UNIVERSITÉ
CAEN
NORMANDIE

Master Informatique
Internet, Données et Connaissances

Mémoire de stage

Systèmes NLP et NLU en entreprise

Le traitement du langage naturel au service de l'utilisateur

Clément Vétillard

Tuteur de stage : **Marc Spaniol**
Jury : **François Rioult**

Entreprise d'accueil : KMB Labs
Maitre de stage : **Paul Leménager**
Année universitaire : 2020 / 2021

Contents

Résumé

Résumé en français

Abstract

English resume

Introduction

KMB Labs[1] est une entreprise de développement web spécialisé dans les assistants conversationnels et moteurs de recherches. L'équipe propose à ses clients d'intégrer, au choix, un moteur de recherche spécifique au site client ainsi qu'un assistant chatbot.

Avec en tout 6 employés, KMB Labs possède à son actif plusieurs projets menés à bien dont la plupart sont connus du grand public, tant dans l'immobilier (*Guy Hoquet*, *Century 21*) que dans le recrutement (*Adecco*, *Carrefour*, *La Poste*).

Le domaine des entreprises visées par l'expertise de KMB Labs tant à se diversifier afin d'accueillir des chatbots spécialisés dans d'autres domaines (comme des chatbot interne à une entreprise mise à disposition pour ses employés).

> "*Nous mettons en relation la dernière génération d'algorithmes de matching avec notre technologie d'analyse texte - et tout cela en une fraction de seconde.*"
> **Vu sur kmblabs.com**

Lors de ce stage, plusieurs missions m'ont été confiées afin de participer à la mise en place de différentes fonctionnalités afin d'améliorer les applications déjà en place. Après une présentation du contexte et des différentes parties composant les services fournies par KMB Labs, je m'efforcerai de détailler mon implication dans chacun des projets auquels j'ai pu participer pendant toute la durée de mon stage.

[1] Anciennement Kick My Bot

1 Contexte

Notre époque connait un nombre croissant d'entreprises choisissant de réaliser une vitrine sur le web et d'y proposer des services via celui-ci. Certains site peuvent devenir une structure conséquente de données, aussi bien dû aux nombreux services proposés qu'aux informations dont ils disposent. Même si certaines pratiques, commes les foires aux questions, tandent à regrouper les questions les plus fréquentes des utilisateurs, cela ne saurait regrouper efficacement toutes les informations d'un site.

Pour remédier à cette problématiques, plusieurs approches ont été utilisées. Les barres de recherches, indexant le contenu des différentes pages ainsi que des fenêtres permettant de communiquer avec un support sous la forme d'une discussion.

Cependant, la première approche n'est pas forcément intuitive pour les plus néophites (là où on recherche une certaine information, il faudra choisir les bons mot-clefs pour obtenir le résultat escompté). La deuxième solution nécessite du personnel pour répondre aux questions des utilisateurs.

Avec l'émergence des systèmes de NLP[1] chacune de ses deux approchent ont pu grandement évoluée. Ces outils permettent d'avoir une compréhension plus profonde de ce que recherche l'utilisateur afin de l'aiguiller vers le contenu qu'il recherche avec une plus grande facilité.

L'objectif ici est de permettre à l'utilisateur de faire des requêtes à une interface en formant des phrases ou bien en parlant via un micro. Cette solution permet d'avoir une solution conviviale pour répondre aux questions de l'utilisateur et de pouvoir le rediriger vers le contenu souhaité rapidement, sans mobiliser de personnel.

1.1 Les intentions

Les différentes parties de se rapport seront étroitement liées aux intentions. Les différents services proposés par KMB Labs cherche à détecter l'intention de l'utilisateur pour lui fournir la réponse la plus appropriée.

[1] Natural Language Processing

Ainsi, les différentes entités grammaticales[2] nécessaires à la réalisation de l'intention sont extraites. De fait, une utilisateur cherchant à contacter une personne de l'entreprise pourra formuler sa demande de plusieurs façons :

- "je peux joindre quelqu'un de chez vous ?"
- "vous avez un numéro de téléphone ?"
- "j'ai besoin de vous contacter, je fais comment ?"
- etc...

Toutes ses manières, et bien d'autres encore, permettent de rediriger vers une page précise contenant les informations, ou bien de donner les informations nécessaire à la prise de contact. L'important est que l'*intention* de l'utilisateur pour toutes ces phrases est la même.

[2]Sujet, action, etc...

2 KMB Labs

Afin de procurer ses services, KMB Labs a développé plusieurs applications et services web. Faisons une courte présentation de chacun de ses projets afin d'appréhender l'organisation de ceux-ci.

2.1 Les interfaces

2.1.1 Le chatbot

Le chatbot se présente comme une fenêtre de discussion, on peut y converser avec un assistant virtuel qui répondre à nos questions. En parlant naturellement via l'interface, l'intention de l'utilisateur y est détectée et une réponse lui ai donné.

Ladite réponse est préalablement fournie par le client afin de choisir quelles interrogations sont prises en charge, ou non, par l'assistant virtuel.

La fenêtre peut-être agrémentée d'un onglet avec des questions pré-établies, un peu à la manière des foires aux questions, d'actions rapide où l'utilisateur n'a qu'à cliquer dessus pour obtenir le résultat rapidement.

