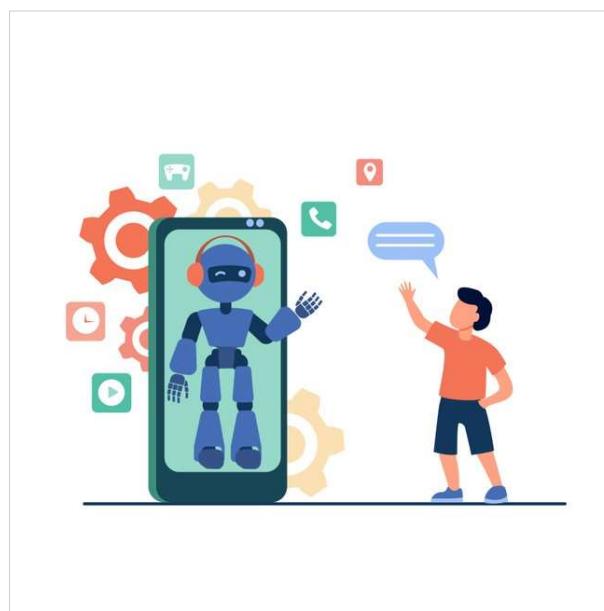


SYSTÈME DE RÉSERVATION INTERACTIF VIA CHAT-BOT



Projet de bachelor présenté par

Fabio RODRIGUES DOS SANTOS

**Informatique et systèmes de communication avec orientation en
Informatique Logicielle**

Août, 2024

Professeur-e HES responsable

Niklaus EGGENBERG

Légende et source de l'illustration de couverture :

Illustration d'un utilisateur conversant avec un Chatbot depuis son téléphone.

Source : <https://shopifieddnacom/wp-content/uploads/2021/07/chatBot.jpeg>

TABLE DES MATIÈRES

<i>Remerciements</i>	1
<i>Résumé</i>	2
<i>Liste des acronymes</i>	3
<i>Liste des illustrations</i>	4
<i>Liste des tableaux</i>	6
<i>Introduction</i>	7
<i>Chapitre 1 : Définition de Chatbot</i>	10
1.1. Un Chatbot, qu'est-ce ?	10
1.2. Trois composants d'un Chatbot	11
a) L'interface	11
b) Le moteur de traitement de données	11
c) Le moteur de réponses	11
1.3. trois types de Chatbots	12
a) Les Chatbots à Menus	12
b) Les Chatbots par Tokens	15
c) Les Chatbots “intelligents”	17
<i>Chapitre 2 : Étude des solutions existantes</i>	22
2.1. Agenda.ch	22
2.2. Klara.ch	23
2.3. Meetme.io	24
2.4. Comparaison à ce projet	25
<i>Chapitre 3 : Fondements Théoriques</i>	26
3.1. Natural Language Processing	26
a) Natural Language Understanding	31
b) Natural Language Generation	32
3.2. TAPAS	35
a) Transformers	36
b) BERT	37
c) TAPAS : Fonctionnement	40
3.3. Chatbots par NLU	44
a) Les intents	44
b) Les Entités	45
c) Les Stories	45
d) Les Actions	46
e) Les Règles	46
f) Les Slots	47
g) Les Lookup Tables	47

h)	Les Forms	47
i)	Les Pipelines	48
Chapitre 4 : Outils existants		49
4.1.	Rasa	49
a)	Rasa X	49
b)	Rasa Open Source	52
c)	Rasa Pro	54
d)	Duckling	56
4.2.	Chatterbot	58
4.3.	Microsoft Bot Framework.....	58
4.4.	NLTK	59
4.5.	OpenNLP	59
Chapitre 5 : Prototypages et Évaluations.....		61
5.1.	Prototypes NLP	61
a)	Java	61
b)	Python.....	62
5.2.	Prototype TAPAS	64
a)	Données de test.....	65
b)	Résultats	66
c)	Conclusion	67
5.3.	Prototype Rasa	68
a)	Données d'entraînement	69
b)	Exemple de conversation.....	71
c)	Conclusion	71
5.4.	Réflexion sur l'ensemble des Prototypes	72
Chapitre 6 : Conception du projet.....		74
6.1.	Technologies utilisées.....	74
a)	Docker	74
b)	FastAPI + Uvicorn	75
c)	Express JS.....	76
d)	Socket.IO.....	76
e)	Vite + React.....	76
f)	Rasa	77
g)	Duckling	77
h)	Postgresql	77
i)	SQLAlchemy	78
j)	API Telegram	78
6.2.	Architecture Globale	79
6.3.	Déroulement du projet.....	80
6.4.	Chatbot Rasa	82
6.5.	Base de données	85
6.6.	API	87

6.7. Interfaces du Chatbot	91
a) Telegram.....	91
b) Application Web	92
Chapitre 7 : Réalisation du projet	94
 7.1. Implémentation Chatbot	94
a) Architecture de fichiers Rasa.....	95
b) Stories	96
c) Forms	99
d) Rules.....	99
e) Action Server.....	100
 7.2. Implémentation base de données	102
 7.3. Implémentation API	103
a) Mise en place API.....	104
b) Routes	105
c) Connexion à la base de données.....	106
d) Interaction avec la base de données.....	107
e) Calendrier Google	108
f) Fichier ICS.....	109
g) Opérations asynchrone	110
 7.4. Implémentation Interfaces	111
a) Chatbot Web - Frontend	111
b) Chatbot Web – Backend.....	114
c) Telegram.....	115
Chapitre 8 : Résultats	117
 8.1. Chatbot Web.....	118
 8.2. Chatbot Telegram	125
 8.3. Commentaires sur les résultats	126
Conclusion	128
Problèmes rencontrés	129
Améliorations Possibles	131
Références documentaires	133

Dédié à mon père et ma mère qui ont tout fait pour m'aider à en arriver là et de continuer
à persévérer malgré tout.

REMERCIEMENTS

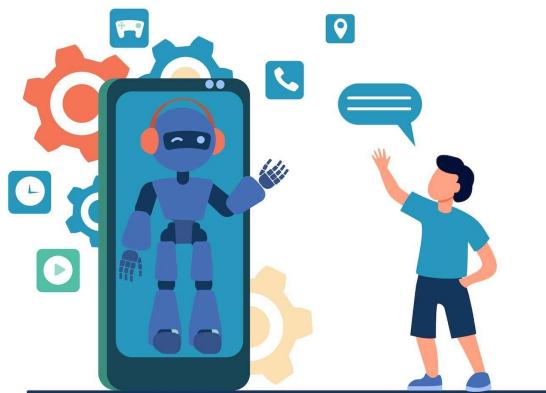
Je remercie avant toute chose mon professeur attiré, M. Niklaus Eggenberg, m'ayant suivi tout au long de la réalisation de ce projet et mes camarades avec qui j'ai fait face à maintes épreuves tout au long de ces trois années. Tout particulièrement mes camarades : Yann Rüegger, Yona Walther, Juliano Souza Luz et Dimitri Bavarel.

Des remerciements à un bon nombre de professeurs à l'HEPIA qui de par leurs cours et pour quelques-uns l'aide particulière fournie à la compréhension de divers sujets dont certains sont mentionnés dans ce rapport. Je remercie par ailleurs mon ami Français Romain F. et mon amie Polonaise M. pour l'aide fournie à des fins de tests de l'application.

Finalement, je tiens à remercier mes proches pour tout le soutien apporté avant et pendant la rédaction de ce mémoire.

RÉSUMÉ

La majorité des moyens de réservation de nos jours sont soit au travers de formulaires sur une page web, d'appels téléphoniques ou de prises de rendez-vous en personne. Il existe parfois la possibilité de réserver par téléphone mais ces cas sont rares et nécessitent l'intervention d'un opérateur humain. C'est donc dans ce cadre-ci que le projet de réalisation d'un Chatbot intervient ; mettant au passage à disposition divers canaux de communications tels que Telegram et dans le but d'émuler une discussion par messages, une interface web. Le but premier étant de fournir un nouvel outil qui permettrait de donner un entre-deux aux clients ne souhaitant pas passer par des pages web mais ne voulant pas non plus passer des appels téléphoniques. Or, afin de mener à bien ce projet, il a été dans un premier temps nécessaire de comprendre quels composants sont nécessaires au fonctionnement d'un Chatbot et de par ce modèle, quels types de Chatbots existent et surtout comment sont-ils implémentés. Un concept important étant le langage naturel et comment il est traité dans un contexte d'automatisation d'un opérateur conversationnel afin que ce dernier soit capable de comprendre ce qu'il lui est dit et ainsi de pouvoir répondre correctement à sa requête. Afin de trouver la technologie la plus pertinente à la réalisation de ce Chatbot, une multitude de prototypes concernant divers aspects du bot ont été réalisés et étudiés afin de finalement pouvoir réaliser un Chatbot conversationnel de réservation selon les observations finales effectuées.



Candidat-e :

RODRIGUES DOS SANTOS FABIO

Filière d'études : ISC

Professeur-e(s) responsable(s) :

EGGENBERG NIKLAUS

Travail de bachelor soumis à une convention de stage en entreprise : non

Travail soumis à un contrat de confidentialité : non

LISTE DES ACRONYMES

NLP Natural Language Processing

NLU Natural Language Understanding

NLG Natural Language Generation

I.A. Intelligence Artificielle

UX User Experience

POS Part-of-Speech

NER Named Entity Recognition

LLM Large Language Model

TAPAS TAble PArSing

SDK Software Development Kit

CALM Conversational AI with Language

API Application Programming Interface

LUIS Language Understanding Intelligent Service

RNN Recurrent Neural Network

ORM Object-Relational Mapping

CORS Cross-Origin Resource Sharing

LISTE DES ILLUSTRATIONS

Illustration 1: Graphique de la relation entre les trois composants de base	10
Illustration 2: Graphique d'un Chatbot à menus	12
Illustration 3: Arbre de décision d'un Chatbot à menus	14
Illustration 4: Graphique d'un Chatbot de règles	15
Illustration 5: Exemple de sélection de règle pour Chatbot par tokens	16
Illustration 6: Graphique d'un Chatbot Intelligent	18
Illustration 7: Pyramide de la difficulté technologique et UX selon le type de Chatbot	21
Illustration 8: Séquence de traitement NLP	27
Illustration 9: Représentation de la relation NLP, NLU et NLG.....	31
Illustration 10: Cas exemples du moteur google utilisant BERT	37
Illustration 11: Représentation de l'entrée du modèle BERT	39
Illustration 12: Entrée du modèle TAPAS	40
Illustration 13: Schéma de fonctionnement du modèle TAPAS.....	42
Illustration 14: Interface contenant le menu de Rasa X	49
Illustration 15: Page de gestion des modèles avec Rasa X	51
Illustration 16: Example de conversation parmi l'historique des conversations	51
Illustration 17: Architecture Rasa	53
Illustration 18 : Illustration de comparaison entre une Machine Virtuelle et des conteneurs Docker	75
Illustration 19: Diagramme d'architecture général du projet.....	79
Illustration 20 : Diagramme en arbre de la structure Docker Compose du projet	80
Illustration 21 : Liste de labels sur le GIT du projet	81
Illustration 22 : Capture d'écran d'un issue board sur Gitlab	81
Illustration 23 : Diagramme représentant une interaction quelconque entre l'utilisateur et le bot	82
Illustration 24 : Diagramme d'un échange portant sur la demande de ressource	83
Illustration 25 : Diagramme du fonctionnement du Chatbot Rasa.....	84
Illustration 26 : Diagramme Entité-Association de la Base de données	85
Illustration 27 : Capture d'écran du bot Telegram	91
Illustration 28 : Capture d'écran d'une conversation avec le bot web.....	92
Illustration 29 : Capture d'écran de sélection de choix lors d'une réservation.....	93
Illustration 30 : Arbre de fichiers principaux du bot Rasa	95
Illustration 31 : Maquette des composants du Frontend	112
Illustration 32 : Capture d'écran de début de conversation	118
Illustration 33 : Captures d'écran des deux débuts de conversation	119
Illustration 34 : Captures d'écran d'une réservation.....	120
Illustration 35 : Capture d'écran d'une création d'événement Google Calendar	121
Illustration 36 : Capture d'écran d'un événement Google Calendar après réservation	122

Illustration 37 : Captures d'écran démontrant le changement dynamique de date	123
Illustration 38 : Captures d'écran démontrant le changement dynamique d'heure.....	123
Illustration 39 : Captures d'écran d'utilisation des radio boutons.....	124
Illustration 40 : Captures d'écran d'une réservation sur Telegram	126

Références des URL

URL01	https://www.ovationc xm.com/blog/3-kinds-chatbots-youll-meet
URL02	https://www.javatpoint.com/nlp
URL03	https://datasolut.com/natural-language-processing-vs-nlu-vs-nlg-unterchiede-funktionen-und-beispiele/
URL04	https://www.youtube.com/watch?v=ioGry-89gqE
URL05	https://humboldt-wi.github.io/blog/research/information_systems_1920/bert_blog_post
URL06	http://arxiv.org/abs/2004.02349
URL07	https://www.datocms-assets.com/30881/1608730961-screenshot-2019-05-21-at-12-46-37.png
URL08	https://www.datocms-assets.com/30881/1608730956-screenshot-2019-05-21-at-12-44-11.png
URL09	https://www.datocms-assets.com/30881/1608730966-screenshot-2019-05-21-at-12-48-34.png
URL10	https://rasa.com/docs/rasa/arch-overview
URL11	https://www.netapp.com/blog/containers-vs-vms/

LISTE DES TABLEAUX

Tableau 1: Représentation du tableau fourni en entrée de TAPAS	41
Tableau 2: Tableau de dimensions de Duckling.....	58
Tableau 3: Extrait de Terrains.csv	65
Tableau 4: Contenu de Horaires.csv.....	66
Tableau 5 : Tableau des routes disponibles sur l’API	90

Références des URL

URL01 <https://github.com/facebook/duckling>

INTRODUCTION

De nos jours et comme il a été le cas par le passé, que ce soit pour aller chez le médecin ou au restaurant ; il est commun de placer des réservations. Or pour ce faire il est d'usage de passer un coup de fil ou dans d'autres cas de placer la réservation en se rendant au dit lieu du rendez-vous. Aujourd'hui, il est possible de placer ses réservations directement depuis de nombreux sites web. Qu'ils soient possédés par le particulier chez qui on souhaite placer rendez-vous ou par une autre entité chargée de gérer l'aspect de placement de réservations et/ou rendez-vous. Ces sites web permettant de prendre rendez-vous se présentent très régulièrement sous la forme de boutons cliquables et de quelques champs à remplir telle que la date du rendez-vous, le nom/prénom et autres informations. Cependant, il est bien plus rare qu'une alternative à cela existe et ne nécessitant pas de devoir se rendre sur une page internet quelconque. Le projet qui a été réalisé a pour optique d'apporter un autre moyen de placer une réservation se trouvant à l'intersection d'un appel téléphonique et un simple formulaire cliquable : Un Chatbot servant à prendre des rendez-vous depuis des applications de messageries tel que Telegram ou alors pouvant être aisément inclus dans un site web existant ; permettant de rendre plus fluide le processus de prise de rendez-vous.

Ce Chatbot s'apparente plus au premier abord à un composant applicatif auquel il est possible de fournir du texte et en recevoir en sortie. Bien que le but premier soit de fournir un Chatbot avec lequel il sera possible de converser sur une application de messagerie, il est tout à fait possible de mettre en place une section Chatbot sur un site quelconque et d'y lier le Chatbot, le rendant de ce fait facile à mettre en place sans avoir à se soucier de l'interface graphique de l'unique nécessité du Chatbot étant un champ textuel mis à disposition à l'utilisateur.

Une des raisons primaires de ma décision de réaliser ce projet étant qu'à présent, en 2024, de nombreux progrès ont été réalisés dans le secteur du Machine Learning et de

l'I.A. et bien que de nombreuses compagnies aient commencées à doucement mais sûrement à implémenter des technologies diverses en termes d'I.A. ou Machine Learning. Il y a encore un grand nombre de secteurs dans lesquels la présence de ces dernières se fait encore discrète ou simplement car peu ont considéré faire usage de ces technologies dans ces derniers. Le secteur en question présentement est celui du Service Client et plus spécifiquement l'assistance à réservation et prise de rendez-vous. Je trouve tout particulièrement pertinent l'exploration de cette voie qu'est l'usage de ces technologies pour la réalisation d'un Chatbot pour prendre rendez-vous car moi-même devant de temps à autre prendre rendez-vous, il est bien pénible pour bon nombre de gens de passer des coups de fil, moi inclus. C'est pour cela que fournir une solution qui soit modulaire et ne nécessitant pas d'architecture particulière que simplement un moyen d'incorporer une boîte de dialogue ou champ textuel quel que soit la plateforme choisie faciliterait grandement son utilisation par les particuliers de tout secteurs et qui fournirait une alternative viable aux appels et sites web.

Ce rapport de Bachelor fait suite au travail de semestre en Informatique et Systèmes de Communications à L'Haute Ecole du Paysage, d'Ingénierie et d'Architecture de Genève et a pour objectif d'expliciter les divers concepts proéminents dans le projet, l'étude des divers technologies existantes ayant servi à la réalisation des prototypes puis par la suite du projet, l'étude de divers produits existant similaire, montrer des exemples concrets de leur utilisation à l'aide de quelques prototypes et finalement détailler comment le Chatbot a été réalisé.

La réalisation de ce rapport s'est effectuée de manière régulière chaque semaine lorsque le temps le permettait et un état des lieux fut régulièrement effectué à l'aide d'entretiens hebdomadaires avec M. Niklaus Eggenberg. Afin de réaliser ce rapport, l'outil Git a servi à versionner et stocker les divers prototypes réalisés pour tester les technologies ainsi que le journal de bord tenu à jour régulièrement pour faire compte du travail effectué à chaque session de travail. D'ailleurs une copie du rapport a été régulièrement mise à

jour sur Google Drive. Le stockage s'est effectué sur la plateforme Gitlab mise à disposition par la HES et sur un projet GitEdu¹ possédé par M. Niklaus Eggenberg. L'outil Google Docs fut d'une grande aide lors de l'écriture d'ébauches, de prises de notes, d'explications préemptives des divers technologies recherchées et pour la formulation de mes idées concernant le travail à réaliser par la suite.

La principale méthode de recherche de sources fut grâce au moteur de recherche Google et occasionnellement Google Scholar. Une grande partie de mes sources sont divers sites de blogs et articles tels que certains venant de Medium.com, IBM ou encore Analytics Vidhya expliquant certains concepts détaillés tout au long de ce rapport. Néanmoins une certaine quantité d'articles de recherche notamment celui du modèle TAPAS par des ingénieurs de Google qui ont servi à comprendre certaines technologies.

Concernant la structure du présent rapport, elle se compose de cinq chapitres. Le premier définit de manière concrète et conceptuelle ce qu'est un Chatbot. Le deuxième porte sur l'étude des solutions existantes de réservation et prises de rendez-vous se trouvant sur le marché et comment elles comparent à celle proposée ici. Le troisième est une énonciation et explicitation des divers concepts piliers composant les divers structures de Chatbot énoncées au chapitre précédent. Le quatrième a pour but de faire un état des nombreuses technologies existantes et utiles pour la réalisation de ce projet. Le cinquième chapitre sert à présenter les quelques prototypes réalisés de certaines des technologies énoncées précédemment ainsi que les diverses réflexions les concernant. Le sixième défini de manière conceptuelle comment le Chatbot a été réalisé. Finalement le septième comment il a été implémenté.

¹ [HTTPS://GITEDU.HESGE.CH/NIKLAUS.EGGENBER/2324_RODRIGUESDOSANTOS](https://gitedu.hesge.ch/niklaus.eggenber/2324_RODRIGUESDOSANTOS)

Chapitre 1 : DÉFINITION DE CHATBOT

Avant même de commencer à parler technologies, il est important d'expliciter concrètement ce qu'est un Chatbot.

1.1. UN CHATBOT, QU'EST-CE ?

Les Chatbots sont des applications disponibles sous une multitude de formes et plateformes et ayant pour but premier d'engager dans une conversation avec un utilisateur afin de tenter au mieux de répondre à quelque requête que ce soit. Les Chatbots font affaire à un grand nombre de requêtes variées comme fournir des renseignements, fournir un service ne nécessitant pas d'opérateur humain, faire office de service client afin de tenter de résoudre tout éventuel problème dans la limite des capacités du Chatbot (le cas échouant, le bot peut rediriger l'utilisateur vers un opérateur humain), récolter du feedback, et bien d'autres usages encore. Bien que des Chatbots existent dans un nombre incalculable de formes et structures, il y a néanmoins une manière de décomposer n'importe quel bot en trois éléments distincts

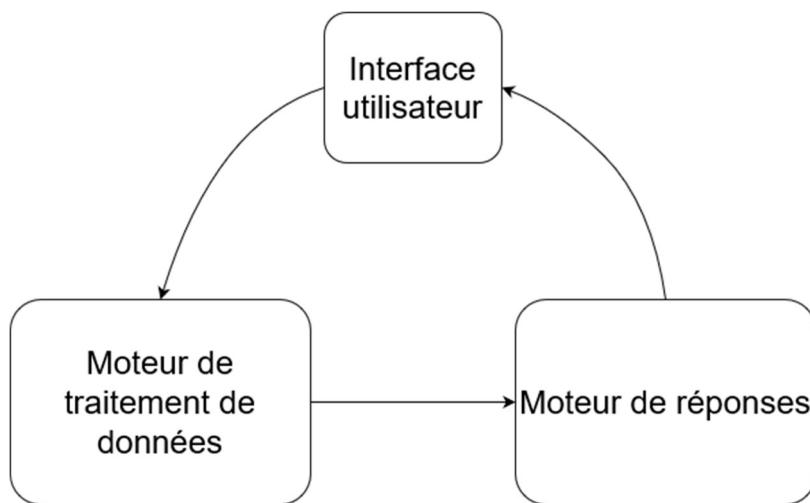


Illustration 1: Graphique de la relation entre les trois composants de base

Source : Rodrigues dos Santos Fabio

1.2. TROIS COMPOSANTS D’UN CHATBOT

a) L’INTERFACE

Aucun Chatbot ne pourrait avoir la dénomination de *Chatbot* s’il ne possédait pas d’interface avec laquelle l’utilisateur peut interagir. Au plus simple, une interface pourrait être une simple ligne de commande de terminal et au plus poussé, une fenêtre de dialogue ayant l’apparence d’un logiciel de messagerie comme sur nos smartphones. L’essentiel ici est que l’interface mise à disposition comporte au moins une zone de saisie textuelle ou autre support de communication tel que des boutons cliquables par exemple. C’est au travers de ce support que l’utilisateur fera parvenir ses diverses requêtes au bot pour traitement ultérieur.

b) LE MOTEUR DE TRAITEMENT DE DONNÉES

Une fois que l’utilisateur a exprimé sa demande au travers d’une des interfaces disponibles, cette demande sous forme de texte va être immédiatement acheminée au moteur de traitement de donnée. Les requêtes utilisateurs dans le cas où le canal de communication est un champs de texte se présentent sous la forme de suites de caractères. Ce qui est tout à fait compréhensible pour un humain mais pas pour une machine, c’est donc pourquoi un traitement se doit d’être appliqué au texte utilisateur. Le texte se présentant sous formes de phrases sera verra le plus souvent mis sous la forme de mots-clés ou instructions pouvant eux être compris par la machine.

c) LE MOTEUR DE RÉPONSES

Une des dernières étapes du cycle d’échange entre l’utilisateur et le bot est le passage des données précédemment traitées dans le moteur de réponse. Son rôle est de faire sens des divers mots-clés et/ou instructions qu’il a reçues et d’envoyer à l’utilisateur la réponse correspondant le plus possible à sa requête.

Ces trois composants, une fois mis ensemble forment un Chatbot fonctionnel capable de recevoir des requêtes d'utilisateurs, de les rendre compréhensibles par le système et d'en sortir la réponse adéquate à l'utilisateur et ce ainsi de suite jusqu'à ce que l'échange prenne fin.

1.3. TROIS TYPES DE CHATBOTS

Le simple cycle d'échange de données expliqué ci-dessus permet à présent de diviser cette fois les Chatbots en trois catégories distinctes définissant les variantes de Chatbots les plus communes.

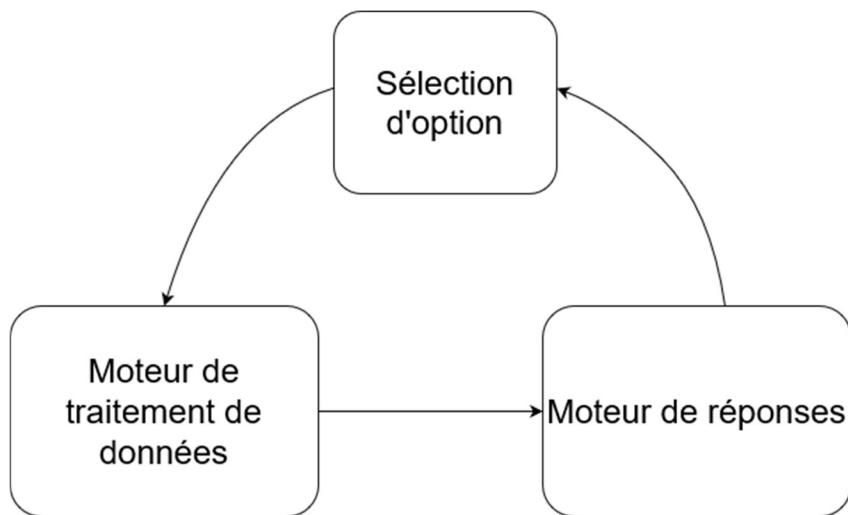


Illustration 2: Graphique d'un Chatbot à menus

Source : Rodrigues dos Santos Fabio

a) LES CHATBOTS À MENUS

Les Chatbots les plus basiques sont ceux fonctionnant à l'aide d'un menu. L'utilisateur discute avec le bot en sélectionnant, soit par un clic ou par une entrée texte, une option.

Les Chatbots à Menus, bien qu'étant factuellement très simplistes ne sont pour autant pas mis de côté du fait de leur aspect simpliste et parfois préféré par rapport à d'autres types de Chatbots. En effet, un Chatbot à menus permet d'éviter tout soucis qui

pourraient advenir lors de l'implémentation des autres types de Chatbots plus bas. Grâce à leur canal de communication on ne peut plus facile à utiliser pour l'utilisateur : Des boutons. L'intégralité de l'échange est réalisée grâce aux pressions successives des multiples boutons s'affichant à l'écran de l'utilisateur et ne nécessitant aucune autre forme d'interaction de ce dernier.

Ce type de bots est notamment très prisé dans des scénarios où les requêtes pouvant être effectuées par l'utilisateur sont prédéfinies. Par exemple, pour un système de commande de plats en ligne ou un système de support téléphonique qui lui, va imiter un Chatbot en demandant à l'utilisateur de presser des boutons sur leur téléphone pour spécifier quel type d'aide ce dernier nécessite. Dans ces deux exemples, l'usage d'un menu permet de s'affranchir d'un opérateur humain à cette étape de l'interaction et donc de gagner du temps. Le résultat de l'interaction avec le client va communiquer des informations à l'opérateur humain avant même qu'il n'ait à demander quel type de soucis le client rencontre dans le cadre d'un support téléphonique ou encore lors de la prise de commande de ce dernier. Actuellement, ce type de Chatbot est l'un des plus répandus dans la majorité des systèmes de réservations.

La séquence de pression de boutons peut être représentée sous la forme d'un arbre de décision. Comme le moyen d'interaction usuel de ce bot se représente sous la forme de boutons cliquables, le système impose un cadre prédéfini ainsi qu'un nombre limité d'interactions possibles et prohibe toute sortie de ce cadre. En effet, à chaque pression de bouton, l'utilisateur prends un des nombreux chemins existants dans cet arbre de décision jusqu'à arriver (ou non) à une réponse convenable et renvoyé à l'utilisateur au travers du moteur de réponses.

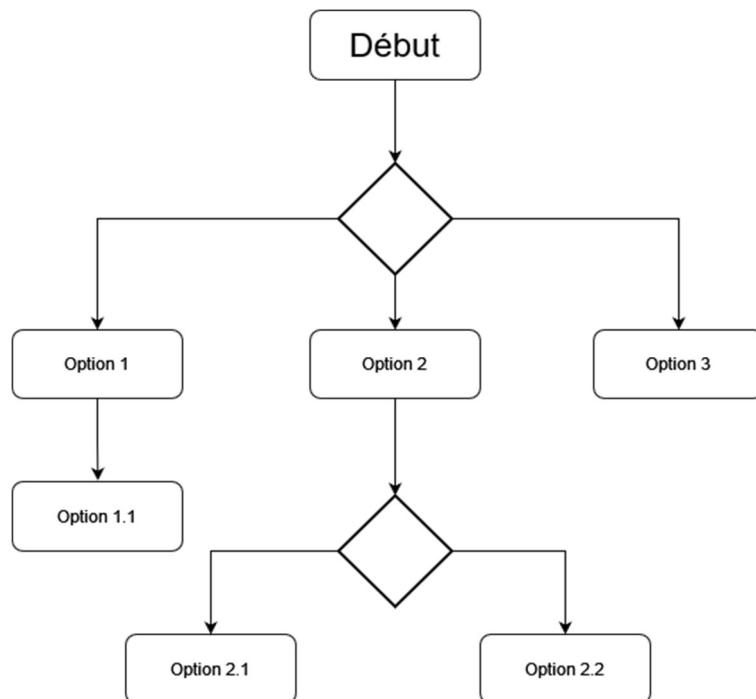


Illustration 3: Arbre de décision d'un Chatbot à menus

Source : Rodrigues dos Santos Fabio

Un des inconvénients possibles de ce Bot est en termes de scalabilité. Plus l'on souhaite étendre le panel de décisions possibles plus il faudra adapter l'affichage en conséquence et entre autres, un panel trop large pourrait être un désavantage plus qu'un avantage car bien que donner le plus de choix possible à l'utilisateur semble être optimal, en avoir trop et donc obliger l'utilisateur à cliquer sur une trop grande quantité de boutons peut ruiner l'expérience utilisateur.

Un autre inconvénient possible est que bien que dans certains cas, il est davantage souhaité d'avoir un cadre fixe, il y en a d'autant plus d'autres où l'utilisateur souhaite effectuer une requête qui ne correspond à aucune des options disponibles. Ce qui obligera à créer une très grande quantité de variations dans les possibles choix que l'utilisateur peut prendre et qui n'arrangerait pas plus le problème. Au contraire, ça l'aggraverait.

b) LES CHATBOTS PAR TOKENS

Se trouvant un cran au-dessus niveau complexité, se trouvent les Chatbots par tokens. Ces bots ajoutent un degré de liberté qu'il n'est pas aussi aisément trouvable avec un Chatbot à menus grâce à l'utilisation de règles.

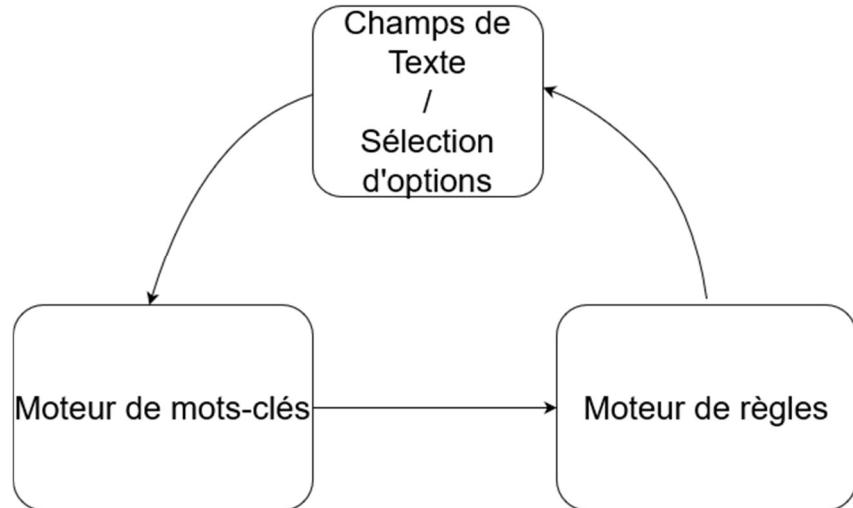


Illustration 4: Graphique d'un Chatbot de règles

Source : Rodrigues dos Santos Fabio

Ce type de bot est presque aussi populaire que ceux à base de menus car tout comme eux, ils répondent à des demandes similaires. Les interfaces souvent rencontrées avec ce genre de Chatbots sont soit des boutons ou cette fois-ci des champs de texte. Il est important de noter la distinction entre les Chatbots à Menus et par Tokens, car l'un possède un chemin prédéfini avec des options se succédant, alors que l'autre possède un ensemble de règles qui selon les attributs ressortis de la requête utilisateur, la règle la plus adéquate sera sélectionnée.

Un exemple pour le cas avec boutons serait qu'au lieu qu'un bouton ne fasse simplement avancer dans l'arbre de décision, que chaque bouton soit pondéré et que même avec une succession de pressions de boutons différentes on puisse arriver à la même réponse.

Dans le cas où l'entrée utilisateur est un texte, la méthode la plus facile et répandue est le simple parage de mots-clés dits « Tokens ». Selon les tokens ayant été ressortis de la requête, une règle comprenant un certain nombre de tokens correspondant ou tous se verra sélectionnée. Ci-dessous, on observe bien le fait que la règle n°3 a été choisie car comportant une succession de mots correspondant à cette règle.

Ceci est une phrase **d'exemple**, bien qu'elle paraisse **importante**



Illustration 5: Exemple de sélection de règle pour Chatbot par tokens

Source : Rodrigues dos Santos Fabio

Il existe deux approches possibles à la tokenisation du texte

La première consistant à faire usage d'un dictionnaire de mots-clés établi au préalable. Lorsqu'une entrée utilisateur se voit traitée par le moteur de mots-clés, ce dernier ira simplement vérifier si le token est présent dans le dictionnaire. Si c'est le cas, alors il est ajouté à la suite de tokens qui sera envoyée au moteur de réponse.

