

**Supervisé par : Pr. Djotio**

ANNEE ACADEMIQUE :

2022-2023

MEMBRES DU GROUPE

1. AROGA PAULE SIMONNE
2. AZOA ETOUNDI SOPHIE
3. DIFFO ALEX
4. DJOMGUEM FONKOU HERVE
5. ENYEGUE OTTOU DORIA
6. FOTSO DJEUNDJIN LANDRY
7. NCHOUWET GHANE LOIC
8. NGANA NGUEFACK IGOR
9. SARAH ZOCKNEYI GILDASE
10. ZIINAM DANATA VICTOIRE

UNIVERSITY OF YAOUNDE I

\*\*\*\*\*\*\*

NATIONAL ADVANCED SCHOOL OF ENGINEERING

\*\*\*\*\*\*\* COMPUTER SCIENCE

DEPARTMENT

UNIVERSITE OF YAOUNDE I

\*\*\*\*\*\*\*

ECOLE NATIONALE SUPERIEURE POLYTECHNIQUE

\*\*\*\*\*\*\* DEPARTEMENT DE GENIE INFORMATIQUE

Your text here

Your text here

**CHAT BOT « LETS GO »**

Table des matières

[Table des figures 4](#_Toc139222353)

[INTRODUCTION 5](#_Toc139222354)

[I. PRESENTATION DU PROJET 6](#_Toc139222355)

[1. CONTEXTE DU PROJET 6](#_Toc139222356)

[2. OBJECTIFS DU PROJET 6](#_Toc139222357)

[3. PERIMETRE DU PROJET 6](#_Toc139222358)

[4. LA CIBLE 6](#_Toc139222359)

[II. EXPRESSION DES BESOINS 8](#_Toc139222360)

[1. BESOINS FONCTIONNELS 8](#_Toc139222361)

[2. BESOINS NON FONCTIONNELS 8](#_Toc139222362)

[III. MODELISATION MATHEMATIQUE DU PROBLEME 10](#_Toc139222363)

[1. Modélisation mathématique du problème de catégorisation pour un chatBot 10](#_Toc139222364)

[2. Principe et formules mathématiques 10](#_Toc139222365)

[3. Situation illustrative 11](#_Toc139222366)

[4. Application au chatBot let’s go via la librairie apache opennlp afin de déterminer l’intention d’un message. 12](#_Toc139222367)

[IV. DEMARCHE A SUIVRE OU DEROULEMENT DU PROJET 13](#_Toc139222368)

[1. PLANIFICATION 13](#_Toc139222369)

[2. PLAN D’ASSURANCE QUALITE 14](#_Toc139222370)

[3. DOCUMENTATION 14](#_Toc139222371)

[V. DIAGRAMMES D’ANALYSE ET DE CONCEPTION 15](#_Toc139222372)

[1. DIAGRAMME DE CONTEXTE 15](#_Toc139222373)

[2. DIAGRAMME DE PACKAGE 15](#_Toc139222374)

[3. DIAGRAMME DE CLASSES 16](#_Toc139222375)

[4. DIAGRAMME DES CAS D’UTILISATION 17](#_Toc139222376)

[5. DIAGRAMME D’ACTIVITE 19](#_Toc139222377)

[6. DIAGRAMME DE SEQUENCE TECHNIQUE 20](#_Toc139222378)

[VI. GUIDE DE DEVELOPPEMENT 22](#_Toc139222379)

[1. Les dépendances utilisées pour implémenter notre chabot 22](#_Toc139222380)

[2. La couche d’accès aux données 24](#_Toc139222381)

[3. La couche de service 25](#_Toc139222382)

[4. La couche de contrôle 28](#_Toc139222383)

[VII. GUIDE D’INSTALLATION DE L’ENVIRONNEMENT DE DEVELOPPEMENT 31](#_Toc139222384)

[1. Technologies Requises 31](#_Toc139222385)

[2. Prise en Main du projet 31](#_Toc139222386)

[3. Création de L’image Docker du service 41](#_Toc139222387)

[VIII. ARCHITECTURE D’INTEGRATION DES DIFFERENTS MICROSERVICES 42](#_Toc139222388)

[1. Le frontend : 42](#_Toc139222389)

[2. Format d’échange -JSON : 42](#_Toc139222390)

[3. ChatBot service : 42](#_Toc139222391)

[4. Base de données CASSANDRA 42](#_Toc139222392)

[5. API let’sgo 43](#_Toc139222393)

[IX. GUIDE D’INTEGRATION (Avec Let’s Go) 45](#_Toc139222394)

[A. Description 45](#_Toc139222395)

[B. Les Endpoints exposés 45](#_Toc139222396)

[1. POST /api/messages 45](#_Toc139222397)

[2. Post /login 46](#_Toc139222398)

[3. GET /api/messages-all 47](#_Toc139222399)

[C. Exemple d’utilisation 49](#_Toc139222400)

[D. Collection de tous les endpoints édités avec des exemples : 49](#_Toc139222401)

[X. GUIDE D’UTILISATION 51](#_Toc139222411)

[A. Accueil du Chatbot 51](#_Toc139222412)

[B. Administration du Chatbot 52](#_Toc139222413)

[CONCLUSION 55](#_Toc139222414)

[Bibliographie 56](#_Toc139222415)

# Table des figures

[Figure 1: diagramme de contexte 15](#_Toc138313295)

[Figure 2: diagramme de package 16](#_Toc138313296)

[Figure 3: diagramme de classes 17](#_Toc138313297)

[Figure 4: diagramme des cas d'utilisation 18](#_Toc138313298)

[Figure 5: diagramme d'activité 20](#_Toc138313299)

[Figure 6: diagramme de séquence technique 21](file:///D:\SOPHIE\4GI_2022_2023\Semestre2\Réseaux_mobiles_et_intelligents\projet\rapport\rapport%20réseau%20Chatbot%20(1).docx#_Toc138313300)

[Figure 9: architecture microservice 44](file:///D:\SOPHIE\4GI_2022_2023\Semestre2\Réseaux_mobiles_et_intelligents\projet\rapport\rapport%20réseau%20Chatbot%20(1).docx#_Toc138313301)

# INTRODUCTION

La présente réalisation porte sur le développement d’un chatBot, un système de dialogue automatisé capable d’interagir avec les utilisateurs en situation de covoiturage de manière conversationnelle. Le chatBot vise à fournir une assistance et à répondre aux questions des utilisateurs de manière efficace et conviviale. Dans cette réalisation, nous avons utilisé une approche basée sur l’apprentissage automatique et le traitement du langage naturel pour permettre au chatBot de comprendre les intentions des utilisateurs et de générer des réponses appropriées. Pour atteindre notre objectif, nous avons commencé par la réalisation du cahier de charge, puis nous avons continué avec l’analyse et la conception (diagrammes), ensuite l’implémentation des cas d’utilisation (résultats), enfin les guides de l’utilisateur et de déploiement.

# PRESENTATION DU PROJET

## CONTEXTE DU PROJET

Un chatBot aussi appelé dialogueur ou agent conversationnel, est un agent logiciel qui dialogue avec un utilisateur. Dans un monde où les consommateurs (clients) sont de plus en plus exigeants sur la qualité des services et surtout sur la disponibilité des agents de renseignement, cet outil devient donc indispensable pour les entreprises en manque de personnels et qui néanmoins voudraient offrir une disponibilité 24H/24, 7J/7 à leur clientèle.

Le tout premier chatBot a été créé par JOSEPH WEIZENBAUM en 1966 et portait le nom de ELIZA. Depuis lors, plusieurs autres ont vu le jour et peuvent se classer dans trois catégories : chatBots simples, chatBots intelligents et chatBots hybrides.

Il sera alors question dans ce projet de concevoir un chatBot qui sera utilisé pour améliorer la qualité de service d’une application de transport interurbain.

## OBJECTIFS DU PROJET

L’objectif principal du projet est la conception d’un chatBot qui aidera les utilisateurs de l’application de transport *Let’s Go* à planifier leur déplacement quels que soient le jour et l’heure. Ceci par le traitement efficace des différentes requêtes qui lui seront soumises au travers de la plate-forme.

## PERIMETRE DU PROJET

Il est question ici de concevoir un chatBot simple c’est-à-dire qui sera à mesure de traiter les requêtes et d’interagir avec les utilisateurs de la plate-forme de déplacement interurbain.

## LA CIBLE

Cette rubrique concerne ceux qui peuvent utiliser cette plateforme ainsi que les fonctionnalités auxquels ils peuvent bénéficier. Le chatBot peut être proposé à plusieurs catégories d’utilisateurs notamment :

* Les entreprises
* Les applications
* Les enseignants
* Les particuliers

# EXPRESSION DES BESOINS

## BESOINS FONCTIONNELS

Cette section présente les services attendus par l'utilisateur de l'interface. Nous décrivons ce que l'interface doit offrir comme fonctions sans pour autant spécifier les détails de leurs fonctionnements :

* Le chatBot devra être capable de répondre à des questions simples des clients.
* Le chabot devra être capable d’identifier un nouvel utilisateur et lui expliquer ce qu’il sait faire et ce qu’il peut apporter comme informations.
* Tout utilisateur devra être capable d’envoyer un message.
* Il devra être capable d’afficher des boutons, des listes afin de guider les conversations et de s’assurer que l’utilisateur reste dans le flux de la conversation.
* Le chatBot doit gérer le contexte et mémoriser des informations afin de ne pas les redemander.
* Il doit pouvoir afficher des composants graphiques beaucoup plus parlants que du texte tels que les images.
* Il devra garder un historique sur toutes les conversations.
* En cas d’incapacité de réponse ou de dialogue, le chatBot doit orienter le client vers un conseiller (ou un agent).
* Il doit pouvoir mettre en avant des liens de contact comme des emails ou des sites web.
* L’administrateur aura la possibilité d’activer ou désactiver le chatBot pour tous les clients.

## BESOINS NON FONCTIONNELS

Il s'agit des besoins qui caractérisent le système. Ce sont des besoins en matière de performance, de type de matériel ou le type de conception. Ces besoins peuvent concerner les contraintes d'implémentation (langage de programmation, type SGBD, de système d'Exploitation...)

Dans le cadre de ce travail, le chatBot devra être extensible, c'est-à-dire qu'il pourra y avoir une possibilité d'ajouter ou de modifier de nouvelles fonctionnalités.

La conception d'une interface utilisateur doit prendre en compte les besoins, l'expérience et les capacités de l'utilisateur. De ce fait deux contraintes doivent être prises en compte :

· La convivialité et la simplicité de l'interface vis-à-vis de l'utilisateur auquel elle est destinée.

· La conformité des fonctionnalités offertes au traitement exigé.

Le chatBot devra donc être :

* **Evolutif** : Possibilité d’amélioration avec ajout de nouvelles fonctionnalités. Ce sera fondamental pour le développement d’une application mobile.
* **Accessible** : Disponibilité pour tous les utilisateurs 24H/24, 7J/7 (quel que soit la distance) ; Il doit être accessible et adaptable à tous les appareils (responsive) : tablettes, Smartphone, ordinateur…
* **Compatibilité** : Etre compatible avec n'importe quel système d'exploitation
* **La spontanéité dans la réponse** : le chatBot doit répondre rapidement à l’utilisateur
* **La scalabilité** : Le système ne doit pas limiter le nombre d’utilisateurs
* **La performance** : L’augmentation du nombre des requêtes ne doit pas réduire les aptitudes du système
* **La fiabilité** : Le chatBot doit pouvoir gérer rapidement les erreurs, résister aux attaques et en cas de problème doit être remis en état dans les meilleurs délais
* **La sécurité** : Il doit garder les données de l’utilisateur confidentielles et leur protection

# MODELISATION MATHEMATIQUE DU PROBLEME

## Modélisation mathématique du problème de catégorisation pour un chatBot

Le calcul de probabilité via la bibliothèque OpenNLP utilise le modèle du maximum d’entropie. La maximisation de l'entropie est une approche utilisée dans la reconnaissance de langage naturel pour modéliser la probabilité d'une séquence de mots. L'entropie mesure l'incertitude ou l'imprévisibilité d'un événement. Dans le contexte de la modélisation du langage, l'entropie est utilisée pour estimer la probabilité d'une séquence de mots étant donné un ensemble de données d'apprentissage. Il s’agit d’une approche populaire en reconnaissance de langage naturel en raison de sa capacité à prendre en compte un large éventail de caractéristiques et à modéliser des relations complexes entre les mots. Cependant, elle peut nécessiter beaucoup de données d'apprentissage et peut être coûteuse en termes de calcul. D'autres approches telles que les modèles de langage neuronaux basés sur des réseaux de neurones récurrents ou transformer ont également gagné en popularité ces dernières années.

## Principe et formules mathématiques

Le modèle d'entropie maximale, également connu sous le nom de modèle de maximum d'entropie, peut être formalisé mathématiquement de la manière suivante :

Soit X un ensemble de caractéristiques (features) et Y l'ensemble des catégories possibles.

**X = {x1, x2, ..., xn}** : L'ensemble des caractéristiques observées pour une instance donnée.

**Y = {y1, y2, ..., ym}** : L'ensemble des catégories possibles.

Soit P(Y|X) la probabilité conditionnelle de la catégorie Y sachant les caractéristiques X. Dans le modèle d'entropie maximale, nous cherchons à estimer cette probabilité conditionnelle.

Pour cela, nous définissons un modèle probabiliste paramétrique avec des poids (coefficients) θ = {θ1, θ2, ..., θk}, où k est le nombre de caractéristiques distinctes dans X.

