IUT Dijon
Dijon-Auxerre

2017-2018

RAPPORT DE PROJET

PROJET CHATBOT

ALEXIS GUYOT – EMMANUEL DEREPAS – DORIAN NAAJI – RAPHAËL CORDEROT – LELIAN RUFFIN - VALENTYN NOËL – BASTIEN GRENIER

PROJET ENCADRE PAR LE PROFESSEUR CHRISTOPHE NICOLLE

Table des matières

Introduction

Au cours de leur troisième semestre, les élèves de seconde année de DUT Informatique ont la possibilité d'effectuer un projet tutoré d'envergure professionnelle, par groupe de 4 à 7, et encadrés par un tuteur.

Dans ce contexte-là, notre groupe, constitué d'Emmanuel DEREPAS, de Dorian NAAJI, d'Alexis GUYOT, de Raphaël CORDEROT, de Bastien GRENIER, de Lélian RUFFIN et de Valentyn NOEL, s'est formé. Nous nous sommes rassemblés par affinités, mais aussi parce que nous avions presque tous eu l'occasion de travailler ensemble par le passé dans de très bonnes conditions.

Nous avons décidé de choisir le projet nommé « Chatbot » proposé par le professeur Nicolle. Ce choix était motivé par notre envie à tous d'en savoir plus sur les intelligences artificielles, mais également sur les robots comme Siri, Cortana ou l'assistant Google, qui sont aujourd'hui une part de plus en plus importante de nos vies. De plus, Lélian RUFFIN et Valentyn NOEL étaient intéressés par le développement web, un aspect de l'informatique qui allait être traité dans ce projet. Pour toutes ces raisons, nous avons choisi le chatbot.

I. En quoi consiste le projet ?

1. Les besoins

Aujourd'hui, M. Nicolle possède une version d'un chatbot très peu performant. Il souhaite obtenir une version plus évoluée et fonctionnelle sur laquelle ses ingénieurs pourraient par la suite travailler et qu'ils pourraient approfondir.

2. Les objectifs à réaliser

Nous sommes tout d'abord partis sur une amélioration du chatbot originel qui n'était fonctionnel que sous certaines conditions. Dans l'intitulé du second livrable, nous nous étions fixés comme objectif d'obtenir un chatbot capable de déterminer la nature des différents mots dans une phrase préalablement tapée par l'utilisateur. Il devait renvoyer les définitions ainsi que les synonymes et traductions dans plusieurs langues. Nous nous sommes vite rendus compte que les objectifs n'étaient pas assez poussés. Nous avons donc créé une liste de « Competency questions » qui était adaptée à la charge de travail que nous pouvions traiter

Rédaction des « competency questions »

Dans le monde de l'intelligence artificielle, on a l'habitude de représenter les capacités et habilités d'un robot à partir d'une liste de questions appelées « Competency questions ». Il s'agit d'un questionnement de type « Que peut faire notre chatbot ? », qui résume les fonctionnalités attendues de la part de notre bot.

Lors du troisième cycle, nous avions pour consigne de rédiger cette liste pour nous permettre de communiquer rapidement et clairement à des personnes extérieures à notre projet ce dont est capable notre bot.

Voici le rendu final de notre liste de « competency questions »

Le chatbot sait-il **saluer** l'utilisateur ?
Le chatbot sait-il donner la **définition** d'un mot sans mise en forme ?
Le chatbot sait-il remonter les **traductions** d'un mot sans mise en forme ?
Le chatbot sait-il remonter les **synonymes** d'un mot sans mise en forme ?
Le chatbot sait-il donner les **mots apparentés** du mot sans mise en forme ?
Le chatbot sait-il répondre aux tâches précédentes avec une mise en forme quelconque ?
Le chatbot est-il capable de mettre en forme les traductions d'un mot sous forme d'un tableau à 2 colonnes : - Colonne 1 : **Mot traduit** - Colonne 2 : **Langue** + éventuellement un **drapeau**
Le chatbot est-il capable de mettre en forme les **synonymes** et **mots apparentés** d'un mot sous forme d'une **carte mentale** ?
Le chatbot est-il capable de restituer **toutes les informations simultanément** concernant un mot : définition, traduction(s), synonymes, mots apparentés
Le chatbot est-il capable de restituer les informations concernant un mot en comprenant le **type d'information** que veut l'utilisateur ? (Définition si l'utilisateur demande une définition, synonyme(s) si l'utilisateur demande des synonymes, etc.)
Le chatbot sait-il répondre à une question de **quantité** sans mise en forme particulière ? (type : Combien de fois la terre tourne-t-elle sur elle même pour une durée de 1 an)
Le chatbot sait-il répondre à une question de **quantité** avec mise en forme particulière de type camembert ou histogramme

3. Pourquoi Rasa ?

Après nos recherches, l'une des API ayant retenu notre attention est RASA NLU, que nous appellerons RASA. C'est un outil de traitement automatique du langage naturel (NLP - Natural Language Processing), utile pour analyser et comprendre une phrase, ouvert (open-source) et donc gratuit, utilisant le langage Python.

L'API Rasa est totalement adaptée à notre problème pour plusieurs raisons. Tout d'abord, rasa est **open source,** ce qui permet par la suite de pouvoir commercialiser l'application. L'API est totalement **gratuite** et est développée par 14 ingénieurs/chercheurs au MIT, ce qui nous accorde une certaine garantie concernant son efficacité. Une **documentation très complète** accompagnée d'exemples est disponible pour aider les nouveaux utilisateurs à comprendre le fonctionnement du programme, ce qui en a fait un candidat très intéressant pour des étudiants ne possédant pas de formation particulière dans le domaine.

II. Réalisation du projet

1. Contenu des différents cycles

Encore une fois dans un souci de respect de la méthode agile, la partie réalisation du projet a été découpée en 6 cycles de 2 ou 3 semaines chacun :

- **Cycle 1 :** Du 21 septembre au 4 octobre 2017
- **Cycle 2 :** Du 4 au 18 octobre 2017
- **Cycle 3 :** Du 18 octobre au 8 novembre 2017
- **Cycle 4 :** Du 8 au 29 novembre 2017
- **Cycle 5 :** Du 29 novembre au 20 décembre 2017
- **Cycle 6 :** Du 20 décembre 2017 au 8 janvier 2018

Nous allons dans cette partie décrire précisément le contenu de ceux-ci.

Analyse et Conception

La première phase du projet tutoré, période s'étalant sur les deux premiers cycles, avait pour but d'assimiler les objectifs du projet ainsi que de rédiger trois dossiers d'analyse et de conception : Un état de l'art, un dossier UML et un dossier IHM.

C'est également pendant cette période que nous avons décidé de choisir l'API Rasa afin de nous aider dans le traitement et la compréhension des questions posées à notre chatbot. Pour en savoir plus sur cette phase, nous vous invitons à lire les différents rapports rédigés à l'issue de celle-ci.