2.1.2 L'interface conversationnelle

L'interface conversation prend la forme d'une barre de recherche sur une page complète. L'utilisateur peut décrire ce qu'il recherche (caractéristiques d'un appartement, les domaines de compétences d'un emploi, etc...) et le moteur se charge de :

1. voir si une intention est détectée (à la manière du chatbot)
2. effectuer la recherche en appliquant les filtres trouvés la requête (localisation, salaire, loyer, etc...)

Cette interface se présente plus comme un moteur de recherche, mais l'intégration des systèmes NLP permettent une certaines flexibilités quant à la forme de la demande

de l'utilisateur. Elle simplifie également l'utilisation, là où différents filtres (toujours existants) devaient être rempli manuellement, l'interface ici va remplir les filtres correspondant aux critères de recherche.

2.2 Le Backoffice

Le Backoffice est une application web permettant au client de paramétrer les réponses qui seront fournies à ses utilisateurs et de s'informer du bon fonctionnement du chatbot.

Dans cette optique, plusieurs pages sont disponibles. En voici une présentation non exhausitve :

- l'éditeur : permet de définir les réponses du chatbot pour chaque intention disponible
- les statistiques : regroupent des graphiques visant à informer le client du taux d'utilisation et de compréhension de sa solution conversationnelle fournie par KMB Labs
- ML[1] Monitor : cette partie, pour utilisateur averti uniquement, permet de tester la compréhension d'une phrase et, si besoin, d'entrainer un modèle avec des phrases non comprises qui aurait dû l'être
- le Lab fourni une interface simple permettant de voir la configuration de la fenêtre du chatbot telle qu'elle serait dans le site de destination mis à part l'environnement graphique

2.3 L'API

L'API a été écrite avec le langage GraphQL avec le service AppSync de Amazon Web Service[2]. Cette approche permet de ne récupérer que les données nécessaires à l'usage, et ceux même si la structure et complexe.

AppSync donne la possibilité de lier le schéma à des *resolvers*, qui sont en charge de retourner la donnée voulu. Ce procédé permet une grande flexibilité, les résolvers peuvent être associé à :

- des requêtes directes aux bases de données (via le langage VTL)

[1] Machine Learning
[2] Abrégé "AWS" ci-après

- des *Lambda* qui sont des fonctions dans le langage désiré (Node.js, Python, Go, Java, etc...[3])

La possibilité de pouvoir lier un résolver à une fonction permet d'y effectuer plusieurs traitement avec que les données ne soient retournées à l'application cliente. Conclusion, cela allège la complexité des applications utilisant les lambdas dans leur cycle de vie, ou plus précisément cela reporte une partie de la charge sur les serveurs d'AWS.

2.4 Le Framework

Le Framework est la base de chaque serveur derrière le chatbot. Il est nécessaire pour mettre en ligne une instance unique à chaque client qui va prendre en charge chaque requête des différentes interfaces.

[3]Source : `https://aws.amazon.com/fr/lambda/`

3 Projets

Lors de ce stage, plusieurs projets m'ont été confiés. Dans cette partie je détaillerai mes différentes interventions. J'y détaillerai l'objectif du projet ainsi que les différentes tâches qui m'ont été confiées.

3.1 Les templates

Le premier projet auquel j'ai participé, afin de me faire à l'agencement du Backoffice et de comprendre le fonctionnement des différents éléments qui le composent, avait pour objectif de paramétrer plus rapidement les projets.

Les projets, associé aux clients, doivent être lié à des modèles d'apprentissage pour permettre la compréhension d'un domaine. Faisons la comparaison entre l'immobilier et les ressources humaines.

Dans l'immobilier, le chatbot doit pouvoir comprendre les intentions liées loyer, prix d'achat et de vente, surface, etc. . . De l'autre côté, un chatbot lié aux ressources humaines devra comprendre les intentions liés aux horaires de travail, au salaire, etc. . . En plus des intentions spécifiques au domaine du chatbot vient s'ajouter des intentions plus générales comme les formules de politesse pour saluer l'utilisateur, réponde à ses remerciements et plus encore.

> Pour associer un modèle de compréhension à un chatbot, une personne de chez KMB Labs devaient jusqu'alors sélectionner manuellement dans les paramètres du projet ledit modèle.

N.B: ce projet est très diversifié dans sa manière de s'intégrer à l'existant, nous verrons que celui-ci touche à beaucoup d'aspect différents du Backoffice.

