



UNIVERSITÉ
CAEN
NORMANDIE

MASTER INFORMATIQUE
Internet, Données et Connaissances

Mémoire de stage

Systemes 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

Introduction	1
1 Contexte	2
1.1 Les intentions	2
1.2 NLP Provider	3
1.3 Amazon web Services	4
2 KMB Labs	6
2.1 Le chatbot	6
2.1.1 Le Framework	6
2.2 Le Backoffice	7
2.3 L'API	8
2.4 Technologies	8
3 Projets	10
3.1 Les templates	10
3.1.1 Objectif	11
3.2 ML Monitor	15
3.2.1 Gestion des fournisseurs NLP	17
3.2.2 L'entraînement des modèles	18
3.3 Éditeur WYSIWYG	18
3.4 ChatWindow	19
3.5 Interface conversationnelle	20
3.5.1 La page d'aiguillage	21
3.6 Optimisation Chat-Window	21
3.6.1 Webpack - Chunks	22
3.6.2 Les imports dynamiques	23
4 Conclusion	25

Remerciements

Je tiens à remercier Alexandre Ragaleux pour m'avoir permis de réaliser mon stage au sein d'une équipe dynamique.

Je remercie également Paul Leménager qui a pris le temps de m'expliquer le fonctionnement de tous les éléments techniques me permettant de m'intégrer rapidement à l'équipe.

Je remercie l'équipe dans son intégralité pour sa patience et sa bienveillance à mon égare, pour avoir éclairci les morceaux de codes obscures et pour la bonne ambiance au travail, ainsi que les nombreuses parties de Smash lors des pauses déjeuner.

Enfin je remercie mes camarades de promotion avec qui j'ai pu apprendre dans la bonne humeur, et ensemble nous avons pris le temps de découvrir et maîtriser les outils qui seront notre quotidien.

Résumé

Pour mon stage, j'ai eu la chance d'intégrer une jeune start-up, KMB Labs, utilisant des technologies innovantes telles qu'AWS et Serverless. J'ai pu participer au développement de solutions utilisant ces services et comprendre les avantages qui leurs sont propres. KMB Labs est spécialisée dans les interfaces conversationnelles (*chatbot*, moteur de recherche) facilitant le parcours de l'utilisateur dans les sites clients.

KMB Labs articule ses services autour de plusieurs grands projets, certains côté serveur, d'autres servant d'interface avec le client ou bien avec l'utilisateur. On y retrouve la fenêtre du *chatbot*, le produit phare de KMB Labs, une interface conversationnelle couplée à un moteur de recherche, un Backoffice permettant de fournir les éléments de réponses ainsi que l'instance serveur permettant de gérer les requêtes des différentes interfaces. Le tout est couplé avec une puissante API développée avec les services AppSync (GraphQL), Lambda et API Gateway¹ de AWS.

Abstract

During my internship, I had the chance to work with KMB Labs, a new start-up that uses new technologies like AWS or Serverless. I was able to work on improving existing projects and understand the benefits of these tools. KMB Labs specializes in conversational interfaces (*chatbot*, search engine) that facilitate the user experience within the client's website.

KMB Labs develops few services to achieve this. There's the chatbot, of course, and a conversational interface that works with a powerful search engine. There is also a Backoffice that allows the client to provide a response item as well as a server instance for each client's project. All of these also communicate to an efficient API created with AWS AppSync, Lambda, and API Gateway services to provide the GraphQL API.

¹Permet de rendre accessible l'API via une URL

Introduction

KMB Labs² est une entreprise de développement web spécialisée 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 tend à se diversifier afin d'accueillir des chatbots spécialisés dans d'autres domaines (comme des chatbots internes à une entreprise mis 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 et d'améliorer les applications déjà en place. Après une présentation du contexte et des différentes parties composant les services fournis par KMB Labs, je m'efforcerai de détailler mon implication dans chacun des projets auxquels j'ai pu participer pendant toute la durée de mon stage.

²Anciennement Kick My Bot

1 Contexte

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

Pour remédier à cette problématique, 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 néophytes (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, le coût de cette solution peut donc augmenter graduellement si le flux d'utilisateurs augmente.

Avec l'émergence des systèmes de traitement du langage naturel¹ chacune de ses deux approches ont grandement évolué. 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 ce rapport seront étroitement liées aux intentions. Les différents services proposés par KMB Labs cherchent à détecter l'intention de l'utilisateur pour lui

¹Natural Language Processing, abrégé ci-après NLP

fournir la réponse la plus appropriée.

Ainsi, les différentes entités grammaticales² nécessaires à la réalisation de l'intention sont extraites. De fait, un 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 nécessaires à la prise de contact via la fenêtre du chatbot. L'important est que l'*intention* de l'utilisateur pour toutes ces phrases est la même.

1.2 NLP Provider

Les fournisseurs de traitement du langage naturel, ou *NLP Providers*, sont des services externes qu'utilise KMB Labs.

L'objectif de ces *providers* est de pouvoir analyser les demandes formulées en langue naturelle³. Le résultat de l'analyse est de détecter l'intention, citée plus haut, qui se trouve dans le sens de la demande.

Au lieu de simplement faire une recherche par mot clef, les *providers* vont permettre d'entraîner des modèles de données. Chaque modèle aura un jeu spécifique d'intentions, lié à un domaine qui lui est propre.