Recherches et familiarisation avec les technologies à utiliser

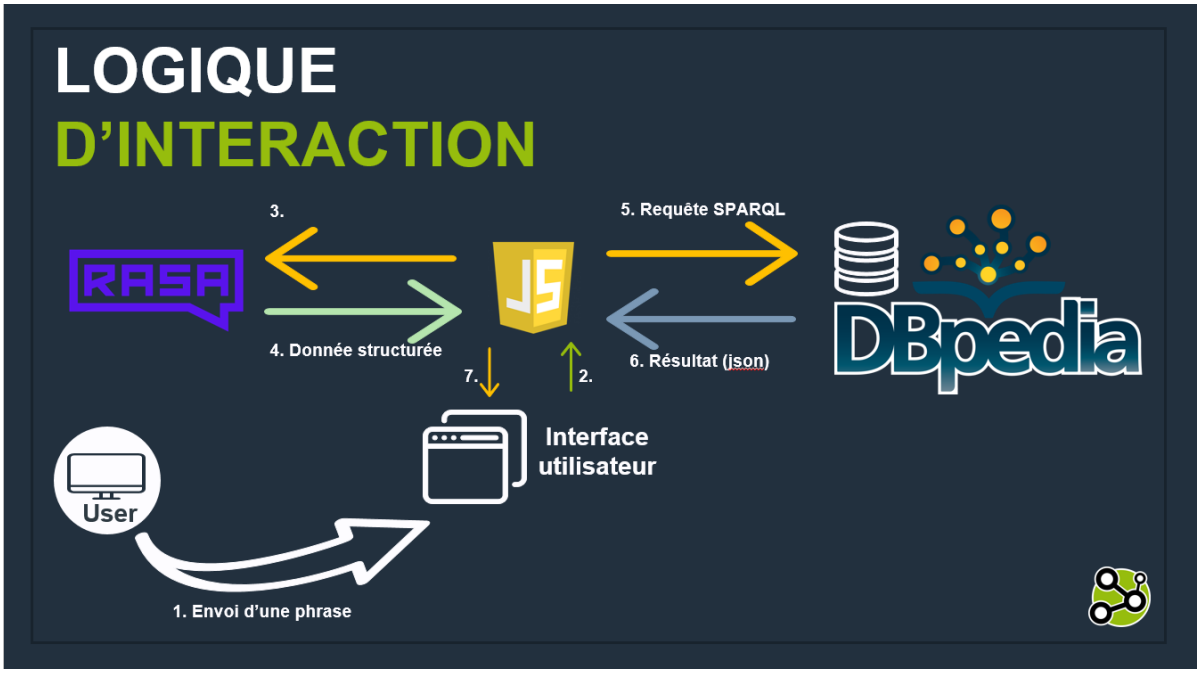
Après la phase d'analyse/conception et avant de se lancer dans la réalisation technique et l'implémentation « tête baissée » des fonctionnalités demandées, M. Nicolle nous a conseillé un certain nombre de technologies utiles dans le contexte de développement d'applications telles qu'un chatbot.

Parmi elles, nous retrouvons **Swagger** pour la documentation, **Angular.js** pour la liaison entre le client et le serveur, **NGinx** à utiliser comme serveur web, **Node.js** pour l'exécution du javascript côté serveur, **SPARQL** pour le requêtage vers une base de connaissance[1] distante (ici DBpedia), **DBpedia** pour en savoir plus sur ladite base et **D3.js** pour l'affichage des résultats.

Chaque membre du groupe a donc eu comme tâche pour le premier cycle de la phase de réalisation de rédiger un court descriptif d'une des 7 technologies pour en présenter aux autres l'utilité et/ou le fonctionnement vis-à-vis de notre projet.

[1] Ensemble structuré de termes et de concepts représentant les éléments d'un domaine de connaissances.

A partir des recherches précédemment effectuées, nous avons pu schématiser le fonctionnement de notre chatbot de la manière suivante.



Une fois ce schéma obtenu, nous n'avions plus qu'à faire toutes les installations nécessaires pour les différentes technologies pour pouvoir commencer le développement (voir ci-après).

Mise en place du serveur

La mise en place du serveur a été une tache très importante. Nous avons tout d'abord commencé par regarder vers un serveur que nous pouvions héberger par nous-même. Pour de multiples raisons comme la qualité de la connexion ou une peur de manque de mémoire et de puissance, nous avons opté pour un serveur hébergé par un hébergeur en ligne.

Notre chef de projet louait un serveur pour héberger son site web, nous nous sommes donc dit que nous pourrions créer le chatbot sur celui-ci.

Il existe au moins deux types d'hébergement :

- Le cloud sharing : consiste à louer une partie d'une machine dédiée sans donner les droits administrateurs (pour des raisons de sécurité)
- Le serveur dédié : consiste à louer la totalité d'une machine en procurant les droits d'administrateur

Il s'est avéré qu'après avoir contacté l'hébergeur, nous nous sommes rendu compte qu'il était impossible d'installer par nous-mêmes rasa ou autres programmes que l'hébergeur n'avait pas pré-configurés.

Nous avons donc décidé de louer un serveur dédié pour avoir les droits de manipuler rasa et autres.

Lors du quatrième cycle, toutes nos recherches ont été mises à profit lors de l'implémentation sur notre serveur des technologies demandées. Ainsi, et d'un point de vue technique cette fois-ci, nous avons mis en commun toutes les connaissances que nous avions accumulées pour décider de comment mettre en relation de manière pertinente vis-à-vis du projet les différentes technologies conseillées par notre tuteur.

Nous avons alors passé notre serveur sous **NGinx**, avant d'installer **Node.js** sur celui-ci. Une fois cela fait, nous avons recherché les requêtes **SPARQL** dont nous allions avoir besoin, puis nous avons trouvé un moyen de les envoyer depuis **Javascript** et **Node.js**. Après cela, il ne restait plus qu'à mettre en forme les résultats grâce à **D3.js**.

Puisque sans un système de requêtage qui fonctionne, il s'avère compliqué d'afficher un quelconque résultat, nous avons utilisé quelques exemples fournis par Virtuoso, un outil en ligne permettant d'effectuer des requêtes SPARQL à DBPedia et de récupérer les résultats sous différentes formes. Nous avons également testé les requêtes à l'aide de cet outil.

Requêtage de DBpedia

A partir de ce moment-là, l'équipe du projet s'est scindée en deux sous-équipes : Une travaillant sur ce qu'on pourrait appeler le « back-end », c'est-à-dire tout ce qui est nécessaire au fonctionnement de l'application mais que l'utilisateur ne voit pas (traitement de la phrase avec rasa, requêtage vers DBpedia, etc), et une autre sur le « front-end », qui est donc l'inverse (ce que voit l'utilisateur, soit l'IHM et la documentation).