La deuxième, elle, consiste à faire usage du NLP (Natural Language Processing) qui permet de faciliter davantage l'implémentation du dictionnaire. Car dans la première approche, une limitation qui pourrait rapidement se faire ressentir est pour commencer le fait que pour chaque token ajoutés dans le dictionnaire, il faudra les associer à des règles et donc plus il y a de token, davantage il faudra créer d'associations règles-tokens. De plus, pour chaque token il peut exister une infinité de variations à cause de fautes d'orthographe

ou tokens similaires en écriture et/ou sens ce qu'une simple comparaison mot à mot ne pourrait pas détecter correctement.

En d'autres termes, le problème sous-jacent à une implémentation si simpliste est celui de la scalabilité de l'arbre de décisions. Pour un petit projet, ce n'est pas un grand soucis mais avec le temps cela peut vite devenir ingérable. C'est donc pour cette raison qu'il est judicieux de faire usage d'un moteur NLP si le projet commence à prendre de l'ampleur pour que la première approche soit viable car ce dernier permet d'éviter de devoir par exemple y inscrire toutes les variations d'un mot grâce à l'usage de diverses techniques de NLP voire de regrouper des familles de mots sous un seul type de token. De ce fait, la taille du dictionnaire final se verra grandement réduite et donc même avec un nombre de règles, l'application reste bien plus maintenable et scalable.

Les inconvénients à prendre en compte venant avec l'utilisation d'un Chatbot de règles est que bien qu'il permette un plus grand degré de liberté avec l'usage de champs textes et par conséquent de tokens qui élargissent le panel de possibilités à disposition de l'utilisateur en termes de requête, on reste tout de même dans un cadre restreint car tous les comportements sortant du moteur de règles doivent être implementés au préalable et continuellement mis à jour au besoin. Ce qui donne l'illusion à l'utilisateur qu'il peut poser quelque requête que ce soit mais il se verra vite confronté à une quantité de réponses limitées s'il sort trop du cadre initial par inadvertance.

c) LES CHATBOTS “INTELLIGENTS”

Le dernier type de Chatbot étant à la fois le plus complexe et pouvant apporter le plus de qualité en termes d'expérience utilisateur sont les Chatbots dit « Intelligents ».

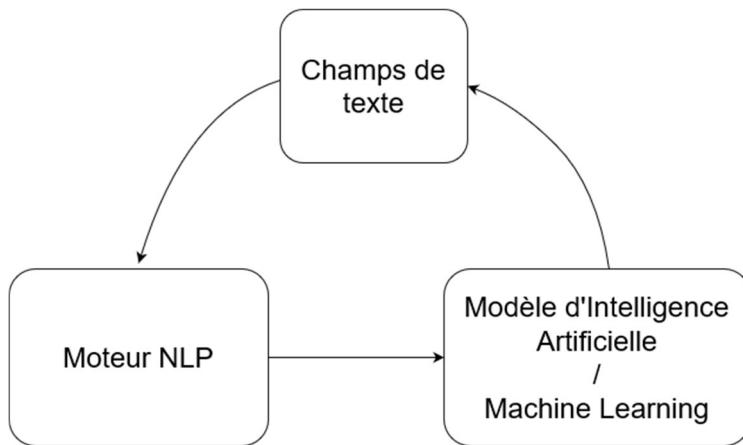


Illustration 6: Graphique d'un Chatbot Intelligent

Source : Rodrigues dos Santos Fabio

Les Chatbots intelligents permettent de pallier au problème commun aux deux types de Chatbots précédents étant le contexte restreint. Dans le cas du Chatbot de règles, il a bien été soulevé qu'ajouter de nouveaux tokens et règles rends le problème exponentiellement complexe dû à l'infini des tournures que peut prendre une phrase. Or, un Chatbot intelligent est capable d'outrepasser ces limitations par sa capacité à comprendre le sens des mots, le contexte dans lequel ils sont employés et d'engager dans une conversation avec un utilisateur en se rappelant des informations que ce dernier a pu fournir tout au long de la discussion. Étant donné la grande variété de modèles d'intelligence artificielle, il est difficile de donner un exemple d'implémentation spécifique. Le point commun entre tous cependant est qu'ils ont tous pour but de simuler une discussion fluide et dynamique avec l'utilisateur, comme s'ils discutaient avec un agent humain.

Ce type de Chatbot est souvent retrouvé actuellement dans des assistants virtuels tels qu'Alexa, Google Assistant, Siri, Amazon Echo et bien d'autres. Il en existe aussi sous la forme de Chatbots web comme le très populaire ChatGPT, Google Bard, Bing AI qui lui est une intelligence artificielle de type Générative qui a pour but de générer du nouveau contenu à partir d'un contenu existant. Ce qui, bien que cela est intéressant, n'est pas le focus du projet ici.

Cependant, un point commun à toutes ces I.A. est le traitement des requêtes par le biais du NLU (Natural Language Understanding). Selon le graphique présenté plus haut, une fois que l'application reçoit la requête utilisateur sous forme de texte, elle subit un prétraitement grâce à l'utilisation d'un moteur NLP. Ce prétraitement peut être plus ou moins utile selon le modèle d'I.A. ou de Machine Learning utilisé car certains modèles comme des I.A. conversationnelles sont programmée pour se charger de toute la partie de Tokenisation et autres traitements NLP avant d'appliquer un traitement NLU afin de comprendre le contexte, intention et sens des phrases et mots présents dans la requête. Le tout étant finalement traité par le modèle correspondant et sa réponse renvoyée à l'utilisateur.

Il est évident qu'un Chatbot de ce type apporte une grande plus-value à l'expérience utilisateur car ce dernier donne l'impression de comprendre quelque requête que ce soit et de rendre l'expérience bien plus personnelle qu'un Chatbot avec règles ou menus pourrait fournir. Hélas, cela ne s'obtient pas aussi aisément car tout comme le moteur NLP qui nécessite une certaine quantité de données pour être entraîné et donc, efficace; il faut une vaste quantité de données d'entraînement si l'on souhaite avoir un Chatbot capable de gérer toute situation se présentant ou alors une quantité assez large et variée pour au moins gérer la plupart des situations dans le cadre d'un projet de moins grande envergure. Cela implique qu'il n'est pas impossible de mettre en place un Chatbot intelligent pour un petit projet mais qu'il faut nécessairement assez de données à disposition. C'est pourquoi un prétraitement par un moteur NLP contribue à réduire le besoin de données le plus possible si l'entrée utilisateur peut être transformée en quelque chose de plus général et moins propice à être des cas uniques non pris en compte. Car l'usage de ce moteur permet de réduire le nombre de variations possibles pour, par exemple, un mot donné et à grande échelle réduire considérablement la taille des jeux de données.

L'essentiel à retirer de ces types de Chatbots se présente ainsi :

- Les Chatbots à Menus sont simplistes tant niveau interface que dans la manière de les utiliser. Ils permettent facilement d'arriver à une réponse en peu d'entrées utilisateur, la complexité d'implémentation étant basse les rend attractif selon le besoin, surtout si le cadre dans lequel il est utilisé est déjà un cadre restreint. Cependant, l'expérience n'est pas très personnelle et ne permet pas de sortir du cadre imposé par le menu lorsque la réponse à la requête utilisateur pourrait ne pas se trouver dans le cadre imposé par l'application. Très utilisé quand l'on sait d'avance que l'utilisateur ne peut pas sortir du cadre imposé et que l'on souhaite avoir un Chatbot nécessitant le moins de maintenance possible.
- Les Chatbots par tokens permettent d'obtenir des résultats similaires à ceux d'un Chatbot à menus mais en permettant une plus grande liberté côté utilisateur avec la possibilité de mettre un champs de texte à sa disposition. En plus d'une liberté accrue, l'expérience fournie sera plus personnelle car l'utilisateur peut formuler par ses propres mots sa requête. Or, un soucis de cadre existant persiste car ce qui définit le cadre est la quantité de règles et mots-clés gérés par le Chatbot. S'il n'y a pas assez de règles ou mots-clés pour couvrir tous les cas d'usage, cela empêterait sur l'expérience utilisateur et faire en sorte qu'aucune solution viable ne soit trouvée.
- Les Chatbots intelligents sont très prisés en ce moment de par leur avantages attractifs. Ces derniers étant la personnalisation de chaque discussion, de la liberté donnée à l'utilisateur en matière de requêtes tout en gardant un cadre et que le Chatbot tentera d'y ramener l'utilisateur s'il s'en écarte trop lors d'un échange, la facilité à implémenter un système dynamique sans avoir à se préoccuper de tout l'aspect de compréhension par NLU et traitement du texte par NLP. Cela permet après coup de n'avoir qu'à donner la sortie de ces traitements au modèle pré-entraîné qui trouvera la réponse adéquate à la requête

utilisateur. Mais pour y parvenir le plus grand obstacle reste toujours les données utilisées lors de l'entraînement des divers composants du Chatbot qui impactent grandement sa qualité. S'il n'y en a pas assez ou qu'elles ne soient pas assez variées pour coller aux spécificités souhaitées, le bot pourrait peiner en premier lieu à comprendre l'entrée utilisateur et par la suite à trouver la solution adéquate. De plus, l'usage de modèles d'I.A. ou de Machine Learning nécessitent un plus grand niveau de compréhension de ces derniers afin de déterminer quels modèles et implémentations sont les plus adéquates aux cas d'usages du projet.

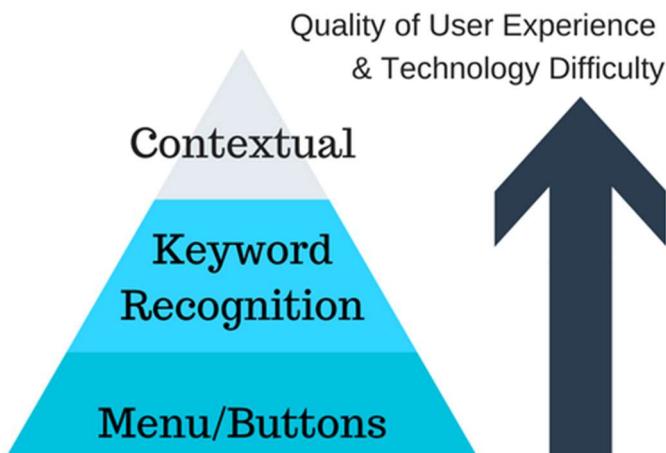


Illustration 7: Pyramide de la difficulté technologique et UX selon le type de Chatbot

Source : Ovationcsm, réf. : URL01

L'illustration ci-dessus démontre de manière visuelle la relation entre la difficulté de la technologie elle-même dans la création de Chatbots par Menus, par Règles ou Intelligents. Il est important de noter que le choix final du type de Chatbot souhaité ne dépend pas uniquement du niveau de qualité de l'expérience utilisateur fournie car sinon seul les Chatbots intelligents seraient utilisés mais la complexité d'implémentation, l'étendue des possibles interactions (tant larges que plus restreintes), ressources à disposition et autres paramètres sont à prendre en compte afin de choisir le Chatbot répondant au mieux aux attentes du projet.

Chapitre 2 : ÉTUDE DES SOLUTIONS EXISTANTES

Dans la partie qui suit, nous nous intéressons surtout aux fournisseurs locaux proposant des solutions de réservations. D'autant plus qu'il n'existe pas une grande quantité de ces derniers et que par conséquent nous les analysons et comparons entre eux.

2.1. AGENDA.CH

Agenda.ch est une application de réservation en ligne Suisse basée à Genève et lancée en 2011. Agenda.ch présente actuellement 5 services dédiés et 1 personnalisable à la demande. Les services à disposition sont :

1. Application dédiée aux physiothérapeutes
2. Application dédiée aux thérapeutes et autres services médicaux
3. Application dédiée aux centres de beauté et Bien-être
4. Application dédiée aux centres sportifs et loisirs
5. Application à visée d'administration publique

Fondamentalement, Agenda.ch fournit une application qui est un très clair exemple de Chatbot à menus. Peu importe le service, elle permet de réserver des ressources étant une salle de sport, une heure pour un rendez-vous médical, une séance pour une coupe de cheveux, etc., de choisir son créneau horaire et finalement tout autre information complémentaire.

L'aspect Chatbot n'est pas le seul utilitaire fourni par cette entreprise :

- L'application n'est pas fournie sous forme d'un module à intégrer directement dans le site web ou d'une API à appeler mais elle est disponible chez les serveurs loués par Agenda.ch et tout l'aspect de réservation est mis en place chez Agenda.ch et non chez le client.

- Selon le type de service fourni, il est possible de stocker une multitude de fichiers pertinents à la réservation des clients sur les serveurs d'Agenda.ch
- Selon le type de service fourni, un grand panel de fonctionnalités sont disponibles.
- La possibilité de configurer des messages envoyés automatiquement aux clients pour des rappels de réservations, informations, newsletter, etc.

Ce qui ressort de l'analyse de l'application d'Agenda.ch est qu'elle ne demande aucune implémentation de l'application par le client lui-même et fourni l'ensemble de ses services depuis leur site web que ce soit la configuration ou l'utilisation même du système de réservations. L'application est donc détachée du site web du client ce qui peut être vu comme un avantage étant donné que cela facilite son usage car il ne suffit que d'intégrer un simple bouton « réserver » sur le site.

Si l'on se penche sur les tarifs, on remarque qu'il y en a trois. Un à 35 CHF/mois qui n'offre pratiquement que le système de calendrier et de rappel aux clients d'une réservation. Ce n'est qu'à partir de 60 CHF/mois que l'on obtient les fonctionnalités telles qu'un mini-site permettant aux clients d'effectuer leurs réservations dessus ainsi que d'avantages de fonctionnalités en ligne pour faciliter l'aspect réservation. Ainsi, le système de réservation en ligne n'est pas forcément fourni avec tous les tarifs, rendant cette dernière comme un produit premium. Le dernier tarif étant à 80 CHF/mois propose principalement des systèmes de gestion de documents et ressources diverses directement effectuées par Agenda.ch, pratique si l'on ne possède pas de grande infrastructure ou que l'on ne souhaite pas avoir à se préoccuper d'en gérer une soi-même. Il existe évidemment des modules supplémentaires ou agendas supplémentaires selon des coûts additionnels variables.

2.2. KLARA.CH

Klara.ch est une entreprise Suisse fondée en 2016 basée à Berne et a pour but de fournir un panel varié d'outils d'administration. Contrairement à la solution précédente, Klara.ch ne se spécialise pas uniquement dans la mise en place d'un système de réservations mais d'une panoplie d'outils d'administration comme : des gestionnaires de client, gestionnaires de budget, gestionnaires d'inventaire, création de shop en ligne, mise en place d'un système de réservations et bien d'autres. L'application de réservation en ligne ainsi que les autres fonctionnent sous forme de widgets soit des modules applicatifs que le client intègre directement à son site web. La configuration de ce dernier se fait directement au travers du panneau de configuration Klara. Le système de réservation se présente sous la forme d'un Chatbot à menus, similaire à celui d'Agenda.ch. Le tarif pour le widget de réservation en ligne est de 39 CHF/mois.

La solution de Klara.ch est intéressante de par son coût réduit par rapport à Agenda.ch et permet d'avoir l'ensemble des fonctionnalités disponibles sur le site client et de ce fait, ne requiert pas de devoir se rendre sur une autre page internet. Cependant le degré de personnalisation et de fonctionnalités supplémentaires spécifiques à certains services ne semble pas comparable à celui d'Agenda.ch, expliquant ainsi son coût plus élevé.

2.3. MEETME.IO

Meetme.io est une application de réservation de rendez-vous Suisse basée à Lausanne et ayant pour objectif de fournir un service de réservation en ligne et de gestion d'agenda grâce à une application ou depuis un panneau d'informations. Meetme.io possède un catalogue de fonctionnalités bien plus réduit comparé aux solutions précédentes en ne fournissant qu'un système de formulaire similaire à celui d'Agenda.ch. Tout comme ce dernier, il est uniquement accessible par un lien qui redirige le client sur le site de Meetme.io à l'url du formulaire de réservation du service souhaité. Meetme.io fourni par ailleurs un système de rappel par sms. Similairement aux solutions précédentes, le

formulaire de Meetme.io est comparable à un Chatbot à menus. Le tarif quant à lui s'élève à 69 CHF/mois.

2.4. COMPARAISON À CE PROJET

Après analyse de quelques solutions Suisses disponibles sur le marché actuellement, il est clair qu'il y a une tendance à réaliser un système de réservation par internet. Tous fonctionnent sous la forme d'un formulaire étant ici l'équivalent d'un Chatbot à menus. Bien que certaines solutions offrent parfois plus de fonctionnalités que simplement un système de réservation ou proposent des systèmes de réservation avec des fonctionnalités étendues et spécifiques à certains secteurs, aucun ne propose de solution par téléphone uniquement. C'est donc ici que va se placer la solution qui sera proposée lors du travail de Bachelor étant donné qu'elle permettra de placer des réservations depuis un simple échange par texte avec un Chatbot par téléphone et d'ensuite éventuellement de récupérer facilement l'ensemble des réservation sous format Caldav ou d'exporter le tout en tant que calendrier google. Cela permettrait de complémenter les systèmes de réservations existant d'un particulier en offrant un outil supplémentaire pour placer des rendez-vous. Voire d'entièrement remplacer leur système par un Chatbot de qualité supérieure tel qu'un Chatbot par Tokens ou I.A.

Chapitre 3 : FONDEMENTS THÉORIQUES

Au cours des recherches concernant les diverses technologies associées à la réalisation du Chatbot, nous nous sommes retrouvés à étudier quelques concepts jusque-là nouveaux et qui ont par la suite servi à affirmer le choix technologique pour ce projet.

3.1. NATURAL LANGUAGE PROCESSING

La première technologie et une des plus importantes dans le cadre de ce projet est celle du **Natural Language Processing** (Traitement du langage naturel) (NLP). Cette technologie est une pierre angulaire au sein du domaine du machine learning et de l'intelligence artificielle. En effet, le principal intérêt de ce dernier est de permettre de prendre quelconque texte et d'appliquer divers traitements dessus en identifiant des patterns dans les phrases et mots et qui donne en sortie une suite de données structurée que les divers modèles seront capables d'exploiter afin qu'eux aussi appliquent des traitements variés sur ces données pour finalement obtenir une réponse. Ce qui sans le NLP serait bien plus ardu.

L'être humain en quelque sorte applique une multitude de techniques de NLP tous les jours pour faire sens des paroles et textes que nous lisons. De manière générale, tout traitement NLP suit globalement cette séquence :

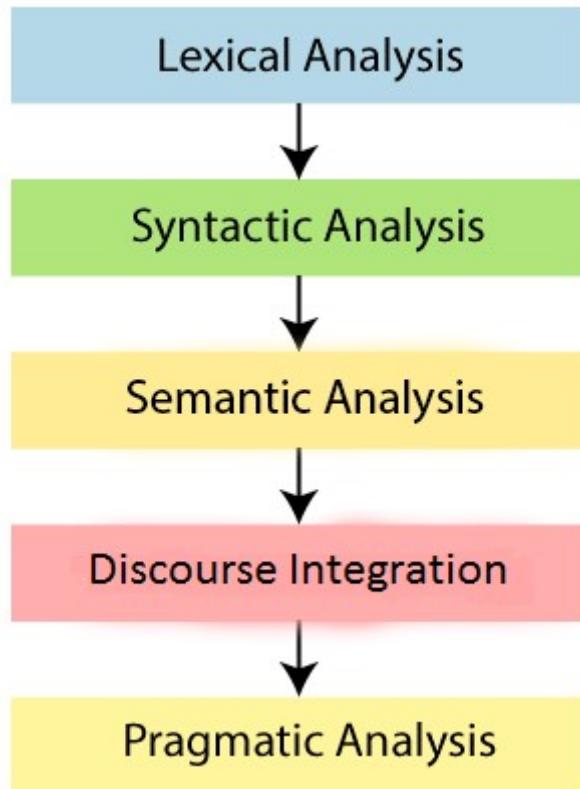


Illustration 8: Séquence de traitement NLP

Source : javapoint.com, réf. : URL02

Pour chaque étape il existe une grande quantité de traitements divers, cependant nous n'allons qu'en présenter ici une liste non exhaustive :

1. Analyse Lexicale

a. Tokenisation

La Tokenisation est une des premières techniques NLP à être appliquée à un texte donné. Cette dernière a pour but de séparer tout ce qui s'apparente à des mots sous forme de Tokens.

Un exemple de Tokenisation serait « Je vais marcher dehors » qui deviendrait « ‘Je’, ‘vais’, ‘marcher’, ‘dehors’ ».

b. Lemmatisation et Stemming

La Lemmatisation et le Stemming sont présentés ensemble car ils ont un but très semblables étant de prendre un mot et de le réduire à une forme commune.

Pour se faire, la Lemmatisation va faire usage d'un dictionnaire de mots comprenant des groupes de mots similaires et étant tous lié par une même racine commune.

Par exemple : « Jouer, Joueur, Jeu, joué, ... » vont avoir pour mot racine « Jeu »

Ensuite, le Stemming lui, de par la signification du mot en anglais Stem voulant dire tige d'une plante et étant la base de cette dernière, consiste à prendre chaque mot et de les séparer en deux parties. Comme des mots de la même famille possèdent une base commune, retirer l'excès de lettres permet d'en dégager une racine commune.

Par exemple : « Concourût, Concourir, Concourant, Concours » vont se voir réduits à « Concour ».

2. Analyse Syntaxique

a. Part-of-speech Tagging (POS)

Le POS Tagging permet de donner un tag/label à chaque mot dans un texte afin de lui assigner sa classe correspondante grammaticalement parlant. L'intérêt principal de ce traitement est de pouvoir par la suite interpréter la structure de la phrase d'une manière grammaticale. Ensuite, donner un sens clair à certains mots qui parfois selon la structure grammaticale de la phrase où ils se trouvent peuvent avoir un tout autre sens. En d'autres mots, cela permet de contextualiser le sens d'un mot.

Par exemple : « Je me rends au pied de la montagne », les labels de POS tagging correspondants seraient « (Je, Pronom), (me, Pronom), (rend, Verbe), (au, Préposition), (pied, Nom), (de, Préposition), (la, Déterminant), (Montagne, Nom) ».

3. Analyse Sémantique

a. *Analyse de sentiments*

L’analyse de sentiments est une sorte de boite à outils composée de nombreux outils permettant de dégager les divers émotions présentes dans un texte. Nous mentionnons cette partie par souci d’exhaustivité sans l’approfondir, car nous estimons cette approche peu adaptée à un système de réservation.

Le principe général de l’analyse de sentiments repose sur des dictionnaires de mots étant associés à des sentiments et ensuite passés au moteur NLP. Si leur présence est détectée, selon la quantité présente de sentiments ou même le manque de mots évoquant un sentiment particulier, un ou plusieurs sentiments seront assignés au texte analysé. Il est possible dans certains cas d’associer des sentiments à des principes ou objets afin d’affiner l’analyse.

b. *Named Entity Recognition*

Le NER ou Reconnaissance d’entités nommées est un des nombreux outils NLP permettant de donner davantage de sens aux tokens présents. Selon un dictionnaire associant des mots à une catégorie prédéfinie ou même une association de classes grammaticale commune avec une catégorie d’entité, l’ensemble de tokens sélectionnés est attribué à un type d’entité ou dans le cas contraire, aucun.

Par exemple : « La semaine dernière, le cours de la bourse chez Twitter a chuté de 4.8% ». Les entités ressorties de cette phrase sont : « (Semaine dernière, Date), (Twitter, Organisation), (4.8%, Pourcent) »

4. Intégration de données

L'intégration des données est le simple fait de prendre les textes précédemment analysés tant dans la même session ou dans un jeu de donnée existant. Ensuite, les diverses analyses précédemment effectuées sont comparées entre elles après cela à celles réalisées sur le texte courant. Selon le niveau de ressemblance un sens/contexte sera dégagé et donné à ce dernier.

Un exemple serait que si l'on analyse une multitude de phrases ayant mentionné initialement un personnage homme et un personnage femme dans un livre et que par la suite le nom du personnage homme ne soit plus mentionné mais qu'à la place le pronom « il » est utilisé, par analyse des phrases précédentes il sera défini que les « il » font référence au personnage masculin.

5. Analyse Pragmatique

L'analyse Pragmatique n'indique pas spécialement une technique en particulier mais consiste davantage en la combinaison des techniques précédemment utilisées afin de déterminer le contexte global du texte. L'ensemble des tokens et labels ressortis précédemment vont, une fois comparés entre eux à l'aide de dictionnaires et autre jeu de données permettant d'aider à la définition du lien entre eux, donner un contexte et sens au texte courant.

Le NLP en tant que tel sert principalement à traiter du texte de manière générale pour être utilisé par la suite par d'autre algorithmes. Le NLP peut être vu comme la catégorie principale en matière de traitement de texte, cependant il existe deux sous catégories au NLP étant le Natural Language Understanding ou Compréhension de langage Naturel (NLU) et le Natural Language Generation ou Génération de langage Naturel (NLG).

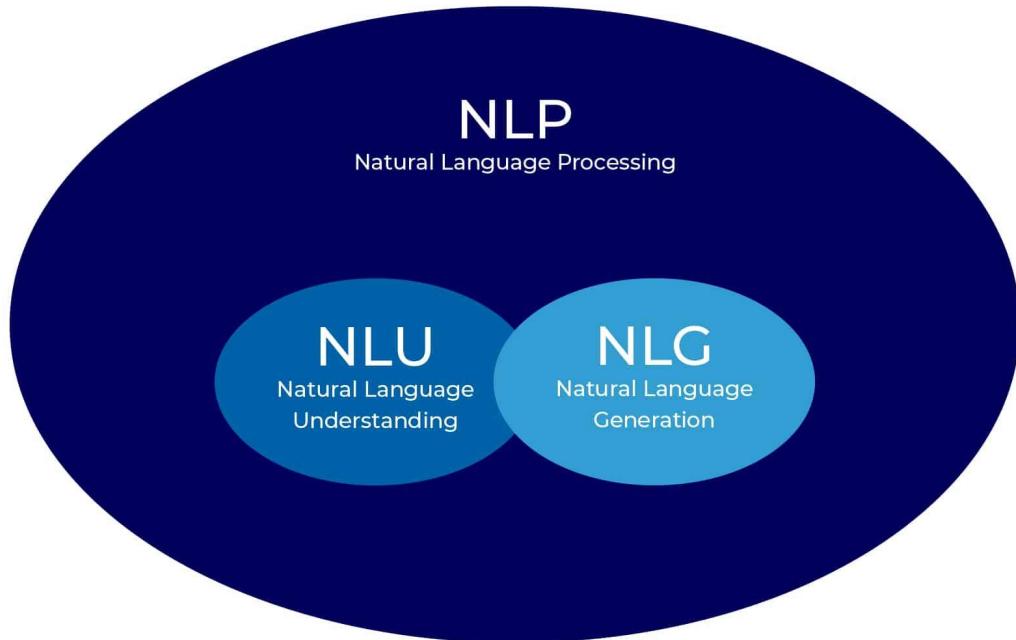


Illustration 9: Représentation de la relation NLP, NLU et NLG

Source : Datasolut, réf. : URL03

a) **NATURAL LANGUAGE UNDERSTANDING**

Selon Datasolut, “Le domaine du NLU s’occupe de la compréhension du langage naturel. Diverses méthodes de compréhension de texte sont utilisées à cet effet. Spécifiquement, la grammaire et le contexte de paires de mots ou mots uniques sont analysés afin d’en ressortir la signification de ces derniers et de la phrase. En outre, la sémantique, syntaxe, intention et émotion dégagée par un texte sont examinés.”²

² WUTTKE, LAURENZ, 2023. NLP vs. NLU vs. NLG: UNTERSCHIEDE, FUNKTIONEN UND BEISPIELE. DATASOLUT GMBH [EN LIGNE]. 24 MAI 2023. DISPONIBLE À L’ADRESSE : [HTTPS://DATASOLUT.COM/NATURAL-LANGUAGE-PROCESSING-VS-NLU-VS-NLG-UNTERSCHIEDE-FUNKTIONEN-UND-BEISPIELE/](https://datasolut.com/natural-language-processing-vs-nlu-vs-nlg-unterschiede-funktionen-und-beispiele/) [CONSULTÉ LE 1 MARS 2024].

Deux concepts principaux pour parvenir à la compréhension de textes par NLU sont :

1. Détection de l'intention : Le processus d'identification des sentiments d'un utilisateur et de déterminer son objectif. Un des buts premiers du NLU étant l'établissement du sens du texte donné.
2. Détection d'entités : Un processus plus spécifique qui identifie des entités spécifiques et extrait les informations les plus importantes de ces dernières. Il existe deux types d'entités : Celles dites nommées ou numériques.

Les éléments nommés sont groupés en catégories telles que des noms d'individus, de business et de lieux.

Les éléments numériques sont reconnus en tant que quantités, dates, monnaies et pourcentages.³

Le NLU est une technologie prévalente dans le développement de Chatbots et autres assistants intelligents ayant pour but de converser avec un utilisateur et celle-ci leur permet de comprendre ce que ce dernier leur répond.

b) NATURAL LANGUAGE GENERATION

Le NLG est la deuxième sous-catégorie de NLP ici servant à générer du texte en langage naturel. Cette catégorie de NLP est une partie vitale de nombre d'I.A. conversationnelles car il ne suffit pas simplement de comprendre ce que l'utilisateur leur dit mais d'aussi pouvoir générer une réponse cohérente et intelligible par un humain. Pour se faire, on emploie de la génération de langage naturel.

³ WHAT IS NATURAL LANGUAGE UNDERSTANDING (NLU)? | DEFINITION FROM TECHTARGET, ENTERPRISE AI [EN LIGNE]. DISPONIBLE À L'ADRESSE : [HTTPS://WWW.TECHTARGET.COM/SEARCHENTERPRISEAI/DEFINITION/NATURAL-LANGUAGE-UNDERSTANDING-NLU](https://www.techtarget.com/searchenterpriseai/definition/natural-language-understanding-nlu) [CONSULTÉ LE 1 MARS 2024].

Le fonctionnement du NLG varie selon son application, cependant un modèle général est détaillé par R. Dale et E. Reiter⁴ dans une de leurs publications comme suit :

1. Détermination de Contenu et planification du discours : La détermination de contenu est une des premières étapes du NLG qui se charge de trouver quelles informations seront communiquées sous la forme d'un texte à l'utilisateur. Afin d'y parvenir, un filtre est appliqué sur l'ensemble des données afin de ne récupérer que ce qui semble pertinent et de résumer chaque donnée afin de ne garder que l'essentiel.

La planification du discours, lui, est l'action de réordonnancement des informations précédemment récupérées dans un ordre spécifique afin de ne pas exprimer ces dernières de manière disparate.

2. Agrégation de phrases : Cette étape a pour but de prendre plusieurs messages et de les regrouper en une seule phrase. Par exemple si de multiples messages mentionnent des informations concernant un cours comme suit :

- “Le cours de Français est donné par Mr. Bournon”
- “Le cours de Français aura lieu en salle A201”
- “Le cours de Français commence à 15h”

Sera probablement agrégé de la manière suivante : “Le cours de Français qui aura lieu en salle A201, donné par Mr. Bournon, commence à 15h”.

Il est important de mentionner qu'il existe diverses techniques d'aggregation et que la phrase présentée ci-dessus n'est qu'un exemple.

⁴ REITER, EHUD ET DALE, ROBERT, 1997. BUILDING APPLIED NATURAL LANGUAGE GENERATION SYSTEMS. NATURAL LANGUAGE ENGINEERING. VOL. 3, NO 1, PP. 57-87.
DOI 10.1017/S1351324997001502.

3. Lexicalisation : Cette étape du processus de NLG n'est pas obligatoire mais elle sert à remplacer des données présentes sous forme de concepts ou entités et de les transformer en un ou plusieurs mots à afficher.

Un exemple de lexicalisation est : Dans un cadre d'une réservation de tables dans un restaurant ("RESERVATION" -> "Une réservation", "MARINIÈRE" "à la Marinière", "4 TABLES" -> "de quatre tables"), on obtiendrait donc après lexicalisation : "Une réservation de quatre tables à la Marinière"

4. Génération d'expressions référentielles : Cette étape va se concentrer sur la désambiguisation des diverses entités trouvées dans les informations à disposition en les remplaçants par des noms, pronoms, descriptions détaillées et autre type d'expressions référentielles.

Par exemple, si les informations contiennent des références à Microsoft, il est possible qu'après un traitement par Génération d'expressions référentielles qu'initiallement Microsoft soit mentionné comme tel dans une phrase puis que par la suite ce dernier soit mentionné sous la forme de "Ce dernier", "La compagnie ayant créé le système d'exploitation Windows", "Ce géant de la Silicon Valley" et bien d'autres encore.

5. Réalisation : Cette dernière étape se charge surtout d'appliquer des correctifs orthographiques et de forme aux mots déjà présents dans la phrase mais ne se charge plus de la syntaxe, seulement de l'orthographe, conjugaison, accords et tout ce qui ne s'apparente pas à de la syntaxe.

Ce qui est intéressant à dénoter si l'on se réfère à nouveaux à *l'illustration 9*, le fait que la bulle NLU se retrouve partiellement couverte par le NLG n'est pas une coïncidence car comme on le comprends avec le modèle précédemment établi, le NLU est une partie intégrante du processus de NLG. Afin de pouvoir générer une quelconque phrase, il faut

tout d'abord comprendre les données à la disposition du modèle afin de les assembler en une phrase éligible et faisant sens.

La technologie du NLP ainsi que ses deux sous-catégories enfants NLU et NLG sont des éléments fondateurs des modèles d'I.A. conversationnelles comme ChatGPT qui utilise ces deux principes pour interpréter les entrées des utilisateurs et leur renvoie une réponse.

3.2. TAPAS

La seconde technologie qui a été étudiée est TAPAS ou Parseur de Tables. C'est un modèle d'I.A. à faible supervision pré-entraîné développé par des ingénieurs de chez Google en 2020, basé sur le modèle BERT, comme détaillé dans un article de recherche publié par l'ACL (Association for Computational Linguistics)⁵.

Avant sa conception, le parsage de textes sous forme de langage naturel se faisait principalement à l'aide de moules logiques décrivant en simples termes la forme que prends l'information que l'on cherche sémantiquement parlant. Ce qui demandait une quantité considérable de pré-calculs en amont du parsage du texte.