3.1.1 Objectif

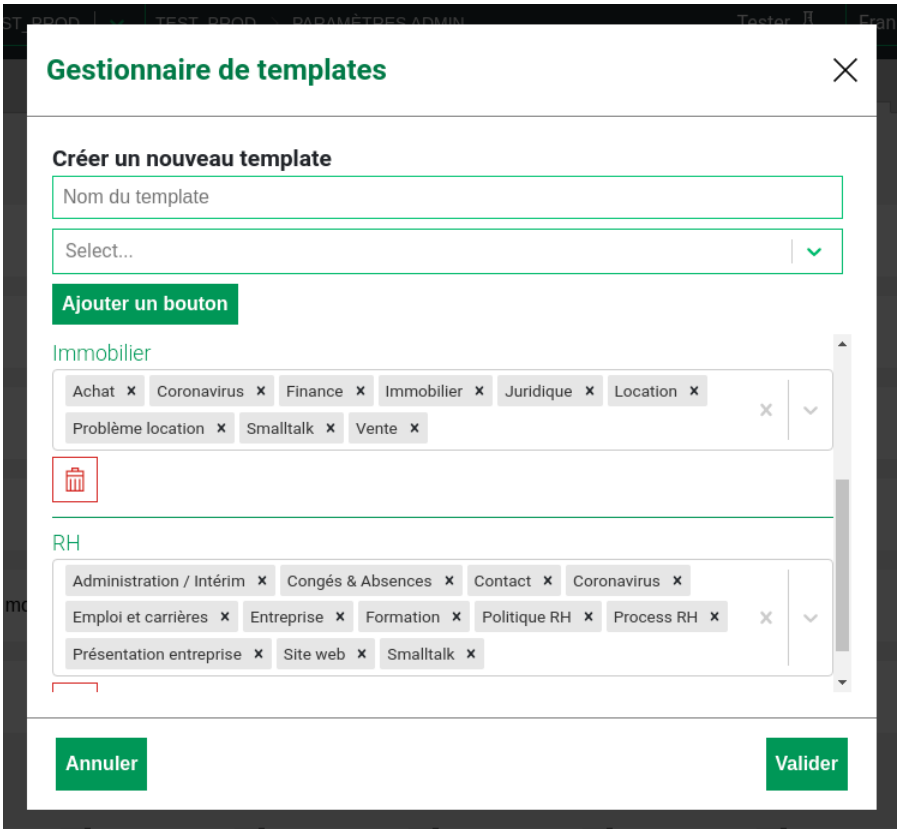
L'objectif d'ajouter des templates étaient de pouvoir facilement associer plusieurs modèle à un projet en sélectionnant un template. Pour se faire, deux points sont à prendre en compte :

- la création du template
- l'association à un projet

La création de template

Pour créer différents éléments du Backoffice (utilisateur, projets, intention) un admin ouvre ce que l'on appelle une *popin*[1] en utilisant le bouton correspondant dans les paramètres globaux.

Pour les templates, le même principe a été utilisé. Pour créer un template il faut donc ouvrir la *popin* ce qui ouvre l'interface suivante :



[1]interface pouvant s'afficher par dessus le reste de l'application web

L'objectif ici était donc d'ajouter une nouvelle *popin* à l'interface en utilisant l'existant. React étant orienté composant, il est très facile de réutiliser l'existant : le squelette de la fenêtre était déjà réalisé, il ne me restait plus qu'à comprendre comment une *popin* était affichée.

En effet la gestion de celle-ci passe par un *store* Redux[2]. Le gestionnaire des *popins* est toujours affiché, cependant une *popin* n'est affichée seulement si son identifiant est présent dans le store.

Une fois le composant permettant de créer un template intégré à l'interface, il a fallu le relier à la base de donnée. Pour ce premier projet, le schéma de l'API ainsi que la mutation permettant de modifier la base de données ont été réalisée par mon maître de stage, Paul Leménager. De mon côté, il ne me restait plus qu'à les utiliser en gardant en tête la structure d'une requête GraphQL.

L'association à un projet

Associer un template à un projet à soulever des questions :

- devons nous garder les valeurs existantes ?
- le template doit-il être associé par un identifiant au projet ?

Pour répondre à la deuxième question, il a fallu dans un premier temps définir les templates qui seront disponibles, avec la liste des catégories d'intentions qui leur seront liées. Puis voir si cela concorde avec les projets existants. Il y a quasiment toujours au moins une catégorie qui n'est pas dans le template qui est utilisé dans chacun des projets. De fait, nous devions garder les valeurs existantes, et toujours pouvoir ajouter une catégorie d'intentions sans pour autant ajouter un template.

Pour le second point, la question a été répondue assez aisément. Même si les templates étaient stockés dans la base de données, associer l'identifiant d'un ou de plusieurs templates à un projet aurait demandé des modifications du schéma de l'API, des lambdas concernées et de créer les liaisons nécessaires pour récupérer les catégories associer au templates pour les ajouter à celles existantes du projet. Nous avons donc pris la décision de faire au plus simple : ajouter aux catégories du projets les catégories du templates, en excluant les doublons éventuels.

Sur le plan de l'interface, une simple liste déroulante permettant de sélectionner un ou plusieurs templates a été ajouté parmi les différents paramètres pré-existant[3].

[2]"*A Predictable State Container for JS Apps*" **Vu sur redux.js.org**
[3]Voir figure 5.1 page 18

La sélection de templates est gérée dynamiquement en fonction des catégories sélectionnées. De fait, si l'on ajoute un template, on étend la liste des catégories actuelle du projet de façon à reproduire le comportement de l'utilisateur.

Ainsi, si on ajoute ou supprime une catégorie, la liste des templates est mise à jour en fonction des changements, sans que la base de données n'ait été mise à jour pour intégrer les templates aux projets.