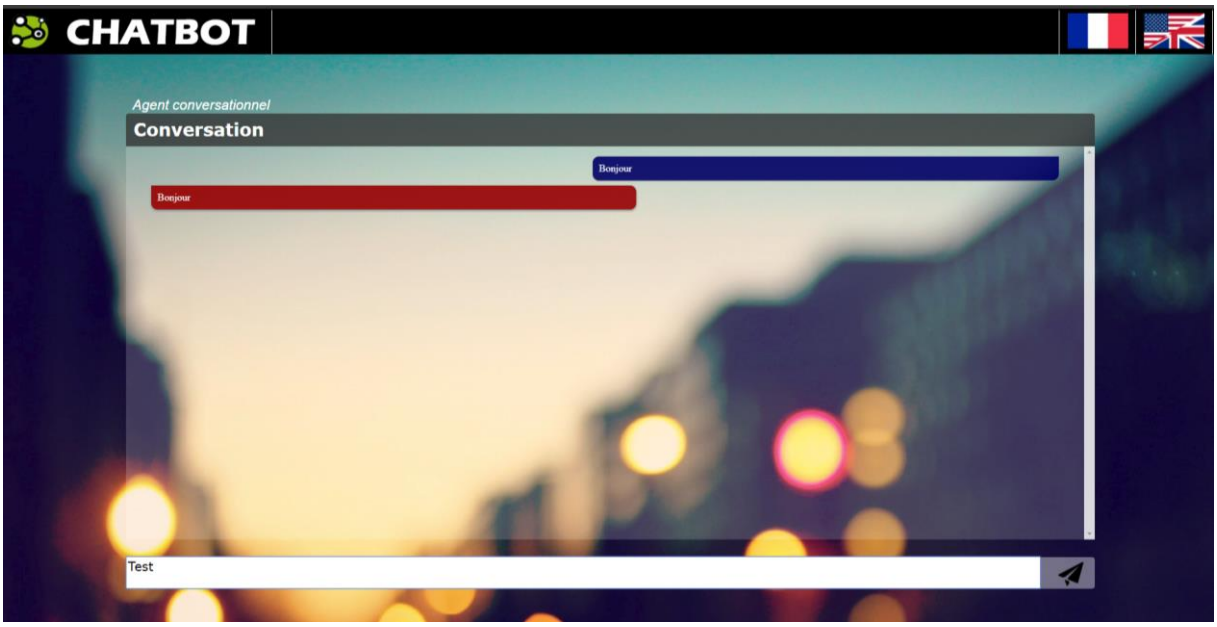
Comme indiqué précédemment, le travail de requêtage SPARQL vers DBPedia a commencé sur l'outil Virtuoso. Sur celui-ci, nous avons construit un ensemble de requêtes qui retournent exactement ce qui était demandé dans les consignes, à savoir pour un mot donné une définition, des traductions, des synonymes et des mots apparentés.

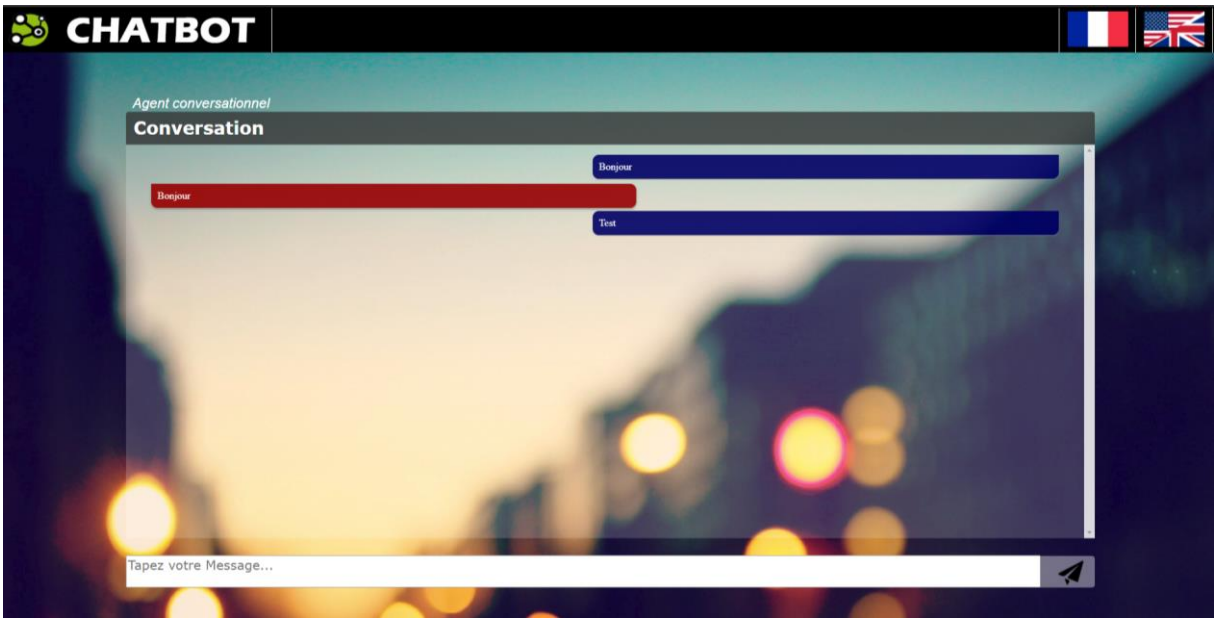
A partir du cycle 5, nous avons cherché à envoyer ces requêtes à DBPedia à travers notre propre serveur, grâce au Javascript. Une fois cette fonctionnalité faite, nous avons directement appliqué D3.js sur les résultats pour travailler rapidement sur la mise en forme.

C'est également à ce moment-là que certaines difficultés sont apparues, notamment au niveau de la partie française de DBPedia qui s'avère assez pauvre en documentation à côté de son homologue anglophone. Un dernier travail a donc été effectué pendant le sixième cycle afin de résoudre les problèmes restants. Les difficultés rencontrées et les solutions trouvées seront traitées plus en détails dans la partie « Difficultés rencontrées et solutions » ci-après.