C'est en réponse à ce problème qu'une approche populaire au parsage sémantique a été choisie : l'utilisation d'une supervision faible. Dans le contexte du machine learning, cette dernière consiste à chercher de manière exacte le label d'une donnée. Les labels utilisés lors de l'entraînement du modèle seront volontairement incomplets car ayant du bruit sous forme de lettres en trop par exemple et peu précis comparé au label cible. Dans notre cas, cela veut dire que lors d'une requête de parsage de table de donnée, au lieu de chercher le label qui correspond exactement à celui fourni par l'utilisateur dans sa requête, on va

⁵ HERZIG, JONATHAN ET AL., 2020. TAPAS: WEAKLY SUPERVISED TABLE PARSING VIA PRE-TRAINING. IN : PROCEEDINGS OF THE 58TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, PP. 4320-4333. 2020. DOI 10.18653/v1/2020.acl-main.398. ARXIV:2004.02349 [CS]

chercher les données correspondant à un label s'approchant le plus de ce dernier sans pour autant chercher une exactitude forte. Or, cette approche reste tout de même couteuse car nécessitant la génération de moules logiques même pour ces labels.

L'approche que prends TAPAS est par conséquent de ne pas avoir à générer ces moules logiques pour trouver les bons labels mais d'effectuer une sélection de cellules ainsi qu'optionnellement d'appliquer divers opérateurs à cette dernière. Comme TAPAS est basé sur BERT, un framework NLP, il est donc possible d'effectuer de simples requêtes sous forme de langage naturel et qu'ensuite TAPAS comprenne cette requête et retourne l'ensemble de données souhaité similairement à une requête SQL mais sans avoir à entrer une requête en langage SQL.

Afin de mieux comprendre le fonctionnement de TAPAS, il faut se pencher sur sa fondation étant BERT et pour se faire, demandons-nous ce qu'est un Transformer, un des composants principaux de BERT.

a) TRANSFORMERS

Un transformer est “un réseau neuronal qui apprends le principe de contexte et de compréhension de mots au travers d’analyse séquentielle de données”⁶. Afin d’y parvenir, une suite de N encodeurs vont analyser de manière parallèle des séquences de tokens et y appliquer le mécanisme d’attention.

Le mécanisme d’attention propre aux transformers est le simple fait de donner un poids à chaque paire de tokens et ce poids est influencé par le lien entre les deux tokens. De ce fait, après avoir passé notre séquence de tokens dans les divers couches d’encodeurs,

⁶ THE ULTIMATE GUIDE TO TRANSFORMER DEEP LEARNING, [EN LIGNE]. DISPONIBLE À L’ADRESSE : [HTTPS://WWW.TURING.COM/KB/BRIEF-INTRODUCTION-TO-TRANSFORMERS-AND-THEIR-POWER](https://www.turing.com/kb/brief-introduction-to-transformers-and-their-power) [CONSULTÉ LE 2 MARS 2024].

le résultat de ces encodeurs seront passés au décodeur. Il va ensuite permettre de générer la séquence résultat selon le contexte dérivé de la valeur d'attention propre à chaque token.

Par exemple, si l'on a une phrase étant “As-tu bien dormis ?”, grâce à un set d'entraînement qui définit le lien entre les mots présents dans cette phrase et d'autres mots commun par une valeur d'attention élevée et selon un contexte donné comme ici étant une question, une sortie possible serait “Oui, j'ai bien dormis”.

En d'autres mots, les transformers sont fondamentaux à la compréhension de textes et prédition de phrases et suites logiques possibles à un texte donné pour le modèle BERT grâce à ce mécanisme d'attention.

b) BERT

Bidirectional Encoder Representations from Transformers (Bert) (Représentation d'encodeurs bidirectionnels de Transformers) a été créé en Octobre 2018 par Google. L'utilité principale de BERT est la compréhension de langage naturel à l'aide de transformers et étant utilisé dans des produits très mondialement connus comme pour ne citer que lui, le moteur de recherche Google :

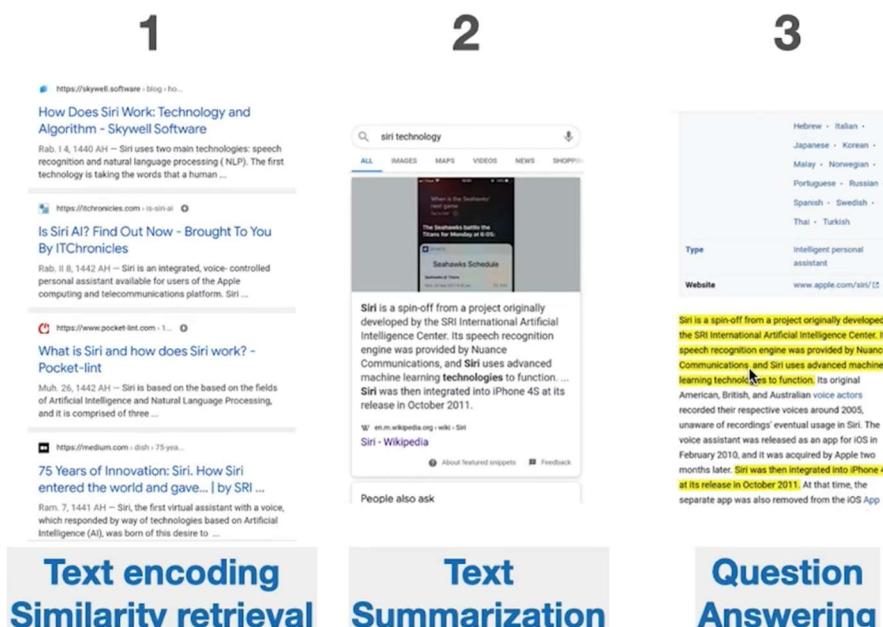


Illustration 10: Cas exemples du moteur google utilisant BERT

Source : Language processing with BERT by Jay Alammar, réf. :URL04

1. Récupération de similarité d'encodage de texte : La requête de l'utilisateur est traitée par des techniques NLP et ressortie sous forme de tags. Ce qui permet à BERT de trouver des pages web qui possèdent des caractéristiques similaires à celles de la requête et lui retourner les pages les plus pertinentes.
2. Résumé de texte : Lors d'une recherche sur un sujet comme par exemple "Siri technology", l'aide de diverses techniques NLP présentes dans le modèle BERT vont permettre de générer un résumé cohérent et concis des informations que l'on cherche à savoir grâce à la compréhension de contexte et sens des mots, ce à quoi BERT excelle de par l'usage de transformers.
3. Question-réponse : Un forme plus simple est en utilisant la requête utilisateur qui une fois passée dans le modèle BERT, ce dernier cherche quel passage du texte wikipedia correspond le plus à une réponse valable à la requête demandée et dans le cas présent le texte sera surligné sur ledit site.

Un exemple plus concret de BERT en application est donné par Techtarget⁷ :

La phrase que BERT reçoit ici est « The animal didn't cross the street because it was too wide ». À un moment donné le mot "it" sera sélectionné lors du passage du texte dans les couches de l'encodeur. Il est ensuite mis face à tous les autres mots présents dans la phrase et selon l'algorithme NLP mis en place dans le modèle BERT courant, il est déterminé que IT est un pronom qui désigne une entité. De ce fait, BERT tente ensuite d'identifier à quelle entité « IT » pourrait faire référence. Les trois mots possibles ici sont

- « The »

⁷ WHAT IS BERT (LANGUAGE MODEL) AND HOW DOES IT WORK?, ENTERPRISE AI [EN LIGNE]. DISPONIBLE À L'ADRESSE :

[HTTPS://WWW.TECHTARGET.COM/SEARCHENTERPRISEAI/DEFINITION/BERT-LANGUAGE-MODEL](https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model) [CONSULTÉ LE 4 JANVIER 2024].

- « animal »
- « street »

Après usage du mécanisme d'attention qui va calculer dynamiquement la connexion entre chaque paire de mots dans la phrase et leur attribuer un poids, le mot ayant la plus forte connexion ici est « Street ». Ayant un poids plus important, il est donc déterminé comme le mot auquel « IT » fait référence, donnant ainsi davantage de contexte.

Grâce à la lecture bidirectionnelle utilisée par BERT, le modèle est donc capable de donner un sens à des mots, phrases ou textes entiers selon la façon dont les mots sont employés et de quelle manière chaque phrase est formée. Sens qui est donné au travers de Tokens qui permettent à BERT d'être utilisé dans le cadre de :

- Question-Réponse à l'aide de divers textes donnés au modèle
- Capacité à résumer efficacement des textes
- Prédiction de phrases
- Génération de réponses lors d'une conversation, similaire à ChatGPT
- Analyse approfondie d'un texte afin de mieux cerner l'intention de ce dernier

L'entrée du modèle BERT est une suite de vecteurs nommés tokens et qui dans le cas de l'illustration qui suit représentent une phrase.

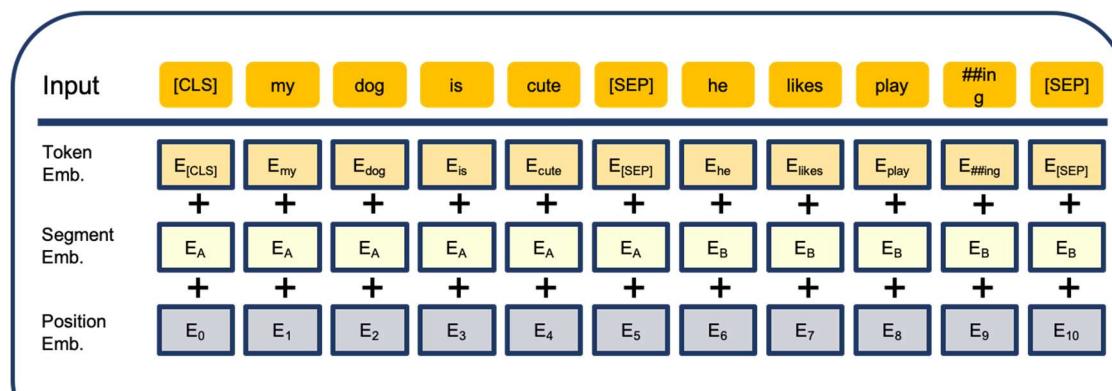


Illustration 11: Représentation de l'entrée du modèle BERT

Source : Humboldt Universität, réf. : URL05

Dans un input, le premier token est toujours un token de Classification [**CLS**], suivi ensuite ici de la première moitié du texte, d'un séparateur [**SEP**] pour marquer la fin de la première partie et le début de la deuxième et marqué pour finir d'un nouveau séparateur et ainsi de suite. Cette suite de tokens est le Token Embedding.

Ensuite une autre couche est ajoutée, le Segment Embedding. Ce dernier consiste à définir quels tokens appartiennent, dans le cas présent, à la première ou deuxième moitié du texte.

La dernière couche, le Position Embedding, permet d'indexer chaque token.

c) TAPAS : FONCTIONNEMENT

En contraste avec l'entrée du modèle BERT, l'entrée pour le modèle TAPAS se présente sous la forme suivante :

Token Embeddings	[CLS]	query	?	[SEP]	col	#1	col	#2	0	1	2	3
Position Embeddings	POS ₀	POS ₁	POS ₂	POS ₃	POS ₄	POS ₅	POS ₆	POS ₇	POS ₈	POS ₉	POS ₁₀	POS ₁₁
Segment Embeddings	SEG ₀	SEG ₀	SEG ₀	SEG ₀	SEG ₁							
Column Embeddings	COL ₀	COL ₀	COL ₀	COL ₀	COL ₁	COL ₁	COL ₂	COL ₂	COL ₁	COL ₂	COL ₁	COL ₂
Row Embeddings	ROW ₀	ROW ₁	ROW ₁	ROW ₂	ROW ₂							
Rank Embeddings	RANK ₀	RANK ₁	RANK ₁	RANK ₂	RANK ₂							

Illustration 12: Entrée du modèle TAPAS

Source : TAPAS: Weakly Supervised Table Parsing via Pre-training, réf. : URL06

Comparativement à l'entrée BERT, celle de TAPAS inclut avant la première séparation la requête de l'utilisateur et après le [**SEP**] le tableau à analyser mis à plat. Le “col” et “#1” ici représentent le nom de la première colonne et celui qui suit celui de la 2ème colonne. Il peut y en avoir autant qu'il y a de colonnes dans le tableau. Suivent ensuite les données qui sont assemblées de manière séquentielle de sorte à traverser le tableau de gauche à droite selon le nombre de colonnes et de haut en bas.

Si l'on devait créer le tableau que l'on a dans l'exemple il ressemblerait à cela :

Col1	Col2
0	1
2	3

Tableau 1: Représentation du tableau fourni en entrée de TAPAS

Source : Rodrigues dos Santos Fabio

La couche Position Embeddings est identique à celle de BERT, servant à donner des indexées à chaque token.

La couche Segment Embeddings permet de définir si un token appartient à la requête ou au tableau.

La couche Column et Row Embeddings ont pour but de spécifier dans l'entrée même si une donnée provient d'une colonne et ligne donnée afin de ne pas perdre la structure initiale du tableau.

La dernière couche, Rank Embeddings, permet de donner un ordre aux divers tokens présents. Un type d'ordre peut simplement être un ordre numérique croissant comme ici : Si les tokens présents sont “3”, “4”, “2” et “7”, le rang correspondant à chaque token serait de l'ordre de “2”, “3”, “1” et “4”.

La procédure qui surviendra après avoir fourni la requête et le tableau duquel l'on souhaite extraire des données, peut être représentée de la forme suivante :

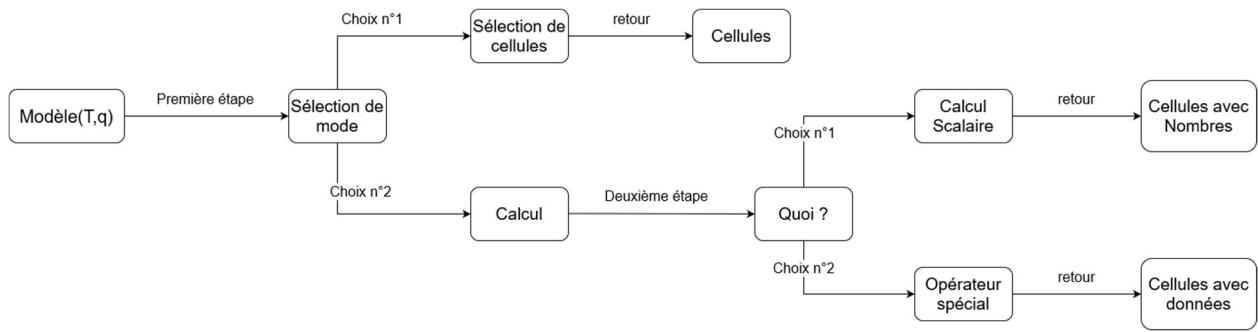


Illustration 13: Schéma de fonctionnement du modèle TAPAS

Source : Rodrigues dos Santos Fabio

La procédure définie ci-dessus démontre les divers choix qu'effectue le modèle afin de déterminer la bonne suite d'opérations à appliquer pour ressortir un résultat.

Premièrement, l'utilisateur fournit une requête et un tableau au modèle. Ensuite vient la première étape étant la sélection de mode. Les modes disponibles sont :

- Sélection de cellules
- Calcul

Si le mode choisi est une sélection de cellules, le résultat en sortie est une simple récupération de la donnée se trouvant dans la/les cellule/s permettant de répondre à la requête en entrée.

Sinon, le modèle n'aura d'autres choix que de choisir le mode calcul qui lui fait basculer le modèle dans une deuxième phase qui a pour but de choisir quelles cellules doivent être sélectionnées pour ensuite appliquer un opérateur parmi les 4 suivants :

- Rien, on renvoie simplement les cellules sélectionnées indiquant un cas de simple sélection de cellules.
- Compter, on compte le nombre de cellules sélectionnées. C'est une réponse ambiguë.
- Somme, on calcule la somme de toutes les cellules sélectionnées. C'est une réponse scalaire.

- Moyenne, on calcule la moyenne de toutes les cellules sélectionnées. C'est une réponse scalaire.

Afin de choisir les bonnes cases et opérations, un softmax (chaque choix possible reçoit un pourcentage basé sur la cohérence avec la requête) est appliqué sur chaque case et ensuite de même pour les opérateurs afin de trouver ce qui répond au mieux à la requête initiale.

En l'état, les opérateurs précédemment énoncés sont les seuls opérateurs disponibles présentement avec le modèle. Il est toutefois possible d'en ajouter soi-même, selon le papier de recherche des ingénieurs de Google⁷.

Le fonctionnement des opérations effectuées par le modèle en son état actuel dans le projet de base de Google⁸ fait qu'un tuple contenant le type d'opération et les cellules sélectionnées sont retournées, demandant donc un post-traitement des données afin d'appliquer l'opérateur choisi par le modèle.

La manière dont est entraîné le modèle est que chaque opérateur, mode et sélection de cellules possèdent, de manière inhérente à tout réseau neuronal, des poids probabilistes et leur valeur correspondante et en faisant la fonction de coût de l'erreur carrée basée sur le résultat attendu et le modèle va effectuer une propagation arrière (mise à jour des poids de chaque neurone dans le réseau selon le coût nouvellement calculé, partant) et ainsi de suite jusqu'à arriver de manière régulière à tomber sur les bons paramètres.

Une particularité est quand l'on peut avoir une solution ambiguë qui nécessite un traitement particulier car le résultat ne se trouve pas explicitement dans le tableau comme dans le cas d'un comptage de cellules. Pour se faire, un softmax sur les deux possibilités

⁸ GOOGLE-RESEARCH/TAPAS [LOGICIEL] [EN LIGNE]. 29 FÉVRIER 2024. GOOGLE RESEARCH [CONSULTÉ LE 3 MARS 2024]. DISPONIBLE À L'ADRESSE : [HTTPS://GITHUB.COM/GOOGLE-RESEARCH/TAPAS](https://github.com/google-research/tapas) [CONSULTÉ LE 3 MARS 2024].

étant une sélection de cellules ou une agrégation sera appliquée pour déterminer la solution optimale.

Une dernière fonctionnalité intéressante de TAPAS est que l'on peut faire usage d'un système de checkpoints ou points de sauvegarde afin de pouvoir poser des questions subséquentes. Or, dans le cadre de ce projet, ce n'est pas nécessairement pertinent.

Après avoir énoncé et approfondi quelques technologies, nous nous apprêtons à présent à présenter quelques prototypes implémentés afin d'évaluer les modèles étudiés les plus prometteurs.

Afin de pouvoir correctement assimiler le fonctionnement des bots qui vont suivre, il est important d'expliciter le fonctionnement de Chatbots par NLU.

3.3. CHATBOTS PAR NLU

À noter que les terminologies explicitées ci-dessous sont des termes trouvés principalement dans des Chatbots créés avec Rasa Open Source.

a) LES INTENTS

Le fonctionnement des Chatbots par NLU commence premièrement par la détection d'**intents**.

Un intent, soit une intention en français, est généralement un type d'action prédéfini dans le modèle d'I.A. utilisé par le Chatbot et servant à démarquer toutes les possibles actions que pourrait entreprendre l'utilisateur.

Un exemple d'intent peut être « **DETAILS_RESERVATION** » et « **RESTAURANT_RESERVATION** ». Dans un premier lieu, l'utilisateur démarre la conversation avec le Chatbot et à un certain point il pourrait dire quelque chose de la

sorte : « J'aimerais réserver une table chez <Nom restaurant> ». L'intent qui sera dégagé ici sera très probablement **RESTAURANT_RESERVATION** car dans ce dernier on pourrait définir que le fait de trouver les mots suivants par l'usage de techniques de NLP : « <nom_restaurant> » et « réserver » consisterait à indiquer l'intention de l'utilisateur à réserver une « table » dans un restaurant. Ensuite il est possible de faire en sorte que le bot demande à l'utilisateur des précisions sur sa réservation et si l'utilisateur fournit les paramètres de la réservation attendus par le bot, l'intention **DETAILS_RESERVATION** sera détectée et les informations fournies par l'utilisateur traitées en conséquence.

b) LES ENTITÉS

Un élément important composant bien souvent des intents sont les Entités ou Entities. Les entités peuvent être définies de multiples manières dans le contexte de développement de Chatbot par NLU mais elles sont généralement représentées par une liste de mots étant associés à une entité en particulier.

Par exemple, une entité définie comme “légumes” pourrait être trouvée lors de l'analyse de textes si les mots “Oignon”, “Carotte”, “Choux blancs”, “Poivron” et bien d'autres, sont détectés.

c) LES STORIES

Afin de faire sens d'une multitude d'intents, l'usage de stories est primordiale pour parvenir à mettre en place un flux logique de conversation et principalement pour l'entraînement du Chatbot qui sans stories, aurait de la peine à associer correctement les divers intents détectés à une action donnée.

Par exemple, si l'on a deux stories définies comme:

- Story 1

- Intent: intent_1
- Intent: intent_2
- Action: bonjour

Et

- Story 2
 - Intent: intent_1
 - Action: bienvenue

Et qu'ensuite, quand l'utilisateur envoie au Chatbot une requête contenant l'intent_1 uniquement, celui-ci sélectionne la story 2. Si la requête contient également l'intent_2, alors la story 1 sera sélectionnée en fin de compte.

d) LES ACTIONS

Un élément apparaissant dans l'explication des stories ci-dessus sont les actions. Une action comme son nom l'indique est une suite d'instructions auxquelles la story courante fait appel. Son fonctionnement est techniquement identique à celui d'une fonction en programmation mais incorporée dans le fonctionnement de Chatbots par NLU.

e) LES RÈGLES

Les règles sont des suites d'actions qui sont exécutées impérativement selon l'intent ou condition spécifiée au préalable. Les règles sont similaires fonctionnellement parlant aux stories mais avec la différence qu'elles s'activent même si l'on est déjà à l'intérieur d'une story.

Par exemple, si une règle définit que lorsqu'**intent_1** est détecté, un message doit être affiché. Si l'on se trouve dans quelque story et qu'**intent_1** est donc trouvé, le message est immédiatement envoyé en plus des autres messages éventuels du Chatbot.

f) LES SLOTS

Lorsque la conversation avec l'utilisateur progresse, il est possible que des informations fournies par l'utilisateur doivent être stockées afin d'influencer le reste de la conversation. C'est dans ce cas présent que les slots entrent en jeu. Les slots sont une façon pour le Chatbot d'enregistrer en mémoire le contenu d'entités ou de tout autre type d'information le temps d'une conversation avec un utilisateur donné.

g) LES LOOKUP TABLES

Il existe une multitude de cas où nous souhaiterions détecter une entité mais qu'il existe une variété de mots correspondant à cette dernière. C'est pourquoi on fait usage d'une lookup table. Les lookup tables fonctionnement d'une manière similaire à une expression régulière dans le sens où si un mot se trouve dans une liste de mots associés à une entité, l'entité en question sera définie comme détectée dans le traitement du message utilisateur courant.

Par exemple, si l'on possède une entité "meuble" et que l'on ne souhaite pas à avoir à définir une entité pour chaque meuble, on peut alors définir une lookup table contenant "Table", "Chaise", "Armoire", "Canapé" et bien d'autres. Si un seul de ces mots figure dans le message de l'utilisateur, alors l'entité meuble sera détectée.

h) LES FORMS

Les forms ou formulaires sont des moules pouvant être utilisés dans de multiples scénarios où l'on veut s'assurer de récupérer un certain nombre de slots. Pour se faire, on définit un formulaire et chaque slot associé et une fois cela fait, on utiliser le formulaire nouvellement créé.

Par exemple, si l'on possède un formulaire identité avec comme slots requis “nom” et “prénom”, on peut paramétrier le Chatbot de sorte qu'il persiste à demander dans la même boucle de conversation tant qu'il n'a pas reçu le **nom** et **prénom** de l'utilisateur.

i) LES PIPELINES

Dans le domaine du Machine Learning, une pipeline est la description d'une suite de traitements par lesquels passent des données allant de l'entrée à la sortie du modèle. Dans le cadre du NLU, les éléments d'une pipeline typique sont des traitements NLP mentionnés précédemment qui, mis ensemble, permettent de créer un modèle fonctionnel.

Chapitre 4 : OUTILS EXISTANTS

4.1. RASA

La première technologie pouvant servir dans le cadre de la réalisation d'un Chatbot est Rasa. Rasa est un framework de création d'assistants conversationnels intelligents à l'aide de diverses techniques de machine learning créée en 2016.

a) RASA X

Avant toute chose, il est pertinent de mentionner qu'il existe un moyen d'entrainer, paramétrier et tester des chatbots créés avec une interface graphique mise à disposition par Rasa étant Rasa X.

Voici à présent ci-dessous un bref aperçu de son utilisation grâce à un tutoriel disponible sur la section blog de Rasa⁹:

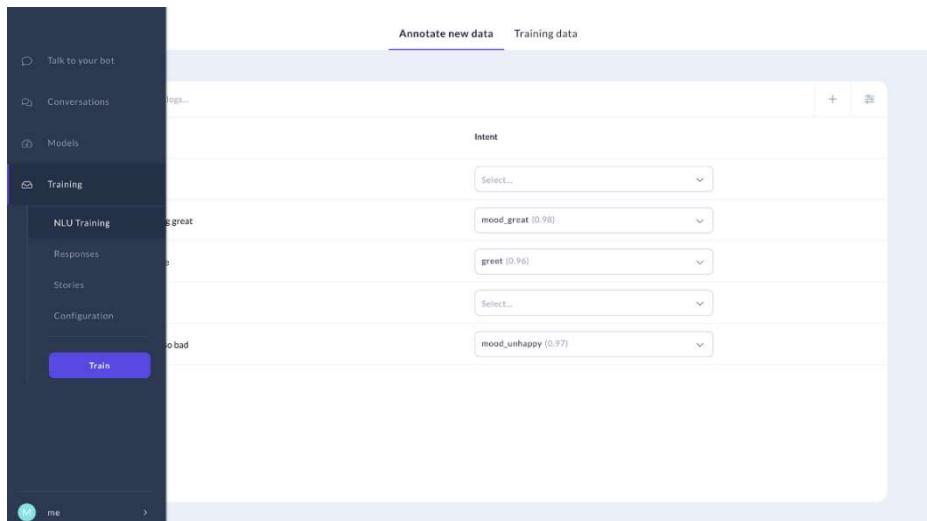


Illustration 14: Interface contenant le menu de Rasa X

Source : Rasa.com, réf. : URL07

⁹ GETTING STARTED WITH RASA X AS AN EXISTING RASA USER | RASA BLOG, RASA [EN LIGNE]. DISPONIBLE À L'ADRESSE : HTTPS://RASA.COM/BLOG/RASA-X-GETTING-STARTED-AS-A-CURRENT-RASA-USER/ [CONSULTÉ LE 3 MARS 2024].

Lorsque l'on démarre Rasa X, le menu que l'on voit ci-dessus présente les diverses pages auxquelles on peut accéder afin de :

- Gérer les modèles existant
- Entrainer un modèle
- Modifier les stories
- Modifier la configuration de la pipeline d'un modèle
- Visualiser l'historique des conversations
- Ouvrir une fenêtre de chat

Si l'on ouvre Rasa X dans un répertoire contenant un projet existant et possédant déjà des modèles pré-entraînés, ces derniers ainsi que leurs configurations apparaîtront dans les diverses pages mentionnées ci-dessus.

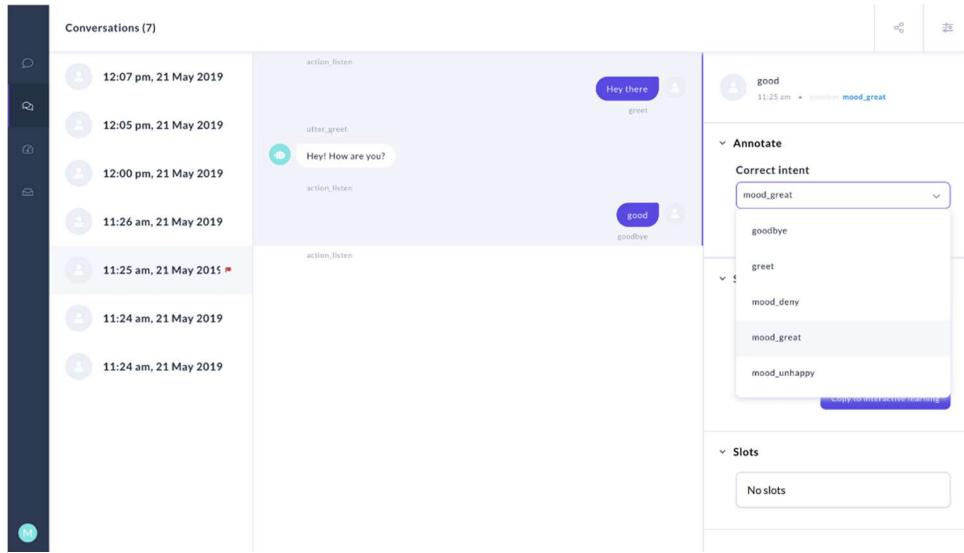


Illustration 16: Example de conversation parmi l'historique des conversations

Source : Rasa.com, réf. : URL08

Ci-dessus se trouve un exemple de conversation entre l'utilisateur et le chatbot et on peut apercevoir qu'il est possible de corriger presque en temps réel les prédictions faites par pas le bot si l'intent trouvé est erroné et d'autres paramètres tels que les slots afin de vérifier si les informations stockées sont correctes.

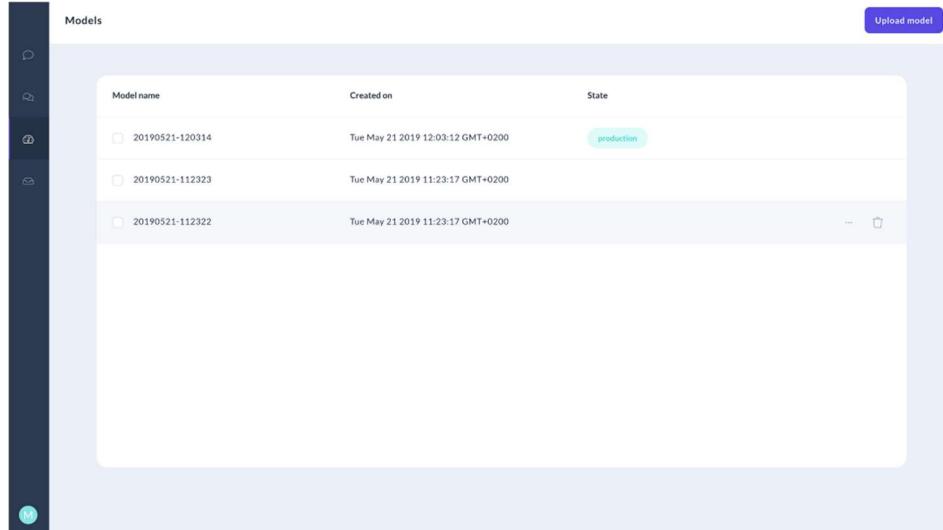


Illustration 15: Page de gestion des modèles avec Rasa X

Source : Rasa.com, réf. : URL09

Cette dernière image démontre la facilité de gestion des modèles existants, d'en supprimer ou ajouter de nouveaux au besoin et ce en quelques clics.

b) RASA OPEN SOURCE

Rasa Open Source est la solution Open Source de Rasa mettant à disposition tout un SDK disponible uniquement en python. Bien qu'il ne soit que disponible en Python il est toutefois possible d'utiliser quelconque autre language pour traiter les données reçues du Chatbot Rasa étant donné qu'il fonctionne grâce à un serveur REST avec des points de sortie pouvant être appelé au besoin.

Rasa Open Source met à disposition un wiki très détaillé quand à l'usage du SDK. Le fonctionnement de Rasa Open Source peut être décomposé en trois parties principales.

La première, le Rasa NLU, correspond au moteur NLP défini plus tôt dans la définition d'un Chatbot. Ce dernier va se charger de détecter les divers entités et intents et de les extraire.

Le deuxième, le Rasa Core, correspond au moteur de réponses défini plus tôt dans la définition d'un Chatbot. Le Core gère tout l'aspect conversationnel en intégrant les entités et intents extraites par le Rasa NLU et de les intégrer au système de dialogues définis par les stories et règles configurées au préalable.

La troisième, L'agent Rasa, permet de fournir l'interface logicielle qui permet d'intéragir avec les deux composants précédents. L'agent permet de démarrer un entraînement de modèle, gérer la réception de messages, le chargement de modèles de dialogue, la réception d'actions et la gestion de canaux externes afin de permettre au Chatbot Rasa de communiquer directement sur des applications de messageries connues comme Facebook Messenger, Slack, Mattermost et même de mettre en place une intégration sur son propre site personnel.

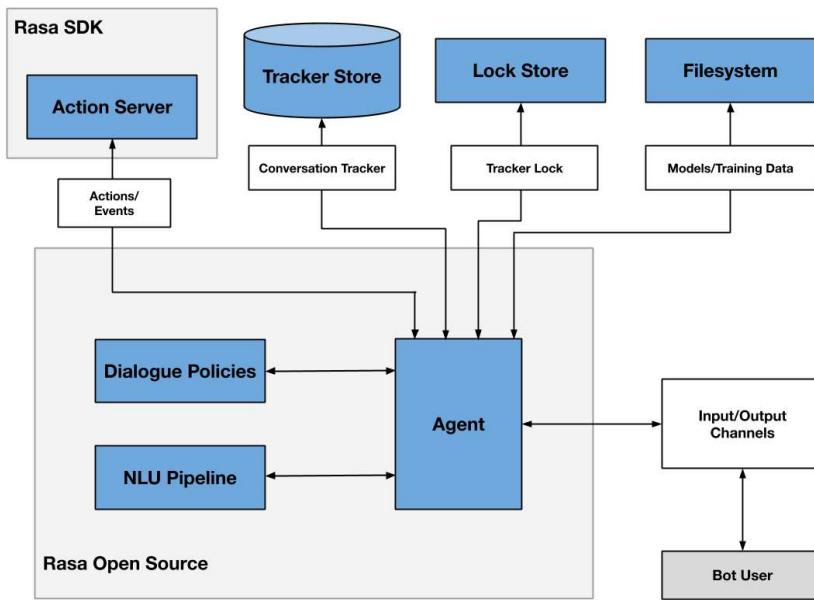


Illustration 17: Architecture Rasa

Source : Rasa.com, réf. : URL10

Rasa Core est représenté ici comme « Dialogue Policies »

Rasa NLU est représenté ici comme « NLU Pipeline »

En plus de ces trois composants, il est possible d'en ajouter quelques autres afin de rendre le Chatbot le plus modulable, efficace et maintenable :

Un quatrième composant dont on peut faire usage est un Rasa Action Server. Ce dernier utilise le Rasa SDK en python et permet fonctionnellement parlant d'utiliser les mêmes actions implicites comme explicites que les diverses commandes Rasa peuvent effectuer telles que :

- des Stories
- affectations de slots
- lancement d'évènement
- Définition de checkpoints

Le Rasa Action Server est utilisé dans le cas où l'on souhaite intégrer des actions personnalisées qui doivent, par exemple, interagir avec une API ou effectuer une multitude d'actions qui ne peuvent pas être simplement définies dans des fichiers YAML.

