



DiscordEat

-BE Intégration-

ETUDIANTS

Lucas Amblard
Manon Augé Couffin
Loan Bernat

Tobias Brissaud
Samuel Missier
Damien Soubirant

ENSEIGNANTS

Isabelle Ferrané
Philippe Truillet

Sommaire

| | |
|---|-----------|
| I. Conception et Architecture globale de l'application..... | 4 |
| 1. Choix du sujet : Assistant dédié à la restauration..... | 4 |
| 2. Architecture de l'application globale..... | 4 |
| 3. Répartition des tâches..... | 5 |
| 4. RASA..... | 5 |
| a. Langage commun..... | 5 |
| b. Commande..... | 6 |
| i. Lookup table..... | 8 |
| ii. Formulaire..... | 9 |
| c. Adresse..... | 9 |
| d. Finalisation de la commande..... | 10 |
| 5. Discord..... | 10 |
| a. Création du bot discord..... | 10 |
| b. Développement des commandes de base..... | 10 |
| c. Communication bot-discord / rasa..... | 12 |
| d. Intégration de la voix et de l'ouïe..... | 13 |
| 6. La Carte..... | 14 |
| a. Transcrire et manipuler la carte..... | 15 |
| b. Gérer le panier et la commande..... | 15 |
| c. Lien au chatbot..... | 15 |
| 7. Crédit de la carte..... | 15 |
| II. Traitement de l'accessibilité (point d'attention, démarche...) | 16 |
| 1. Points d'attention..... | 16 |
| 2. Démarche..... | 17 |
| 3. Impact sur l'expérience utilisateur..... | 17 |
| III. Évaluation du chatBot (performance et ergonomie) | 18 |
| 1. Première évaluation..... | 18 |
| 2. Mesures prises pour optimiser la performance..... | 18 |
| a. Évaluation d'intents..... | 19 |
| b. Évaluation d'entités..... | 20 |
| c. Évaluation d'actions..... | 21 |
| 3. Ergonomie et convivialité de DiscordEat..... | 22 |
| Bilan..... | 23 |
| 1. Bilan technique..... | 23 |
| 2. Bilan humain..... | 23 |
| Annexe..... | 24 |

Le Bureau d'étude intégration de la mineure Interaction Avancée a constitué une semaine d'exploration intensive au cours de laquelle nous avons été appelés à former des groupes et à concevoir un projet en suivant l'un des thèmes préalablement suggérés. Notre équipe, composée de six membres, a opté pour le sujet 2 qui consiste à élaborer un assistant dédié à la restauration et à la prise de commande. Dans cette perspective, un cahier des charges a été mis à notre disposition, définissant des directives cruciales, notamment en matière d'accessibilité, d'interaction vocale, de feedback, d'utilisation d'API, et d'aspect conversationnel.

I. Conception et Architecture globale de l'application

1. Choix du sujet : Assistant dédié à la restauration

Notre projet a vu naissance au sein de cette semaine de travail, prenant la forme d'un voiceBot que nous avons nommé DiscordEat. C'est un chatbot spécialisé dans le domaine de la restauration. Il est destiné à faciliter la prise de commandes et le service via la plateforme Discord. Ses utilisateurs cibles sont des adeptes de Discord ainsi que des amateurs de restauration rapide. L'objectif de cet assistant est d'apporter une assistance pratique pour commander chez Burger King tout en vaquant à d'autres occupations, telles que les jeux vidéo. Le public visé par ce chatbot est donc celui des jeunes adultes, car l'âge moyen des utilisateurs Discord est situé entre 18 et 34 ans.

2. Architecture de l'application globale

Pour l'architecture globale de notre chatbot (Figure 1), nous pouvons la diviser en deux grandes parties :

- Partie RASA : Elle est constituée de RASA et de ses actions. Les deux modules communiquent par des commandes rasa comme '\$ rasa run'. RASA étant une plateforme visant à créer des chatbot intelligents, son utilisation pour créer DiscordEat était indispensable.
- Partie Discord : Lorsque l'on crée un bot Discord, il faut pas mal de modules autres que Discord. En effet, le bot a besoin de passer par des serveurs web pour recevoir des informations ou pour être tout simplement créé.

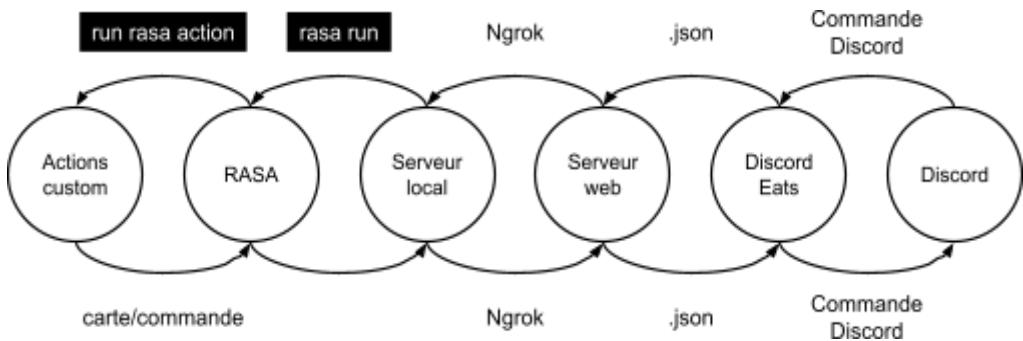


Figure 1 : Architecture globale du logiciel

3. Répartition des tâches

Pour réaliser ce bureau d'étude, nous nous sommes répartis en deux groupes. Un groupe a surtout travaillé sur RASA. Il est composé de Lucas, Tobias et Manon. Ces derniers ont construit tout l'aspect conversationnel du robot ainsi que l'apprentissage des formulations, des synonymes et du vocabulaire qui permet au bot de comprendre l'utilisateur au mieux. Le second groupe est composé de Samuel, Damien et Loan. Ils ont travaillé sur la rédaction du menu, les différentes actions codées en python, le bot discord ainsi que la livraison.