Ainsi, quand une intention est mal détectée, le *provider* associé est entraîné avec la phrase ainsi que ladite intention. On parle alors d'inférence : le *provider*, une fois l'entraînement terminé, saura détecter l'intention pour cette phrase et celles qui s'en rapprochent.

²Sujet, action, etc...

³Parfois même directement transcrit de l'oral

1.3 Amazon web Services

Amazon web Services, ou AWS, est un fournisseur de services web. On y trouve des solutions pour toutes les problématiques liées aux applications connectées, que ce soit des applications mobiles, web ou bien même des jeux vidéos.

Chaque service est spécialisé dans un domaine et peut être mis en relation avec d'autre. Dans ce rapport, il sera fait mention de DynamoDB, d'AppSync et de Lambda.

DynamoDB

DynamoDB est un système de gestion de base de données NOSQL. De même que MongoDB, DynamoDB est orienté document. Spécialisée dans la gestion de grand volume de données, il est particulièrement adapté pour gérer les nombreuses données liées aux catégories, intentions et phrases d'entraînement mentionnées ci-dessus.

AppSync

AppSync est un service permettant de créer une API GraphQL. GraphQL est un langage dédié à la création d'API qui se veut agnostique à tout système de gestion de base de données.

La puissance de GraphQL est de fournir uniquement les données demandées. Ainsi, en effectuant une requête pour une liste d'objet complexe, voir d'objets imbriqués, GraphQL peut extraire les attributs voulu de chaque objet, quelque soit le niveau d'imbrication pour peu que le type de l'objet ait été défini dans le schéma GraphQL.

GraphQL regroupe les *endpoints* en deux catégories, les *queries* et les *mutations*. Chacun de ces *endpoints* est lié à un *resolver* qui se charge de récupérer les données nécessaires pour qu'AppSync puis envoyer le résultat.

Un *resolver* est un morceau de code VTL⁴ permettant de :

- interagir directement avec DynamoDB
- utiliser d'autres *resolvers* (dans le cas des types imbriqués par exemple)
- appeler plusieurs *resolvers* à la suite (on parle alors de *pipeline*)
- appeler une Lambda

⁴Langage propre à l'environnement AWS

Lambda

Les Lambda sont un outil très puissant dans l'univers d'AWS. Celle-ci prennent la forme de fonction codée dans les langages connus, depuis peu elle peuvent également utiliser un *container* (Docker).

Elles fonctionnent comme une source de donnée : liée à un *resolver* elle reçoivent la requête via celui-ci, effectuent les traitements nécessaire (récupération de données sur DynamoDB, filtre, etc...) puis retourne le résultat désiré. Étant codé dans le langage voulu, elles permettent de reporter une partie de la charge d'une application sur le serveur, allégeant ainsi la charge chez l'utilisateur.

À noter que le grande majorité de ces services sont gérés via *Serverless*. Comme son nom l'indique, *Serverless* permet de ne pas avoir à gérer une architecture serveur soit même mais d'automatiser le déploiement, la mise à niveau ainsi que la configuration via des fichiers YAML.

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 Le chatbot

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

La réponse est préalablement fournie par le client afin de choisir quelles intentions 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, sans passer par les services de *NLP Providers*.

2.1.1 Le Framework

Le Framework constitue les ressources communes derrière chaque *chatbot*. Il est nécessaire pour mettre en ligne une instance unique pour chaque client, et permet de fournir toute la logique pour le *chatbot* (traitement des intentions, récupérations des réponses, sauvegarde des statistiques).¹

L'interface conversationnelle

L'interface conversationnelle 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 :

¹Merci à Paul pour cette définition précise.

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

Cette interface se présente plus comme un moteur de recherche, mais l'intégration des systèmes NLP permettent une certaine flexibilité 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.

Afin de rechercher et de retourner les résultats, Elasticsearch est utilisé. Les flux de données des clients sont récupérés afin de toujours proposer les derniers résultats possibles.

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 exhaustive :

- l'éditeur : permet de définir les réponses du *chatbot* pour chacune des intentions disponibles
- 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
- ML² Monitor : cette partie, pour utilisateur averti uniquement, permet de tester la compréhension d'une phrase et, si besoin, d'entraîner 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)

²Machine Learning

2.3 L'API

L'API a été écrite avec le langage GraphQL et mise en ligne avec le service AppSync de Amazon web Services³. Cette approche permet de ne récupérer que les données nécessaires à l'usage, et ceux même si la structure est 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 résolveurs peuvent être associés à :

- une requêtes directes aux bases de données (via le langage VTL pour DynamoDB)
- une *Lambda* qui sont des fonctions dans le langage désiré (Node.js, Python, Go, Java, etc...⁴)
- d'autres *resolvers* qui se chargeront de récupérer des données différentes

La possibilité de pouvoir lier un *resolver* à une fonction permet d'y effectuer plusieurs traitements. Ainsi, les données ne sont retournées à l'application cliente qu'après les avoir traitées en amont. 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 Technologies

Outre les services d'AWS, GraphQL et *Serverless*, les interfaces sont réalisées avec le framework React.

React est un framework en perpétuelle évolution permettant de réaliser des interfaces via un système de composant. Chaque élément est donc codé sous forme de fonction utilisant le JSX pour mettre en place l'HTML.

