

AGENTIC

**SPRING BOOT
LANGCHAIN4J**

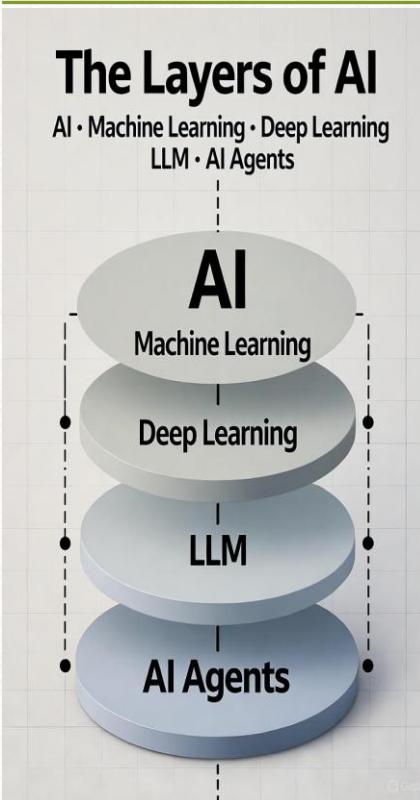
Par Ibrahima DIAVE – Manager en SI
Octobre 2025

Objectifs de la formation

- Dans le cadre de cette formation pratique, nous allons développer une interface de discussion dotée d'un agent IA intelligent. L'objectif est de créer une application concrète permettant aux étudiants de comprendre comment intégrer l'IA dans une application Java moderne avec Spring Boot.
- L'agent IA sera amenée à interagir avec les utilisateurs et avec tout l'environnement de l'application (services, models, repositories, base de données, controllers...).

Un peu de théorie d'abord 

Les grandes couches de l'IA :



1. Machine Learning (ML) :

- L'IA apprend à partir de données.
- Exemples : classification d'images, prédiction de ventes, détection de fraudes.
- Algorithmes : régression linéaire, arbres de décision, etc.

2. Deep Learning (DL) :

- C'est du ML utilisant des **réseaux de neurones profonds**.
- Exemples : reconnaissance vocale, vision, traduction automatique.
- Frameworks : TensorFlow, PyTorch, Keras.

3. Large Language Models (LLM) :

- Réseaux de neurones massifs entraînés sur du texte pour comprendre et générer du langage humain.
- Exemples : GPT, Claude, Gemini, Mistral, LLaMA.
- Ces modèles **raisonnent, dialoguent, résument et créent du contenu**.

4. Agents IA :

- Les LLM deviennent **opérationnels** : ils **agissent, utilisent des outils, se souviennent, et s'adaptent**.
- C'est le niveau **intelligent + autonome** de l'IA moderne.

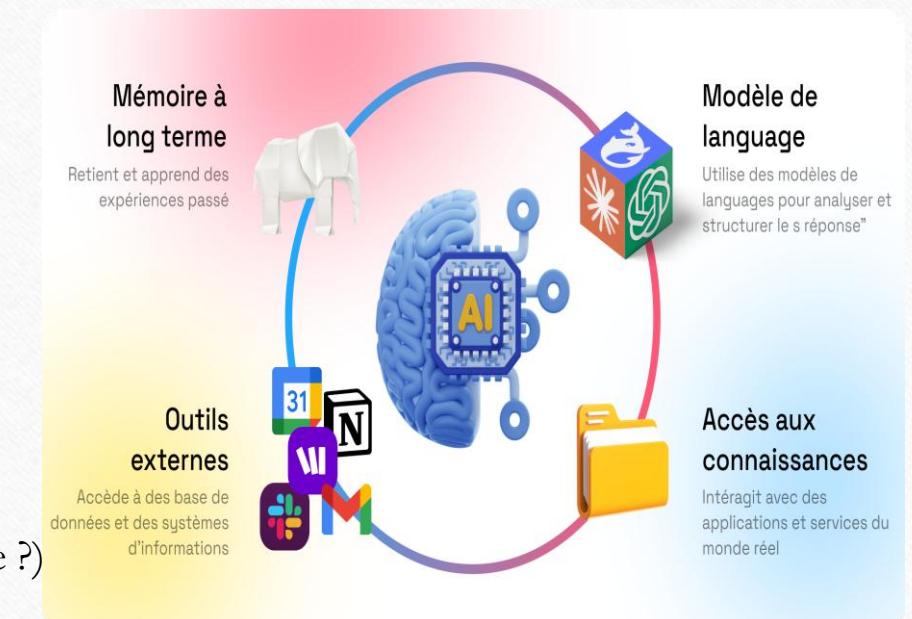
Qu'est qu'un agent IA

Modèle de langage + mémoire + outils + raisonnement.

