodes de score ou de distances, ainsi que les pourcentages d'identité, de substitution, et de gap retrouvé entre les deux séquences. Enfin,

nous affichons le meilleur score de l'alignement global de ces deux séquences. Dans le cas où plusieurs alignements seraient réalisés avec différents paramètres, nous affichons également la date et l'heure de réalisation afin de pouvoir garder un historique des différents tests réalisés.

Concernant l'affichage de **l'alignement** des deux séquences en lui-même, nous décidons de présenter ce dernier par tranche de 30 bases, et d'indiquer les positions des premières bases. Nous présentons la première séquence seq1 en haut, et la deuxième séquence seq2 en bas. Les gaps sont représentés par des tirets - (Figure 8).

Figure 8: Représentation de l'alignement de deux séquences au niveau du fichier résultat produit par le script. L'alignement est représenté par tranches de 30 positions, et la portion de l'alignement des positions 91 à 180 est ici affichée.

Nous pouvons noter que la **complexité** de cet algorithme est proportionnelle au produit des longueurs des séquences. Ainsi, si son utilisation est envisageable dans le cadre de la réalisation d'un alignement deux à deux de séquences relativement courtes, ce sont d'autres solutions qui doivent être envisagées lorsqu'une requête est alignée sur de nombreuses autres séquences au sein d'une base de données, ou que la longueur de ces dernières atteint des millions voire des milliards de bases. Nous pouvons citer le cas de l'outil **Blast**, qui permet de rechercher des alignements locaux de scores élevés entre une séquence requête et de très nombreuses séquences cibles contenues au sein de bases de données. Cet outil utilise ainsi une heuristique.

4.3 Blast de la séquence du gène mutant sur le portail du NCBI

Nous proposons un troisième traitement qui est un **Blast** de la séquence du gène mutant sur le portail du NCBI à l'aide de la fonction **NCBIWWW.qblast** de la librairie **Biopython**. Un **Blastn** est réalisé avec la séquence nucléotidique du gène mutant sur la base **nucleotide** du NCBI. Les résultats sont sauvés dans un fichier XML dans le dossier **Résultats** et nous proposons à l'utilisateur, via une invite dans le terminal, d'afficher le nombre de résultats de sont choix (du meilleur au moins bon, classement standard du Bast NCBI, en fonction de l'e-value). Les fonctions de Blast sont implémentés dans le fichier **Blast.py** dans le dossier **lib**.

5 Résultats

Notre tentative d'implémentation d'une base de données contenant les informations fournies par les élèves du Master Biologie des Plantes, se fait dans le cadre d'un paradigme **relationnel**. Le script Python permettant la récupération des données au sein du fichier CSV, la création des différents tables, leur enrichissement se trouvent au niveau du fichier **creationBDRelationnelle.py**. Quelques exemples de requêtes de la base ainsi qu'une fonction de requêtage à la demande se trouvent dans le fichier **requestBD.py**. Le script est ainsi divisé en plusieurs parties, correspondant chacun à une étape. Ce script exploite en outre la librairie **SQLite** et prend en paramètre le fichier CSV contenant les données relatives aux traits morphologiques des plantules. Il permet également de générer un fichier résultat **Resultats_requetes_SQL** au niveau duquel se trouve les résultats des requêtes SQL adressées à notre base de données.

5.1 Modèle relationnel

Afin de réaliser la base contenant les données à l'étude, en suivant un modèle **relationnel**, nous utilisons le langage **SQL** : Structured Query Language. Celui-ci permet notamment de réaliser les différentes opérations désignées par l'acronyme CRUD : Create, Read, Update et Delete. Ce langage nous permettra ainsi de construire les différentes tables définies au niveau des diagrammes UML, de les enrichir avec les données fournies, puis d'accéder aux informations désirées par le biais de diverses requêtes.

Nous créons les différentes tables avec la commande CREATE TABLE. Dans le cadre de cette base de données relationnelle, nous ajoutons aux tables du diagramme de classes UML de la Figure 1 **l'attribut IdSeedling**, permettant de lier les différents objets **Feature**, **Plate**, et **Grouping**, avec les objets **Seedling** adéquats. Nous précisons qu'ayant élaboré la base de données un certain temps après l'élaboration des diagrammes UML, que nous pensions initialement destinés à une base de données graphe, certains changements ont été effectués entre les deux.