La probabilité conditionnelle P(Y|X) est alors calculée à l'aide de la fonction softmax :

: Un vecteur de caractéristiques qui capture les interactions entre X et Y. Chaque élément de est une fonction indicatrice définie comme suit :

= 1 si = Y pour une caractéristique donnée et une catégorie Y, sinon= 0.

= {= 1, = 1, ..., = 1 }

Z(X) : La fonction de partition, qui est la somme de toutes les valeurs exponentielles possibles pour les catégories Y, afin de normaliser les probabilités. C'est-à-dire :

pour chaque catégorie Y.

Le modèle d'entropie maximale cherche à trouver les poids θ qui maximisent l'entropie conditionnelle H(Y|X) sous certaines contraintes. Cette maximisation est généralement réalisée à l'aide d'une méthode d'optimisation telle que la descente de gradient.

L'entropie conditionnelle H(Y|X) est définie comme suit :

H(Y|X) = -Σ P(Y|X) log P(Y|X)

L'objectif du modèle d'entropie maximale est de trouver les poids θ qui maximisent l'entropie conditionnelle tout en satisfaisant les contraintes, généralement exprimées par des moments empiriques ou des contraintes de régularisation.

En utilisant ce modèle, nous pouvons estimer les probabilités conditionnelles P(Y|X) pour prédire les catégories correspondantes pour de nouvelles instances à partir de leurs caractéristiques observées.

## Situation illustrative

Supposons que vous travaillez sur un système de classification automatique des e-mails en fonction de leur contenu. Vous avez collecté un ensemble de données d'apprentissage comprenant des e-mails avec leurs étiquettes correspondantes (par exemple, "spam" ou "non-spam"). Vous voulez construire un modèle qui puisse prédire avec précision si un nouvel e-mail est un spam ou non.

Voici comment vous pourriez utiliser la maximisation de l'entropie dans ce scénario :

* Collecte des données d'apprentissage : Vous avez un ensemble de données d'apprentissage qui contient des e-mails étiquetés comme "spam" ou "non-spam".
* Extraction des caractéristiques : Vous extrayez des caractéristiques pertinentes des e-mails, telles que la fréquence des mots, la présence de certains termes-clés, les informations d'en-tête, etc.
* Construction du modèle : Vous utilisez ces caractéristiques pour construire un modèle d'entropie maximale. Le modèle apprendra à attribuer des poids ou des probabilités aux caractéristiques pour prédire si un e-mail donné est un spam ou non.
* Entraînement du modèle : Vous entraînez le modèle à l'aide de l'algorithme d'optimisation de maximisation de l'entropie. L'algorithme ajuste les poids ou les probabilités des caractéristiques pour maximiser l'entropie et améliorer la précision de prédiction du modèle.
* Évaluation et prédiction : Vous évaluez les performances du modèle en utilisant un ensemble de données de test. Ensuite, vous pouvez l'utiliser pour prédire si de nouveaux e-mails sont des spams ou non.

Dans cet exemple, la maximisation de l'entropie vous permet de construire un modèle qui utilise les caractéristiques extraites des e-mails pour prédire avec précision leur étiquette (spam ou non-spam). Le modèle cherche à maximiser l'entropie, ce qui signifie qu'il cherche à attribuer des probabilités équilibrées aux différentes caractéristiques, ce qui peut être bénéfique pour traiter des données complexes et variables dans les e-mails.

C'est juste un exemple, et la maximisation de l'entropie peut être appliquée à une variété de problèmes de reconnaissance de langage naturel où la modélisation de la probabilité d'une séquence de mots est essentielle.

## Application au chatBot let’s go via la librairie apache opennlp afin de déterminer l’intention d’un message.

L‘entrainement du modèle et le calcul de probabilité est effectuée par la classe **DocumentCategorizerME**.

La méthode **categorize(tokens)** de la classe **DocumentCategorizerME** d'OpenNLP effectue le calcul de probabilité pour chaque catégorie possible d'un document donné. Voici comment cela fonctionne :

* Prétraitement des tokens : Avant de calculer les probabilités, la méthode effectue un prétraitement sur les tokens du document. Cela inclut des étapes telles que la suppression des mots vides (stop words), la normalisation des mots (par exemple, la mise en minuscule) et d'autres transformations pour améliorer la qualité des caractéristiques utilisées pour la classification.
* Calcul des caractéristiques du document : Une fois les tokens prétraités, la méthode utilise un extracteur de caractéristiques intégré pour générer un ensemble de caractéristiques représentant le document. Ces caractéristiques sont basées sur des modèles statistiques tels que les n-grammes, la présence de certains mots-clés, la fréquence des mots, etc.
* Calcul des probabilités : En utilisant les caractéristiques du document, la méthode utilise le modèle d'entropie maximale entraîné pour calculer les probabilités pour chaque catégorie possible. Ces probabilités sont calculées en utilisant les poids attribués à chaque caractéristique dans le modèle. Plus la valeur d'une caractéristique est importante pour une catégorie donnée, plus son poids contribue à la probabilité de cette catégorie.
* Renvoi des probabilités : Une fois les probabilités calculées pour chaque catégorie, elles sont renvoyées sous forme d'un tableau de double. Chaque élément du tableau représente la probabilité d'une catégorie spécifique.

Il est important de noter que les probabilités calculées par la méthode **categorize(tokens)** ne sont pas directement interprétables comme des probabilités absolues. Elles sont plutôt comparables entre elles pour déterminer quelle catégorie a la probabilité la plus élevée, ce qui permet de prédire la catégorie la plus probable pour le document donné.

# DEMARCHE A SUIVRE OU DEROULEMENT DU PROJET

## PLANIFICATION

Notre projet se déroulera comme suit :

* **Etude de l'existant et proposition** : A travers nos recherches sur le fonctionnement des assistants similaires déjà existants, on pourra ainsi clarifier le champ de notre investigation.
* **Etablissement du cahier de charge** : Tout au long de cette phase, nous schématiserons l'expression préliminaire des besoins et nous présenterons une modélisation de fonctionnalités préliminaires de notre application. L’activité principale sera la capture des besoins.
* **Analyse du projet** : Dans notre application, nous allons définir les étapes : les acteurs et les activités ; les diagrammes des cas d'utilisations ; les diagrammes de séquences ; les diagrammes de classes, etc… (diagrammes statiques et dynamiques d’analyse).
* **Conception du projet** : Il s’agit ici d’une description logique de la façon dont le système va fonctionner (étude technique).
* **Résultats possibles** : Nous présenterons les résultats éventuels de notre application dans sa phase de test et d'essai, ce qui nous permettra d'envisager les améliorations possibles. Les difficultés rencontrées nous permettront aussi de dégager un ensemble de pistes susceptibles de nous aider.

## PLAN D’ASSURANCE QUALITE

Pour contrôler la qualité de notre application, nous utiliserons des tests logiciels (opération de vérification). Ce sont des processus d’analyse qui permettent de détecter des anomalies ou de valider le logiciel. Il en existe deux types :

* **Test de boite noire** : Ce type de test est utiliser pour tester un programme en vérifiant que les sorties obtenues sont bien celles prévues pour des entrées données. Le test boîte noire ne nécessite pas de connaître la structure interne du système, il est basé sur la spécification de l’interface du système et de ses fonctionnalités.
* **Test de boite blanche** : C’est une méthode de test logiciel visant à analyser un programme informatique dont on connaît exactement le fonctionnement interne. On peut ainsi utiliser le code source du programme.

## DOCUMENTATION

Notre documentation logicielle sera un manuel qui décrira simplement comment le chatBot sera employé et il permettra à l'utilisateur de comprendre les fonctionnalités et le fonctionnement du logiciel. Elle apportera toutes les informations importantes aux utilisateurs sur ce que fait exactement le logiciel, de telle sorte que leurs attentes soient en phase avec ce qu'ils vont recevoir. Nous y exposerons également les exigences qui motivent l'existence d'une telle fonction avec une suggestion des approches pour des conceptions de plus bas niveau. Enfin les instructions d'utilisation du matériel informatique (ordinateur) y seront inclues.

# DIAGRAMMES D’ANALYSE ET DE CONCEPTION

## DIAGRAMME DE CONTEXTE

Nous avons identifié les acteurs qui interagissent avec le système (chatBot) à savoir :

* **L’utilisateur, acteur principal** : il soumet des requêtes au système (chatBot).
* **Le serveur, acteur secondaire** : qui dialogue avec l’utilisateur.

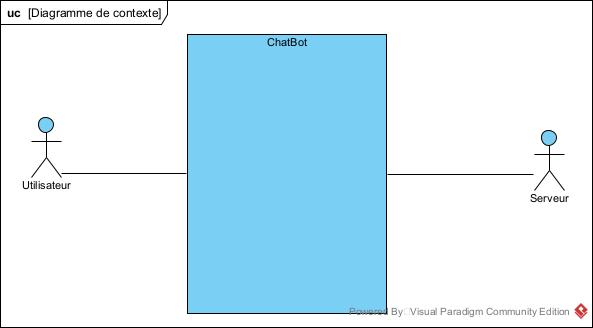


Figure : diagramme de contexte

## DIAGRAMME DE PACKAGE

Dans un système de chatBot, vous pouvez avoir plusieurs packages qui interagissent les uns avec les autres. Voici les packages qu’on a pu relever pour notre système de chatBot :

* **Package d’entrée utilisateur** : Ce package contient des classes qui gèrent les entrées utilisateur, telles que les messages texte, les messages vocaux, etc.
* **Package de réponse de chatBot** : Ce package contient des classes qui gèrent les réponses du chatBot, telles que les messages texte, les messages vocaux, etc.
* **Package de traitement du langage naturel** : Ce package contient des classes qui gèrent le traitement du langage naturel, telles que la reconnaissance vocale, la reconnaissance de texte, etc.
* **Package de stockage de données** : Ce package contient des classes qui gèrent le stockage de données, telles que les bases de données, les fichiers, etc

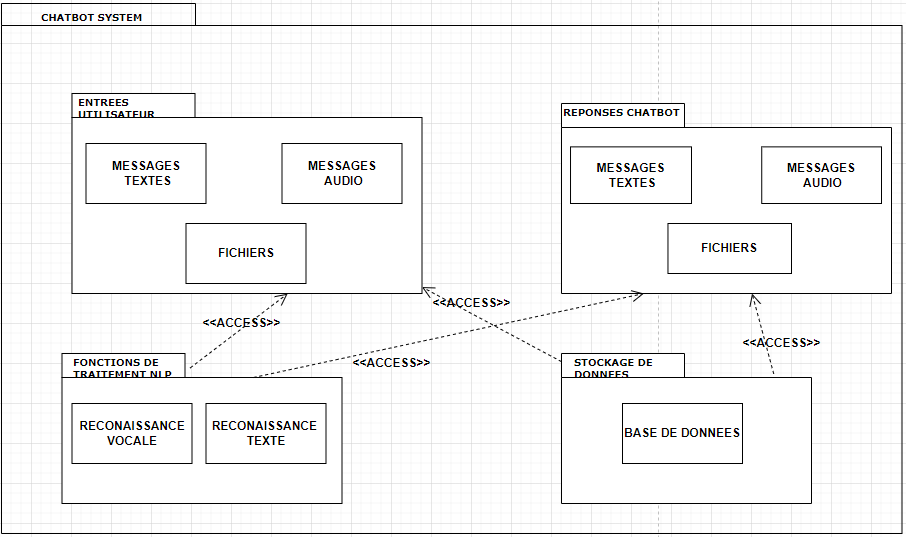


Figure : diagramme de package

## DIAGRAMME DE CLASSES

Il est question d’identifier les données persistantes qui doivent être stockées dans la base de données et qui répondent à des besoins spécifiques. Dans le cadre de la réalisation du chatBot pour un usager en situation de covoiturage, nous avons relevé une seule entité persistante : Message.

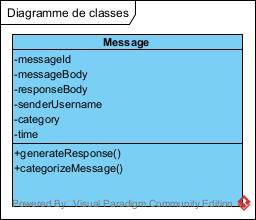


Figure : diagramme de classes

## DIAGRAMME DES CAS D’UTILISATION

En langage de modélisation unifié (UML), un diagramme de cas d'utilisation peut servir à résumer les informations des utilisateurs de votre système (également appelés acteurs) et leurs interactions avec ce dernier. La création de ce type de diagramme requiert un ensemble de symboles et de connecteurs spécifiques. Ci-suit, le diagramme des cas d’utilisation de notre système de chatBot :

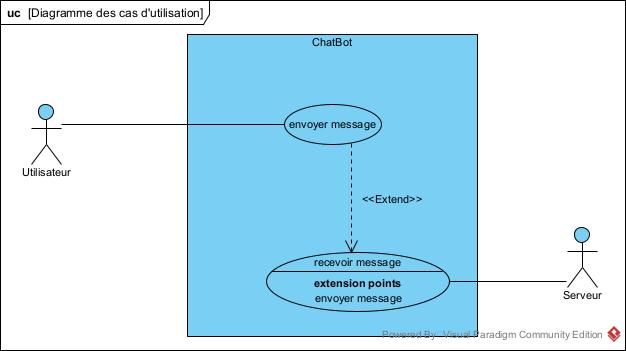


Figure : diagramme des cas d'utilisation

**DESCRIPTION TEXTUELLE DES CAS D’UTILISATION :**

* *Cas d’utilisation « envoyer message » :*
* **Objectif** : Permettre l’envoi de la requête de l’utilisateur.
* **Acteurs concernés** : Utilisateur.
* **Pré conditions** : Aucune.
* **Post conditions** : L’utilisateur reçoit une réponse du serveur.
* **Scénario nominal** :
* L’utilisateur saisit sa requête et envoie.
* Son message est enregistré.
* **Scénarios alternatifs** : Aucun.
* *Cas d’utilisation « recevoir message » :*
* **Objectif** : Permettre à l’utilisateur d’acquérir la réponse à sa requête.
* **Acteurs concernés** : Utilisateur, Serveur.
* **Pré conditions** : L’utilisateur envoie un message.
* **Post conditions** : Aucune.
* **Scénario nominal** :

-Le serveur catégorise le message et génère la réponse.