Un cinquième composant est Rasa Tracker Store. Un élément récurrent lors du développement avec le SDK de Rasa est le Tracker. Le Tracker est en pratique, la mémoire du Chatbot. Par défaut, le Tracker enregistre l'ensemble des conversations et événements passés. Comme le Tracker enregistre le tout en mémoire, si le bot est redémarré, l'ensemble des données seront perdues. C'est dans ce cas qu'il est possible d'utiliser un Tracker Store qui permet de lier le Tracker Rasa à une base de données pour stocker toutes ces informations.

Un dernier composant pouvant être utilisé est le Rasa NLG Server. Lors de l'entraînement du Chatbot, l'ensemble des réponses sont sauvegardées dans le modèle. Ainsi, si l'on souhaite modifier ne serait-ce qu'un caractère dans une réponse, il serait nécessaire de réentraîner l'ensemble du modèle. Afin d'éviter de perdre du temps à réentraîner le modèle constamment, on peut faire usage d'un NLG Server qui aura pour but de fournir les réponses à la requête utilisateur selon divers paramètres, séparant ainsi l'aspect de gestion de conversations et de gestion de réponses. Comme l'ensemble des réponses se trouve sur le serveur NLG, il ne suffit que de le redémarrer pour mettre à jour les réponses pouvant être retournées à l'utilisateur par le Chatbot.

Les divers composants présentés ci-dessus sont une liste non exhaustive des composants mis à disposition par l'outil Rasa et grâce à ceux-ci, on peut déjà si l'on le souhaite mettre en place un Chatbot bien assez robuste et efficace.

c) RASA PRO

Rasa Pro est un service fourni par Rasa qui contrairement à sa contrepartie Rasa Open Source, est-elle Open Core. Open Core dans ce cas-ci voulant dire que toutes les

fonctionnalités de Rasa Open Source sont utilisables par les usagers de Rasa Pro. Sauf que les usagers de Rasa Pro possèdent des fonctionnalités supplémentaires et propriétaires.

Ce qui est ressorti au long de mes recherches sur Rasa Pro est que ce service fournit principalement un produit : CALM.

CALM soit **Conversational AI with Language Models** (I.A. conversationnelle avec des modèles de langage), permet de presque entièrement se débarrasser du fonctionnement du Chatbot par NLU de Rasa et de faire usage de divers fournisseurs de LLM tels que OpenAI¹⁰, Cohere¹¹, Hugging Face Hub¹² ou Vertex AI¹³. Il est possible de faire usage d'autres fournisseurs mais cela nécessite une configuration manuelle.

CALM permet de se débarrasser du système d'entités et de stories et de faire à la place usage d'un couplage entre des Flows et un LLM au choix.

Un Flow est similaire à une story dans l'idée qu'il décrit la logique conversationnelle mais à la différence d'une story, un Flow n'a pas besoin de spécifier tous les branchements différents de conversation. À la place, il ne suffit que de définir une **description** qui sera utilisée lors de la communication du message de l'utilisateur au LLM afin de pouvoir déterminer l'action à exécuter ensuite. Il est aussi possible de faire usage de **Slots** et de **Conditions** qui seront ensuite appliqués sur ces slots selon l'état de ces derniers. Similairement à des Forms il est possible de faire usage de **Collects** qui ne passera pas à la suite de la conversation tant que l'utilisateur n'a pas donné les informations souhaitées.

Il existe actuellement des limitations aux Flows qui sont dû à la taille des contextes mis à disposition par les divers LLMs disponibles. La quantité de Flows pouvant être gérés par Rasa Pro dépend de :

¹⁰ [HTTPS://OPENAI.COM/](https://openai.com/)

¹¹ [HTTPS://COHERE.COM/](https://cohere.com/)

¹² [HTTPS://HUGGINGFACE.CO](https://huggingface.co)

¹³ [HTTPS://CLOUD.GOOGLE.COM/VERTEX-AI](https://cloud.google.com/vertex-ai)

- La longueur des descriptions des Flow
- Le nombre de slots dans chaque Flow
- La longueur de la description de chaque slot
- Les types de slots et nombres de valeurs autorisées

De plus, grâce à l'usage d'un LLM, il est possible de l'intégrer au processus de NLG du Chatbot afin de pouvoir générer sur mesure et dynamiquement des réponses aux divers messages de l'utilisateur.

Une multitude d'autres fonctionnalités supplémentaires sont mis à disposition par Rasa Pro mais sont plus liés à de l'aide à la mise en place d'infrastructures autour du Bot et pas aussi conséquents que leur fonctionnalité CALM.

Il n'existe pas de prix fixe pour faire l'acquisition de Rasa Pro car le coût varie selon le client et ce qu'il cherche à réaliser avec ce dernier. Pour en faire l'acquisition il faut contacter directement le département des ventes de Rasa.

d) DUCKLING

Un composant découvert lors de la réalisation du prototype énoncé plus tard est Duckling. Duckling est une librairie écrite en Haskell créée par Facebook en 2017 et servant à parser du texte en une structure de donnée. Ce composant est particulièrement important car il permet de valider/vérifier aisément une multitude de données et de les rendre en un format plus compréhensible par le code.

Les divers types de données étant actuellement supportées par Duckling sont appelés **domaines** et les domaines disponibles sont les suivants selon la documentation¹⁴:

¹⁴ FACEBOOK/DUCKLING [LOGICIEL] [EN LIGNE]. 21 FÉVRIER 2024. META. [CONSULTÉ LE 21 FÉVRIER 2024]. DISPONIBLE À L'ADRESSE : [HTTPS://GITHUB.COM/FACEBOOK/DUCKLING](https://github.com/facebook/duckling) [CONSULTÉ LE 21 FÉVRIER 2024].

Dimension	Exemple d'entrée	Example de sortie
Quantité d'argent	42€	{"value":42,"type":"value","unit":"EUR"}
Numéro de carte de crédit	4111-1111-1111-1111	{"value":"4111111111111111","issuer":"visa"}
Distance	6 miles	{"value":6,"type":"value","unit":"mile"}
Durée	3 mins	{"value":3,"minute":3,"unit":"minute","normalized": {"value":180,"unit":"second"}}
Email	<u>duckling-team@fb.com</u>	{"value":"duckling-team@fb.com"}
Nombre	Quarante-cinq	{"value":45,"type":"value"}
Nombre Ordinal	22ème	{"value":22,"type":"value"}
Numéro de téléphone	+1 (650) 123-4567	{"value": "(+1) 6501234567"}
Température	28°C	{"value":28,"type":"value","unit":"celcius"}
Adresse Web	<u>https://api.wit.ai/message?q=hi</u>	{"value":"https://api.wit.ai/message?q=hi","domain":"api.wit.ai"}
Volume	4 litres	{"value":4,"type":"value","unit":"litre"}

Quantité	3 cuillères à soupe de sucre	{"value":3,"type":"value","product":"sucre","unit":"tablespoon"}
Temps	Le 24 janvier 2024 à 9h00	{"values":[{"value":"2024-01-24T09:00:00.000-08:00","grain":"hour","type":"value"}],"value":"2016-12-14T09:00:00.000-08:00","grain":"hour","type":"value"}

Tableau 2: Tableau de dimensions de Duckling

Il est d'ailleurs possible d'utiliser Duckling dans divers langages et même d'ajouter soi-même des dimensions personnalisées.

4.2. CHATTERBOT

Chatterbot est une librairie Python Open Source créée en 2014 qui permet l'élaboration de Chatbots capables d'engager dans des conversations avec un utilisateur. Chatterbot est capable de recevoir les messages utilisateur en entrée de par divers sources telles que par appels d'API, reconnaissance vocale, entrées console, etc.

À la différence de Rasa, Chatterbot n'utilise pas de techniques de compréhension de texte avancées telles que du NLU mais utilise plutôt de la détection de patterns et de similarités dans le texte. Pour parvenir à déterminer la réponse adéquate dans un cas donné, Chatterbot utilise des algorithmes de machine learning entraînés sur des conversations préparées à l'avance.

4.3. MICROSOFT BOT FRAMEWORK

Le Microsoft Bot Framework est un framework créé par Microsoft en Mars 2016 permettant de mettre en place un Chatbot conversationnel dans divers langages tels que

C#, JavaScript, Python et Java (À noter que le SDK de Java n'est plus supporté depuis Novembre 2023).

Ce framework offre une panoplie de fonctionnalités similaires à Rasa comme :

- Un SDK de création de Bot
- Intégration possible de diverses applications de communication/messageries comme Microsoft Teams, Slack, Telegram et bien d'autres
- Possibilité d'utiliser du NLP et plus précisément du NLU avec LUIS (**L**anguage **U**nderstanding **I**ntelligent **S**ervice)
- Gestion de Dialogues et Conversations
- Outils d'analyses de performances et de surveillance
- Intégration facilitée avec d'autres services Microsoft comme Azure

Ce framework n'est hélas pas open source ou gratuit et nécessite un abonnement pour en faire usage pleinement. Il existe cependant un modèle gratuit mais avec des fonctionnalités réduites et un nombre de messages pouvant être envoyés aux divers canaux de messageries limité.

4.4. NLTK

NLTK est un package Python créé en 2001 et permettant de faire usage d'une multitude d'outils NLP. NLTK en tant que tel ne peut pas créer de Chatbots mais son usage peut être très intéressant lorsque l'on souhaite appliquer des pré-traitements à une entrée utilisateur avant de fournir ce dernier au chatbot pour accroître la précision des réponses et efficacité du bot.

4.5. OPENNLP

OpenNLP est un package développé par la fondation Apache depuis 2001 et ayant pour principale utilisation, comme NLTK, l'usage des divers outils NLP qu'il propose. Cependant, il est important de noter que contrairement à NLTK qui est en Python, NLTK ne peut être qu'utilisé en Java et langages supportant l'interopérabilité avec Java comme Scala ou Kotlin. De plus, le langage Python est souvent favorisé dû à la grande quantité de modèles existant et le nombre de librairies à disposition pour divers processus de machine learning.

Chapitre 5 : PROTOTYPAGES ET ÉVALUATIONS

Afin d'évaluer certaines technologies ainsi que la faisabilité du projet en faisant usage de ces dernières, quelques prototypes ont été réalisés.

5.1. PROTOTYPES NLP

Dans un premier lieu, nous avons pensé faire usage uniquement de librairies fournissant des outils NLP afin de réaliser le moteur de traitement de données par NLU qui permettra de dégager les mots-clés nécessaires à l'établissement de règles pour mettre en place un Chatbot par règles.

Pour se faire, deux librairies ont été testées. Une en Java : OpenNLP et l'autre en Python : NLTK.

a) JAVA

Concernant le premier prototype en Java avec OpenNLP, dans un premier temps un Tokenizer a été implémenté afin de pouvoir séparer chaque mot dans les phrases en tokens afin de faciliter la suite des opérations.

```
Phrase en entrée : Je veux réserver un terrain de tennis

Je
veux
réserver
un
terrain
de
tennis
```

Comme on le remarque, la sortie de ce programme est bien la phrase d'origine mais correctement tokenisée. Cependant, la raison pour laquelle nous n'avons pas poussé bien plus loin le développement de ce prototype est pour la simple et bonne raison que bien qu'il existe une certaine variété de librairies, modèles de machine learning pré-entraînés et jeux de données divers; ces derniers ne sont pas en nombre assez conséquent et reviendraient à demander une charge de travail bien plus grande que si le prototype était réalisé en Python. En effet, le manque niveau quantité des éléments précédemment cités implique que pour avoir un moteur de traitement NLP en Java, il faudrait créer un bon nombre de jeux de données qui n'existent simplement pas pour les librairies Java existantes.

b) PYTHON

Après la tentative précédente en Java, nous avons donc décidé de changer le langage pour passer à Python et faire usage de la librairie NLTK.

En l'état, le prototype permet de récupérer une entrée texte ou un fichier et de traiter ce dernier afin de recevoir en sortie une décomposition de chaque phrase avec chaque mot devenu des tokens et ayant leur classe grammaticale spécifiée. De plus, un traitement appliqué après le traitement par POS tagging est l'application d'un NER afin de détecter quand une date se trouve dans une phrase et lorsqu'un sport comme Tennis est détecté.

Texte de base : "Je souhaite réserver un terrain de tennis. Je veux réserver pour le 24 Novembre 2024."

NEXT SENTENCE:

(Je, PRON) (souhaite, VERB) (réserver, VERB) (un, DET) (terrain, NOUN) (de,
ADP) (tennis, SPORT) (., PUNCT)

NEXT SENTENCE:

Special case(le 24 Novembre 2024, DATE)

(Je, PRON) (veux, VERB) (réserver, VERB) (pour, ADP) (('le', 'DET'), ('24',
'NUM')) (., PUNCT)

Afin de parvenir à ce résultat, divers jeux de données et librairies ont été utilisés :

- Punkt Tokenizer disponible avec NLTK : Punkt fournit des fonctions liées à la tokenisation de phrases et mots.
- Maxent_ne_chunker disponible avec NLTK : Maxent permet d'effectuer du « chunking ». Le chunking est le fait de prendre de multiples mots ayant été traités par du POS Tagging au préalable et selon des patrons donnés, ressortir un chunk. Un chunk est une nouvelle entité qui est le produit de la combinaison d'un certain nombre de tokens. L'exemple dans le cas du Prototype est que dans son implémentation actuelle, une date est détectée s'il y a au moins un déterminant, un nombre, un nom et à nouveau un nombre ce qui correspond à « Le 17 Janvier 2024 » par exemple.

Le résultat sera donc l'ensemble de ces tokens agrégés en une entité de type **DATE**.

- Averaged_perceptron_Tagger disponible avec NLTK : Le but d'un tagger est de, selon un modèle fourni au préalable, de tagger les divers tokens selon leur POS correspondant ou classe grammaticale en Français. Le modèle de POS Tagging utilisé ici est celui de Stanford étant disponible en Français parmi d'autres langues¹⁵.
- Words Corpora disponible avec NLTK : Bien que ce package n'est pas directement utilisé dans le prototype, il serait utile dans le cadre d'identification d'erreurs orthographiques dans l'entrée utilisateur grâce à l'usage de la Distance de Levenshtein ou Edit Distance qui va permettre de mesurer la distance entre

¹⁵ THE STANFORD NATURAL LANGUAGE PROCESSING GROUP, [EN LIGNE]. DISPONIBLE À L'ADRESSE : [HTTPS://NLP.STANFORD.EDU/SOFTWARE/TAGGER.SHTML](https://nlp.stanford.edu/software/tagger.shtml) [CONSULTÉ LE 5 MARS 2024].

deux mots commet étant « Le nombre de suppressions, insertions ou substitutions requises pour transformer le mot initial à celui comparé »¹⁶.

Ce prototype fut intéressant car il a permis de prendre en main diverses techniques de NLP et de mieux assimiler leur fonctionnement afin de pouvoir davantage se projeter dans la bonne direction quant à l'usage de NLP pour la réalisation du Chatbot. Or, nous l'avons vite compris que le Chatbot ne serait pas réalisé de zéro avec seulement des librairies NLP pour réaliser un Chatbot de règles.

Cependant, nous ne laissons pas complètement de côté ces librairies et jeux de données employés dans le prototype Python car ils pourraient être très utile dans le cadre d'un pré-traitement de texte avant de le passer au Chatbot comme pour appliquer une correction de fautes d'orthographe pour réduire la possibilité que le Chatbot ne comprenne pas un message.

5.2. PROTOTYPE TAPAS

En attendant, une autre problématique que nous avons tenté de résoudre était celle de la recherche de ressources. Dans le cadre de ce projet, l'utilisateur sera éventuellement capable de demander une réservation pour une ressource donnée et que l'entrée utilisateur soit utilisée par le Chatbot pour qu'il puisse se renseigner sur ces mêmes ressources. C'est donc là qu'est venu l'idée de faire usage de TAPAS, qui comme expliqué plus tôt, permet de parcourir un tableau de données par langage naturel ou en d'autres termes, du texte.

Le langage préféré par la majorité de ceux qui ont employé TAPAS est Python de par le fait que le modèle réalisé par Google est aussi en Python.

¹⁶ MEHTA, ANSH ET AL., 2021. SPELL CORRECTION AND SUGGESTION USING LEVENSHTEIN DISTANCE. . VOL. 08, NO 09.

De ce fait, un petit prototype a été réalisé avec l'aide d'une fondation de code venant de Christian Versloot sur divers exemples d'implémentations de modèles de Machine Learning, dont TAPAS.¹⁷

L'exemple implémenté pour tester le modèle est le suivant :

a) DONNÉES DE TEST

Un certain nombre de Terrains de sport sont définis dans un fichier Terrains.csv qui comporte des données comme suit :

Id	Sport	Nom Terrain	Horaire
1	Tennis	1	1
4	Tennis	2	2
5	Badminton	1	1
8	Petanque	1	1

Tableau 3: Extrait de Terrains.csv

Source : Rodrigues dos Santos Fabio

<i>Id</i>	<i>Nom</i>	<i>Lundi</i>	<i>Mardi</i>	<i>Mercredi</i>	<i>Jeudi</i>	<i>Vendredi</i>	<i>Samedi</i>	<i>Dimanche</i>
1	<i>Horaire jour</i>	<i>9h00- 16h00</i>	<i>8h30- 15h50</i>	<i>8h30- 15h50</i>	<i>9h00- 17h00</i>	<i>11h30- 14h00</i>	<i>11h30- 14h00</i>	<i>None</i>

17 VERSLOOT, CHRISTIAN. MACHINE LEARNING FOR TABLE PARSING: TAPAS. GITHUB [EN LIGNE]. DISPONIBLE À L'ADRESSE : [HTTPS://GITHUB.COM/CHRISTIANVERSLOOT/MACHINE-LEARNING-ARTICLES/BLOB/MAIN/EASY-TABLE-PARSING-WITH-TAPAS-MACHINE-LEARNING-AND-HUGGINGFACE-TRANSFORMERS.MD](https://github.com/CHRISTIANVERSLOOT/MACHINE-LEARNING-ARTICLES/blob/main/EASY-TABLE-PARSING-WITH-TAPAS-MACHINE-LEARNING-AND-HUGGINGFACE-TRANSFORMERS.MD) [CONSULTÉ LE 5 MARS 2024].

2	<i>Horaire nuit</i>	18h00-22h00	18h00-22h00	18h00-22h00	18h00-22h00	18h00-22h00	<i>None</i>	<i>None</i>
---	---------------------	-------------	-------------	-------------	-------------	-------------	-------------	-------------

Tableau 4: Contenu de Horaires.csv

Source : Rodrigues dos Santos Fabio

b) RÉSULTATS

Avant toute chose, les requêtes décrites ci-dessous sont en anglais pour la simple et bonne raison qu'en l'état, le modèle utilisé pour la compréhension du Français a bien plus de peine à saisir l'intention des requêtes que celui en Anglais.

Le prototype va premièrement charger dans le modèle TAPAS **Terrains.csv** avec la requête suivante “What are all the available tennis courts by name column ?”. Le résultat sortant du modèle TAPAS après traitement :

```
[['Tennis 1', 'Tennis 2', 'Tennis 3', 'Tennis 4', 'Tennis 1', 'Tennis 4']]
```

Une fois parsé et utilisé comme moyen de récupérer les données dans le fichier .csv on obtient :

	Id_Dataframe	id	sport	nom	terrain	horaire
0	1	1	Tennis	1		1
10	11	1	Tennis	1		2
1	2	1	Tennis	2		1
2	3	1	Tennis	3		1
3	4	1	Tennis	4		2
11	12	1	Tennis	4		1

Comme on le voit, nous avons bien reçu l'ensemble des terrains de Tennis comme demandé. Après cela, la requête qui suit est la suivante : « Which horaire by id allows for a reservation on Tuesday at 18h30 ? »

Le modèle nous répond ensuite :

```
[['horaire nuit']]
```

c) CONCLUSION

Bien que dans le cadre de ces requêtes allant de simples à un niveau de complexité supérieur, une problématique principale posée par TAPAS est que le modèle n'est pas capable, comme une base de données traditionnelle, de faire usage de clés étrangères et de références à d'autres tableaux par un identifiant unique. De ce fait, nous sommes contraints d'appliquer des traitements supplémentaires entre chaque requête.

De plus, la raison pour laquelle les requêtes ci-dessus sont en anglais est que bien que le modèle soit capable de comprendre les requêtes effectuées en Français, il lui arrive bien plus souvent de ne pas comprendre assez clairement la nature de la demande et de ne pas retourner le résultat attendu.

Pour finir, un des problèmes majeurs est le simple format de données car dans le cas où le Chatbot parle avec une multitude de clients, ce dernier devra modifier en continu le contenu .csv en mémoire ou le fichier lui-même ce qui ajoute un temps de traitement considérable et pourrait résulter en des soucis de cohérence et de concurrence car plus il y aurait de clients, plus le bot serait lent. En effet, pour assurer le fonctionnement du système de réservation, il est nécessaire d'effectuer des pré-réservations et comme chaque instance du bot essayerait d'interagir en même temps avec le fichier, des soucis de synchronicité surviendraient.

En conclusion, TAPAS est un modèle très intéressant lorsque l'on souhaite plus aisément chercher des informations dans un tableau sans avoir à effectuer des manipulations NLP diverses soi-même. Or, dans le cadre de ce projet, l'implémentation de la gestion des réservations et des ressources serait bien trop complexe et lente. Les raisons étant :

- La quantité de données étant dépendante d'autres à l'aide de références par clé, ce que TAPAS n'arrive pas à interpréter par lui-même.
- La synchronicité du système lorsque de multiples utilisateurs viendraient à utiliser le bot en même temps

- Un traitement presque nécessaire et complexe des requêtes utilisateurs en amont.
Car dans le prototype actuel, l'utilisateur pourrait demander presque ce qu'il souhaite au bot et TAPAS n'étant qu'un outil de sélection de données n'est pas capable de permettre une véritable conversation avec choix à effectuer.
Un moyen de palier à ce problème serait de rédiger une certaine quantité de requêtes au préalable avec lesquelles on est sûrs que TAPAS nous retournera le résultat attendu. Mais cela ajoute un nouveau niveau de complexité car ces requêtes peuvent varier grandement selon le cas d'usage.
- Le taux de précision des requêtes en Français implique qu'il faut entraîner à chaque fois le modèle TAPAS avec des jeux de données en lien avec notre type de réservation et plus encore pour essayer d'atteindre un niveau acceptable de précision. Ce qui actuellement est loin d'être le cas comparé à l'alternative de faire des requêtes en Anglais.

5.3. PROTOTYPE RASA

Après les essais précédents et quelques recherches, nous nous sommes finalement penchés sur Rasa Open Source en Python pour réaliser un prototype de Chatbot très simpliste.

Tout d'abord, afin de pouvoir réaliser le Chatbot il faut des données pour l'entraînement du gestionnaire de dialogues afin que le Chatbot soit capable de les reconnaître par la suite.

a) DONNÉES D'ENTRAÎNEMENT

```
- lookup: resource
examples: |
  - Tennis
  - Tenis
  - TEnnis
  - tennnis
  - Badminton
  - badminton
  - pétanque
  - petanque
  - Pétanque
  - Petanque
```

```
- intent: book_date
examples: |
  - 29 Juillet 2024
  - 05/02/2024
  - 04.11.2025
  - le 19.01.2020
  - le 02/14/2022
  - le 21 Avril 2024
  - le 12 mars 2024
```

```
- intent: book_resource
examples: |
  - Je souhaite réserver un terrain de [tennis] (resource)
  - J'aimerais réserver un terrain de [Tennis] (resource)
  - Réserve un terrain de [Tenis] (resource)
  - Je souhaite réserver un terrain de [petanque] (resource)
  - J'aimerais réserver un terrain de [Pétanque] (resource)
  - Réserve un terrain de [Petanque] (resource)
  - Je souhaite réserver un terrain de [badminton] (resource)
  - J'aimerais réserver un terrain de [Badminton] (resource)
  - Réserve un terrain de [Badminton] (resource)
```

```
- intent: book_resource+book_date
examples: |
  - Je souhaite réserver un terrain de [petanque] (resource) pour le 6 Septembre 2025
  - J'aimerais réserver un terrain de [badminton] (resource) le 28 Octobre 2024
  - Je veux réserver un terrain de [tennis] (resource) le 4 Janvier
```

Divers Intent utilisés pour la réalisation du prototype de réservation

```
- story: reservation 2
steps:
  - intent: book_resource
  - action:
    action_reserve_resource
  - intent: book_date
  - action: action reserve date
```

```
- story: reservation 3
steps:
  - intent: book_resource+book_date
  - action: action_reserve_resource
  - action: action_reserve_date
  - checkpoint: confirm_reservation
```

```
- story: confirmation_cancelled  
steps:  
- checkpoint: confirm_reservation  
- intent: cancel  
- action: action_goodbye
```

```
- story: confirmation_confirmed  
steps:  
- checkpoint:  
confirm_reservation  
- intent: confirm  
- action:  
utter_final_reservation  
- action: action_goodbye
```

Les stories principales utilisées dans le prototype

Selon les intents actuellement présents dans le prototype, le Chatbot est capable de :

- Reconnaître quelques types de salles/terrains de sports selon le sport
- Reconnaître une demande de réservation étant soit une demande de réservation + date ou une demande de réservation simple suivie d'une date
- Reconnaître des dates grâce à l'utilisation de Duckling

Grâce ensuite aux stories décrites après, le Chatbot est capable de :

- Recevoir une demande de réservation et si aucune date n'est spécifiée, le bot demandera à quelle date
- Recevoir une demande de réservation et si une date est spécifiée, le bot va procéder à la suite de la conversation directement
- Une fois qu'une des deux manières de réserver est effectuée, le bot demandera une confirmation de la réservation et le client pourra accepter ou l'annuler.

b) EXEMPLE DE CONVERSATION

```
Votre message ->Je souhaite réserver un terrain de badminton
Rasa's response -> [{"recipient_id": "user", "text": "Et à quelle date ?"}]
Votre message ->Le 12 janvier cette année
Rasa's response -> [{"recipient_id": "user", "text": "Souhaitez-vous donc réserver pour le 2024-01-12T00:00:00.000-08:00 un terrain de badminton ?"}]
Votre message ->oui
Rasa's response -> [{"recipient_id": "user", "text": "Vous avez bien réservé pour le 2024-01-12T00:00:00.000-08:00 un terrain de badminton"}, {"recipient_id": "user", "text": "Au revoir"}]
```

```
Votre message ->Je souhaite réserver un terrain de pétanque le 29/05/2024
Rasa's response -> [{"recipient_id": "user", "text": "Souhaitez-vous donc réserver pour le 2024-05-29T00:00:00.000-07:00 un terrain de pétanque ?"}]
Votre message ->Oui
Rasa's response -> [{"recipient_id": "user", "text": "Vous avez bien réservé pour le 2024-05-29T00:00:00.000-07:00 un terrain de pétanque"}, {"recipient_id": "user", "text": "Au revoir"}]
```

Deux exemples de conversation avec le Chatbot du prototype

De par ces deux exemples de conversation avec le bot, on peut remarquer dans un premier temps le fait que le parsing des dates est géré correctement par Duckling, même quand une date comme “Le 12 janvier cette année” est fournie et que Duckling arrive à parser cette dernière correctement en “12 janvier 2024”.

De plus, on remarque bien avec les divers messages de rappels et de confirmation que le Chatbot enregistre bien les choix de l’utilisateur et les utilise pour déterminer les prochains messages comme le fait d’inclure ou non la date de la réservation directement dans le message initial.

c) CONCLUSION

Grâce à ce prototype simpliste de Chatbot conversationnel avec Rasa Open Source, on remarque à quel point il est aisément avec un petit jeu de données, de mettre en place un Chatbot ainsi que tout un dialogue de réservation de terrain de sport avec des branchements selon comment l’utilisateur répond.

En plus de son fonctionnement actuel, il y a la possibilité de lui ajouter diverses interactions avec une API, base de données, système externe, etc. L'implémentation d'une ou plusieurs de ces actions agrandirait largement la quantité de branchements et interactions complexes pouvant être effectuées avec le bot. Avec l'implémentation d'un scripts de parage de ressources fournies en CSV il serait très facile de remplir les jeux de données du Chatbot afin d'ajouter davantage de ressources et de rendre ce dernier facilement modulable dans cet aspect.

5.4. RÉFLEXION SUR L'ENSEMBLE DES PROTOTYPES

La réalisation de ces divers prototypes a permis de venir à quelques conclusions concernant la future implémentation de ce projet.

En premier lieu, concernant les prototypes NLP, nous sommes de l'avis que s'il fallait réaliser un programme faisant usage de Machine Learning ; nous le coderions en Python. Mais, par soucis d'exhaustivité, nous avons tenté une implémentation en Java en plus du langage standard dans le monde du Machine Learning, à savoir Python. Bien que le prototype en Java soit tout à fait performant et capable d'atteindre des résultats cohérents, il ne reste néanmoins pas aussi facile à utiliser et mettre en place que son compère en Python pour les raisons qui suivent :

- Davantage de solutions et librairies sont uniquement créées en Python dû à sa popularité dans ce domaine.
- Pour la même raison qu'au-dessus, il y a bien moins de modèles pré-entraînés de disponibles et compatibles avec les librairies Java de NLP. Nécessitant ainsi de réaliser une plus grande portion des jeux de données utilisés soi-même.
- Une certaine quantité de techniques NLP ou autres fonctionnalités sont manquantes en Java alors que très présentes en python.

Le prototype Python reste lui pertinent car comme précisé au premier chapitre , il y a toujours un traitement NLP qui se fait avant de procéder vers le moteur de réponses et pouvoir en appliquer soi-même à l'aide d'une librairie est toujours un plus.

Deuxièmement, concernant le prototype TAPAS, il est évident que les attentes que nous avions à l'égard de ce modèle étaient bien au-delà de la réalité et qu'il a été judicieux de réaliser un prototype initial afin de mettre à l'épreuve ce modèle pour ne pas se retrouver bloqués par la suite. TAPAS est un très bon modèle dans ce qu'il fait étant la recherche de données dans un tableau à l'aide de langage naturel. Hélas, les obstacles tels que :

- La langue, par le fait que la majorité des résultats retournés sont erronés après avoir mis en place le modèle Français. En contraste à cela, le modèle Anglais fonctionne bien mieux. Afin de recevoir de meilleurs résultats il serait nécessaire de Fine Tuner davantage et d'ajouter des jeux de données pour compléter le modèle Français et le rendre fonctionnel.
- L'impossibilité de faire usage de multiples tables à la fois sans avoir à faire recourt à une combinaison de tables et quand bien même nous avons essayé d'en combiner, les résultats étaient rarement fructueux.
- La nécessité d'être souvent réentraîné si l'on ajoute continuellement des nouvelles données et types de tableaux. Parcourir ces tableaux est fastidieux et plus coûteux que de faire des requêtes à une base de données car nécessitant des lectures de fichiers ou récupération de tableaux d'autres sources avant de même pouvoir les faire passer dans TAPAS.

Dernièrement, le prototype de Chatbot fait avec Rasa fut un des plus intéressants car permettant de simuler un véritable cas d'usage étant ici une réservation et de manière claire et efficace grâce à la documentation abondante disponible pour Rasa.

Chapitre 6 : CONCEPTION DU PROJET

Cette section et celles qui suivent présentent le travail réalisé lors de ce projet de Bachelor. Dans un premier temps, nous détaillerons les diverses technologies employées. Dans un second temps, nous verrons comment est structurée l'application dans son ensemble. Ensuite, la méthode de travail employée pour réaliser le projet. Puis, nous nous intéresserons aux composants principaux dans leur fonctionnement.

6.1. TECHNOLOGIES UTILISÉES

a) DOCKER

Docker est un logiciel sorti en 2013 et ayant pour but premier de virtualiser un système et créant ainsi des containers pour y exécuter divers programmes par exemple. Un container dans ce contexte est comme une machine virtuelle qui va imiter tout un environnement de travail d'un système d'exploitation.

La différence ici étant qu'au lieu de virtualiser l'intégralité du système et son système d'exploitation, seul l'explorateur de fichiers sera virtualisé grâce à un moteur de conteneurs Docker. De ce fait, il n'est pas nécessaire de simuler tout un système complet et le conteneur peut librement (selon ce qui a été spécifié au préalable) faire usage des divers ressources système. Une différence à noter aussi est que contrairement à une machine virtuelle l'environnement en question n'est pas entièrement isolé de la machine hôte.

Cependant dans un cas comme le nôtre où nous souhaitons simplement déployer de manière constante avec les mêmes packages et paramètres toute l'infrastructure

nécessaire au fonctionnement du bot, ce n'est pas un problème.