Ce premier projet m'a permis d'appréhender les différents concepts qui seront réutilisés dans les autres projets liés au Backoffice:

- le store Redux
- les *functional component* en React
- les *query* et *mutation* GraphQL
- les *PopIn*

Mais aussi les différentes notions propre au domaine de l'entreprise :

- les intentions
- les catégories
- les projets

3.2 ML Monitor

ML Monitor est un outil visant à tester et entraîner les services NLP utilisés. Il est l'aboutissement de réflexion qui ont vu se succéder deux versions *standalone*[4]. J'ai eu la chance lors de ce stage de participer à l'intégration de cette outil au sein même du Backoffice.

Cette intégration a permis de gérer d'une nouvelle manière l'entraînement ainsi que de tester[5] des modèles de *machine learning*. Une nouvelle API a été mise en place permettant d'unifier les interactions avec plusieurs fournisseurs NLP. Actuellement, deux fournisseurs sont utilisés : WIT et Luis (Microsoft). En utilisant cette interface entre

[4]L'outil se trouvait alors sur une application web externe au Backoffice
[5]Voir figures pages 19-19

NLP et le Backoffice, il est possible d'interagir les modèles de chaque fournisseur. Cette solution permet d'avoir un fournisseur de secours dans le cas où le fournisseur principal ne serait plus disponible.

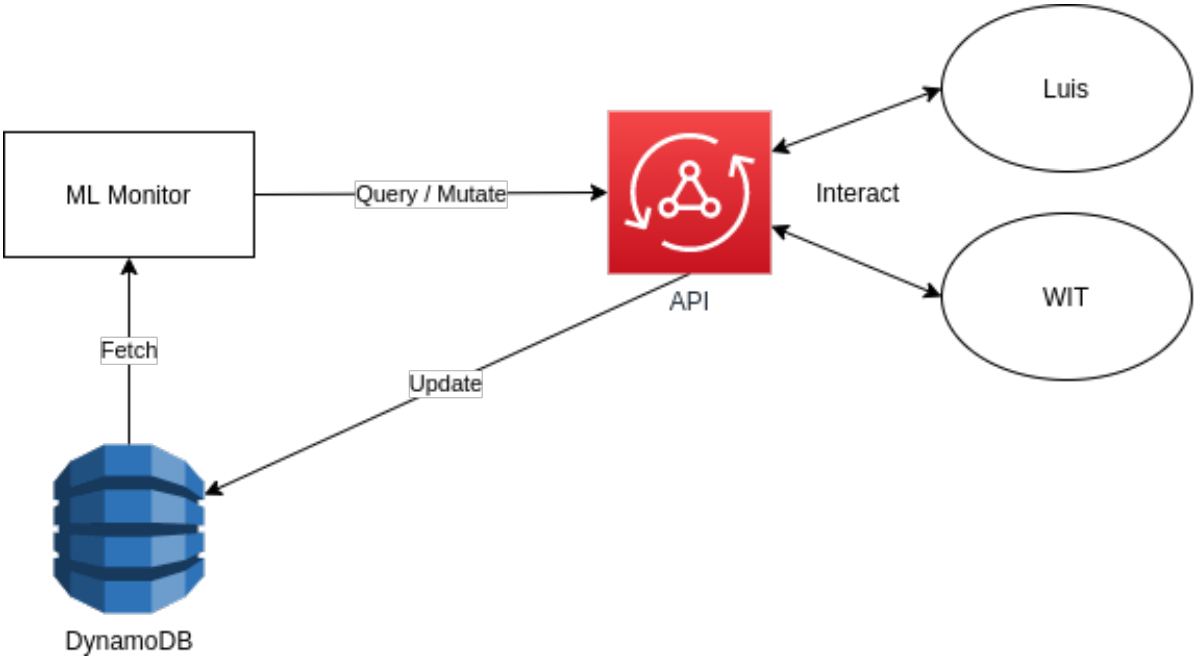


Figure 3.1: Schéma fonctionnel de ML Monitor

Ce projet étant une nouvelle partie du Backoffice, j'ai pu participer à l'élaboration de l'interface ainsi que la partie concernée de l'API. J'ai donc pu découvrir les technologies suivantes :

- AppSync (service AWS pour gérer une API GraphQL)
- Lambda (service AWS associer une fonction Node.JS à un *endpoint*)
- DynamoDB et ses requêtes en VTL
- CloudWatch (service AWS permettant de surveiller les différents services)
- Serverless (permet de gérer l'architecture AWS via des fichiers YAML)

3.2.1 Gestion des fournisseurs NLP

Un des points auquel j'ai pu participer activement est la gestion de multiple fournisseurs NLP[6] sur l'interface.

En effet, même si la venu de l'API permettant d'utiliser plusieurs providers de façon simultanée, il est nécessaire de pouvoir visualiser les données d'un seul en particulier.