Donner la capacité à un LLM d'interagir avec son environnement.

Composants clés :

- **LLM** : le cerveau (comprend et génère du texte)
- **Tools** : les actions qu'il peut exécuter (API, recherche, DB, etc.)
- **Memory** : conserve le contexte et les échanges précédents
- **Reasoning** : logique de décision (quel outil appeler ? comment répondre ?)



Exemples d'Agents IA

1. Un agent pour la SODAGRI qui répond sur la 7^e Lettre de mission

- Le modèle est connecté à un index vectoriel contenant le document PDF.
- L'utilisateur pose une question ("Quels sont les axes stratégiques 2025-2029 ?")
- L'agent recherche le passage pertinent et le résume.

3. Agent d'analyse commerciale

- Connecté à une base SQL des ventes.
- L'utilisateur pose : "Quels produits se vendent le mieux à Ziguinchor ce trimestre ?"
- L'agent génère une requête SQL, exécute, et résume les résultats.

2. Un agent intégré à un IDE (comme GitHub Copilot ou Cody)

- Comprend le code source du projet.
- Peut générer des tests unitaires, corriger des erreurs, ou documenter du code.
- Interagit avec les fichiers locaux.

4. Agent RH

- Peut générer une fiche de poste, planifier un entretien, ou rédiger un email.
- Intégré à un SIRH (système RH interne).
- Interagit avec Google Calendar, Gmail, et le serveur RH.

Questions/réponses

**Donnez des exemples d'agents IA que vous connaissez ou des idées
d'agents IA que vous trouvez intéressantes à développer !**

*« Si vous avez des contributions à donner pour mieux étayer les
notions abordées, n'hésitez surtout pas à participer !!! »*

Prompt engineering

Le **prompt engineering** est l'art de **formuler des instructions claires et précises** pour obtenir les meilleures réponses possibles d'un modèle d'intelligence artificielle.

Les composants principaux d'un prompt :

-  **System message** : définit le **rôle et le comportement** de l'IA (ex. : "Tu es un assistant expert en programmation").
-  **User message** : c'est la **question ou demande** de l'utilisateur (ex. : "Explique-moi le prompt engineering").
-  **Assistant message** : la **réponse générée** par le modèle.
-  **Temperature** : contrôle la **créativité** de la réponse (0 = réponse plus précise et stable, 1 = plus originale et variée).
-  **Max tokens** : fixe la **longueur maximale** de la réponse.
-  **Context / history** : correspond à la **mémoire de la conversation**, utilisée pour garder le fil logique des échanges.

Voir [Google AI Studio](#) pour exemple



Spring Boot est un Framework basé sur Spring (Framework JEE) qui simplifie le développement d'applications Java en réduisant considérablement la configuration manuelle. Il fournit une autoconfiguration intelligente, un serveur web intégré (comme Tomcat, Jetty ou Undertow), et des starters qui regroupent les dépendances nécessaires pour différents modules (Spring Web, Spring Data JPA, Spring Security, etc.).

Avec Spring Boot, il est facile de créer des applications autonomes et prêtes à déployer, ainsi que des micro-services modulaires, tout en suivant les bonnes pratiques du Framework Spring. Il permet aussi d'intégrer rapidement des bases de données, des systèmes de messagerie, ou des API externes.

LangChain4j

LangChain4j est une bibliothèque Java qui permet d'intégrer facilement des **modèles de langage (LLM)** dans des applications Java ou **Spring Boot**. C'est la version Java du framework **LangChain**, très populaire dans l'écosystème Python.

- **LangChain** (Python) : framework pour construire des agents et des chaînes de raisonnement.
- **LangChain4j** (Java) : adaptation pour l'écosystème Java/Spring.
 - ✓ **ChatLanguageModel** → interface vers GPT, Ollama, etc.
 - ✓ **Tool** → permet d'ajouter des fonctions/actions personnalisées.
 - ✓ **Memory** → stocke le contexte de conversation.
 - ✓ **Embeddings model** : convertit texte → vecteur
 - ✓ **Agent** → combine tout pour créer une entité intelligente.