-La réponse est envoyée à l’utilisateur.

* **Scénarios alternatifs** : Aucun.

## DIAGRAMME D’ACTIVITE

Les diagrammes d'activités permettent de mettre l'accent sur les traitements. Ils sont donc particulièrement adaptés à la modélisation du cheminement de flots de contrôle et de flots de données. Ils permettent ainsi de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation.

Dans notre cas, l’utilisateur envoie un message au chatBot. S’il s’agit d’un message prédéfini, le chabot répond instantanément avec la réponse prédéfinie. S’il s’agit d’un message aléatoire, le chatBot tentera de le catégoriser. Après si la catégorisation échoue, le chabot demande à l’utilisateur de reformuler le message ; si la catégorisation réussit, le chatBot répond avec le message approprié après avoir éventuellement collecté des données en provenance de l’API *Let’s Go* en cas de besoin.

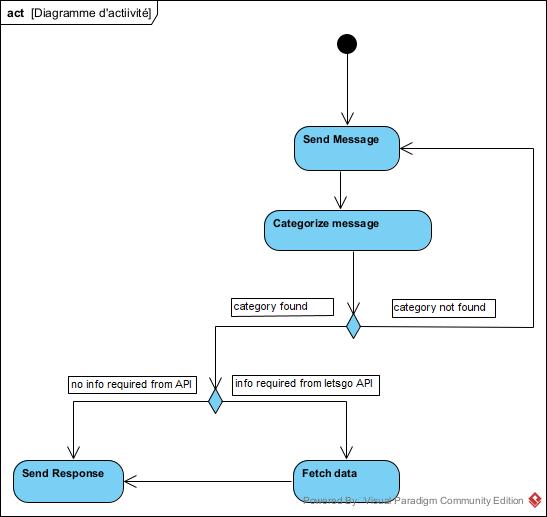


Figure : diagramme d'activité

## DIAGRAMME DE SEQUENCE TECHNIQUE

Les diagrammes de séquence technique décrivent de manière temporelle les interactions entre objets et acteur. Ci-après le diagramme de séquence technique de l’action envoyer-recevoir message :

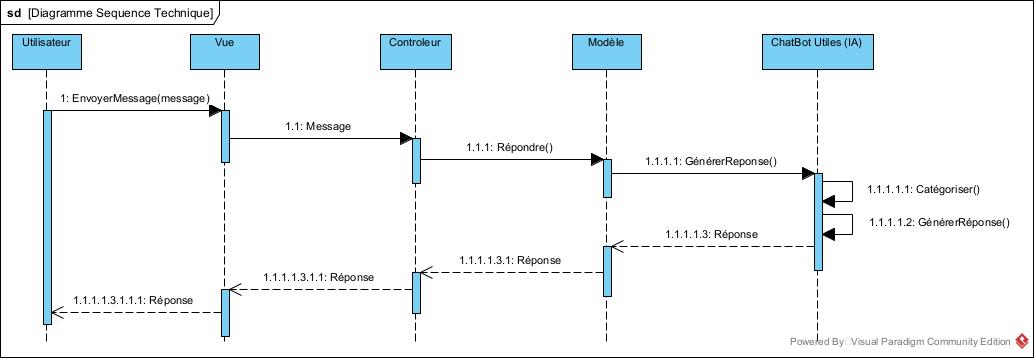


Figure : diagramme de séquence technique

# GUIDE DE DEVELOPPEMENT

### Les dépendances utilisées pour implémenter notre chabot

#### Les dépendances Maven

Les dépendances Maven ajoutées à ce projet sont les suivantes :

* **spring boot actuator**

Spring Boot Actuator est une fonctionnalité puissante du framework Spring Boot qui fournit plusieurs points de terminaison (endpoints) prêts à l'emploi pour la surveillance et la gestion de vos applications Spring Boot en production. Il permettra de collecter des informations et de gérer l’application dans un environnement de production sans nécessiter de développement supplémentaire. Grâce au endpoint **“actuator/health”** qu’il expose, nous pourrons obtenir des informations sur l'état de santé de notre application, indiquant si elle est opérationnelle ou rencontre des problèmes.

* **spring-boot-starter-data-cassandra**

spring-boot-starter-data-cassandra est une dépendance de démarrage (starter) de Spring Boot qui facilite l'intégration et l'utilisation de Cassandra dans une application Spring Boot. Cette dépendance inclut les bibliothèques et les configurations nécessaires pour utiliser Cassandra en tant que base de données dans le projet.

* **spring-boot-starter-web**

spring-boot-starter-web est une dépendance de démarrage (starter) de Spring Boot qui facilite le développement d'applications web avec Spring Boot. Cette dépendance inclut les bibliothèques et les configurations nécessaires pour créer des applications web à l'aide de Spring MVC (Model-View-Controller).

* **Lombok**

Lombok est une bibliothèque Java qui permet de réduire la quantité de code boilerplate (code redondant) nécessaire à l'écriture de classes Java. Elle permet de générer automatiquement certains éléments du code source, tels que les getters et setters, les constructeurs, les méthodes equals() et hashCode(), et bien d'autres, en utilisant des annotations.

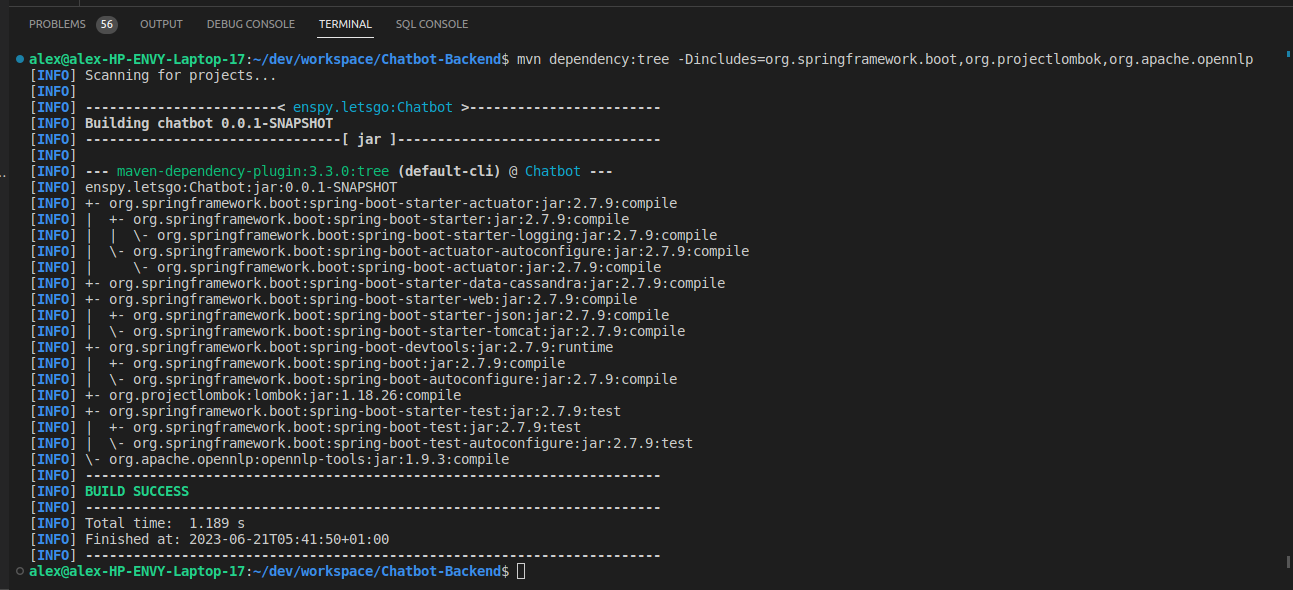
* **spring-boot-starter-test**

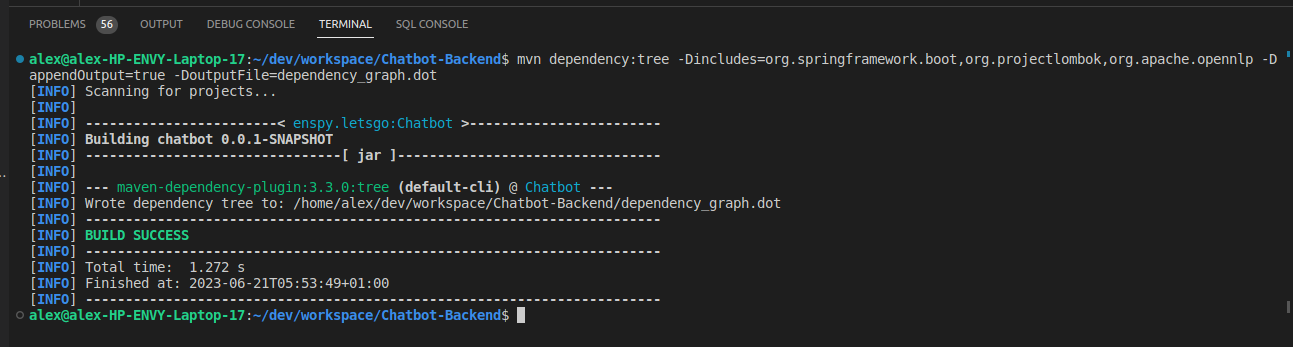
spring-boot-starter-test est une dépendance de test pour les projets Spring Boot. Elle facilite l'écriture et l'exécution de tests unitaires et d'intégration dans vos applications Spring Boot.

* **Opennlp-tools *(version 1.9.3)***

opennlp-tools est une bibliothèque Java open source développée par Apache OpenNLP (Natural Language Processing) qui fournit des outils et des modèles pré-entrainés pour le traitement automatique du langage naturel.

#### Le graphe de dépendances Maven

Il a été généré automatiquement à partir de la commande **“mvn dependency:tree -Dincludes=org.springframework.boot,org.projectlombok,org.apache.opennlp”**

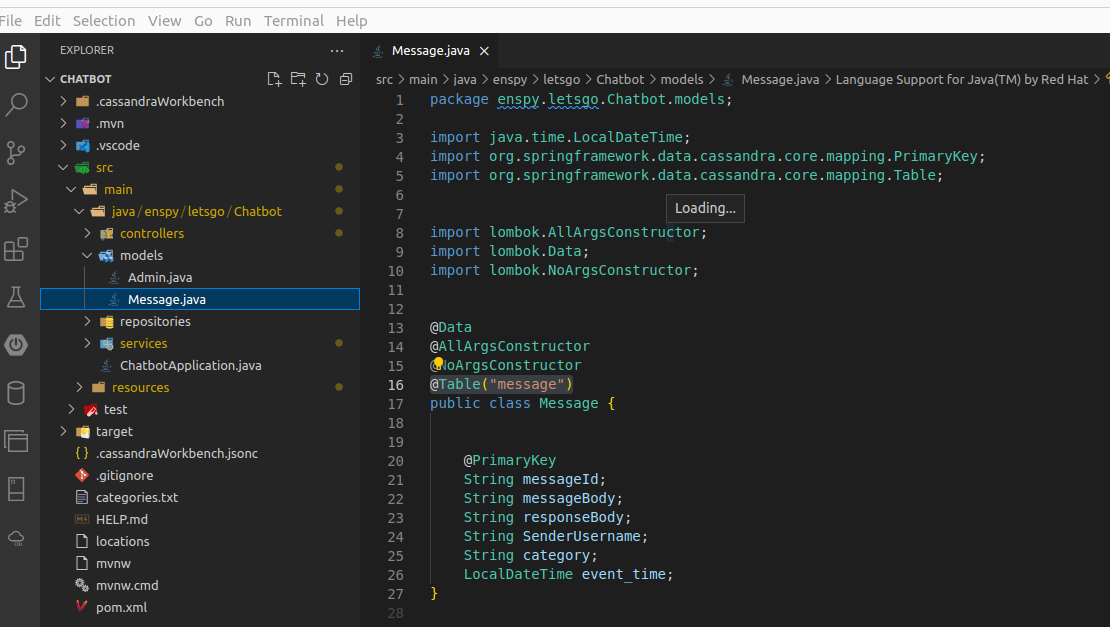
Pour le visualiser graphiquement, nous générons un script dans le fichier **dependency\_graph.dot** à la racine de la repository du projet l’aide de Maven, ce qui nous permet d’utiliser l’application graphitz pour visualiser le graphe de dépendance (imprimé au format A3 et ajouté au document).

### La couche d’accès aux données

#### *Les entités du package models*

* La classe message

C’est la seule classe persistante, explicitement annotée avec @Table("message") pour indexer la table équivalente en base de données. Elle se trouve dans le package models accessible à travers le chemin src/main/java/enspy/letsgo/chatbot/models/message.java



* La classe Admin

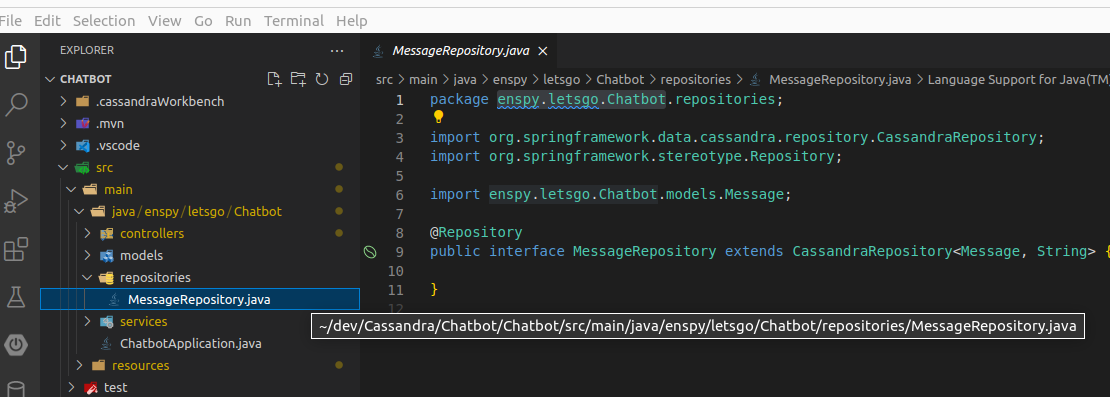
C’est une classe non persistante, elle sert pour le mappage de l’objet json {username, password} provenant du coté client afin de l’authentifier pour qu’il puisse visualiser l’historique des messages et des réponses.

