BOT DE GESTION DE CONNAISSANCES

Massinissa Mohamed BELAID
massinissa-mohamed.belaid@etu.umontpellier.fr
Ryadh BELGHANEM
ryadh.belghanem@etu.umontpellier.fr
Alexy LEFEVRE
alexy.lefevre@etu.umontpellier.fr

UncompilatedName

Mai 2020

Table des matières

Remerciements

Avant de commencer, nous voudrions remercier nos deux encadrants, Madame Prince et Monsieur Lafourcade. Merci à Madame Prince, qui nous a grandement aidé pour la définition des patterns de discussion, à la désambiguïsation des propos, ainsi qu'à la compréhension du fonctionnement d'une intelligence artificielle, dans le domaine du TALN. Merci également à Monsieur Lafourcade pour son aide précieuse avec les relations ainsi que sur JeuxDeMots et son réseau pas facile à appréhender. Merci à eux deux, pour leur temps et leur savoir, pour nous avoir aiguillé, orienté et encadré. Sans eux nous ne savons pas si nous aurions pu arriver aussi loin en aussi peu de temps dans ce projet.

Chapitre 1

Introduction

Dans le cadre du TER (Travail Encadré de Recherche) pour le Master 1 informatique AIGLE, nous avons choisi le sujet intitulé "Bot de gestion de connaissances". Ce que nous devions réaliser, c'est un bot capable de répondre à des questions de connaissances telles que : "est-ce qu'un homme est un humain ?" ou bien "est-ce qu'une voiture est composée d'un moteur ?" ou encore "pourquoi un soldat possède un fusil ?". Ceci dans le but de consolider (augmenter et corriger) une base de connaissances (en l'occurrence, la base RezoJDM). Étant très libres sur la réalisation du sujet, nous avons choisi le langage python pour le développer. Après plusieurs semaines de test, au fur et à mesure, notre bot a grandi, nous lui avons trouvé un nom : Greg.

Nous allons dans ce rapport vous détailler comment fonctionne Greg, ce à quoi il peut actuellement répondre, et ce à quoi il n'a pas encore les réponses. Tout d'abord nous commencerons avec une présentation de JeuxDeMots, de sa base de connaissance et comment nous nous en servons. Par la suite nous vous présenterons comment sont construites et interprétées les discussions entre l'utilisateur et Greg. Ensuite nous verrons les outils qui nous ont permis de mener à bien l'avancement du projet, en passant par le framework Django avec son architecture MVT, ainsi que par une base de données SQL hébergée en utilisant le SGBD MySQL. Avant de conclure, nous consacrerons un dernier chapitre à la gestion de notre projet, ceci pour répondre à la demande qui a été faite dans le cadre de l'UE de Conduite de projet que nous avons suivi en parallèle ce semestre. Finalement nous conclurons sur ce que ce travail nous a apporté et ce pourquoi il pourra être utilisé dans la suite des évènements.

Chapitre 2

JeuxDeMots

2.1 Présentation générale

JeuxDeMots est un jeu en ligne développé par le Laboratoire d'Informatique Robotique Micro-électronique de Montpellier (LIRMM), lancé en juillet 2007 par Mathieu Lafourcade. L'utilisateur peut y lancer une partie pendant laquelle il lui sera proposé un mot. Il devra ensuite dans un temps limité associer le plus de mots possible en rapport avec celui qui lui est proposé pour engendrer un maximum de points. Les points ainsi gagnés sont fonction du nombre de mots en commun entrés par les différents utilisateurs. Plus un mot est utilisé par les joueurs, plus il rapporte de points. Le but pour le joueur est de collectionner des termes. Ceux-ci peuvent être gagnés en réalisant des scores importants, ou par d'autres moyens laissés à la découverte du joueur.

L'intérêt d'un tel jeu pour le LIRMM réside dans le fait qu'il produit un réseau lexical en fonction des réponses données par les joueurs. Ce jeu s'adresse aux amoureux de la langue et du vocabulaire, qu'il soit général ou de spécialité, c'est l'outil d'un programme de recherche en Traitement Automatique du Langage Naturel (TALN) [1].

2.2 Base de connaissance

Le projet JeuxDeMots (JDM) vise donc à construire un réseau lexical et sémantique du Français (rezoJDM). Une telle base de connaissances est utile pour de nombreuses applications, dont l'analyse sémantique de textes, la recherche d'informations, la traduction automatique, etc...
Le réseau lexical de JeuxDeMots est sous-forme d'un graphe, où les noeuds correspondent aux termes, et les arcs les reliant sont des types de relation entre ces termes. Les arcs (relations) sont orientés (ont un sens) et vont donc d'un terme

à un autre. Ces relations ont un poids qui démontre la fréquence avec laquelle les deux termes sont associés. Un poids négatif représente l'inexistence (ou bien la forme négative) d'une relation entre deux termes.

Exemple : On peut avoir deux termes : "voiture" et "moteur", puis un arc entre les deux avec un poids positif, représentant la relation "est composé" (r_has_part) allant de "voiture" à "moteur", ce qu'on pourrait traduire en français : une voiture peut être composée d'un moteur.

Cette base de connaissance très conséquente, a quand même un petit inconvénient : ce sont les joueurs qui l'alimentent. On peut alors trouver par endroit de légères incohérences. Ces incohérences amènent parfois Greg à se tromper ou à affirmer des choses qui en réalité s'avèrent fausses.