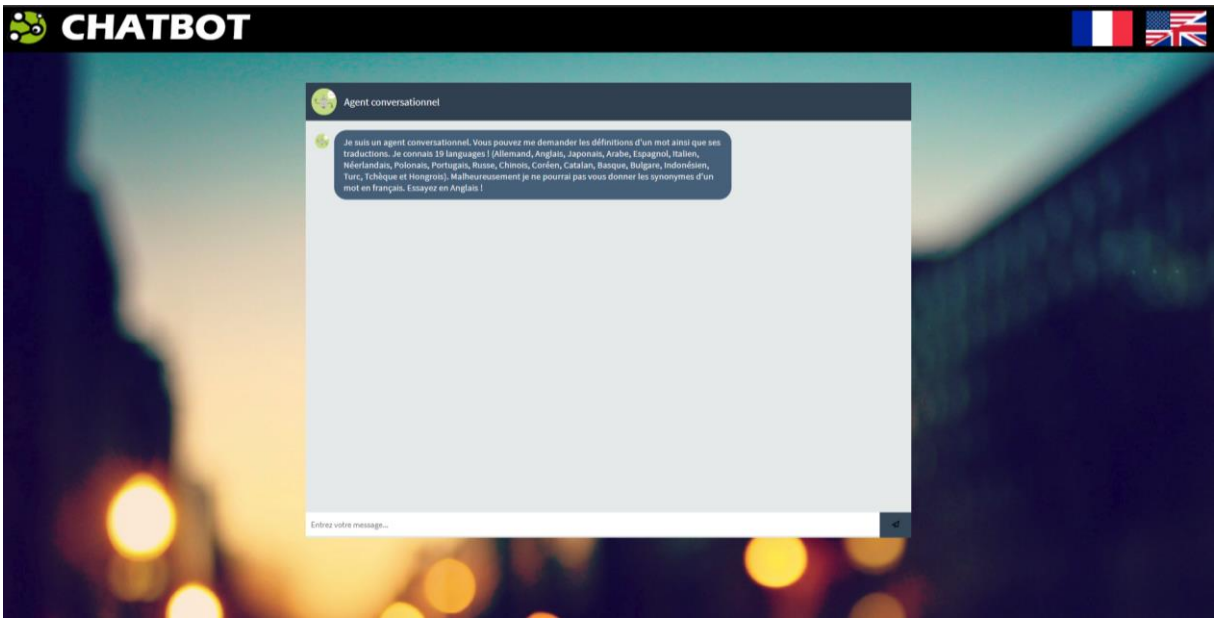
Création de l'interface

Le premier travail de l'équipe « front-end » a donc été d'implémenter en HTML/CSS les maquettes IHM fournies lors de la phase de conception, et de produire un système de tchat qui fonctionne. Ainsi, à la fin du cinquième cycle, ceux-ci étaient en mesure de fournir ce résultat :





Cependant, suite à la mise en relation des fonctionnalités, cette interface a évolué pour mieux correspondre à nos besoins. A la fin du dernier cycle, l'interface de notre chatbot ressemblait à cela :



Traitement du langage par Rasa NLU

RASA NLU est une composante de l'API Rasa Stack. C'est un outil open-source codé en python permettant de classifier des intentions et extraire des entités. Par exemple :

Transformer une phrase :

```
"I am looking for a Mexican restaurant in the center of town"
```

en :

```
{
  "intent": "search_restaurant",
  "entities": {
    "cuisine" : "Mexican",
    "location" : "center"
  }
}
```

Ce genre de fichier au format JSON est utile pour les développeurs d'agents conversationnels, notamment pour l'affichage avec D3.js.

Dans notre cas, nous avons défini grâce à l'API des entités et intentions afin de récupérer des données structurées relatives à plusieurs cas : la recherche de définitions, la recherche de synonymes et la recherche de traductions, par le biais du chatbot. Pour ce faire, nous avons mis en place des exemples de phrases en rapport avec ces différents cas, comme par exemple « Qu'est-ce qu'un … ? » ou « Quelle est la définition de … ? » ou même « Comment dit-on … en … ? ». A partir de cela, l'API, via son fonctionnement, comprend si une phrase appartient à telle ou telle intention.

Le traitement nécessaire est ensuite fait en javascript. Ces intentions permettent à notre bot d'entrer dans des états (état « définition », état « traduction », etc.), puis de décider de la procédure à suivre lorsque le bot entre dans l'état en question.

Durant le dernier cycle, l'API RASA NLU a été intégrée à notre interface web.

La documentation a été un des points les plus délicats de notre projet. Nous sommes d'abord partis sur une optique de tout documenter grâce à Swagger. Après étude et essais avec notre projet, nous nous sommes rendus compte que l'API n'était pas vraiment très adaptée dans notre cas. Nous avons donc décidé de commenter et de documenter très précisément tous les fichiers de code et de fournir des fichiers « readme » très régulièrement pour expliquer des concepts ou des API.

De plus, afin d'aider les ingénieurs qui travailleront par la suite à partir de notre travail, nous avons décidé de quand même utiliser Swagger pour fournir la documentation de Rasa, notre API de traitement du langage. De ce fait, ils pourront se familiariser plus facilement et efficacement à ce module du chatbot.

Mise en relation des fonctionnalités

Comme expliqué précédemment, l'équipe s'est séparée en deux sous-équipes travaillant sur des fonctionnalités différentes pour à la fin arriver au résultat attendu. L'enjeu du dernier cycle était donc évidemment de tout mettre en commun et de s'assurer du fonctionnement du projet une fois tous les modules rassemblés.

La première étape était donc de rassembler le travail fait sur Rasa NLU (traitement du langage, pour analyser les phrases fournies) avec le travail fait sur le requêtage SPARQL vers DBPedia. L'objectif était donc de pouvoir poser directement une question dans une barre de recherche et d'obtenir une réponse précise, plutôt que de rentrer juste un mot et de tout recevoir comme c'était le cas jusqu'à ce moment-là.

Une fois cela fait, nous nous sommes rendus compte qu'une interface plus proche d'une fenêtre de discussion aurait été plus adaptée pour notre chatbot. Nous l'avons donc légèrement modifiée par rapport au design original pour plus aller dans ce sens. Une fois cela fait, nous avons ajouté les fichiers de « back-end » à l'interface, et notre chatbot était maintenant fonctionnel.

Déploiement

Pour le déploiement, M. Nicolle nous a donné comme consigne d'utiliser **Docker**, une API open source facilitant cette démarche et surtout la portabilité future de l'application. Il a donc fallu dans un premier temps se renseigner sur la technologie, et dégager un tutoriel afin de pouvoir déployer rapidement notre chatbot.

Le déploiement s'avère relativement compliqué notamment par rapport au proxy qui a dû être mis en place ainsi qu'à l'API rasa qui est modifiée. Raphaël CORDEROT a travaillé avec un des ingénieurs représentant Docker France pour manipuler et pouvoir déployer RASA le plus facilement possible.

Tout d'abord, il a fallu comprendre le système d'image et de container (similaire à un ISO pour une machine virtuelle). Il a fallu ensuite comprendre l'utilisation et la pertinence des données sur le serveur sur lequel nous voulons déployer le chatbot.

Le premier problème a été de réussir à lister toutes les "bibliothèques logicielles" utiles pour le fonctionnement de rasa, car rappelons que l'objectif est que le déploiement soit le plus simple et rapide possible. Il a donc fallu que les personnes qui travaillaient sur le "back-end" listent ce dont ils avaient besoin pour que rasa fonctionne.

