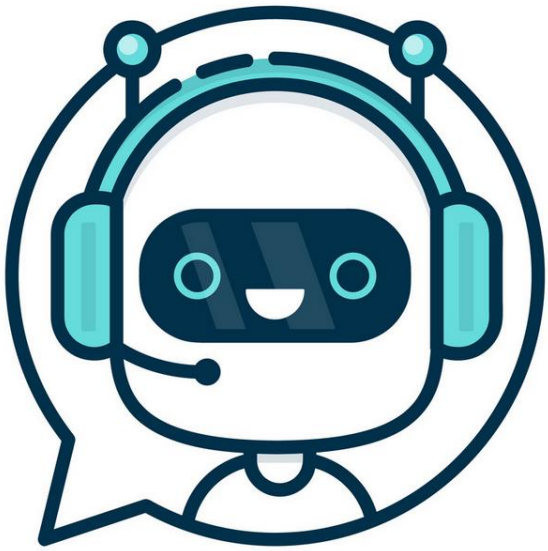
Hes·so VALAIS WALLIS
Haute Ecole de Gestion & Tourisme
Hochschule für Wirtschaft & Tourismus

Travail de Bachelor 2018 – 656-1 – Filière Informatique de Gestion

Chatbot : Le pont entre clients et professions libérales



THE COMPUTER FIRM

Etudiant-e : Samuel Coppey

Professeur : Prof. Dr. Michael Ignaz Schumacher

Dépôt : 08.08.2018 à 12h

Source des images de titre : The Computer Firm et (vectorstock)

Avant-propos

Ce document, ses annexes, le code source ainsi que toutes les applications relatives à ce travail de Bachelor ont été réalisés dans le cadre de la HES-SO Valais, filière informatique de gestion à Sierre et en collaboration avec l'entreprise The Computer Firm Sàrl. Ce travail écrit ainsi que l'oral de défense font partie intégrante de la validation du module 656-1.

Remerciements

En préambule, je désire remercier Monsieur le professeur Dr. Michael Ignaz Schumacher qui a encadré mon travail dans sa totalité. Son sens des priorités ainsi que sa rigueur ont contribué largement à m'améliorer.

Ma gratitude va également à Monsieur Fabien Dubosson qui m'a guidé pour toutes les questions d'ordre technique.

Je remercie particulièrement l'équipe de The Computer Firm de leur confiance et de leur travail, lié à l'élaboration de ce travail. Merci à Monsieur Raphaël Sculati qui m'a apporté toute son expérience pour l'entier de ce projet.

Enfin, je remercie les quelques personnes qui ont contribué à la présentation de ce travail et qui m'ont permis de m'améliorer jusqu'au dernier moment.

Un merci chaleureux à vous tous.

Table des matières

Table des illustrations

Liste des tableaux

Glossaires et abréviations

TCF	The Computer Firm
TB	Travail de Bachelor
NLP	Natural Language Processing
CMS	Content Management System
Bot	Logiciel d'intelligence artificielle
SDK	Software Development Kit
MEAN	MongoDB, Express, Angular, Node.JS
API	Application Programming Interface
JSON	JavaScript Object Notation

Introduction

Dans le cadre du travail de bachelor de la filière « Informatique de gestion » 2017-2018, j'ai travaillé en collaboration avec l'entreprise The Computer Firm. Depuis août 2017, j'ai effectué divers mandats de développement auprès de cette jeune entreprise de 3 associés, basée à Villeneuve.

The Computer Firm développe pour ses clients, principalement du domaine de la finance et du droit, des solutions digitales avec par exemple un module de prise de rendez-vous ou un CMS[1].

L'un d'entre eux est un module de prise de rendez-vous permettant aux visiteurs de fixer directement la date désirée de contact avec le prestataire. Ce module se synchronise avec les agendas de chacun et permet aux deux parties de trouver efficacement des plages disponibles pour se rencontrer.

Désireux d'améliorer leurs prestations, The Computer Firm avait prévu de développer un *Chatbot* capable de répondre à toutes les questions fréquentes, relatives aux professions de la fonction publique. D'un commun accord, nous avons décidé de mettre en place ce module au travers de mon travail de bachelor.

The Computer Firm a prospecté auprès de ses clients, des volontaires acquis à une phase pilote d'intégration d'un *ChatBot* sur leur site web. La fiduciaire « Monney Conseils Sàrl » basée à Genève s'est portée volontaire.

[1] Content Management System ou Système de Gestion de Contenu en français. Famille d'outils destinés à la conception et à la mise à jour de site web et d'applications multimédia.

Problématique

Un obstacle important à la croissance des PMEs est le manque d'informations fiscales, juridiques et de relations clients. Le module intervient comme intermédiaire sans engagement pour les responsables de PME et les particuliers. Il permet de créer une première interaction entre les experts de professions libérales (ex. fiscalistes, juristes, avocats etc.), et leurs clients.

De la collaboration avec The Computer Firm est issue l'idée suivante : « *ChatBot* : Le pont entre clients et professions libérales ».

The Computer Firm décide de me confier l'analyse des possibilités et des outils existants. La mise sur pied d'un serveur permettant la communication avec l'algorithme choisi, ainsi que la gestion des données font également partie de mon cahier des charges.

Etant propres à chaque entreprise, l'interface graphique et l'expérience utilisateur ont été moins approfondis afin de trouver les meilleures solutions d'un point de vue logique business.

La solution que je propose et détaille n'est en aucun cas la seule option à la disposition des utilisateurs. Les technologies ont été choisies pour permettre à The Computer Firm une intégration rapide et simple à leurs applications existantes.

1 État de l'art

1.1 Introduction aux *Chatbots*

Historiquement, les premiers *chatbot*s étaient utilisés sous formes d'agents virtuels, représentés par une image ou un avatar humain.

Les *chatbots* sont des outils de plus en plus utilisés par les services de marketing des entreprises. Ceux-ci permettent une interaction quasi humaine (voire humaine pour certains *chatbots* performants) et sont capables de répondre à des questions fréquentes, de donner des documents, des urls ou des images. Ils permettent également la connexion à des APIs telles que celle de Wikipédia ou d'une météo online. En définitif, tout ce qui peut être développé de manière logique dans une application est potentiellement applicable à un *chatbot*. Ce dernier retranscrira simplement l'information sous forme de dialogue humain.

Notre *chatbot* va s'orienter vers un service de réponses aux questions fréquentes. Il pourra également fournir des urls contenant des PDFs ou liste de documents. Il sera doté d'un champ lexical relativement large, ce qui lui permettra de répondre à un éventail de questions.

Figure 1 : Probablement le bot le plus connu : Siri (Rossignol, 2017)

1.2 Leurs utilisations

L'utilisation d'un *chatbot* est relativement intuitive et spontanée. Elle permet de discuter avec notre interlocuteur, dans notre cas une intelligence artificielle, qui répondra aux questions posées, comme le ferait un être humain au service de support derrière son écran. Afin que l'interaction soit le plus simple possible pour l'utilisateur, un travail important de développement lié à la gestion des taux de certitudes servira à définir si le *chatbot* doit ou non donner une information pour laquelle il est hésitant.

Les *bots* se retrouvent aujourd'hui dans une quantité d'applications sans que l'on s'en aperçoive. *Ils* ont débarqué dans la quasi-totalité des applications de messagerie instantanée. Skype, Messenger, Telegram ou encore Slack possèdent actuellement un service de *bots*. À relever que c'est un terrain de jeu parfait. Plus de 60 milliards de messages sont échangés chaque jour. Cela représente une richesse du point de vue de la récolte de données. (Eaton-Cardone, 2017)

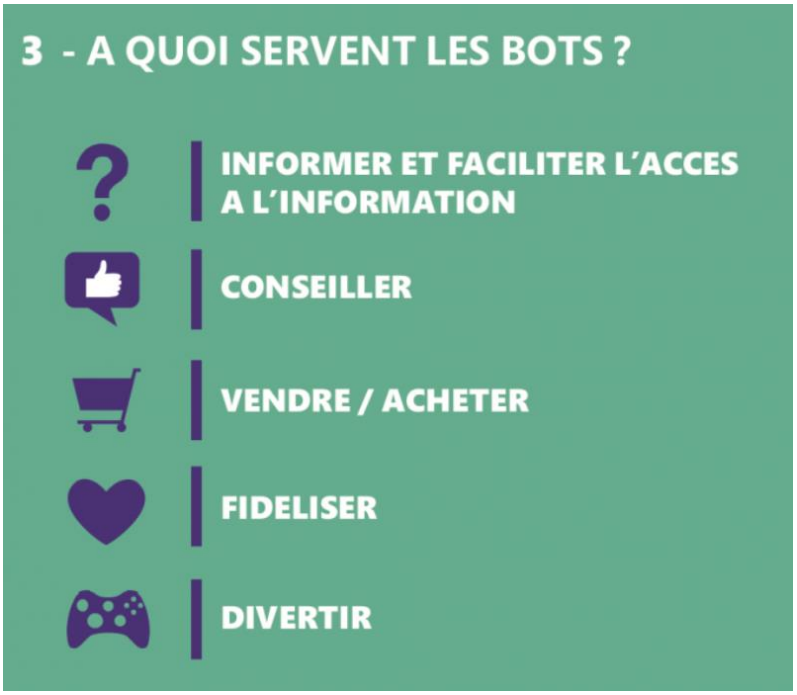


Figure 2 : A quoi servent les bots ? (Bathelot, 2018)

1.3 Le Natural Language Processing

1.3.1 Définition

Le Natural Language Processing (NLP) ou traitement automatique du langage naturel est un domaine multidisciplinaire impliquant la linguistique, l'informatique et l'intelligence artificielle. Cette technologie est destinée à la création d'outils de traitement de la langue naturelle pour toutes sortes d'applications, comme les *chatbots*.

Actuellement, les méthodes de traitements de langage naturel se basent sur diverses procédures stochastiques, probabilistes ou statistiques.

L'analyse d'une phrase est très compliquée et peut correspondre à des millions d'arbres syntaxiques différents. La taille de la grammaire et l'impossibilité de choisir simplement la structure de la phrase analysée en sont les raisons. Actuellement, les analyses grammaticales ne sont plus utilisées, une approche d'apprentissage automatique est privilégiée.

La méthode utilisée par notre service est l'extraction d'entities et d'intents. Deux termes définis par la suite. Ces éléments nous permettront de définir les intentions de l'utilisateur, ses envies ou encore ses sentiments. (Wikipédia, Traitement automatique du langage naturel, 2018)

1.3.2 Entity

La définition de l'entity peut être relativement abstraite. Pour cette raison, l'utilisation d'exemples concrets illustrera au mieux les nuances qui en ressortent.

Dans le Natural Language Processing, une entity, au sens purement littéraire, recherche des objets textuels (mot ou groupe de mots) catégorisables dans des classes.

Imaginons que nous créons un *chatbot* de livraison de pizza. Les entities à définir seront par exemple :

Entity	Exemple
Type_de_pizza	Hawaï, Marguerite, Calzone
Taille	Petite, grande
Ingredient_suppl	Oignon, extra fromage, poivre

Tableau 1 : Exemple d'entity

Lorsque les entities seront définies, il sera possible d'entraîner notre *bot*. Des phrases d'entraînement pourront être saisies par le développeur, afin de mapper les entities avec des mots ou groupe de mots :

« J'aimerais commander une pizza Marguerite avec un supplément de fromage. »

« Pourriez-vous me livrer une petite pizza 4 saisons. »

L'intelligence artificielle, est capable d'associer ces mots ou groupes de mots aux entities.

« J'aimerais commander une pizza Marguerite avec un supplément de fromage. »

« Pourriez-vous me livrer une petite pizza 4 saisons. »

Le type de pizza étant représenté en bleu, la taille en vert et les ingrédients supplémentaires en rouge.

L'exemple ci-dessus le plus concret possible, simplifie la compréhension de l'entity.

Afin de rebondir sur le champ lexical cité au-dessus, il est évident que des géants du web comme Facebook, Google ou IBM possèdent des milliards de données utilisées à les alimenter. Ceci servira à rendre attentif notre *chatbot* que les expressions « bonjour » ou « salut » font partie du même groupe de mots et assureront une interprétation plus simple. (Wikipédia, Reconnaissance d'entités nommées, 2018)

1.3.3 Intent

En respectant la même structure que pour l'entity, la définition au sens strict d'un intent correspond à la question d'un utilisateur qui interagit avec un *chatbot* : qu'elle est son intention, qu'elle est sa demande ?

L'intent sert à définir ce que désire l'utilisateur. Pour illustrer cela, reprenons notre exemple sur les pizzas. À la lecture, après extraction de nos entity, nous connaissons que l'utilisateur parle de pizzas, de leurs types et si supplément il y a. À première vue nos deux phrases sont plutôt similaires :

« J'aimerais commander une pizza Marguerite avec un supplément de fromage. »

« Pourriez-vous me livrer une petite pizza 4 saisons. »

En réalité, nos deux utilisateurs ont des intentions différentes. L'un désire commander une pizza et l'autre aimerait se la faire livrer. C'est là qu'interviennent nos *intents* qui seront capables de déchiffrer les intentions de l'utilisateur, afin d'avoir une réponse adaptée à la demande. (Perisetla, 2016)

1.3.4 L'entraînement

L'entraînement d'un *chatbot* est le meilleur moyen de s'assurer qu'il ne fera pas ou peu d'erreurs d'analyse de texte. Grâce aux phrases et mots-clés fournis au service de NLP, dans notre cas wit.ai, la solution étendra ses capacités à renvoyer des entities correctes lors de l'analyse des phrases.

Il peut être effectué de diverses manières, dans notre cas wit.ai donne la possibilité de saisir à la main une phrase ou un mot après l'autre :

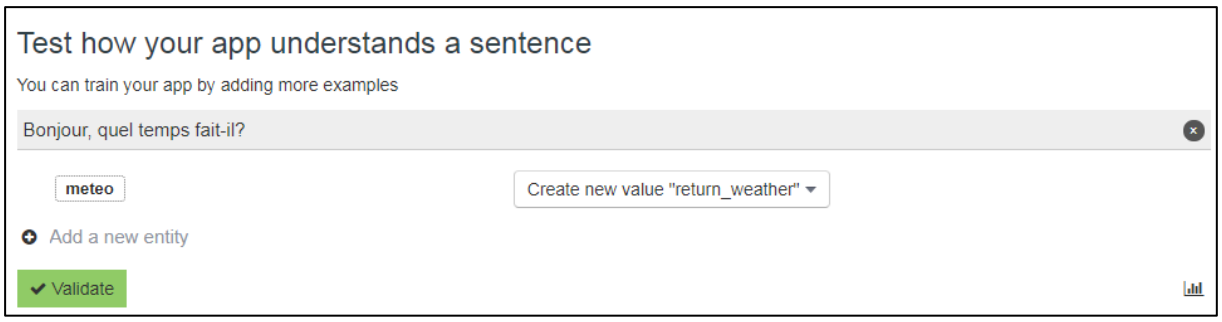


Figure 3 : Exemple d'entraînement

Il suffit d'entrer une phrase, de définir l'entity correspondante respectivement son intent. Cette manière d'agir pose un problème car il est seulement possible de créer une phrase après l'autre. Ce procédé est lent et rébarbatif.

Le SDK Node.JS fournit au développeur un moyen d'entraîner son *bot* à partir de données existantes. Il est possible d'extraire des conversations écrites en format CSV et de les transférer dans wit.ai afin d'entraîner notre chat. Cela se réalise grâce à une fonction écrite du côté serveur.

1.3.5 La confiance

Nous allons profiter de la section entities de notre application pour aborder la question liée à la confiance de notre *bot*. Afin d'avoir des indications précises sur la confiance, nous avons besoin de données, d'interactions avec le *bot* et d'un *bot* entraîné.

La confiance est une valeur utilisée par notre logiciel pour nous indiquer le niveau de précision relatif à la détection de l'entity, toujours en rapport avec la phrase de l'utilisateur.

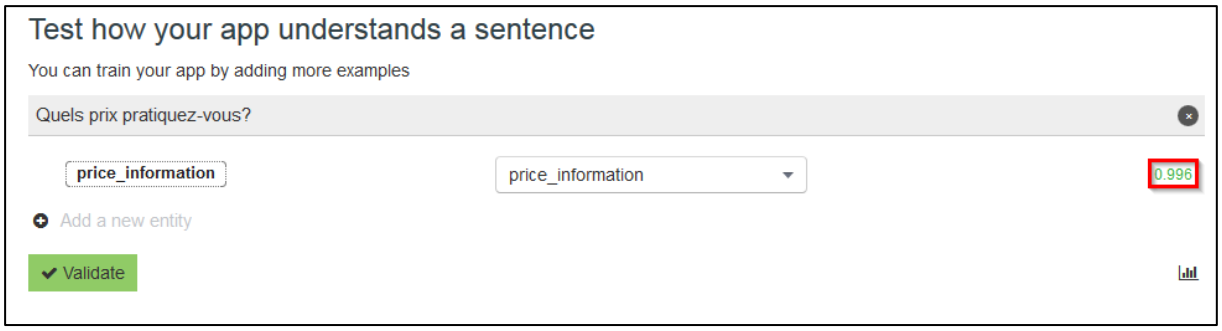


Figure 4 : Niveau de confiance

Dans cet exemple, notre algorithme pense qu'il y a 0.996 chance sur 1 (99.6%) que notre phrase corresponde à l'entity, price_information, respectivement l'intent price_information. Ce taux élevé est atteint car notre application est déjà entraînée.