2.3 Notre utilisation

La base de connaissances de JeuxDeMots étant très riche, nous avons dans un premier temps fait naître notre bot dans un plus petit environnement. C'est seulement une fois qu'il fut capable de comprendre nos questions et d'y répondre, que nous lui avons fourni la base de connaissance de JeuxDeMots.

Greg peut être vu comme un robot intelligent avec lequel on peut entretenir une pseudo conversation. L'utilisateur doit avoir la possibilité de poser différentes questions, donc Greg doit avoir une grande capacité de compréhension. Nous verrons comment il est capable de comprendre l'utilisateur dans les chapitres qui suivent. Une fois la phrase de l'utilisateur comprise, Greg va piocher toutes les informations nécessaires à la réponse dans la base de connaissance.

Pour notre projet "Bot de gestion de connaissances", on se sert donc des relations et termes appartenant à la base de connaissance de JeuxDeMots, pour répondre aux questions de l'utilisateur. Pour ce faire, Greg extrait de JeuxDeMots les termes et la relation qui les lient s'ils existent, sinon il répond qu'il ne sait pas et pourra poser la question à un autre utilisateur plus tard.
Lorsque c'est Greg qui pose une question à l'utilisateur, les réponses de ce dernier nous permettent également d'augmenter la base de connaissance avec de nouvelles relations, ou bien de corriger les relations qui sont déjà dans JeuxDeMots.
JeuxDeMots possède plus d'une centaine de différents types de relations, nous avons travaillé sur une petite sélection de relations disponibles dans le réseau JeuxDeMots, pour les utiliser dans les échanges avec l'utilisateur. On pourrait bien entendu travailler avec toutes les relations disponibles, mais cela impacterait fortement les temps de recherche dans la base et donc le temps de réponse. Nous avons donc fait un choix, en accord avec nos encadrants, pour ne sélectionner que les relations les plus importantes.

Chapitre 3

Conception

Dans ce chapitre nous décrivons les différentes phases qui nous ont permis de mettre au monde Greg ainsi que les différentes possibilités d'interaction avec lui.

3.1 Base de données

Vous l'aurez peut-être compris, notre base de données est créée et augmentée au fur et à mesure que Greg découvre de nouveaux mots, en allant les chercher chez JeuxDeMots ainsi qu'en interagissant avec les utilisateurs. Voici son schéma entité-association :

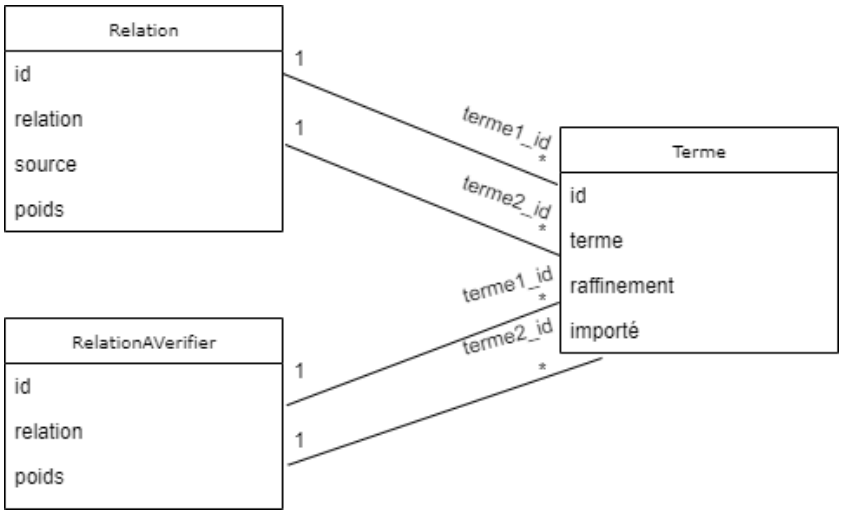

FIGURE 3.1 – Schéma de la base de données

Pour la table **Terme** :
— l'identifiant du terme
— le terme
— le raffinement si ce n'est pas un terme générique (nous reviendrons sur ce point dans la partie qui traite les raffinements sémantiques)
— importé qui est un booléen qui nous dit si les relations de ce termes ont été importées dans la base ou non

Pour la table **Relation** :
— l'identifiant de la relation
— la relation qui est en fait son type parmi celles présentes (r_isa, r_has_part, ...)
— la source qui indique la provenance de la relation (JDM ou UN). Cela indique si elle a été directement importé de JeuxDeMots (JDM) ou bien si ce sont les utilisateurs qui ont donné ces informations à Greg (UN). C'est un attribut qui est important pour M. Lafourcade s'il veut récupérer des informations et pouvoir alimenter son RezoJDM.
— le poids de la relation qui aide à nuancer les propos et interprétations de Greg
— les identifiants du premier terme (terme1_id) et du second terme (terme2_id) de la relation qui sont, bien entendu, des clés étrangères vers la table **Terme**

La table "Relation à vérifier" est très similaire à celle de "Relation", la seule différence est qu'elle n'a pas besoin de l'attribut "source". Effectivement, c'est, entre autre, lorsque ces relations sont retirées de cette table et ajouté dans la table "Relation", qu'elles reçoivent comme attribut "source" la valeur "UN" (UncompilatedName).

3.2 Construction des relations

3.2.1 Analyse des relations

Nous nous sommes basés sur les relations : r_isa, r_has_part, r_carac, r_own, disponibles dans JeuxDeMots pour construire les différents patterns de discussions dont nous aurons besoin pour échanger avec l'utilisateur.