4. RASA

Notre module RASA a été divisé de trois grandes tâches : commande, adresse et finalisation de commande.

| | Intents | Entities | Responses | Actions | lookUpTable | Form |
|----------------|---------|----------|-----------|---------|-------------|------|
| Langage commun | 4 | | 3 | | | |
| Commande | 9 | 5 | 7 | 12 | 4 | 1 |
| Adresse | 1 | 1 | 1 | 1 | | |
| Finalisation | 1 | | 1 | | | |

Tableau 1 : Différentes tâches définies sur RASA

a. Langage commun

| Intents | Fonctions |
|----------|---------------|
| greeting | Saluer |
| goodbye | Prendre congé |
| confirm | Confirmer |
| deny | Infirmer |

Tableau 2 : Liste des intents issus du langage commun

| Réponses | Fonctions |
|----------------|---|
| utter_greet | Saluer en retour, introduit le bot et demande ce que veut le client |
| utter_goodbye | Prendre congé en retour |
| utter_feedback | Confirmer très brièvement ("très bien", "ok") |

Tableau 3 : Liste de réponses issues du langage commun

b. Commande

| Intents | Fonctions |
|-------------------------------------|--|
| ask_info_produit | Demander ce qui est disponible dans une partie de la carte (parmi les burgers par exemple) |
| ask_info_produit_ingredient_with | Obtenir des informations sur des produits avec des aliments spécifiques |
| ask_info_produit_ingredient_without | Obtenir des informations sur des produits sans certains aliments |
| ask_info_type_produit | Demande des informations sur un certain type de produit |
| ask_produit | Demander ce qui est disponible dans la carte |
| order_menu | Commander un menu |
| order_produit | Commander un menu |
| deny_menu | Annuler la commande d'un menu |
| deny_produit | Annuler la commande d'un produit |

Tableau 4 : Liste des intents de la tâche commande

| Entities | Exemples |
|----------------|-------------------------|
| produit | WHOPPER, STEAKHOUSE... |
| accompagnement | FRITES, OIGNON RINGS... |
| boisson | EAU, FANTA... |
| ingredient | TOMATE, FROMAGE... |
| type_produit | BURGER, WRAP... |

Tableau 5 : Liste des entities de la tâche commande

| Actions | Fonctions |
|---------------------------|--|
| action_look_produit | Lancer la recherche d'informations sur un produit par nom |
| action_look_infos | Lancer la recherche d'informations |
| action_look_infos_with | Lancer la recherche de produits avec un certain ingrédient |
| action_look_infos_without | Lancer la recherche de produits sans un certain ingrédient (allergène ou préférence) |
| action_add_menu | Ajouter un menu à la commande |
| action_add_produit | Ajouter un produit à la commande |
| action_confirm_menu | Confirmer le menu |
| action_confirm_produit | Confirmer le produit |
| action_confirm_commande | Confirmer la commande |
| action_effacer_commande | Effacer la commande |
| action_retirer_produit | Retirer un produit de la commande |
| action_retirer_menu | Retirer un menu de la commande |

Tableau 6 : Liste des actions de la tâche commande

| Réponses | Fonctions |
|---------------------------|--|
| utter_ask_another_produit | Demander à l'utilisateur si autre chose doit être commandé |
| utter_ask_menu | Proposer à l'utilisateur s'il souhaite le |

| | |
|--------------------------|---|
| | format menu |
| utter_ask_produit | Demander à l'utilisateur le produit souhaité |
| utter_ask_accompagnement | Demander à l'utilisateur l'accompagnement souhaité |
| utter_ask_boisson | Demander à l'utilisateur la boisson souhaitée |
| utter_ask | Attendre le souhait de l'utilisateur |
| utter_help | Fournir des informations pour aider et guider l'utilisateur |

Tableau 7 : Liste des réponses de la tâche commande

De plus, pour la tâche “commande”, nous avons été amenés à exploiter des fonctionnalités de RASA.

i. Lookup table

Tout d'abord, nous nous sommes penchés sur l'utilisation de “Lookup table”.

Ces dernières permettent de généraliser une phrase d'intent, à plusieurs entités pouvant être trouvées dans la phrase.

En effet, dans notre cas, nous avions une multitude de produits disponibles ; lesquels représentent l'entité à trouver dans la phrase. Pour que RASA soit capable de trouver le produit souhaité, ça signifie que nous aurions dû avoir une multitude d'exemples de phrases, du type :

- Je souhaiterais commander un X₁
- Je souhaiterais commander un X₂
- ...
- Je souhaiterais commander un X_n

Avec “n” représentant le nombre de produits disponibles. On comprend très vite que ça devient trop volumineux et que ce n'est pas optimal comme solution.

La lookup table nous a permis de créer une table, répertoriant l'ensemble des “n” produits, pour généraliser le tout à une seule phrase :

- Je souhaiterais commander un X

Avec X pouvant être n'importe lequel des produit commandé.

Au final on a qu'une seule phrase qui permet de traiter l'ensemble des cas.

| LookUpTables | Exemples |
|----------------|-------------------------|
| produit | WHOPPER, STEAKHOUSE... |
| accompagnement | FRITES, OIGNON RINGS... |
| boisson | EAU, FANTA... |
| type_produit | BURGER, WRAP... |

Tableau 8 : Liste des lookup de la tâche commande

ii. Formulaire

Ensute, lorsqu'un utilisateur passe une commande, il a bien sûr la possibilité de prendre un menu et donc d'indiquer en une seule interaction : le produit, l'accompagnement, la boisson. Lors de cette interaction, on a de ce fait plusieurs entities à prendre en compte et le formulaire nous permet alors de remplir plusieurs slots en même temps lorsque plusieurs informations sont données en même temps.

| Form | Fonction |
|-----------|----------------------------------|
| menu_form | Formulaire de commande d'un menu |

Tableau 9 : Formulaire de la tâche commande

c. Adresse

| Intent | Fonction |
|--------------|--|
| give_adresse | Renseigner des informations de livraison |

Tableau 10 : Intent de la tâche adresse

| Actions | Fonctions |
|-----------------------|---|
| action_generer_trajet | Lancer la génération du trajet par rapport aux renseignements fournis |

Tableau 11 : Action de la tâche adresse

| Réponses | Fonctions |
|-------------------|--------------------|
| utter_ask_adresse | Demander d'adresse |

Tableau 12 : Réponse de la tâche adresse

d. Finalisation de la commande

| Intent | Fonction |
|----------------|--|
| finalize_order | L'utilisateur indique qu'il ne souhaite rien commander de plus et souhaite finaliser l'interaction |

Tableau 13 : Intent de la tâche finalisation de la commande

| Réponses | Fonctions |
|--------------|------------------------------------|
| utter_thanks | Remercier et confirmer la commande |

Tableau 14 : Réponse de la tâche finalisation de la commande

5. Discord

Comme expliqué précédemment, ce chatbot rasa a pour but d'être directement interfacé à l'application Discord. Par chance, cette dernière propose de nombreux outils aux développeurs tels que la création de bots personnels.