En entrant dans les spécifications de chaque entity, il est possible d'avoir accès à un graphique nous permettant d'évaluer les taux de fiabilité de chaque demande.

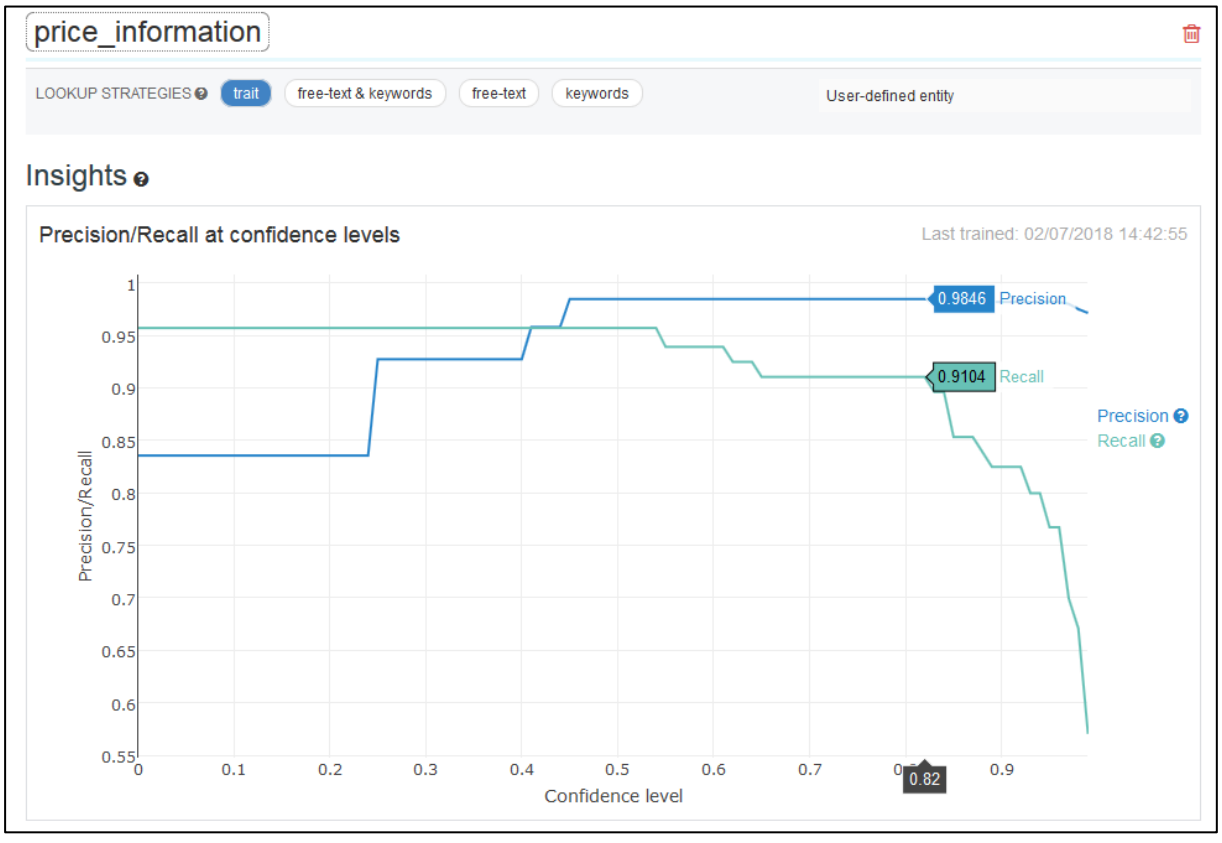


Figure 5 : Graphique du niveau de précision

Le courbe bleu concerne la précision avec laquelle le logiciel détecte notre entity. La courbe verte rapporte la fréquence à laquelle Wit détecte cette entity lorsqu'elle est réellement dans la phrase. Il est très important d'avoir une grande précision dans les

réponses, par conséquent la courbe de précision doit être élevée. Si nous désirons que notre entité soit extraite souvent, la courbe de rappel doit être haute.

Dans notre cas, le modèle a une précision de 0.9846 et un rappel de 0.9104, ce qui est excellent, 0.95 étant la valeur maximale pour un rappel. Ceci nous permet de déduire que pour avoir une précision de 0.9846 et un rappel de 0.9104, il nous faut un niveau de confiance (axe X) d'au moins 0.82.

Cette analyse nous permet de définir un seuil à 0.82. Ainsi toutes les valeurs de confiance inférieures à 0.82 doivent être gérées différemment car jugées trop basses. (wit.ai, Using Wit.ai programmatically)

1.3.6 Analyse des solutions existantes

De prime abord, nous analysons les différents services de *chatbot* présents sur la toile, ainsi que leur technologie de NLP respective. L'analyse se base sur une série de critères définis en collaboration avec The Computer Firm, afin de parfaitement convenir aux besoins de ces derniers. De nos « brainstorming », sont retenus huit points sur lesquels les services seront jugés. Chaque point est pondéré en fonction de son importance tant pour le développement que pour les besoins métiers de The Computer Firm.

Critères	**Pondération**	**Explication**
Documentation	8 / 37	La documentation est un des critères pondérés le plus conséquent, car elle servira de base de travail à la création du *chatbot*. La technologie étant nouvelle, les informations trouvées liées au développement sont importantes.
Communauté	5 / 37	Afin de quantifier la qualité de la communauté, la base de nos critères tels que les étoiles sur GitHub, les informations disponibles sur des sites comme Reddit.
Licences et tarifs	5 / 37	Possibilité d'utiliser ce service de manière commerciale, prix des différentes prestations.
Fonctionnalités	3 / 37	Manière dont le service construit ses *bots* (Entities, intents, machine learning…)
Langage de développement	5 / 37	Langage dans lequel le *bot* est disponible, facilité d'intégration du back-end.
Langue	7 / 37	Langue dans laquelle le *bot* est capable de communiquer.
Rayon d'action	1 / 37	Rayon dans lequel est possible l'utilisation du *bot* (Business, application simple, complexe, domaine de la santé)
Droit à la données	3 / 37	Clarification de la part du fournisseur de *bot* ainsi que des données qui passent à travers celui-ci

L'analyse de 8 fournisseurs de NLP a été réalisée afin d'identifier l'élément qui répond au mieux aux exigences du développement et de l'entreprise.

Ci-dessous un résumé commenté des éléments principaux de chaque fournisseur :

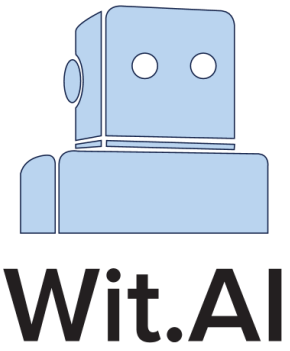
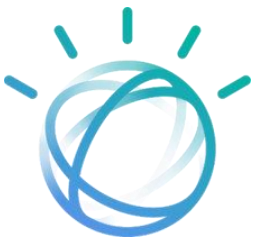
Chatbot	Notation	Court résumé
 *Figure 6 : Luis.ai*	29 / 37	Luis.ai, service de NLP de Microsoft, profite d'un grand nombre de données afin d'entraîner au mieux leurs *bots*. L'analyse lexicale était disponible dans une dizaine de langues dont le français. Il manquait selon nous quelques informations plus avancées. (Microsoft, About Language Understanding, 2017)
 *Figure 3 : Wit.ai*	34 / 37	Wit.ai, ancien service de NLP de Facebook est utilisé par plus de 120'000 développeurs à travers le monde, ce qui en fait la plus grande communauté à ce jour. Il est disponible dans une cinquantaine de langues dont le français. Avec une note de 34 sur 37, il obtient selon nos critères la plus haute note. Il est gratuit, inclus usage commercial. (Wit.ai, 2018)
 *Figure 7 : DialogFlow - Google*	33 / 37	Dans la même lignée que les précédents, DialogFlow, le *bot* de Google a à choix beaucoup de langues ainsi qu'un service de machine learning. Utilisé pour le *bot* chez Mercedes ou encore Giorgio Armani. Il est fiable et fait partie des meilleurs services de NLP au monde. (Google)
 *Figure 8 : Watson - IBM*	29 / 37	Watson d'IBM est certainement le plus connu des services d'intelligence artificielle. Globalement la documentation de l'API et de tous les guides de départ sont bons. La communauté est aussi assez conséquente. Il est connu par les développeurs pour être un des plus performant *bot* du marché. Seul gros défaut, le français n'est pas disponible. (IBM)

 *Figure 9 : Chatfuel*	24 / 37	Chatfuel est un service de NLP considéré comme bon pour les débutants. C'est un outil de type « CMS ». L'assistance et l'aide à la saisie sont guidées. Malheureusement, jugé trop basique pour nos besoins. Points forts : sa gratuité et beaucoup de langues à disposition (ChatFuel)
 *Figure 10 : Rebot.me*	15 / 37	Leur site web et les données à disposition sont relativement maigres. Selon nos lectures, le service est limité et n'est pas adapté pour des applications business ou fonctionnalités plus avancées. Plutôt orienté « CMS », à entraîner comme un formulaire de questions – réponses. Pas de code requis pour le développer (Rebot.me)
 *Figure 11 : Pandorabot*	25 / 37	Bonne documentation de l'API et de son utilisation. 250'000 développeurs sont inscrits et plus de 300'000 *chatbots* ont été créés avec ce service. Malgré cela, des points négatifs ont été relevés. Il est payant, le français n'est pas disponible, deux conditions sine qua non pour The Computer Firm. Regrettable car un *bot* ayant passé le « Test de Turing[2] » a été créé avec ce service de NLP (Pandorabot)
 *Figure 12 : Reply.ai*	14 / 37	Peu de choses sont à disposition pour cet outil, une vidéo est disponible sur leur site web. Il faut leur écrire afin d'avoir des informations plus précises en retour. Nous n'avons pas trouvé de Github. Une plateforme est fournie où il est possible de créer son *bot* qui utilise wit.ai et dialogflow comme service de NLP. C'est en fait un intermédiaire. (Reply.ai)

[2] Test d'un être humain mis en confrontation verbale, à l'aveugle, avec une intelligence artificielle et un autre être humain. Si la personne ne discerne pas lequel de ses interlocuteurs est un ordinateur, le test est considéré comme réussi.

1.3.7 Choix de l'outil de NLP

A l'analyse de la grille, quatre outils se démarquent clairement au niveau de leurs notations respectives à savoir, dans l'ordre :

- Wit.ai
- DialogFlow
- Luis.ai
- Watson

À l'évaluation, les 4 protagonistes se tiennent de près. La fourniture d'un service *chatbot* en français est une condition importante pour The Computer Firm. Watson ne l'étant pas, se trouve éliminé du carré final. Luis.ai est également écarté pour son manque de documentation avancée. Le choix final se décide entre Wit.ai et DialogFlow.

Après discussion, les différents outils étant relativement proches dans les valeurs de référence de la grille d'analyse, décision est prise de comparer le côté pratique de chaque service de NLP.

1.3.8 Confirmation du choix de wit.ai

Pour ce scénario simple, nous décidons de créer une entity ainsi qu'un intent afin de nous rendre compte des avantages et inconvénients de chaque service.

La création d'une nouvelle entity dans wit.ai s'effectue de manière relativement intuitive. Il suffit de saisir une phrase type, laquelle sera liée à une entity puis à un intent.

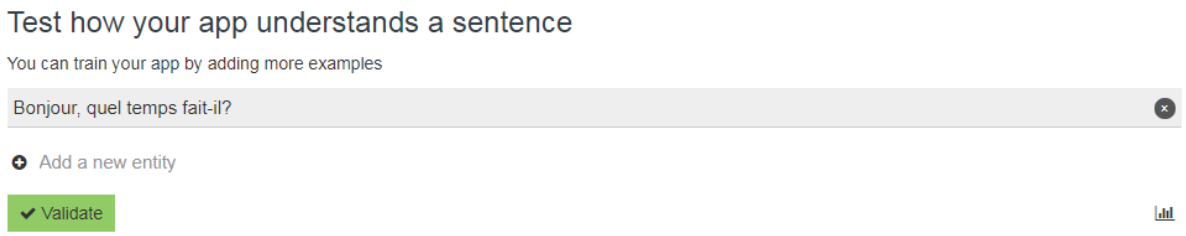


Figure 13 : Exemple de création d'une entité sous wit.ai

Comme le système est nouveau, n'étant pas entraîné, il ne comprend pas la question qui lui est posée. Nous allons donc saisir le nom d'une entity dans le champs « Add a new entity ». Ceci nous permettra par la suite d'y lier un intent. Il est nécessaire de préciser que le vocabulaire de wit.ai est différent car le mot « value » fait référence à un intent.

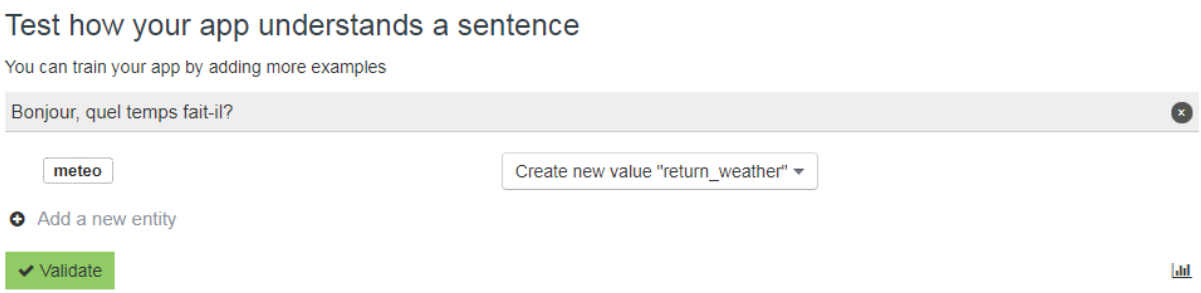


Figure 14 : Création et nommage de l'entity

En cliquant sur « Validate », nous allons entraîner notre système à répondre de manière correcte aux prochaines questions de ce type.

Nous avons maintenant à disposition une entity qui a, pour l'instant, une phrase de référence : « Bonjour, quel temps fait-il ? ». Cette phrase pourra être légèrement déclinée par l'utilisateur. Avec le champ lexical à sa disposition, l'algorithme réussira à décomposer l'expression et en déduire l'équivalence fournie au *bot*.

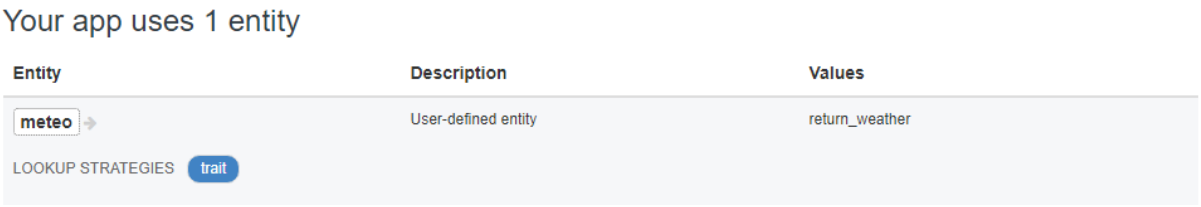


Figure 15 : Première entity

En cliquant simplement sur l'encadré « meteo », il nous est possible d'accéder à plus de détails concernant notre entity.

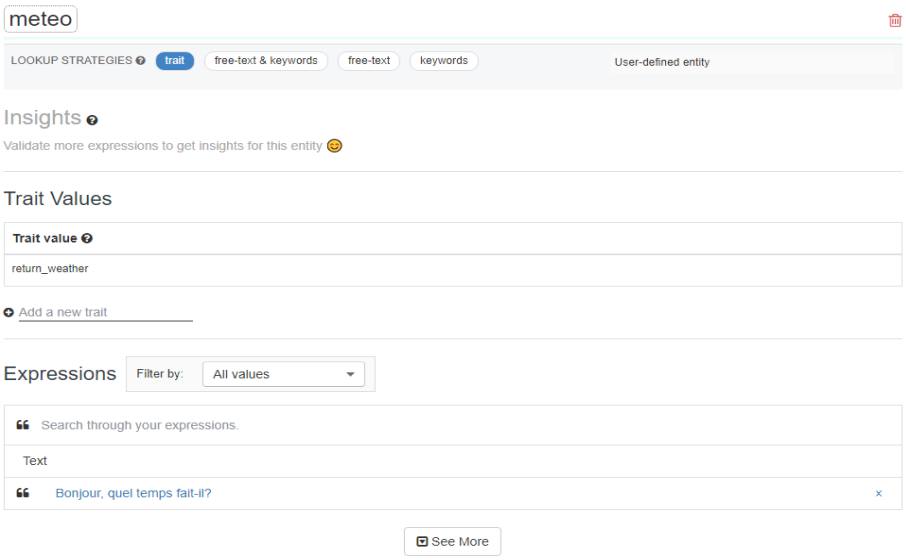


Figure 16 : Détails d'une entity

Wit.ai possède plusieurs stratégies d'analyse :

- Trait : lorsque la phrase entière saisie est égale à notre intent. (une intention, un sentiment)
- Texte libre : lorsque nous avons besoin d'extraire une partie du message non-définie dans les listes des valeurs possibles. (un contact ou le corps d'un texte)
- Mot-clé : lorsque la valeur d'une entity appartient à une liste prédéfinie à chercher dans un mot-clé de cette phrase (type de pizza, un pays).