— "r_isa" représente une relation de sous-classe entre deux éléments, par exemple : "un chat est sous-classe de animal", qui se traduit dans la base de connaissance par une relation de type r_isa allant du terme "chat" au terme "animal" avec un poids positif. Cela donne :

terme1 = 'chat' ; relation = r_isa ; terme2 = 'animal' ; poids = 50

— "r_has_part" représente une relation de composition entre deux éléments, par exemple : "une main est composée de doigts", qui

se traduit dans la base de connaissance par une relation de type r_has_part allant du terme "main" au terme "doigts" avec un poids positif. Cela donne :

terme1 = 'main' ; relation = r_has_part ; terme2 = 'doigt' ; poids = 150

— "r_carac" représente une relation définissant la caractéristique d'un élément, par exemple : "un loup peut être qualifié de dangereux", qui se traduit dans la base de connaissance par une relation de type r_carac allant du terme "loup" au terme "dangereux" avec un poids positif. Cela donne :

terme1 = 'loup' ; relation = r_carac ; terme2 = 'dangereux' ; poids = 70

— "r_own" représente une relation de possession entre deux éléments, par exemple : "un soldat possède un fusil", qui se traduit dans la base de connaissance par une relation de type r_own allant du terme "soldat" au terme "fusil" avec un poids positif. Cela donne :

terme1 = 'soldat' ; relation = r_own ; terme2 = 'fusil' ; poids = 80

3.2.2 Les relations dans des phrases

Nous avons donc réalisé des templates de phrases pour chacune des relations que nous avons choisies. Ces patterns nous permettent d'exprimer par une phrase en français une relation allant d'un premier terme (T1) à un deuxième terme (T2). Nous allons vous présenter les différents patterns de phrases actuellement compris par Greg. On pourrait bien entendu en ajouter et donc améliorer sa compréhension autant que nécessaire. Les mots entre parenthèses sont optionnels, lorsqu'ils sont séparés par un slash alors il faut en choisir un seul parmi les propositions.

Relation "r_isa" :

— (un/une/des/le/la/le/l') T1 est/sont (une) sous-classe (de/d'un/ d'une/de la/de le/des/de l'/d') T2

— (un/une/des/le/la/le/l') T1 appartient/appartiennent à la classe (de /d'un/d'une/de la/de le/des/de l'/d') T2

— (un/une/des/le/la/le/l') T1 est/sont (un/une/des) T2

Relation "r_has_part" :

— (un/une/des/le/la/le/l') T1 est/sont (une) est composé(e) (de/d'un/d'une/de la/de le/des/de l'/d') T2

— (un/une/des/le/la/le/l') T2 est une partie (de/d'un/d'une/de la/de le/des/de l'/d') T1

— (un/une/des/le/la/le/l') T2 fait partie (de/d'un/d'une/de la/de le/des/de l'/d') T1

Relation "r_carac" :

— (un/une/des/le/la/le/l') T1 est/sont (une) peut être qualifié(e) (de/d'un/d'une/de la/de le/des/de l'/d') T2

— (un/une/des/le/la/le/l') T1 est qualifié(e) (de/d'un/d'une/de la/de le/des/de l'/d') T2

— (un/une/des/le/la/le/l') T1 avoir comme propriété (de/d'un/d'une/de la/de le/des/de l'/d') T2

Relation "r_own" :

— (un/une/des/le/la/le/l') T1 peut posséder (de/d'un/d'une/de la/de le/des/de l'/d') T2

— (un/une/des/le/la/le/l') T2 possède (de/d'un/d'une/de la/de le/des/de l'/d') T1

3.2.3 Les interprétations du poids

Pour que Greg soit assez précis dans ce qu'il répond, il faut qu'il puisse pondérer ses réponses. Nous avons déjà parlé de poids positifs, qui informent qu'une relation est vrai, et négatifs, qui informent qu'une relation est fausse, mais cela ne suffit pas réellement pour pouvoir décrire le monde qui nous entoure. C'est pourquoi nous avons mis en place une échelle des poids qui permet de catégoriser ce dernier. Nous allons parler de "oui fort", "oui faible","sais pas", "non faible et "non fort" :

— **oui fort** : c'est le oui absolu, utilisé lorsqu'une relation est vraie dans une très grande majorité des cas, voire presque tout le temps. Exemple : un chat est composé de griffes.

— **oui faible / non faible** : c'est un oui/non partiel, utilisé lorsqu'une relation est parfois vraie et parfois fausse, lorsqu'on ne sait pas trop. Exemple : un chat peut posséder un jouet.
— **sais pas** : utilisé seulement lorsqu'une relation n'est pas trouvée dans la base de connaissance. Si c'est le cas alors Greg la garde en mémoire pour pouvoir poser la question à un utilisateur et ainsi la pondérer selon sa réponse.
— **non fort** : c'est le non catégorique, utilisé pour une relation qui est fausse tout le temps. Exemple : un chat est composé d'un moteur.

3.3 Les questions utilisateur

Pour interagir avec Greg, l'utilisateur doit respecter les quelques formats décrits dans nos patterns. Il peut poser deux types de questions : "est-ce que..." et "pourquoi...".

3.3.1 Questions "Est-ce que ?"

Lorsque l'entrée d'un utilisateur débute par "est-ce que", Greg interprète cela comme une question à laquelle il devra répondre par oui ou par non.