a. Création du bot discord

La première étape a tout d'abord été de créer le squelette de notre bot directement sur le site [Discord Developer Platform](#). On a pu notamment y définir son nom, son icône et surtout toutes les autorisations d'interactions qu'on voulait laisser. Cette étape de création est assez rapide et c'est suite à celle-ci qu'on reçoit le **token** qui permet de mettre en marche notre bot (à garder secret).

b. Développement des commandes de base

La deuxième étape a été de créer les possibilités d'interaction entre le bot et les utilisateurs de discord. Nous avons donc créé un fichier python permettant de mettre en marche le bot (grâce au token) et de créer le comportement de ce dernier. Nous avons pour cela utilisé la librairie discord.py qui est une enveloppe d'API dont la documentation est disponible directement sur internet : [discord.py Documentation](#)

Par convention, les utilisateurs communiquent avec les bots discords par des commandes précédées de préfixes. Les préfixes permettent aux utilisateurs de ne pas confondre les bots entre eux si jamais deux bots présents sur un même serveur proposent la même commande. Nous avons décidé d'avoir par défaut le préfixe `!`. Nous avons également codé la possibilité (pour les personnes ayant un grade administrateur sur le serveur) de pouvoir changer ce préfixe. Cela permet de donner aux admins une possibilité de gestion de l'ergonomie du serveur. Voici donc les deux commandes liées à la gestion des préfixes :



Figure 2 : Définition du préfixe du bot

Une autre commande permettant une meilleure expérience utilisateur est la commande '`help`' qui permet à l'utilisateur d'avoir une vue globale des interactions qu'il peut avoir avec le bot :

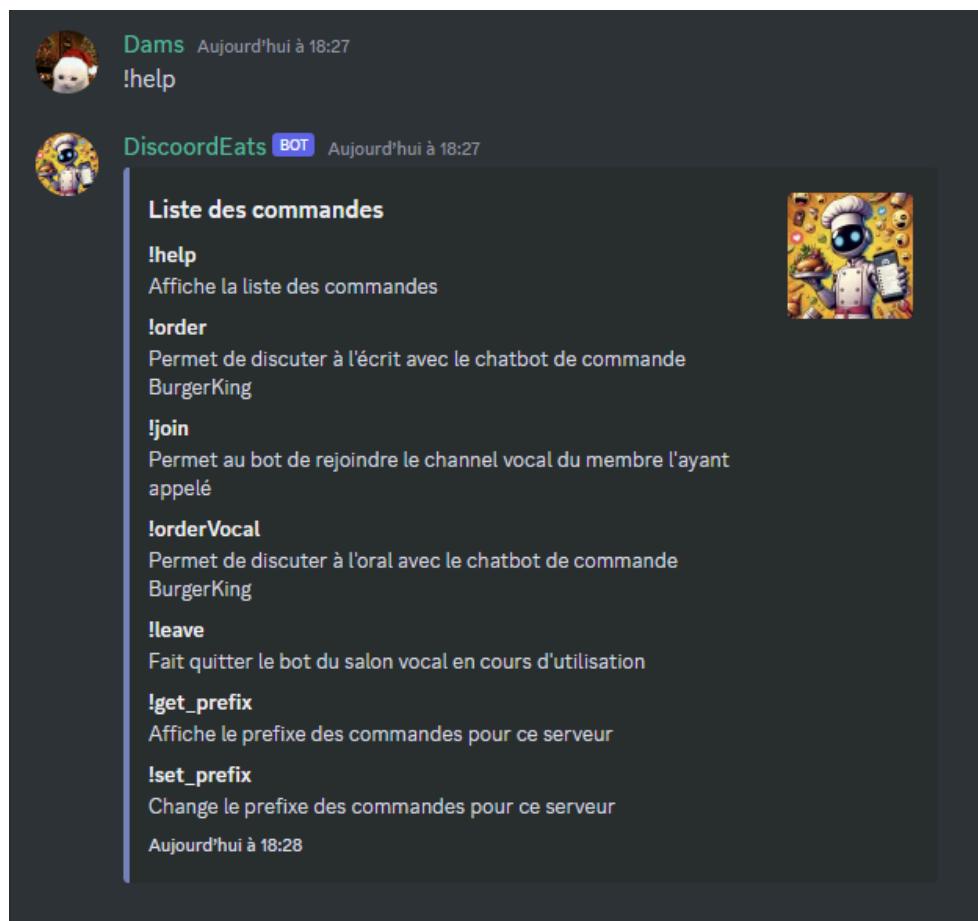


Figure 3 : Listes des commandes que l'on peut adresser au bot

c. Communication bot-discord / rasa

Pour communiquer avec le chatbot de commande burger King, nous avons dû mettre en place la commande '*order*' permettant de passer commande. Pour l'utiliser, Il faut écrire préfixe+*order* suivi du message destiné au chatbot rasa. Le bot discord va alors récupérer le contenu du message, le mettre au format json puis va l'envoyer grâce à la librairie requests sur le serveur ngrok qui fait la passerelle avec notre serveur local rasa. Rasa va récupérer le message et envoyer la réponse sur le serveur ngrok, duquel nous récupérerons le contenu au format json. Il ne reste alors plus qu'à afficher le message de réponse sur Discord, à la suite du message de l'utilisateur.