5.2 Explication du chargement

Afin de pouvoir récupérer les données à mettre dans notre base, nous parsons avec Python le fichier CSV et mettons l'ensemble des informations dans un dictionnaire de dictionnaires. Nous pouvons ainsi ensuite parcourir ce dictionnaire afin d'enrichir nos tables créées précédemment, par le biais des commandes SQL et de la **librairie SQLite**.

Nous traduisons notamment les milieux renseignés sous la forme "Milieu_i" au niveau des fichiers CSV par leur composition dans le code Python. Autrement dit, les tables sont enrichies directement avec les valeurs "Standard" ou "Saccharose", par exemples.

Dans une version plus avancée du projet, nous aurions également pu enrichir la table **NucleicAcidSequence** avec les résultats obtenus par le biais des requêtes réalisées avec Biopython et permettant de récupérer certaines séquences nucléiques au sein d'autres base de données sur NCBI.

5.3 Manipulation avec un jeu de requêtes et des visuels de résultats

Nous formulons quelques exemples de requêtes SQL sur notre base de données relationnelle dans le script **requestBD.py**. Nous écrivons les résultats de ces requêtes dans un fichier nommé **Resultats_requetes_SQL** enregistré dans le dossier **Resultats**, généré par le script.
Nous offrons la possibilité à l'utilisateur de formuler ses propres requêtes en langage **SQL**. Le résultat de la requête est directement affiché à l'écran (Figure 9).

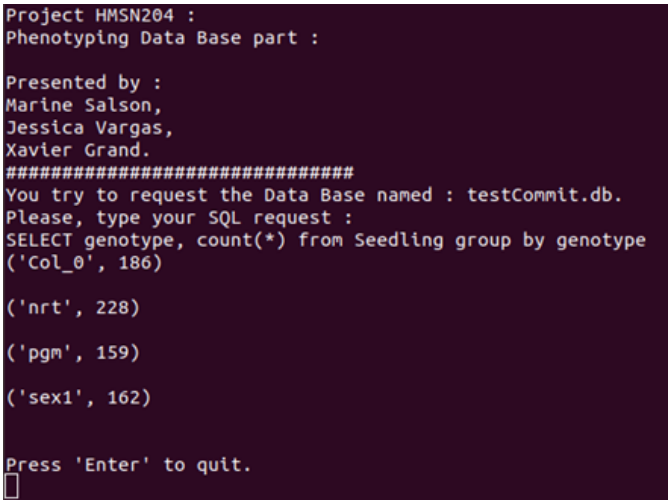

Figure 9: Affichage du résultat d'une requête SQL lancée par l'utilisateur via l'invite de l'API **MainCreationDB**.

5.4 Affichage des résultats de l'analyse des séquences

Afin de rendre le programme plus interactif, nous avons codé des messages destinés à l'utilisateur pour le tenir informé de l'avancement des opérations et de la localisation des fichiers de résultats, voire de résultats eux-mêmes. La figure 10 présente les messages renvoyés à l'écran.

Par ailleurs, un exemple concret de résultat d'alignement par paire global entre NRT2 et le gène muté nrt2.5 se trouve au sein du fichier NRT2_Vs_nrt2.5_SeqAlignment.res au niveau du répertoire Resultat.