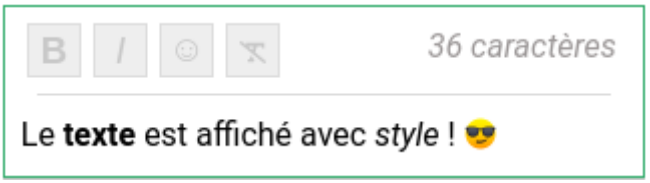
La mise en place de cette fonctionnalité dans l'interface a tout d'abord nécessité une nouvelle lambda qui sera ensuite liée au schéma GraphQL. Chaque catégorie d'intentions est liées à au moins un provider pour entraîner un modèle d'apprentissage avec des données traitant un thème commun (*E.G: salutation*).

La lambda, qui se présente sous la forme d'une fonction Node.JS dans notre cas[7], se charge donc de récupérer toutes les catégories liées au projet courant afin d'extraire l'ensemble des providers utilisés.

Une fois fait, le schéma GraphQL de l'API a dû être mis à jour avec la nouvelle *query*, et lier la source de données de la *query getProviders* à la lambda en utilisant Serverless.

3.3 Éditeur WYSIWYG

L'éditeur WYSIWYG[8] est un ajout que j'ai pu réaliser, avec l'intégration de Draft.js[9] et de différents plugins permettant ainsi d'obtenir un éditeur plus agréable pour l'utilisateur.



Cette éditeur a été utilisé en remplacement d'un éditeur plus basic où le style n'était pas interprété. Le format utilisé nativement par l'éditeur est de l'HTML, cependant un système de conversion a été mis en place afin de correspondre aux formats utilisés dans les fenêtres de discussion de *chatbot*.

> L'éditeur est une partie plus accessoire mais qui a vu quelques complications dû à la gestion de l'état de l'éditeur Draft.JS.
>
> Celui-ci est assez complexe, et comme tous les composants React, doit être pré-

[6] Mentionné "provider" ci-après.
[7] Les Lambda AWS supportent de nombreux langages connus comme Node.JS, Python, Java...
[8] What You See Is What You Get
[9] Facebook opensource

dictible. De fait, chaque modification manuelle implique de reconstruire l'état en utilisant les valeurs précédentes pour garder un éditeur stable (sélection, position du curseur, style, etc. . .).

3.4 ChatWindow

La section ChatWindow du Backoffice est la dernière partie de cette application sur laquelle j'ai particpé durant mon stage. Cette dernière consiste à paramétrer de façon visuel quelques aspects de la fenêtre de discussion des chatbots.

Entre autres, cette section va permettre de configurer les *Fast Action* qui prennent la forme de bouton au dessus de l'icône du *chatbot* comme le montre l'image suivante.

Figure 3.2: Image issue du Backoffice illustrant la fenêtre du *chatbot*

L'avantage des *Fast Actions* est que le *chatbot* n'utilise pas les providers afin d'obtenir les résultats, elles sont directement reliées à une intentions ou bien une réponse définie. De cette façon, l'utilisateur du chatbot peut avoir une réponse instantannée en utilisant ces boutons.

Afin de donner une certaine polyvalence aux *Fast Actions*, celle-ci sont regroupées par page. Chaque page possède une configuration qui lui est propre et qui est configurage via l'interface ChatWindow. On peut donc spécifier une URL d'activation, qui peut être une URL complète, la base d'une URL ou bien une expression régulière.

3.4.1 Création des réponses

La mise en place de ChatWindow dans le Backoffice a permis de réutiliser les éléments déjà en place et d'enrichir les réponses possibles. Avant l'apparition de cette section,

les actions rapides pouvaient être liées à une intention, une réponse texte ou bien à une redirection vers une URL donnée.

Bien que ces trois éléments étaient déjà présent lorsque la configuration du *chatbot* ne se faisait que via l'édition d'un fichier JSON, l'interface permet de visualiser les éléments de réponse :

- l'intention sélectionné est affichée via son nom au lieu de son identifiant
- une réponse texte est affichées avec le style interprété (éditeur WYSIWYG)

  - ces réponses sont sauvegardées de la même manière que celles liées à une intention, de cette façon plusieurs réponses successives peuvent être ajouter et des médias peuvent y être joints.
- la redirection URL est validée avant sauvegarde (expression régulière)

3.5 Interface conversationnelle

L'interface conversationnelle se présente sous la forme d'une page entière (à contrario du *chatbot*). Utilisée comme moteur de recherche, cette interface peut tout de même répondre aux questions liées à des intentions.

La partie recherche est augmentée par le NLP ce qui permet de détecter les filtres possibles dans la recherche de l'utilisateur (localisation, salaire, loyer, surface, etc...). Cette solution rend l'expérience de recherche plus simple pour l'utilisateur, permettant ainsi d'effectuer des recherches en utilisant le langage naturel.

La particularité de l'interface conversationnelle est qu'un seul projet gère les intégrations de plusieurs clients. En effet, la réalisation de ce projet est orientée *config-first*. C'est à dire que chaque élément spécifique au client (nommage, pages disponibles, filtres disponible, etc...) est entièrement géré dans un dossier de configuration qui lui ait propre. Par la suite, l'application du client peut-être générée grâce à Webpack qui nous permet de sélectionner la bonne configuration facilement avec les variables d'environnement.