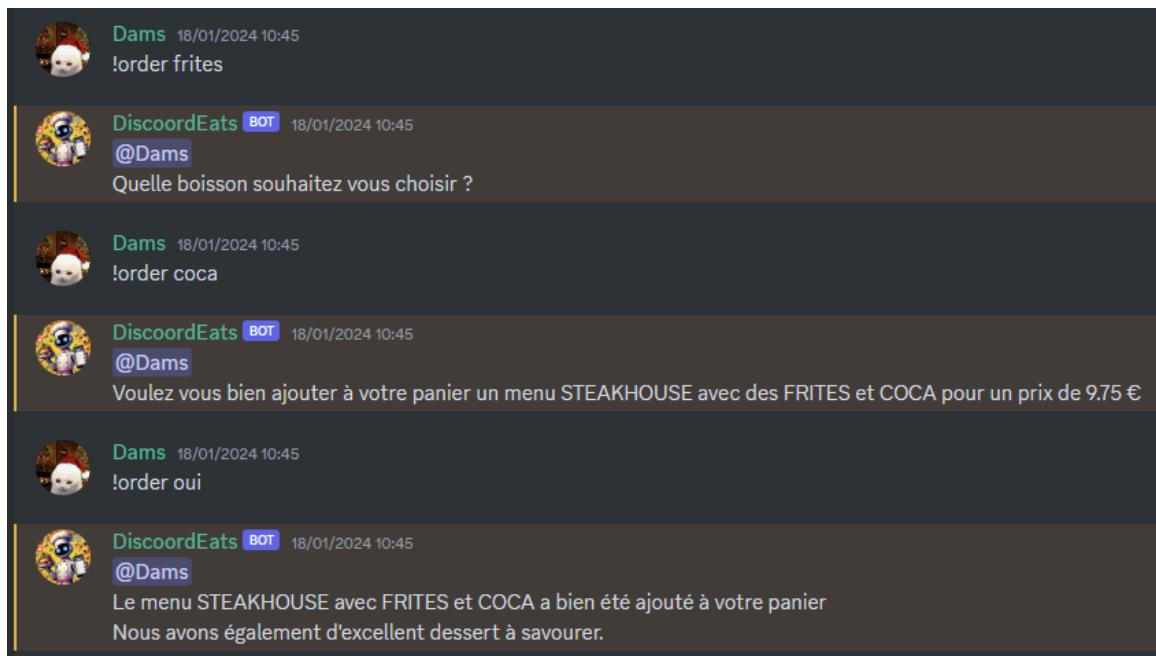


Figure 4 : Exemple d'une discussion par écrit avec le bot

d. Intégration de la voix et de l'ouïe

Enfin, pour satisfaire les enjeux d'accessibilité du bot discord, nous avons travaillé sur des commandes permettant d'interagir à l'oral avec lui. Les commandes ‘join’ et ‘leave’ permettent respectivement au bot discord de rejoindre et de quitter le salon vocal dans lequel se situe l'utilisateur ayant fait la commande. Une fois ayant rejoint le salon vocal, le bot énoncera à l'oral tout ce que le chatbot rasa renverra comme message. Cette interaction est possible grâce à la librairie gTTS de Google qui permet de faire du text to speech de manière très rapide. On donne en entrée de gTTS un texte (String) et on récupère en sortie un fichier.mp4 que l'on diffuse ensuite avec le bot discord.

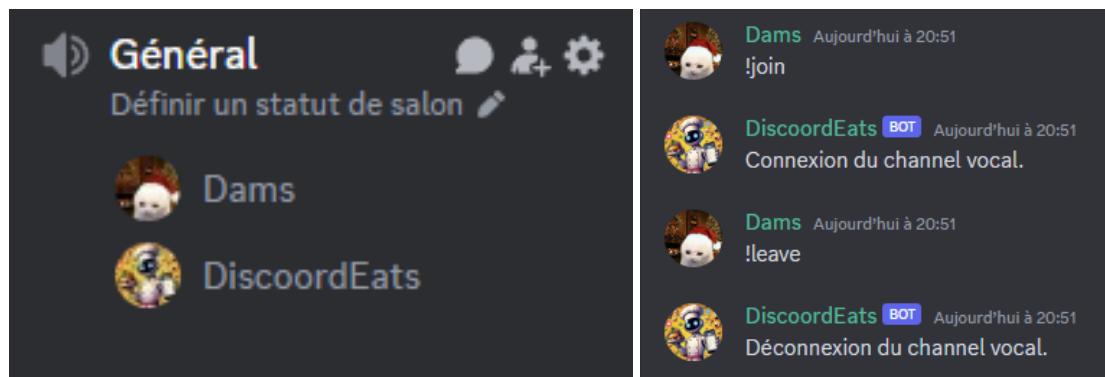


Figure 5 : Comment discuter avec le bot

Concernant l'ouïe, la version python de discord ne permet pas encore à un bot d'enregistrer le contenu d'un salon vocal. Nous avons donc dû trouver une solution alternative pour

mettre en place la reconnaissance vocale. La commande orderVocal permet de lancer en parallèle un script local qui enregistre le son environnant et qui le transforme en texte (grâce à la librairie python speech_recognition). Le bot discord retransmet alors le texte que le script a compris, l'envoi à Rasa (à l'instar de la commande order) et retransmet la réponse dans le chat. Cette solution reste un système de fortune, car la reconnaissance fonctionne donc seulement sur le pc où est hébergé le bot discord. En utilisant la commande orderVocal les utilisateurs lancent un script local sur le pc hébergeur qui va capturer le signal sonore environnant. Cette solution marche pour nous puisque le pc qui héberge le bot est le nôtre. Cependant, des utilisateurs du monde entier ne pourraient pas utiliser la reconnaissance vocale, parce qu'ils ne seraient pas entendus par le pc hébergeur du script. Une solution viable serait de recoder le bot discord en nodeJs pour pouvoir avoir accès à l'enregistrement vocal des utilisateurs dans les salons vocaux. Dans le cadre de notre projet, cette solution n'était pas envisageable parce que cela aurait été trop chronophage suivant l'avancement.



Figure 6 : Exemple d'une communication vocale avec le bot sur Discord

6. La Carte

Pour réaliser notre carte de produit, nous nous sommes appuyés sur la gamme de chez Burger King. Nous avons pensé que, disposant d'une carte déjà pré-faite, variée et complète, nous gagnerions du temps. Ce fut une erreur. Le fait que la carte soit construite de manière à simplifier le marketing de la marque a compliqué l'implémentation sur RASA. Prenons par exemple pour différencier des burgers avec des noms similaires (ex : Whopper, Double whopper, Whopper cheese & Bacon, Double whopper Cheese), ou encore, reconnaître lorsque l'utilisateur parle de salade comme ingrédient (laitue, roquette, ...) et non pas comme un type de produit.

La seconde erreur, c'est que nous avons conservé un niveau de détails "trop" important dans notre carte. Allant jusqu'à exposer le grammage de sauce dans les burgers.

```
"DOUBLE WHOPPER": {
    "type": "BURGER",
    "prix_produit": 6.75,
    "prix_menu": 8.75,
    "composition": {
        "BUN WHOPPER 5": 2,
        "TOMATE": 2,
        "RONDELLE OIGNONS": 1,
        "CORNICHON": 4,
        "PATTY WHOPPER": 2
    },
    "sauce": {
        "MAYONNAISE": 21,
        "KETCHUP": 14
    },
    "accompagnements": {
        "LAITUE": 21
    }
},
```

a. Transcrire et manipuler la carte

Pour que le bot puisse interagir avec la carte, nous avons fait le choix de l'implémenter sous un format JSON. Puis coder un module python permettant de la lire, d'effectuer des recherches de produit selon différents critères (avec/sans ingrédients, budget, par nom, par type...).