#### *Les interfaces d’accès aux données du package Repositories*

Pour permettre l’accès à la base de données :

* L’interface MessageRepository

Elle étend la classe cassandraRepository de la librairie org.springbootframework.data.cassandra.repository et permet d’implémenter l’accès à la base de données lorsqu’un service a besoin d’accéder à la BD pour effectuer une opération.



### La couche de service

Elle contient tous les services de l’application implémentés à travers les méthodes de ses classes.

#### Les Classes de service

* La classe MessageService du package Services

Accessible à travers le chemin src/main/java/enspy/letsgo/chatbot/models/MessageService.java, elle contient tous les services à travers les méthodes qui y sont implémentées, ainsi qu’une injection de dépendance de la classe MessageRepository pour accéder à la base de données. Parmi les services on a :

* saveMessage()

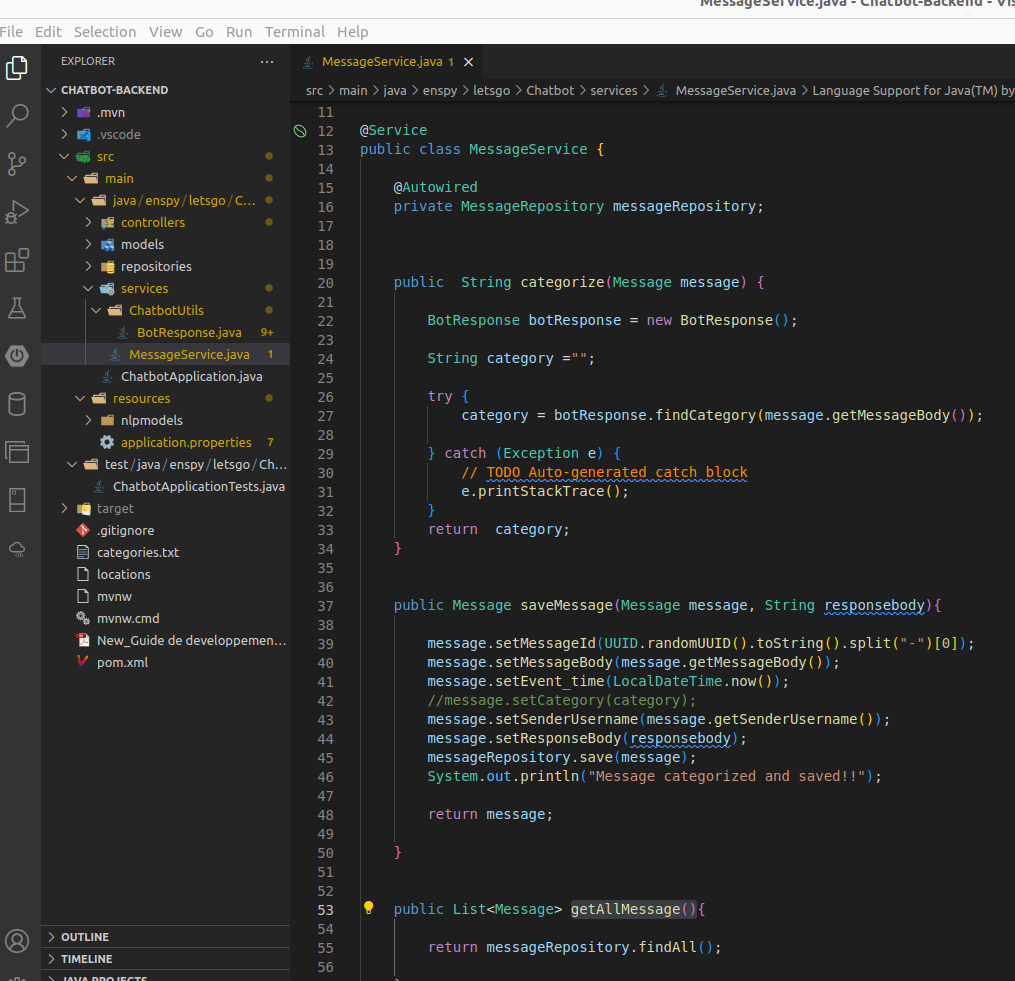
Permet de sauvegarder un message incluant sa réponse et sa catégorie, ainsi que l’horaire et la date correspondante.

* categorize()

Permet de catégoriser les messages afin d’y apporter la réponse appropriée. C’est le nœud central du business logique de cette application.

* getAllMessage()

Permet de récupérer l’historique de tous les messages pour la vue administrateur, ce qui permettra d’optimiser les fichiers de catégorisation afin de mieux entrainer le modèle plus tard en vue de le rendre davantage performant.

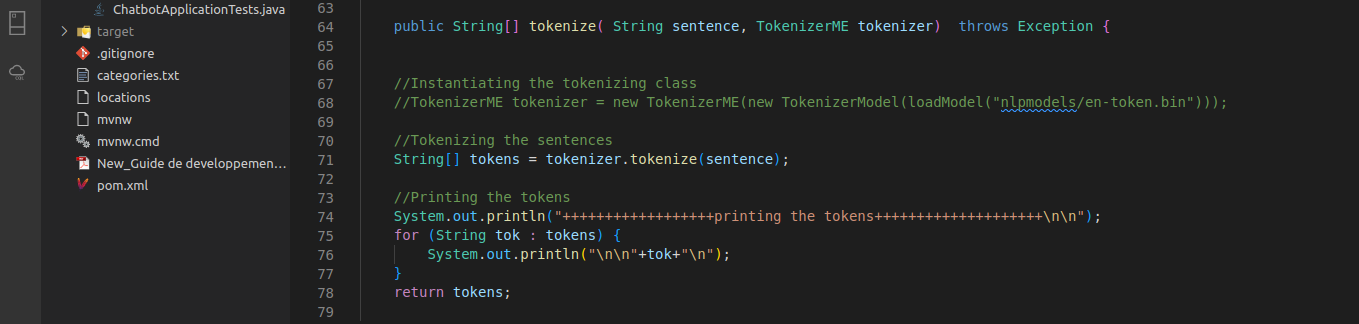


* Les classes utilitaires **BotResponse**

Elle est disponible dans le package Services>chatbotUtils. Elle contient les méthodes du pipeline de catégorisation d’apache openNLP. À savoir :

* tokenize()

C’est la méthode qui constitue la première étape de notre pipeline. Elle utilise des librairies d’apache openNLP pour tokenizer le message reçu et le décomposer en token afin de pouvoir le traiter.



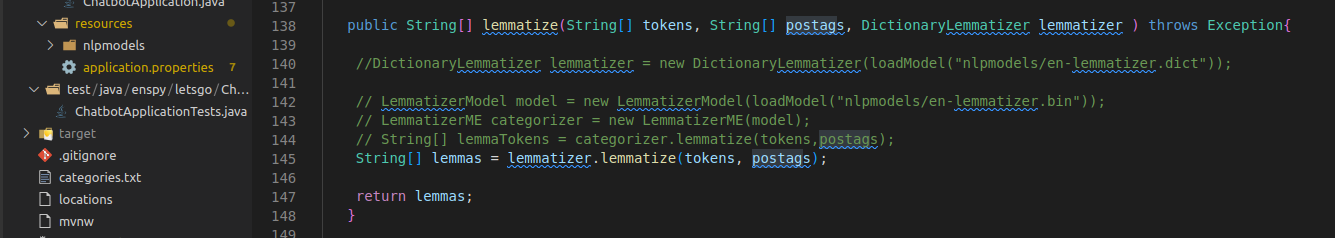
* Postag()

C’est le part of speech tagging qui constitue la seconde étape de notre pipeline. Elle utilise les librairies d’apache opennlp pour catégoriser chaque mot dans sa catégorie grammaticale appropriée, telle que nom, verbe, adjectif, adverbe, pronom, préposition, conjonction, etc. le part of speech tagging est une étape essentielle dans les tâches de traitement automatique du langage naturel (TALN), car il fournit des informations précieuses sur la structure syntaxique et les relations grammaticales au sein d'une phrase.



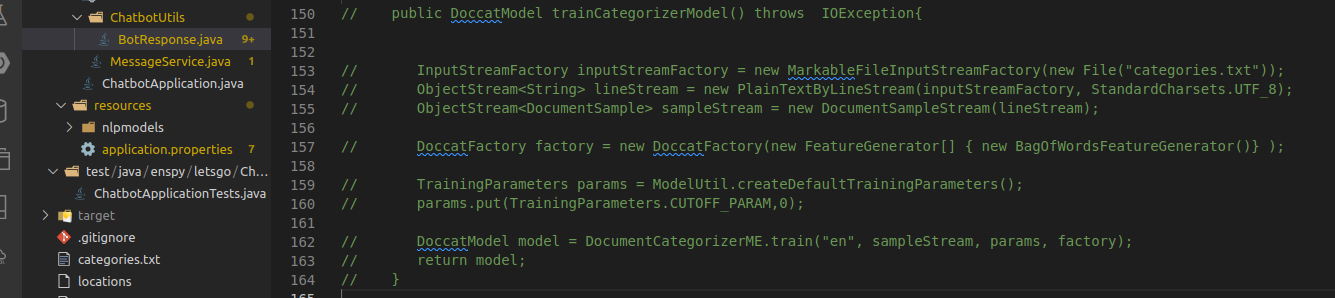
* Lemmatize()

Elle constitue la troisième étape de notre pipeline, elle utilise des librairies d’apache openNLP pour effectuer un processus de réduction d'un mot à sa forme de base ou de dictionnaire, appelée "lemme". Cela implique la normalisation des mots en les ramenant à leur forme canonique, ce qui facilite leur analyse et leur traitement ultérieur.



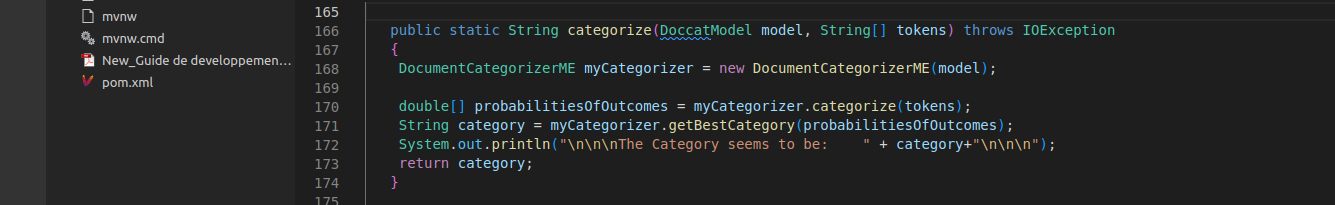
* Traincategorizer()

C’est la quatrième étape de notre pipeline (entièrement commentée parce qu’elle a été factorisée pour améliorer le temps de réponse de l’application), elle permet d’entrainer le modèle de catégorisation d’apache openNLP à partir du fichier categories.txt qui se trouve à la racine du projet.



* Categorize()

Elle constitue la cinquième et dernière étape du pipeline, cela implique d’utiliser le modèle entrainé à la quatrième étape pour catégoriser les messages reçus afin de générer une réponse appropriée :



### La couche de contrôle

Elle gère l’interaction entre la vue et le modèle, permettant ainsi de déclencher les services en fonction de la requête HTTP entrante.

#### Les classes de contrôle (package Controller)

* La classe MessageController

Elle se trouve à l’adresse src/main/java/enspy/letsgo/chatbot/controller/MessageController.java et contient les méthodes de déclenchement des services suivants :

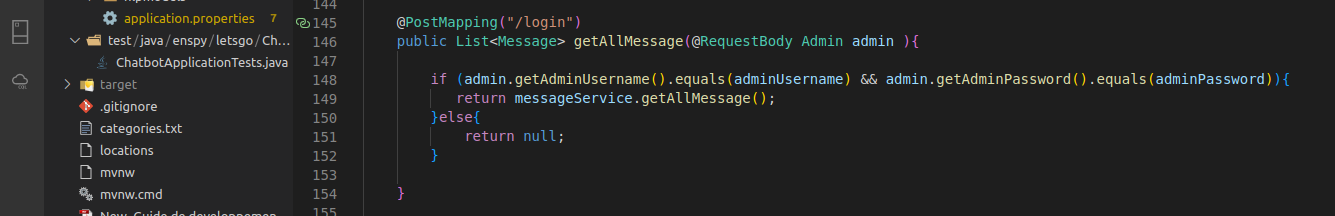
* HandleMessage()

Permet de recevoir et traiter tout message entrant sur le endpoint **“/message”** et retourne un objet JSON de type message contenant le champ Responsebody qui inclut la réponse et le champ Category qui inclut la catégorie déterminer par notre module d’intelligence artificielle.