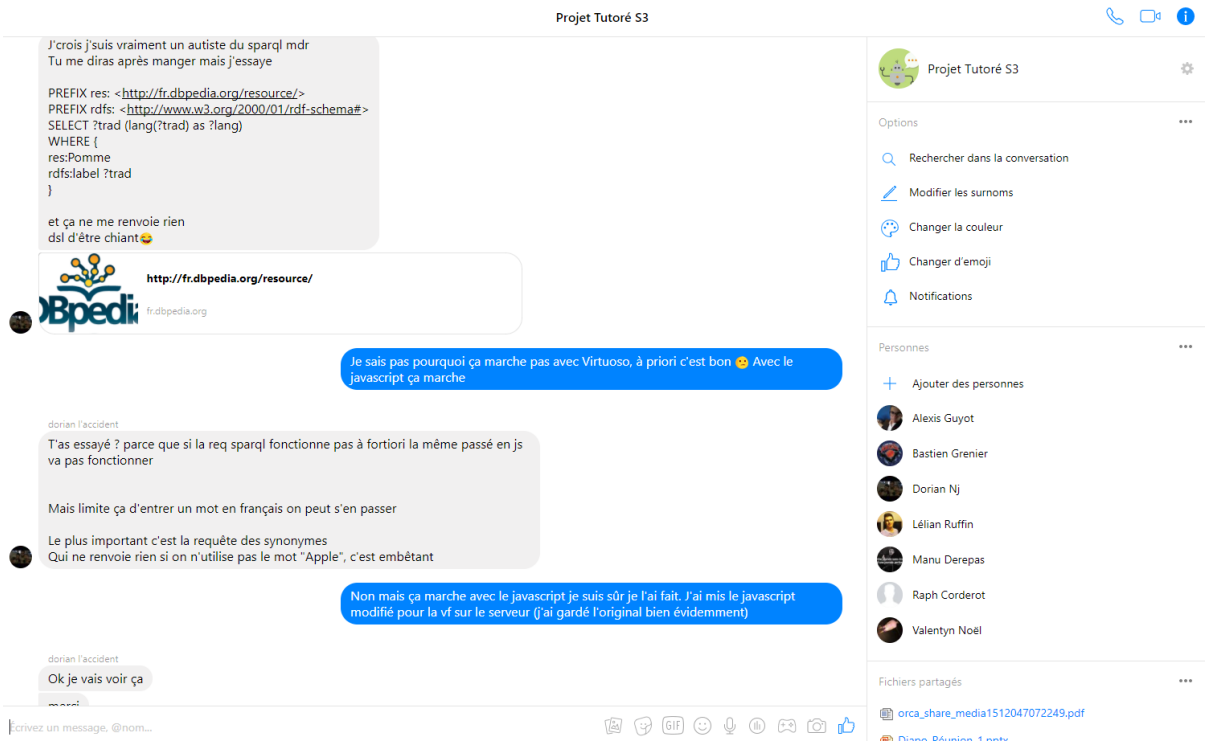
Le second problème, celui qui nécessite l'aide d'un représentant de docker, est le fait que rasa n'est pas ou du moins très difficilement déployable. Pour des raisons qui sont encore obscures, rasa ne peut pas être ajoutée à un container contrairement aux autres applications.

Pour assurer nos arrières, nous avons décidé en parallèle d'exporter notre chatbot sous la forme d'une archive accompagnée d'un tutoriel d'installation. Celui-ci se trouve dans l'archive en question dans le fichier « setting-up-the-chatbot.md » ou ci-dessous.

2. Organisation

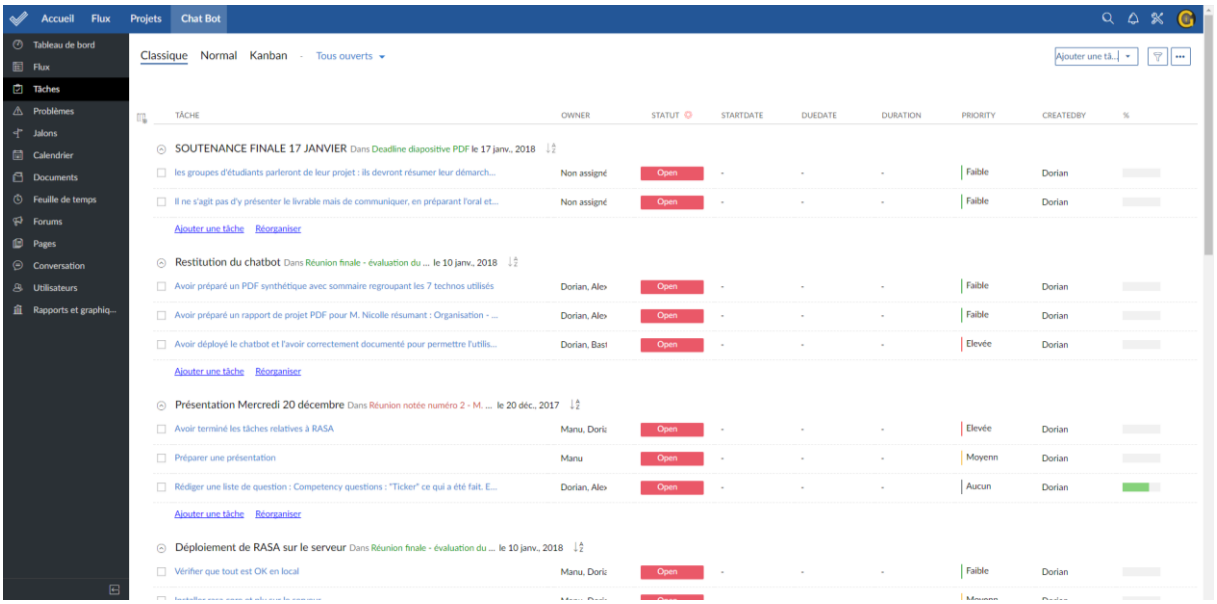
Outils utilisés

Nous avons décidé d'utiliser plusieurs outils pour nous aider et pour nous faciliter la vie pendant le déroulement du projet. Afin de communiquer quand nous n'étions pas ensemble, nous passions la plupart du temps par le service de messagerie instantanée **Messenger**, car nous le possédions tous et qu'il était assez facile et naturel de communiquer grâce à celui-ci.



Pour nos réunions, nous sommes passés par le logiciel **Discord**, une variante à Skype et à Teamspeak très pratique pour le travail de groupe.

Pour l'organisation du projet en agile, nous avons utilisé le site internet **zoho project**, celui-ci nous permettant de gérer très efficacement le système de to-do list.



Il était important pour nous d'utiliser un outil en ligne pour gérer l'organisation de notre projet puisque nous avons beaucoup travaillé chacun de notre côté. Un tel outil nous a ainsi facilité la synchronisation à l'intérieur du groupe, Messenger restant trop peu pratique pour vraiment s'organiser et répartir les tâches.

Enfin, nous avons créé un dossier sur **Dropbox** afin de se partager facilement les fichiers et dossiers créés tout au long du projet.

0. Liens - supports - autre	03/10/2017 22:30	Dossier de fichiers	
1. Suivi du projet	15/12/2017 19:55	Dossier de fichiers	
1.1 Réunions	15/12/2017 19:57	Dossier de fichiers	
2. Étude de marché	09/10/2017 23:11	Dossier de fichiers	
3. Livrable 1 - dossier rédigé	06/11/2017 14:48	Dossier de fichiers	
4. Livrable 2 - dossier d'analyse	06/11/2017 14:48	Dossier de fichiers	
4.1 CM co	26/11/2017 16:21	Dossier de fichiers	
5. Livrable 3 - Maquette IHM	06/11/2017 14:48	Dossier de fichiers	
6. Phase 1 PTS3 NAAJI GUYOT RUFFIN CORDEROT NOEL GRENIE...	26/11/2017 16:21	Dossier de fichiers	
7. Documentation	26/11/2017 16:21	Dossier de fichiers	
8. RASA Comment ça marche	06/11/2017 23:55	Dossier de fichiers	
9. Tout savoir sur SPARQL	26/12/2017 17:33	Dossier de fichiers	
10. Serveur NGINX	26/11/2017 16:21	Dossier de fichiers	
11. Sources d3.js (à exécuter en serv local)	26/11/2017 16:21	Dossier de fichiers	
12. Présentation de mercredi - réunion NICOLLE	07/01/2018 17:31	Dossier de fichiers	
13. Site Web	19/12/2017 21:05	Dossier de fichiers	
14. chatbot	19/12/2017 21:05	Dossier de fichiers	
16. Présentation réunion Chatbot 20-12 à compléter	03/01/2018 11:10	Dossier de fichiers	
17. Dossier chatbotproject du site (pour la doc Bastien)	26/12/2017 17:33	Dossier de fichiers	
18. TODO - DOCUMENTATION	03/01/2018 11:07	Dossier de fichiers	
19. Diapo pour la réunion du 8..12-01-2018	03/01/2018 11:10	Dossier de fichiers	
Nouveau dossier	31/12/2017 11:04	Dossier de fichiers	
.dropbox	21/09/2017 18:08	Fichier DROPBOX	1 Ko
14. Présentation réunion Chatbot.pdf	04/12/2017 21:41	Chrome HTML Do...	3 791 Ko
15. Competency questions - exemple de ROBERTO en CM.pdf	06/12/2017 14:54	Chrome HTML Do...	308 Ko
desktop.ini	21/09/2017 18:08	Paramètres de co...	1 Ko
Rasa Core.docx	23/12/2017 23:55	Document Micros...	16 Ko