Vous pouvez retrouver les fichiers *discordEats.json* et *lire_carte.py* dans le dossier /ChatbotRasa.

b. Gérer le panier et la commande

Similairement au système pour gérer la carte, nous avons créé : un fichier *commande.json* pour sauvegarder les commandes en cours et un module *gestion_menu.py* pour interagir avec le fichier *commande.json* et gérer le panier. Lorsqu'une commande est validée par le chatbot, le fichier est vidé dans un .txt portant le numéro de la commande. Ce mécanisme est pensé de manière à ce qu'une application tierce puisse accéder à la commande.

Figure 7 : Extrait de Carte

On aurait pu imaginer y intégrer d'autres informations telles que l'adresse, le nom du client etc et que le fichier soit utilisé par le livreur.

c. Lien au chatbot

Le chatbot interagit avec la carte et le panier en utilisant les modules python depuis des actions personnalisées. Il n'y a pas besoin d'installer de package ou autres. Directement dans les codes des actions, le chatbot génère des phrases de réponse exhaustives et gère les accords syntaxiques.

7. Crédit du trajet

Une fois sa commande passée, le client arrive à l'étape où il doit renseigner son adresse pour la livraison. Lorsqu'il donne une adresse, l'action personnalisée *générerTrajet.py* se lance et récupère le slot correspondant à l'adresse donnée. Une première étape de

géocodage s'effectue alors et transforme le texte de l'adresse en des coordonnées GPS (latitude, longitude). La seconde étape consiste ensuite à faire appel à une API qui trace le trajet entre les coordonnées GPS de l'utilisateur et les coordonnées GPS du burger King. On enregistre le trajet renvoyé sur une carte sous le format HTML. Le bot discord n'a alors plus qu'à envoyer à l'utilisateur la map.html pour qu'il puisse visualiser le trajet.

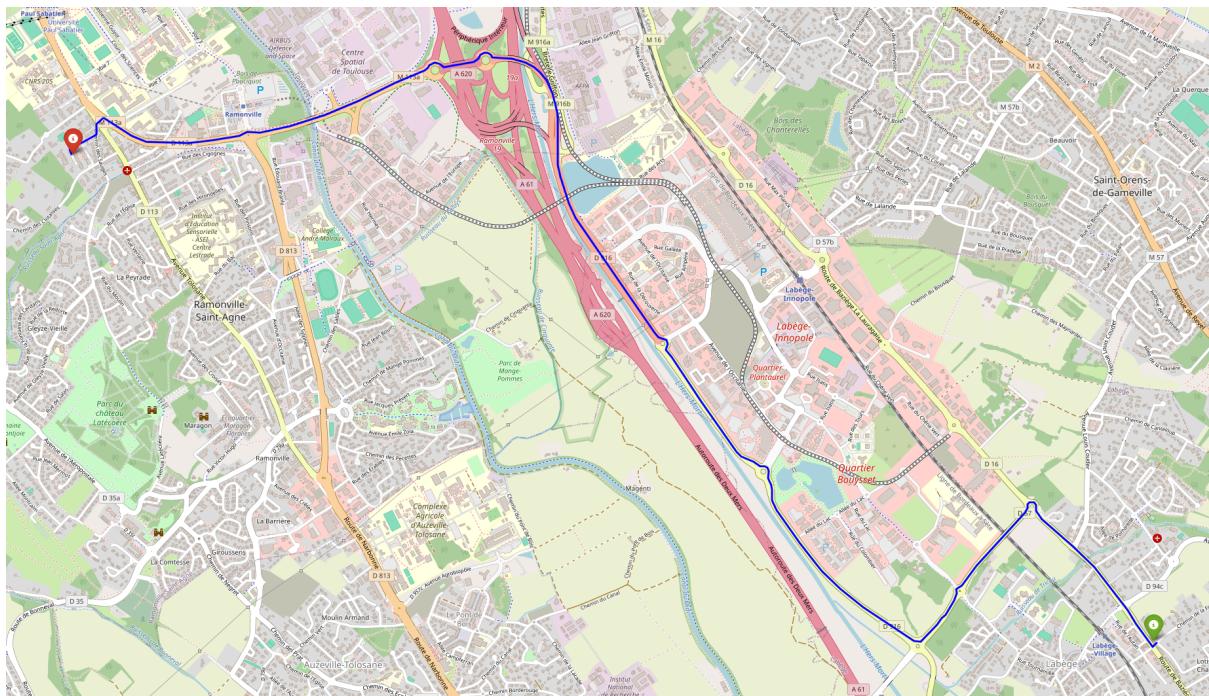


Figure 8 : Exemple d'une livraison

II. Traitement de l'accessibilité (point d'attention, démarche...)

1. Points d'attention

Pour tout projet mettant en avant l'interaction avec l'utilisateur afin de lui offrir un service, quel qu'il soit, l'accessibilité du système doit être le point central du développement de toutes les interfaces.

En effet, un système réussi se doit d'être au point en termes d'accessibilité et de facilité de prise en main.

Lors de la phase de conception de notre système, notre attention s'est portée sur cet aspect afin de rendre notre assistant de commande plus attrayant. Dans un souci de complétude, nous avons fait le choix de fournir différentes options de communication à l'utilisateur. Tout d'abord en implémentant la discussion par écrit via les commandes Discord, vastement utilisées et reconnues par les habitués de la plateforme.

Ensuite, l'étape suivante évidente était de mettre en place la discussion vocale pour offrir une seconde option plus ergonomique et plus efficace temporellement.

Enfin, notre système étant basé sur les normes de l'agroalimentaire, cela pose d'autres questions en termes d'accessibilité, afin d'offrir une utilité de notre système à tous les profils. Notre carte propose donc un nombre très varié d'éléments, pour ravir toutes les papilles.

2. Démarche

Afin d'évaluer et de mettre en place notre système et son accessibilité, nous avons travaillé de manière itérative. Une fois le modèle de dialogue complet et fonctionnel, nous avons décidé d'implémenter son utilisation par écrit via Discord, puis de le tester par nous-mêmes pour déterminer si ce mode d'utilisation était convenable et efficace.

Une fois cette première étape validée, il a fallu l'adapter pour intégrer la communication vocale, et faire que celle-ci puisse se fondre et se confondre avec la première.

Enfin, en testant finalement le système global, en essayant tout type de produits, et de toutes les manières différentes, cela nous a permis de détecter les points à améliorer pour offrir une meilleure accessibilité, ainsi que les points d'impact principaux sur l'expérience utilisateur.

3. Impact sur l'expérience utilisateur

L'accessibilité de notre système n'est pas parfaite, et présente quelques défauts qui pourraient être réglés dans des prochaines mises à jour.