3.5.1 La page d'aiguillage

La page d'aiguillage est un nouvel ajout nécessaire pour l'intégration d'un client. Celle-ci permet de récupérer le résultat de plusieurs flux de données, selon une recherche donnée, et de présenter les dix premiers résultats.

Avant la création de cette page, plusieurs flux pouvaient être géré par l'interface mais pas de façon simultanée. Une recherche était liée à un flux unique.

Cette particularité a demandé la modification d'une partie des composants avec l'ajout d'une page entière.

> Ma participation sur ce projet n'a pas été importante. Cependant, l'approche *config-first* a été une nouveautée avec laquelle j'ai apprécier travailler malgré les difficultées rencontrées. J'ai pu enrichir les composants globaux ainsi que ceux spécifiques à un projet.
>
> Le principale difficulté ici est de s'intégrer à l'existant : comprendre le fonctionnement des configurations, la manière de récupérer la configuration selon la page courante, etc...

3.6 Optimisation Chat-Window

Le *Chat-Window* est le produit le plus commercialisé par KMB Labs. Au fur et à mesure des ajouts le poids du script JavaScript permettant son exécution a augmenté et est devenue problématique.

En effet, le temps de chargement du script, pourtant décalé vis à vis du chargement du site hôte, provoque des ralentissements pour l'utilisateur. Ce ralentissement dépend de la machine utilisée ainsi que du débit internet, cependant il existe.

Pour palier ce problème, j'ai réalisé des recherches quant aux possibilités à notre disposition pour remédier à ce problème.

> Comment diminuer le poids du *bundle* tout en gardant l'intégralité des fonctionnalités disponibles ?

3.6.1 Webpack - Chunks

React embarque l'outil Webpack afin de compresser tout un projet web dans un seul fichier JavaScript.

Diverses plugins et options permettent de découper les projets afin de séparer les ressources. Globablement, l'utilisation des *chunks* est souvent utilisée pour séparer les *assets* ainsi que les modules issue de Node.js et de NPM dans des chunks à part. Ceux-ci étant rarement modifié, les chunk resteront en cache dans le navigateur et ne seront chargé qu'une seule fois.

Cette solution n'empêche pas les ralentissement au premier chargement, mais peut grandement accélérer les chargements ultérieurs. Or, dans le cas de chatbot, l'utilisateur n'est pas forcément amené à utiliser régulièrement l'assistant. J'ai donc cherché une autre possibilité permettant de charger les modules uniquement au moment de leur première utilisation.

3.6.2 Les imports dynamiques

L'ECMAScript 2020 voit apparaître la possibilité d'import dynamique via la fonction **import()**. Cette fonction va retourner une *Promise* à l'import d'un module.

En rencontrant cette notation lors du processus de "*bundling*", Webpack va autommatiquement commencer à découper le code sous-jacent en chunk indépendant tout en regroupant les dépendances communes à plusieurs chunk.

La grande force de cette approche est que React propose depuis la version 17 les outils nécessaires à son utilisation. Chaque composant peut donc utiliser cette fonction pour charger sa dépendance à l'intérieur même de celui-ci (au lieu d'un import en début de fichier en temps normal).

Les composants fonctionnels deviennent alors des fonctions asynchrones, en utilisant la notation *async/await* pour importer les modules et ainsi permettre d'attendre le téléchargement du module la première fois que le composant est rendu dans le DOM.

> À l'état actuel du projet de Chat Window, cette approche ne peut être utilisée car nécessite la dernière version des différents outils (Webpack et React). Cependant, d'autres dépendances sont également impactées car leur version n'est pas compatible avec la mise à jour de ces deux outils.
>
> La première étape pour la réalisation de cette optimisation est de mettre à jour toutes les dépendances nécessaires avant de pouvoir commencer à mettre à jour le code.

> Cette approche a été choisie car permet un découpage plus précis du code permettant d'avoir un chargement initial comportant uniquement les éléments vitals à l'affichage de la bulle du *chatbot* sans charger tous les comportements de la fenêtre.