Exemple :
Utilisateur : Est-ce qu'un chat est une sous-classe d'animal ?
Greg : Oui, absolument.

En recevant une question du type "est-ce que ...", on isole les deux termes et le type de la relation de la phrase. Dans l'exemple ci dessus, les termes 'chat', 'animal' et la relation r_isa sont donc isolés puis extrait de la base de connaissance. Nous reviendrons sur ce point là dans le chapitre dédié. Qu'en est-il de la réponse ? Pour répondre correctement, Greg vérifie si la relation est directe sinon il essaie de la trouver par transitivité.

Réponse directe

Dans le cas où la relation serait directe, c'est à dire qu'il n'y a pas de ramification possible, Greg peut directement répondre en interprétant le poids de la relation qu'il vient de trouver. Il vérifie donc si la relation du bon type, allant du premier terme au deuxième, existe bel et bien dans la base de connaissance. Si elle existe, Greg répond par rapport au poids assimilé à la relation (oui fort / oui faible / non faible / non fort).

Réponse par inférence

Si la relation directe n'existe pas, alors on la cherche par transitivité. C'est-à-dire qu'on cherche un troisième terme pour lequel, une relation de ce type existe avec les deux autres termes, cette relation allant donc du premier terme vers le troisième et allant du troisième vers le deuxième terme. C'est en fait la recherche d'un terme intermédiaire que Greg fait.

Exemple : si les deux relations, "un chat est une sous-classe de mammifère", et "un mammifère est une sous-classe de animal" existent, alors elles vont nous permettre de savoir que "un chat est une sous-classe de animal".

Cette relation ainsi obtenue sera créée mais ajoutée dans une table de notre base de données bien spécifique : "relations à vérifier". C'est, entre autre, dans cette table que Greg pioche les questions qu'il pose aux utilisateurs. Cela permet de faire des vérifications sur ce qui est inféré et ainsi éviter que Greg "apprenne" mal.

Recherche de la relation pour les sous-classes du premier terme

Si l'on ne trouve pas de troisième terme avec lequel une inférence de ce genre est possible, alors on cherche une relation de ce même type allant des sous-classes du premier terme vers le deuxième terme.

Exemple : si les deux relations, "chat est une sous-classe de animal", et "animal est composé de tête" existent, elles vont nous permettre de savoir que "un chat est composé de tête".

Cette relation obtenue sera crée mais, comme précédemment, ajoutée dans la table des "relations à vérifier".

Raffinement sémantique des termes

Dans JeuxDeMots se trouve la relation "r_raff_sem", qui porte sur un terme polysémique (qui a plusieurs sens), et donne ses différentes définitions.

Exemple : le terme "pièce" qui peut avoir plusieurs sens tels que : "salle", "pièce de monnaie", "unité"... la relation nous donne donc ses raffinements sémantiques :

- pièce ; r_raff_sem ; salle
- pièce ; r_raff_sem ; pièce de monnaie
- pièce ; r_raff_sem ; unité

Nous utilisons cette relation, pour répondre aux questions "est-ce que...", lorsque l'on ne trouve pas la relation (qu'elle n'existe pas donc), ou si elle est trouvée, mais que son poids est négatif (donc un "non") pour le terme géné-

rique. Si c'est le cas alors il demande à l'utilisateur s'il veut parler de chacun des raffinements disponibles, et il répond en fonction de son choix.

Quand nous parlons de terme générique nous distinguons, en fait, le terme de son raffinement. Dans l'exemple précédent, "pièce" est également un terme qui existe dans la base de connaissances sans aucun raffinement, c'est celui là, le terme générique. C'est ce terme là qui, généralement, possède le plus de relations, c'est également le terme utilisé couramment, indépendemment du contexte.
Étant donné qu'il est difficile pour Greg de définir un contexte et donc de détecter quel sens l'utilisateur veut réellement donner à son terme, Greg peut demander à l'utilisateur pour être sûr de ne pas répondre à côté.

Exemple :
Utilisateur : Est-ce qu'un homme est un humain ?
Greg : Veux-tu parler d'un homme au sens soldat ?
Utilisateur : Non
Greg : Veux-tu parler d'un homme au sens être humain ?
Utilisateur : Non
Greg : Veux-tu parler d'un homme au sens être humain de sexe masculin ?
Utilisateur : Oui
Greg : Alors absolument.

Si Greg ne propose pas le raffinement qu'attend l'utilisateur, alors Greg lui répond qu'il ne sait pas et ajoute la relation dans la table "relations à vérifier".

S'il n'existe toujours pas de relation, que ce soit pour le terme générique ou pour ses raffinements, Greg répond qu'il ne sait pas, toujours en créant cette nouvelle relation dans la table "relations à vérifier".

3.3.2 Questions "Pourquoi ?"

L'utilisateur peut également interroger Greg d'une autre façon. En effet, en entrant une question qui commence par "pourquoi", le bot va devoir expliquer à l'utilisateur pourquoi il pense que cela est correct, si c'est la cas bien entendu.

Exemple :
Utilisateur : Pourquoi un chat est une sous-classe de animal ?
Greg : Je pense que c'est parce que chat est une sous-classe de petit félin, qui est une sous-classe de animal.

Pour répondre à une question "pourquoi", il faut, comme pour tout traitement de phrase, isoler les deux termes et la relation. Après avoir extrait les deux termes et le type de la relation de la question, on cherche à construire une transitivité avec un troisième terme intermédiaire. Pour cet intermédiaire, une relation de ce type existe avec les deux autres termes, allant du premier

terme vers le troisième et allant de ce troisième vers le deuxième terme. Si l'on trouve cette relation transitive alors on peut expliquer pourquoi, grâce à l'intermédiaire.