Ensuite, nous avons à disposition la valeur d'un trait ou, pour les mots clés et les textes libres, des synonymes. Toutes ces valeurs composent l'intent.

Pour terminer, les phrases fournies à notre *bot* afin de l'entraîner.

1.3.9 Simple cas d'utilisation DialogFlow

Concernant DialogFlow, le système de création d'intent et d'entity est très similaire à celui de wit.ai. Dans un premier temps, nous créons notre entity, à laquelle nous ajoutons un intent.

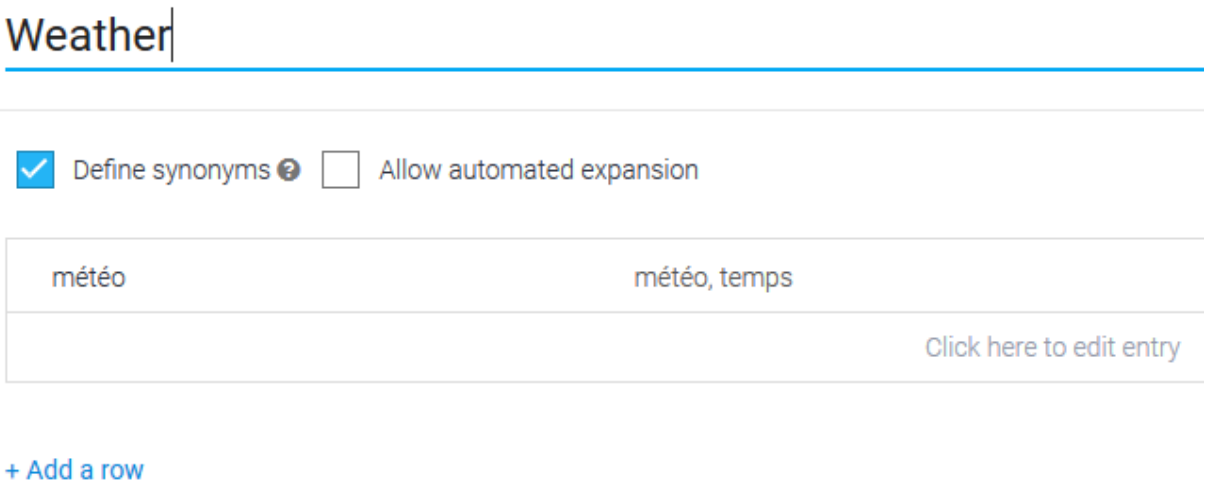


Figure 17 : Création d'une entity dans DialogFlow

Au niveau de l'intent, tout se passe de la même manière :

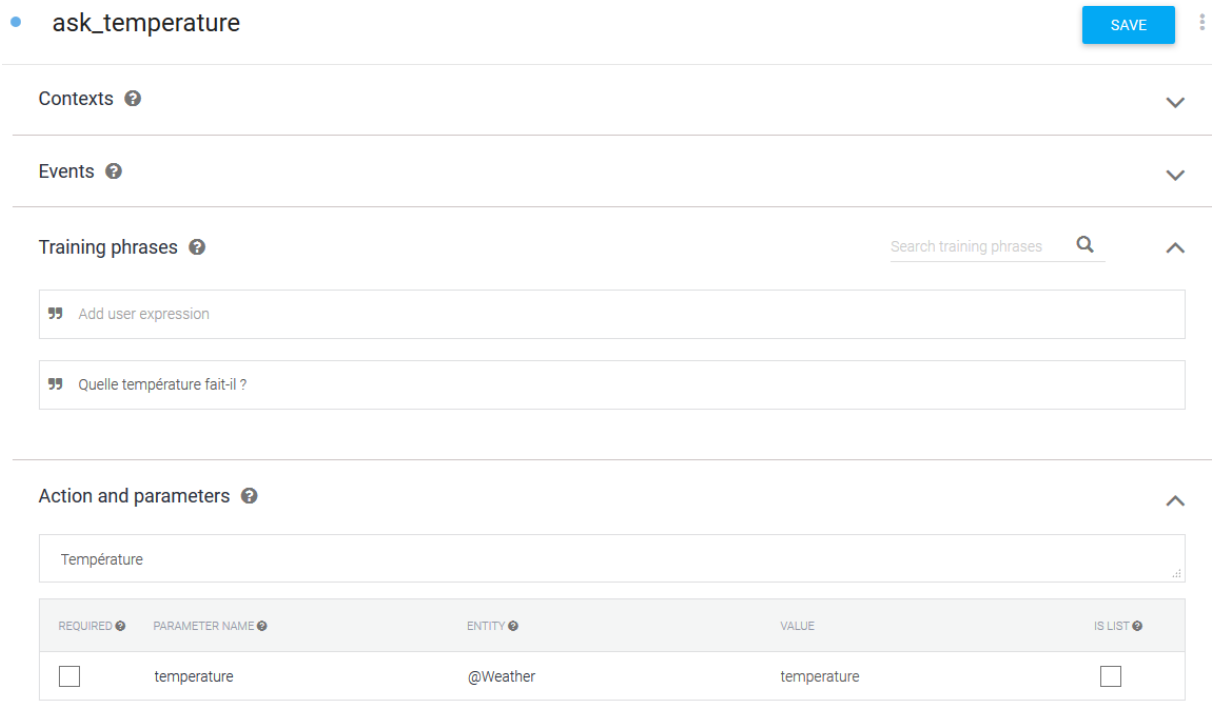


Figure 18 : Création d'une intent

Nous créons notre intent, auquel nous lions simplement notre entity. La principale différence entre les deux systèmes réside sur le fait que DialogFlow propose d'intégrer la réponse directement à l'intent. Dans le cas de wit.ai la réponse est gérée dans notre base de données, ce qui à mon avis est préférable.

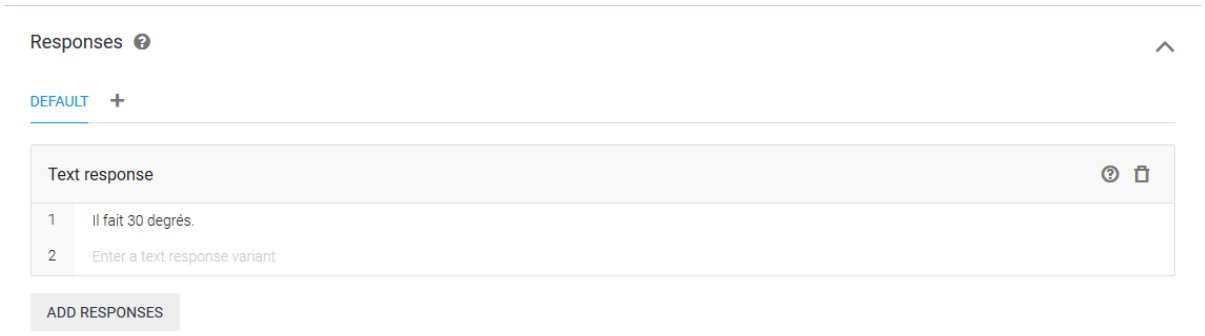


Figure 19 : Ajout de la réponse à l'intent

Les tests des intents sont similaires à wit.ai. Il est possible de saisir des phrases afin de vérifier si les entrées sont bien comprises.

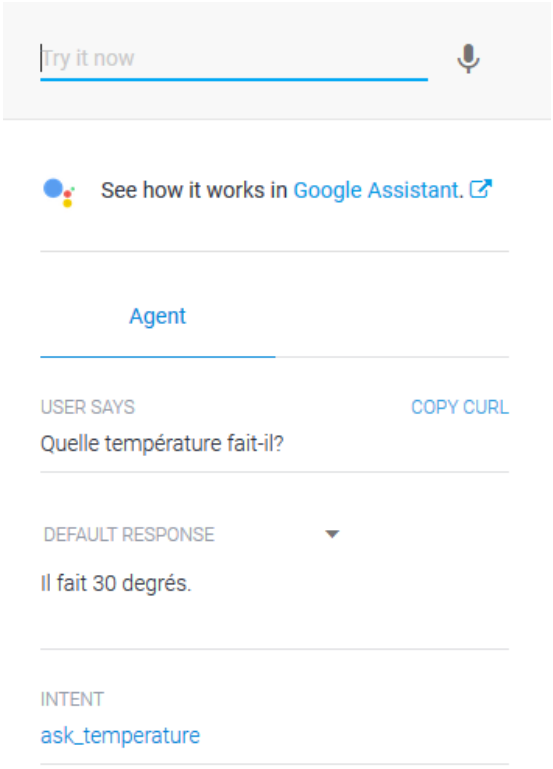


Figure 20 : Test de DialogFlow

1.3.10 Choix définitif

Notre choix final se porte sur wit.ai car la flexibilité du service de NLP nous a séduit prioritairement. En effet, wit.ai laisse une grande place au développeur afin qu'il puisse gérer certaines fonctionnalités selon ses besoins.

Un critère séduisant est également le fait que wit.ai autorise l'usage commercial du produit sans aucune contrepartie.

En finalité, aucune différence notable n'a été constatée dans la performance des deux algorithmes. Entendu que le plein potentiel des deux outils n'a pas été exploité.

Le choix définitif de l'outil s'est fait dans le courant du mois de mars. Considérant le peu d'expérience acquise en un mois dans ce domaine, il serait intéressant d'y revenir dans le chapitre « Conclusion », en fin de travail.

1.4 RGPD

En janvier 2012, la Commission européenne a proposé un ensemble de mesures législatives afin d'actualiser et moderniser les règles contenues dans l'ancienne directive de 1995 sur la protection des données.

Entré en vigueur le 25 mai 2018 dans l'Union Européenne, le nouveau Règlement sur la protection des données peut être directement applicable à tous les acteurs actifs sur le territoire de l'Union Européenne.

Il devrait permettre aux citoyens d'avoir un meilleur contrôle sur leurs données personnelles en responsabilisant les entreprises.

Figure 21 : RGPD

Dans le cadre de ce travail, The Computer Firm ayant plusieurs clients basés dans l'Union européenne, nous avons tenu compte de cette variable lors de l'analyse de base.

Les clients principaux de The Computer Firm sont des fiduciaires ou des avocats. Les messages et questions posés à notre *bot* pouvant être confidentiels ou privés, nous nous baserons sur les critères donnés par la GDPR.

Afin d'entraîner notre *bot* et d'améliorer son efficacité, nous allons créer une base de données pour y stocker tous les messages envoyés par l'utilisateur. Cela permettra à court terme de les utiliser pour améliorer notre *chatbot*. Pour être en accord avec cette loi, nous demanderons à l'utilisateur d'accepter ou non cette condition. En cas d'accord, nous sauvegarderons ses messages dans notre base. En cas de refus, l'utilisateur pourra tout de même utiliser l'application, et ses messages ne seront pas sauvés.

La RGPD demande aux entreprises de laisser le choix à l'utilisateur de supprimer à tous moments les données saisies, même s'il a accepté notre police de confidentialité. (econocom, 2018)

La loi étant encore relativement floue, nous nous contenterons pour ce travail de ces mesures. (Suisse, 2018)

1.5 Problématique et marché

Une statistique met en évidence le fait que les gens ont peur d'échanger avec un robot. Le 75% des utilisateurs désirent connaître l'instant du dialogue avec une intelligence artificielle.

À ce jour, les *chatbots* ne sont pas encore totalement autonomes. Comme expliqué plus haut, le développeur effectue une certaine quantité de tâches pour veiller à son bon fonctionnement.

Statistiquement, un *chatbot* a en moyenne un taux de précision d'environ 85%.

Accessoirement, l'inventeur et collaborateur de Google, Monsieur Ray Kurweil, plus connu pour ses prédictions comme la singularité technologique, est actuellement directeur de service du « Natural Language Recognition ». Selon ses prédictions, aux environs de 2029, l'intelligence artificielle aura un niveau de conversation égal à celui de l'être humain. (Popper, 2016)

2 Architecture et environnement de développement

Avant d'aborder les parties techniques, nous allons détailler l'environnement de développement utilisé.

Partant de l'idée de ne rien réinventer, nous avons utilisé la *MEAN stack* qui est une collection de technologies basée sur du Javascript, utilisée pour développer des applications web. *MEAN* est l'acronyme pour MongoDB, Express, Angular.JS, Node.JS. Du client au serveur et à la base de données, chaque couche de notre solution est réalisée en Javascript.

Afin de rester en adéquation avec l'environnement de The Computer Firm, nous avons décidé de remplacer Angular.JS par React.JS, une autre librairie Javascript permettant de créer des interfaces utilisateurs. (sitepoint, 2014)

Schématiquement, l'environnement se définit de la manière suivante :

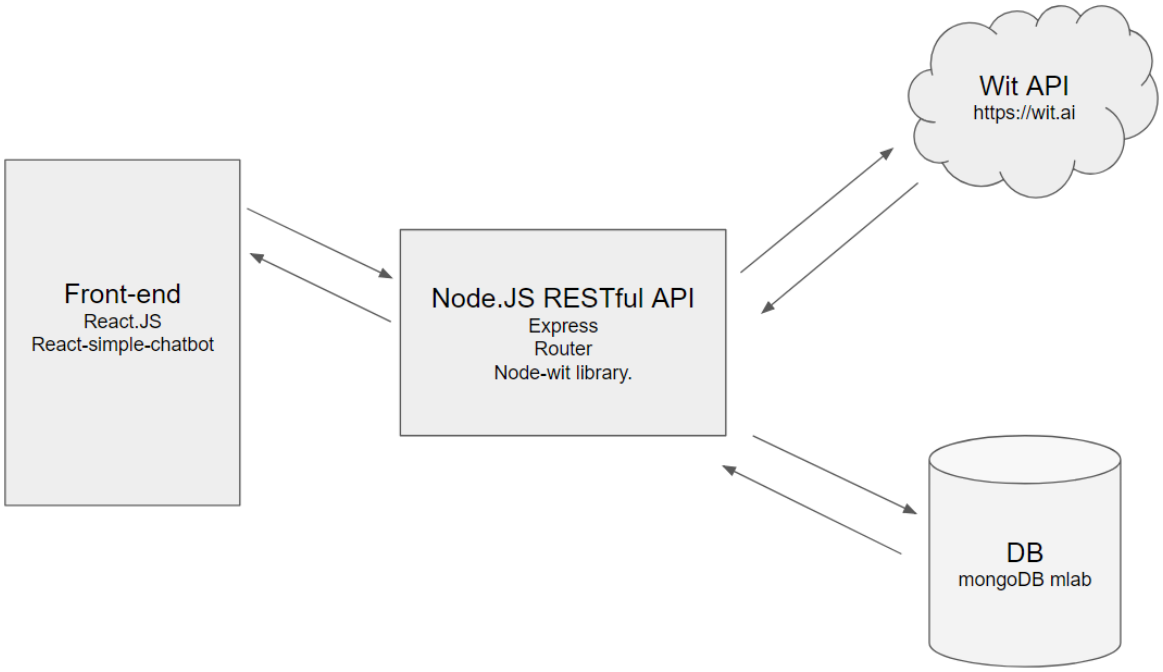


Figure 22 : Différentes couches de l'application

2.1 Node.JS

La plaque tournante de notre application sera notre API RESTful développée avec la plateforme de développement Javascript Node.JS. Cette plateforme est un réel environnement de développement « côté-serveur ». Elle permet de construire des applications hautement évolutives et modernes.

L'environnement Node.JS comprend le nécessaire à l'exécution de programmes écrits en Javascript. Grâce à Node.JS, notre Javascript peut faire beaucoup plus que de l'interaction dans un site web.

Le serveur recevra les requêtes depuis le front-end, à savoir le chat et transmettra ces demandes à la base de données pour la sauvegarde, ainsi qu'à l'API de wit.ai qui analysera le texte.

Dans un second temps, wit.ai enverra le résultat de son analyse vers le serveur Node.JS qui l'interprétera et, en collaboration avec la base de données, renverra la réponse de notre *bot* vers le chat. (Patel, 2018)

2.1.1 Choix de Node.JS

Node.JS est clairement un des pionniers du développement des serveurs web en Javascript. Grâce à ses outils et son comportement asynchrone, Node.JS évite toute perte de temps aux processeurs en attente d'une réponse « Input / Output ».

Côtés performance et rapidité, Node.JS a un fonctionnement *single threaded* et engagera toujours un seul *thread* en lieu et place d'en utiliser un par requête. Ce système évite au processeur de changer constamment de contexte et de nombreuses exécutions mémoire.

Notre choix s'est porté sur Node.JS car il permet la création d'un serveur en Javascript de la manière la plus stable et optimale possible.

Le fait que The Computer Firm possède déjà une architecture fonctionnant sur cette structure confirme notre choix. La nécessité de nous adapter ne peut que faciliter la future intégration de notre chat au panel d'outils que propose déjà cette société.