4 Conslusion

5 Annexes

5.1 Paramètres du projet

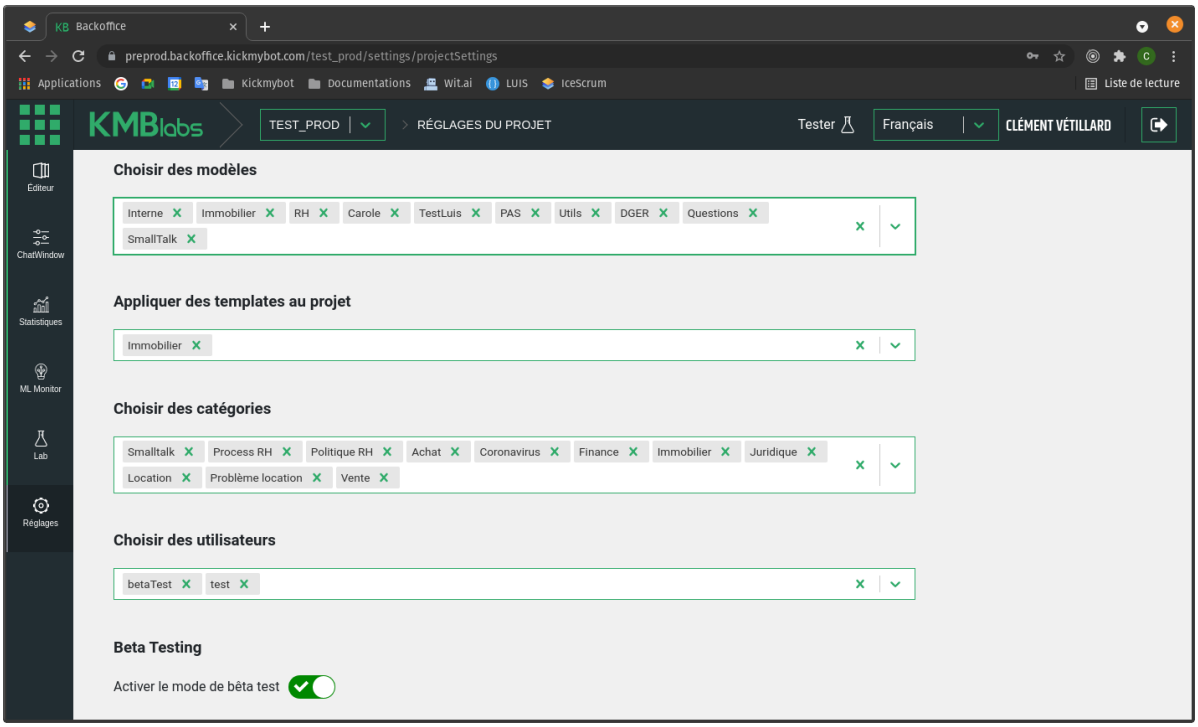


Figure 5.1: Page de paramétrage d'un projet

5.2 ML Monitor

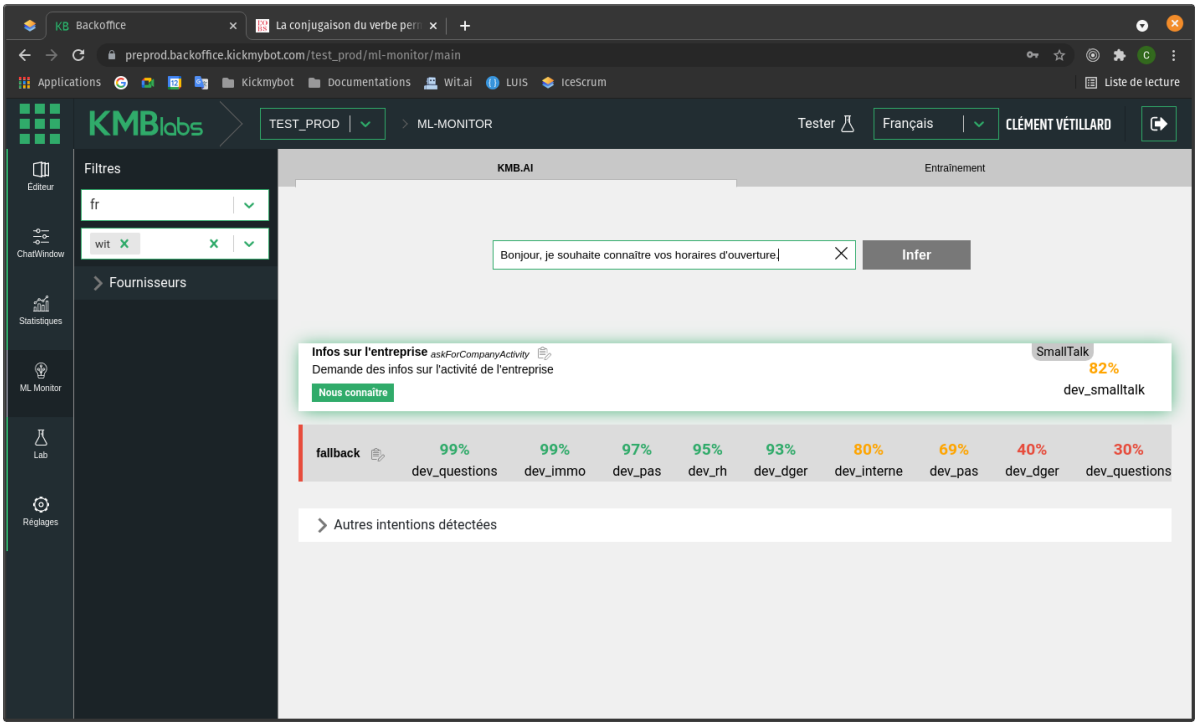