Le JSX, en plus de pouvoir utiliser les balises HTML, permet d'utiliser d'autres composants comme s'ils étaient des balises HTML.

En plus de React, Redux est utilisé dans toutes les interfaces. Redux est un framework connu pour la gestion de *store*, et pour le cas de React un conteneur à état commun à toutes l'application.

En effet, les composants React fonctionnent via une gestion d'états. Chaque changement d'état peut provoquer des changements dans l'interface en mettant à jour le rendu

³Abrégé "AWS" ci-après

⁴Source : <https://aws.amazon.com/fr/lambda/>

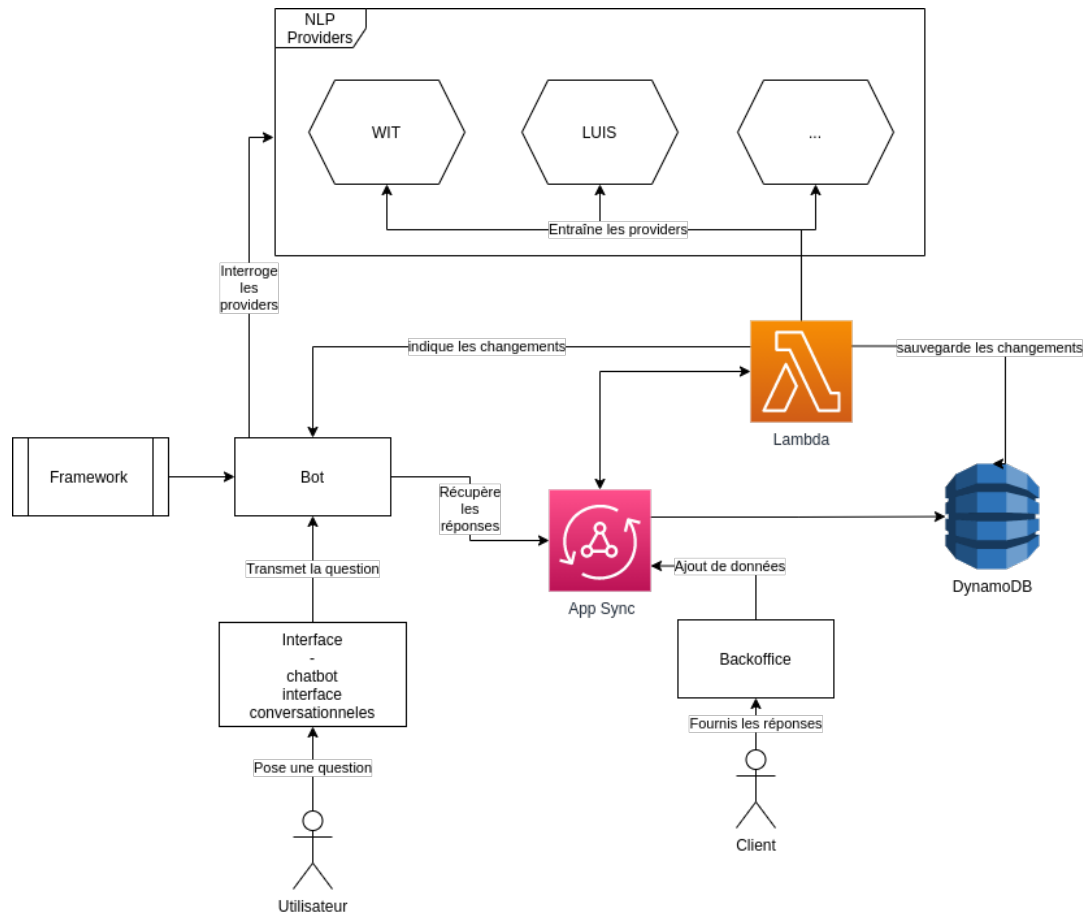


Figure 2.1: Relation entre les différents projets de KMB Labs

des composants. Redux nous permet de stocker des états que plusieurs composants peuvent utiliser de façon à synchroniser les composants utilisant une donnée commune lorsque celle-ci change.

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.

Chaque projet de client est lié à ce qu'on appelle une ou plusieurs catégories. Les catégories regroupent les intentions et sont liées à un *provider*.

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.

On retrouve alors des catégories comme l'immobilier, le recrutement, les phrases de politesse, etc...