Premièrement, la diversité, voulue, de la carte des produits, entraîne une certaine rigueur en termes d'orthographe que l'utilisateur doit respecter afin que le système ne se trompe pas : certains produits ayant des noms proches. Cela concerne donc la communication par écrit.

Pour la communication orale, le fonctionnement du système implique de s'annoncer au bot à chaque fois que l'on souhaite prendre la parole pour lui parler, au moyen d'une commande Discord. Cela induit un léger délai entre le moment où l'utilisateur veut parler, et celui où le bot est prêt à l'écouter. Cela entraîne aussi une certaine saccade dans le rythme de la discussion si celle-ci est réalisée à l'oral uniquement.

Toutefois, l'utilisateur peut à tout moment passer de la communication à l'écrit à celle à l'oral, en fonction de ses préférences et contraintes, et ce même en plein milieu de discussion.

III. Évaluation du chatBot (performance et ergonomie)

1. Première évaluation

Durant toute la durée de développement de ce chatbot, nous avons effectué plusieurs évaluations afin de tester l'avancement de notre code. Ces dernières nous ont permis de voir les points à améliorer ainsi que ce qu'il fallait conserver comme tels. Il faut aussi noter que toutes les premières évaluations ont porté sur les intents, les entités ainsi que les stories. Mais malgré le fait que toutes les évaluations soient bonnes, on se rend compte que parfois le bot accepte des cas de figure qui ne sont pas corrects, surtout lors de la prise de commande de menu.

2. Mesures prises pour optimiser la performance

Tout au long du processus de développement de ce chatbot, nous avons procédé à plusieurs évaluations pour tester la progression de notre code. Ces évaluations ont été cruciales pour identifier les aspects à améliorer et ceux à maintenir inchangés. Il est important de souligner que les premières évaluations se sont principalement concentrées sur les intents, les entités et les stories. Cependant, malgré des résultats globalement positifs, nous avons constaté que parfois le chatbot accepte des scénarios qui ne sont pas corrects, en particulier lors de la prise de commande de menus. Ce constat nous pousse à affiner davantage le comportement du bot afin de garantir une interaction plus précise et conforme aux attentes, surtout dans des situations aussi cruciales que la commande de repas.

a. Évaluation d'intents

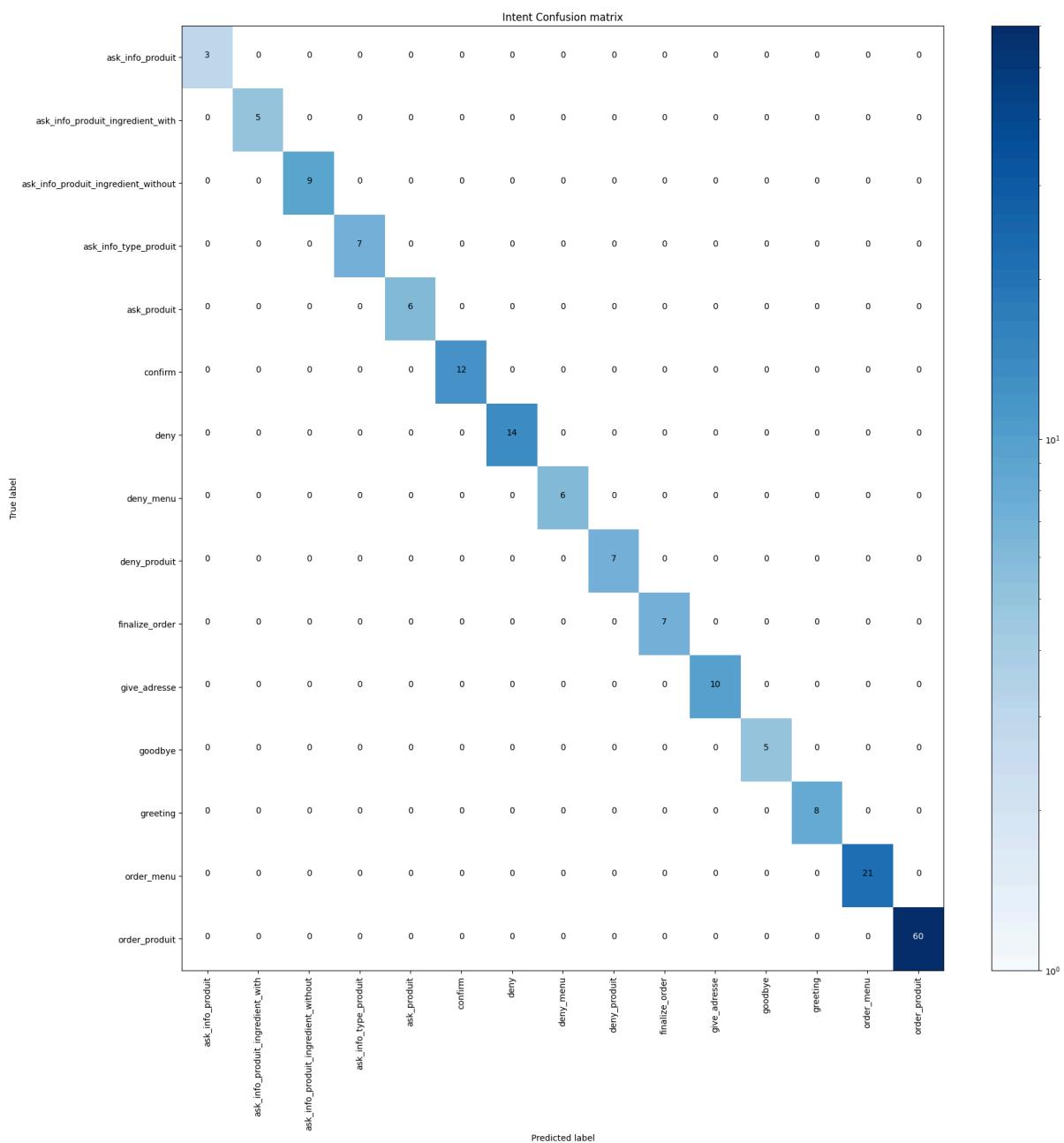


Figure 9 : Matrice de confusion des intents

À la suite de la dernière évaluation de notre voiceBot, nous avons obtenu cette matrice de confusion pour les intents. Comme on peut le voir sur la Figure 9, il n'y a aucune erreur de reconnaissance et tous sont testés un bon nombre de fois. Ceux qui sont peu testés sont ceux qui ne sont pas souvent utilisés.