Chaque cycle du projet avait une durée moyenne de 3 semaines et était délimité par une réunion avec le tuteur durant laquelle nous présentions le travail effectué pendant la période précédente ainsi que les éventuels problèmes rencontrés, et où nous parlions des consignes pour la période d'après.

Chaque réunion avec le tuteur était suivie d'une réunion avec toute l'équipe pour parler de la répartition des tâches. Celles-ci avaient lieu soit dans un endroit physique, typiquement à l'intérieur de l'IUT, soit sur Discord.

Tout au long du cycle, le chef de projet (Emmanuel DEREPAS) devait s'assurer que tout le monde avançait et que personne n'avait de problème. Nous communiquions également régulièrement par Messenger pour tous se tenir au courant de notre avancement, et éventuellement pour demander de l'aide.

En cas de problème vraiment bloquant, un membre de l'équipe se chargeait d'aller demander l'avis du tuteur. Il s'agissait bien souvent du premier qui avait la possibilité de le voir en cours de Génie Logiciel ou dans les couloirs.

Répartition des tâches - Organisation

Emmanuel DEREPAS était chef de projet. Il s'occupait de distribuer les tâches, de communiquer avec le tuteur et de s'assurer que tout le monde faisait bien ce qu'il avait à faire.

A la fin de chaque réunion avec le tuteur, Alexis GUYOT, qui possède un meilleur niveau à l'écrit qu'Emmanuel DEREPAS, était en charge de rédiger un compte-rendu pour résumer le travail qui était à faire, les remarques du tuteur et le travail à faire pendant le prochain cycle à partir des notes qu'il devait prendre.

Dorian NAAJI était responsable du Zoho Project. Il devait le mettre à jour régulièrement et répartir les tâches aux différents membres du groupe avec l'aide d'Emmanuel DEREPAS.

Répartition des tâches – Phase 1 – Analyse et Conception

Le travail lors de la phase 1 du projet tutoré s'est déroulé en deux temps.

D'abord, nous avons tous travaillé sur le livrable 1, qui correspondait au dossier rédigé contenant notamment la présentation du projet, l'état de l'art et la présentation des technologies choisies. Chaque membre du groupe devait travailler sur un point du plan qui lui était attribué (définition d'un chatbot, historique, acteurs importants, …). Alexis GUYOT était en charge de mettre en commun le travail de tout le monde, d'harmoniser le tout et de corriger les fautes d'orthographe.

Dans un second temps, nous nous sommes séparés en deux sous-groupes : un s'occupant de la partie conception UML, l'autre s'occupant de la partie IHM.

Dans le sous-groupe UML, Alexis GUYOT s'est chargé des diagrammes de séquences, Dorian NAAJI du diagramme de cas d'utilisation et Raphaël CORDEROT du diagramme de classes. Emmanuel DEREPAS, en tant que chef de projet, devait vérifier la justesse et la cohérence des différents diagrammes et devait retravailler avec les trois personnes précédentes sur les premières tentatives pour les affiner et les améliorer. Dorian NAAJI était en charge de mettre en commun les travaux de chacun et d'exporter le tout en format Word. Alexis GUYOT devait s'assurer de l'orthographe du document.

Dans le sous-groupe IHM, Lélian RUFFIN, Valentyn NOEL et Bastien GRENIER devaient se concerter pour créer une maquette à l'aide d'Illustrator et de Photoshop. A partir du résultat obtenu, ils devaient ensuite imaginer ce à quoi allait ressembler l'interface dans différents cas d'utilisation. Alexis GUYOT s'est chargé de confectionner les diagrammes de tâches et a corrigé les fautes d'orthographes après qu'Emmanuel DEREPAS se soit chargé de tout vérifier et de tout mettre en commun.

Répartition des tâches – Phase 2 – Réalisation du projet

Notre façon de travailler pour la deuxième phase reste au final assez proche de celle de la première phase : D'abord ensemble pour les réalisations générales sur le projet puis en deux sous-équipes, une sur le développement « back-end » (ce que l'utilisateur ne voit pas), et une sur le développement « front-end » (ce que voit l'utilisateur).

Dans un premier temps donc, chaque membre s'est vu attribuer une technologie possiblement utile dans le cadre de notre projet. Le but était d'en comprendre le fonctionnement, l'utilité et de voir comment nous aurions pu l'utiliser dans notre chatbot (voir ici pour plus de détails, notamment sur les technologies). Après avoir effectué ses recherches, chacun devait rédiger quelques pages de présentation à destination des autres.

Emmanuel DEREPAS devait travailler sur **NGinx**, Valentyn NOEL sur **Angular.js**, Dorian NAAJI sur **D3.js**, Lélian RUFFIN sur **DBPedia**, Raphaël CORDEROT sur **Node.js**, Alexis GUYOT sur **SPARQL** et Bastien GRENIER sur **Swagger**.

Une fois toutes les technologies assimilées et l'architecture globale de notre chatbot terminée, l'équipe s'est de nouveau scindée en deux, comme annoncé ci-dessus.

Pour la sous-équipe « back-end », Emmanuel DEREPAS a choisi d'étudier et d'essayer de comprendre le fonctionnement de Rasa Core (un complément à NLU qui simplifierait le back-end en nous permettant de mettre de côté une partie du javascript). Dorian NAAJI était en charge de Rasa NLU, la partie traitement et compréhension de la phrase proposée par l'API Rasa. Alexis GUYOT devait trouver les bonnes requêtes SPARQL pour questionner la base de connaissances DBpedia, en français et en anglais.

Pour la sous-équipe « front-end », Lélian RUFFIN et Valentyn NOEL devaient s'occuper de la programmation de l'interface web et du système de « tchat ». Bastien GRENIER devait travailler sur Swagger et sur la documentation du projet. C'est également lui qui devait trouver un design pour les diaporamas avant chaque présentation.

Raphaël CORDEROT, quant à lui, était en charge du serveur web. C'est lui qui devait en chercher un qui nous convenait, le prendre et ensuite faire les installations nécessaires (nginx, node.js, …). Par la suite, il a également travaillé sur le déploiement du site web grâce à Docker.