Exemples

Présentation de quelques applications embarquant des agent IA qui interagissent avec le système

- Agent de correction automatique de texte
- Agent de modération de discussions dans un forum et réactif aux messages des utilisateurs s'il est tagger
- Agent qui extrait des données d'un CV pour les affichés dans un formulaire
- Agent qui analyse le contenu d'un dossier médical d'une consultation nouvelle pour proposer un pré-diagnostique, un traitement conservatoire et des analyse supplémentaires à effectuer

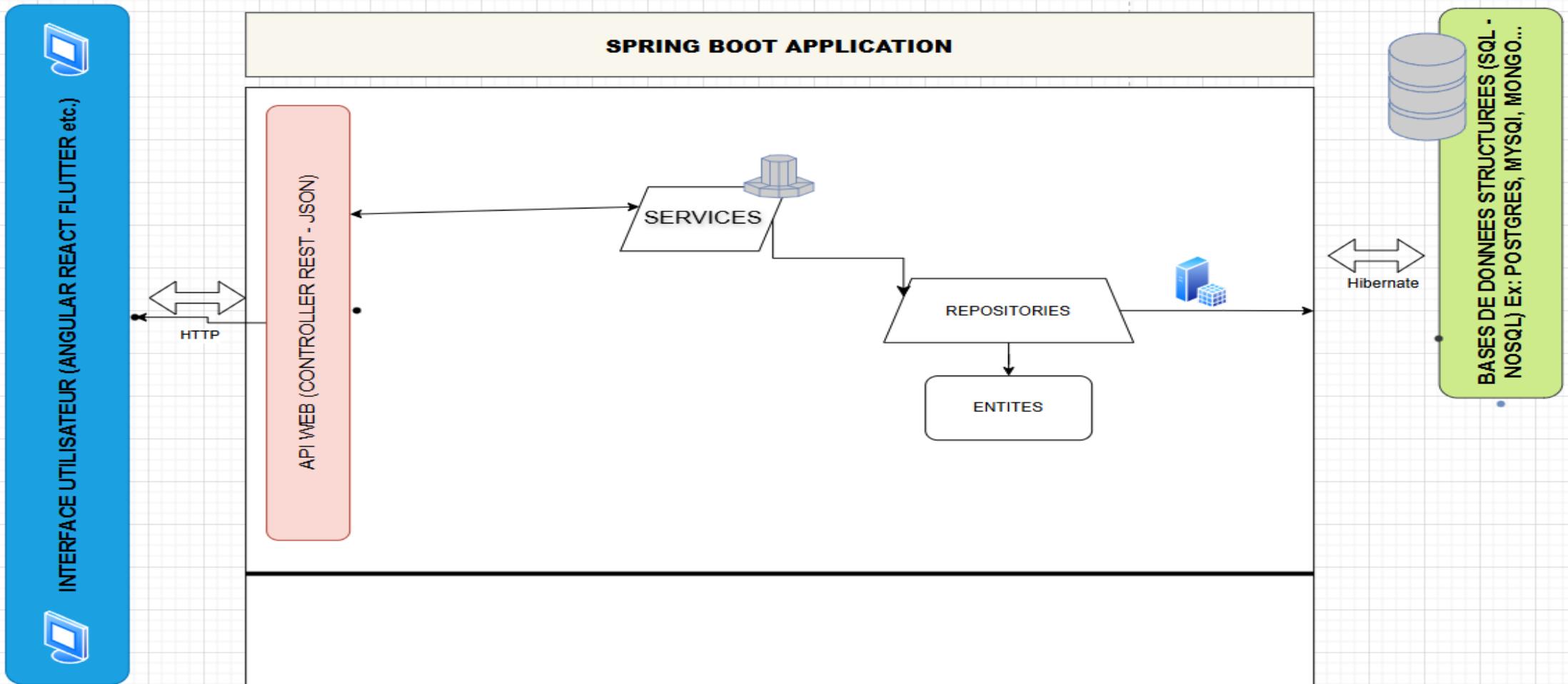
INTERACTIONS

Partagez vos avis et impressions tout en ouvrant des perspectives sur ce qu'on pourrait mettre en place comme système basé sur les agents AI afin d'améliorer la vie des étudiants au campus social !