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

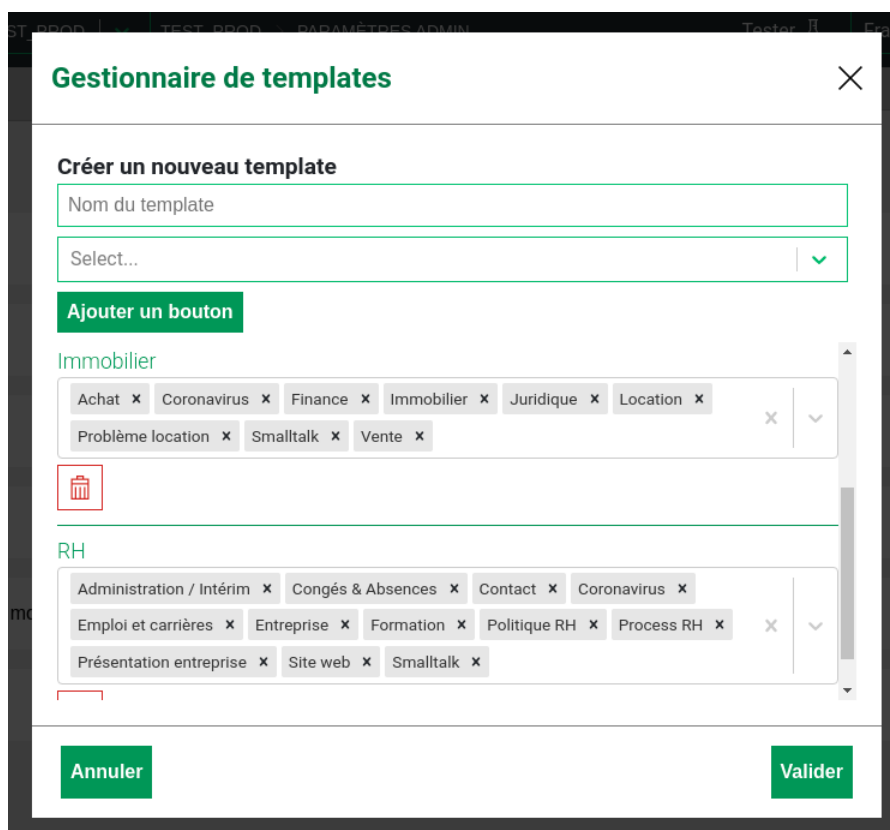
L'objectif d'ajouter des templates était de pouvoir facilement associer plusieurs modèles à un projet en sélectionnant un template. Pour se faire, deux points sont à prendre en compte :

- 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 :



The screenshot shows a modal window titled "Gestionnaire de templates" with a close button (X) in the top right corner. The main section is "Créer un nouveau template", which includes a text input field for "Nom du template" and a dropdown menu labeled "Select..." with a green checkmark icon. Below this is a green button labeled "Ajouter un bouton". The interface is divided into two categories: "Immobilier" and "RH". Under "Immobilier", there are buttons for "Achat", "Coronavirus", "Finance", "Immobilier", "Juridique", "Location", "Problème location", "Smalltalk", and "Vente". Under "RH", there are buttons for "Administration / Intérim", "Congés & Absences", "Contact", "Coronavirus", "Emploi et carrières", "Entreprise", "Formation", "Politique RH", "Process RH", "Présentation entreprise", "Site web", and "Smalltalk". At the bottom of the modal, there are two green buttons: "Annuler" and "Valider".

¹interface 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* Redux². 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 première 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 utilisée dans chacun des projets. De fait, nous devons 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 (Voir figure 3.1).

²“A Predictable State Container for JS Apps” **Vu sur redux.js.org**

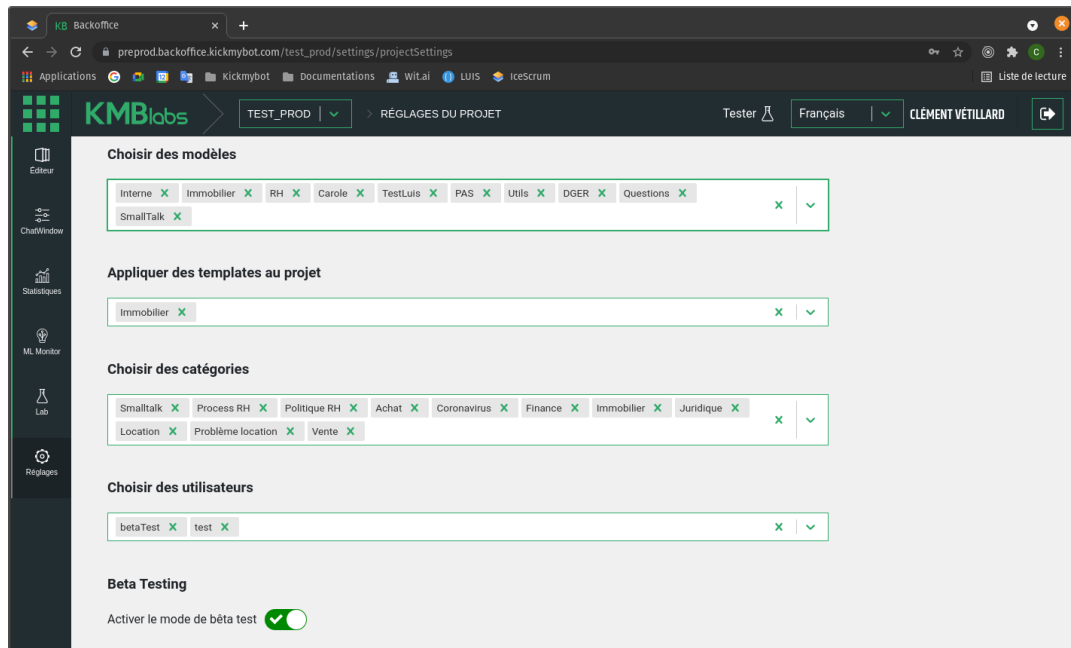


Figure 3.1: Page de paramétrage d'un projet

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 ré-utilisés 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