Enfin, Alexis GUYOT s'est chargé de rédiger le rapport final de projet, avec l'aide de Raphaël CORDEROT.

Pour la répartition des tâches lors de la deuxième phase, nous avons essayé d'être cohérents avec ce sur quoi chacun avait travaillé lors de la phase précédente, mais aussi avec les affinités de chaque membre par rapport aux différentes tâches, le but étant que chacun puisse travailler dans les meilleures conditions possibles.

3. Difficultés rencontrées et solutions

D'abord, d'un point de vue organisationnel, aucune grosse difficulté n'a été rencontrée. En effet, la communication au sein de l'équipe et avec le tuteur a été très bonne tout au long du projet, ce qui a facilité la mise en œuvre de la méthode agile. De plus, celle-ci ayant été abordée très tôt en cours cette année, nous avons pu rapidement choisir et mettre en place les quelques outils utiles à son déroulement (le système de to-do list notamment).

Le contenu de la phase 1 n'a pas vraiment posé problème non plus puisque les livrables demandés étaient globalement les mêmes que ceux travaillés lors du dernier projet tutoré (celui de S2), donc pas grand-chose de nouveau de ce côté-là. La seule difficulté était à l'époque de bien comprendre et visualiser ce qu'on voulait de nous. En effet, nous n'avions à l'époque que très peu d'indications ni d'idées sur « comment » nous allions procéder, donc comprendre exactement les attentes et les contraintes était un point indispensable pour réussir. Il a donc fallu plusieurs réunions et de nombreuses questions au tuteur avant de surpasser cette première difficulté.

Du côté de la réalisation du projet, plus de difficultés sont à noter.

D'abord, nous pouvons parler du requêtage vers DBpedia pour obtenir une définition en français. En effet, l'ontologie française contenant l'ensemble des définitions, accessible à travers un préfixe défini par DBpedia, semblait inaccessible depuis une requête SPARQL. Pour résoudre ce problème, nous avons d'abord décidé de contourner le système en passant par deux étapes au lieu d'une : Dans un premier temps, le mot entré en français par l'utilisateur est traduit en anglais grâce à la même requête que celle utilisée pour les traductions, puis une recherche de définition est faite avec le résultat sur le DBPedia anglais. Grâce à un filtre SPARQL, seule la version française de la traduction était gardée pour l'afficher ensuite.

Mais après avoir mis en place cette méthode, nous nous sommes rendus compte que pour de nombreux mots un nouveau problème apparaissait : La traduction anglaise pouvait être en deux mots, ce qui provoquait un tas d'erreurs. Nous avons alors de nouveau étudié DBpedia, et cette fois-ci plus en détail l'ontologie française. A la fin de cette étude, nous avons réussi

Nous avons ensuite rencontré un nouveau problème avec Swagger, une API censée rendre la documentation plus interactive et plus facile à mettre à jour. Après plusieurs essais pour documenter notre chatbot à partir de cette technologie, nous nous sommes rendus compte que notre projet n'était pas vraiment adapté à son fonctionnement. Nous avons alors pris la décision de juste documenter précisément tous nos fichiers et dossiers grâce à plusieurs fichiers « README » et grâce à de nombreux commentaires afin de faciliter la prise en main du code du chatbot.

Enfin, nous avons également éprouvé certaines difficultés au moment du déploiement de notre chatbot. En effet, l'utilisation de Rasa nécessite un passage par un proxy et par une redirection de port. Or, il était impossible pour nous de « Dockeriser » (utiliser l'API Docker) un fichier qui modifiait le .conf du serveur. De plus, le chatbot avait besoin pour fonctionner d'une version installée de rasa, impossible à ajouter dans un fichier Docker. Pour régler ce problème, nous avons pris la décision d'archiver le dossier contenant notre chatbot, et de fournir cette archive avec un fichier expliquant comment l'installer sur un serveur.

III. Résultats obtenus

1. Etat final du projet

Afin de décrire l'état final de notre chatbot, il est intéressant de jeter un œil à notre liste de « competency questions » initiale, regroupant les principales compétences que nous envisagions pour le bot à la fin de la phase de développement.

Le chatbot sait-il **saluer** l'utilisateur ?	
Le chatbot sait-il donner la **définition** d'un mot sans mise en forme ?	
Le chatbot sait-il remonter les **traductions** d'un mot sans mise en forme ?	
Le chatbot sait-il remonter les **synonymes** d'un mot sans mise en forme ?	
Le chatbot sait-il donner les **mots apparentés** du mot sans mise en forme ?	
Le chatbot sait-il répondre aux tâches précédentes avec une mise en forme quelconque ?	
Le chatbot est-il capable de mettre en forme les traductions d'un mot sous forme d'un tableau à 2 colonnes : - Colonne 1 : **Mot traduit** - Colonne 2 : **Langue** + éventuellement un **drapeau**	
Le chatbot est-il capable de mettre en forme les **synonymes** et **mots apparentés** d'un mot sous forme d'une **carte mentale** ?	
Le chatbot est-il capable de restituer **toutes les informations simultanément** concernant un mot : définition, traduction(s), synonymes, mots apparentés	
Le chatbot est-il capable de restituer les informations concernant un mot en comprenant le **type d'information** que veut l'utilisateur ? (Définition si l'utilisateur demande une définition, synonyme(s) si l'utilisateur demande des synonymes, etc.)	
Le chatbot sait-il répondre à une question de **quantité** sans mise en forme particulière ? (type : Combien de fois la terre tourne-t-elle sur elle même pour une durée de 1 an)	
Le chatbot sait-il répondre à une question de **quantité** avec mise en forme particulière de type camembert ou histogramme	

Les lignes vertes correspondent à des compétences fonctionnelles, les lignes rouges à des compétences abandonnées ou pas encore implémentées.

Pour des raisons de lisibilité au niveau de l'interface, nous avons abandonné au cours du développement la compétence de remonter toutes les informations d'un coup. En effet, notre robot était assez compétent pour comprendre les questions, donc autant profiter de son intelligence pour afficher uniquement les informations nécessaires.

Nous avons également abandonné les questions en rapport avec des quantités, qui étaient à la base proposées pour aller plus loin, car DBpedia n'était pas assez fourni concernant les informations numériques.

Cependant, à part cela, tout le reste est fonctionnel, de la salutation de base aux questions spécifiques qui étaient demandées dès le début du projet.

2. Installation du chatbot

Voici la démarche pour installer le chatbot à partir de l'archive .rar. Cette partie est également présente dans le fichier « setting-up-the-chatbot.md ».

- ➔ Installer NGINX sur le serveur hébergeur du bot
- ➔ Glisser le fichier « nginx.conf » fourni à la racine de l'archive dans le dossier de configuration de nginx (par défaut à cet endroit : /etc/nginx/conf.d)
- ➔ Installer Rasa et ses composants : Les commandes et la méthode pour l'installer sont disponibles dans le fichier website/chatbot/readme.html de l'archive.