CAS PRATIQUE SIMPLE

Nous allons développer ensemble une application Spring boot qui va permettre à des doctorants de soumettre des articles via un formulaire, l'application disposera d'un agent IA qui va à chaque fois, analyser la pertinence du contenu de l'article et lui attribuer une note entre 10 et 100, tout en proposant un résumé de l'article. Puis, une deuxième interface de l'application permettra de discuter avec un agent IA via un « chatbot » pour analyser les articles. *Si le temps nous le permet, on va mettre en place un système de traitement de documents qui permettra de lire le contenu d'un fichier pdf, de le stocker dans un base de données vectorielle et d'utiliser un agent IA pour répondre à des questions relatives au contenu du fichier en question.*

ARCHITECTURE APPLICATION SPRING BOOT



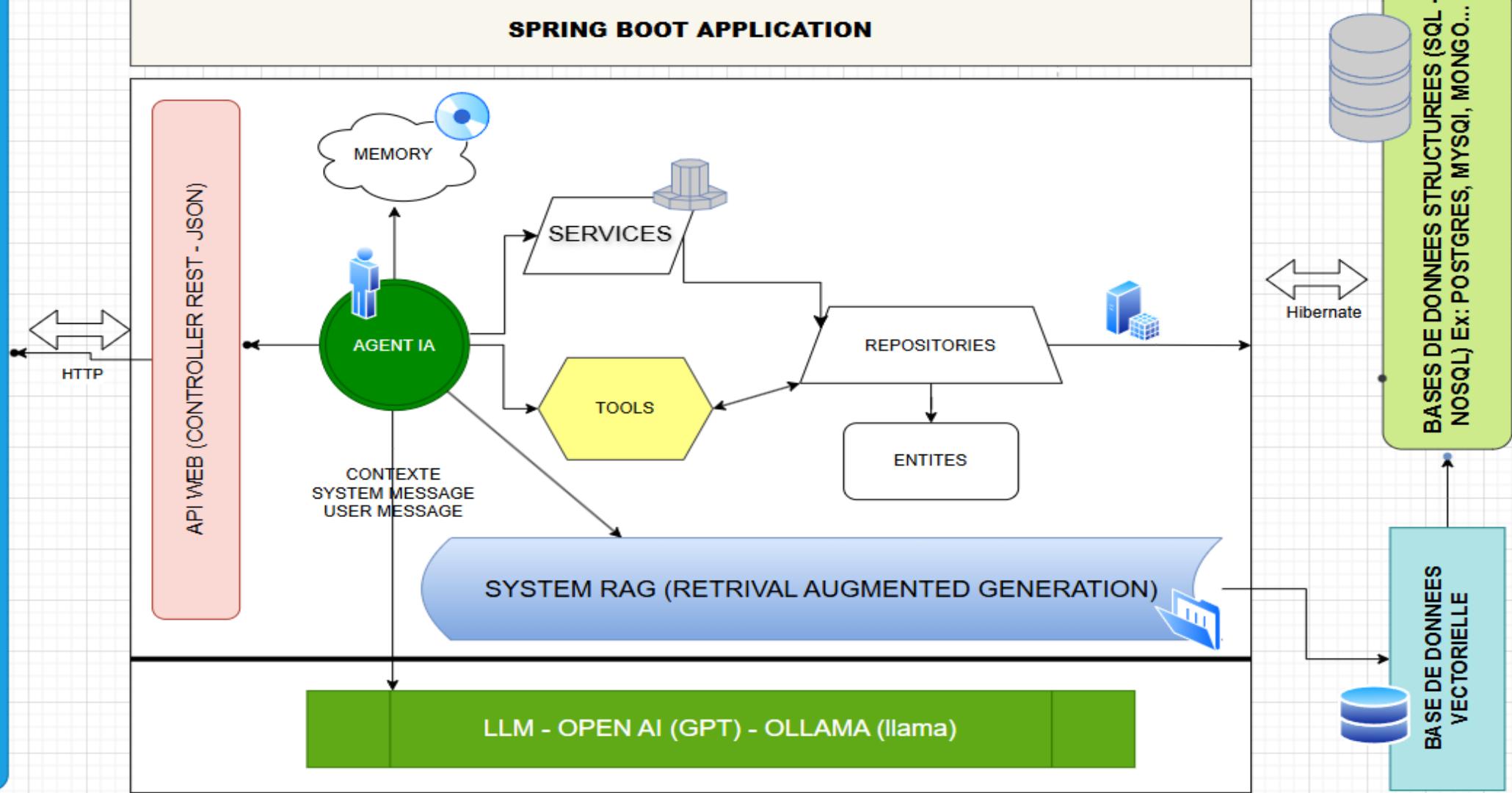
Workflow 1 de l'application

1. Création de l'application Spring boot et ajout des dépendances (web, jpa, lombok, postgres, langchain)
2. Présentation du contenu du projet et création des packages (entity, repository, service, controller, config...)
3. Configuration globale : présentation et mise à jour des fichiers pom.xml et application.properties
4. Création des fichiers java (ArticleEntity.java, ArticleRepository.java, ArticleService.java, ArticleController.java)
5. Test du backend avec des opérations CRUD sur l'entité Article (on va utiliser postman)