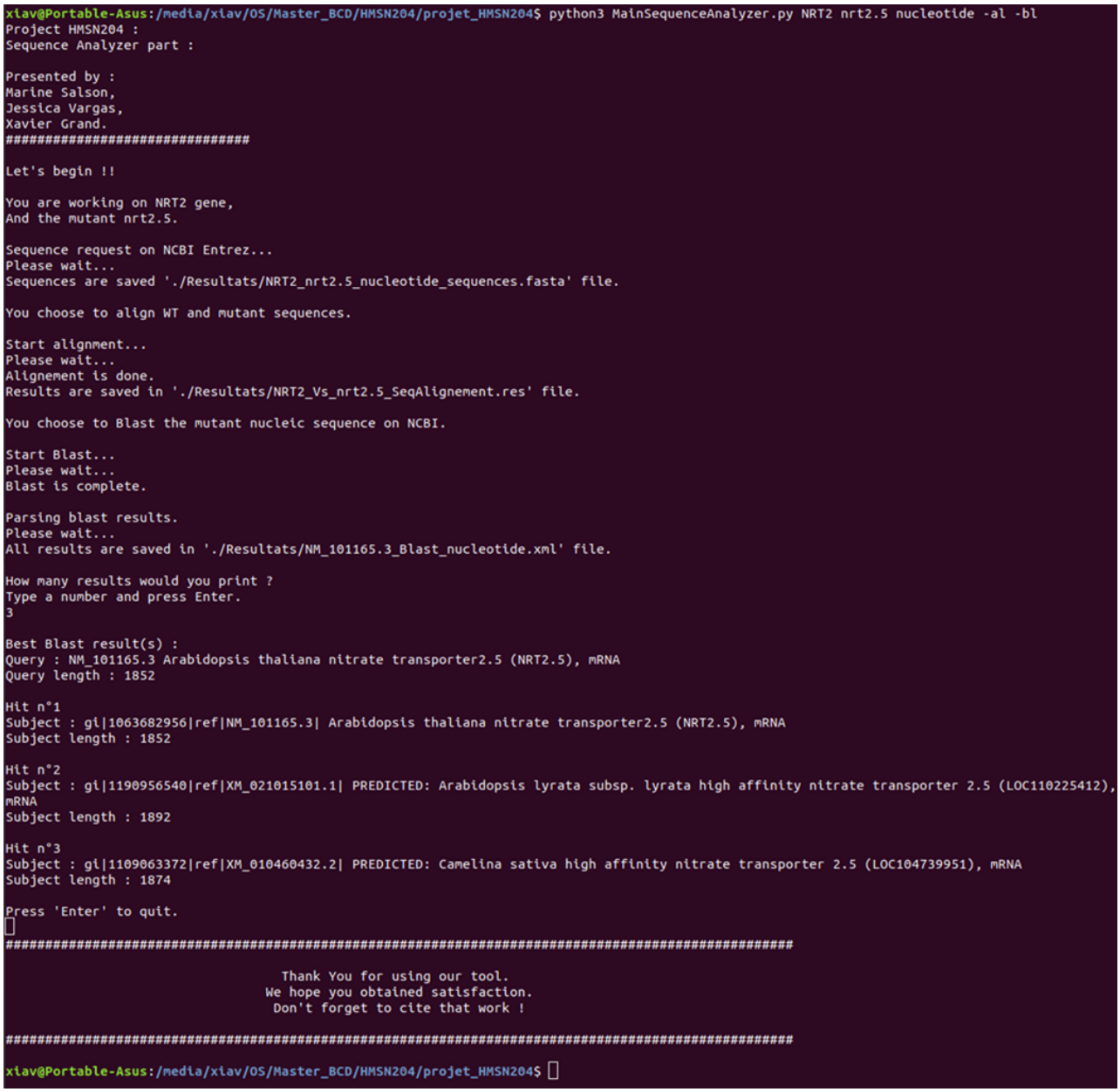

Figure 10: Messages renvoyés par le programme à destination de l'utilisateur.

6 Conclusion et perspectives

A l'issue de ce projet nous avons pu développer une API en partie fonctionnelle qui permet la création de manière persistante d'une base de données de type relationnelle et de l'interroger. Ces fonction sont intégrées dans un programme implémenté en python, simple d'utilisation, qui permet à un utilisateur non bioinformaticien de stocker et exploiter ses données de phénotypage de façon automatisée. De plus, nous avons réalisé une API complémentaire, qui pourra être intégrée à la première, et qui permet également de travailler sur les séquences des gènes sauvage et mutant considérés, c'est à dire les gènes altérés des plantes phénotypées. Toutefois, de nombreuses améliorations peuvent être apportées à ce programme.

6.1 Paradigme de la base de données

La réalisation de la base de données relationnelle nous a été plus facile à mettre en place qu'une base de données de type graphe. Nous n'avons pas exploité l'architecture en graphe des termes d'ontologie PO et PATO, seulement les termes. La base de données de type graphe est plus adaptée à l'architecture ramifiée de l'Ontologie.