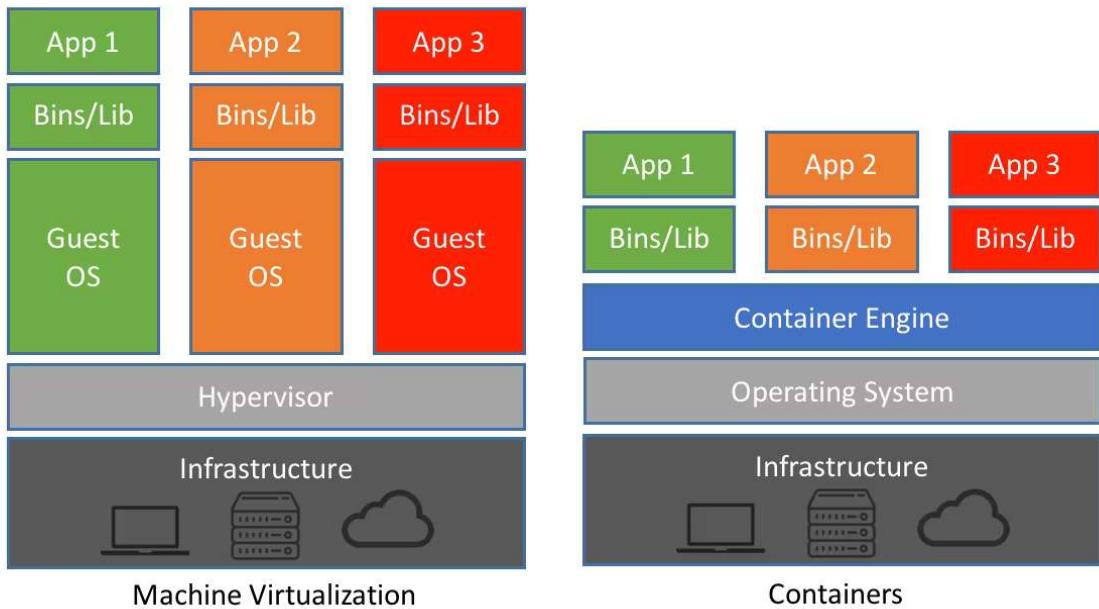


Illustration 18 : Illustration de comparaison entre une Machine Virtuelle et des conteneurs Docker

Source : Netapp.com Réf. : URL11

b) FASTAPI + UVICORN

Une API (Application Programming Interface) a pour but de mettre à disposition divers fonctionnalités et services sans pour autant qu'il soit nécessaire de les télécharger comme dans le cas de packages. Dans ce projet, nous avons donc mis en place une API réalisée en Python avec FastAPI et fonctionnant grâce à Uvicorn. Uvicorn est ici un package python qui permet la mise en place d'un serveur Web étant donné que FastAPI fournit principalement le framework pour une API REST mais pas le serveur requis.

Une API est généralement un URL auquel on envoie des requêtes et de laquelle on reçoit des informations en retour. Les requêtes sont faites sur des routes mises à disposition par l'API. Une route se présente souvent sous la forme d'un lien comme : <lien api>/Route 1, <lien api>/Route 1/Sub-route 2, ...

c) EXPRESS JS

Express JS rempli le même rôle que FastAPI à la différence du fait que le framework est disponible en Javascript/TypeScript. Express a l'avantage de pouvoir aisément intégrer une multitude de Middlewares et du fait qu'il est dans les langages mentionnés précédemment ce qui le rends intéressante lorsque l'on doit réaliser des projets web ayant des fonctionnalités autre que servir des routes. D'ailleurs, Express fait usage de NodeJS qui est un environnement serveur facilitant la mise en place et déploiement d'applications web en Frontend et Backend couramment utilisé et ce depuis bien longtemps ; justifiant ainsi davantage son utilisation dans le projet pour viser à être le plus stable et faciliter un développement continu sur le temps.

d) SOCKET.IO

Afin de pouvoir mettre en place un système de Chat, il faut pouvoir recevoir et envoyer des messages en temps réel. C'est donc ce rôle que Socket.IO va remplir. Socket.IO est une librairie disponible en Javascript, Java, Python, Golang et Rust qui permet l'ouverture de canaux de communications entre un client et un serveur. Grâce à cela, le client peut communiquer instantanément et directement avec le serveur et recevoir une réponse uniquement adressé à ce-dernier. Très utile dans le cadre de la réalisation d'un Chat textuel entre deux partis.

e) VITE + REACT

Bien que Vite et React sont deux frameworks séparés, ils sont couramment utilisés pour la réalisation de Frontends dynamiques, sans latence et modulaires.

Vite a pour principal intérêt de fournir un serveur pour héberger un frontend très facilement. Cependant ce n'est pas l'unique raison de ce choix de technologie car Vite possède une intégration native à deux autres framework de frontend single-page étant VueJS et React, ce qui dans notre cas est pertinent car nous utilisons React. Et finalement,

l'avantage d'utiliser Vite avec React est que le serveur fonctionne en mode “Hot Reload” qui va rafraîchir la page dès qu'un changement survient dans les fichiers du serveur Frontend et facilitant ainsi le développement.

React est un framework Frontend en Javascript et compatible avec Typescript qui permet d'aisément mettre en place des sites web “Single Page”. Un site Single Page a comme principal intérêt de ne pas nécessiter le rafraîchissement complet de la page car cette dernière comporte une multitude de composants qui sont mis à jour ou remplacés dynamiquement. La nature modale de ce type d'apps web le rends très attractif car le code sera naturellement modulaire et donc plus facile à maintenir et améliorer.

f) RASA

Comme discuté au chapitre « Évaluations et Prototypages », le framework Rasa est optimal en ce qui concerne les besoins du projet et a par conséquent été sélectionné comme le pilier central.

g) DUCKLING

Explicité précédemment, Duckling a pour intérêt primaire de traiter divers données comme des dates, heures, numéros de téléphones, etc. aisément et de multiples manières. Ce logiciel est inclus dans la logique du Chatbot et du traitement des entrées utilisateur.

h) POSTGRESQL

Postgresql est un moteur de base de données OpenSource. Le choix d'utiliser ce dernier est car il possède autant ou plus de fonctionnalités qu'un serveur SQL, est open source et par conséquent non sujet à des licences commerciales et car il fut utilisé lors d'un projet précédent et afin d'apprendre à mieux l'employer il est devenu le système principal pour la base de données. Et avant tout, Postgresql est une alternative réputée comme préférable à son compère SQL car OpenSource.

i) SQLALCHEMY

Dans l'optique de manipuler la base de données directement depuis le code de l'API, le package Python SQLAlchemy est employé. SQLAlchemy est un toolkit python créé en 2006 et permet d'intéragir avec des bases de données en langage SQL ainsi qu'offrir la possibilité d'utiliser de l'ORM ou Object Relational Mapping. Le principe d'ORM est explicité dans le chapitre “Réalisation du projet” mais en quelques mots permet de se débarasser de l'utilisation de requêtes SQL en dur et d'à la place utiliser des objets Python avec attributs et fonctions pour par exemple récupérer des données d'une table selon son objet. La possibilité d'employer de l'ORM est la raison principale du choix de ce package parmis d'autres.

j) API TELEGRAM

Telegram est une application de messagerie instantanée lancée en 2013. Ce service possède une multitude de fonctionnalités similaires à d'autres applications comme What's app, Facebook Messenger, etc. Mais les avantages/désavantages ne sont pas ce qui importent ici car ce service mets à disposition une API libre à l'utilisation et un moyen de mettre en place des bots sans coûts additionnels. L'API, elle, permet donc de gérer le comportement du bot dans la façon qu'il a d'intéragir avec les utilisateurs et de traiter les messages leurs étant retournés.

6.2. ARCHITECTURE GLOBALE

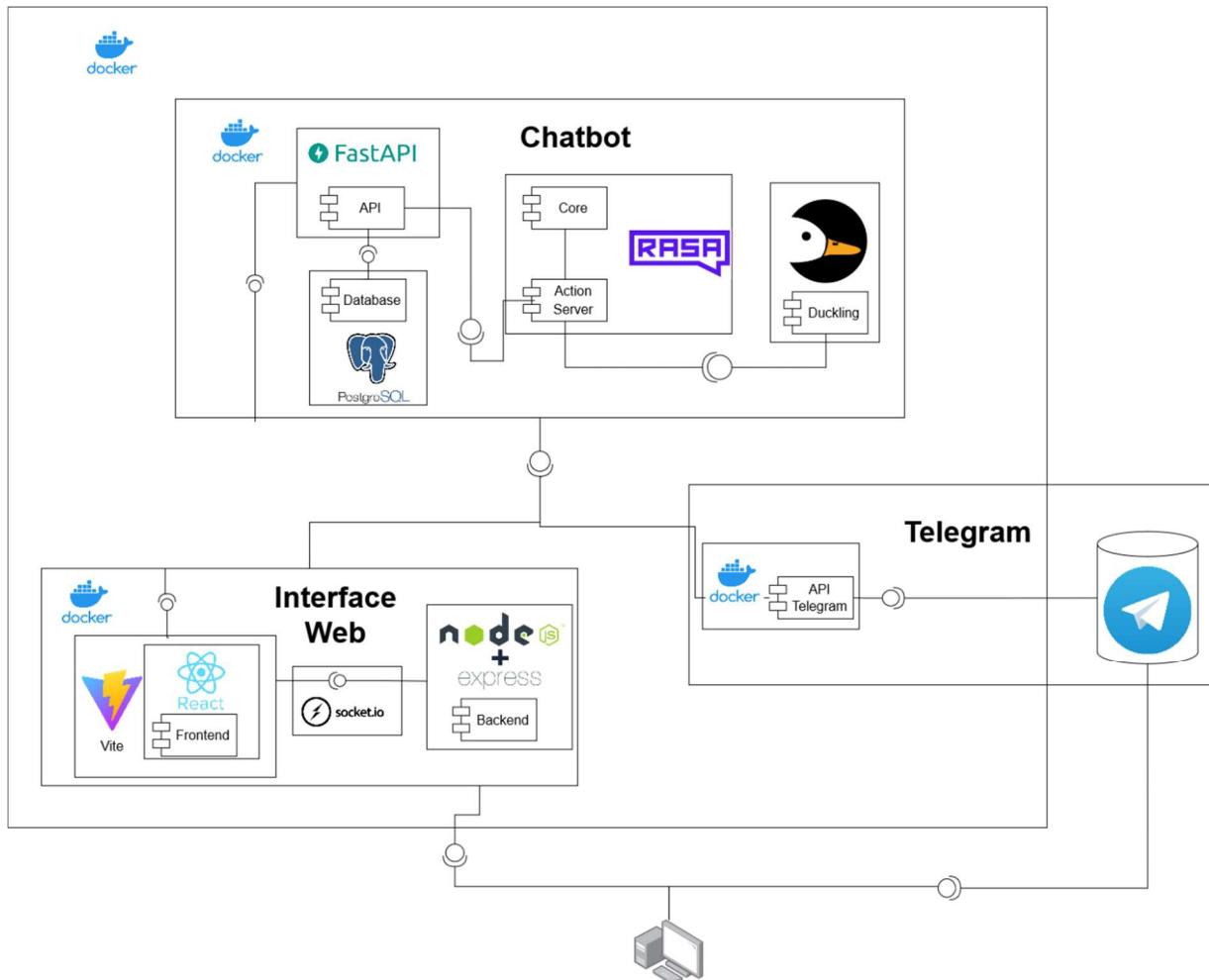


Illustration 19: Diagramme d'architecture général du projet

Source : Rodrigues dos Santos Fabio

Dans l'état, le projet peut être divisé en trois parties : **Le Chatbot**, **l'interface Web** et **l'interface Telegram**. Le chatbot concerne l'ensemble de la logique qui permet d'enregistrer des réservations et discuter avec l'utilisateur. Les deux interfaces proposées ont pour utilité de diversifier les points d'interaction avec le chatbot mis à disposition pour l'utilisateur.

Afin de faciliter l'usage du projet, il a été entièrement Dockerisé. Cela permet de réduire les soucis de réplicabilité de l'environnement requis pour utiliser les composants présents ainsi que rendre le démarrage de l'ensemble de ces derniers très aisé et rapide.

Les divers fichiers nécessaire à la dockerisation du projet se trouvent dans les dossiers spécifiés ci-dessous et tous appelés dans un fichier docker-compose commun permettant de démarrer tous les conteneurs dans un seul groupe de conteneurs liés au projet.

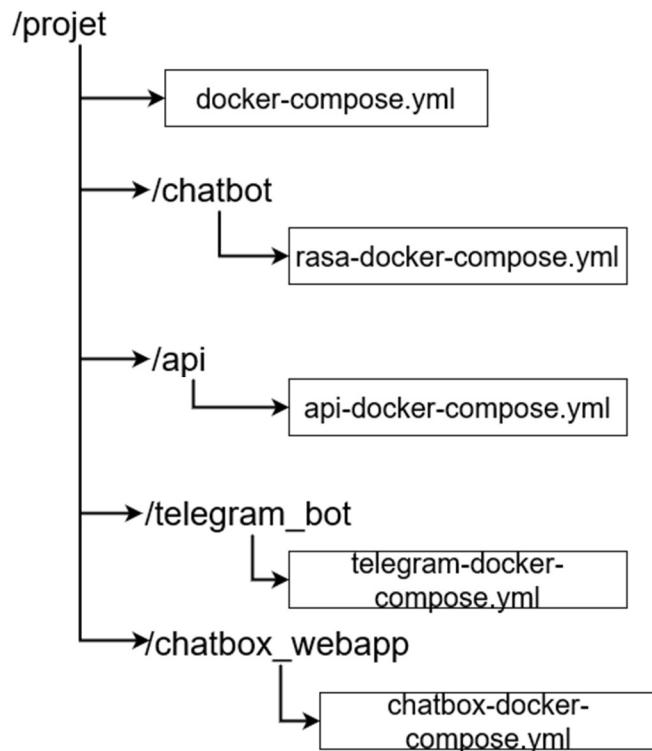


Illustration 20 : Diagramme en arbre de la structure Docker Compose du projet

Source : Rodrigues dos Santos Fabio

6.3. DÉROULEMENT DU PROJET

Afin de s'assurer du bon déroulement du projet, la plateforme d'hébergement de GITs « Gitlab » possédé par la HES de Genève a été l'élément central concernant la gestion du projet de bachelor.¹⁸

Un outil mis à disposition par Gitlab est l'issue board :

¹⁸ [HTTPS://GITEDU.HESGE.CH/NIKLAUS.EGGENBER/2324_RODRIGUESDOSANTOS](https://GITEDU.HESGE.CH/NIKLAUS.EGGENBER/2324_RODRIGUESDOSANTOS)

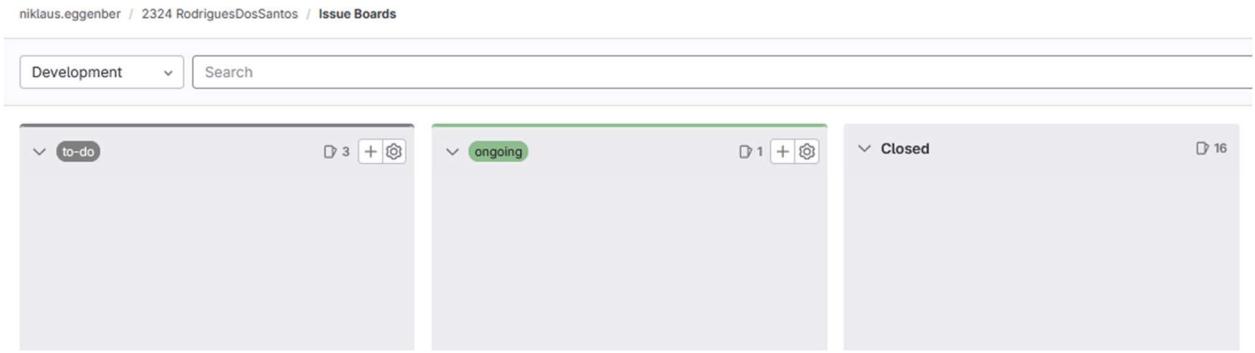


Illustration 22 : Capture d'écran d'un issue board sur Gitlab

Source: Rodrigues dos Santos Fabio

Sur GIT, un moyen de définir des tâches à réaliser est au travers d'**issues**. Une issue comporte un titre, une description, possiblement des sous-tâches et peut être labelisée.

Other labels
Difficulty-1
niklaus.eggenber / 2324 RodriguesDosSantos
Difficulty-3
niklaus.eggenber / 2324 RodriguesDosSantos
Difficulty-5
niklaus.eggenber / 2324 RodriguesDosSantos
Difficulty-7
niklaus.eggenber / 2324 RodriguesDosSantos
ongoing
niklaus.eggenber / 2324 RodriguesDosSantos
Priority-1
niklaus.eggenber / 2324 RodriguesDosSantos
Priority-2
niklaus.eggenber / 2324 RodriguesDosSantos
Priority-3
niklaus.eggenber / 2324 RodriguesDosSantos
to-do
niklaus.eggenber / 2324 RodriguesDosSantos

Illustration 21 : Liste de labels sur le GIT du projet

Source : Rodrigues dos Santos Fabio

Afin de mieux définir les tâches, ont été attribués deux labels principalement par tâches :

- Un label de difficulté

- Un label de priorité

Le label “ To-Do ” et “ Ongoing ” sont attribués automatiquement lorsque les tâches sont déplacées d'une colonne à l'autre de l'issue board. De ce fait, l'utilisation de l'issue board permet de donner un aspect similaire à celui de l'agile bien qu'aucun sprint ne fût défini chaque semaines pour la sélection des tâches à réaliser.

De sorte qu'un compte rendu du travail effectué chaque jour soit gardé, un journal de bord a été rédigé tout au long de la réalisation du projet.

Tout comme lors du projet de semestre, des rendez-vous hebdomadaires ont eu lieu avec le professeur attitré au projet avec pour but de confirmer l'avancée et orienter dans la bonne direction au besoin.

6.4. CHATBOT RASA

À présent, nous entamons la section dans laquelle nous déterminerons comment le bot doit se comporter de manière générale et quelles sont les diverses interactions qu'il va avoir entre chaque composants du Bot et ce qui va être finalement retourné à l'utilisateur.

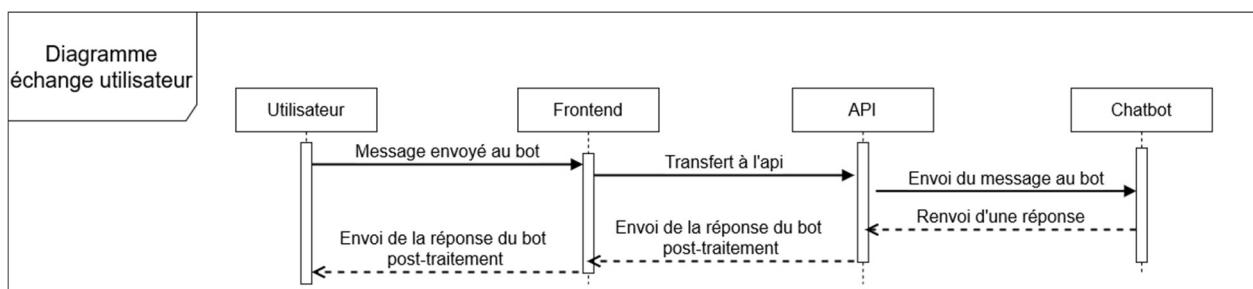


Illustration 23 : Diagramme représentant une interaction quelconque entre l'utilisateur et le bot

Source : Rodrigues dos Santos Fabio

De manière simpliste, Trois composants sont en jeu ici et permettant à l'utilisateur d'avoir des échanges. Le frontend ici pouvant être dans le cadre de ce projet à la fois celui fourni par Telegram ou le projet Web; la partie backend de chacun étant compris dans l'élément

Frontend dans le diagramme ci-dessus. L'API ici fait référence à la seule et unique interface communiquant avec le Chatbot directement. De plus, l'API communique aussi lorsque nécessaire avec la base de données pour récupérer les informations nécessaires au traitement du message. Le message reçu par le frontend de l'API sera le même qu'importe le frontend en question mais ce qui sera retourné à l'utilisateur différent car chaque frontend possède des implémentations différentes.

Intéressons-nous de plus près à un échange basique qui sera présent dans presque tous les échanges avec le bot.

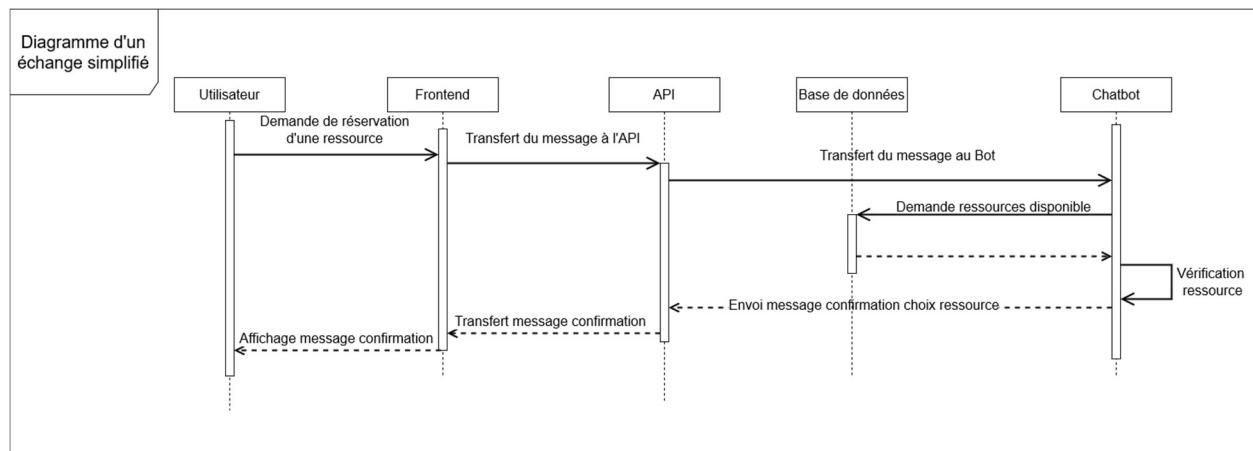


Illustration 24 : Diagramme d'un échange portant sur la demande de ressource

Source : Rodrigues dos Santos

Ci-dessus se présente un cas où un utilisateur demande à réserver une ressource. Afin de clarifier dans un premier temps les acteurs en jeu lors d'un échange Utilisateur-Chatbot, nous supposons qu'aucune erreur ne survient pendant ce dernier. Dans un premier temps, l'utilisateur pose la question depuis une l'interface frontend. Cette dernière transmet immédiatement le message à l'API et cette dernière retransmet directement le message au Chatbot. Après avoir reçu la requête, le Chatbot effectue une multitude de traitements et peut faire appel à la base de données pour récupérer des informations telles qu'ici la liste des ressources. Une fois les informations réceptionnées, il va confirmer que la ressource

existe et est disponible puis envoyer la confirmation ainsi que d'autres informations à afficher à l'utilisateur. Le/les message(s) de confirmation seront retransmis à la chaîne jusqu'à parvenir à l'utilisateur au bout.

Ce cycle se répète presque à l'identique lors de l'intégralité de l'échange entre l'utilisateur et le Bot, à quelques différences près selon le cas et la requête.

Afin de davantage comprendre le fonctionnement interne du Chatbot, nous nous penchons

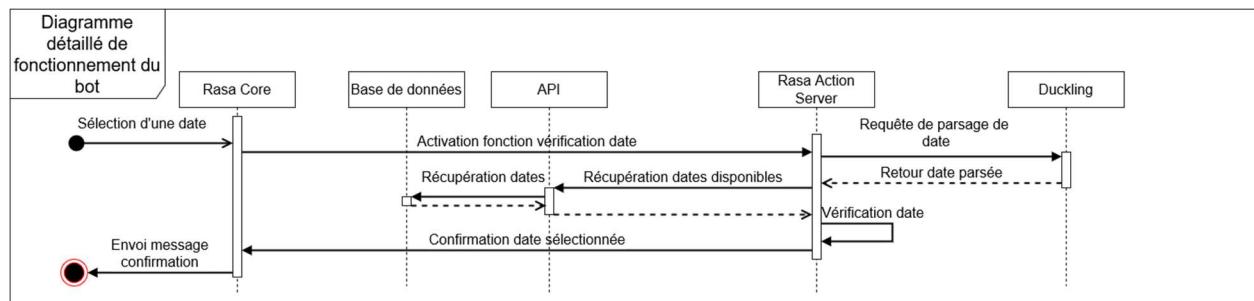


Illustration 25 : Diagramme du fonctionnement du Chatbot Rasa

Source : Rodrigues dos Santos Fabio

à présent sur la logique liée à Rasa ci-dessus. Dans ce scénario, l'utilisateur a choisi une date et cette date a été transférée jusqu'au bot qui doit à présent valider et enregistrer le choix.

Pour se faire, après l'analyse du message de l'utilisateur et récupéré son intent et ses entités par Rasa Core, Le Rasa Action Server appelle l'action de validation de date. La date est envoyée au composant Duckling afin de parser la date en un format compréhensible par le code python. Une fois cela fait, l'action server récupère les dates disponibles en demandant à l'API chargée de communiquer avec la base de données. Lorsque l'action server a reçu les dates disponibles et validé la date de l'utilisateur, elle est déclarée comme confirmée auprès de Rasa Core qui finalement envoie un message de confirmation à l'utilisateur et passer à l'étape suivante de la discussion.

6.5. BASE DE DONNÉES

Explicitons à présent le composant base de données du bot. Bien qu'un calendrier Google soit mis en place afin de stocker les réservations des clients, il est bien plus sûr et facile d'avoir une base de données répertoriant les divers réservations. Une des raisons principales étant qu'à partir de la base de données il n'est pas si compliqué d'implémenter une interface afin de visualiser les réservations. Et une autre est que cela évite d'avoir toutes ces données importantes dans un service non local et que nous ne possédons pas. D'autant plus que même si les divers ressources sont spécifiées dans des fichiers fournis au Chatbot, il est davantage plus rapide d'appeler une base de données que de les ouvrir à chaque fois.

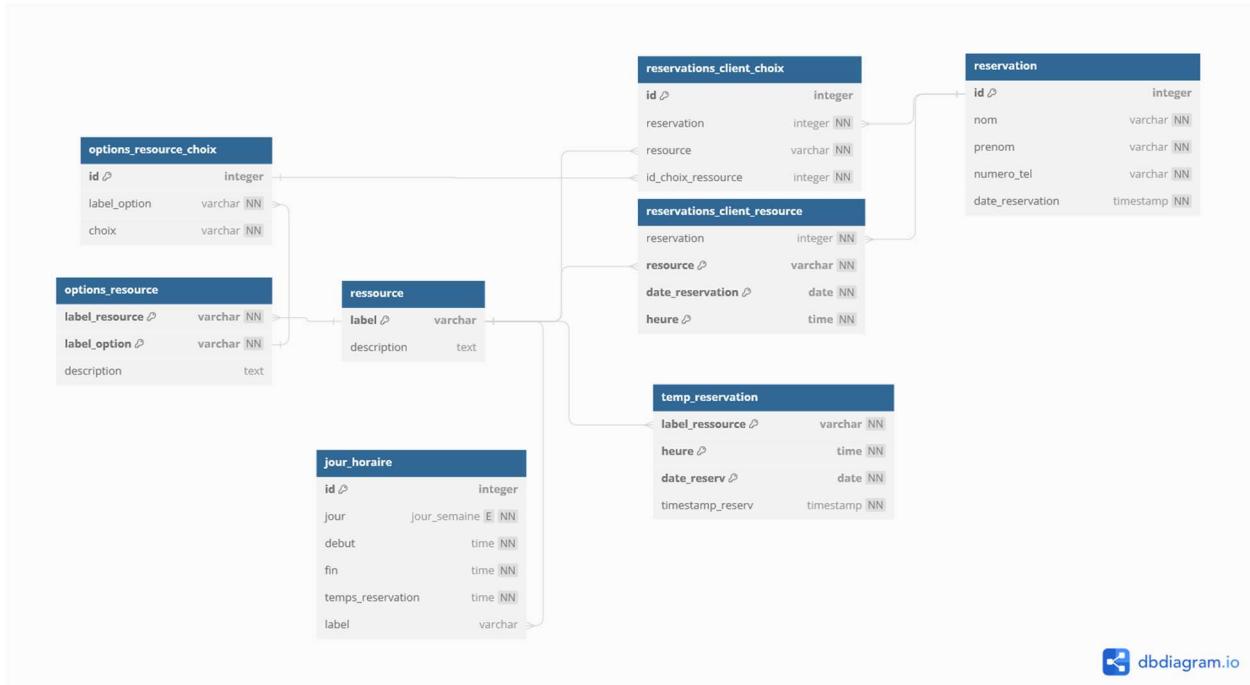


Illustration 26 : Diagramme Entité-Association de la Base de données

Source : Rodrigues dos Santos Fabio

Ci-dessus est une représentation en diagramme d'Entité-Association de la base de données utilisée dans le projet. Dans un premier lieu, la table qui sera régulièrement sélectionnée est celle des ressources. Celle-ci est liée à toutes les tables concernant les informations nécessaires à une réservation telles que :

- Jour_horaire : Table contenant les horaires par jour de semaine pour une ressource donnée.
- Options_ressource : Table contenant chaque option liée à une ressource.
- Options_ressources_choix : Table contenant cette fois-ci les choix liés à une option.
- Temp_reservation : Table contenant les réservations enregistrées temporairement afin d'éviter de multiples réservations sur un même créneau pour une ressource.

Ensuite, lorsque les informations nécessaires à une réservation sont récupérées, une réservation temporaire est en premier lieu créée dans la base de données. Après, lorsque le client fournit son nom et autres informations supplémentaires, une réservation finale est insérée. À la suite de la création de la réservation finale, les tables suivantes sont remplies :

- Reservation : Table possédant les informations du client pour une réservation.
- Reservations_client_ressource : Table enregistrant l'heure, la date et la ressource réservée par une réservation.
- Reservations_client_choix : Table enregistrant les choix effectués par le client lorsqu'il y en a pour une ressource.

6.6. API

L'API au centre du fonctionnement du Chatbot est un serveur REST possédant une multitude de routes afin de récupérer diverses informations. Ci-dessous se trouve la liste des routes mises à disposition :

Route	Méthode HTML	Description	Retour de route
/add-reservation/	POST	Ajout d'une réservation à la base de données. Les données à fournir à la route sont : nom, prenom, numéro de téléphone, date, heure et liste d'options.	{ 'message' : ..., 'data' : { 'lien_google' : ..., 'reservation' : ..., 'reservation_ressource' : ..., 'reservation_choix' : ..., 'id_fichier_ical' : ... } }
/communicate-rasa	POST	Route permettant de communiquer directement avec le bot Rasa. Les données à fournir sont : un id client et un message.	[{'sender' : <id_client>, 'message' : <message>},]]
/add-temp-reservation	POST	Ajout d'une réservation temporaire lorsqu'un	{}

		<p>client est au milieu d'une réservation afin de ne pas perdre son créneau horaire. Les données à fournir sont : ressource, heure et date réservation</p>	<p>‘message’ : <message>, ‘data’ : {‘reservation’ : <réservation ajoutée>} }</p>
/get-options-choix/{ressource}	GET	<p>Récupération des choix disponibles pour une ressource. Les données à fournir sont : Nom ressource</p>	<p>{‘options’ : <options>} }</p>
/get-reservations-ressources	GET	<p>Récupération de toutes les réservations</p>	<p>[[id_reservation ,ressource ,date_reservation, heure], ...]</p>
/get-reservations-ressources/{ressource}	GET	<p>Récupération de toutes les réservations pour une ressource. Les données à fournir sont : Nom de ressource</p>	<p><Même retour que la route précédente></p>

/get-reservations-ressources-from-date/{ressource}/{date}	GET	Récupération de toutes les réservations pour une ressource et date. Les données à fournir sont : Nom de ressource et date	<Même retour que la route précédente>
/get-jours-semaine/{ressource}/{num_jours}	GET	Récupération des dates disponibles pour une ressource et allant d'aujourd'hui à x jours plus tard. Les données à fournir sont : Nom de ressource et nombres de jours	{ ‘dates’ : <suite de dates>, ‘horaires’ : <objets horaires> }
/get-jours-semaine/{ressource}/{num_jours}/{heure}	GET	Récupération des dates disponibles pour une ressource et heure et allant d'aujourd'hui à x jours plus tard. Les données à fournir sont : Nom de ressource, nombres de jours et heure	<Même retour que la route précédente>

/get-horaires/{ressource}	GET	Récupération des horaires et heures disponibles pour une ressource. Les données à fournir sont : Nom de ressource	{ ‘heures_dispo’ : <liste d’heures disponibles>, ‘horaires’ : <liste d’horaires> }
/get-horaires/{jour}/{ressource}	GET	Récupération des horaires et heures disponibles pour une ressource et date. Les données à fournir sont : Nom de ressource et date	<Même retour que la route précédente>
/get-ressources	GET	Récupération de toutes les ressources disponibles	[<liste de ressources>]
/get-ics-file/{ics_name}	GET	Récupération du fichier ICS en format base 64. Les données à fournir sont : Nom du fichier ics (ID)	{ ‘file_base64’ : <fichier base64> }

Tableau 5 : Tableau des routes disponibles sur l'API

Source : Rodrigues dos Santos Fabio

6.7. INTERFACES DU CHATBOT

Pour pouvoir communiquer plus aisément, deux interfaces ont été implémentées.

Une application Web et un Chatbot Telegram qui est géré grâce à un package Telegram en Python.

a) TELEGRAM

La première interface, le Bot Telegram, est disponible pour tout utilisateur de Telegram à l'adresse de « https://t.me/WiREV_bot ».

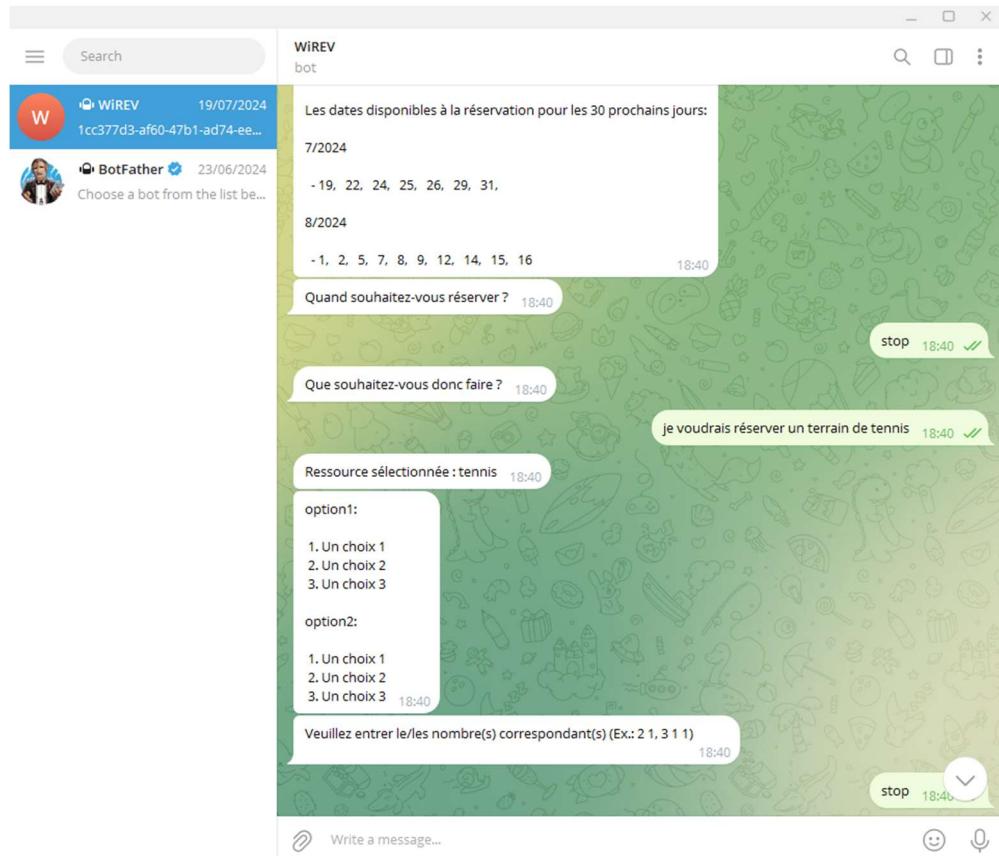


Illustration 27 : Capture d'écran du bot Telegram

Source : Rodrigues dos Santos Fabio

Une fois le bot ajouté dans Telegram, il est possible d'interagir directement avec ce dernier en lui envoyant des messages.

b) APPLICATION WEB

La seconde interface, une application web, hébergée localement peut être aussi accédée de toute part si le réseau local est correctement configuré pour accepter des connexion externes.