Server.js - Code source

```
1.  // set up ========================================================================
2.  // get all the tools we need
3.  var express  = require('express');
4.  var app      = express();
5.  var mongoose   = require('mongoose');
6.  var morgan = require('morgan');
7.  var bodyParser = require('body-parser');
8.  var cookieParser = require('cookie-parser');
9.  var flash    = require('connect-flash');
10. // var session      = require('express-session');
11. var config = require('./config/config.js');
12. var port = process.env.PORT || 6060;        // set our port
13. // configuration ===================================================================
14. mongoose.connect(config.MONGO_URL); // connect to our database
15. app.use(morgan('dev'));
16. app.use(cookieParser()); // read cookies (needed for auth)
17. app.use(bodyParser()); // get information from html forms
18. app.use(flash()); // use connect-flash for flash messages stored in session
19.
20. // middleware to use for all requests
21. app.all('/*', function(req, res, next) {
22.   res.header("Access-Control-Allow-Origin", "*");
23.   res.header('Access-Control-Allow-Methods', 'OPTIONS, GET,PUT,POST,DELETE');
24.   res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-
    Type, Accept");
25.   next();
26. });
27.
28. // routes ==========================================================================
29. require('./config/routes.js')(app); // load our routes and pass in our app and fully
     configured passport
30.
31. // launch ========================================================================
32. app.listen(port);
33. console.log('The magic happens on port : ' + port);
```

Tableau 2 : Server.js

Une première version du serveur Node.JS se lance en local sur le port 6060. Cela nous permet de poursuivre en traitant de certains modules utilisés dans le serveur, destinés à faire tourner notre application. (Digitalfirefly, 2013)

2.1.2 Modules Node.JS

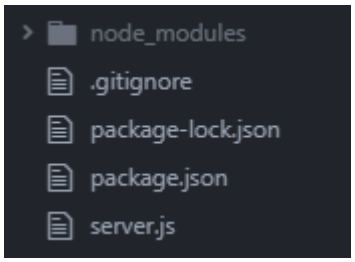


Figure 23 : Dossier module Node.JS

La structure de dossier de notre serveur se présente de la manière suivante. Le dossier *node_modules* va contenir tous les modules[3] installés dans notre serveur.

Package.json - Code source

```
1.  {
2.    "name": "node-api",
3.    "main": "server.js",
4.    "dependencies": {
5.      "bluebird": "^3.5.1",
6.      "body-parser": "~1.0.1",
7.      "connect-flash": "^0.1.1",
8.      "cookie-parser": "^1.4.3",
9.      "express": "~4.0.0",
10.     "mongoose": "^4.3.7",
11.     "morgan": "^1.9.0",
12.     "node-wit": "^5.0.0",
13.     "nodemon": "^1.17.5",
14.     "universal-cookie": "^2.2.0",
15.     "universal-cookie-express": "^2.2.0"
16.   }
17. }
```

Tableau 3 : Package.json

2.1.2.1 Express

Express est un framework utilisé à la construction d'applications web basée sur Node.JS. Ce framework fait partie intégrante de la « stack » MEAN (MongoDB, Express, Angular, Node.JS).

Il permet à l'utilisateur l'ajout d'une couche de logique serveur extrêmement légère et fonctionnelle. Il donne accès à plusieurs fonctions relativement utiles qui nous permettront, par exemple, de faire fonctionner notre serveur sur un port d'écoute défini. Cette librairie est inspirée du populaire framework Ruby « Sinatra » (ExpressJs)

[3] Petit programme ou outil ajouté à notre serveur afin d'étendre ses fonctionnalités. Par exemple, un module de cookies pourra nous permettre d'en créer dans le navigateur de l'utilisateur quand il se connecte.

Dans le tableau 3, lignes trois et quatre, nous initialisons le module express et le faisons pointer vers une variable nommée *app*. App aura ensuite la possibilité d'appeler un panel de fonctions telle la fonction *listen* qui donnera à notre serveur, l'accès à une adresse, locale ou web. Express pourra aussi nous aider à définir et pointer vers des routes.

2.1.2.2 Mongoose

Mongoose est un module facilitant l'accès à une base de données MongoDB qui sera détaillée dans un second temps. Notre choix s'est porté unanimement sur cette base de données. Chaque modèle, qui donnera accès aux collections de notre base de données, sera créé à l'aide de Mongoose. Dès lors, à chaque utilisation de nos modèles, nous aurons accès à des fonctions telles que *create* ou *find. Ceci* afin de créer ou récupérer des documents relatifs aux modèles, situés dans notre base de données.

À l'image d'un *entity Framework* pour le C#, Mongoose nous permettra de gérer aisément, de manière claire, les requêtes dirigées vers notre base de données (Mongoose)

2.1.2.3 Node-wit

Node-wit concerne le SDK[4] fourni par wit.ai pour l'intégration à Node.JS de ses requêtes.

Wit.ai possède une API permettant de faire des requêtes HTTP. Afin de simplifier la tâche des développeurs, wit.ai a mis en place des SDK pour Node.js, Python et Ruby. Dans notre grille d'analyse, ces éléments ont été pris en compte afin d'avoir une plateforme supportant un SDK Node.js (Wikipédia, Kit de développement)

Dans le menu « documentation » sur le site de wit.ai, nous avons la possibilité de choisir entre l'API et ses requêtes HTTP ainsi que les trois SDKs mentionnés au-dessus. Nous sommes redirigés vers le repository GitHub de wit.ai. L'installation du SDK nécessite la création d'un server Node.JS. (wit.ai, Node.js SDK for Wit.ai)

[4] Software development kit : ensemble d'outils logiciel destinés aux développeurs afin de faciliter le développement d'un logiciel sur une plateforme donnée.

2.1.2.4 Autres modules

Notre intention n'est pas de faire l'inventaire complet de tous les modules utilisés, les principaux étant définis ci-dessus. Les autres seront utilisés pour des tâches très précises. Notre application peut évidemment être développée sans ces outils. Ils sont à disposition principalement pour une question d'optimisation.

2.2 React.JS

Le front-end lui servira de partie exclusivement graphique. Il recevra les entrées clavier de l'utilisateur et les enverra simplement vers le serveur. Il affichera ensuite le retour fourni par le serveur à l'utilisateur.

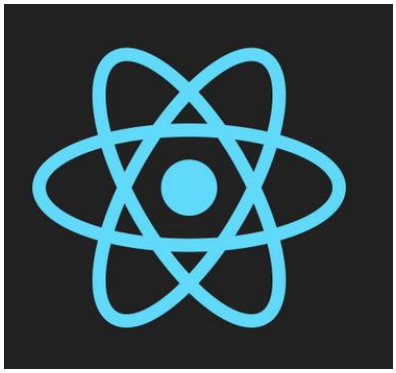


Figure 24 : Logo React.JS

L'intégration du front-end se fera avec le framework créé par Facebook : React.JS. Cette librairie Javascript nous permettra la création d'un interface utilisateur séparé en composants. Cette séparation offrira une maintenance facilitée. Chaque composant sera défini avec ou sans état « stateful or stateless » et sera capable de les gérer de manière autonome. (Facebook)

2.2.1 Choix de React.JS

De prime abord, intéressons-nous aux principales fonctionnalités de cette librairie.

Tout d'abord, React utilise le *JSX*[5] en lieu et place du traditionnel Javascript. Pour cause, JSX est du Javascript simple permettant d'y insérer des balises HTML.

Ensuite, React possède une librairie native permettant à une architecture React de tourner sur des plateformes comme Android ou IOS.

[5] Le JSX est une extension du Javascript. Elle est utilisé en collaboration avec React.JS et permet l'ajout de balises HTML dans le code Javascript

Enfin, l'architecture orientée composants permet de passer des valeurs à travers chacun des composants enfants. Ces composants ne pouvant pas modifier directement les valeurs peuvent passer des fonctions *Call back*[6] qui permettent certaines modifications.

La maintenance d'un code séparé en composants est largement accrue. Nous pourrions également citer un critère plus subjectif. La technologie étant nouvelle, il est toujours intéressant de travailler avec des outils récents.

React.JS a été définie, comme pour le serveur, en tant que technologie de choix car The Computer Firm utilise cela pour ses modules déjà en place.

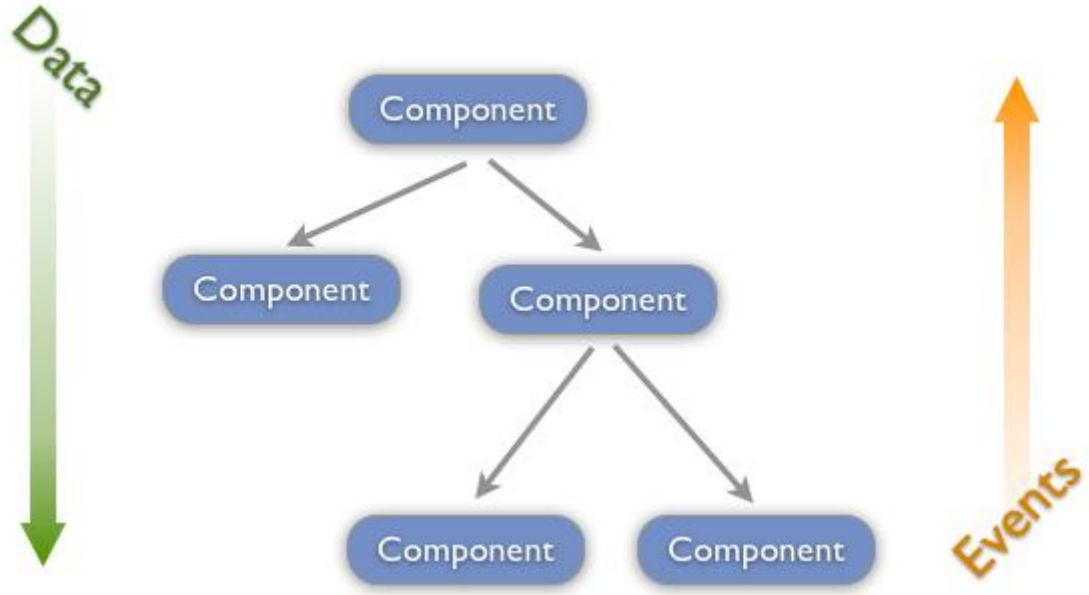


Figure 25 : Direction des flux

[6] Une fonction « Call-back » est une fonction passée dans une autre fonction en tant qu'argument. C'est régulièrement utiliser pour continuer l'exécution d'un code asynchrone. (docs)

2.3 MongoDB

MongoDB a été téléchargé plus de 40 millions de fois et par plus de 850'000 étudiants universitaires.

MongoDB est une base de données NoSQL[7] open-source utilisée dans la *stack MEAN.* Elle est multi-plateformes, orientée document, ce qui nous donne des performances élevées ainsi qu'une grande évolutivité. Elle travaille sur le concept de collections et de documents ; une collection étant un groupe de documents ou l'équivalence d'une table dans un modèle relationnel. Le document est quant à lui une série de paires « clé-valeur ». Cela signifie que les documents d'une même collection ne doivent pas forcément avoir les mêmes champs ou la même structure.

Ce type de base de données est très apprécié des développeurs Javascript car les données sont stockées dans des champs similaires au JSON.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

Figure 26 : Parallèle entre MongoDB et modèle relationnel

Le modèle de cartographie orienté documents permet un mapping rapide des objets dans notre application, rendant les données facilement modelables.

[7] Base de données n'étant pas basée sur un modèle relationnel.

2.3.1 Le choix de MongoDB

Le choix s'est fait naturellement car ce type de base de données est très facilement intégrable à notre architecture, de par ses facilités avec le Javascript. Comme déjà relevé, MongoDB possède une solide communauté et des développeurs dans le monde entier qui améliorent le produit chaque jour, celui-ci étant open-source.

Il n'y a aucun concept de relation, MongoDB étant totalement sans schéma. La base de données est orientée sur des collections possédant différents documents, eux-mêmes composés de nombres, de types. Le contenu d'un document à l'autre peut être totalement variable.

Du fait de cette orientation, la structure d'un seul objet est très claire. Nous ne trouverons également aucune jointure complexe, sans oublier sa grande évolutivité.

Enfin, elle est très appropriée aux grandes quantités de données. Ce qui pourrait être notre cas avec un grand nombre de messages à sauver.

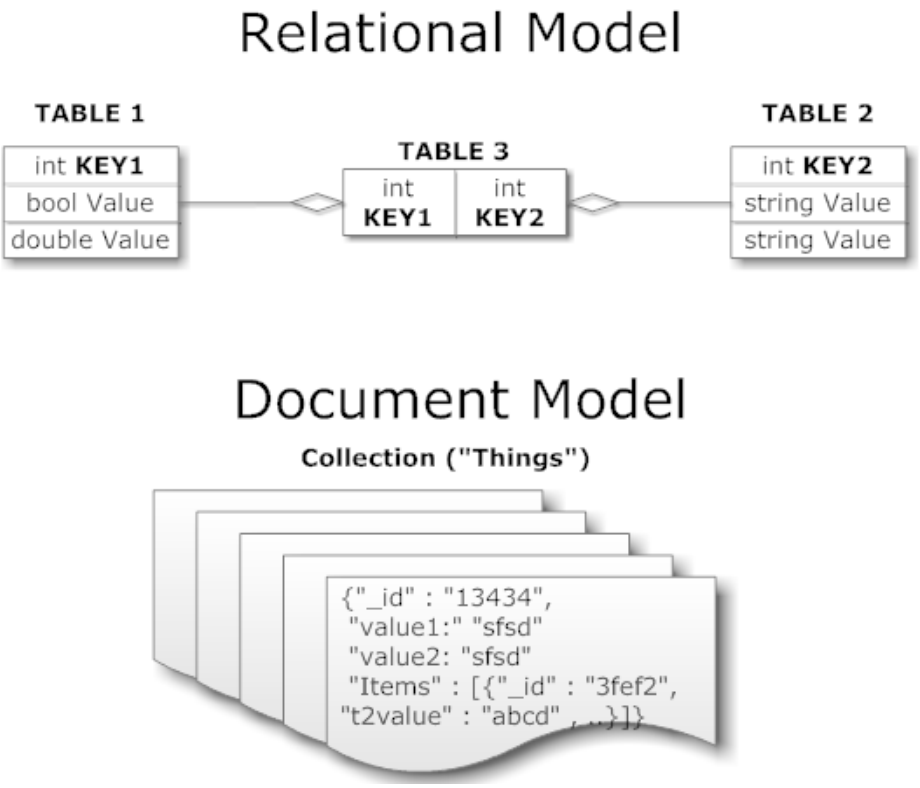


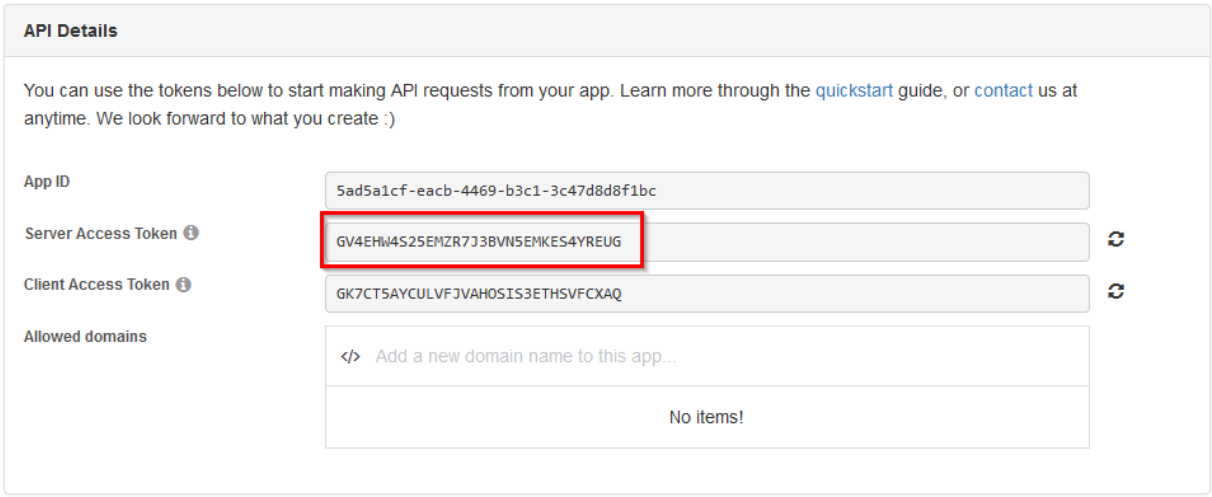
Figure 27 : Documents vs Relationnel

Evidemment The Computer Firm travaille avec cette base de données. Ce n'est malgré tout pas la raison principale de son utilisation, comme en témoignent les avantages cités ci-dessus. (MongoDB)

2.4 API/SDK de wit.ai

Le SDK fourni par wit.ai pour les serveurs Node.JS n'est pas excessivement grand. Il donne simplement accès aux outils permettant l'envoi d'un message pour analyse.