S'il n'existe pas de terme intermédiaire avec lequel on pourrait construire une transitivité, alors on essaie de trouver un terme qui serait en relation r_isa (une sous-classe) avec le premier terme. De ce fait, s'il possède une relation de ce type, allant de lui vers le deuxième terme, alors cela nous permet de formuler des réponses du style :
Utilisateur : "Pourquoi un chat est composé d'une tête ?"
Greg : "Certainement car un chat est une sous-classe de animal, qui est composé d'une tête"

S'il n'existe pas ce terme intermédiaire qui nous permet de répondre à la question, alors Greg répond en disant que c'est factuel si la relation est vraie. Si la relation a un poids négatif et est donc fausse, alors Greg lui répond qu'il pense que ce n'est pas le cas.
Utilisateur : "Pourquoi un chat est un moteur ?"
Greg : "Il me semble que ce n'est pas le cas."

3.4 Mode question

Une autre façon de dialoguer avec Greg, est de lui demander de nous poser une question. Greg construit alors des questions compréhensibles par l'utilisateur avec les relations contenues dans la table "relation à vérifier". L'utilisateur peut aussi demander au bot de lui parler d'un terme en particulier, le bot cherchera alors une relation avec ce terme. L'utilisateur va ensuite répondre à la question de l'utilisateur (oui, non, je sais pas, ...)

Exemple :
Utilisateur : Pose moi une question
Greg : Est-ce qu'un homme peut être qualifié de cultivé ?
Utilisateur : Oui
Greg : D'accord je note que c'est correct.
Utilisateur : Parle moi de chanteur
Greg : Est-ce qu'un chanteur peut avoir comme propriété artiste ?

Greg enregistre les réponses aux questions des utilisateurs. Une fois qu'il a assez de réponse pour une relation qu'il devait vérifier, il la déplace vers la table "relation" en lui affectant un poids qui est calculé en fonction des réponses des différents utilisateurs.
Dans le cas où Greg n'a pas de relation à vérifier pour construire une question, il utilise les relations qu'il connaît de la table "relation" pour construire une question. Ceci lui permet de moduler le poids de cette relation en fonction de la réponse de l'utilisateur, et donc adapter dynamiquement la base de connaissance.

3.5 Extraction des termes et relations de JeuxDeMots

Pour que Greg ait la compréhension la plus grande possible, il faut qu'il puisse reconnaître un maximum de termes. Nous avons dans un premier temps ajouté un module d'extraction de plus de 1200 termes les plus courants en français pour que Greg parte avec une base de connaissance assez solide. Les relations directement liées aux termes sont également récupérées, c'est donc une étape assez longue (plusieurs heures) mais primordiale au bon fonctionnement de Greg.

De plus, chaque entrée d'un nouveau terme par l'utilisateur, si Greg ne l'a pas en sa connaissance donc, est récupérée de JeuxDeMots avec toutes ses relations directes, pour les ajouter dans la base de données. Le temps d'exécution est assez variable ici, cela va dépendre du terme récupéré. Pour certains termes contenant beaucoup de relations, cela peut prendre un petit moment. Par exemple pour "chanteur", c'est un terme assez "gros" car il a des relations r_isa avec tous les noms propres qui sont des instances de chanteur, Jean-jacques Goldmann entre autre. Greg met un peu plus de 9 secondes à tout importer et répondre à l'utilisateur. C'est pour éviter ces temps de réponses assez longs, qui peuvent rendre la discussion barbante, que l'on a essayé d'importer le plus de mots couramment utilisés en amont.

Chapitre 4

Outils

Nous avons implémenté Greg sous forme d'une application web créée avec le framework Django en python, notre base de données, quant à elle, avec le SGBD MySQL.

4.1 Framework Django

Django est un framework open source Python destiné au web. Il a été créé en 2003 par quatre développeurs pour le journal local de Lawrence (dans le Kansas en Amérique) : Adrian Holovaty, Simon Willison, Jacob Kaplan-Moss et Wilson Mine. Depuis juin 2008, la Django Software Foundation a repris le relais pour développer et promouvoir le framework. Le but principal de Django, qui a présidé à sa création, est de simplifier la création de sites web complexes utilisant une base de données.[3]

4.1.1 Python

Python est un langage de programmation interprété, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.[4]

4.1.2 Présentation MVT (Models Views Template)

Le framework django utilise l'architecture MVT : Modèle-Vue-Template. Le MVT représente une architecture orientée autour de trois pôles : le modèle, la

vue et le template. Elle s'inspire de l'architecture MVC, très répandue dans les frameworks web. Son objectif est de séparer les responsabilités de chaque pôle afin que chacun se concentre sur ses tâches. On peut dire que dans le MVT, le View joue le rôle du contrôleur, et le template joue le roue de la View du MVC.