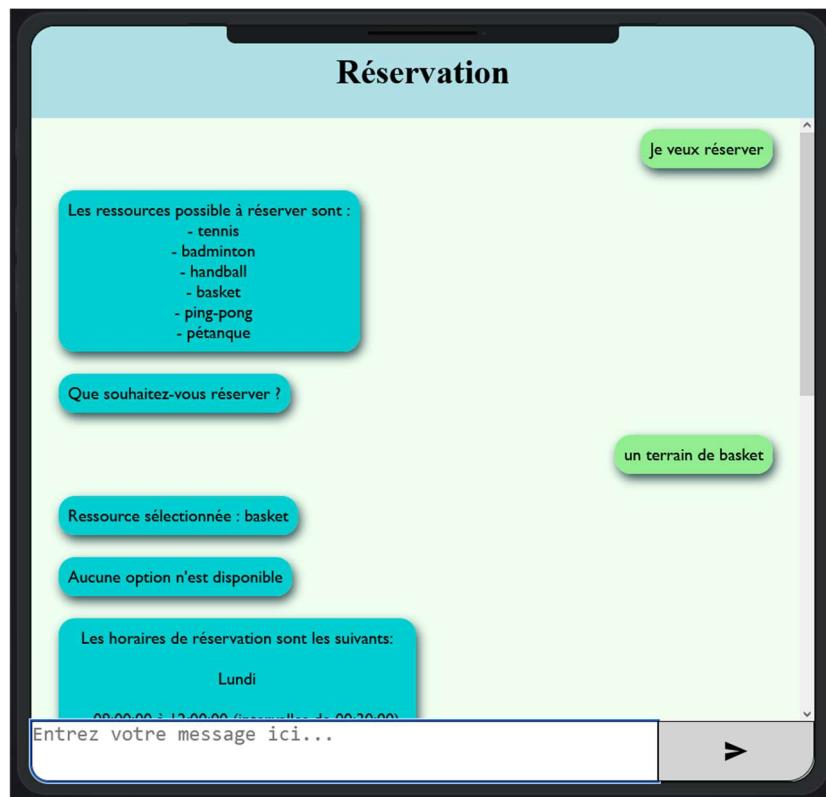


Illustration 28 : Capture d'écran d'une conversation avec le bot web.

Source : Rodrigues dos Santos Fabio

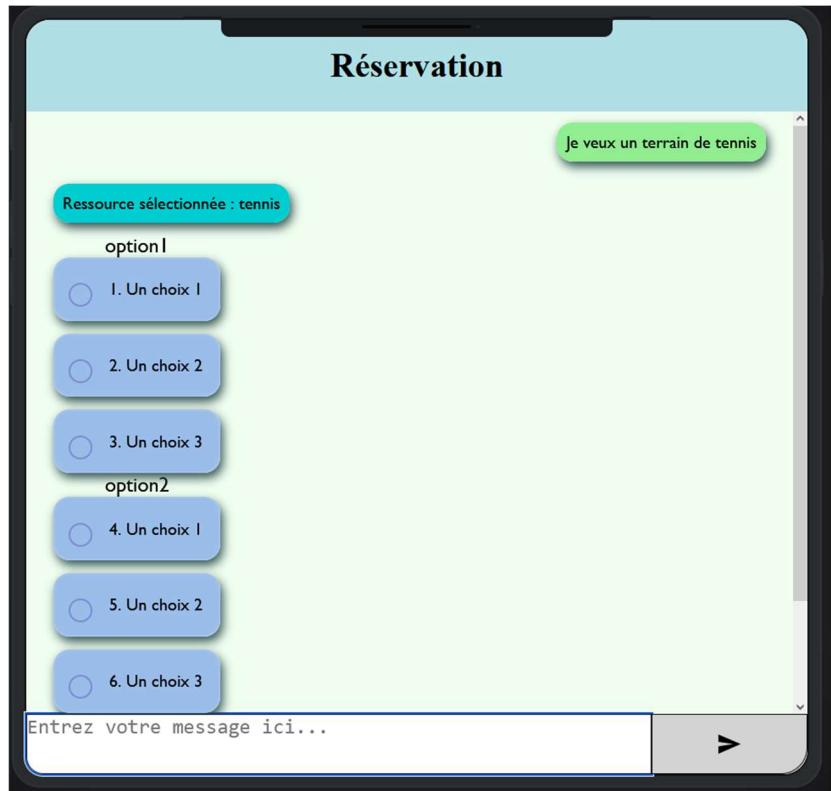


Illustration 29 : Capture d'écran de sélection de choix lors d'une réservation

Source : Rodrigues dos Santos Fabio

Chapitre 7 : RÉALISATION DU PROJET

Dans cette dernière section, nous abordons enfin comment ont été réalisé les divers composants formant le Chatbot.

7.1. IMPLÉMENTATION CHATBOT

Comme décrit précédemment, le bot au centre de ce projet est réalisé à l'aide du Framework de Bot Rasa. Framework qui on le rappelle sert à mettre en place des assistants conversationnels grâce à du NLU.

De la même manière que pour le prototype réalisé, le bot présent dans le projet actuel contient un composant Rasa Core et Action server et aussi un composant crucial étant un conteneur Duckling. Comme vu au chapitre 3.3 « Chatbots par NLU », les éléments principaux sont :

- Les stories : Suite d'actions, d'intents, forms, etc qui décrivent un chemin que peut prendre le chatbot lors d'une discussion.
- Les forms : Fonctions dans le bot qui vont boucler en posant un/des questions à l'utilisateur jusqu'à l'obtention des informations nécessaires.
- Les rules : Fonctionnement similaire à celui des stories, à la différence qu'elles peuvent être appelées dans d'autres stories pour exécuter de manière impérative une/des actions. Permet aussi d'assurer qu'une action aura lieu comme la fermeture d'un form complété.
- L'action server : Un ensemble d'actions implémentées en Python qui manipulent et interagissent avec les divers variables à disposition comme les slots et intents d'une conversation courante.

a)

ARCHITECTURE DE FICHIERS RASA

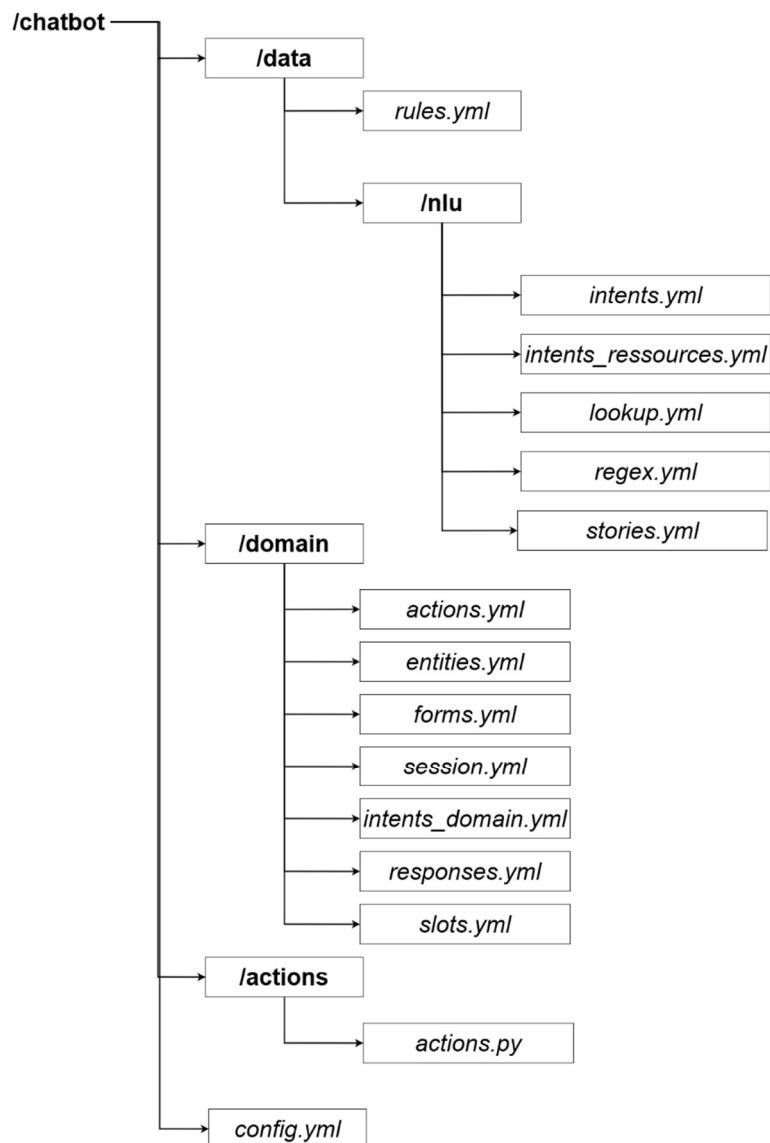


Illustration 30 : Arbre de fichiers principaux du bot Rasa

Source : Rodrigues dos Santos Fabio

Concernant l'architecture du Bot Rasa, elle est divisée en trois dossiers principaux :

- **/data** : Dossier comportant l'ensemble des “jeux de données” du bot. Présents à l'intérieur sont le fichier contenant les règles et un dossier NLU avec l'ensemble des données liées à la logique conversationnelle.
- **/domain** : Dossier comportant diverses déclarations d'éléments comme les **actions**, **entités**, **forms**, **réponses**, **slots** et **intents**. Un fichier définissant quelques configurations liées à chaque session ouvertes par un utilisateur.

b) STORIES

Le cœur de la logique conversationnelle est réparti en quelques stories dont une principale.

- salutation_bot : Le bot salue l'utilisateur et propose à l'utilisateur quelques options sur ce qu'il souhaite faire.
- ask_horaire : Permet de directement demander l'horaire d'une ressource sans avoir besoin d'initier tout le système de réservation.
- stop_convo : Permet de mettre fin à une conversation subitement.
- goodbye : Le bot répond au revoir et mets fin à la conversation courante.
- selection_ressource : La story principale décrivant les étapes et interaction nécessaires pour faire une réservation.

Regardons de plus près un extrait de **selection_ressource** afin de comprendre comment une réservation peut être effectuée par un utilisateur.

```
stories.yml
- story: selection_ressource
  steps:
    - intent: ressource_gather
    - action: predefine_ressource
    - action: get_ressource_form
    - active_loop: get_ressource_form
    - slot_was_set:
        - ressource: Basket
    - slot_was_set:
        - requested_slot: null
    - active_loop: null
    - action: utter_ressource
    - slot_was_set:
        - accept_deny: None
    - action: get_options_reserv_form
    . . .
```

Ci-dessus, nous avons un extrait de la story précédemment mentionnée. Observons les premières lignes de plus près :

```
stories.yml
. . .
- intent: ressource_gather
- action: predefine_ressource
. . .
```

Dans la première ligne est précisé un des intents qui, s'il est détecté dans l'entrée utilisateur, indiquera qu'une réservation veut être effectuée.

```
intents_ressource.yml
nlu:
  - intent: ressource_gather
    examples: |
      - Je veux réserver un terrain de [tennis] (ressource)
      - J'aimerai réserver un terrain de [badminton] (ressource)
      - Je souhaite réserver un terrain de [handball] (ressource)
      - Je veux réserver une table de [basket] (ressource)
      . . .
```

Lorsqu'une phrase similaire à ce qui est spécifié dans l'intent **ressource_gather** (dont un extrait est donné ci-dessus) est détecté par Rasa Core, l'action **predefine_ressource** est appelée dans le serveur d'action Rasa.

```
actions.py
Action_predefine_ressource():
    pre_ressource = tracker.get_slot("ressource_prereserv")
    .
    .
    .
    ressource = None
    final_return = []
    if pre_ressource is not None:
        pre_ressource_processed = str(pre_ressource).lower()
        if pre_ressource_processed not in get_ressource_list():
            dispatcher.utter_message(f"La ressource
{pre_ressource_processed} n'est pas disponible")
            final_return.append(SlotSet("ressource_prereserv",None))
        else:
            ressource = pre_ressource_processed
            final_return.append(SlotSet("ressource",pre_ressource_pr
ocessed))
    .
    .
```

L'action **predefine_ressource** permet de valider une multitude d'options en une fois pour une réservation, cependant dans l'extrait présenté nous ne prêtions attention qu'à la réservation de ressources. Tout d'abord, nous récupérons le contenu du slot **ressource_prereserv**. Après s'être assuré que le slot n'est pas vide, on confirme que la ressource est bien présente dans la liste des ressources avec l'appel de la fonction **get_ressource_list()**. Cette dernière fait un appel à l'API pour récupérer toutes les ressources mises à disposition dans la base de données. Si la ressource est bien présente, elle est directement affectée au slot **ressource** ce qui permet de directement

présélectionner une ressource ; dans le cas contraire aucune ressource ne sera présélectionnée.

```
intents_ressource.yml
. . .
- action: get_ressource_form
- active_loop: get_ressource_form
- slot_was_set:
-   ressource: Basket
-   slot_was_set:
-     - requested_slot: null
- active_loop: null
. . .
```

Ce qui suit est l'initialisation d'un **Form** qui on le rappelle permet de créer une boucle demandant une question à l'utilisateur et ce jusqu'à ce que les informations souhaitées soient fournies. Dans le cas ci présent, l'utilisateur est questionné sur la ressource qu'il souhaite réserver. Si comme expliqué précédemment l'utilisateur a déjà préréservé une ressource, alors le form prendra fin et continuera l'exécution de la conversation.

```
intents_ressource.yml
. . .
- action: utter_ressource
- slot_was_set:
-   accept_deny: None
- action: get_options_reserv_form
. . .
```

Ce qui suit est l'énonciation de la ressource choisie et l'initialisation du prochain form.

Une multitude d'actions vont suivre jusqu'à arriver à une réservation complète.

c) FORMS

La réalisation du bot a nécessité l'ajout de Forms afin d'assurer que l'utilisateur fournit les informations nécessaires à la réservation. Ci-dessous se trouve quelques-uns des forms.

```
forms.yml
forms:
    get_ressource_form:
        required_slots:
            - ressource
        ignored_intents:
            - inform_date
            - inform_heure
            - ressource_gather

    get_date_heure_form:
        required_slots:
            - date
            - heure
        ignored_intents:
            - ressource_gather
        . . .
```

d) RULES

Les règles possèdent un rôle similaire aux stories à la différence qu'elles peuvent être appelées n'importe quand même pendant des stories. Elles servent ici principalement à assurer le début et la fin de la boucle de chaque form et permettre l'exécution de certaines actions selon des **intents** ou actions précises.

```
rules.yml
- rule: Activer messages de debug
  steps:
    - intent: debug
    - action: action_activate_debug_mode

- rule: recommencer
  steps:
    - intent: stop
    - action: restart_convo

- rule: activate Form ressource
  steps:
    - intent: ressource_gather
    - action: predefine_ressource
    - action: get_ressource_form
    - active_loop: get_ressource_form
    . . .
```

Les deux premières règles sont ces cas où l'on veut pouvoir activer une suite d'action peu importe où l'on se trouve telle que l'activation de messages de débogage ou de pouvoir recommencer la conversation. La règle qui suit est une des nombreuses règles qui sont presque requises pour faire fonctionner les forms, sans quoi il se pourrait qu'ils ne s'activent pas ou que des comportements inattendus se produisent.

e) ACTION SERVER

L'action server est composé d'une série de classes qui possèdent un nom d'action et des fonctions associées. L'action serveur du projet possède principalement deux types d'actions :

- FormValidationAction : Une classe appelée lors de la validation de forms
- Action : Une simple fonction

Ces derniers se présentent sous cette forme :

```
actions.py
class AskForRessourceAction(Action):
    def name(self) -> Text:
        return "action_ask_ressource"

    def run(
        self, dispatcher: CollectingDispatcher, tracker: Tracker,
domain: Dict
    ) -> List[EventType]:
        . . .
```

```
actions.py
class ValidateHeuresForm(FormValidationAction):
    def name(self)->Text:
        return "validate_get_date_heure_form"
    def validate_date(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:
        . . .
```

Dans le cas d'actions de type **FormValidationAction**, le nom de la classe est important car il doit toujours être de la forme « validate_<nom du form> » car sinon les actions

liées à ce dernier ne seront pas correctement appelées lors de la validation du form en question. Ensuite, chaque slots validé possède une fonction de la forme « validate_<slot> ». Comme pour la classe, si le nom de fonction ne respecte pas ce format, elle ne sera pas appelée.

L'exemple d'action choisi ici n'est pas aléatoire car c'est pour démontrer un autre cas particulier toujours lié aux forms. Si l'on définit une classe avec le nom « action_ask_<nom slot> », lorsque l'utilisateur entre dans un form possédant ce slot à remplir, le code présent dans l'action sera exécuté et permettant ainsi de poser une question.

Toutes les actions possèdent trois paramètres donnant accès à certaines données importantes :

- Dispatcher : l'interface permettant de diffuser des messages à l'utilisateur avec la fonction **dispatcher.utter_message()** par exemple.
- Tracker : Objet contenant une multitude d'informations liés au message et état de divers données stockées lors de la conversation courante tels que :
 - o L'état courant du bot (Dans un form, dans un événement, affectation de slot, etc.)
 - o Le dernier message reçu
 - o Les intents parsés lors du dernier message de l'utilisateur et plus tôt encore
 - o Les slots de la conversation courante
- Domain : Un dictionnaire contenant les diverses information spécifiées lors de la configuration du bot comme les slots, intents, réponses, forms, règles et actions présents.

7.2. IMPLÉMENTATION BASE DE DONNÉES

Concernant la base de données, il n'y a pas une grande quantité d'étapes à réaliser pour mettre en place la base de données.

En premier lieu, la base de données spécifiée lors du chapitre conception est réalisée dans le format d'un fichier **.sql** qui est copié dans le conteneur associé et sert de fichier initialisant la base de données. Ensuite, comme la base de données est entièrement dockerisée ; image docker de Postgresql comprise, il suffit de démarrer le conteneur avec Docker Compose pour que la base de données soit instanciée. Cependant, elle ne possède aucune données.

Donc, ce qu'il manque à faire est d'ajouter les ressources requises dans le dossier **/db/data** sous la forme de fichiers **.csv**. Une fois ces derniers ajoutés (tout en suivant les templates fournis par défaut), il ne suffit que d'exécuter une fois le script présent dans le chemin **/db/scripts/data_insert.py**.

Le script possède une fonction principale prenant comme arguments **Nom de fichier csv** et **nom de table**. Les fichiers fournis doivent correspondre de manière plus ou moins exact les colonnes présentes, dans le même ordre et les mêmes noms de colonnes que dans la base de données.

```
data_insert.py
def process_csv(file:str,table_name:str,*args:str):
    pass

def update_sql_script():
    pass

def execute_script():
    pass

if __name__ == "__main__":
    process_csv("ressource.csv","ressource")
    process_csv("horaire.csv","jour_horaire")
    process_csv("options.csv","options_resource")
    process_csv("options_choix.csv","options_resource_choix")

    update_sql_script()
    execute_script()
```

Ci-dessus se trouve les signatures de fonctions principales ainsi que le code exécuté. Les fichiers fournis sont :

- Les ressources
- Les horaires de ressources
- Les options de ressource
- Les choix de chaque option

Une fois que chaque fichier est traité et ajouté dans la mémoire temporaire du script, un script SQL est créé à partir de ces derniers et finalement exécuté.

```
data_insert.py
COPY {table_name}({col_string})
FROM '/data/{file}'
WITH (FORMAT csv, HEADER);
```

Le code SQL ci-dessus est exécuté pour chaque ressource et la raison pour laquelle le format des fichiers CSV est si strict. Ce script permet de faciliter l'insertion de données en une commande qui à partir d'un fichier CSV donné va remplir la base de données.

7.3. IMPLÉMENTATION API

Pour faire suite au détail du Chatbot, penchons-nous sur l'API qui sert d'entremetteur entre le Chatbot et la base de données. API réalisée en Python et grâce à FastAPI pour le serveur REST et SQLAlchemy pour tout ce qui est lié à des interactions Serveur-Base de données.

a) MISE EN PLACE API

L'implémentation d'un serveur REST avec FastAPI s'est fait de la manière suivante :

```
server.py
origins = [
    "http://0.0.0.0",
    "http://0.0.0.0:5500",
    "http://localhost:5500",
    "http://localhost:3000"
    "http://localhost",
    "*",
]
. .
app = FastAPI(lifespan=start_remove_reserv_task)
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
if __name__ == "__main__":
    uvicorn.run("server:app", host="0.0.0.0" , port=5500,
log_level="info", reload =
True,workers=4,reload_dirs=["/app"],reload_excludes=["/app/.venv"])
```

Le serveur est simplement créé en appelant le constructeur d'objets de FastAPI avec **FastAPI()**. Ensuite est ajouté les divers middleware nécessaires au fonctionnement du serveur. Middleware qui on le rappelle est l'ensemble des actions et fonctions qui seront déclenchée lors de chaque appel ou renvoi de requête à l'API. Ici le middleware important est celui du CORS et des origines autorisées car sans ça, l'API ne pourrait pas dialoguer aisément avec la base de données qui ici se trouve en général sur la même machine donc la même IP.

Le CORS ou Cross-Origin Resource Sharing est un mécanisme permettant d'autoriser une application web à accéder d'autres applications ou ressources se trouvant sur un domaine différent. Ici, le domaine en question est le backend que le frontend essaie d'accéder et sans le CORS aucune requête ne peut passer. D'où la nécessité d'ajouter le middleware précédemment mentionné.

Il ne reste finalement qu'à démarrer le serveur avec l'appel de fonction **.run(. . .)** et les divers paramètres associés tels que l'IP, le port, l'activation du mode hot reload si

nécessaire et conséquemment fournir les chemins à observer pour le hot reload. Hot Reload qui permet de rafraîchir le code tournant sur le serveur courant et facilitant le développement de ce dernier.

b) ROUTES

L'implémentation des routes avec FastAPI se font de la manière suivante :

```
server.py
@app.post("/communicate-rasa")
async def talk_to_rasa(data:Communicate_Rasa):
    try:
        . . .
        return
    JSONResponse(content=response.json(),status_code=status.HTTP_200_OK)

    except Exception as e:
        raise
    HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,detail=
f"Une erreur s'est produite lors de la communication avec Rasa:
{str(e)}")
```

L'extrait ci-dessus montre un morceau de la route servant à communiquer avec le chatbot Rasa. La première chose à faire lors de l'ajout de routes est la spécification de la méthode HTML à utiliser ainsi que le chemin de la route elle-même. Ensuite, les paramètres à fournir sont spécifiés dans la signature de la fonction associée.

Une bonne pratique est d'encapsuler les sections de code pouvant générer des erreurs et dans le cas d'un code en python, mettre des **try...except**. Ces derniers permettent d'exécuter le code spécifié si une erreur survient; ici étant le retour d'une erreur 500 avec un message spécifique décrivant le problème. Sinon, une réponse en format JSON est retournée ainsi qu'un code 200.

c) CONNEXION À LA BASE DE DONNÉES

Contrairement à des appels d'API basiques, dans notre cas il n'est pas simplement possible de faire une requête HTTP à la base de données mais une connexion au travers Postgres est nécessaire. Pour se faire, à l'aide un tutoriel¹⁹ décrivant les étapes et éléments nécessaires à l'établissement d'une connexion à la base de données

```
server.py
config = load_config()

url = URL.create(
    drivername="postgresql",
    username=config["user"],
    host=config["host"],
    database=config["database"],
    password=config["password"],
    port=config["port"]
)

engine = create_engine(url)
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=engine)
```

La fonction `load_config()` récupère premièrement le contenu du fichier `.ini` présent dans le dossier courant et charger son contenu dans la variable `config`. À partir de cette dernière, un moteur connecté à la base de données spécifiée en paramètres. Finalement, un créateur de sessions est initialisé à l'aide du moteur fourni afin de pouvoir créer dynamiquement de nouvelles sessions pour la même base de données afin de faciliter l'accès parallèle et rapide à cette dernière.

¹⁹ POSTGRESQL PYTHON: CONNECTING TO POSTGRESQL SERVER, POSTGRESQL TUTORIAL [EN LIGNE]. DISPONIBLE À L'ADRESSE : [HTTPS://WWW.POSTGRESQLTUTORIAL.COM/POSTGRESQL-PYTHON/CONNECT/](https://www.postgresqltutorial.com/postgresql-python/connect/) [CONSULTÉ LE 31 JUILLET 2024].

d) INTERACTION AVEC LA BASE DE DONNÉES

Maintenant que nous avons une connexion directe à la base de données, il nous faut un moyen d'aisément interagir avec les données. Pour cela, nous allons faire usage du principe d'ORM (Object-relational mapping ou Affectation relationnelle d'objets). Au lieu de devoir écrire des requêtes SQL en dur et de les envoyer à la base de données, l'ORM permet de créer des objets auxquels on peut affecter des données et/ou directement en récupérer depuis la base de données.

```
server.py
Base = declarative_base()
class Ressource(Base):
    __tablename__ = 'ressource'

    label = Column(VARCHAR(), primary_key=True)
    description = Column(Text(), nullable=True)

class Jour_Horaire(Base):
    __tablename__ = 'jour_horaire'

    id = Column(Integer(), primary_key=True, autoincrement="auto")
    jour = Column(Enum(Jours_Semaine), nullable=False)
    debut = Column(Time(), nullable=False)
    fin = Column(Time(), nullable=False)
    temps_reservation = Column(Time(), nullable=False)
    label =
    Column(VARCHAR(), ForeignKey('ressource.label'), nullable=True)
    . . .
```

Comme l'indique son nom, il est nécessaire de déclarer des objets qui sont chacun attachés à une table dans la base de données. Ci-dessus se trouvent deux objets afin de montrer le format de déclaration des objets.

```
server.py
async def add_temp_reserv(data:Temp_Reservation_API):
    new_temp_res = Temp_Reservation(. . .)

    try:
        output_pre_res = {}
        with Session.begin() as session:
            session.add(new_temp_res)
            session.flush()
            session.refresh(new_temp_res)
    . . .
```

Après avoir déclaré les divers tables sous forme d'objets, l'ajout d'une nouvelle entrée dans la base de données se fait en quelques lignes. Ci-dessus se trouve un extrait de l'insertion

d'une nouvelle réservation temporaire. Nous créons un objet réservation temporaire et les données liées à cette dernière remplies puis après avoir initialisé une nouvelle session :

- **.add()** ajoutera l'entrée dans la queue de requêtes à envoyer à la base de données.
- **.flush()** enverra directement le contenu de la queue à la BDD.
- **.refresh()** rafraîchira le contenu de l'objet fournit afin de, par exemple, faire concorder l'ID de l'objet nouvellement insérer à celui dans le code.

```
server.py
@app.get("/get-reservations-ressources-from-date/{ressource}/{date}")
async def
get_reservations_ressources_from_date(ressource:str,date:datetime.date):
:
    try:
        with Session.begin() as session:
            query =
session.query(Reservations_Client_Resource).where(Reservations_Client_R
esource.resource ==
ressource).where(Reservations_Client_Resource.date_reservation >=
date).distinct().all()
    . . .
```

La démonstration ci-dessus présente comment est réalisé la route qui sert à récupérer les réservations à une date donnée. En observant en détail la suite d'appel de fonctions, elle est très similaire à une requête SQL mais sans nécessiter de la rédiger en dur.

```
SELECT * FROM reservations_client_resource WHERE resource ==
<ressource> AND date_reservation >= <date> DISTINCT ;
```

Équivalent en SQL de la requête en format ORM.

e) CALENDRIER GOOGLE

Pour qu'il soit plus facile de consulter les réservations effectuées, on met en place un calendrier Google au préalable afin d'y stocker les réservations de tous les clients.

Afin de mettre en place le composant de calendriers google, il est important d'exécuter le script **google_cal.py** qui une fois lancé et que localement aucun token n'est trouvé, un

lien sera fourni afin de pouvoir se connecter au service google et permettre au projet de modifier le calendrier associé au compte. Cependant pour pouvoir faire cela il est important de suivre le tutoriel fournit par google²⁰ car il détaille les étapes à suivre pour créer un projet pour que l'API google employée dans le code du Chatbot puisse fonctionner. Une fois le projet créé, il ne manque que d'ajouter le fichier **credentials.json** dans le dossier de l'API et d'exécuter une fois le script précédemment mentionné.

f) FICHIER ICS

Un autre élément renvoyé à l'utilisateur en plus d'un événement google calendar est un fichier au format Ical. Les fichiers en **.ics** suivent un standard couramment utilisé dans la majorité des appareils électroniques de nos jours. Ce format a pour intérêt de stocker et échanger avec aisance des calendriers, événements et autres informations diverses.

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:WiREV calendar
BEGIN:VEVENT
SUMMARY:Réservation de basket
DTSTART;TZID=Europe/Paris:20240719T090000
DTEND;TZID=Europe/Paris:20240719T093000
DESCRIPTION:Une réservation de basket\nParms Narad\nNuméro: 0974462112
END:VEVENT
END:VCALENDAR
```

Contenu d'un fichier ICS généré lors d'une réservation avec le Chatbot.

20 PYTHON QUICKSTART | GOOGLE CALENDAR, GOOGLE FOR DEVELOPERS [EN LIGNE]. DISPONIBLE À L'ADRESSE : [HTTPS://DEVELOPERS.GOOGLE.COM/CALENDAR/API/QUICKSTART/PYTHON](https://developers.google.com/calendar/api/quickstart/python) [CONSULTÉ LE 1 AOÛT 2024].

Le format ICal possède une grande variété d'informations supplémentaires pouvant être fournies mais ici elles n'indiquent que l'essentiel : Le titre de l'événement, la date/heure de début et fin et une description.

```
google_cal.py
def
create_ical_event(title:str,attendee_phone:str,attendee_name:str,attend
ee_surname:str,description:str,date_start:datetime.datetime,date_end:da
tetime.datetime)->str:
    cal = Calendar()
    cal.add('prodid','WiREV calendar')
    cal.add('version','2.0')
    event = Event()
    event.add('summary',title)
    event.add('description',f"{description}\n{attendee_surname}
{attendee_name}\nNuméro: {attendee_phone}")
    event.add('dtstart',date_start.astimezone(ZoneInfo('Europe/Paris')))
    event.add('dtend',date_end.astimezone(ZoneInfo('Europe/Paris')))
    cal.add_component(event)
    .
    .
    .
    cal.to_ical()
    .
    .
```

Pour faciliter l'implémentation de la création de fichiers .ics, un package python “**icalendar**” est employé comme démontré ci-dessus. Un objet calendrier est initiallement créé avec **Calendar()** puis un composant **event** est instancié et les données nécessaires fournies à ce dernier. Finalement, il est ajouté au calendrier précédemment créé et la fonction **.to_ical()** permet de générer les données du fichier ICal qui est créé à la fin de l'exécution de la fonction.

g) OPÉRATIONS ASYNCHRONE

Un dernier détail technique que comporte l'API est la présence de tâches exécutées de manière asynchrone et en tâche de fond. Pour y parvenir, nous employons le package Async.io ; Async.io étant un package python qui permet d'ajouter une multitude de fonctions pour faciliter l'implémentation d'asynchronisme dans une application.

Nous utilisons Async.io ici dans le but de retirer constamment les réservations temporaires de la BDD.

