CDI Langchain4j

Avant d'entrer dans le dur du sujet (live coding d'une build compatible extension déroulé par Yann)

Je vais vous présenter rapidement le projet sur lequel nous allons nous appuyer pour effectuer ce travail.

L’application s’appelle Helidon car booking et à été créé par Jean-François james.

(Slide stack :

* Java 22 (Temurin OpenJDK distro)
* Helidon 4.0.7
* Helidon CLI 3.0.4
* Maven 3.9.5
* LangChain4j 0.30.0
* LLM provider (Azure for now)

)

Cette application, est en java et comme son nom l’indique développée avec un conteneur hélidon

Bien évidemment on a le tooling qui va avec soit helidon CLI + maven

Et enfin langchain4j pour tester un peu les capacités des llm ainsi que la facilité ou non d’intégration dans java

Pourquoi cette app ? Jean françois james à fait un talk a jchateau sur une app quarkus qui s’appuyait sur l’extension quarkus-langchain4j

Il a souhaité créer cette même app sans l’extension pour bien comprendre les bienfait de celle-ci.

A noter que, l’extension quarkus langchain4j possède environ 600 classes pour 400 pour la lib langchain4j, donc on peut dire que quarkus à fait une intégration en profondeur. Peut être causé par la nécessité d’être natif ?

Elle fourni 2 services de chat, un premier qui va permettre aux clients de la société miles of smiles de gérer leurs réservations de voiture ou d’en effectuer

Et le deuxième service offre un système de détection de fraude

Qui peuvent être appelés via 2 API rest

Par simplicité, l’application ne dispose pas de bdd

Je vous propose donc de rentrer plus en détail dans l’application :

## CarBookingResource

On va commencer par le commencement, la classe CarBookingResource

Cette classe contient les deux routes d’API qui nous permettrons de générer le swagger à l’aide d’openAPI pour pouvoir tester nos deux services d’ia

Comme vous pouvez le voir, chaque route appelle son propre service d’ia qui à été injecté spécifiquement

Je vous propose donc d’entrer plus en détail sur nos deux AiServices

## ChatAiService

Cette classe va représenter nos échanges avec le llm afin de gérer les réservations d’un client

Selon la spec langchain4j, un des moyen pour interagir avec un llm est de fournir une interface, contenant une ou plusieurs fonction selon notre besoin. Langchain4j s’occupera de générer un objet, autrement dit proxy qui implémentera votre interface.

Sur cette fonction, nous pouvons ajouter tout un tas d’annotations :

* + @timeout

Comme son nom l’indique.. l’appelle de cette méthode partira en timeout au bout de 5mn

* + @retry

Le nb de rety.. mais pas pour des exceptions normales et prévues que nous pourrions throw plus tard

* + @fallback

En cas d’exception non prévue

* + @SystemMessage

L’annotation la plus intérressante, elle va permettre de soit mettre en contexte, soit donner des instructions au llm avant chaque réponse qu’il va générer

Là par exemple on lui fait un petit brief sur qui il est

Sur les règles de sécu, qu’il doit demander à chaque fois le nom et nom de famille avant de donner des détails sur un booking (ce qu’il ne fait pas btw :D)

Qu’il ne doit pas raconter sa vie et discuter uniquement de la companie

Ainsi que des règles concernant l’annulation de réservation qui pourrait être demandé par un client

NB : vous remarquerez que normalement dans une annotation @UserMessage quelquechose entre {{}} signifie que le proxy qui sera généré par langchain injectera l’argument de votre fonction à la place de ce placeholder

Sauf que {{current\_date}}, time et datetime sont réservé pour l’injection de LocalDate, Time DateTime et peuvent donc être utilisés dans systemMessage

## ChatAiFactory

En voyant cette classe je ne sais pas si certains d’entre vous commencent à voir où va-t-on 😊

Maintenant que nous avons créé notre interface, qui spécifie nos interactions avec le llm pour le chat, avec les instructions ainsi que la gestion d’erreur, nous allons devoir créer une factory qui va nous permettre de créer le service au runtime avec les configurations requises.

* + chatLanguageModel

Bien évidemment, avant de pouvoir utiliser un LLM il va falloir que l’on choisisse lequel nous voulons utiliser, le configurer et le donner à langchain pour qu’il puisse le requêter.

* + chatMemory

Demande a langchain de garder en mémoire les N derniers messages peut importe le nombre de token qu’ils consomment.

Ensuite, on va s’intéresser à deux champs qui sont optionnels mais cruciaux dans notre utilisation du llm.

Je ne sais pas si vous souvenez mais dans notre chatService, le chatbot est censé répondre aux question des clients concernant leurs réservations de voiture ainsi que, si besoin, effectuer une réservation pour eux.

Mais comment le llm pourrait-il avoir accès à notre système que ce soit pour checker les réservation, en créer ou en annuler ?

C’est là que le tool entre en jeu

* + tools

BookingService contient plusieurs choses que nous aurions pu séparer. Mais soit

Il contient la liste des bookings mocké pour ne pas avoir besoin de bdd

Mais ensuite il contient 3 fonctions annotées @Tool

Cela nous permet d’offrir au llm ces fonctions, qu’il pourrait appeler s’il en a besoin

Par exemple si un customer demande la liste de ses bookings, le llm devrait reconnaitre que la fonction getBookingsForCustomer, de par son annotation, pourrait lui permettre de retrouver les bookings pour ce customer donné et va donc l’appeler afin de retourner les bookins au customer

Cela nous permet donc d’offrir au llm des fonctions pour tout un tas de traitement ou d’accès à nos données.

Et pour finir, si l’on revient sur notre chatAiService, souvenez-vous que notre llm est censé pouvoir gérer les réservations, que ce soit les faire ou les annuler, mais pour se faire, il a besoin de connaitre la liste des voitures ainsi que la politique d’annulation.

Sauf que ceux-ci se trouvent dans des fichiers.

C’est pour cela que nous allons avoir besoin d’un RAG (Retrieval-Augmented Generation)

* + contentRetriever

je ne vais pas rentrer dans les détails de ce qu’est un RAG, car j’en sais rien.

Ce que je sais juste c’est que ça nous permet d’entrainer le llm sur des documents afin de développer ses connaissance d’un domaine spécifique, là par exemple les termes d’utilisation de la société miles of smiles, etc..

Pour bénéficier d’un COntentRetriever, il va nous falloir déterminer de quelle manière nous voulons stocker les documents et dans quel type de Store.

Une fois que nous avons cela, il nous faut utiliser un