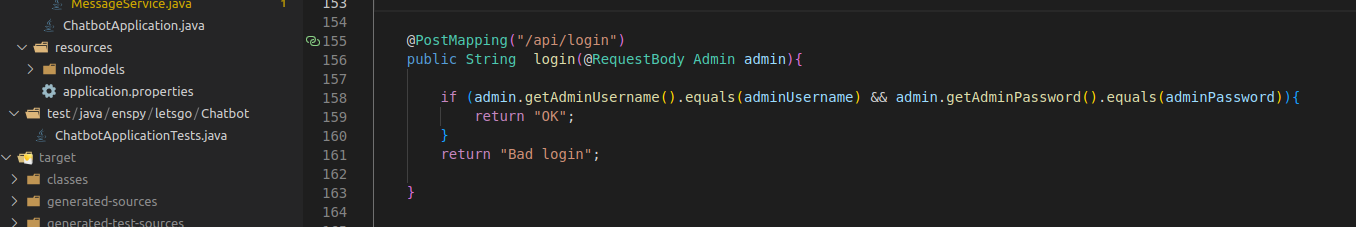
* getAllMessage()

Permet de retourner tous les messages contenus dans la base de données si les informations de connexion sont correctes ou un *null* si l’une des informations est fausse.



* login()

Retourne “OK” si les informations de connexion contenues dans le payload sont correctes et “bad login” si l’une de ces informations est erronée.



#### Le package utilitaire utils

Il contient les classes utilitaires locationChecker.java et locationUtils.java qui permettent de vérifier si la destination entrée par l’usager est connue de l’application et en cas d’erreur, il lui fait des suggestions de destination dont l’orthographe se rapproche au maximum de la destination entrée par le client. L’approche pour déterminer cette destination proposée est basée sur la distance de Levenshtein

.

# GUIDE D’INSTALLATION DE L’ENVIRONNEMENT DE DEVELOPPEMENT

## Technologies Requises

* **VS code (ou IntelliJ) :** C’est un éditeur de code source léger et puissant, développé par Microsoft. Il est conçu pour être hautement personnalisable et adaptable à différents langages de programmation et flux de développement, téléchargeable à travers le site officiel <https://code.visualstudio.com/>. Comme alternative, on peut utiliser IntelliJ, Eclipse, Spring tools suite.
* **Docker**: Pour créer les images et conteneur du service <https://www.docker.com/>.
* **Redis Insight**: Pour visualiser les données de la base de données <https://redis.io/>.
* **Java JDK 17**: kit de développement pour Java téléchargeable à travers le lien officiel <https://www.oracle.com/java/technologies/javase-jdk17-downloads.html>.
* **Cassandra**: Apache Cassandra est un système de gestion de base de données distribuée téléchargeable à travers le site officiel : <https://cassandra.apache.org/>

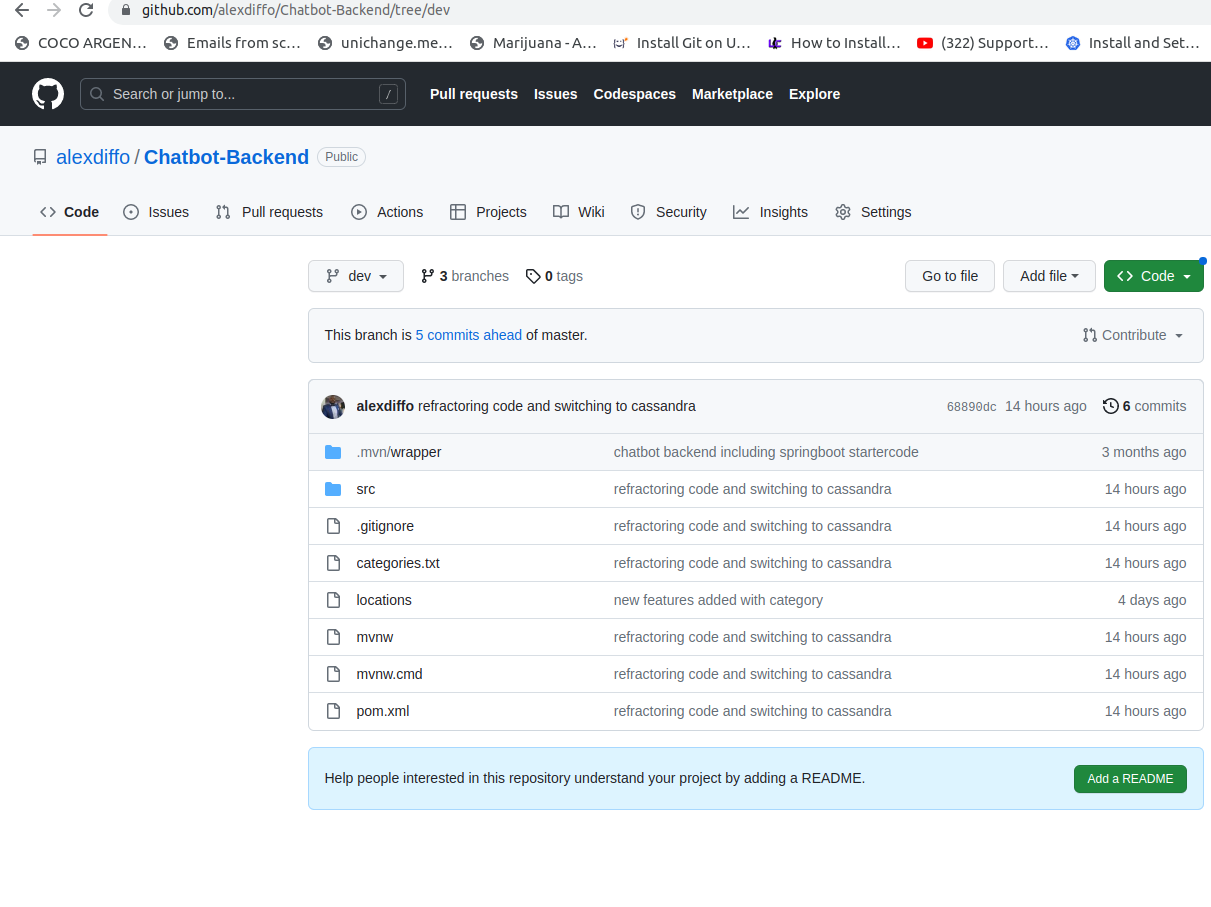
## Prise en Main du projet

Pour mettre en place l’environnement de développement, il faudra suivre rigoureusement les différentes étapes suivantes :

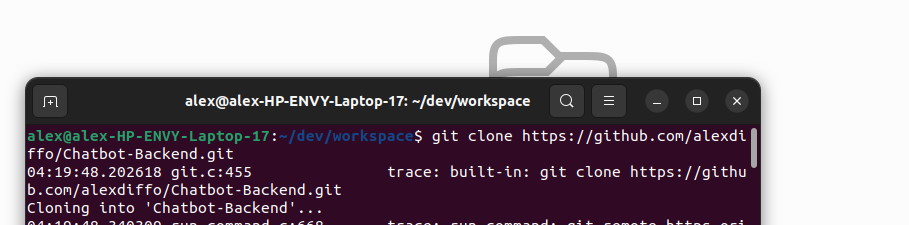
1. *Cloner le Projet*

Pour cloner le projet, il faut :

* Se rendre sur le dépôt Github du microservice qui se trouve à l’adresse [https://github.com/alexdiffo/ChatBot-Backend.git](https://github.com/alexdiffo/Chatbot-Backend.git) La dernière version qui est encore au stade de développement se trouve sur la Branche Dev.

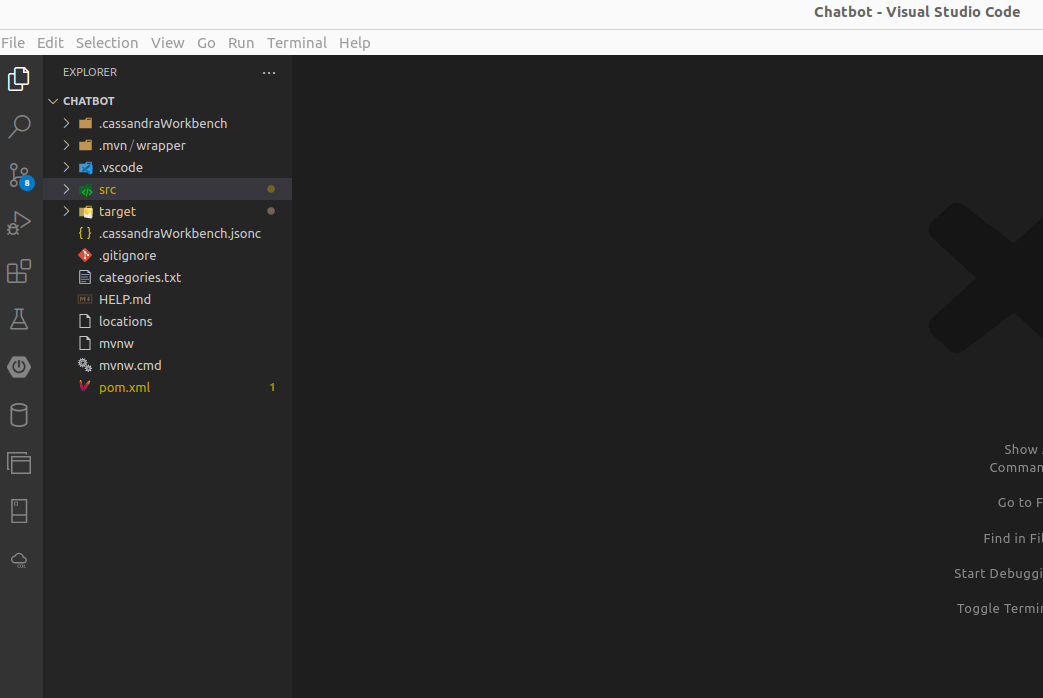


* Ouvrir le terminal en se positionnant sur le repertoire local du workspace et cloner le projet localement à travers la commande: “g**it clone** [**https://github.com/alexdiffo/ChatBot-Backend.git**](https://github.com/alexdiffo/Chatbot-Backend.git)**”**

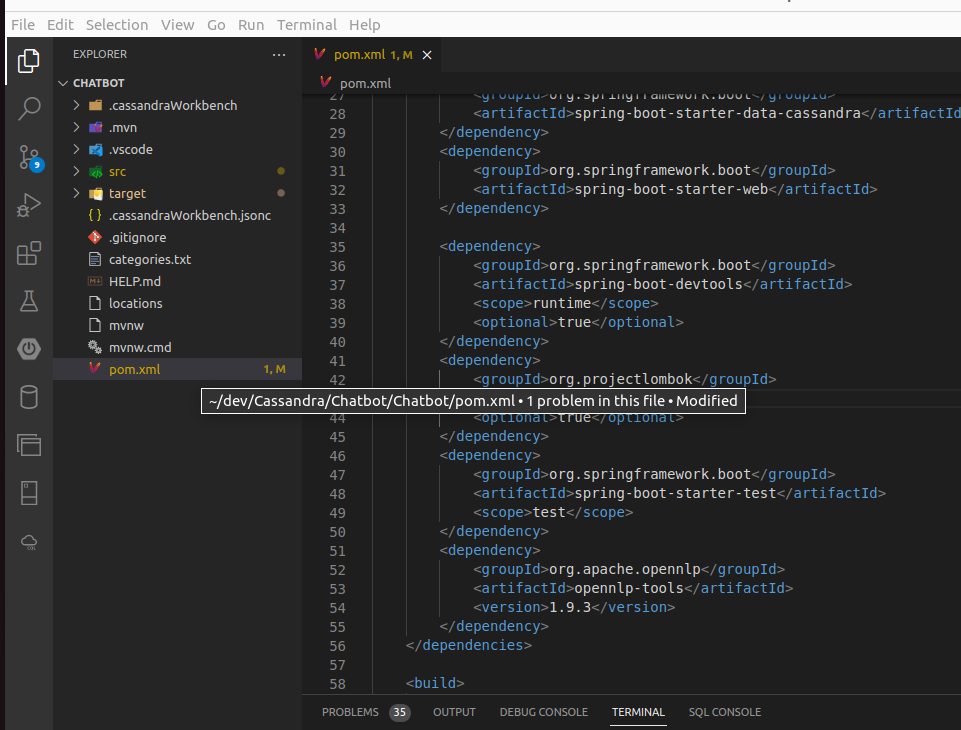


1. *Mettre en place l’environnement de développement*

Une fois le clonage terminé, naviguer pour se positionner à la racine du dossier du projet à travers la commande **“cd chatBot “** puis lancer l’IDE Vscode à travers la commande **“code . “** , ce qui aura pour effet d’ouvrir VS code en chargeant le repository du projet.



* Actualiser le fichier POM xml pour télécharger les dépendances Maven. Au préalable, il faut se rassurer d’être connecté à internet :

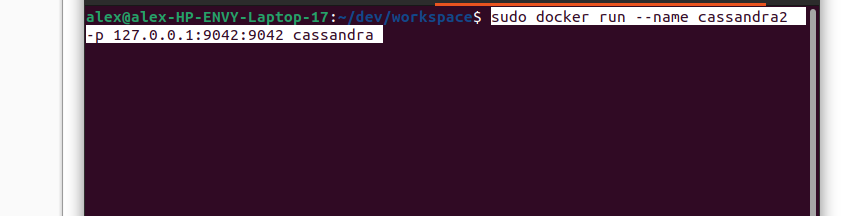


1. *Installer la base de données*

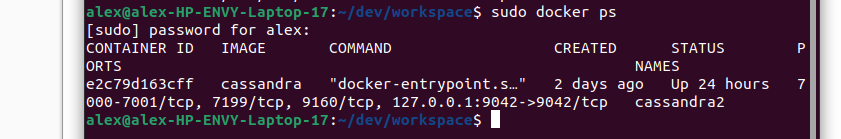
Le système utilisé pour cette démonstration est Linux Ubuntu

* Installer et démarrer docker
* Lancer l’image docker de Cassandra dans un conteneur à travers la commande docker

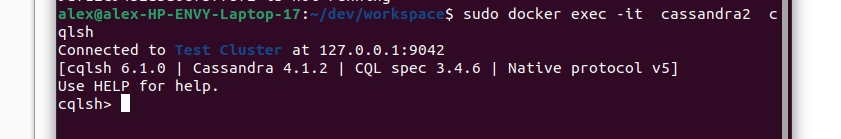
**“docker run --name cassandra2 -p 127.0.0.1:9042:9042”** pour caster sur le port 9042 en local (où cassansdra2 est le nom du conteneur).