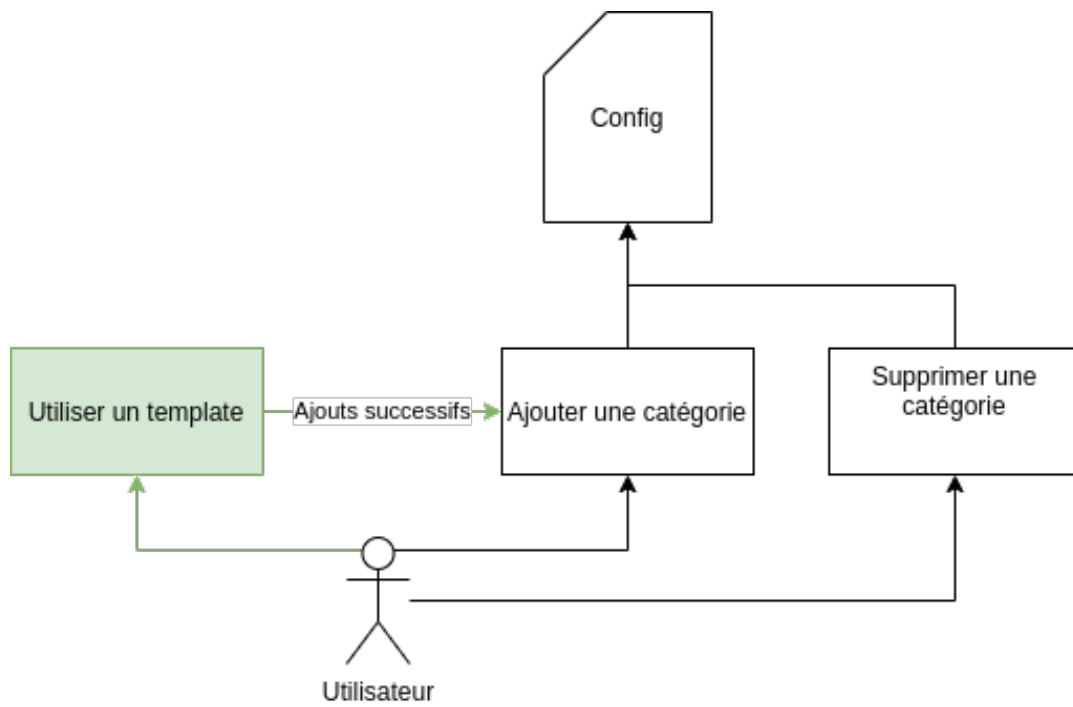


Figure 3.2: Schéma fonctionnel du système de template (coloré en vert)

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*³. 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 (Voir figures ci-dessous) des modèles de *machine learning*.