Le constructeur de wit.ai attend en trois paramètres en entrée, deux étant optionnels. Pour notre cas, nous n'avons fourni que le token wit.ai, utilisé pour retrouver notre projet de NLP sur wit.ai



Ce token récupéré, nous l'ajoutons simplement à notre serveur et pouvons instancier notre classe Wit appelée « client ». La bonne pratique veut que nous exportions certaines constantes dans un fichier config.js

config.js- Code source

```
1.  module.exports = {
2.      'WIT_TOKEN' : 'GV4EHW4S25EMZR7J3BVN5EMKES4YREUG',
3.      'MONGO_URL' : "mongodb://admin:chatmee1234_@ds161710.mlab.com:61710/chatmee",
4.  };
```

Tableau 4 : config.js

Pour la suite, nous instancions simplement une variable *config* qui nous permettra d'appeler notre constante token :

message_constroller.js- Code source

```
1.  const Message = require('./message_model');
2.  const Intent = require('../intent/intent_model');
3.  const intentController = require('../intent/intent_controller');
4.  // config WIT IA
5.  let Wit = null;
6.  let interactive = null;
7.  const WIT_TOKEN = config.WIT_TOKEN;
8.  try {
9.    Wit = require('../').Wit;
10.   interactive = require('../').interactive;
11. } catch (e) {
12.   Wit = require('node-wit').Wit;
13.   interactive = require('node-wit').interactive;
14. }
15. const client = new Wit({
16.   accessToken: WIT_TOKEN,
17. });
18. .
19. .
20. .
```

Tableau 5 : Configuration de wit.ai

Notre objet wit.ai est prêt, à partir de clients nous pourrons appeler chaque méthode présente dans le SDK Node.JS de wit.ai.

Le principal appel d'API qui sera consommé sera *message()*. Cette fonction demande en paramètre d'entrée, le message de l'utilisateur ainsi qu'un contexte optionnel. Une fois analysé par wit.ai, nous recevrons en retour les entities et intents relatifs à cette phrase.

Envoie	Retour
`client.message('Hello', {});`	`{ "_text": "Hello", "entities": { "greetings": [ { "confidence": 1, "value": "salutation_informel", "type": "value" } ] }, "msg_id": "0i58qdFiKHFI4vh93" }`

Tableau 6 : Envoie de message et résultat

Le retour va permettre à notre serveur d'orienter la réponse envoyée vers l'interface graphique, en guise de réponse à la question posée. (wit.ai, Node.js SDK for Wit.ai)

3 Développement de la solution

Le développement du chat concerne plusieurs couches de notre architecture. D'abord, nous portons notre attention sur la mise en place de la base de données puis, à la plaque tournante de notre application à savoir le serveur Node.JS, avant de terminer par la partie graphique.

3.1 Montage de la base de données

Afin de simplifier l'utilisation de MongoDB, nous avons installé un outil appelé MongoDB Compass. Cet outil permet de nous connecter à notre base et de visualiser les collections et leurs documents, comme le ferait SQL Management Studio.

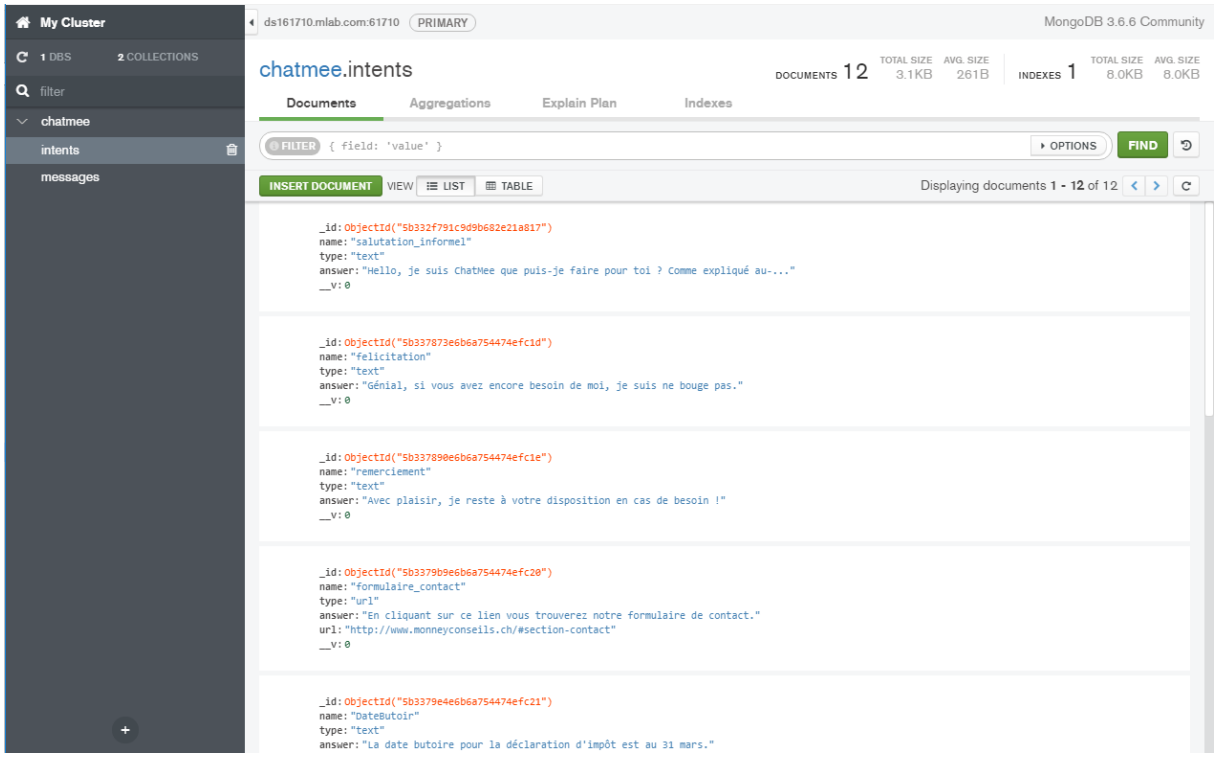


Figure 28 : Interface de MongoDB Compass

3.1.1 Modélisation de la base

Concernant la modélisation de la base de données, nous nous sommes limités à deux collections. Une contiendra nos intents avec leurs réponses respectives, l'autre nos messages avec toutes les données utiles tels que le texte écrit, son ID, l'ID de la conversation ou encore la date.

3.1.2 Intents et réponses respectives

La simplicité de gérer les réponses depuis la base de données a dicté le choix d'y stocker les intents. Pour rappel, après analyse, wit.ai nous renvoie un résultat similaire à celui-ci.

```
{
  "_text": "J'aimerais la liste des documents à founir pour ma déclaration d'impôt.",
  "entities": {
    "Liste_document": [
      {
        "metadata": "Voici les documents",
        "confidence": 0.99930111776096,
        "value": "get_document_informations"
      }
    ]
  },
  "msg_id": "0SvruY5HWuFxm8V9B"
}
```


Figure 29 : Résultat de l'analyse Wit.ai

Nous avons stocké dans notre base de données, tous les intents à savoir la *value* de l'image du dessus. Cela nous permet de comparer la value reçue de wit.ai et le nom de l'intent dans notre base. Dès qu'un match est trouvé, la réponse est envoyée depuis notre base de données vers le chat.

3.1.3 Sauvegarde des messages utilisateurs

La sauvegarde des messages utilisateurs est régie par le fait qu'ils nous laissent ou non stocker ces derniers. Dans le cas d'un refus de l'utilisateur, une logique business dans notre serveur va couper la sauvegarde des messages saisis.

Dans le cas contraire, chaque message envoyé est sauvegardé dans notre base avec un ID de conversation stocké dans les cookies. Comme nous n'avons pas de système d'authentification, ceci permet de garder une trace des conversations de l'utilisateur. Le choix de l'utilisateur relatif à la sauvegarde de ses données est également géré dans les cookies.

La RGPD étant arrivée en cours d'élaboration, les solutions de sauvegarde et suppression des messages ne figuraient pas dans l'analyse de base. Nous nous sommes adaptés afin de répondre au mieux à cette contrainte.

Toutes les requêtes vers notre base de données se font via *Mongoose* dans le serveur Node.JS. Elles seront détaillées dans la partie développement du serveur.

3.2 Structure et élaboration du serveur Node.JS

Pour rappel, le serveur Node.JS va recueillir les requêtes provenant de l'interface graphique, les traiter et les envoyer vers wit.ai ainsi que vers la base de données. Puis, recevoir en retour de wit.ai le résultat de l'analyse du langage et répondre en conséquence.

3.2.1 Architecture du serveur

Nous avons décidé l'utilisation d'une architecture conventionnelle à savoir le pattern MVC[8]. Celui-ci correspond exactement à nos besoins par sa facilité de maintenance ainsi que sa capacité à segmenter le code.

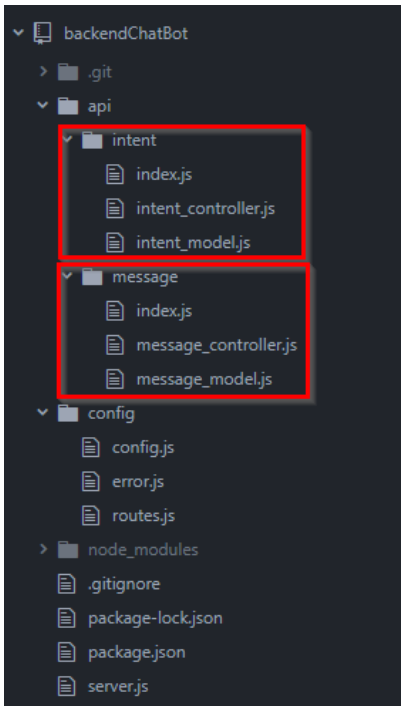


Figure 30 : Structure du projet

Pour des raisons logiques, les dossiers *message* et *intent* représenteront nos deux collections de la base de données portant le même nom. À l'intérieur de ceux-ci trois fichiers javascript représentant chacun, la vue, le controller et le modèle. Le modèle nous servira à créer nos objets intents et messages dans la base de données via le module mongoose (cf. point 3.1.2.2). Le controller contiendra les requêtes CRUD HTTP que nous utiliserons dans l'application. Pour terminer l'index fera le joint avec le fichier routes.js.

Dans le dossier « config », les fichiers portent des noms relativement explicites, le fichier config.js contient nos constantes, le fichier error.js nos erreurs et le fichier routes sera détaillé plus bas

3.2.2 Protocole HTTP

Le *Hypertext Transfer Protocol* plus connu sous l'abréviation HTTP, est un protocole de communication client-serveur développé pour le World Wide Web. Il est utilisé pour échanger toute sorte de données entre client HTTP et serveur HTTP.

Les principales requêtes HTTP utilisées dans le cadre d'une API RESTfull sont :

- GET
- POST
- PUT
- DELETE

Respectivement, récupérer, ajouter, mettre à jour ou supprimer une donnée. Ce sont ces mots clés que nous avons utilisés pour communiquer avec wit.ai et notre base de données.

[8] Model-Vue-Controller, est une architecture qui sépare l'application en trois composants logiques. Le modèle va servir aux appels base de données, le controller s'occupera de toute la partie logique et la vue aura pour rôle d'afficher les données dans l'interface graphique.

3.2.3 Les routes

Dans notre solution, nous avons mis en place un système de routing. Cette méthode nous permet non seulement d'effectuer nos requêtes aux bons endroits, mais aussi de les structurer.

Comme nous pouvons le voir dans la figure 25 ci-dessus, dans le dossier *config*, nous avons créé un fichier routes.js. Ce fichier assez court, est le « GPS » de notre application car il dirige les requêtes http vers les objets et tables correspondants.

route.js - Code source

```
1.  var express = require('express');
2.  var router = express.Router();
3.  var messageAPI = require('../api/message');
4.  var intentAPI = require('../api/intent');
5.  var converstionAPI = require('../api/conversation');
6.
7.  module.exports = function(app) {
8.
9.    app.use("/styles", express.static(__dirname + "/styles"));
10.   // api routes
11.   app.use('/api/messages', messageAPI);
12.   app.use('/api/intents', intentAPI);
13.   app.use('/api/conversations', converstionAPI);
14.
15.   // else by default redirect to home
16.   app.route('/*').get((req, res) => {
17.     res.json("Welcome to Chatmee api");
18.   });
19. };
```

Tableau 7 : routes.js

Nous avons une route qui récupère les styles de notre application, d'autres routes concernent les différents objets utilisés, à savoir les messages et les intents. Et enfin une route par défaut qui nous renverra un message simple.

Afin d'affiner les requêtes, nous avons créé une variable par collections à savoir *messageAPI* et *intentAPI* qui pointent vers leurs dossiers respectifs. Comme seul leur chemin est spécifié, ils vont automatiquement entrer dans le fichier index.js (cf. figure 25).

./api/message/index.js - Code source

```
1.  'use strict';
2.
3.  var express = require('express');
4.  var router = express.Router();
5.  var controller = require('./message_controller.js');
6.
7.  // Get functions
8.  router.get('/:id', controller.getMessageByConvId);
9.  router.get('/', controller.getMessages);
10.
11. // Put functions
12.
13. // Post functions
14. router.post('/', controller.userMessage);
15.
16. // Delete functions
17. module.exports = router;
```

Tableau 8 : index.js

Le principe ci-dessus est d'améliorer les routes en y affinant le chemin. Par exemple lorsque la route : *http:///localhost:6060/api/messages/TCF-3242j34230* sera appelée, notre fichier routes.js va simplement solliciter l'index.js correspondant à la première partie de l'URL, ici messages. L'index.js contrôle également la suite de la requête. Si le paramètre *id* est là, la fonction *getMessageByConvId* sera appelée, si aucun paramètre n'est saisi, la fonction *getMessages* sera appelée.

Cette méthode de travail offre une meilleure maintenance du code, elle permet de ne modifier que certaines parties de la route.

Dès lors, il nous est possible d'utiliser ces routes pour faire les requêtes HTTP relatées auparavant.

Nous utiliserons le programme Postman. Cet outil permet aisément de faire des requêtes vers notre serveur en utilisant nos routes. Une requête *GET* va demander au serveur l'envoi des données :

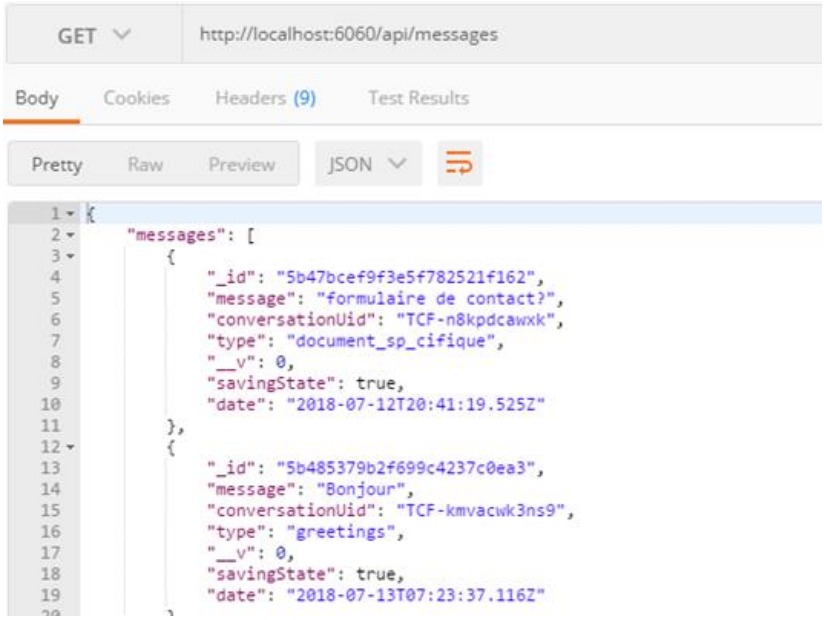


Figure 31 : Exemple de requête GET

L'exemple de requête ci-dessus nous retourne un tableau d'objet message contenant toutes informations comme le texte saisi par l'utilisateur, l'id de conversation ou encore la date. Toutes ces données proviennent de notre base de données mongoDB.

Dans le cadre d'une requête POST, requête inscrite qui va écrire dans notre base de données, il faut lui fournir de l'information à envoyer vers la base. Elle va donc attendre une entrée de l'utilisateur. Typiquement, lorsqu'un message est envoyé à travers le chat. Ce message va déclencher une requête qui va, de manière asynchrone, être sauvegardée dans la base de données et questionner wit.ai sur l'intention du message. Dès que wit.ai nous retournera le résultat, nous l'associerons à un intent de la base de données pour renvoyer la réponse à l'écran de l'utilisateur.