```
server.py
async def remove_old_reservations(interval: int):
    while True:
        await delete_reservation_query()
        await asyncio.sleep(interval)

@asynccontextmanager
async def start_remove_reserv_task(app:FastAPI):
    print("Starting execution of removal of temp reservations")
    task =
asyncio.create_task(remove_old_reservations(interval_exec_remove))
    yield
    print("Stopping execution of removal of temp reservations")
    task.cancel()

app = FastAPI(lifespan=start_remove_reserv_task)
```

Pour se faire, une fonction exécutée au démarrage du serveur est créée. Dans cette dernière, une tâche Async.io est créée et dans laquelle le code lié exécute la fonction de suppression périodique des réservations temporaires selon une intervalle spécifiée.

7.4. IMPLÉMENTATION INTERFACES

Pour finir, un des derniers éléments à couvrir est l'implémentation des diverses interfaces.

a) CHATBOT WEB - FRONTEND

L'application Web du Chatbot étant réalisé grâce à Vite et React fait que c'est une application dite "Single Page". Comme expliqué plus tôt, le principe de ce type d'applications est que l'utilisateur n'a pas besoin de changer de page ou de devoir la rafraîchir complètement.

Pour y arriver, la page comporte une multitude de composants au lieu de simple HTML pur.

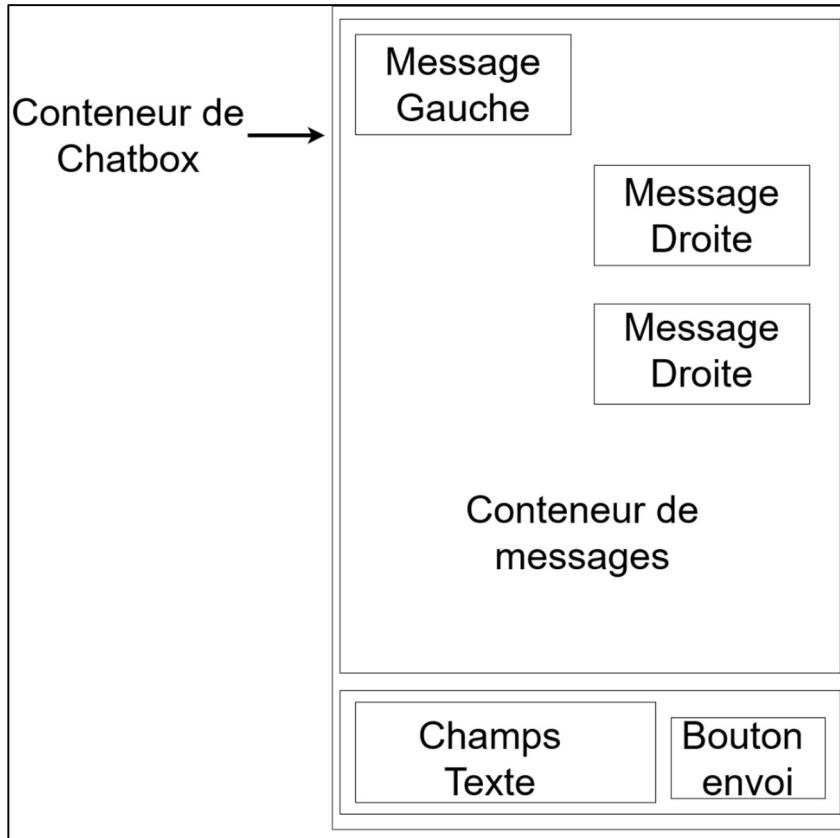


Illustration 31 : Maquette des composants du Frontend

Source : Rodrigues dos Santos Fabio

La maquette ci-joint est une représentation simpliste de l'apparence de la Chatbox. Celle-ci comporte de multiples composants s'imbriquant les uns les autres et ayant chacun leur fonctionnement propre à eux.

chatbox.tsx

```
const Chatbox_text:FC<ChatboxText> = ({message,isBot,key_value}) => {
  return isBot ? (
    <div key={key_value} className="text-left"
    dangerouslySetInnerHTML={{ __html:message }} />
  ): (
    <div key={key_value} className="text-right"
    dangerouslySetInnerHTML={{ __html:message }} />
  );
}
```

Chaque composant possède des paramètres nommés « props » qui sont donnés à ces derniers et qui dans le cas ci-dessus, permettent de déterminer si un message doit être affiché à gauche ou à droite.

chatbox.tsx

```
interface ChatboxText extends ChatboxMessage{
    message:string,
    isBot:boolean,
    type:'text'
}
```

Interface définissant les props du composant ChatboxText

Du fait du fonctionnement intrinsèque des applications single page, il est nécessaire d'utiliser des « states ». Ces variables retiennent les informations leurs ayant été fournies même après un rafraîchissement des composants.

chatbox.tsx

```
const Chatbox_container: FC<ChatboxContainer> = ({} ) => {

    // Utilisation de contexte afin d'avoir un socket global unique
    const [socket] = useState( useContext(SocketContext))

    const [messages, setMessages] = useState<ChatboxMessage []>([])

    const [radioIndex, setRadioIndex] = useState(0)

    const [user_id] = useState(uuidv4())
    . .
.
```

Déclaration des states liés au conteneur de messages

Similairement à ce qui est présent dans le backend, des sockets sont employés afin de communiquer directement avec le bot.

chatbox.tsx

```
useEffect(()=>{
    socket.on('chat', (bot_messages:string[])=> {
        const messagesBotTreated : ChatboxMessage []=
        bot_messages.map((message, index)=>{
            return process_message(message, index);
        })
        setMessages(prevMessages =>
        [...prevMessages, ...messagesBotTreated])
    })
    return () => {
        socket.off('chat');
    };
}, [])
```

La fonction **useEffect()** est employée par React pour exécuter du code lorsque les variables fournies dans les « [] » sont mises à jour. Ou bien s'il n'y en a pas, le code est

exécuté une seule fois. Dans ce cas présent, le code ne nécessite que d'être exécuté une seule fois car la fonction `socket.on()` étant un listener restera en mode écoute jusqu'à ce que le socket soit explicitement fermé. Donc, tant que l'application tourne, les messages envoyés par le bot seront réceptionnés à l'événement « Chat », traités et ajoutés au fil de messages à afficher.

Dans le but d'entièrement visualiser les interactions avec le backend par le socket, se trouve juste dessous le code permettant d'envoyer un message au bot.

chatbox.tsx

```
function sendMessageToBot(message:string,uuid:string) {
    socket.emit("chat", message,uuid)
}
```

b) CHATBOT WEB – BACKEND

Le Backend implémenté pour l'interface Web est similaire à l'API du Chatbot dans le sens où les deux sont des API REST mais dans des langages différents. Cette fois-ci en Typescript

```
main.ts
const app = express();
const http = require("http");
const server = http.createServer(app);
const PORT = process.env.PORT;
const io = new Server(server, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"],
    allowedHeaders: []
  },
  allowEIO3: true
});
app.use(cors({origin:["*"]}));
server.listen(PORT, () =>
  console.log(`Server is listening on
${server.address().address}:${PORT}...`)
);
```

Similairement à FastAPI, une instance de serveur Express est créée avec `express()`. Express étant le framework REST, il nécessite une base étant un serveur et c'est donc pour cela qu'un est créé avec `http.createServer()`. Tout comme pour l'API, le

middleware CORS est ajouté afin de permettre la communication avec le Frontend sur la même machine. Finalement, le serveur est mis en mode écoute à l'adresse et port fournis.

Le serveur ne possède à son actif aucune route à proprement parler mais fait usage d'un socket pour ouvrir le canal de communication avec le frontend client.

```
main.ts
io.on("connection", (socket: Socket) => {
  socket.on("disconnect", () => {
    console.log(`a user with id \`${socket.id}\` has disconnected`);
  });
  console.log(`a user with id \`${socket.id}\` connected`);
  socket.on('chat',async (message:string,uuid:string) => {
    const rasa_messages :string[] = await query_rasa(message,uuid);
    socket.emit("chat", rasa_messages);
    console.log(uuid + ": " + message)
  })
});
```

Tout comme le frontend, lorsque le backend reçoit un message par un événement « chat », il va transférer ce dernier à Rasa avec la fonction `query_rasa()` et renvoyer par un `socket.emit()` un événement « chat » avec la réponse du bot.

c) TELEGRAM

La mise en place du service Telegram s'est fait par l'utilisation de l'API fournie par Telegram et disponible dans une multitude de langages dont Python.

Comme Telegram est un service de messagerie à part entier, les seules parties nécessaires à implémenter sont comment traiter les messages reçus. Pour se faire, nous implementons la fonction suivante:

```
telegram_bot/main.py
async def handle_message(update:Update, context:ContextTypes.DEFAULT_TYPE):
    text = update.message.text
    id_user = update.message.chat.id
    # headers = {"Content-Type": "application/json"}
    data = {
        "message":text,
        "sender": str(id_user)
    }
    response = requests.post("http://api:5500/communicate-rasa",json={
        "message": str(text),
        "sender": str(id_user)
    })
    .
    .
    .
    await update.message.reply_text(resp)
```

La fonction de traitement de message récupère le message ainsi que l'ID de l'utilisateur courant et s'en sert pour former la requête à envoyer au bot Rasa. Une fois la réponse reçue et traitée, elle est envoyée à l'utilisateur avec la fonction **update.message.reply_text()**.

L'ajout de la fonction dans le handler des messages du Bot Telegram, c'est-à-dire définir quelle fonction sera exécutée à chaque message reçu, est implémenté de la sorte :

```
telegram_bot/main.py
if __name__ == "__main__":
    dotenv.load_dotenv(dotenv_path=".env-api")
    TOKEN = os.getenv('TELEGRAM_API_KEY')
    dotenv.load_dotenv(dotenv_path=".env-api")
    print(os.getenv('BOT_USERNAME'))
    app =
        Application.builder().token(TOKEN).concurrent_updates(True).build()
        app.add_handler(CommandHandler('start', start_command))
        app.add_handler(MessageHandler(filters.TEXT, handle_message))
        app.add_error_handler(error)
        print('Polling...')
        app.run_polling(poll_interval=1)
```

Nous récupérons le token de l'API Telegram dans le fichier **.env-api** et l'utilisons pour créer une instance de serveur d'API telegram. Ensuite, nous ajoutons la fonction de traitement de message, de message d'accueil et le code gérant l'affichage des erreurs au serveur avec **.add_handler()**. La fonction mentionnée précédemment permet de spécifier au serveur quel code exécuter lorsqu'un message doit être traité ou qu'un message de bienvenue doit être affiché. La dernière étape est celle du démarrage du serveur en mode écoute, prêt à recevoir et traiter des messages.

Chapitre 8 : RÉSULTATS

L’application résultante de l’ensemble du travail réalisé au cours du projet de semestre puis du projet de bachelor atteint bien le niveau d’exigence que nous nous étions imposés. Le Chatbot en l’état est capable d’engager dans une discussion pour aboutir à une réservation de manière plutôt fluide et sans trop d’accros. Une fonctionnalité du bot dont nous sommes particulièrement satisfaits est l’ajout des divers scripts de remplissage d’intents et de base de données. Ces derniers offrent donc la possibilité d’aisément modifier le type de réservation en fournissant seulement quelques fichiers CSV dans le format correspondant.

Jetons un œil au fonctionnement du bot depuis les deux interfaces mises en place. Les scénarios qui vont suivre se passent dans le cadre d’une réservation pour un terrain d’airsoft ou de paintball.

8.1. CHATBOT WEB

Pour commencer, deux choix sont proposés à l'utilisateur lorsqu'il ouvre la discussion:

- Entamer une réservation
- Voir les horaires d'une ressource

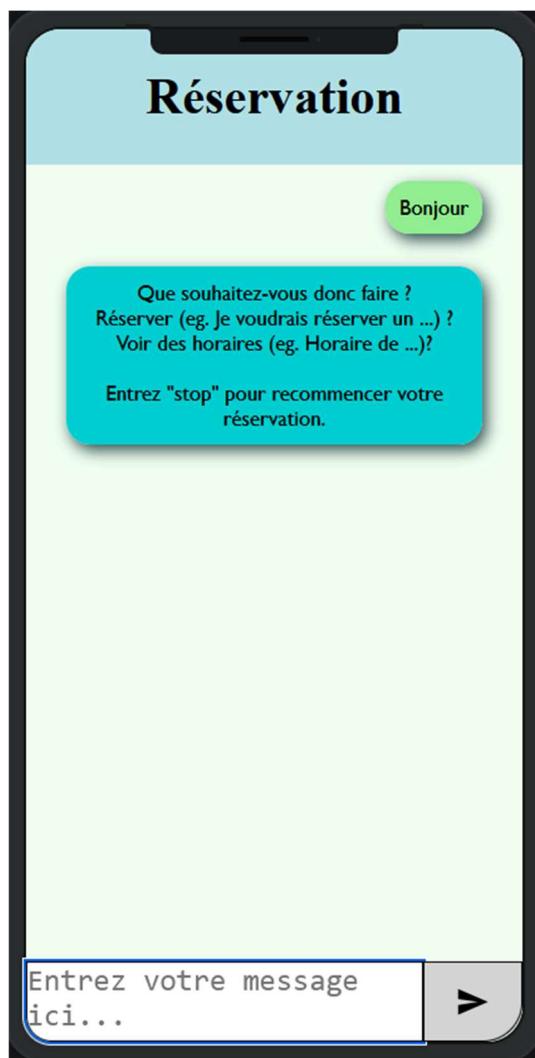


Illustration 32 : Capture d'écran de début de conversation

Source : Rodrigues dos Santos

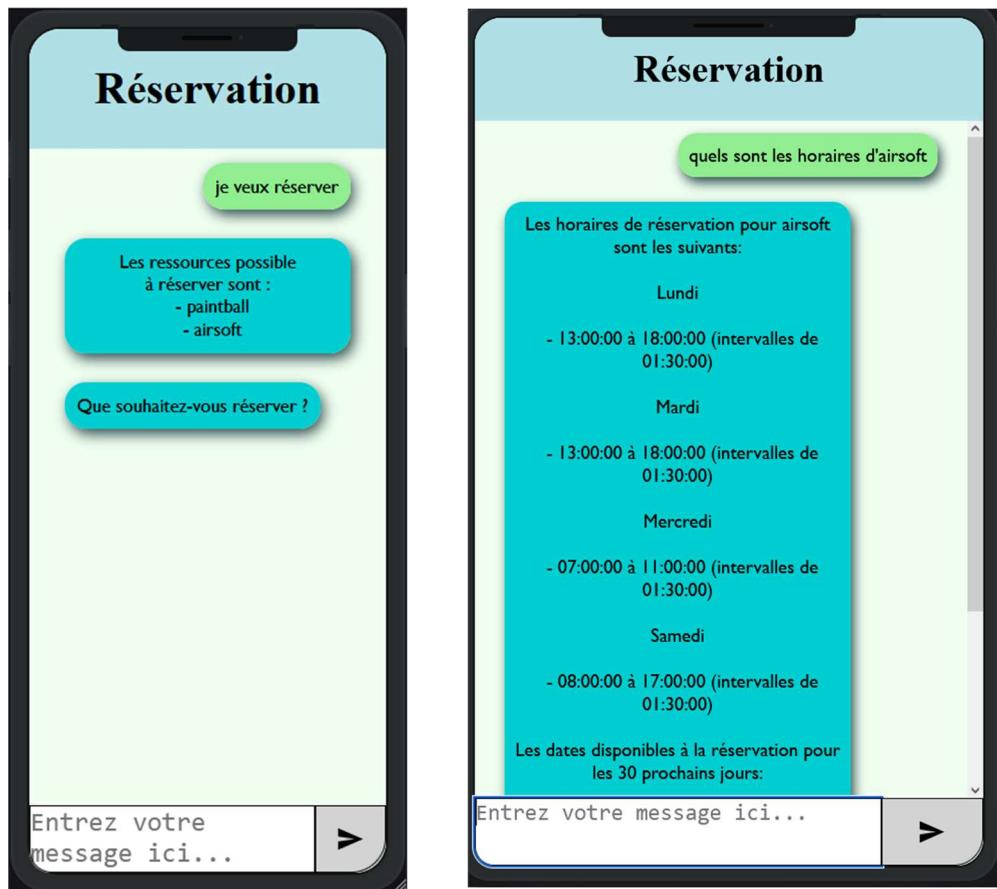


Illustration 33 : Captures d'écran des deux débuts de conversation

Source : Rodrigues dos Santos Fabio

Ci-dessus, on voit que le bot répond correctement aux deux demandes ; fournissant l'horaire d'airsoft ou proposant les divers ressources que l'on peut réserver. Il n'est pas possible actuellement dû à des limitations techniques de l'implémentation du bot de demander à vouloir réserver et d'ensuite demander les horaires d'une ressource cependant. La raison étant qu'il n'est pas possible pour le bot de sortir d'un form pour rentrer dans une story différente et seul l'annulation du form au préalable pourrait permettre de sortir du form.

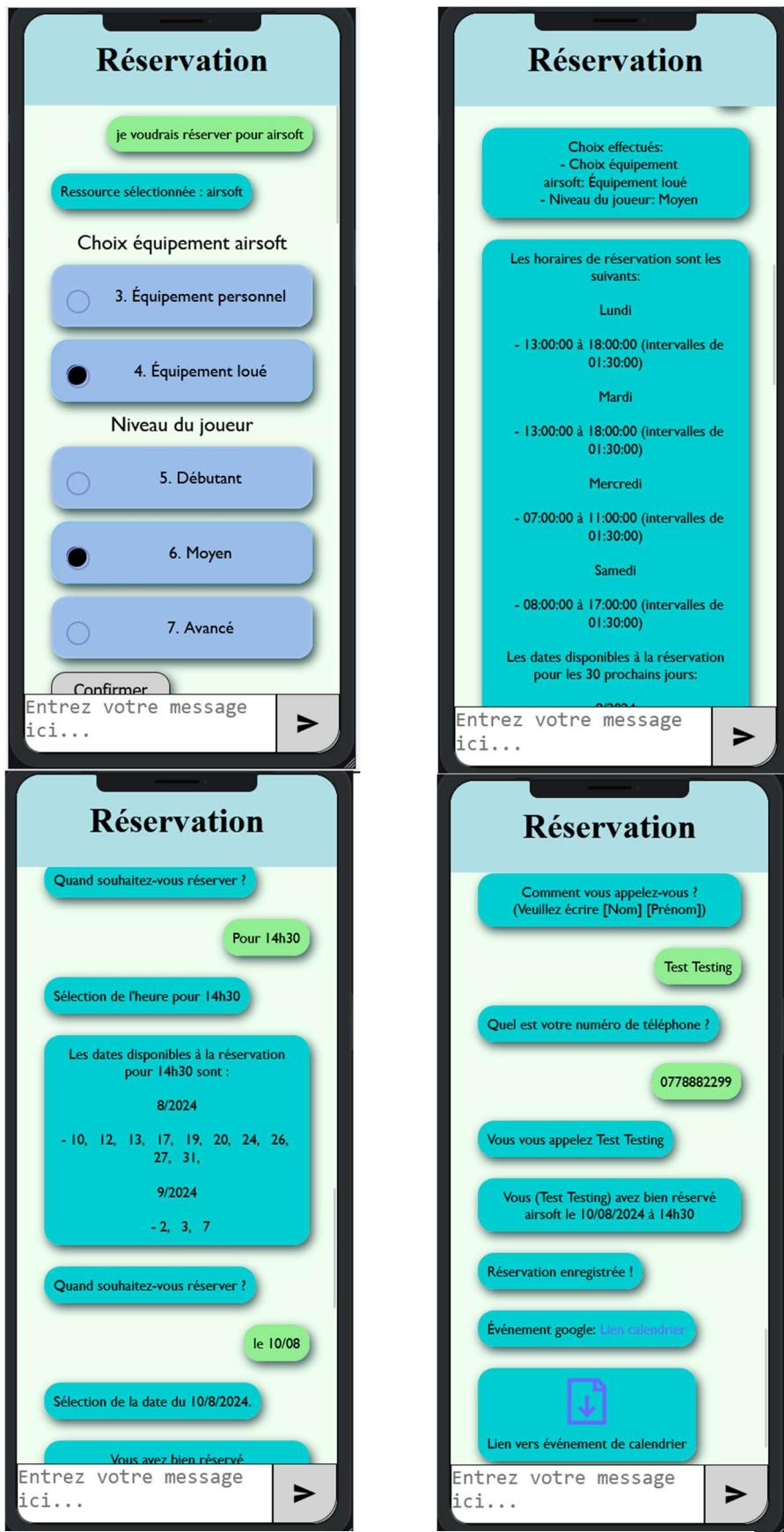


Illustration 34 : Captures d'écran d'une réservation

Source : Rodrigues dos Santos Fabio

Les liens fournis en plus de la confirmation de réservations donnent ce qui suit :

- Un lien Google Calendar qui permet d'enregistrer un événement comportant les informations fournies pour la réservation ainsi que l'heure et date de cette dernière
- Un fichier au format CalDAV qui permet à tout appareil de nos jours d'enregistrer un événement dans le calendrier par défaut de l'appareil.

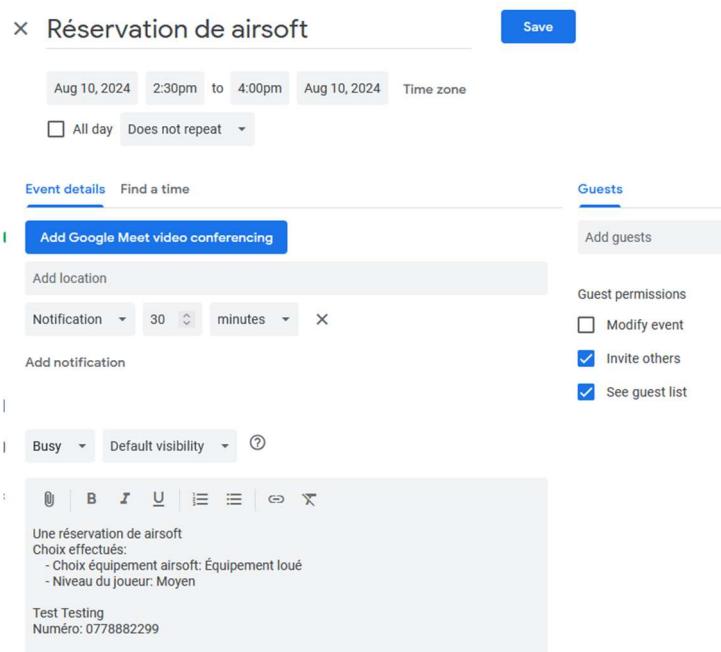


Illustration 35 : Capture d'écran d'une création d'événement Google Calendar

Source : Rodrigues dos Santos Fabio

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:WiREV calendar
BEGIN:VEVENT
SUMMARY:Réservation de airsoft
DTSTART;TZID=Europe/Paris:20240810T143000
DTEND;TZID=Europe/Paris:20240810T160000
DESCRIPTION:Une réservation de airsoft
Choix effectués:
- Choix équipement airsoft: Équipement loué
- Niveau du joueur: Moyen
Test Testing
Numéro: 0778882299
END:VEVENT
END:VCALENDAR
```

Contenu du fichier .ICS/au format CalDAV reçu

Après qu'une réservation est finalisée, un événement est créé dans le calendrier Google associé au projet local :

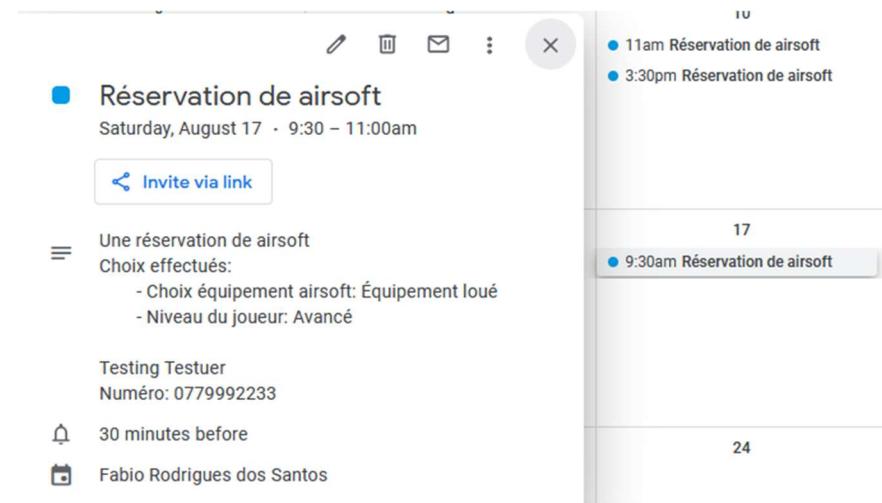


Illustration 36 : Capture d'écran d'un événement Google Calendar après réservation

Source : Rodrigues dos Santos Fabio

Des subtilités mises en place dû au fait que nous avons un bot intelligent est que l'on peut plutôt aisément changer d'heure ou date sélectionnée initialement ou même ouvrir la discussion en choisissant la ressource et une date/heure.

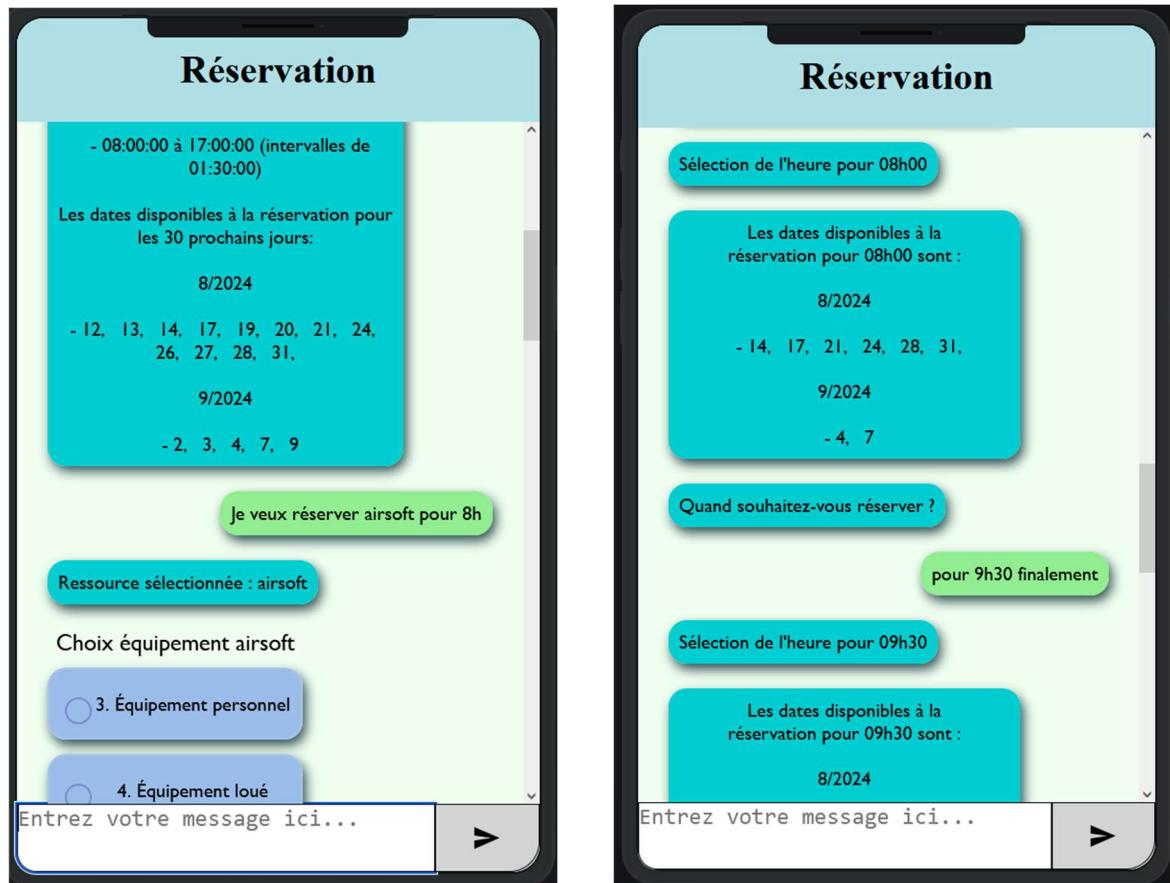


Illustration 38 : Captures d'écran démontrant le changement dynamique d'heure

Source : Rodrigues dos Santos Fabio

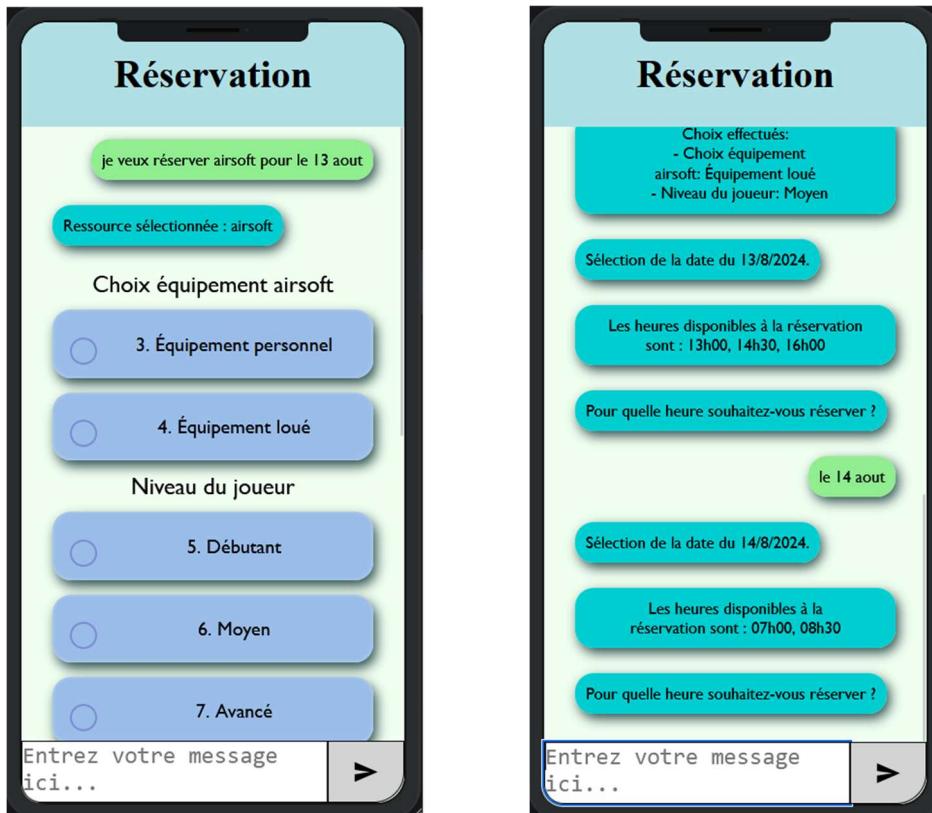


Illustration 37 : Captures d'écran démontrant le changement dynamique de date

Source : Rodrigues dos Santos Fabio

Il y a dans l'interface web la présence de radio boutons qui permettent de sélectionner en cliquant dessus les divers choix concernant une ressource. Afin d'assurer le fonctionnement en tout temps, peu importe l'interface, une attention particulière est de mesure lors de la sélection de choix. Ci-dessous quelques exemples concrets :

- Sélection d'un seul choix par option
- Sélection d'index de choix uniquement parmi ceux proposés

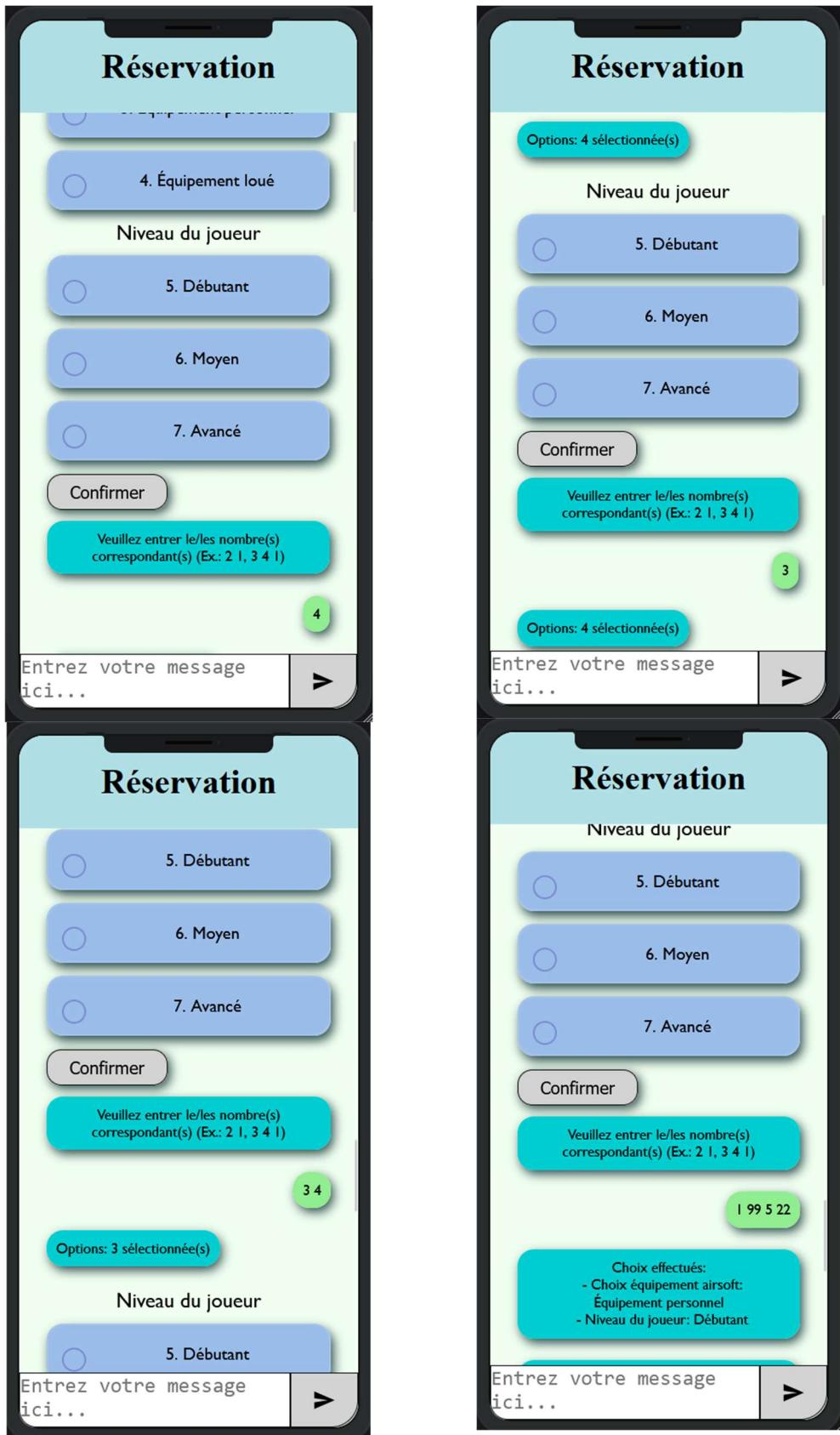


Illustration 39 : Captures d'écran d'utilisation des radio boutons

Source : Rodrigues dos Santos Fabio

8.2. CHATBOT TELEGRAM

Concernant l'interface Telegram, elle est moins polie que celle web car le temps nécessaire à la documentation et réalisation de fonctionnalités de confort comme les radio boutons pour les choix était bien plus long que le temps restant à la réalisation du projet.