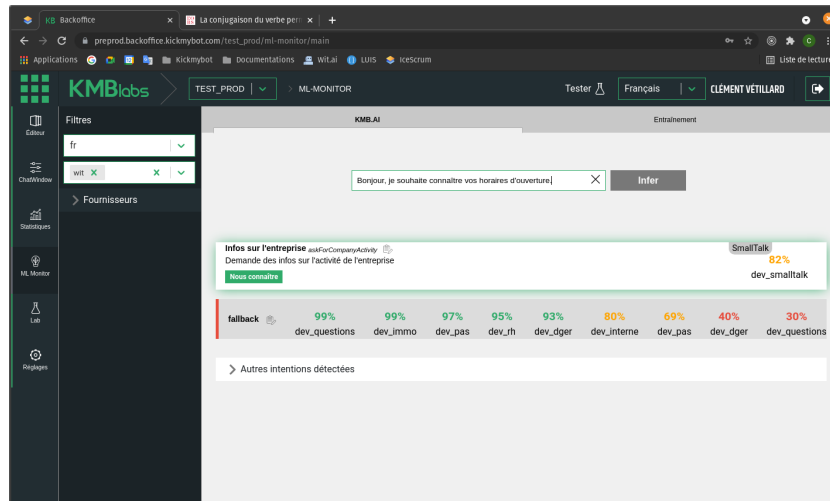


Figure 3.3: Page de test de ML Monitor

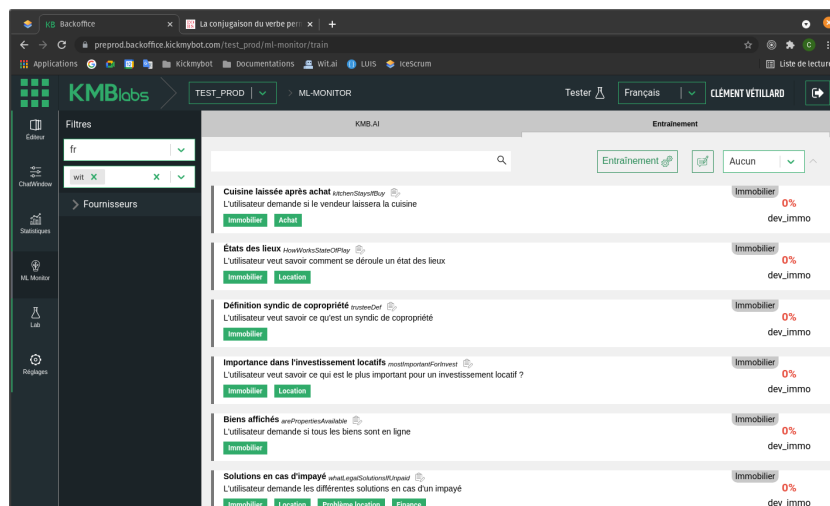


Figure 3.4: Page listant les intentions de ML Monitor

³L'outil se trouvait alors sur une application web externe au Backoffice

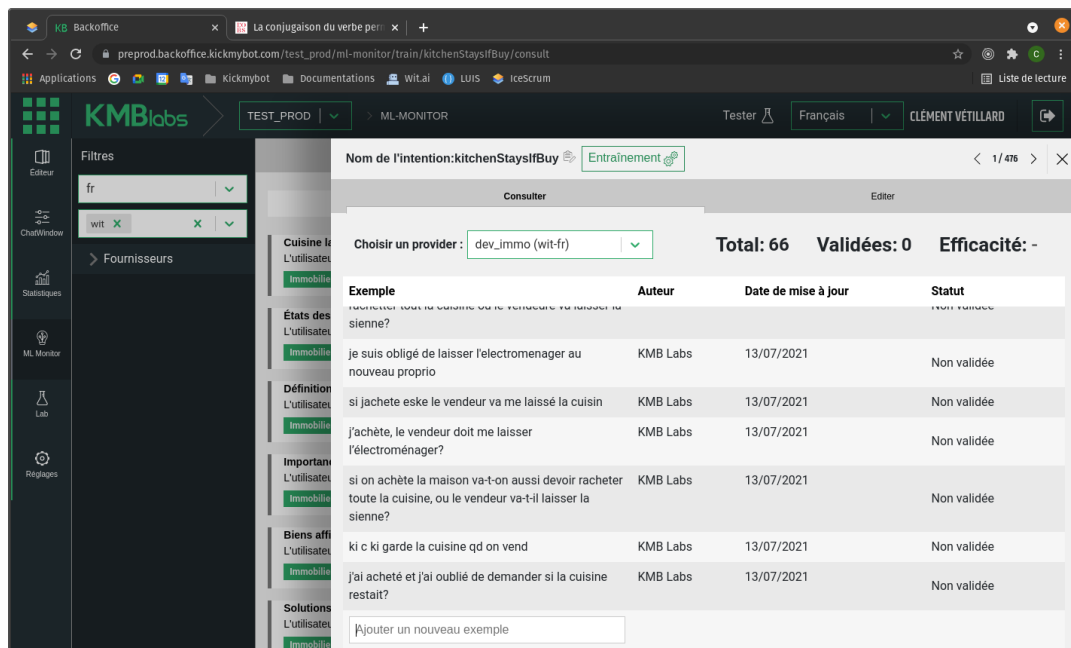


Figure 3.5: Page d'entraînement de ML Monitor

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 NLP et le Backoffice, il est possible d'interagir avec 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.

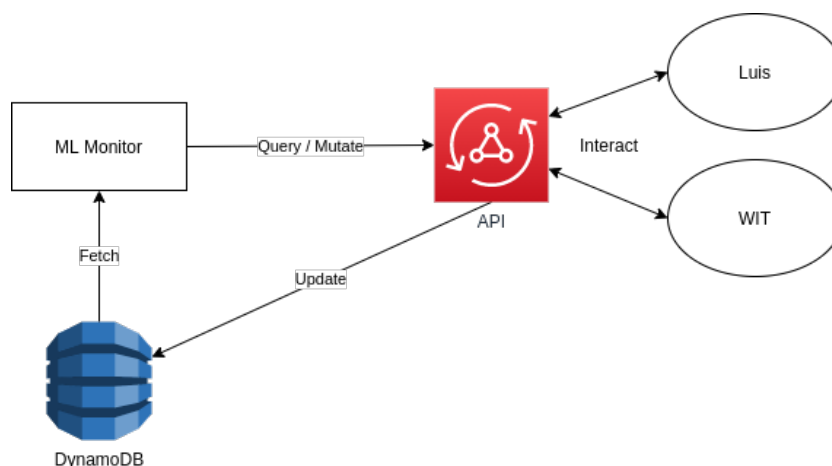


Figure 3.6: Schéma fonctionnel de ML Monitor

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 auxquels j'ai pu participer est la gestion de multiple fournisseurs NLP⁴ sur l'interface.

En effet, même si l'API permet d'utiliser plusieurs providers de façon simultanée, il est nécessaire de pouvoir visualiser les données d'un seul en particulier afin de s'assurer de son fonctionnement.

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 liée 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 cas⁵, 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. Une fois la connexion entre l'*endpoint* et la *lambda* réalisée, chaque demande de cette donnée sera redirigée vers la nouvelle lambda qui se chargera de récupérer et traiter les informations des catégories pour en extraire la liste des *providers*.

⁴Mentionné "provider" ci-après.

⁵Les Lambda AWS supportent de nombreux langages connus comme Node.JS, Python, Java...

3.2.2 L'entraînement des modèles

Le second point qui a été ma principale participation est l'ajout de l'interface permettant d'entraîner le modèle lié à une intention. L'objectif de cette interface est de pouvoir entraîner un modèle avec plusieurs phrases.

De cette façon, les phrases d'entraînements peuvent être rédigées au préalable, via un tableur par exemple, et une fois validée par les personnes compétentes, peuvent être importées dans le modèle via un simple copier/coller.

La mise en place de cette solution a nécessité quelques ajustements, notamment au niveau du schéma de l'API afin de prendre en charge une liste de phrases et non plus une simple phrase, afin de limiter les appels à l'API, puis de modifier la *lambda* associée en conséquence.