Mettre en place le site web

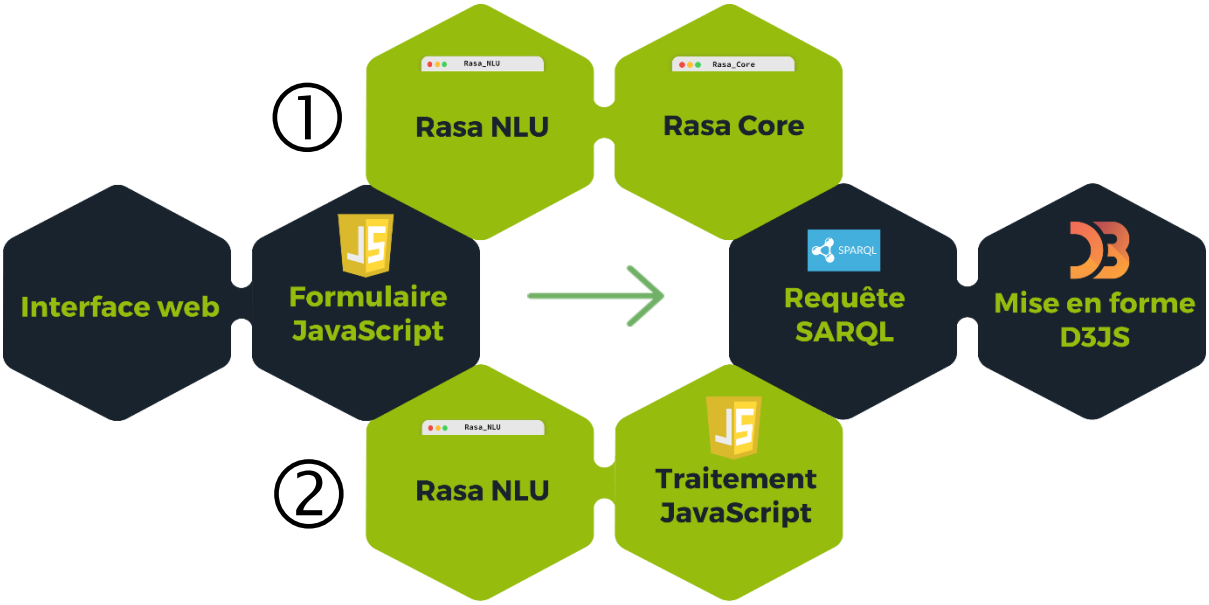
- ➔ Placer les dossiers « chatbot », « final » et « proxy » du dossier « website » de l'archive dans le dossier partagé du serveur nginx.

Configurer le serveur http pour Rasa NLU

- ➔ Aller dans le dossier « chatbot » sur le serveur avec une invite de commande (putty par exemple) et taper la commande « nohup ./run.sh » (Pour plus d'informations, se référer à la documentation Rasa fournie)

Après toutes ces étapes, le chatbot est installé sur votre site web ! N'hésitez pas à consulter les différentes documentations pour mieux comprendre le fonctionnement de ce dernier.

3. Extensions et améliorations possibles



Dans le schéma précèdent, nous pouvons distinguer le squelette global de notre chatbot. Dans le cadre de notre projet, nous avons fait le choix de l'architecture ② afin de rendre notre chatbot totalement « portable », et d'ouvrir la possibilité au client de décider librement de l'API qu'il utilisera.

Dans le choix ①, nous aurions eu la possibilité de développer le traitement et le déroulement des actions du chatbot par l'intermédiaire de la seconde partie de l'API Rasa Stack appelée « Rasa Core ». L'utilisation de cette dernière rend la configuration et l'entrainement du chatbot plus compliqués et plus long. En revanche, une fois ces étapes passées, le traitement d'une requête sera bien plus performant qu'avec le traitement en JavaScript. De plus, l'utilisation de Rasa Core permet de pratiquer un entrainement interactif et offre la possibilité de faire du machine learning, c'est à dire que le chatbot apprendra de lui-même dans le cas où il ne connait pas de réponse adaptée à son cas, et dans le cas contraire il renforcera ses choix (en posant des questions à l'utilisateur).

Avec Rasa Core, nous aurions pu développer avec beaucoup plus d'efficacité et de performance le dialogue avec l'utilisateur afin de tenir une conversation avec lui dans le but d'augmenter son intelligence, et de surcroît éviter le cas problématique où le chatbot n'arrive pas à répondre correctement. En plus de cela, Core permet d'emmagasiner des informations propres à l'utilisateur, comme par exemple son nom, âge, goût, etc… Très pratique lors d'une discussion.

Avec un peu plus de temps de travail et une base de données/connaissances adaptée, nous aurions également pu envisager traiter les questions de quantités. L'ontologie DBpedia n'est pas spécialement très adaptée pour répondre à de telles questions, mais concrètement le traitement resterait similaire à celui déjà mis en place. On peut d'ailleurs remarquer que rendre le chatbot plus intelligent en lui offrant la possibilité de comprendre plus de questions est totalement possible, ce qui en fait un projet extensible.

Conclusion

Pour conclure, nous avons eu au cours du troisième semestre de notre DUT Informatique la possibilité de mener à bien un projet tutoré encadré par le professeur Nicolle, consistant à mettre en place un chatbot, aussi appelé agent conversationnel en français.

Celui-ci devait à la fin du projet être capable de saluer un utilisateur, puis de lui exposer la définition, les traductions et les synonymes d'un mot que ce dernier aurait préalablement choisi.

Notre projet, étalé sur une phase de documentation, une phase d'analyse et une phase de réalisation, elles-mêmes séparées en cycles, a permis la création d'un chatbot répondant à ces attentes. L'équipe s'est organisée autour d'une méthode de travail agile, facilitée par l'utilisation de quelques outils informatiques et d'une bonne communication à l'intérieur du groupe et avec le tuteur.

Aujourd'hui, le chatbot est utilisable et étudiable par les ingénieurs du laboratoire de M. Nicolle dans le but de pousser plus loin le projet selon leurs besoins.

Si on devait porter un regard critique sur notre organisation durant le projet, nous pouvons remarquer qu'elle a globalement plutôt bien marché. Un peu plus de rigueur au niveau du formalisme aurait peut-être pu être préférable, avec notamment l'utilisation de plannings qui auraient évité les « rushs » en fin de cycle pour que tout marche au mieux. Sinon pour le reste, tout a plutôt bien marché et l'ambiance au sein du groupe a su rester au long du projet agréable et studieuse, ce qui est un plus pour travailler dans de bonnes situations.

Ce projet aura été une belle expérience pour tous les membres du groupe qui pourront par la suite mentionner ce travail dans leur CV, dans une lettre de motivation ou lors d'un entretien. En effet, les thèmes étudiés, à savoir l'intelligence artificielle et les bases de connaissances, sont deux domaines de l'informatique intéressants et relativement compliqués. Avoir la capacité d'en parler à des

professionnels nous permettrait de nous démarquer. En plus de cela, les technologies étudiées comme SPARQL, Nginx, Node.js, D3.js, etc., sont également des arguments intéressants pour nous valoriser.

Enfin et pour finir, nous tenons à remercier notre enseignant tuteur, Monsieur Christophe Nicolle, pour nous avoir encadrés et nous avoir offerts une aide précieuse tout au long du projet d'un œil bienveillant.