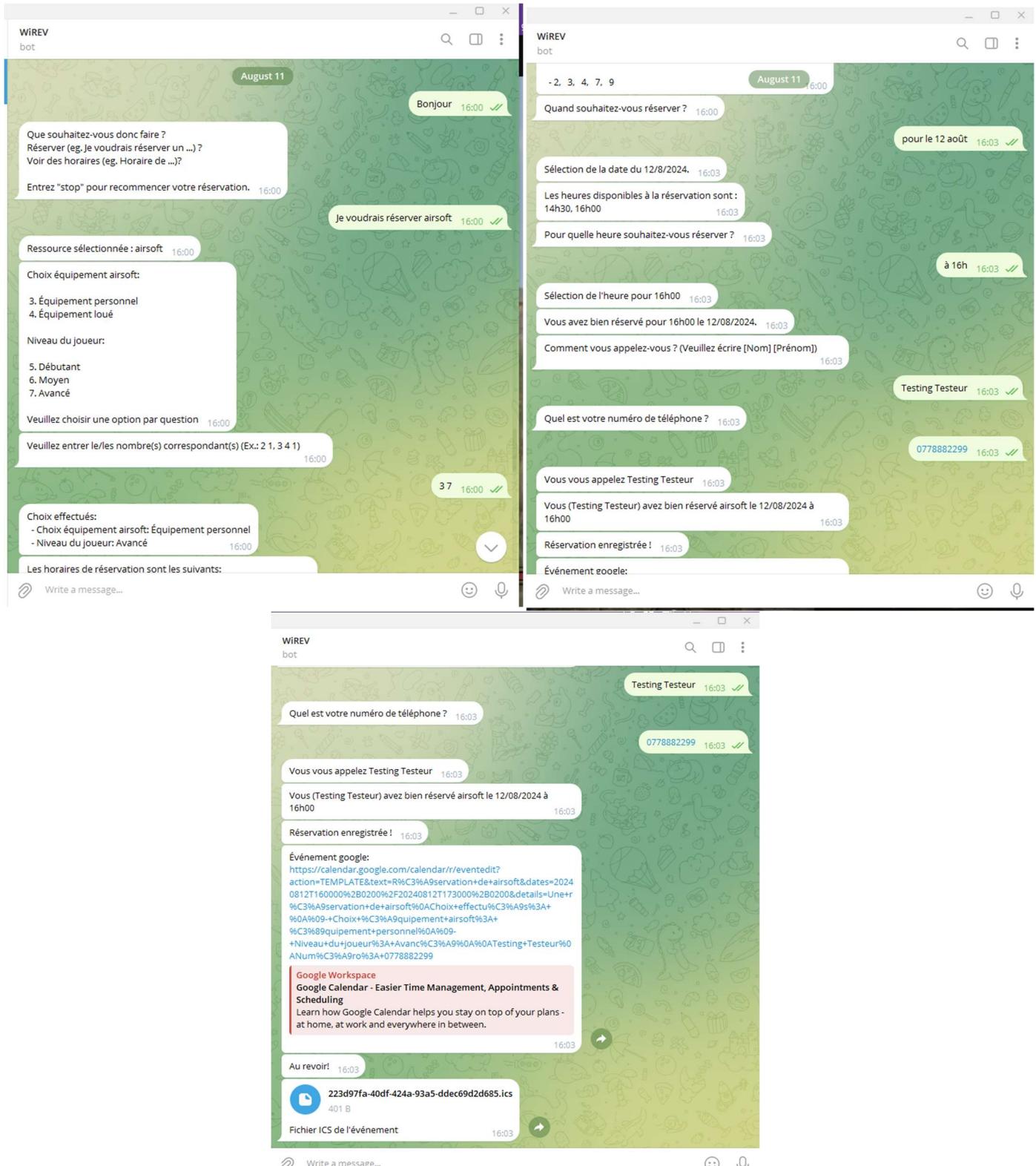


Illustration 40 : Captures d'écran d'une réservation sur Telegram

Source : Rodrigues dos Santos Fabio

8.3. COMMENTAIRES SUR LES RÉSULTATS

Les fonctionnalités prévues initialement sont majoritairement implémentées et fonctionnelles telles que :

- Effectuer une réservation
- Pouvoir changer d'avis sur un choix de date/heure/ressource en pleine réservation
- Modification facilité des ressources réservables pour la mise en place du bot
- Implémentation de formulaires qui vont boucler tant que l'information demandée n'est pas fournie
- Possibilité de mettre en place le bot sans restriction d'interface
- Possibilité d'avoir des ressources avec une multitude d'options sélectionnables

Cependant, une fonctionnalité n'indique pas qu'elle est parfaite et marche à coup sûr. Il subsiste encore quelques soucis notamment :

- Les choix de dates tout particulièrement ; Duckling ne supportant pas les dates en format xx.yy/xx.yy.zzzz/etc.
- Le fait que par moment le bot ne pourrait simplement pas comprendre ce que l'utilisateur dit s'il formule son message d'une façon non prévue dans le jeu de données présent.
- Il n'existe pas de manière concrète gérant les cas spéciaux où l'utilisateur entre quelque chose de non prévu par le bot ce qui pourrait dérouter la conversation principale.

- Il est possible que le bot provoque des erreurs lors de la réception du nom-prénom de l'utilisateur car il pourrait les confondre par un intent autre que « inform_nom_prenom »

En somme, une certaine quantité des soucis éventuels viennent du bot et du fonctionnement inhérent de la technologie NLU derrière. Comme tout modèle de machine learning, il nécessite un nombre de données suffisant et du temps passé à fine tuner le modèle jusqu'à atteindre un résultat satisfaisant ; ce qui peut prendre beaucoup de temps. Il est fort possible que d'autres soucis liés à cela soient présents mais n'ayant pas été détectés lors des tests.

Pour donner un exemple concret du problème, voici un cas survenus lors de la définition d'intents. L'idée étant que les intents définissant la réservation de ressource pouvait être séparés en deux de la sorte :

- Demande de réservation : eg. « Je voudrais réserver », « Je souhaite réserver », « Je veux réserver », etc.
- Demande de réservation avec ressource : eg. « Je veux réserver un terrain de <ressource> », « J'aimerais une salle de <ressource> », etc.

Comme ces deux intents étaient très similaires, Rasa avait des soucis à déterminer lequel choisir et provoquait beaucoup de problèmes au point où il n'était pas possible d'entamer la conversation. Il a fallu un bon nombre d'heures afin de réécrire le code de multiples fois jusqu'à trouver une solution qui convienne et qui soit fonctionnelle.

Par ailleurs, le manque de « spellchecking », soit un correcteur orthographique fait que par exemple :

Si l'entrée utilisateur est « airsoft », « airssoft » ou même « airsoftt », aucune entité ne sera détectée bien que l'intent, lui, sera correctement détecté. Si le mot est correctement orthographié le retour du bot sera donc : « **entities** : [**entity** : airsoft , ...] ». Le manque d'entité empêche par la suite la sélection de ressources et retourne une erreur.

CONCLUSION

En ce qui concerne le travail effectué pour ce travail de bachelor, les principaux intérêts furent la compréhension de ce qu'était un Chatbot concrètement parlant, comment on pouvait le décomposer en plusieurs parties distinctes qui une fois combinée permettent d'avoir un bot fonctionnel et finalement d'arriver à en implémenter de la meilleure manière possible selon les résultats observés lors l'étape de prototypage. Le focus initial a été principalement mis sur la compréhension du moteur de traitement de données étant un moteur NLP, les diverses recherches approfondies réalisées et la programmation de quelques prototypes qui ont permis de donner un bon moyen de valider notre compréhension théorique du NLP.

Ensuite, grâce à la réalisation de davantage de prototypes, nous avons pu nous rendre compte que la technologie que l'on pensait initialement pouvoir être bien utile n'était finalement pas ce qui correspondait le mieux à la réalisation d'un Chatbot de réservations. Cependant, cela fut un moyen d'en apprendre davantage sur les mécanismes inhérents de divers autres applications faisant usage de NLP dans des modèles de Machine Learning. Le moteur de recherche Google avec BERT et que certains aspects étudiés lors de mes recherches sur TAPAS sont revenus comme les transformers dans la pipeline de traitement NLU avec Rasa. Et bien que cela n'ait pas porté ses fruits en fin de compte, nous avons pu effectuer un travail d'exploration des possibilités qui a été particulièrement utile car cela a permis de faire les meilleurs choix technologiquement parlant pour la réalisation du bot.

Pour ce qui est de la rédaction du rapport, elle fut très instructive car bien qu'ayant déjà fait usage de réseaux convolutifs, perceptrons ou de Réseaux neuronaux récurrents ou RNN dans d'autres cours qui ont permis d'analyser des images ou prédire des valeurs numériques, nous n'avions pas encore fait usage de modèles de machine learning dans le cas de traitement de texte comme certaines des techniques en NLP comme du POS

Tagging, NER, analyses de sentiments, etc. D'ailleurs, grâce aux rendez-vous hebdomadaires avec M. Niklaus Eggenberg, j'ai pu à la fois préciser davantage la direction dans laquelle j'allais mener ce projet et dans les moments plus troubles recevoir de la guidance afin de ne jamais rester complètement bloqué, ce qui a un moment donné arriva presque lors de mes premières tentatives d'implémentation d'un prototype utilisant TAPAS. Additionnellement, ces rendez-vous ont eu pour effet de m'encourager à garder un rythme régulier de travail tout au long de la rédaction, réalisation du chatbot et de mes recherches.

PROBLÈMES RENCONTRÉS

La réalisation de ce projet ne s'est pas faite sans accrocs à divers étapes clé. Initialement, des soucis de dockerisation du projet sont présents étant des problèmes de montage de volumes dans lequel les divers bouts de codes sont répertoriés et d'autres de communication inter-conteneur. Le soucis de communication étant dû au fait que les conteneurs interagissent entre eux avec leur nom de service et non par IP. Ces problèmes de début de projets n'étaient pas très contraignants et ont été résolus assez rapidement dans la journée même; ce qui n'était pas le cas des soucis à venir.

Un second et un des problèmes majeurs de tout le projet était toute la story principale de réservation qui menait, de manière aléatoire et presque imprévisible, à des fallbacks soit des erreurs qui mettaient fin à la conversation lorsque le bot ne savait pas quoi faire. En effet, pour faciliter la lisibilité et modulariser la logique de conversation, la story était subdivisé en de multiples stories interconnectées par des « checkpoints ». L'utilité des checkpoints permettaient de simplement couper une longue story en de multiples plus petites qui étaient sélectionnées selon les actions précédemment effectuées. Hélas, la présence des nombreux checkpoints empiétaient sur le bon fonctionnement du code et a

nécessité quelques jours de réflexion et de restructuration complète des stories. La solution fut de n'employer qu'une seule story et pas de checkpoints bien que davantage de tests seraient utiles pour déterminer s'il serait tout de même possible d'en faire usage à des endroits clés.

Un troisième problème était de réussir à trouver une implémentation la plus concise et propre possible de pouvoir ouvrir la discussion avec un message comportant à la fois la ressource voulant être réservée et une date et/ou heure. Finalement, après réflexion une solution a été trouvée en améliorant la fonction de préservation de ressources pour y inclure la pré-réservation de dates et/ou heures.

Après avoir réglé les problèmes majeurs venant du bot, nous nous sommes ensuite concentrés sur l'implémentation d'un frontend. Mais dû à la nature du développement d'applications web, une multitude de soucis sont survenus côté affichage des informations récupérées du bot. Comme le framework react fonctionne à base de rafraîchissement de composants de manière dynamique, cela a demandé de réfléchir à la façon d'implémenter l'affichage du chat tout en gardant l'historique des messages et les messages spéciaux comme les choix cliquables. Et pour conclure, un des derniers soucis survenus lors de l'ajout du frontend fut l'implémentation des sockets. Le soucis survenus n'était pas tant dans la logique du code lié aux sockets mais dans la possibilité de connecter le backend et le frontend ensemble sans que cela résulte en des problèmes de CORS ou d'adresse inatteignable. Le temps requis à comprendre d'où venait le problème ainsi que de le régler était dans l'ordre de presque une journée.

AMÉLIORATIONS POSSIBLES

En l'état, le Chatbot est fonctionnel et assez généralisé dans son implémentation pour permettre aisément de mettre en place un système de réservation pour quelconque ressource que ce soit. Cependant il y a tout de même une variété de façons de rendre l'expérience utilisateur la plus confortable possible.

- L'ajout de la possibilité de réserver/débuter la réservation en demandant une place pour un jour de la semaine en plus du choix de la date et heure. De cette manière, le panel des choix et interactions utilisateurs est davantage élargit et permet de lever encore plus le coté restreint pouvant exister dans de tels systèmes de réservations.
- L'ajout d'un moyen de confirmer le numéro ou adresse mail fournie par l'utilisateur pour endiguer la possibilité que l'utilisateur se soit trompé et ne reçoive pas d'appels lorsqu'un imprévu survient pour sa réservation par exemple.
- Ajout de la possibilité de modifier une réservation existante voire de pouvoir l'annuler.
- Proposer, lorsque l'utilisateur demande une heure erronée par exemple, plusieurs heures adjacentes qui elles sont valides.
- Ajout d'un correcteur orthographique traitant les entrées utilisateur. Actuellement, si l'utilisateur n'entre pas exactement le même nom de ressource, elle n'est pas détectée par le bot ou comprise. De ce fait, ajouter un correcteur permettrait au bot de comprendre la ressource même si en entrée la ressource fournie possède une erreur d'orthographe.
- Permettre la réservation jusqu'à une limite définie d'un même créneau horaire par de multiples clients.

- L'ajout d'un panel administrateur permettant de visualiser toutes les réservations, de les modifier ou même supprimer

RÉFÉRENCES DOCUMENTAIRES

1. A Beginner's Introduction to NER (Named Entity Recognition), [en ligne]. Disponible à l'adresse : <https://www.analyticsvidhya.com/blog/2021/11/a-beginners-introduction-to-ner-named-entity-recognition/> [consulté le 4 janvier 2024].
2. A complete Guide to Named Entity Recognition (NER) in 2024, 2022 *Nanonets Intelligent Automation, and Business Process AI Blog* [en ligne]. Disponible à l'adresse : <https://nanonets.com/blog/named-entity-recognition-with-nltk-and-spacy/> [consulté le 12 février 2024].
3. ALBURGER, Jenna. Rule-Based Chatbots vs. AI Chatbots: Key Differences. *Hubtype* [en ligne]. Disponible à l'adresse : <https://www.hubtype.com/blog/rule-based-chatbots-vs-ai-chatbots> [consulté le 4 janvier 2024].
4. Apache OpenNLP, [en ligne]. Disponible à l'adresse : <https://opennlp.apache.org/> [consulté le 8 février 2024].
5. AYER, Benjamin, 2020. What Makes a Chatbot Conversational? *Inbenta* [en ligne]. 20 mars 2020. Disponible à l'adresse : <https://www.inbenta.com/articles/what-makes-a-chatbot-conversational/> [consulté le 4 janvier 2024].
6. BOCKLISCH, Tom et al., 2024. *Task-Oriented Dialogue with In-Context Learning* [en ligne]. arXiv:2402.12234. arXiv. arXiv:2402.12234. Disponible à l'adresse : <http://arxiv.org/abs/2402.12234> [consulté le 29 février 2024]. arXiv:2402.12234 [cs]
7. CAPACITY, Team. How does an AI chatbot work, and what does it mean for the future? *Capacity* [en ligne]. Disponible à l'adresse : <https://capacity.com/learn/ai-chatbots/how-does-an-ai-chatbot-work/> [consulté le 4 janvier 2024].

8. Chatbot Evolution: ChatGPT Vs. Rule-based - Analytics Vidhya, [en ligne]. Disponible à l'adresse : <https://www.analyticsvidhya.com/blog/2023/05/chatbot-evolution-chatgpt-vs-rule-based/> [consulté le 4 janvier 2024].
9. CHOUDHRY, Arjun et al., 2023. *Adversarial Adaptation for French Named Entity Recognition* [en ligne]. arXiv:2301.05220. arXiv. arXiv:2301.05220. Disponible à l'adresse : <http://arxiv.org/abs/2301.05220> [consulté le 11 février 2024]. arXiv:2301.05220 [cs]
10. CHURCH, Bella, 2023. 5 types of chatbot and how to choose the right one. *IBM Blog* [en ligne]. 5 septembre 2023. Disponible à l'adresse : <https://www.ibm.com/blog/chatbot-types/www.ibm.com/blog/chatbot-types> [consulté le 4 janvier 2024].
11. Command Line Interface, 2024 [en ligne]. Disponible à l'adresse : <https://rasa.com/docs/rasa/command-line-interface/> [consulté le 21 février 2024].
12. CONTRIBUTORS (HTTPS://GITHUB.COM/HUGGINGFACE/TRANSFORMERS/GRAPHS/CONTRIBUTORS), The Hugging Face team (past and future) with the help of all our. *transformers: State-of-the-art Machine Learning for JAX, PyTorch and TensorFlow* [logiciel]. Version 4.36.2. [consulté le 4 janvier 2024]. Disponible à l'adresse : <https://github.com/huggingface/transformers> [consulté le 4 janvier 2024].
13. Conversational AI with Language Models, 2024 [en ligne]. Disponible à l'adresse : <https://rasa.com/docs/rasa-pro/calm/> [consulté le 28 février 2024].
14. Difference between a bot, a chatbot, a NLP chatbot and all the rest? | Botpress Blog, [en ligne]. Disponible à l'adresse : <https://botpress.com/blog/nlp-chatbot> [consulté le 4 janvier 2024].

15. DUTTA, Bhumika. What is Topic Modelling in NLP? | Analytics Steps. [en ligne]. Disponible à l'adresse : <https://www.analyticssteps.com/blogs/what-topic-modelling-nlp> [consulté le 4 janvier 2024].
16. *facebook/duckling* [logiciel] [en ligne]. 21 février 2024. Meta. [consulté le 21 février 2024]. Disponible à l'adresse : <https://github.com/facebook/duckling> [consulté le 21 février 2024].
17. Getting Started with Rasa X as an Existing Rasa User | Rasa Blog, *Rasa* [en ligne]. Disponible à l'adresse : <https://rasa.com/blog/rasa-x-getting-started-as-a-current-rasa-user/> [consulté le 3 mars 2024].
18. *google-research/tapas* [logiciel] [en ligne]. 29 février 2024. Google Research. [consulté le 3 mars 2024]. Disponible à l'adresse : <https://github.com/google-research/tapas> [consulté le 3 mars 2024].
19. HERZIG, Jonathan et al., 2020. TAPAS: Weakly Supervised Table Parsing via Pre-training. In : *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4320-4333. 2020. DOI [10.18653/v1/2020.acl-main.398](https://doi.org/10.18653/v1/2020.acl-main.398). arXiv:2004.02349 [cs]
20. How AI chatbots like ChatGPT or Bard work – visual explainer | Technology | The Guardian, [en ligne]. Disponible à l'adresse : <https://www.theguardian.com/technology/ng-interactive/2023/nov/01/how-ai-chatbots-like-chatgpt-or-bard-work-visual-explainer> [consulté le 4 janvier 2024].
21. How do Chatbots work? A Guide to the Chatbot Architecture, *Maruti Techlabs* [en ligne]. Disponible à l'adresse : <https://marutitech.com/chatbots-work-guide-chatbot-architecture/> [consulté le 27 février 2024].
22. How Menu-Based Chatbots Can Streamline Your Business Processes, 2023 [en ligne]. Disponible à l'adresse :

<https://www.docomatic.ai/blog/chatbots/menu-based-chatbot/> [consulté le 4 janvier 2024].

23. How to Create a Postgres Database in Docker, 2023 *1kevinson* [en ligne]. Disponible à l'adresse : <https://1kevinson.com/how-to-create-a-postgres-database-in-docker/> [consulté le 14 mai 2024].
24. 2023. *How To Create A Telegram Bot In Python For Beginners (2023 Tutorial)*, INDENTLY [en ligne]. 27 mars 2023. Disponible à l'adresse : <https://www.youtube.com/watch?v=vZtm1wuA2yc> [consulté le 2 août 2024].
25. How to Interact with Databases using SQLAlchemy with PostgreSQL - CoderPad, 2022 [en ligne]. Disponible à l'adresse : <https://coderpad.io/blog/development/sqlalchemy-with-postgresql/> [consulté le 31 juillet 2024].
26. IBM TECHNOLOGY, 2022. *What are Transformers (Machine Learning Model)?* [en ligne]. 11 mars 2022. Disponible à l'adresse : <https://www.youtube.com/watch?v=ZXiruGOCn9s> [consulté le 2 mars 2024].
27. IBM TECHNOLOGY, 2023. *Generative vs Rules-Based Chatbots* [en ligne]. 5 juin 2023. Disponible à l'adresse : https://www.youtube.com/watch?v=lZjUS_8btEo [consulté le 4 janvier 2024].
28. Introduction to Rasa Open Source & Rasa Pro, 2024 [en ligne]. Disponible à l'adresse : <https://rasa.com/docs/rasa/> [consulté le 21 février 2024].
29. JABLONSKI, Real. Natural Language Processing With Python's NLTK Package – Real Python. [en ligne]. Disponible à l'adresse : <https://realpython.com/nltk-nlp-python/> [consulté le 4 janvier 2024].
30. JAISWAL, Abhishek, 2022. NLP Tutorials Part -I from Basics to Advance. *Analytics Vidhya* [en ligne]. 13 janvier 2022. Disponible à l'adresse :

<https://www.analyticsvidhya.com/blog/2022/01/nlp-tutorials-part-i-from-basics-to-advance/> [consulté le 4 janvier 2024].

31. JAY ALAMMAR, 2021. *Language Processing with BERT: The 3 Minute Intro (Deep learning for NLP)* [en ligne]. 15 juin 2021. Disponible à l'adresse : <https://www.youtube.com/watch?v=ioGry-89gqE> [consulté le 2 mars 2024].
32. KAKODKAR, Saish, 2022. Different Types of Chatbots. *ClaySys Technologies* [en ligne]. 31 octobre 2022. Disponible à l'adresse : <https://www.claysys.com/blog/types-of-chatbots/> [consulté le 4 janvier 2024].
33. KAVLAKOGLU, Eda, 2020. NLP vs. NLU vs. NLG: the differences between three natural language processing concepts. *IBM Blog* [en ligne]. 12 novembre 2020. Disponible à l'adresse : <https://www.ibm.com/blog/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts> [consulté le 28 février 2024].
34. KHANNA, Chetna, 2021. Text pre-processing: Stop words removal using different libraries. *Medium* [en ligne]. 10 février 2021. Disponible à l'adresse : <https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a> [consulté le 4 janvier 2024].
35. KLARA | Logiciels de gestion pour les PME, [en ligne]. Disponible à l'adresse : <https://www.klara.ch/fr-ch/> [consulté le 28 février 2024].
36. KUMARI, Kajal, 2023. Building Language Models: A Step-by-Step BERT Implementation Guide. *Analytics Vidhya* [en ligne]. 29 juin 2023. Disponible à l'adresse : <https://www.analyticsvidhya.com/blog/2023/06/step-by-step-bert-implementation-guide/> [consulté le 4 janvier 2024].
37. Lemmatization, *Engati* [en ligne]. Disponible à l'adresse : <https://www.engati.com/glossary/lemmatization> [consulté le 4 janvier 2024].

38. Meetme - Simplify Your Appointment Scheduling, 2023 [en ligne]. Disponible à l'adresse : <https://www.meetme.io/en> [consulté le 28 février 2024].
39. MEHTA, Ansh et al., 2021. SPELL CORRECTION AND SUGGESTION USING LEVENSHTEIN DISTANCE. . Vol. 08, no 09.
40. Models & Languages · spaCy Usage Documentation, *Models & Languages* [en ligne]. Disponible à l'adresse : <https://spacy.io/usage/models> [consulté le 21 février 2024].
41. MUNNANGI, Manikanta, 2019. A Comprehensive Guide To NLP. *Medium* [en ligne]. 22 novembre 2019. Disponible à l'adresse : <https://towardsdatascience.com/nlp-with-spacy-part-1-beginner-guide-to-nlp-4b9460652994> [consulté le 1 mars 2024].
42. Named Entity Recognition, 2021 *GeeksforGeeks* [en ligne]. Disponible à l'adresse : <https://www.geeksforgeeks.org/named-entity-recognition/> [consulté le 4 janvier 2024].
43. Natural Language Processing (NLP) Tutorial, 2023 *GeeksforGeeks* [en ligne]. Disponible à l'adresse : <https://www.geeksforgeeks.org/natural-language-processing-nlp-tutorial/> [consulté le 4 janvier 2024].
44. Natural Language Processing (NLP): What Is It & How Does it Work?, *MonkeyLearn* [en ligne]. Disponible à l'adresse : <https://monkeylearn.com/natural-language-processing/> [consulté le 4 janvier 2024].
45. NLTK: A Beginners Hands-on Guide to Natural Language Processing, [en ligne]. Disponible à l'adresse : <https://www.analyticsvidhya.com/blog/2021/07/nltk-a-beginners-hands-on-guide-to-natural-language-processing/> [consulté le 4 janvier 2024].

46. NLTK :: Natural Language Toolkit, [en ligne]. Disponible à l'adresse : <https://www.nltk.org/> [consulté le 11 février 2024].
47. PIRGE, Gursev, 2023a. TAPAS — Question Answering from Tables in Spark NLP. *John Snow Labs* [en ligne]. 26 avril 2023. Disponible à l'adresse : <https://www.johnsnowlabs.com/tapas-question-answering-from-tables-in-spark-nlp/> [consulté le 4 janvier 2024].
48. PIRGE, Gursev, 2023b. TAPAS — Question Answering from Tables. *John Snow Labs* [en ligne]. 2 mai 2023. Disponible à l'adresse : <https://medium.com/john-snow-labs/tapas-question-answering-from-tables-1796a7b90f49> [consulté le 4 janvier 2024].
49. PostgreSQL Python: Connecting to PostgreSQL Server, *PostgreSQL Tutorial* [en ligne]. Disponible à l'adresse : <https://www.postgresqltutorial.com/postgresql-python/connect/> [consulté le 31 juillet 2024].
50. Prise de rendez-vous en ligne avec une application professionnel, [en ligne]. Disponible à l'adresse : <https://agenda.ch> [consulté le 28 février 2024].
51. Python quickstart | Google Calendar, *Google for Developers* [en ligne]. Disponible à l'adresse : <https://developers.google.com/calendar/api/quickstart/python> [consulté le 1 août 2024].
52. RAJ, Nikhil, 2021. Starters Guide to Sentiment Analysis using Natural Language Processing. *Analytics Vidhya* [en ligne]. 15 juin 2021. Disponible à l'adresse : <https://www.analyticsvidhya.com/blog/2021/06/nlp-sentiment-analysis/> [consulté le 4 janvier 2024].
53. RASA, 2021. *Conversational AI with Rasa: Custom Actions* [en ligne]. 7 décembre 2021. Disponible à l'adresse : <https://www.youtube.com/watch?v=VcbfcjsjBBIg> [consulté le 21 février 2024].

54. Rasa Documentation, [en ligne]. Disponible à l'adresse :
<https://rasa.com/docs/rasa/pages/http-api/> [consulté le 21 février 2024].
55. Rasa Platform: Rasa Pro, 2022 *Rasa* [en ligne]. Disponible à l'adresse :
<https://rasa.com/product/rasa-pro/> [consulté le 4 mars 2024].
56. RasaRestaurantBot/README.md at master · kothiyayogesh/RasaRestaurantBot, GitHub [en ligne]. Disponible à l'adresse :
<https://github.com/kothiyayogesh/RasaRestaurantBot/blob/master/README.md> [consulté le 4 janvier 2024].
57. REDDY, Ajay Vardhan, 2021. Getting started with NLP using NLTK Library. *Analytics Vidhya* [en ligne]. 30 juillet 2021. Disponible à l'adresse :
<https://www.analyticsvidhya.com/blog/2021/07/getting-started-with-nlp-using-nltk-library/> [consulté le 11 février 2024].
58. REITER, Ehud et DALE, Robert, 1997. Building applied natural language generation systems. *Natural Language Engineering*. Vol. 3, no 1, pp. 57-87. DOI [10.1017/S1351324997001502](https://doi.org/10.1017/S1351324997001502).
59. RODRIGUEZ, Jesus, 2020. Google TAPAS is a BERT-Based Model to Query Tabular Data Using Natural Language. *DataSeries* [en ligne]. 7 septembre 2020. Disponible à l'adresse : <https://medium.com/dataseries/google-tapas-is-a-bert-based-model-to-query-tabular-data-using-natural-language-2435a386b43f> [consulté le 4 janvier 2024].
60. SAEED, Mehreen, 2022. A Gentle Introduction to Positional Encoding in Transformer Models, Part 1. *MachineLearningMastery.com* [en ligne]. 19 septembre 2022. Disponible à l'adresse :
<https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/> [consulté le 6 février 2024].

61. SANAD, Mohdsanadzakirizvi@gmail.com, 2019. Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework. *Analytics Vidhya* [en ligne]. 25 septembre 2019. Disponible à l'adresse : <https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/> [consulté le 4 janvier 2024].
62. SHILEDARBAXI, Nikita, 2021. Guide To TAPAS (TAble PArSing) - A Technique To Retrieve Information From Tabular Data Using NLP. *Analytics India Magazine* [en ligne]. 13 février 2021. Disponible à l'adresse : <https://analyticsindiamag.com/guide-to-tapas-table-parsing-a-technique-to-retrieve-information-from-tabular-data-using-nlp/> [consulté le 4 janvier 2024].
63. SHRIDHAR, Kumar, 2020. Natural Language Understanding for Chatbots. *NeuralSpace* [en ligne]. 26 décembre 2020. Disponible à l'adresse : <https://medium.com/neuralspace/natural-language-understanding-for-chatbots-2eb7a81b9390> [consulté le 3 mars 2024].
64. Software - The Stanford Natural Language Processing Group, [en ligne]. Disponible à l'adresse : <https://nlp.stanford.edu/software/> [consulté le 8 février 2024].
65. Stages of Natural Language Processing (NLP) | byteiota, 2020 [en ligne]. Disponible à l'adresse : <https://byteiota.com/stages-of-nlp/> [consulté le 1 mars 2024].
66. Stemming, Engati [en ligne]. Disponible à l'adresse : <https://www.engati.com/glossary/stemming> [consulté le 4 janvier 2024].
67. STOY, Lazarina, 2023. Five phases of NLP and how to incorporate them into your SEO journey. *Oncrawl - Technical SEO Data* [en ligne]. 28 février 2023. Disponible à l'adresse : <https://www.oncrawl.com/technical-seo/five-phases-nlp-how-incorporate-them-into-seo-journey/> [consulté le 1 mars 2024].

68. TAPAS, [en ligne]. Disponible à l'adresse : https://huggingface.co/docs/transformers/model_doc/tapas [consulté le 4 janvier 2024].
69. The 3 Kinds of Chatbots You'll Meet | OvationCXM, [en ligne]. Disponible à l'adresse : <https://www.ovationc xm.com/blog/3-kinds-chatbots-youll-meet> [consulté le 4 janvier 2024].
70. The Stanford Natural Language Processing Group, [en ligne]. Disponible à l'adresse : <https://nlp.stanford.edu/software/tagger.shtml> [consulté le 5 mars 2024 a].
71. The Stanford Natural Language Processing Group, [en ligne]. Disponible à l'adresse : <https://nlp.stanford.edu/software/tagger.shtml> [consulté le 11 février 2024 b].
72. The Ultimate Guide to Transformer Deep Learning, [en ligne]. Disponible à l'adresse : <https://www.turing.com/kb/brief-introduction-to-transformers-and-their-power> [consulté le 2 mars 2024].
73. Understanding Part-of-Speech Tagging in NLP: Techniques and Applications - Shiksha Online, [en ligne]. Disponible à l'adresse : <https://www.shiksha.com/online-courses/articles/pos-tagging-in-nlp/> [consulté le 4 janvier 2024].
74. Using Neural Networks to Find Answers in Tables, 2020 [en ligne]. Disponible à l'adresse : <https://blog.research.google/2020/04/using-neural-networks-to-find-answers.html> [consulté le 4 janvier 2024].
75. VERSLOOT, Christian. Machine Learning for Table Parsing: TAPAS. *GitHub* [en ligne]. Disponible à l'adresse : <https://github.com/christianversloot/machine-learning-articles/blob/main/easy-table-parsing-with-tapas-machine-learning-and-huggingface-transformers.md> [consulté le 5 mars 2024].

76. What Is a Chatbot? | IBM, [en ligne]. Disponible à l'adresse : <https://www.ibm.com/topics/chatbots> [consulté le 27 février 2024].
77. What is BERT | BERT For Text Classification, [en ligne]. Disponible à l'adresse : <https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/> [consulté le 4 janvier 2024].
78. What is BERT (Language Model) and How Does It Work?, *Enterprise AI* [en ligne]. Disponible à l'adresse : <https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model> [consulté le 4 janvier 2024].
79. What is Named Entity Recognition (NER)? Methods, Use Cases, and Challenges, [en ligne]. Disponible à l'adresse : <https://www.datacamp.com/blog/what-is-named-entity-recognition-ner> [consulté le 4 janvier 2024].
80. What Is Natural Language Processing, 2020 *MonkeyLearn Blog* [en ligne]. Disponible à l'adresse : <https://monkeylearn.com/blog/what-is-natural-language-processing/> [consulté le 4 janvier 2024].
81. What is Natural Language Understanding (NLU)? | Definition from TechTarget, *Enterprise AI* [en ligne]. Disponible à l'adresse : <https://www.techtarget.com/searchenterpriseai/definition/natural-language-understanding-NLU> [consulté le 1 mars 2024].
82. What Is Natural Language Understanding (NLU)?, 2022 [en ligne]. Disponible à l'adresse : <https://thelevel.ai/blog/natural-language-understanding/> [consulté le 26 février 2024].
83. WUTTKE, Laurenz, 2023. NLP vs. NLU vs. NLG: Unterschiede, Funktionen und Beispiele. *datasolut GmbH* [en ligne]. 24 mai 2023. Disponible à l'adresse : <https://datasolut.com/natural-language-processing-vs-nlu-vs-nlg-unterchiede-funktionen-und-beispiele/> [consulté le 1 mars 2024].

84. YANNIC KILCHER, 2020. *TAPAS: Weakly Supervised Table Parsing via Pre-training (Paper Explained)* [en ligne]. 5 mai 2020. Disponible à l'adresse : <https://www.youtube.com/watch?v=cIUtRNhY6Rw> [consulté le 6 janvier 2024].
85. YOG, 2021. *kothiyayogesh/RasaRestaurantBot* [logiciel] [en ligne]. 14 septembre 2021. [consulté le 4 janvier 2024]. Disponible à l'adresse : <https://github.com/kothiyayogesh/RasaRestaurantBot> [consulté le 4 janvier 2024].