* Vérifier à travers la commande **“docker ps”** que le serveur est bien lancé ; L’ID du conteneur de Cassandra doit s’afficher explicitement ainsi que son nom.



* Se connecter au shell CQLSH de cassandra pour effectuer des requêtes sur la base de données à travers la commande : **“sudo docker exec -it cassandra2 cqlsh “**



* Activer le keyspace à travers la commande **“use chat\_bot”** pour pouvoir effectuer des requêtes.
* Créer le Keyspace correspondant au nom de la base de données que vous souhaitez utiliser à travers la requête

**“CREATE KEYSPACE chat\_bot**

**WITH replication = {'class': 'Strategy', 'replication\_factor': N};”.**

Dans ce cas, *chat\_bot* est le keyspace name choisi.

* Créer la table **message** à travers la requête :

-- change table name and structure

**“CREATE TABLE chat\_bot.Message(**

**messageid text PRIMARY KEY,**

**messagebody text,**

**responsebody text,**

**senderusername text,**

**category text,**

**event\_time TIMESTAMP**

**);”**

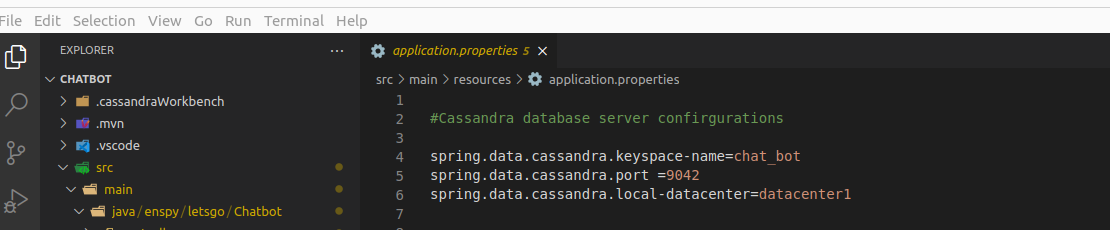
* Confirmer à travers la commande CQLSH **“describe tables”** que la table a bien été créée ; le nom de la table doit s’afficher explicitement



1. *Configurations de base ide l’application*

* Pour l’accès à la base de données :

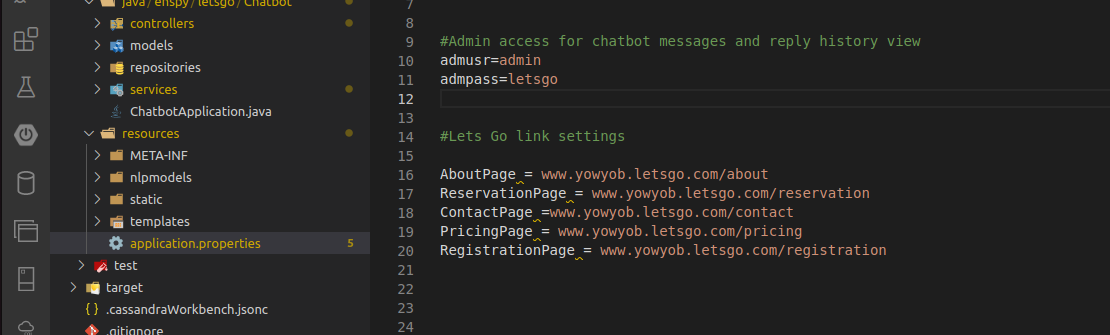
Créer le fichier application.properties(s’il n’existe pas) dans le repertoire ressources src/main/ressources et renseigner les informations sur le keyspace, ainsi que le port correspondant au port de casting Cassandra :



* Configuration d’exploitation de l’application :

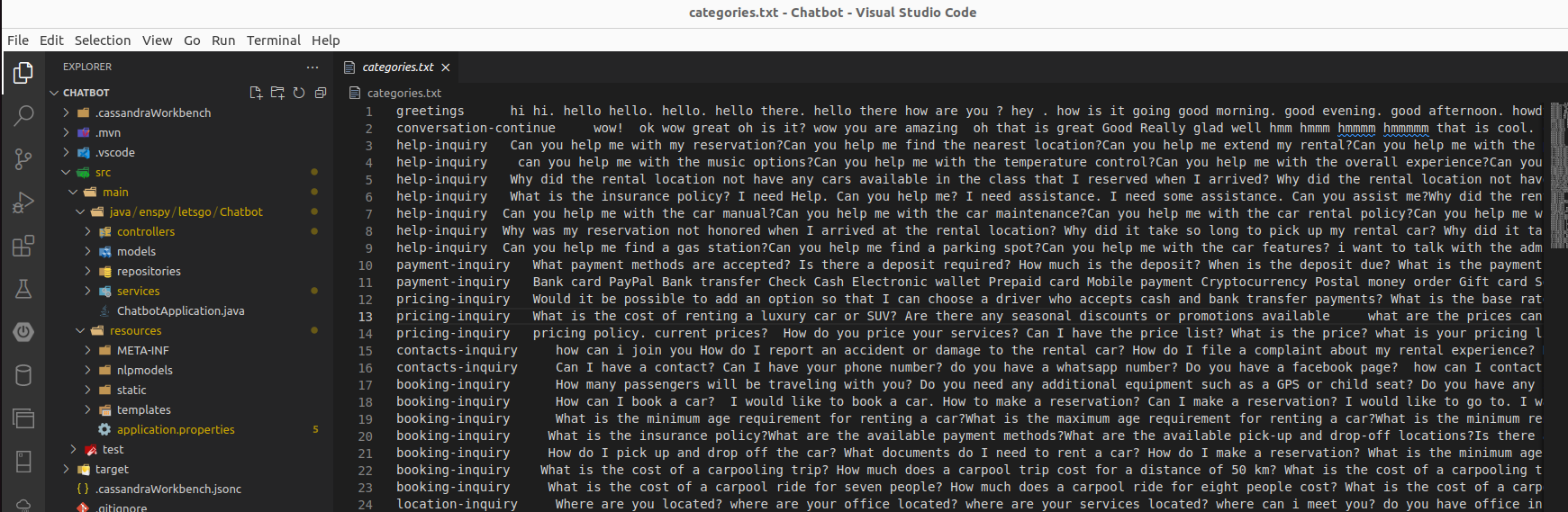
Il faut renseigner les informations suivantes dans le fichier application.properties :

* **admusr** et **admpass** doivent correspondre respectivement aux username et password de l’administrateur qui souhaite visualiser les messages enregistrés dans la base de données.
* **AboutPage, ReservationPage, ContactPage, PricingPage, RegistrationPage** doivent respectivement contenir les liens publics internet qui pointent vers toutes ces pages.



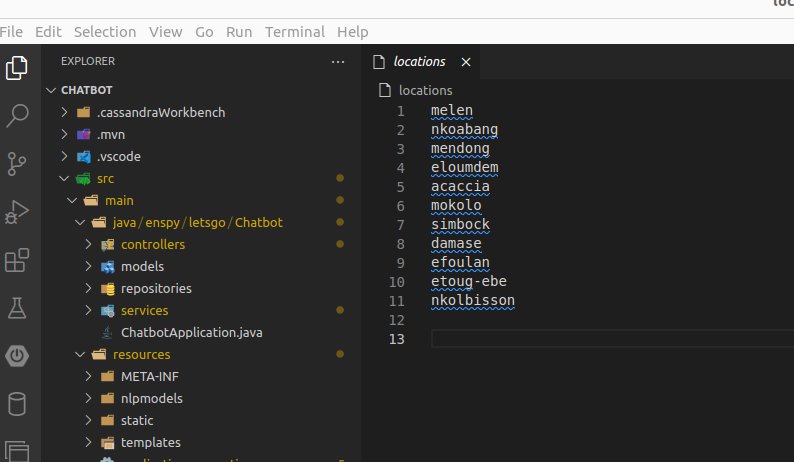
1. *Les fichiers d’exploitation*

* Le fichier de catégorisation

A la racine du dossier du projet, se trouve un fichier nommé categorisation.txt, qu’il faut compléter avec les catégories pertinentes, qui permettent d’entrainer le modèle d’intelligence artificielle afin de bien classifier le message reçu pour y répondre de manière adéquate.

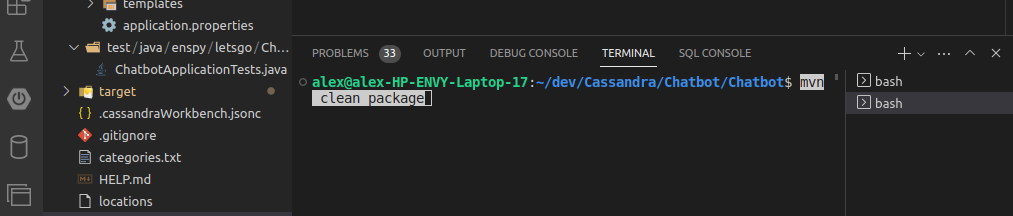
* Le fichier des points de destination possibles

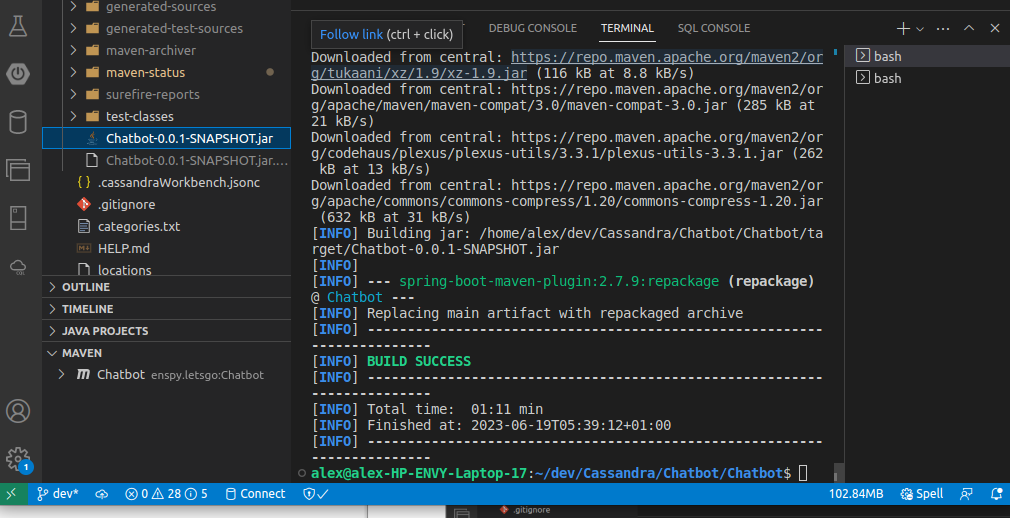
Il se trouve également à la racine du projet. Il faut le compléter avec les lieux de tous les points qui peuvent correspondre aux destinations des clients de *Let’s Go.*



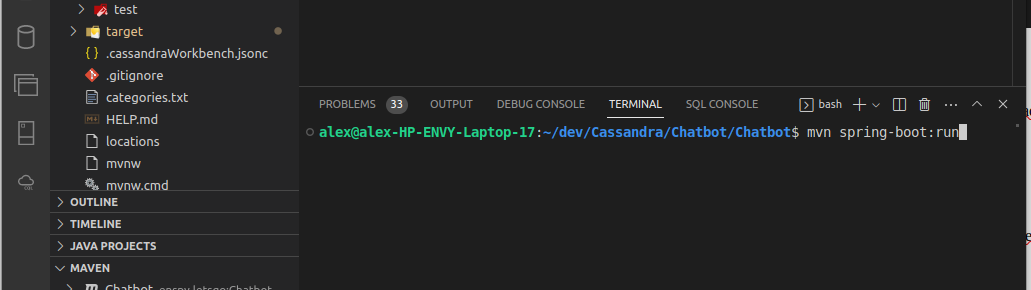
1. *Démarrer l’application*

* Effectuer le build à travers la commande mvn **“clean package”** ce qui aura pour effet de générer le Jar de l’application dans le repertoire et vous recevrez un message de succès si le build réussi. En cas d’échec du build, verifier les configurations.