La lambda a pour objectif de :

1. mettre à jour la base de données DynamoDB (qui fait foi dans le cas d'un entraînement de l'intégralité du modèle)
2. ajouter les phrases dans chaque provider et lancer la procédure d'inférence afin que les changements soit pris en compte le plus rapidement possible

3.3 Éditeur WYSIWYG

L'éditeur WYSIWYG⁶ est un ajout que j'ai pu réaliser, avec l'intégration de Draft.js⁷ et de différents plugins permettant ainsi d'obtenir un éditeur plus agréable pour l'utilisateur.

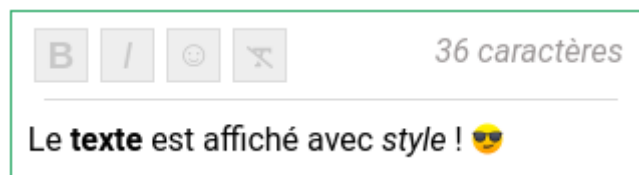


Figure 3.7: Aperçu de l'éditeur tel qu'il est présent dans le Backoffice

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*.

⁶What You See Is What You Get

⁷Facebook opensource

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 prédictible. 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. . .).

De plus, la gestion des tags personnalisés en lieu et place des balises HTML a nécessité la mise en place d'expression régulière permettant de sélectionner les tags pour l'import dans l'éditeur, puis des balises HTML pour l'export.

3.4 ChatWindow

La section ChatWindow du Backoffice est la dernière partie de cette application sur laquelle j'ai participé 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 3.8: 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 configurée

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.

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 sélectionnée 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 formulant une demande.

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 nouveauté avec laquelle j'ai apprécié travailler malgré les difficultés rencontrées. J'ai pu enrichir les composants globaux ainsi que ceux spécifiques à un projet.

La 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*⁸ 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 toujours⁹.

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 ?

⁸Interface du *chatbot*.

⁹Notamment via les tests Lighthouse

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. Globalement, 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 modifiés, les chunk resteront en cache dans le navigateur et ne seront chargés 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.

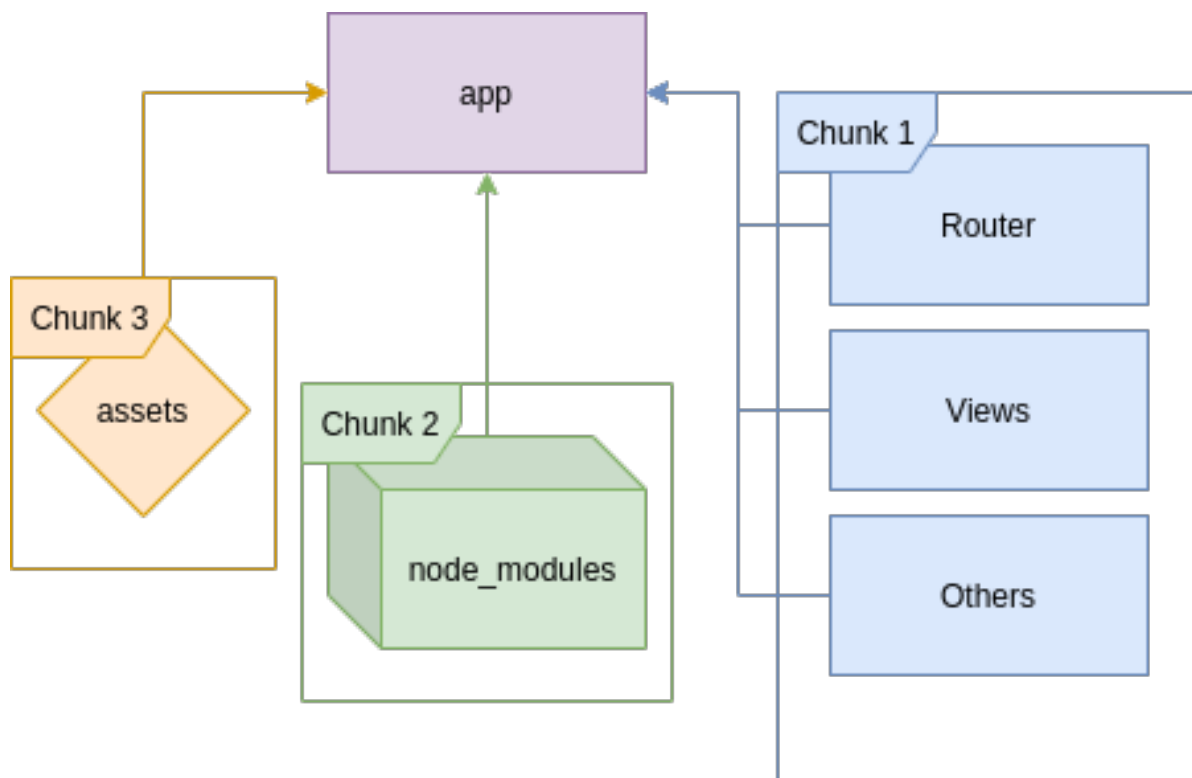


Figure 3.9: Exemple de répartition manuelle des chunks

3.6.2 Les imports dynamiques

L'ECMAScript 2020 voit apparaître la possibilité d'importer dynamiquement une dépendance 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 automatiquement commencer à découper le code sous-jacent en chunks 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 à importer dynamiquement peut être chargé via la fonction *lazy* de React. Cette fonction prend en paramètre un *callback* utilisant la fonction d'import citée plus haut.