b. Évaluation d'entités

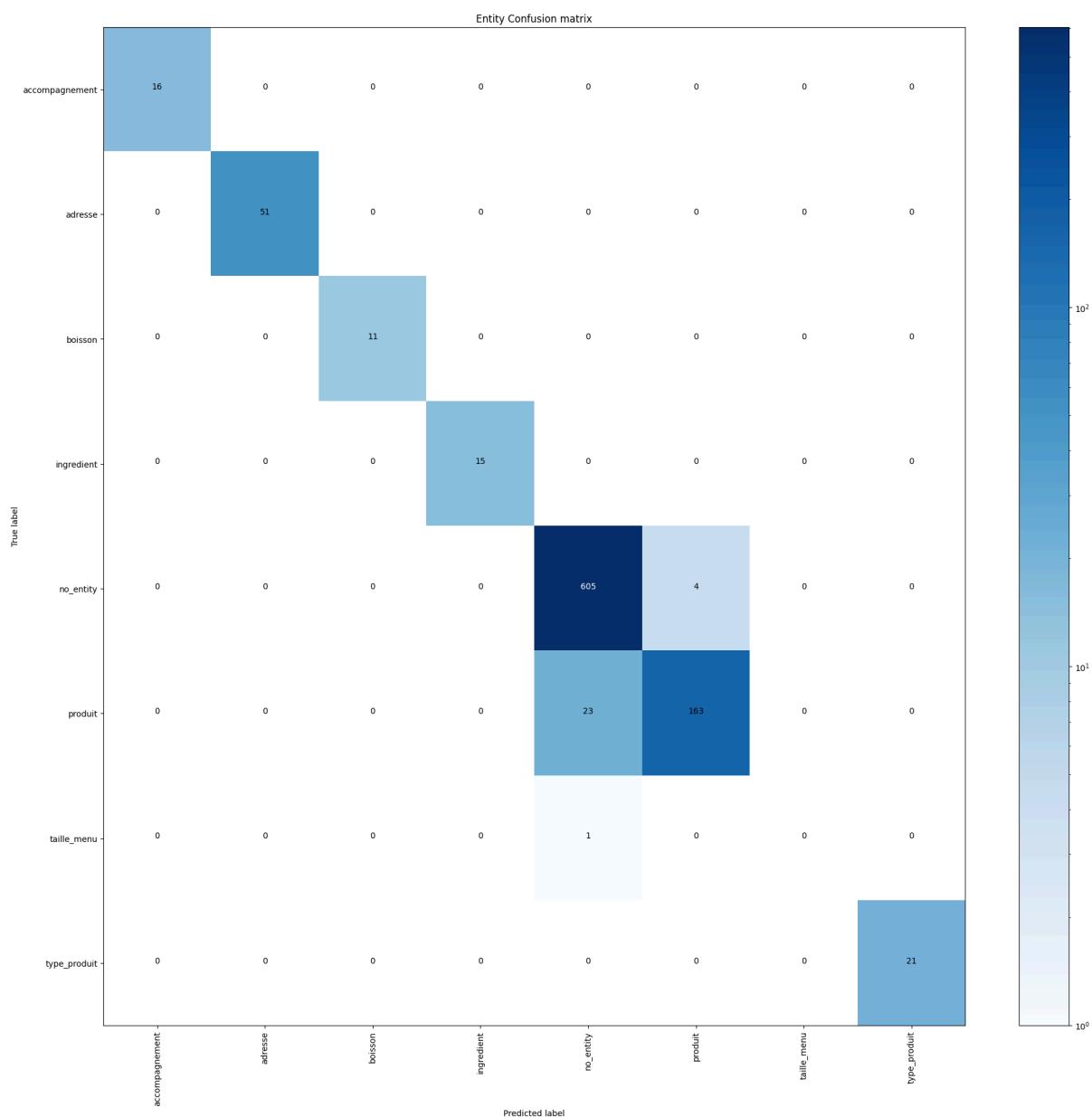


Figure 10 : Matrice de confusion des entités

Suite à la récente évaluation de notre voiceBot, la matrice de confusion des entités a été générée, comme illustré dans la Figure 10. Bien que le bot fonctionne globalement de manière excellente, quelques erreurs ont été identifiées. Tout d'abord, ‘no_entity’ a été associé à ‘produit’ vingt-trois fois en raison de la formulation de la phrase, mais cette confusion a été corrigée dans la suite de la story testée. De plus, l'entité ‘produit’ a également été parfois confondue avec ‘no_entity’ à quatre reprises. Dans ces situations, l'utilisateur peut simplement répéter sa commande, et le bot la comprendra correctement. En ce qui concerne l'entité ‘taille_menu’, elle a été moins testée lors de ces évaluations, ce qui

explique un taux d'erreur plus élevé. Malgré ces quelques confusions, notre bot demeure performant et capable de s'adapter aux formulations variées des utilisateurs.