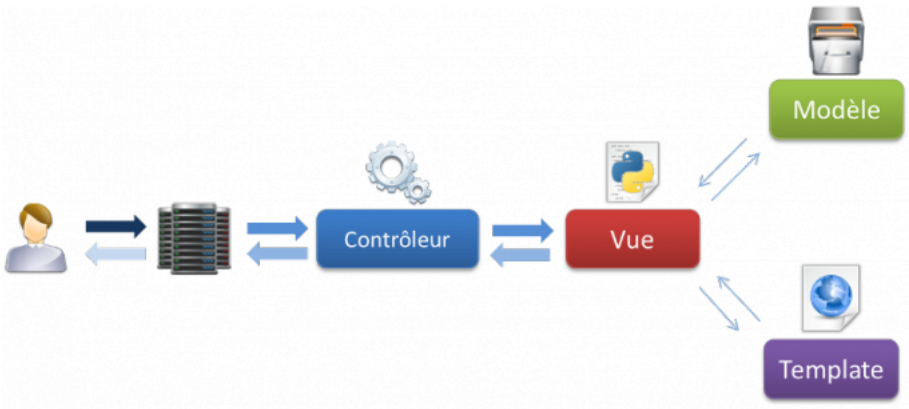

FIGURE 4.1 – Modèle View Template

4.1.3 Nos modèles

Nos modèles sont les classes Terme, Relation et RelationAVerifier qui représentent nos tables dans la base de données. Grâce aux ORM (Object Relational Mapping) on n'écrit plus de requêtes SQL, mais on travaille directement avec les objets de la base de données.

4.1.4 Nos Views

Les Views sont donc là pour jouer le rôle d'intermédiaire entre les templates (l'affichage) et les modèles (base de données). Pour implémenter Greg, nous avons eu besoin d'une seule View. Elle contient deux méthodes :

- Une permettant de prendre en charge les entrées de l'utilisateur lorsqu'il dialogue avec Greg. Elle se charge de récupérer les données de la base de connaissance ou de JeuxDeMots et c'est elle qui appelle les méthodes de tous les traitements avant de répondre à l'utilisateur.
- Une autre méthode pour procéder à l'extraction des termes les plus utilisés depuis JeuxDeMots.

4.1.5 Nos templates

Les templates sont définis par des fichiers HTML et CSS et décrivent nos interfaces homme-machine. Nous en avons crée deux, un premier qui est

le principal et qui représente la page web avec laquelle l'utilisateur peut interagir avec Greg. C'est donc dans ce template qu'il pourra écrire ses phrases et recevoir les réponses de Greg. Au début nous n'avions que ce template principal. Cependant, pour aider l'utilisateur, nous avons fait le choix de créer un deuxième template (une seconde page web donc) qui contient simplement les explications de fonctionnement de Greg et est accessible depuis un bouton sur la page principale.

4.2 Base de données : SQL

Le SQL (Structured Query Language) est un langage informatique qui permet d'interagir avec des bases de données relationnelles. C'est le langage pour base de données le plus répandu. Il a été créé dans les années 1970 et est devenu un standard en 1986 (pour la norme ANSI ; 1987 en ce qui concerne la norme ISO).[5]
La base peut être implémentée avec différents systèmes de gestion de base de données (SGBDR) SQL : Mysql, PostGreSQL ... Nous avons utilisé MySQL au départ, puis fait des tests avec PostGreSQL, finalement nous avons gardé MySQL.

Chapitre 5

Réalisation

Après avoir achevé la phase de conception, et présenté les outils que nous avons utilisé, nous vous présentons l'interface utilisateur réalisée pour la page web.

5.1 Interface web

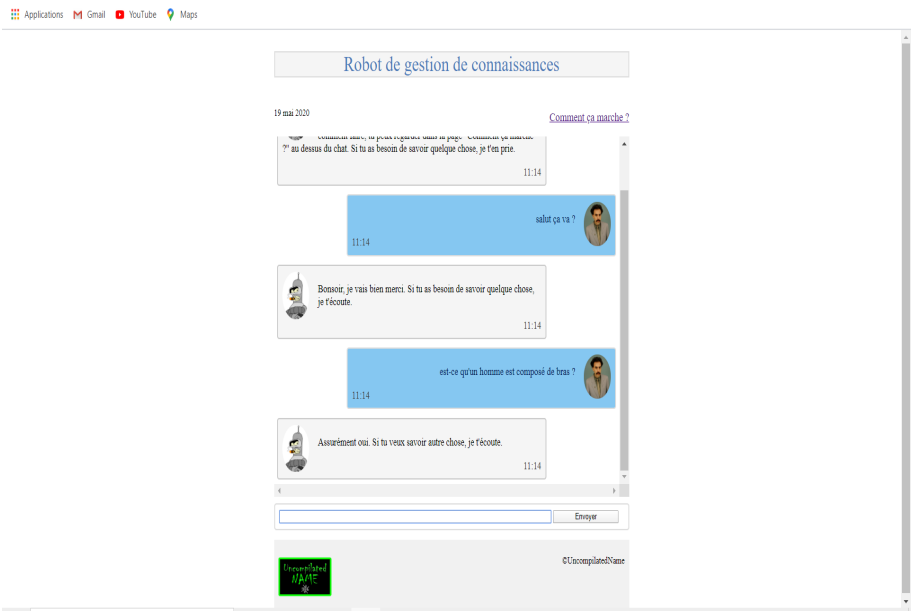

FIGURE 5.1 – Interface web

Voici donc ce que le rendu final donne du côté de l'utilisateur. Nous avons présenté l'interaction sous forme de discussion instantanée avec messages de bas en haut type Messenger. Une zone de saisie de texte où l'utilisateur peut écrire, ainsi qu'un bouton pour envoyer son message à Greg. Les messages utilisateurs sont en bleu collés sur la droite, ceux de Greg sont sur la gauche sur un fond gris.
Nous avons choisi d'ajouter un bouton en haut de page "Comment ça marche ?" pour que l'utilisateur qui veut en savoir plus puisse accéder à une deuxième page qui lui donnera les informations nécessaires pour discuter avec Greg.