Toutefois, l'intégration et la construction automatisée de la base de données implémenté en langage python sont réussies.

6.2 Amélioration de l'interrogation de la base de données

Bien que nous donnions la possibilité à l'utilisateur de réaliser ses propres requêtes sur la base de données, notre API impose de connaître quelques rudiments du langage SQL. Aussi, il serait peut-être possible de faciliter l'interrogation par un jeu de questions concrètes à l'utilisateur qui permettraient de générer une requête SQL de manière automatique. Cela aurait pour avantage d'ouvrir l'utilisation de notre programme à des biologistes qui n'ont pas de connaissance en SQL. En revanche, nous entrevoyons soit un très grand nombre de possibilités de requêtes et donc d'une gestion assez lourde, soit une interrogation limitée de la base de données à un nombre réduit de requêtes possibles.

6.3 Récupération des séquences

Notre programme, dans sa version actuelle, permet de récupérer des séquences d'ARN messager à partir d'un nom de gène uniquement chez *Arabidopsis thaliana*. De même que plusieurs paramètres ont été fixés tels que l'adresse e-mail associée à la recherche sur NCBI, ou encore différentes propriétés ('features'). Il est possible d'intégrer un certains nombre de paramètres à faire remplir par l'utilisateur, via un fichier de configuration par exemple.
Nous n'avons pas non plus de certitudes quant aux éventuels problèmes qui pourraient subvenir, malgré nos différents tests, il doit subsister des cas de figures que nous n'avons pas envisagés et qui pourrait constituer des 'bugs'. Pour cela une phase de beta-test pourrait être nécessaire.
Enfin, dans l'état actuel du programme, les séquences récupérées sont les résultats de la recherche sur le portail NCBI les plus pertinents. Il n'en reste pas moins qu'il s'agissent d'un résultat qui n'est pas contrôlé, c'est à dire ue nous ne pouvons pas garantir que le premier

résultat renvoyé par la recherche soit celui attendu par l'utilisateur. En particulier en ce qui concerne le gène mutant. Nous nous sommes interrogé sur la façon d'envoyer une requête sur une base de séquences dédiée à *Arabidopsis*, TAIR https://www.arabidopsis.org/. En effet, il est possible de trouver des résultats peut-être plus complet avec la contrepartie de ne contraindre notre programme qu'à cet organisme.

6.4 Alignement des séquences

Concernant l'alignement global de deux séquences, il aurait été intéressant de remplir les tables NucleicAcidSequence et GlobalAlignment de la base de données avec les caractéristiques des séquences récupérées, ainsi qu'avec les résultats obtenus pour les alignements réalisés entre elles. Cela aurait permis de stocker au sein de la base ces informations, et de les rendre accessibles ultérieurement, et par d'autres utilisateurs. Il aurait également été possible de comparer les résultats obtenus par notre script avec ceux d'un outil bioinformatique d'alignement global, avec deux mêmes paires de séquences, afin de vérifier la cohérence de celui-ci.

6.5 Blast

Il est possible grâce à notre programme de lancer le Blast de la séquence du gène mutant sur la base 'nucleotide' du NCBI. Les résultats sont sauvé sur la machine de l'utilisateur, ils sont donc conservés et donc consultables à volonté. Nous proposons également d'afficher les résultats les plus pertinents. Toutefois, l'analyse de ces résultats n'est que très succincte. Il pourrait être envisagé de parser les résultats de Blast et de proposer à l'utilisateur d'accéder à plus d'informations que nous ne lui proposons à l'heure actuelle.

6.6 Bilan de réalisation

Dans le but de faciliter l'accès aux données à un utilisateur non bioinformaticien, nous avons cherché à rendre l'utilisation de notre programme la plus simple possible. C'est donc dans cet état d'esprit que nous avons effectué nos choix de développement. La contrepartie de ce choix est que les possibilités disponibles pour l'utilisateur sont réduites.
En fin de compte, nous proposons une première version d'un outil à parfaire qui rempli les fonctions décrites dans le cahier des charges du projet de manière intégrée.