De cette façon, React fournit les outils pour :

- importer de façon asynchrone un composant et ses dépendances
- gérer le cas où un composant n'est pas encore chargé (via un composant issu de React (*Suspense*) encadrant les composants asynchrone)

Dans le diagramme ci-dessous, un exemple de chunks dynamiques regroupés par couleur.

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 et utiliser ces nouveaux outils.

Ensuite, la conversion des composants basés sur le système de *class* vers les *functional components* s'est avérée obligatoire afin d'utiliser les outils fournis par React.

Une fois fait, les modules les plus lourds ont été ciblés en premier lieu afin d'obtenir un résultat probant le plus rapidement possible afin d'alléger le poids du build pour tous les clients utilisant le *chatbot*.

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

À titre de comparaison, avant la mise en place sur quelques composants, le chargement initial comportait un téléchargement d'un fichier d'environ 700ko. Avec l'utilisation du *lazy loading*, le chargement initial était descendu à 400ko en ayant transformé que quelques composants.

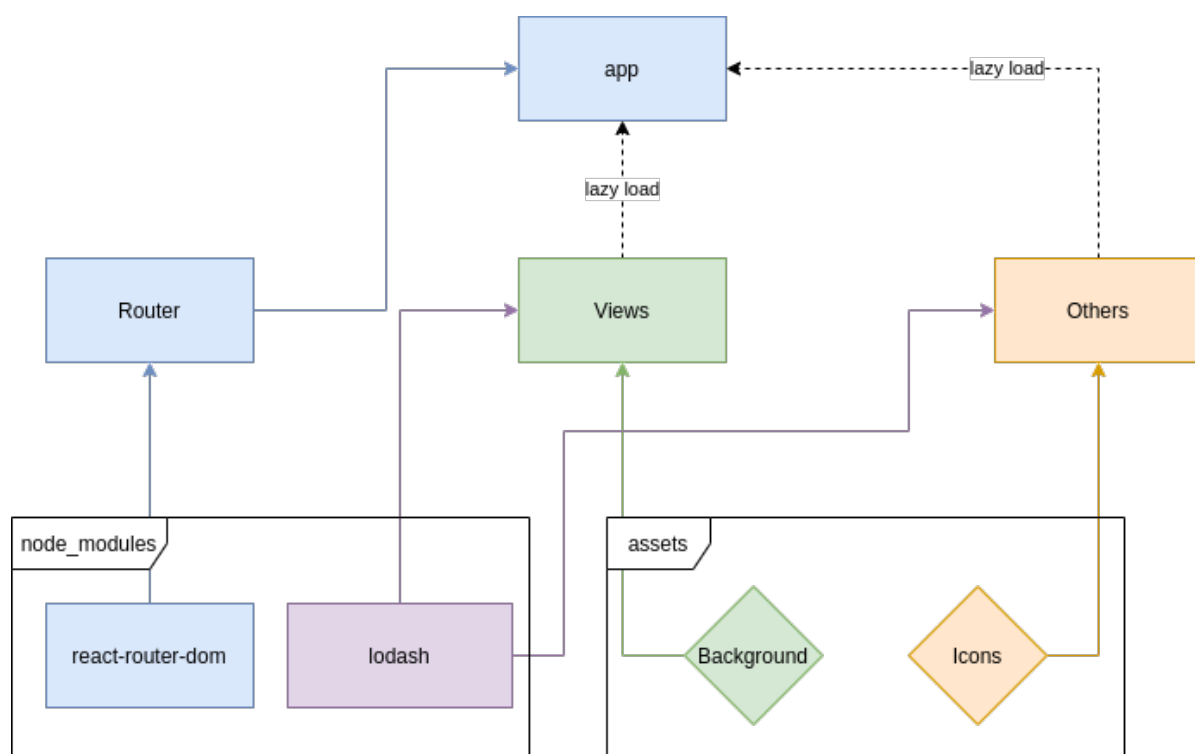


Figure 3.10: Example de répartition dynamique des chunks

4 Conclusion

Lors de ce stage, j'ai pu participer au développement de solutions basées sur des technologies du traitement du langage naturel. KMB Labs est une start-up composée d'une petite équipe de développeurs utilisant des technologies en vogue (aussi bien pour les interfaces utilisateurs que pour la partie serveur).

Ayant participé à de nombreux ajouts de fonctionnalités, j'ai eu l'occasion d'appréhender les enjeux d'un produit commercialisé, avec parfois des problèmes non répertoriés lors des tests qui sont remontés par des clients ; les complications dû à des refontes d'une partie du code. Mais aussi le travail en équipe avec chaque interlocuteur ayant une spécialité, aussi bien en terme de technologies que de programmation ; où l'échange encouragé malgré la pandémie par des outils adaptés (Slack et Google Meet notamment) permet de garder un contact humain.

Ce stage a été pour moi l'occasion de découvrir un monde professionnel dynamique évoluant dans mon domaine d'étude. Où la théorie fait place à de vraies solutions qui ont su faire leur place sur le marché.