Figure 5.2: Page de test de ML Monitor

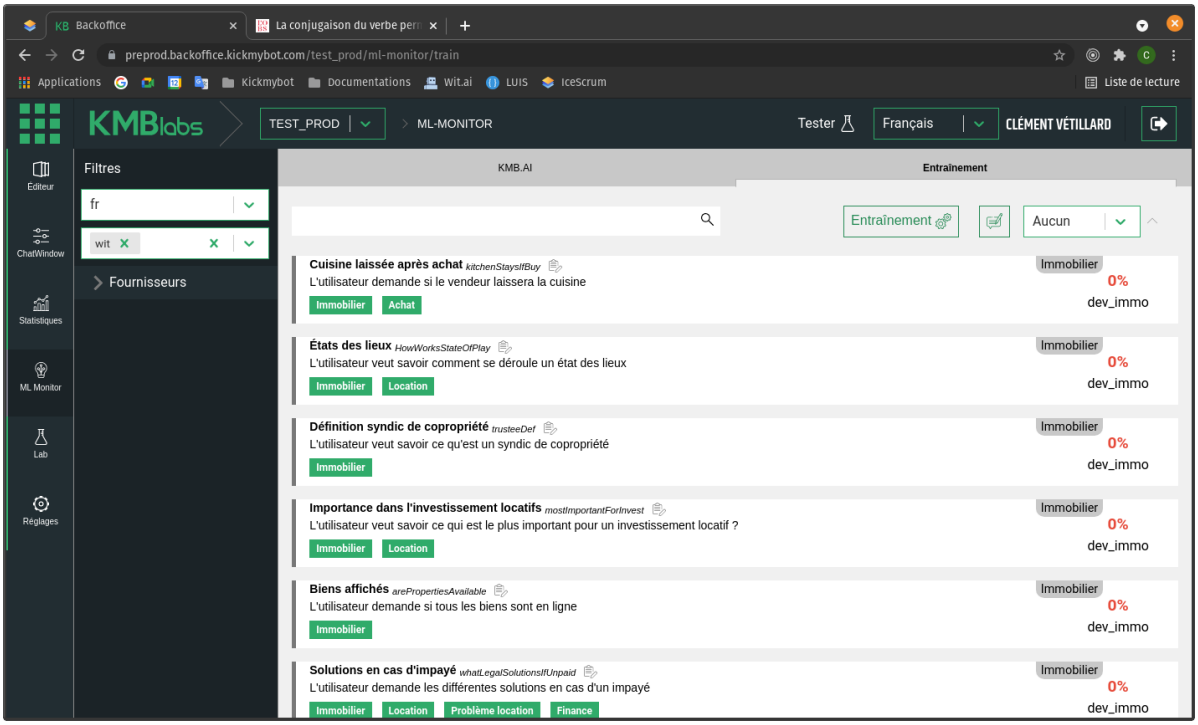


Figure 5.3: Page listant les intentions de ML Monitor

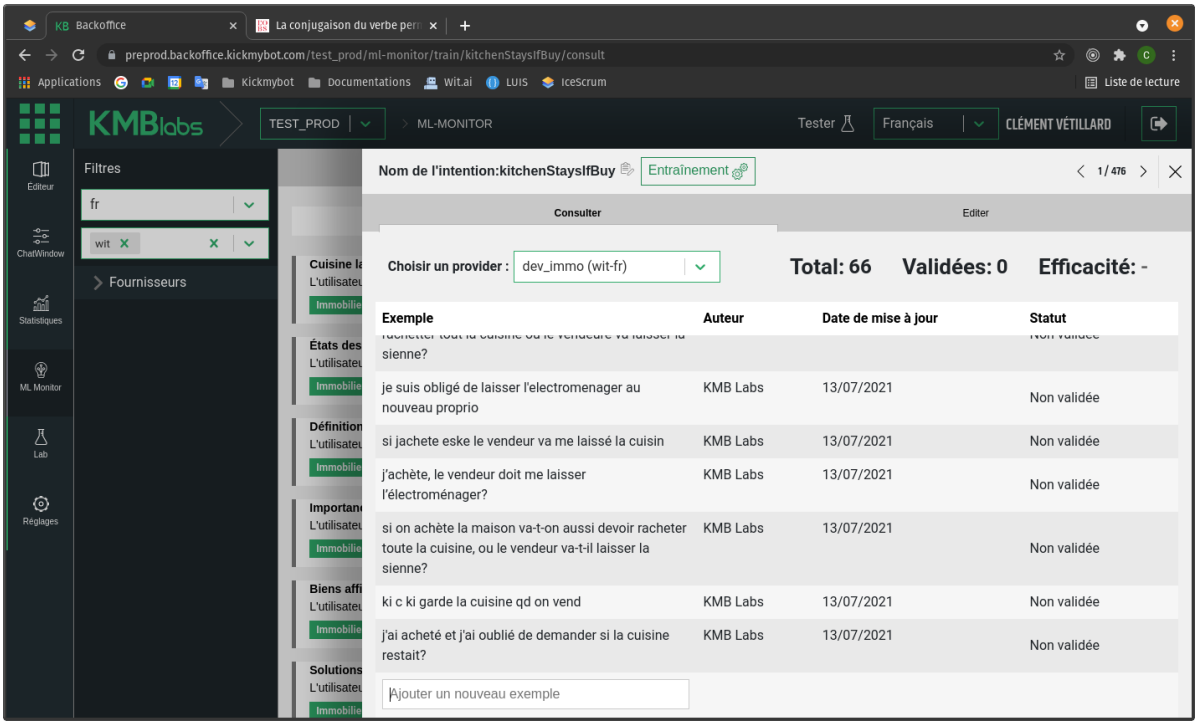


Figure 5.4: Page d'entraînement de ML Monitor