c. Évaluation d'actions

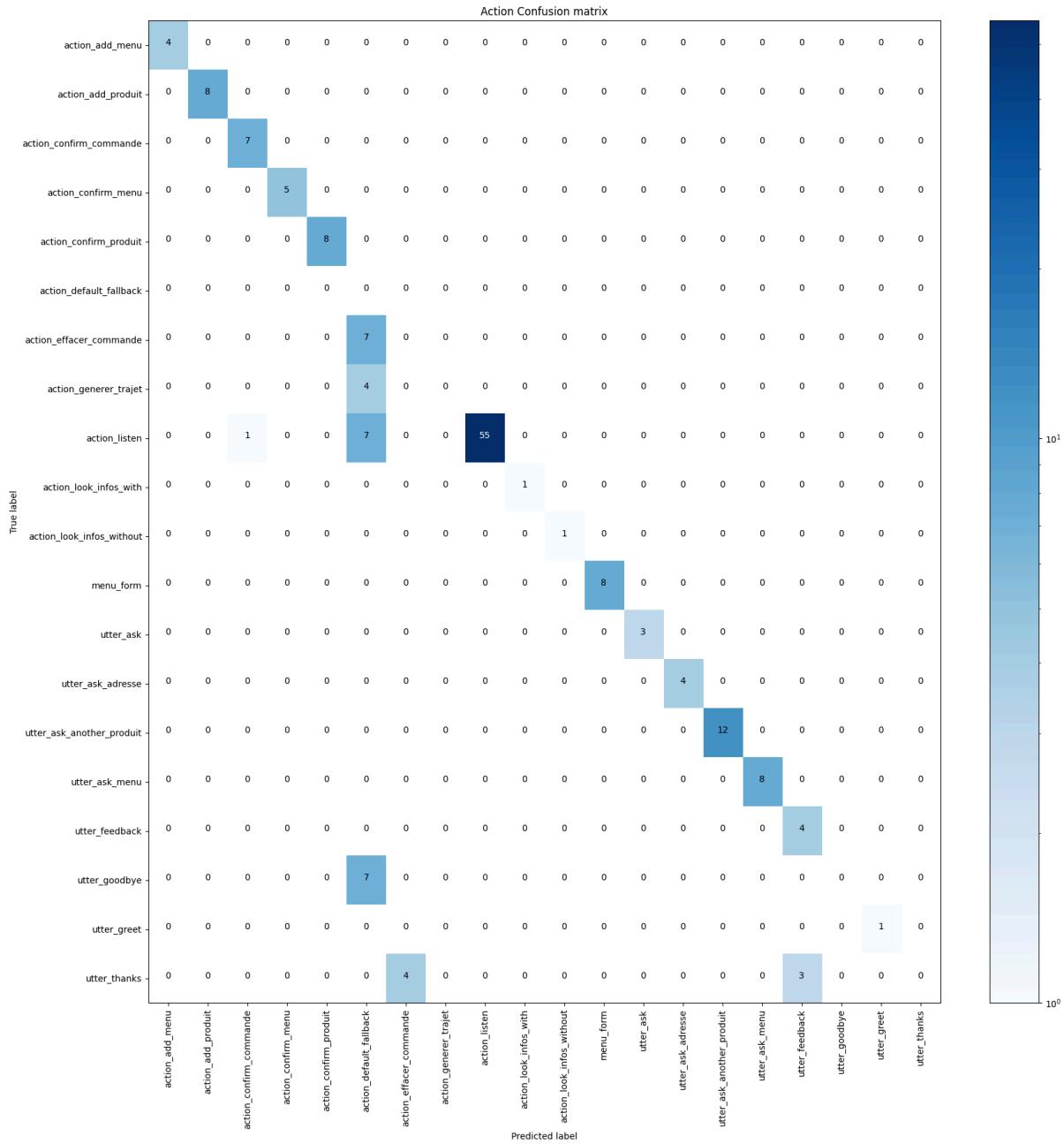


Figure 11 : Matrice de confusion des actions

À la suite de la dernière évaluation de notre voiceBot, nous avons obtenu la matrice de confusion des actions illustrée dans la Figure 11. Bien que la clarté de cette matrice puisse sembler altérée, cela résulte principalement de la complexité de nos règles. En effet, nous avons élaboré un grand nombre de règles, car une construction de commande robuste est essentielle. En analysant, par exemple, l'action qui présente le plus d'erreurs de

reconnaissance, 'action_default_feedback', nous constatons qu'elle a été parfois remplacée par 'utter_thanks', 'actions_listen', et 'action_générer_trajet'. Les seuls cas problématiques surviennent lorsqu'elle est remplacée par 'action_listen', indiquant que le système est en mode écoute au lieu de fournir un retour. Dans d'autres cas, les erreurs se résument souvent à des demandes d'utilisateurs en violation des règles du chatbot. Les détails de ces erreurs sont vérifiables dans les retours complets d'évaluations. En somme, bien que la matrice de confusion des actions puisse manquer de clarté, le système fonctionne efficacement malgré ces quelques nuances.

3. Ergonomie et convivialité de DiscordEat

Dans le souci de garantir une accessibilité optimale, nous avons délibérément intégré un langage accessible à tous, incluant même les fautes d'orthographe courantes. Cette approche vise à éliminer toute barrière linguistique et à rendre l'interaction avec notre chatbot aussi naturelle que possible. De plus, conscient du public majoritairement composé de jeunes adultes, nous avons doté notre chatbot d'un langage courant et décontracté. Cette convivialité dans la communication vise à établir une connexion plus authentique avec les utilisateurs, facilitant ainsi l'expérience de commande en ligne et rendant l'interaction avec DiscordEat aussi plaisante que possible. En intégrant également des touches d'humour inspirées de l'esprit décontracté de Burger King, nous cherchons à offrir une expérience conviviale et engageante à notre communauté d'utilisateurs.

Bilan

1. Bilan technique

Sur le point technique, nous sommes fiers du travail que nous avons accompli. Nous avons exploré une multitude de fonctionnalités de Rasa (lookUpTable, Formulaires...) tout en ayant la plus grande base d'intent, d'actions et de réponses. Nous avons doté le bot d'un "comportement commercial", permettant d'inciter à la consommation et en étant relativement flexible sur la manière de recevoir les informations. Il y a cependant des pistes d'améliorations : pouvoir commander plusieurs produits en même temps (ex : "Trois whopper", "Un steakhouse sans menu et un king fusion Nutella"...), gérer les cas de refus de donner l'adresse. Nous sommes également ravis de la partie intégration du chatbot dans Discord, car nous avons une ergonomie finale très satisfaisante. Le bot répond sans aucun délai, il propose aux administrateurs des options de personnalisation et aux utilisateurs différentes modalités d'interactions (commander à l'écrit, commander à l'oral, avoir un retour auditif). À travers Discord, nous avons même pu accomplir des tâches qui n'étaient pas possibles directement sur l'interface de Rasa X. Par exemple, nous avons réussi à envoyer des fichiers HTML pour afficher le trajet de livraison.

Pour conclure, le temps nous a manqué pour améliorer la robustesse du chatbot à reconnaître et différencier tous les produits et implémenter d'autres mécanismes (tels que le paiement, un formulaire d'adresse, etc.). En revanche, il dispose d'un niveau de détails que l'on trouve vraiment satisfaisant.

2. Bilan humain

Ce projet a nécessité une organisation rigoureuse pour réussir. Heureusement, nous avons réussi à coordonner efficacement nos travaux. Chacun a pris en charge au moins une tâche essentielle, donnant à chacun des responsabilités lors de ce projet. Cela a notamment permis à tout le monde d'approfondir leur compréhension des problématiques liées à la création d'un chatbot et donc, de mieux en comprendre les enjeux. De plus, nous avons bien réagi à la contrainte de temps, en adaptant nos objectifs une fois que nous avons compris que nous ne pourrions pas réaliser tout ce que nous souhaitions.

En conclusion, le projet a bien été mené et fut très instructif pour nous tous.

Annexe

Le lien vers le dépôt GitHub de DiscordEat est disponible ici :

<https://github.com/DESQL-Corporation/discordEats>

Comme demandé dans la consigne, vous y trouverez l'intégralité de notre code. Nous vous encourageons à prendre le temps d'explorer ce dépôt GitHub pour une compréhension approfondie de notre approche.