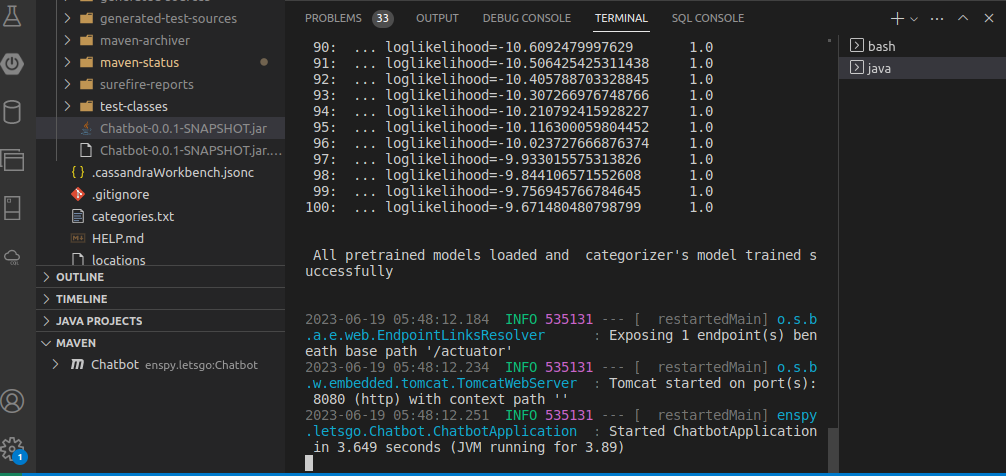




* L’application peut être démarrée à travers la commande **“mvn spring-boot:run”** exécutée dans le terminal de l’IDE.



* Une fois l’application lancée vous devez recevoir dans le terminal les messages confirmant que les modèles préentrainés ont bien été chargés et que le modèle de categorisation a été bien entrainé.



## Création de L’image Docker du service

Pour créer l’image Docker du service, ouvrir le terminal, se rendre dans le dossier du projet et lancer la commande **docker-compose up** (rassurez-vous d’avoir au moins 2Go de data et une connexion stable) pour créer le conteneur et l’image docker du service.

# ARCHITECTURE D’INTEGRATION DES DIFFERENTS MICROSERVICES

## Le frontend :

Nous implémenterons le frontend de notre chatBot avec le Framework javascript React qui a l’avantage d’être simple, rapide et évolutif.

## Format d’échange -JSON :

La communication entre le frontend et le backend se fera via le format d’échange JSON.

Notre application communiquera avec l’API de translation des messages et en cas de besoin, avec l’API de *Let’s Go* via le format d’échange JSON.

## ChatBot service :

Le microservice chatBot service sera une API basée sur Springboot dont la principale fonctionnalité sera de classifier le message à l’aide de certains modèles pré-entrainés offerts par la librairie Apache OpenNLP et d’un modèle d’IA que nous entrainerons avec un jeu de données conforme aux questions récurrentes des utilisateurs afin qu’il catégorise efficacement les questions et produise des réponses adéquates.

## Base de données CASSANDRA

Les messages et leurs catégorisations produites par le chatBot doivent être conservés afin d’évaluer quotidiennement l’efficacité du chatBot et le réentraîner avec les questions supplémentaires posées par les utilisateurs et qui n’avaient pas été prises en compte précédemment. D'où le choix d’une base de données nosql, en occurrence Cassandra.

## API let’sgo

Pour répondre à certains messages, le chatBot pourra éventuellement avoir besoin de récupérer des données en provenance d’un microservice de l’API *Let’s Go* afin de produire une réponse satisfaisante.

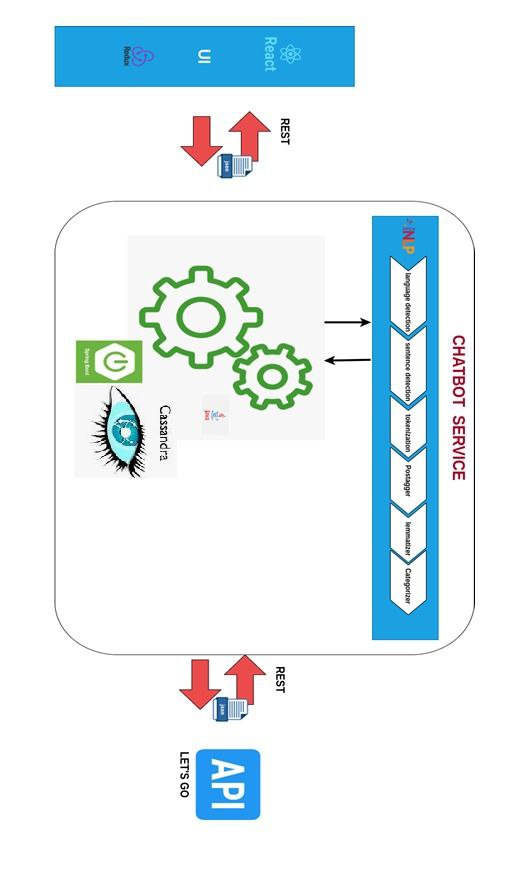


Figure : architecture microservice

# GUIDE D’INTEGRATION (Avec Let’s Go)

## Description

L'API ChatBot\_Let’s Go permet d'interagir avec un chatBot conversationnel via des requêtes HTTP. Cette API a été conçue avec une architecture MVC basée sur Springboot comme framework JEE et Cassandra comme base de données.

## Les Endpoints exposés

### POST /api/messages

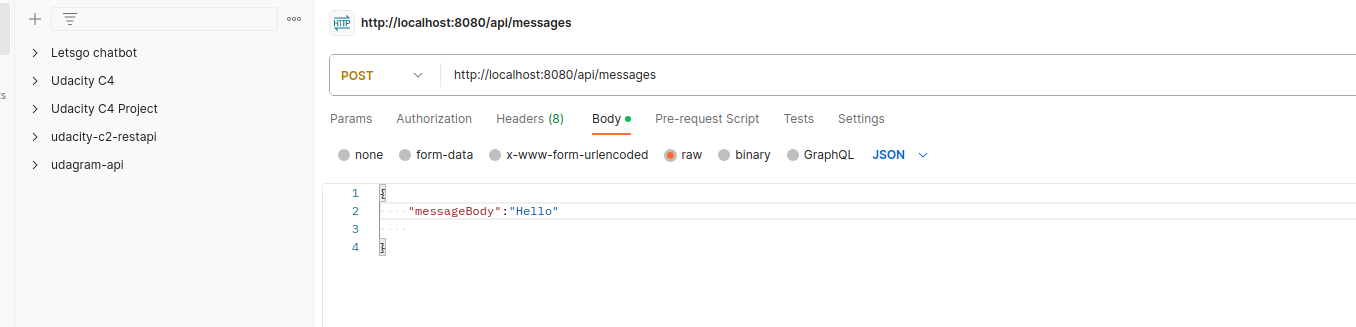
Permet d'envoyer une requête pour interagir avec le chatBot conversationnel.

#### Paramètres de la requête :

Aucun

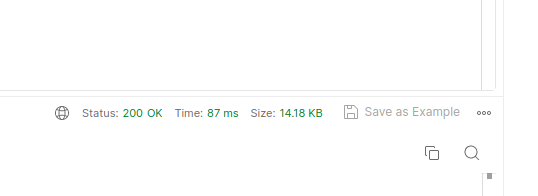
#### Corps du message :

Le request-body contient un message au format JSON



#### Statut réponse :

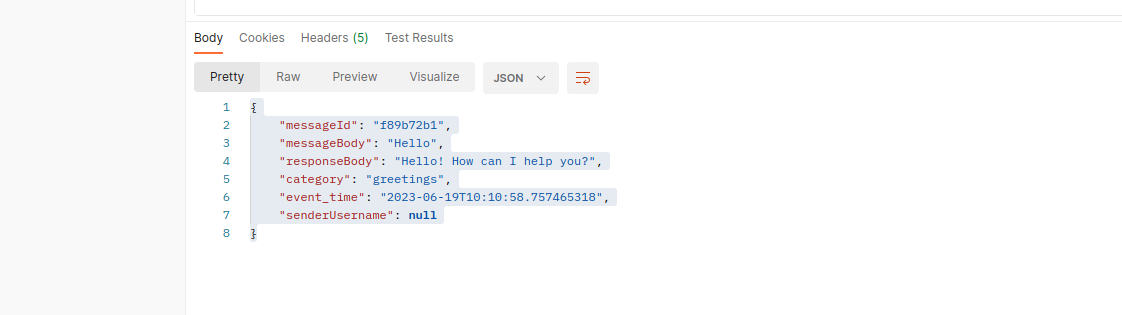
* 200 OK : La requête a réussi. Le chatBot a traité le message et renvoie une réponse.



* 400 Bad Request : La requête est mal formée ou des paramètres sont manquants.
* 500 Internal Server Error : Une erreur interne s'est produite lors du traitement de la requête.

#### Corps de la réponse :

La réponse reçue de l’API est au format JSON comme dans cet exemple :



* messagebody (chaîne de caractères) : La réponse du chatBot.
* responseBody (chaîne de caractères) : C’est le contenu de la réponse du chatBot.
* category (chaîne de caractères) : Contient l’intention de l’usager qui a envoyé le messages.
* event\_time : Contient l’heure d’envoi du message.
* senderUsername : Le nom d’utilisateur de la personne qui a envoyé le message.

### Post /login

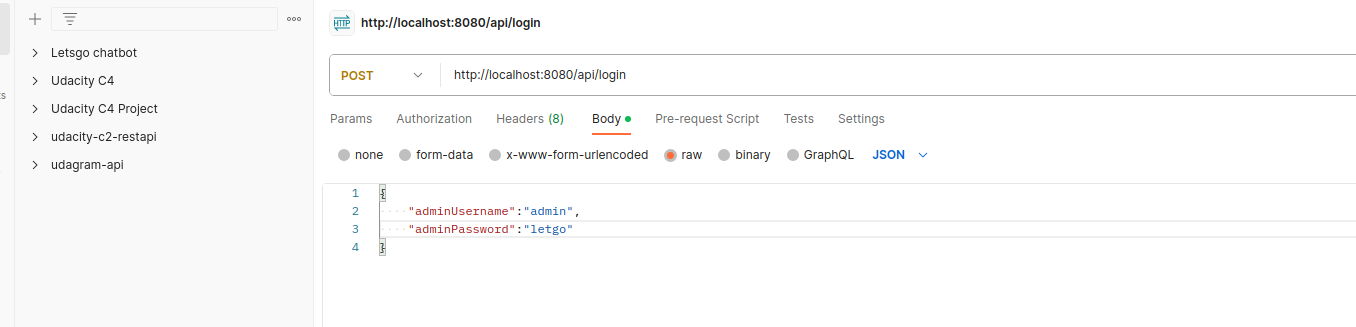
Permet d'envoyer une requête pour authentification

#### Paramètres de la requête :

Aucun

#### Corps du message :

Le request-body contient un message au format JSON

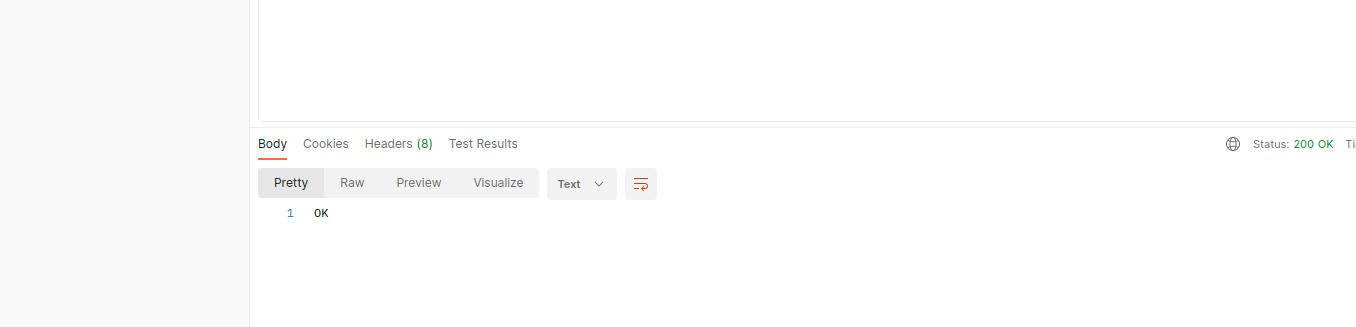


#### Statut réponse :

* 200 OK : La requête a réussi. Le chatBot a traité le message et renvoie une réponse.
* 400 Bad Request : La requête est mal formée ou des paramètres sont manquants.
* 500 Internal Server Error : Une erreur interne s'est produite lors du traitement de la requête.

#### Corps de la réponse :

La réponse reçue de l’API est la chaine de caractère “OK” si l’authentification about it et “bad login” sinon.



### GET /api/messages-all

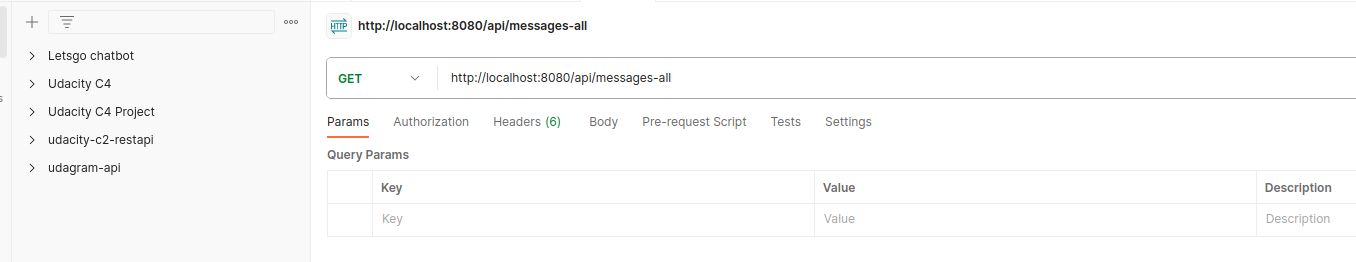
Permet d'envoyer une requête pour interagir avec le chatBot conversationnel.

#### Paramètres de la requête :

Aucun

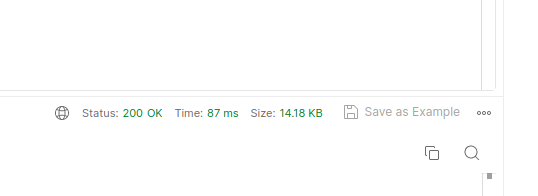
#### Corps du message :

Le request-body contient un message au format JSON



#### Statut réponse :

* 200 OK : La requête a réussi. Le chatBot a traité le message et renvoie une réponse.