5.2 Déploiement

Greg fonctionne très bien en local, mais n'a que peu d'intérêt dans un environnement comme celui-ci. Nous l'avons donc déployé avec Heroku, qui est un hébergeur qui propose certains de ses services gratuitement, mais qui restent limités. Il est accessible à partir du nom de domaine : `https://uncompilatedname.herokuapp.com/`. Pour éviter un soucis de compatibilité, qui empêche la barre de scroll de fonctionner, nous recommandons l'accès à partir du navigateur **Chrome**.

Le "problème" de Heroku est que son mode gratuit permet un nombre de lignes dans la base de données limité à 10 000. De ce fait, on ne peut pas stocker toutes les informations nécessaires aux bonnes réponses de Greg. De plus, Heroku n'accepte pas des temps de chargement trop long des pages. Par conséquent, cela rend difficile l'importation des "gros" termes depuis JeuxDeMots. Pour palier à ces soucis, nous avons transféré sur la base de données Heroku un ensemble de termes et relations pour pouvoir le tester sans qu'il ait à extraire. Il peut donc répondre à des questions portant sur : ordinateur, processeur, informatique, programme...

Chapitre 6

Conduite de projet

Pour mener à bien notre projet nous avons, entre autre, tenu à jour un Google Doc qui faisait office de journal de bord, utilisé un git, hébergé sur GitHub en l'occurrence (`https://github.com/MassiBelaid/uncompilated_name`), ainsi que le logiciel open source ProjectLibre. Après avoir décortiqué le travail à fournir et identifié les différentes tâches majeures, nous avons, fin février, établi un planning prévisionnel. Il a été réalisé à l'aide de ProjectLibre et représente ce à quoi nous pensions que notre projet allait ressembler à ce moment là. Dans un soucis de lisibilité, nous avons fait le choix de n'afficher que le diagramme de Gantt avec le nom des tâches qui leur est associé. Le voici :

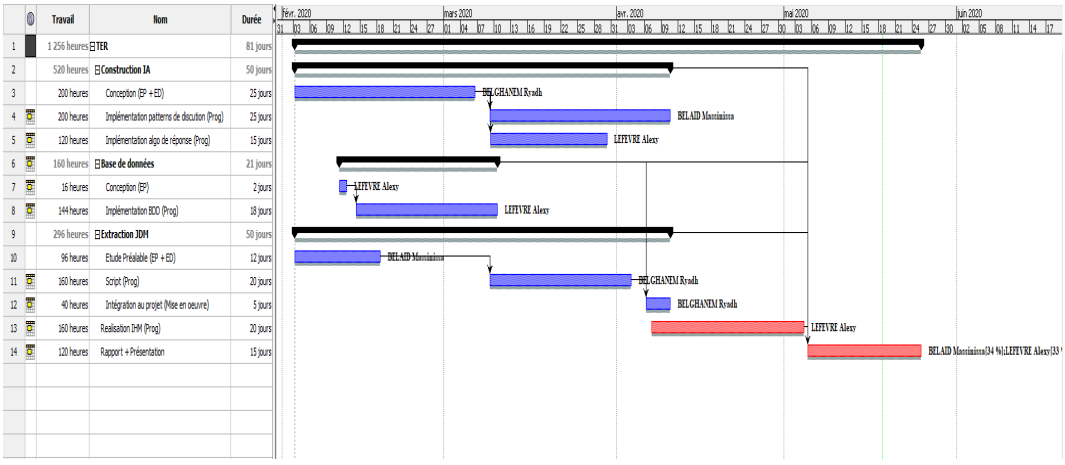

FIGURE 6.1 – Diagramme de Gantt prévisionnel

La date de rendu initiale du TER était prévue pour le 26 mai, c'est donc à ce moment que finit la tâche du bas "Rapport + Présentation". Bien entendu, les temps assez longs présentés ici, sont calculés pour que nous puissions travailler sur d'autres projets en parallèle de celui-ci. Nous ne sommes pas à temps plein sur ce projet là, donc les temps de travail dans la colonne de

gauche (8h par jour / 5j par semaine) ne sont pas représentatifs de notre réel travail. Cela dit, nous considérons avoir passé environ 2 voire 3 demi-journées en moyenne par semaine sur le projet, soit approximativement 10h par semaine par personne.
Voilà donc comment nous imaginions le planning de notre projet. Cependant, tout le monde le sait, ce trimestre aura été particulier, de par les évènements internationaux. Cela a entraîné bon nombre de changements dans le déroulement du semestre dans son ensemble, et donc dans tous les projets en cours, notamment le TER.