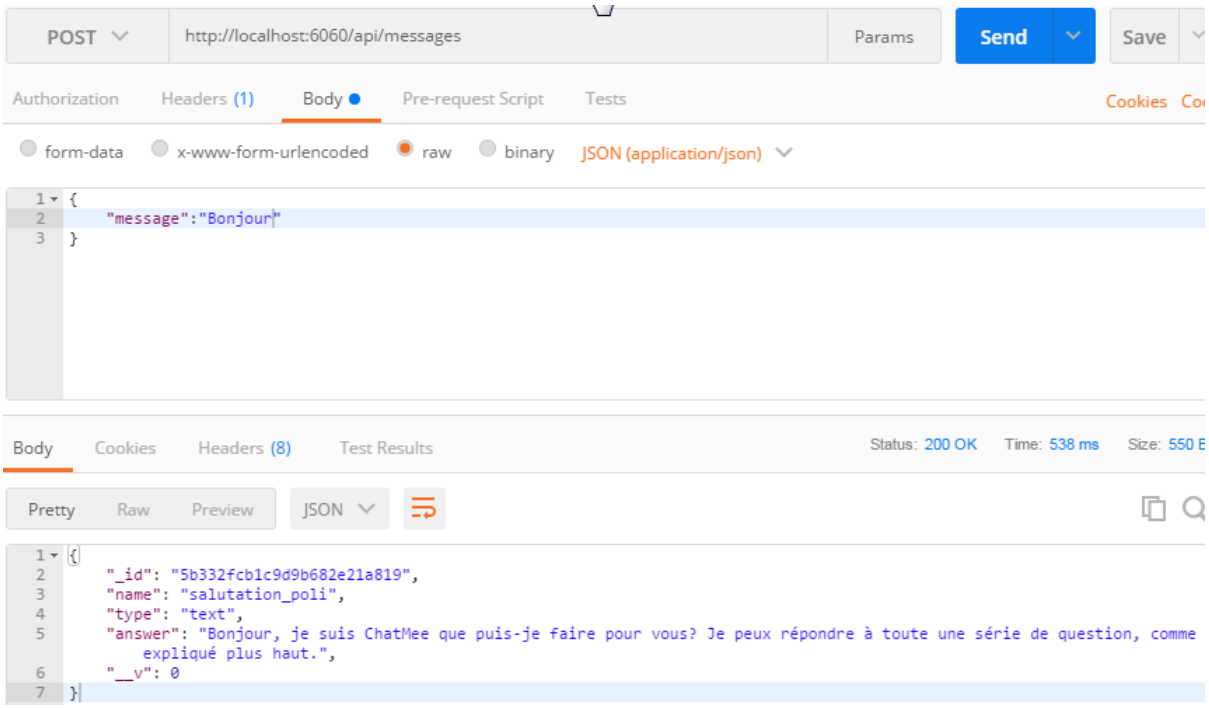


Figure 32 : Exemple de POST

L'écriture dans la base de données se fait au moyen des modèles définis et grâce à l'extension mongoose.

3.2.4 Les schémas mongoose

Les modèles créés utilisent les schémas de l'extension mongoose pour « mapper » l'objet reçu par notre serveur et le document de la collection dans MongoDB.

Ainsi le modèle message aura un schéma (dans notre cas nous avons utilisé la convention messageSchema).

./api/message/message_model.js - Code source

```
1.  // load the things we need
2.  var mongoose = require('mongoose');
3.
4.  // define the schema for our message model
5.  var messageSchema = mongoose.Schema({
6.    message: String,
7.    type : String,
8.    date : { type: Date, default: Date.now },
9.    savingState: { type: Boolean, default: "true" },
10.   conversationUid : String,
11. });
12.
13. // create the model for messages and expose it to our app
14. module.exports = mongoose.model('Message', messageSchema);
```

Tableau 9 : Modèle et schéma du message

Evidemment, pour des raisons de cohérence, les champs et types contenus dans le schéma doivent être identiques dans la base de données. En dernière ligne, nous exportons notre modèle pour l'utiliser dans notre controller.

Dans le controller, nous importons notre modèle message.

./api/message/message_constroller.js - Code source

```
1.  const Message = require('./message_model');
2.  const intent = require('../intent/intent_model');
3.  const intentController = require('../intent/intent_controller');
4.  // config WIT IA
5.  let Wit = null;
6.  let interactive = null;
7.  const WIT_TOKEN = config.WIT_TOKEN;
8.  try {
9.    Wit = require('../').Wit;
10.   interactive = require('../').interactive;
11. } catch (e) {
12.   Wit = require('node-wit').Wit;
13.   interactive = require('node-wit').interactive;
14. }
15. const client = new Wit({
16.   accessToken: WIT_TOKEN,
17. });
18. .
19. .
20. .
```

Tableau 10 : Import du modèle dans le controller

Grâce à notre modèle, il nous est possible de faire nos requêtes vers la collection messages de notre base. Mongoose fournit également des fonctions basiques comme *find()* ou *create()*.

./api/message/message_constroller.js - Code source

```
1.  // Get functions
2.  function getMessageByConvId(req, res) {
3.    var id = req.params.id;
4.    Message.find({"conversationUid":id}).then(messages => res.json({ messages }))
5.  }
6.
7.  function getMessages(req, res) {
8.    Message.find({}).then(messages => res.json({ messages }))
9.  }
```

Tableau 11 : Exemple de requête GET avec et sans condition

3.2.5 Wit.ai

3.2.5.1 Définitions des cas d'utilisation

Dans un premier temps, afin de déterminer quelles entities devaient être créées, il a été nécessaire de définir plusieurs cas d'utilisations et réponses, auxquelles notre *chatbot* pouvait rétorquer. Après une discussion concernant les demandes et besoins de la fiduciaire Monney Conseils, les cas d'utilisations suivants ont été retenus :

- Les salutations
- Les documents à fournir pour la déclaration d'impôt.
- Les documents spécifiques (formulaire de contact).
- Les prix et honoraires de prestations.
- Les dates butoirs de remise de la déclaration d'impôt.
- Les horaires d'ouvertures.

Afin de couvrir le maximum de demandes possibles ou lorsque notre *bot* ne peut répondre à la question, il a été décidé de renvoyer l'utilisateur vers le module de prise de rendez-vous déjà en production auprès de The Computer Firm. Ainsi un rendez-vous pourra être pris avec un spécialiste afin de répondre à des attentes plus spécifiques.

Afin de répondre à la demande de Monney Conseils, 8 entities ont été créées.

Entity	Description	Values
horaire LOOKUP STRATEGIES trait	User-defined entity	horaire
stop_saves_db LOOKUP STRATEGIES free-text & keywords	User-defined entity	stop sauvegarde
price_information LOOKUP STRATEGIES trait	User-defined entity	marie, celibataire, price_information
approbation LOOKUP STRATEGIES trait	User-defined entity	felicitation, remerciement
document_sp_cifique LOOKUP STRATEGIES trait	User-defined entity	formulaire_contact
dateButoir LOOKUP STRATEGIES trait	User-defined entity	DateButoir
Liste_document LOOKUP STRATEGIES trait	User-defined entity	get_document_informations
greetings LOOKUP STRATEGIES free-text & keywords	User-defined entity	salutation_informel, salutation_poli

Figure 33 : Entities créées pour le projet

3.2.5.2 Nos entities

L'entity relative au premier cas d'utilisation concerne les salutations. Afin que le *chatbot* soit capable de comprendre lorsqu'un utilisateur le salue, il a été nécessaire de lui apprendre ce que signifie une salutation. Afin de permettre au chat d'être le plus cordial possible, nous avons décliné les *greetings* en deux catégories : les salutations informelles et les salutations formelles. Ainsi, la réponse du chat pourra être adaptée en fonction de la salutation de l'utilisateur. S'il se présente avec des salutations plus formelles telles que « Bonjour » ou « Bonsoir », la réponse sera différente que s'il formule « Hello » ou « Salut ».

La création d'une simple entity, appelée « approbation » est déclinée en 2 intents, remerciements et félicitations. À l'usage, nous avons remarqué qu'il était naturel pour l'utilisateur d'exprimer un simple « merci » ou « super » lorsque le chat répondait à une question posée. De ce fait, notre *bot* sera capable de répondre et de comprendre ces remerciements. Dans un but plus technique, au niveau de l'entraînement de notre *bot*, ce genre d'entities sert également au développeur. Lorsque l'utilisateur déclenche cette entity, nous considérons que la réponse était satisfaisante, respectivement que l'entity choisie était la bonne. Il pourra valider l'entity choisie en cas de réponse de ce genre et, ajouter une phrase de plus à l'entraînement du *bot*.

Pour la suite, nous avons défini des entities plus orientées business. Elles serviront au chat pour la compréhension des questions que l'utilisateur posera en relation au domaine fiduciaire.

3.3 Application web et chat

L'analyse des différents services de NLP, l'installation d'un serveur capable de communiquer avec ce dernier, ainsi que la mise en place de la logique business étaient les objectifs principaux de notre travail pour The Computer Firm. C'est la raison pour laquelle l'interface graphique de chat devait être une simple vitrine afin de présenter le travail effectué. The Computer Firm ayant déjà leur design et interface graphique pour leurs modules, nous avons décidé d'utiliser une librairie de chat existante et open-source.

3.3.1 Choix de chat

Afin de mettre en place rapidement un interface web permettant l'interaction homme-machine, notre choix c'est porté sur un framework de *chatbot* appelé *react-simple-chatbot*. Développé par Lucas Bassetti, il permet une intégration rapide à un projet React.JS front-end.

Une part importante de développement a été réalisée par nos soins afin de le « customiser » pour qu'il soit conforme à nos besoins.

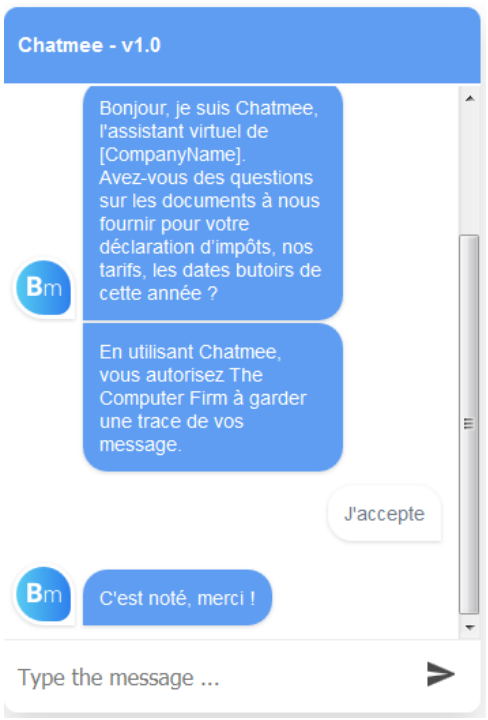


Figure 34 : Chatmee V1.0

Le choix de cette librairie nous permet d'avoir une vue rapide et interface de test sur tout le travail logique et analyses réalisées en amont.

Cependant, il est à préciser qu'une interface développée par nos soins aurait amélioré l'expérience utilisateur d'une part et nous aurait permis de l'orienter vers, par exemple, des choix de réponses d'autre part.

Nous aurions également pu imaginer fournir les documents sous format PDF ou encore améliorer notre logique de développement.

3.3.2 Architectures orientées composant React.JS

La mise en place de la partie graphique de notre chat utilise le framework de développement de Facebook, React.JS. Ce langage orienté composant le pattern MVC n'est pas optimal lorsqu'il est utilisé pour le serveur.

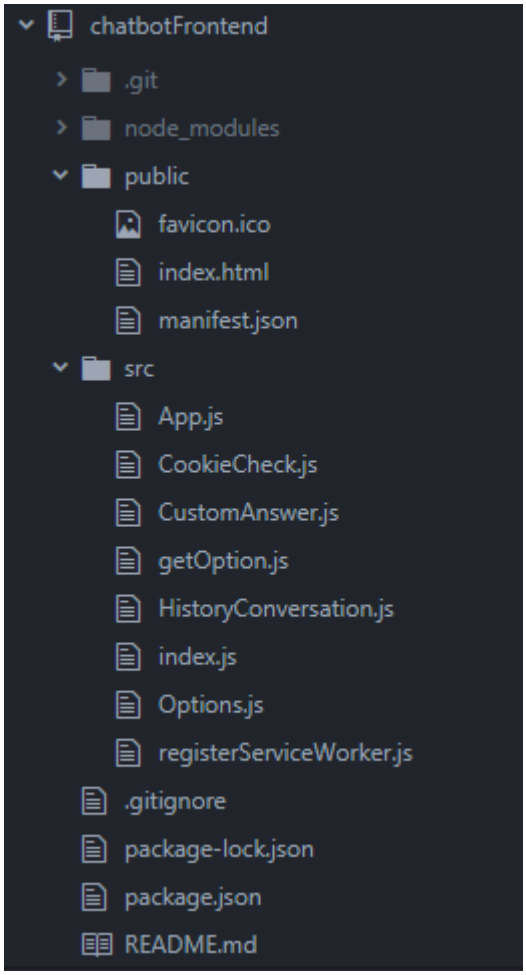


Figure 35 : Structure du front-end

Le point d'entrée et compostant parent de notre application est le fichier *App.js* dans le dossier *src*. Il contient entre autres le chat (via la librairie *react-simple-chatbot*), chaque *step* (récupération de l'historique des messages, contrôle des cookies, l'arrivée sur la page, premier message de bienvenue, entrée utilisateur…). Afin de pouvoir assurer une gestion sur mesure, nous avons créé des composants permettant de modifier les actions de ces *steps*. Les composants sur mesure créés sont également dans le dossier *src* et pourront être appelés dans *App.js*. *HistoryConversation.js* s'occupe d'appeler le serveur afin de transmettre à la base de données toutes les questions posées précédemment par l'utilisateur.

./src/App.js - Code source

```
1.  import React, { Component } from 'react';
2.  import { ThemeProvider } from 'styled-components';
3.  import ChatBot, { Loading } from 'react-simple-chatbot';
4.  import CustomAnswer from './CustomAnswer';
5.  import GetOption from './getOption';
6.  import CookieCheck from './CookieCheck';
7.  import HistoryConversation from './HistoryConversation';
8.
9.  var steps = [
10.   {
11.     id: '0',
12.     component: <HistoryConversation /> ,
13.     // asMessage: true,
14.     trigger: '1',
15.   },
16. .
17. .
18. .
19. renderSteps = () => {
20.     return (
21.     <ChatBot
22.       headerTitle={"Chatmee - v1.0"}
23.       // recognitionEnable={true}
24.       botAvatar={"http://bookmee.thecomputerfirm.com/img/favicon.png"}
25.       hideUserAvatar={true}
26.       steps={steps}
27.     />
28.   )
29. }
30.   render() {
31.     return (
32.       <ThemeProvider theme={theme}>
33.         {this.renderSteps()}
34.       </ThemeProvider>
35.     )
36.   }
37. }
```

Tableau 12 : Exemple de step et rendu du GUI

Ci-dessus, le premier *step* de l'application qui appelle *HistoryConversation*. Son importation a été faite en ligne quatre. La fonction en ligne 19 crée notre interface de chat, nous lui passons en attribut le tableau de *step* initialisé plus haut en ligne neuf. Le tout est rendu en ligne 30 vers le navigateur.

Chaque *step* nécessitant une implémentation supplémentaire a été réalisé sur ce modèle.

./src/HistoryConversation.js - Code source

```
1.  import React, { Component } from 'react';
2.  import ChatBot from 'react-simple-chatbot';
3.  import Cookies from 'universal-cookie';
4.  const cookies = new Cookies();
5.
6.  class HistoryConversation extends Component {
7.  constructor(props) {
8.    super(props);
9.    this.state = {
10.     conversation: null,
11.   };
12. }
13.
14. generateId() {
15.   return 'TCF-' + Math.random().toString(36).substring(2.5);
16. }
17.
18. componentWillMount() {
19. /*
20. Appel de la base de donnée qui récupère toute les conversation de l'UID
21. se trouvant dans le cookie
22. */
23. }
24.
25. renderConversation () {
26.   if(this.state.conversation) {
27.     var table = [];
28.     for (var i = 0; i < this.state.conversation.messages.length; i++) {
29.         table.push(<tr><td>{this.state.conversation.messages[i].message}</td></tr>);
30.       }
31.       return table;
32.   }
33.   else {
34.     return null;
35.   }
36. }
37.   render() {
38.     return (
39.       <div>
40.         <h3>Voici vos dernières questions</h3>
41.         <table>
42.           <tbody>
43.             {this.renderConversation()}
44.           </tbody>
45.         </table>
46.       </div>
47.     );
48.   }
49. }
50. export default HistoryConversation;
```

Tableau 13 : Exemple de composant logique

Le composant ci-dessus est un exemple de ce que nous avons développé afin de personnaliser chaque étape de notre chat. Pour des raisons de lisibilité, le code des lignes 19 à 22 a été remplacé par un simple commentaire.

C'est également au sein de ces composants logiques que nous appelons les routes de notre serveur afin de récupérer ou envoyer des données.

./src/CustomAnswer.js - Code source

```
componentWillMount() {
  const self = this;
  const { steps, previousStep} = this.props;
  const search = steps.search.value;
  const options = steps.search
  const endpoint = encodeURI('http://localhost:6060/api/messages');
  var headers = new Headers();
    var message = {
        "message": search,
        "savingState": cookies.get('chatmee-save'),
        "conversationUid": cookies.get('chatmee-uid'),
    };
  // Tell the server we want JSON back
  headers.set('Content-Type', 'application/json');
  var fetchOptions = {
     method: 'POST',
     headers,
     body: JSON.stringify(message)
   };
   var responsePromise = fetch(endpoint, fetchOptions);
   // 3. Use the response
   responsePromise
  // 3.1 Convert the response into JSON-JS object.
  .then(function(response) {
    return response.json();
  })
  .then(function(jsonData) {
    self.setState({
      result: jsonData.answer,
      type: jsonData.type,
      lastIntent: jsonData.name,
      url:jsonData.url
    })
    self.props.triggerNextStep();
  });
}
```

Tableau 14 : POST d'un message

Dans cette fonction se passe la création de notre objet message, afin qu'il contienne toutes les données voulues. À partir de la ligne 22, nous récupérons déjà la réponse envoyée par le serveur afin de l'afficher à l'écran.