* 400 Bad Request : La requête est mal formée ou des paramètres sont manquants.
* 500 Internal Server Error : Une erreur interne s'est produite lors du traitement de la requête.

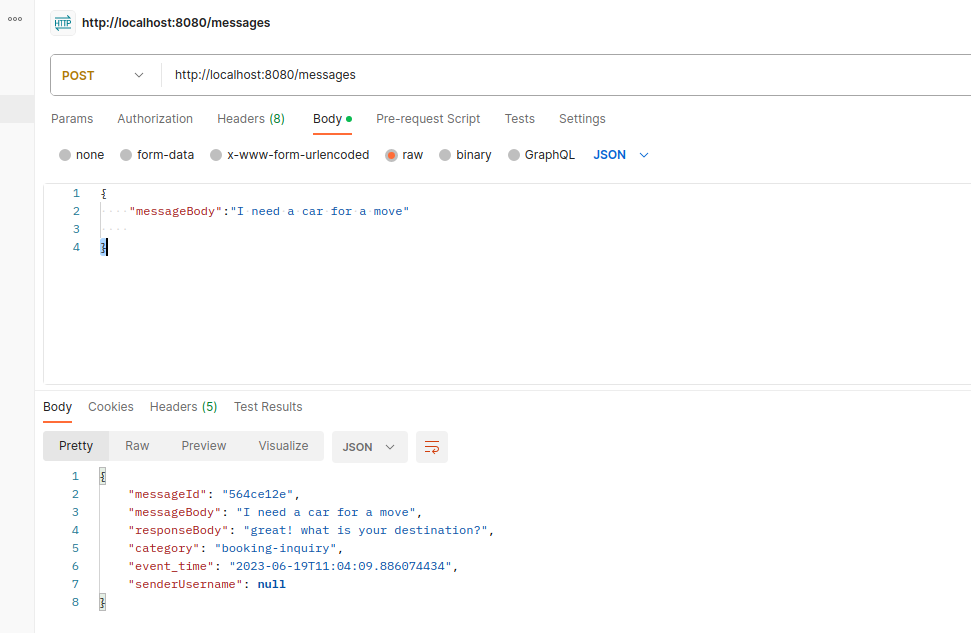
#### Corps de la réponse :

La réponse reçue de l’API est au Format JSON comme dans cet exemple



Toutes les clé-valeurs contenues dans cette réponse ont été expliquées de façon détaillée plus haut.

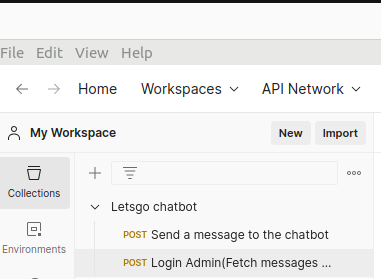
## Exemple d’utilisation



Dans cet exemple, l'utilisateur envoie une requête POST au endpoint /messages avec le message "I need a car for my move". Le chatBot traite la requête, identifie l'intention "booking-inquiry" et renvoie la réponse "great! What is your destination". Ce contrat d'API fournit une base pour l'intégration d'un chatBot dans une application cliente, en spécifiant les paramètres de requête attendus, les réponses possibles et les codes de statut correspondants. Il permet aux développeurs d'interagir avec le chatBot de manière cohérente et prévisible.

## Collection de tous les endpoints édités avec des exemples :

Une collection de tous les endpoints avec des exemples se trouve à la racine de la repository à travers le fichier nommé **Letsgo chatBot.postman\_HttpRequest\_collection.json**. Ouvrir ce fichier avec postman pour effectuer les tests nécessaires.



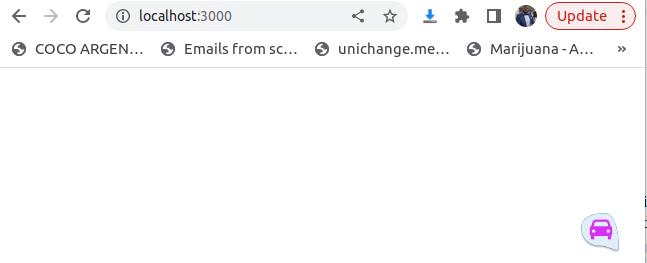


# GUIDE D’UTILISATION

## Accueil du Chatbot

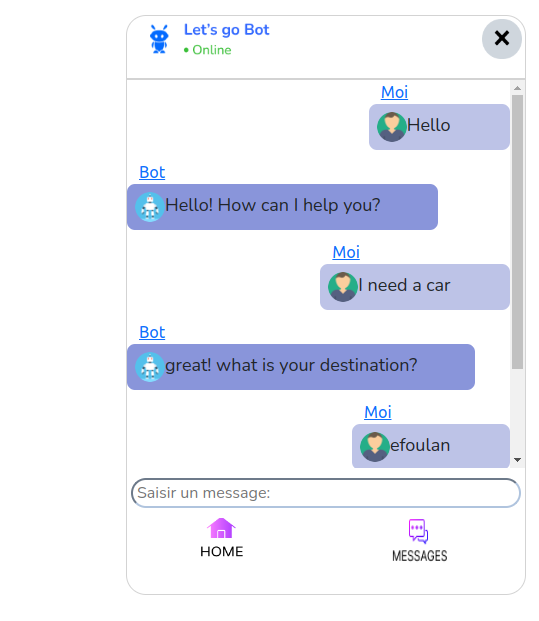
* **Icône de lancement**

Une fois que notre assistant conversationnel intelligent sera intégré au site de letsgo, il apparait au bas de l’écran (angle côté droit), une icône au travers de laquelle les utilisateurs pourront ouvrir l’assistant



* **Fenêtre d’accueil**

Lorsqu’on clique sur l’icône mentionnée précédemment, la fenêtre de conversation se déploie et l’utilisateur peut interagir avec le chatbot en posant des questions auxquelles il répondra spontanément:

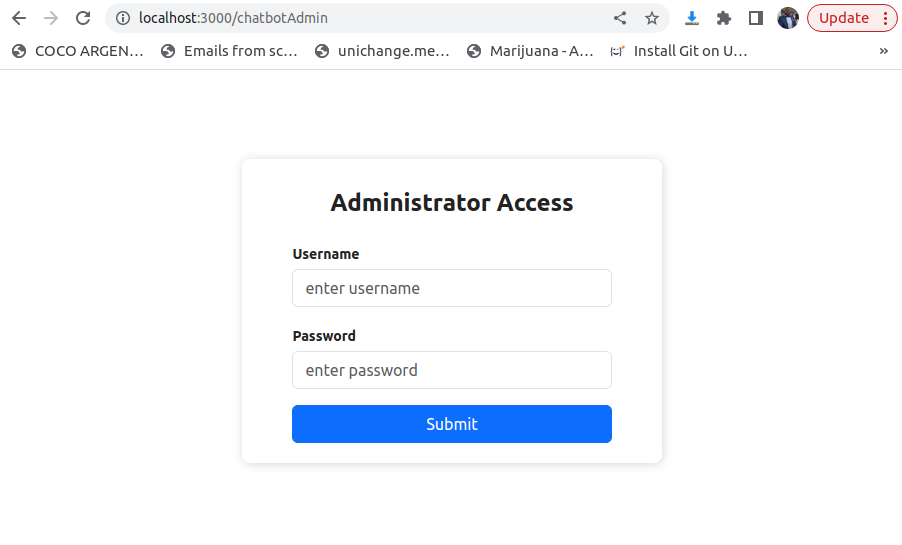


## Administration du Chatbot

L’administration du chatbot nous permet de consulter les messages échangés entre l’utilisateur et le chatbot en vue d’apprécier sa performance. Si le chatbot affiche des défaillances pour certaines questions, elles devront être répertoriées et ajoutées au jeu de données dans la bonne catégorie pour réentraîner notre modèle d’IA, afin qu’il puisse désormais permettre de bien répondre aux questions à venir.

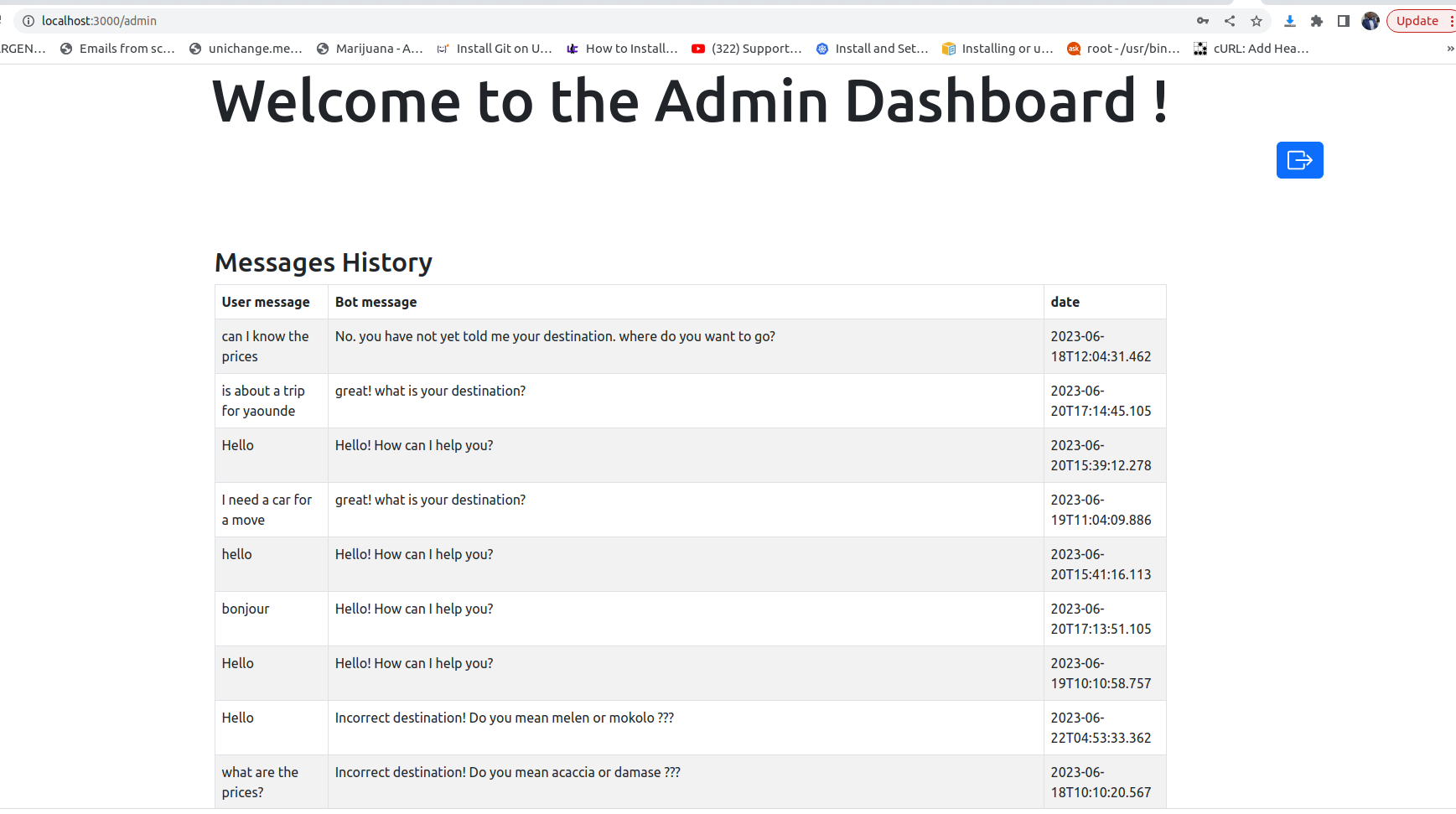
* **Admin login page**

A travers le end-point **/adminchatbot** , nous avons accès à la page de login de l’administrateur :



* **Admin Dashbord**

Une fois la connexion réussie, l’administrateur accède à son panel où il peut consulter l’historique des messages échangés entre les visiteurs et le chatbot en vue d’évaluer la performance du chatbot



# CONCLUSION

En conclusion, la réalisation de ce chatBot a permis de créer un système interactif et intelligent capable de fournir une assistance conviviale aux utilisateurs de l’application de covoiturage *Let’s Go*. Il convient de noter que le chatBot peut être amélioré et affiné en continu. Des itérations supplémentaires peuvent être effectuées pour enrichir la base de connaissances du chatBot, améliorer les modèles de langage et d’extraction d’entités et affiner les algorithmes de décision pour des réponses encore plus précises.

# Bibliographie

s.d. https://www.baeldung.com/apache-open-nlp consulté le 15/05/2023 a 14h36

s.d. https://opennlp.apache.org/release-model.html consulté le 12/04/2023 à 10h30

s.d. https://en.wikipedia.org/wiki/Apache\_OpenNLP consulté le 26/05/2023 à 20h42

s.d. https://www.tutorialkart.com/opennlp/apache-opennlp-tutorial/ consulté le 13/02/2023 à 22h06

s.d. https://mvnrepository.com/artifact/org.apache.opennlp 06/03/2023 à 19h30

s.d. https://www.youtube.com/watch?v=2\_uz5eYgdq8.

s.d. https://www.youtube.com/watch?v=rag1Rv\_22jQ.

s.d. https://www.youtube.com/watch?v=miPv34BeFEQ.

s.d. https://www.youtube.com/watch?v=wCXjxfE0F2o.

s.d. https://www.youtube.com/watch?v=y4jOEWFOtd8.