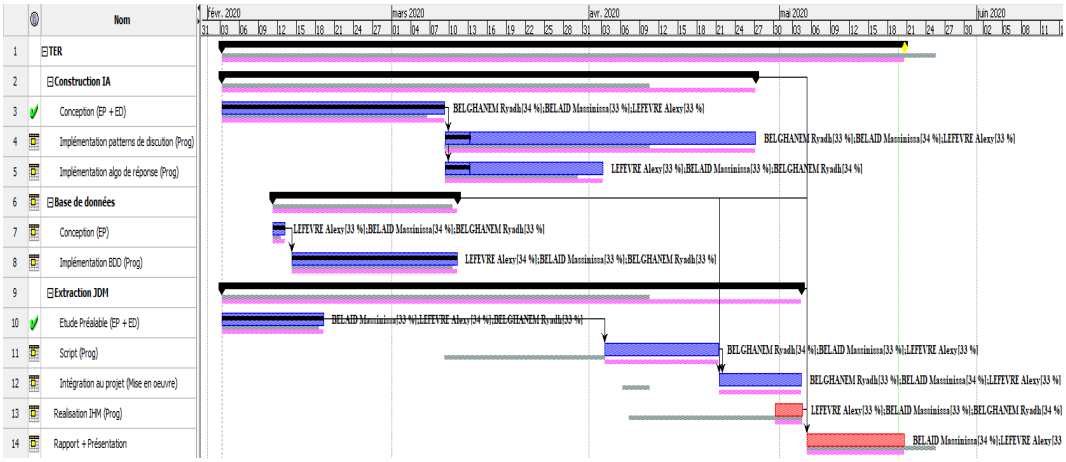

FIGURE 6.2 – Diagramme de Gantt final

On peut donc visualiser dans ce second diagramme, les différences entre le ghost du planning prévisionnel en gris, et celui du planning final en rose. La date de rendu a été avancé au 20 mai. Nous avons mis à jour l'avancement des tâches jusqu'à la date confinement, date à laquelle nous avons revu le planning. Au début nous pensions pouvoir traiter chacun nos parties indépendamment des autres, malheureusement ça n'a pas été le cas. Nous avons beaucoup travaillé en groupe. En effet, durant la période de confinement, nous nous retrouvions sur Discord et nous faisions du télé-travail. Selon nous, le sujet traité nécessitait pas mal de réflexion en réunion, avec des retours en arrière, des choix à faire, le code était souvent au second plan... Nous nous sommes quand même partagés des tâches individuelles, mais de petites tâches et non des tâches qui en englobe bien d'autres, comme celles que l'on peut le voir sur le diagramme. C'est donc pour cela que les ressources sont attribuées à un tiers par personne sur les tâches, nous avons tous, un peu, participé à la réalisation de chaque "grosse" tâche. Nous nous considérions en fait tous les trois comme des acteurs équivalents, des "chefs de projets" qui peuvent tous intervenir dans toute sorte de tâche.

Au moment où nous écrivons le rapport, nous avons environ 1500 lignes de code rien que pour notre programme principal et 42 commits sur GitHub.

FIGURE 6.3 – Fréquence des commits

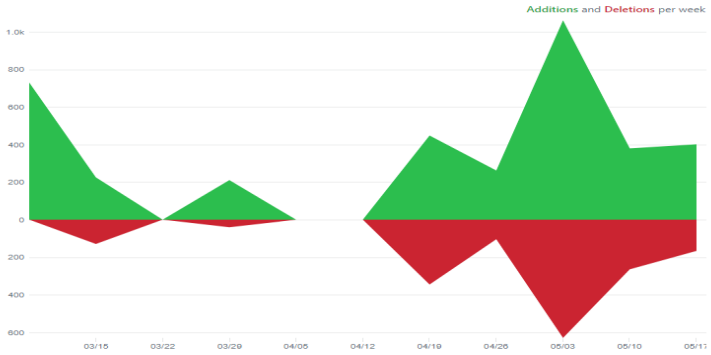

FIGURE 6.4 – Fréquence d'ajout/suppression du code

Comme on peut le constater sur les graphiques, nous avons fourni un travail assez régulier. Nous avions un peu moins de temps à consacrer au TER en début de semestre en raison des autres matières qui prenait beaucoup de temps. Cela a été encore plus vrai au milieu du semestre, notamment début avril ou nous avons dû mettre le TER de côté durant une bonne semaine. C'est en fin de semestre que le plus gros du code a été fourni, lorsque l'on avait les idées bien en place sur ce qu'on voulait réaliser, et un peu plus de temps à y consacrer.

Chapitre 7

Conclusion

En conclusion au travail fourni durant tout ce semestre, nous sommes heureux de ce que nous avons produit. Le programme est fonctionnel et répond aux attentes fixées au départ. Greg est capable de répondre à des questions de connaissance selon des dizaines de patterns. Il peut interpréter les réponses de ses interlocuteurs et modifier ses connaissances. Bien entendu, il n'est pas parfait, et nous pourrions l'améliorer quasiment à l'infini si la contrainte de temps n'existait pas.

Ce projet réalisé en collaboration avec nos encadrants, Mme. Prince et M. Lafourcade, on l'espère, pourra être repris et utilisé à des fins utiles à la recherche. Nous n'excluons pas la possibilité de continuer à travailler et développer Greg durant les prochaines vacances estivales.

Bibliographie

[1] Mathieu Lafourcade,
http://www.jeuxdemots.org/jdm-accueil.php

[2] Traitement automatique des langues,
https://fr.wikipedia.org/wiki/Traitement_automatique_des_langues

[3] Open Classrooms cours django,
https://openclassrooms.com/fr/courses/4425076-decouvrez-le-framework-django/4630701-tirez-parti-de-ce-cours

[4] Wikipédia Python,
https://fr.wikipedia.org/wiki/Python_(langage)

[5] Open Classrooms cours SQL, https://openclassrooms.com/fr/courses/1959476-administrez-vos-bases-de-donnees-avec-mysql/1959710-decouvrez-mysql