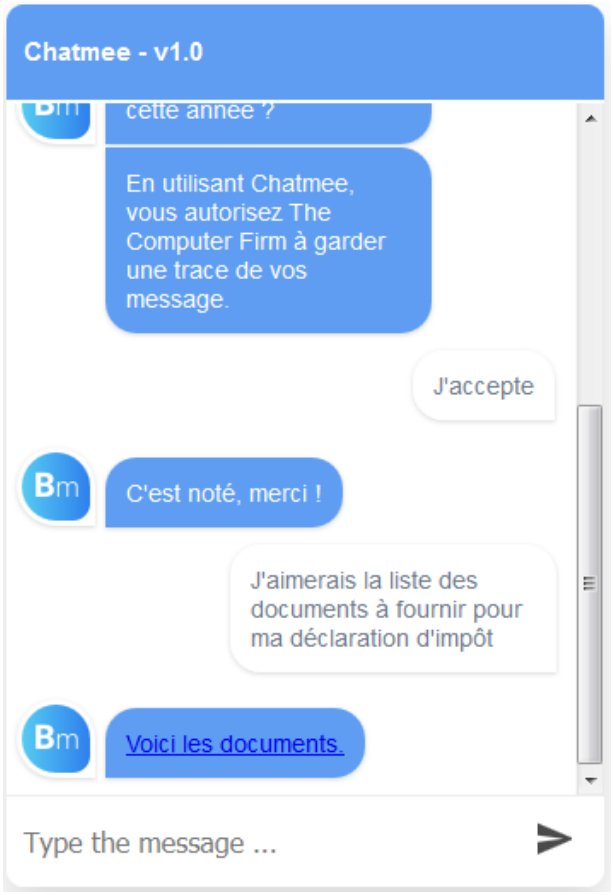


Figure 36 : Exemple de question / réponse

Pour finir, dans le cas ou notre bot ne comprendrait pas la question, il a été prévu de rediriger l'utilisateur vers l'outil de prise de rendez-vous développé par The Computer Firm.

C'était une condition de The Computer Firm, mettre en relation leurs produits afin de les faire marcher aux mieux ensemble.

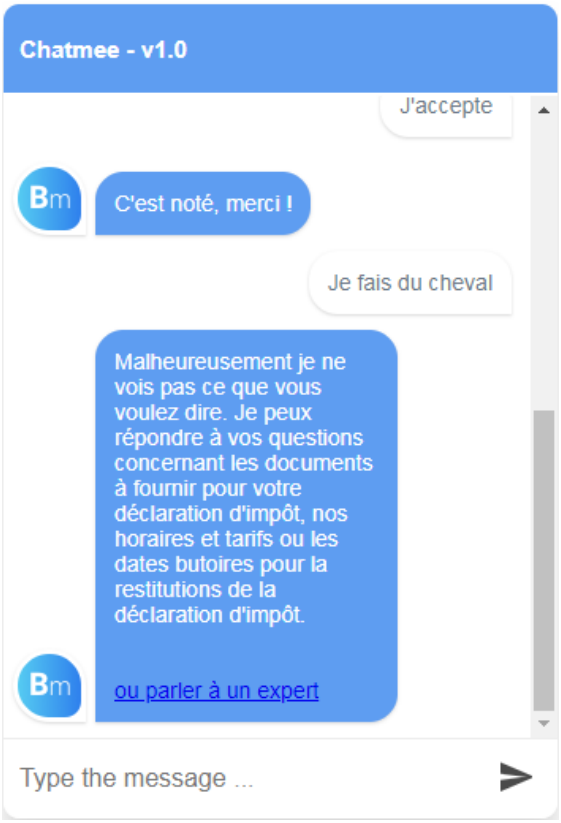


Figure 37 : Le bot n'a pas compris notre question

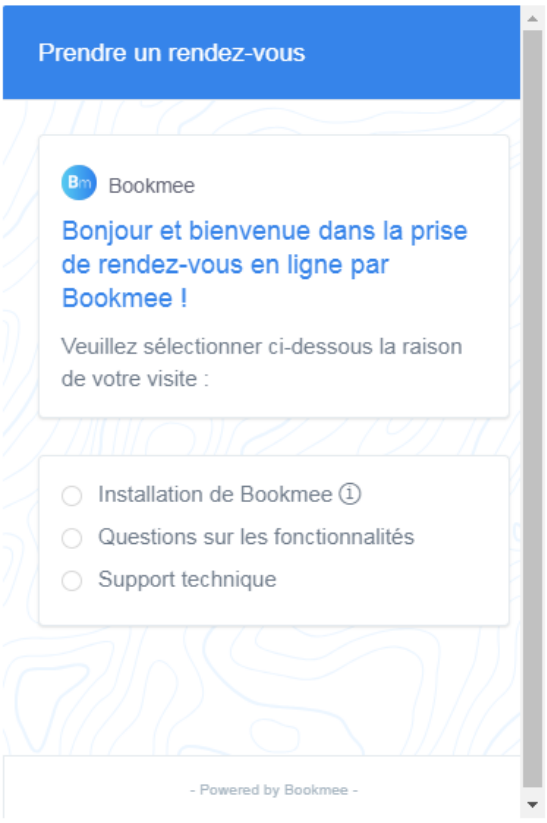


Figure 38 : Module de prise de rendez-vous

4 Méthodologie

La méthodologie de travail discutée en collaboration avec Monsieur Michael Schumacher est la méthode SCRUM. Afin d'être le plus structuré possible, nous utilisons un fichier Excel préparé par Monsieur Cotting dans le cadre du cours sur les méthodologies Agile.

Adopter cette méthode nous a permis de constater une évolution claire dans l'application. L'un des principes du SCRUM est d'avoir un livrable fonctionnel à chaque itération.

4.1 L'agilité et SCRUM

Lors de la première séance avec Monsieur Michael Schumacher, nous avons décidé de nous rencontrer une fois par mois. C'est sur cette base que nous avons décidé de découper chaque sprint.

Pour être le plus cohérent possible avec la méthode SCRUM quelques adaptations ont été nécessaires. Tout d'abord, The Computer Firm porte le rôle de Product Owner. Puis, nous avons retiré les séances d'estimation des *User Stories.* L'étudiant étant seul à travailler sur les tâches, il estime seul chaque *User Story*.

Nous avons élaboré un *product backlog* qui correspond à une feuille de route regroupant toutes les fonctionnalités de notre application, leur estimation de temps, leur priorité ou encore leurs critères d'acceptance. Chaque fonctionnalité étant appelée *User Story*.

US Nr.	Theme		User Stories		Acceptance Criteria	Priority	Status	Story Points	Sprint	US accepted (done done)	MoSCoW
1	Preparation	Developper	Prepare the work environment	I can start working in good conditions	Environment working	1000	Done	5	0	Done	Must
15	Development	Developper	have a deeper analysis about the data right between dialogflow and wit	I really can choose the best NLP service	Get enought information to make the best choice	995	Done	8	1	Done	Must
2	Preparation	Developper	Create a Git	to share my code with every stackholders	Everybody have access to the git like they should	990	Done	1	0	Done	Must
3	Preparation	Developper	Set up the React boilerplate that will be used through the project	have a good structure and a scalable application	The boilertplate have to run properly without errors	980	Done	8	0	Done	Must
4	Preparation	TCF	discuss with our client to ask them to test and try our application	we can have a better understanding of the needs	The client said is ok and agree for the colaboration	970	Done	3	0	Done	Must
5	Preparation	TCF	Do mockups	The client can see what we're doing and give us feedbacks	The client have to aggreed the mockups	960	Done	8	0	Done	Must
6	Preparation	Developper	Do some research on chatbots	have a better understanding and use the		950	Done	13	0	Done	Must
7	Preparation	Developper	An analysis grid of different chatbots	We can chose the best techologies	validation by school and TCF	940	Done	18	0	Done	Must
9	Development	Developper	select the best NLP service to analyze the text send by the end user	the client can have the most optimized chatbot service	Every stakeholder agreed on the choice	930	Done	2	1	Done	Must
11	Development	Client	have a bot that answer the question about the document to provide	the client can have a clear idea of what document he will have to provide to us to fill the tax return	The answer is provided correctly	920	Done	21	2	Done	Must

Figure 39 : Première partie du product backlog

Ci-dessus une partie du *product backlog* avec pour chaque *User Story*, ses différentes valeurs.

4.2 Sprint 0

Sprint planning :

Dans ce sprint 0 nous avons fait un premier sprint planning qui planifie les analyses et recherches à effectuer. Ci-dessous les *User Stories* sélectionnées pour ce premier sprint.

User Stories			Acceptance Criteria	Priority	Status	Story Points
AS	I want to	In order to				
Developper	Prepare the work environment	I can start working in good conditions	Environment working	1000	Done	5
Developper	Create a Git	to share my code with every stackholders	Everybody have access to the git like they should	990	Done	1
Developper	Set up the React boilerplate that will be used through the project	have a good structure and a scalable application	The boilertplate have to run properly without errors	980	Done	8
TCF	discuss with our client to ask them to test and try our application	we can have a better understanding of the needs	The client said is ok and agree for the colaboration	970	Done	3
TCF	Do mockups	The client can see what we're doing and give us feedbacks	The client have to aggreed the mockups	960	Done	8
Developper	Do some research on chatbots technologies	have a better understanding and use the best technologies of the products		950	Done	13

Figure 40 : User Stories concernant le sprint 0

Sprint review :

Pour ces *User Stories*, les tâches suivantes ont été planifiées et effectuées.

Task Name	Resp.	Initial Estimate		Time during Sprint
Prepare work environment	Samuel	2		2
Create a git	Samuel	0.5		0.5
Setup boilerplate react	Samuel	1		1
Discuss with the client	Michaël	1		0.5
Do mockups	Michaël	2		2
Do research on chatbots services	Samuel	20		15
Do the analysis grid	Samuel	20		22
Select the NLP service	Team	2		1
	Total	*49*		
	Remaining	*49*		*44*

Figure 41 : Tâche relative au sprint 0

Un total de 49 heures a été planifié. En réalité nous avons mis 44 heures pour terminer ces tâches.

Au total, ce sprint aura duré du 21.02.2018 au 10.04.2018. La semaine de vacances pascales n'étant pas considérée comme jours de travail.

La remarque générale concernant ce sprint était bonne. Les recherches et les premières analyses étaient satisfaisantes. Le professeur ainsi que le Product Owner demandent une spécification de certaines *User Stories*. Effectivement, modéliser des *User Stories* pour une solution telle qu'un *Chatbot* n'est pas si simple. Celle-ci pouvant souvent être abstraite.

4.3 Sprint 1

Sprint planning :

Pour ce sprint 1, les *User Stories* suivantes ont été choisies. Décision prise de faire un dernier cas d'utilisation, afin de choisir entre les deux derniers concurrents quant au service de NLP. Ensuite, il a été fixé de réaliser une première version basic du chat avec un des scénarios définis par The Computer Firm.

US Nr.	Theme	User Stories			Acceptance Criteria	Priority	Status	Story Points
		AS	I want to	In order to				
15	Development	Developper	have a deeper analysis about the data right between dialogflow and wit	I really can choose the best NLP service	Get enought information to make the best choice	995	Done	8
9	Development	Developper	select the best NLP service to analyze the text send by the end user	the client can have the most optimized chatbot service	Every stakeholder agreed on the choice	930	Done	2
13	Development	User	have a bot that have some greetings and goodbye queries	the bot will be more human like and credible		890	Done	13
17	Development	User	have a bot that can answer questions about deadlines and date	the client have the informations for the tax return	The deadlines and the date are correctly display and the	850	Done	13
23	Development	Developper	Create a training dataset	we can train our bot	Precision graph should have improved value	790	in progress	8

Figure 42 : User Stories du sprint 1

Sprint review :

Pour ces *User Stories*, les tâches suivantes ont été planifiées et effectuées.

Task Name	Resp.	Initial Estimate
do research on creating a node.JS RESTful server	Samuel	5
research about bot training	Samuel	1
create entities and values	Samuel	2
Create a basic node.JS server	Samuel	3
integrate wit.ai SDK	Samuel	5
Use wit.ai SDK	Samuel	6
First use case with .interactive from wit.ai SDK	Samuel	6
create a route to post wit.ai message for further usability	Samuel	2
send message with the post request to wit.ai and get the answer back	Samuel	4
interprate the answer from wit.ai and answer in consequence	Samuel	3
train the document usecase for accurate answer	Samuel	2
add some entities to have a more human like bot	Samuel	4
Review with TCF	Samuel	3
Administration and rapport writting	Samuel	1
first user testing to see if the bot is doing correctly	Samuel	2
	Total	*49*

Figure 43 : Tâche relative au sprint 1

Un nouveau total de 49 heures a été planifié. Nous avons cette fois respecté cette estimation et terminé les tâches en 49 heures.

Au total ce sprint a duré du 10.04.2018 au 15.05.2018. Comme nous avons pris en considération les jours de cours, un total de 11 jours de travail a été agendé.

Un livrable relativement performant a été transmis. Les premières questions pouvaient être posées au *bot* en ligne de commande avec leurs réponses respectives. C'était déjà une amélioration de certaines *User Stories* suggérées.

4.4 Sprint 2

Sprint planning :

Pour ce sprint, nous avons sélectionné uniquement trois *User Stories* conséquentes. Le but était d'ajouter une interface graphique à notre chat, afin de pouvoir présenter un état relativement visuel. L'intention était également d'y ajouter tous les scénarios de questions / réponses.

US Nr.	Theme	User Stories			Acceptance Criteria	Priority	Status	Story Points
		AS	I want to	In order to				
11	Development	Client	have a bot that answer the question about the document to provide	the client can have a clear idea of what document he will have to provide to us to fill the tax return	The answer is provided correctly	920	Done	21
12	Development	User	have a bot that answer the question about the prices we provides.	the client can predict his budget for the tax return	The bot is capable of asking question to have better	910	Done	21
19	Development	User	Have GUI where client can ask theire questions	Users can have better user experience		900	Done	23

Figure 44 : User Stories concernant le sprint 2

Sprint review :

Pour ces *User Stories*, les tâches suivantes ont été planifiées et effectuées.

Task Name	Resp.	Initial Estimate
Do some deeper research about GDPR	Team	4
Do a Use case Diagramm in order to have a better idea of the questions / answer modelisation	Samuel	4
Integrate all use cases to a React.JS front-end	Samuel	18
Update the backend to make it work with the new front-end	Samuel	18
do a training data set for our bot	Samuel	4
Documentation and research about the best practices with the use of entities and intent	Samuel	6
Préparation et sprint review	Samuel	1
	Total	55

Figure 45 : Tâche relative au sprint 2

Un total de 55 heures a été planifié et 55 heures ont été effectuées

Pour l'ensemble, ce sprint aura duré du 15.05.2018 au 13.06.2018. Cette fois, 12 jours de travail ont été nécessaires, toujours en temps partiel.

Des retours positifs sont ressortis de ce sprint avec un front-end fonctionnel qui permet de visualiser le travail effectué.

4.5 Sprint 3

Sprint planning :

Pour ce dernier sprint, les *User Stories* restantes sont agendées. Idéalement, il faut ajouter la base de données derrière notre chat, adapter le backend en conséquence et faire les adaptations nécessaires pour s'approcher de la nouvelle loi RGPD en vigueur.

US Nr.	Theme	User Stories			Acceptance Criteria	Priority	Status	Story Points	Sprint
		AS	I want to	In order to					
8	Development	User	have a bot that can take meeting with Monney using Bookmee	the client can have a fast meeting with Monney	Getting good feedbacks and aggrement from the client	880	To do	21	3
20	Development	User	Improve to UI by integrating the type of answer	The user can finish his CTA		820	Done	13	3
21	Development	User	My user to get a question asking if they're ok with the use of the data	We complient with the GDPR	The information is correctly displayed and the user can accept or not	810	Done	21	3
22	Development	User	to choose wether or not I save my conversation in the database	We are complient with data protection	The conversation is saved, or not in the database according to the user answer	805	Done	13	3
24	Development	User	I want to be able to delete my data	we're compliant with GDPR		710	To do	5	3
16	Development	User	have a bot than can give back some specific document from a list	the client will have all informations he need to fill his tax return		700	in progress	13	3

Figure 46 : User Stories concernant le sprint 3

Sprint review :

Pour ces *User Stories*, les tâches suivantes ont été planifiées et effectuées.

ID US.task	Task Name	Resp.	Initial Estimate
0.1	Setup Mlab	Team	2
	Setup mongoDB with Mlab	Samuel	2
0.3	Create collections in MongoDB	Samuel	3
0.4	Create models in the backend	Samuel	3
0.5	Create post query to add new intent + test	Samuel	6
0.6	Create post message query (save in DB + send the message to Wit.ai) + test	Samuel	8
0.7	Create get message query	Samuel	2
0.8	create get by name query for intent	Samuel	2
0.9	Create an UID for the user	Samuel	2
1	Add the UID to a cookie when the user connect to the chat	Samuel	2
1.1	Create a boolean cookie to check if the user want his message to be save in DB	Samuel	2
1.2	Adapt intent model and add the type variable	Samuel	1
1.3	Conditionnal rendering to check if the user has a cookie, if not create one	Samuel	8
1.4	Conditionnal rendering that check if the boolean cookie is true or not	Samuel	3
1.5	Adapt our chatbot introduction message if the cookie boolean is checked or not, change introduction message	Samuel	8
1.6	Condition that check the intent type in order to return a link, list or text	Samuel	3
1.7	Manage the case when wit.ai return a null intent	Samuel	4
		Total	*61*

Figure 47 : Tâche relative au sprint 3

Pour ce travail, 61 heures sont planifiées. Un dépassement de 12 heures est constaté. Les problèmes avec la librairie de chat choisie pour la partie web en sont la raison.

Au total ce sprint aura duré du 14.06.2018 au 13.07.2018. Cela correspond à 11 jours de travail.

En fin de sprint, il est décidé d'effectuer encore 3 *User Stories* dont 2 étant un *must.* L'intervalle existant entre la fin du sprint 3 et la restitution du travail le 8 août sera utilisé pour les réaliser. Il n'y a plus de séance planifiée jusqu'à la fin du travail.

4.6 Difficultés rencontrées et leurs solutions

Durant tout le projet, de nombreuses difficultés sont apparues. Tout d'abord, l'élaboration du *Product Backlog* a été relativement compliquée car les fonctionnalités d'un *chatbot* sont abstraites. Une importante recherche intellectuelle est à la base de cette représentation.

Lors du développement de la solution. L'interface web du chat nous a posé beaucoup de problèmes. N'étant pas le sujet principal du travail, nous avons dû en tenir compte néanmoins. Si ce genre de solution venait à se représenter, nous partirions avec une interface sur mesure. Cela permettrait une définition plus simple des entities et intents, liée à une meilleure expérience utilisateur.

Pour finir, il n'a pas toujours été simple de communiquer avec les associés de The Computer Firm. Ces derniers étant régulièrement à l'étranger et dans des fuseaux horaires éloignés, nous avons dû adapter nos horaires pour parler de l'état du développement de la solution

5 Conclusion

En clôture de ce rapport, nous présentons un rapide bilan des activités. Nous évoquons ensuite les idées et suggestions rencontrées en cours de développement, conséquence de l'expérience apportée par ce rendu. Finalement, nous donnons quelques ouvertures d'optimisation et d'amélioration du produit mis en place.

5.1 Bilan final

Le bilan de ce travail s'avère très positif, aussi bien d'un point de vue technique que professionnel. Les différentes parties prenantes ont eu l'occasion de travailler ensemble sur un projet qui se veut novateur et très actuel. La HES-SO Valais, The Computer Firm et moi-même avons pu confronter nos idées, apprendre ensemble, et nous enrichir sur un sujet vaste et très intéressant.

L'architecture MEAN et toutes ses couches sont très intéressantes et m'ont personnellement donné envie de trouver un emploi dans ce domaine afin de pouvoir en apprendre encore plus.

Techniquement, nous avons appris énormément de toutes les technologies et framework actuels. Cette expérience sera certainement plus que bénéfique pour le futur.

5.2 Rétrospectives et améliorations futures

Grâce à l'expérience acquise durant ces 6 mois d'études, quelques recommandations nous viennent à l'esprit.

Suite à l'expérience acquise sur wit.ai, il serait intéressant de reprendre des tests et cas d'utilisation avec DialogFlow. Avec cela voir si un impact significatif interviendrait par exemple sur le code. Comme expliqué en début de rapport, DialogFlow est un excellent service de NLP et mérite un certain crédit.

Afin d'exploiter toute la puissance de ce genre d'outil, l'interface graphique doit être plus qu'une simple vitrine servant à l'interaction entre l'homme et la machine. Un chat entièrement sur mesure, avec plusieurs fonctionnalités de choix ou de questions dynamiques pourrait être extrêmement intéressant. Son impact sur la définition des entities et intents n'en serait que plus simple.

Il est également agréable de savoir que le produit va être utilisé par The Computer Firm et que l'analyse et la première version effectuée sera un jour mise en production.

Voici selon nous, les principales suggestions à envisager à moyen terme.

6 Références

Bathelot, B. (2018, avril). *Définition : ChatBot*. Récupéré sur definitions-marketing: https://www.definitions-marketing.com/definition/chatbot/

ChatFuel. (s.d.). *Getting started*. Récupéré sur ChatFuel: https://chatfuel.com/

Classrooms, O. (2013, 10 30). *Les requêtes HTTP*. Récupéré sur openclassrooms: https://openclassrooms.com/fr/courses/1118811-les-requetes-http

Digitalfirefly. (2013, 10 09). *What are good uses for Node.JS*. Récupéré sur digitalfireflymarketing.com: https://digitalfireflymarketing.com/our-blog/what-nodejs-good/

docs, M. w. (s.d.). *Callback function*. Récupéré sur mozilla.org: https://developer.mozilla.org/en-US/docs/Glossary/Callback_function

Eaton-Cardone, M. (2017, juin 17). *The good, the bad and the ugly of chatbots*. Récupéré sur venturebeat.com: https://venturebeat.com/2017/06/17/the-good-the-bad-and-the-ugly-of-chatbots/

econocom. (2018, janvier 22). *Data Protection Officer (DPO) : la fonction très recherchée pour le RGPD*. Récupéré sur Emedia: https://blog.econocom.com/blog/data-protection-officer-dpo-la-fonction-tres-recherchee-pour-le-rgpd/

ExpressJs. (s.d.). *Express*. Récupéré sur ExpressJs: https://expressjs.com/

Facebook. (s.d.). *React*. Récupéré sur Reactjs: https://reactjs.org/

Google. (s.d.). *Basics about Dialogflow*. Récupéré sur DialogFlow: https://dialogflow.com/docs/getting-started/basics

IBM. (s.d.). *Watson assistant*. Récupéré sur IBM: https://www.ibm.com/cloud/watson-assistant

McElhearn, K. (2016). *Siri-ous mistakes*. Récupéré sur MacWorld: https://www.macworld.com/article/3106088/ios/sirious-mistakes.html

Microsoft. (2017, 07 05). *About Language Understanding*. Récupéré sur Microsoft Azure: https://docs.microsoft.com/en-us/azure/cognitive-services/LUIS/Home

Microsoft. (s.d.). *Les chatbots s'implifient l'accès aux contenus et améliore l'expérience utilisateur*. Récupéré sur experiences.miscrosoft.fr:

https://experiences.microsoft.fr/business/intelligence-artificielle-ia-business/chatbot-arte/

MongoDB. (s.d.). *What is MongoDB*. Récupéré sur mongodb: https://www.mongodb.com/what-is-mongodb

Mongoose. (s.d.). *Mongoose*. Récupéré sur Mongose: https://mongoosejs.com/

Morales, E. (2016, 06 29). Récupéré sur Embengineering.com: https://medium.embengineering.com/in-react-js-data-flows-in-one-direction-from-parent-to-child-841103ed3aed

Pandorabot. (s.d.). *Build intelligent conversationnal agents*. Récupéré sur https://home.pandorabots.com/en/

Patel, P. (2018, april 18). *What exactly is Node.Js*. Récupéré sur freecodecamp: https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5

Perisetla, K. (2016, décembre 18). *What is an "intent" in chatbot design?* Récupéré sur Quora: https://www.quora.com/What-is-an-intent-in-chatbot-design

Popper, B. (2016, may 27). *Ray Kurweil is building a chatbot for Google*. Récupéré sur theverge: https://www.theverge.com/2016/5/27/11801108/ray-kurzweil-building-chatbot-for-google

Rebot.me. (s.d.). *Create a chatbot*. Récupéré sur http://rebot.me/

Reply.ai. (s.d.). *Create you own chatbot*. Récupéré sur https://www.reply.ai/

Roey, G. V. (2017, avril 27). *Making your own chatbot with api.ai*. Récupéré sur blog.craftworkz: https://blog.craftworkz.co/making-your-own-chatbot-with-api-ai-21be5444a2df

Rossignol, J. (2017, 10 18). *Apple says "hey siri"*. Récupéré sur Macrumors: https://www.macrumors.com/2017/10/18/apple-explains-how-hey-siri-works/

sitepoint. (2014, 04 17). *An Introduction to the MEAN Stack*. Récupéré sur sitepoint.com: https://www.sitepoint.com/introduction-mean-stack/

Suisse, C. (2018). *Le règlement général de l'UE sur la protection des données*. Récupéré sur edoeb.admin.ch: https://www.edoeb.admin.ch/edoeb/fr/home/documentation/bases-legales/Datenschutz%20-%20International/DSGVO.html

Tutorialspoint. (s.d.). *MongoDB - Overview*. Récupéré sur Tutorialspoint: https://www.tutorialspoint.com/mongodb/mongodb_overview.htm

vectorstock. (s.d.). *vectorstock*. Récupéré sur vectorstock: https://www.vectorstock.com/royalty-free-vector/cute-smiling-funny-robot-chat-bot-vector-19006513

Wikipédia. (2018). *Reconnaissance d'entités nommées*. Récupéré sur Wikipédia: https://fr.wikipedia.org/wiki/Reconnaissance_d'entit%C3%A9s_nomm%C3%A9es

Wikipédia. (2018). *Traitement automatique du langage naturel*. Récupéré sur Wikipédia: https://fr.wikipedia.org/wiki/Traitement_automatique_du_langage_naturel

Wikipédia. (s.d.). *Kit de développement*. Récupéré sur Wikipédia: https://fr.wikipedia.org/wiki/Kit_de_d%C3%A9veloppement

Wikipédia. (s.d.). *Test de Turing*. Récupéré sur Wikipédia: https://fr.wikipedia.org/wiki/Test_de_Turing

Wit.ai. (2018). *Recipes*. Récupéré sur wit.ai: https://wit.ai/docs/recipes

Wit.ai. (s.d.). *Frequently Asked Questions*. Récupéré sur Wit.ai: https://wit.ai/faq

wit.ai. (s.d.). *Node.js SDK for Wit.ai*. Récupéré sur github: https://github.com/wit-ai/node-wit

wit.ai. (s.d.). *Using Wit.ai programmatically*. Récupéré sur github.com: https://github.com/wit-ai/wit-api-only-tutorial

7 Annexes

Bot name	Luis.ai	Wit.ai	Api.ai / DialogFlow	Watson	Chatfuel	Rebot.me	Pandorabots	Reply.ai
Documentation	Good documentation, it's microsoft tools. They provide starter guide. Need to see what's avalable for more advanced tools	Documentation on the basics is provided. Explanations of the API and tools is present too. Need to see for further use cases	Same level of documentation that both of previous bot. The basics are present, the documentation API provide good queries	Good documentation, API and all of his call are provided, a getting started guide too. Default queries are provided too	Basic documentation, It's like a CMS for chatbot	Not many information are avalable	Documentations is correct, everything needed to start is provided. API documentation is provided too	Not that easy to find, video provided on theire website, you've to write them to get answer back
Point : 8	6	6	6	7	4	2	6	3
Community	Found some guide and articles on reddit. The bot is part of the 5 best services	wit.ai got form thant 1300 stars on github for the nodejs bot. It's also part of the best 5 bots services. Used by over 120,000 developers	Stackoverflow, 560 stars for the nodejs on github, google groups avalable too	Pretty good community, It's one of the best known NLP services, with big amount of data to train theire bots	Everything passing via facebook -> annoying. The community is pretty huge (17 millions users globaly)	didn't found many things about the community. This bot won't fit for business or more advanced features. It may works for a limited domain	250'000 registered developpers, and more thant 300'000 chatbots created	no github, not much information find on reddit
Point : 5	3	5	4	5	2	2	5	2
Licences and credential	Free until 10'000 messages, then 0.37€ / 1000 messages	free, including commercial use	standard edition is free, enterprise has a fee -> https://dialogflow.com/pricing/	They provide a free and a premium services -> https://www.ibm.com/cloud/watson-assistant/pricing	Standard version is free - Pro -> 30$ / month	You need to contact the to get information about the pricing	Free up to 1000 messages / month then 0.0025$ per message up to 100'000 messages / month	write them -> hello@reply.ai
Point : 5	2	5	5	5	5	0	4	2
Features	Can be connected to Cortana, use the Bing and Cortana. API. Key concepts are used with Intents, utterance, entities and NLP	Intents, self-build entities. Exemple of substring of the message, location, date, time. Sentiment analysis is avalable too. self-build entities are train by wit.ai dataset. Developers can also train the bot. Context are also avalable	Machine learning to understand the user. Samples are avalable like translate, weather, small talk, time. Intents, training, NLP, context, NLU via Agents, Rich Messages, events, Analytics	provides Intents, entities and dialog. Build on neural network, (with more than one billion words from wikipedia)	Creating bot easely and without coding. AI is managed by a dashboard. You simply add rules to manage it	Creating a basic chatbot, train him with Q&A. Like previous one a CMS for basic chatbot	Using AIML(Artificial Intelligence Mark-up Language) which is a XML dialect. Also using a technologie called A.L.I.C.E ("Artificial Linguistic Internet Computer Entity"). Using categories, recursion, synonyms, keywords, targeting, conditionning, contexts.	provides a platform where you can build your bot, not much code needed, uses, wit.ai and api.ai for NLP services.
Point : 3	3	3	3	3	2	1	3	2
Integration / programming language	C# SDK Python SDK Node JS SDK Android SDK	Ruby Python Node JS Through HTTP API	**V1 is avalable on** Android, iOS, ordova, HTML, JavaScript, Node.js, .NET, Unity, Xamarin, C++, Python, Ruby, PHP (community supported), Epson Moverio, Botkit, Java" **V2 is avalable on** Java NodeJS Python	Node SDK Java SDK Python SDK iOS SDK Unity SDK	Only plugins are avalable : Google Search, Bing Search, JSON API...	no code needed	SDKs Developped by PB: Java Node.js Python SDKs Developped by the community: Ruby PHP Go	you can integrate to your backend
Point : 5	5	5	5	5	2	2	5	3
Language	English, French, Italian, German, Spanish, Brazilian Portuguese, Japanese, Korean and Chinese.	Albanian, Arabic, Azerbaijani, Bengali, Bosnian, Bulgarian, Burmese, Catalan, Chinese, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, French, Georgian, German, Greek, Hebrew, Hindi, Hungarian, Icelandic, Indonesian, Italian, Japanese, Korean, Latin, Lithuanian, Macedonian, Malay, Norwegian, Persian, Polish, Portuguese, Romanian, Russian, Serbian, Slovak, Slovenian, Spanish, Swahili, Swedish, Tagalog, Tamil, Thai, Turkish, Ukrainian and Vietnamese.	Brazilian Portuguese, Chinese (Cantonese), Chinese (Simplified), Chinese (Traditional), English, Dutch, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish, Ukrainian.	English, Japanese	Albanian, Arabic, Azerbaijani, Bengali, Bosnian, Bulgarian, Burmese, Catalan, Chinese, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, French, Georgian, German, Greek, Hebrew, Hindi, Hungarian, Icelandic, Indonesian, Italian, Japanese, Korean, Latin, Lithuanian, Macedonian, Malay, Norwegian, Persian, Polish, Portuguese, Romanian, Russian, Serbian, Slovak, Slovenian, Spanish, Swahili, Swedish, Tagalog, Tamil, Thai, Turkish, Ukrainian, Vietnamese	Azerbaijani, Dutch, English, French, Italian, Portuguese, Russian, Romanian, Spanish, Tagalog, Turkish	Said multilingual, but not more	-
Point : 7	7	7	7	1	7	7	0	0
Range of action	Allow developpers to create smart application. The bot can learn alone, understand what are the client's needs and answer according to them	Create application and devices that you can talk or text to	Used by Mercedes, Giorgio Armani. Suitable for business and complexe bot	Healthcare, business, big companies, complex bots	Used for minor chatbot and simple needs and integration	Small private people who wants to dynamise theire website or app	Mitsuku is a bot 3 times winner of a prestigious prize at a Turing Test, he's considered as the best worldwide humanlike chatbot. This should be enough to prove that the bot can go into complexe interaction	Work with nike or starbucks. Easy integration to every chat engine (telegram, facebook...) can be integrate to a custom app too
Point : 1	1	1	1	1	1	1	1	1
Data right	Not much said, you can disable logs to privacy needs. You can also delete data utterance used to train your bot. All datas are delete permanently 60 days after the account is deleted.	You have the ownership of your data but you agree to let us use them to improve Wit. If your app is private, your data (intents, entities, expressions, logs) will be accessible only by you and the developers you decide to share your app with. If your app is open, you agree to share some data. More info on the privacy policy.	ruled by the google API terms of services	Check the privacy term -->https://www.ibm.com/privacy/us/en/?link=fig-priv-usen	they may use data to customize our content. not much more informations, read the privacy policy	nothing found about this section	not much informations about this section	They don't provide many information about privacy --> https://docs.google.com/document/d/1OFho3tN9BWYDY7LOwCDgTcQrowzgI3IHLD2W2gs2vm4/pub?embedded=true
Point : 3	2	2	2	2	1	0	1	1
More details	https://docs.microsoft.com/en-us/azure/cognitive-services/LUIS/Home	https://wit.ai/faq	https://dialogflow.com	https://www.ibm.com/cloud/watson-assistant	https://chatfuel.com/	http://rebot.me/	https://www.pandorabots.com/	https://www.reply.ai/
Scoring	29	34	33	29	24	15	25	14
Target score	37	37	37	37	37	37	37	37

8 Déclaration de l'auteur

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- Monsieur Professeur Dr. Michael Ignaz Schumacher
- Monsieur Raphaël Sculati
- Monsieur Fabien Dubosson

Samuel Coppey le 08.08.2018