ARCHITECTURE APPLICATION



INTERFACE UTILISATEUR (ANGULAR REACT FLUTTER etc.)



Workflow 2 de l'application

1. Présentation des modèles LLM (exemples d'openAI avec gpt-4o et de ollama avec llama3.2) pour bien faire la différence entre les modèles payants en ligne et les modèles gratuits en local.
2. Configuration de langchain4J, création d'un premier agent avec openAI (gpt-4o) et test de deux endpoints (réponse en string brut et réponse stream)
3. Configuration de langchain4J avec ollama (llama3.2) et test de deux endpoints (réponse en string brut et réponse steam)
4. Mise en place de la mémoire de l'agent IA

Workflow 3 de l'application

1. Création d'une application frontend Angular pour consommer les endpoints de notre backend Spring
2. Mise en place du Service IA qui permet d'analyser le contenu des articles et d'attribuer des notes
3. Mise en place du service IA qui permet de créer un résumé d'article
4. Mise en place d'un chatbot qui permet à l'agent d'interagir avec notre base de données et des répondre à des questions
5. Mise en place d'un système d'analyse de document pdf (RAG)

A copier :

Workflow 1 : contenu fichier application.properties :

```
spring.application.name=uaszai
server.port=8096
spring.datasource.url=jdbc:postgresql://localhost:5432/uasz_ai
spring.datasource.username=postgres
spring.datasource.password=passer
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

A copier :

Workflow 2 : contenu fichier application.properties :

```
#Lanchain config OpenAI (commenté)
langchain4j.open-ai.chat-model.api-key=xxx
langchain4j.open-ai.chat-model.model-name=gpt-4o
langchain4j.open-ai.chat-model.temperature=0.7

#Lanchain streaming config OpenAI (commenté)
langchain4j.open-ai.streaming-chat-model.api-key=xxx
langchain4j.open-ai.streaming-chat-model.model-name=gpt-4o
langchain4j.open-ai.streaming-chat-model.temperature=0.7

#Lanchain config pour Ollama avec Llama 3.2 (actif)
# langchain4j.open-ai.chat-model.base-url=http://localhost:11434/v1
# langchain4j.open-ai.chat-model.api-key=ollama
# langchain4j.open-ai.chat-model.model-name=llama3.2
# langchain4j.open-ai.chat-model.temperature=0.7

# #Lanchain streaming config pour Ollama (actif)
# langchain4j.open-ai.streaming-chat-model.base-url=http://localhost:11434/v1
# langchain4j.open-ai.streaming-chat-model.api-key=ollama
# langchain4j.open-ai.streaming-chat-model.model-name=llama3.2
# langchain4j.open-ai.streaming-chat-model.temperature=0.7
```

A copier :

Workflow 2 : contenu fichier pom.xml :

```
<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-reactor</artifactId>
    <version>1.0.0-alpha1</version>
</dependency>

<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-open-ai-spring-boot-starter</artifactId>
    <version>1.0.0-alpha1</version>
</dependency>

<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-spring-boot-starter</artifactId>
    <version>1.0.0-alpha1</version>
</dependency>
```

A copier :

Workflow 2 : contenu fichier de l'agent IA:

```
@AiService
public interface UaszAI {
    @SystemMessage("you are a usefull agent, answer user's question")
    String chat(String question);

    @SystemMessage("you are a usefull agent, answer user's question")
    Flux<String> stream(String question);

}
```

A copier :

Workflow 2 : contenu du fichier config pour la mémoire ::

```
@Configuration
public class AgentConfig {
    @Bean
    ChatMemoryProvider chatMemoryProvider(Tokenizer tokenizer) {
        return chatId -> MessageWindowChatMemory.withMaxMessages(10);
    }
}
```

A copier :

Workflow 2 : contenu du fichier des outils (tools)

```
@Component
@Slf4j
public class AgentTools {
    @Autowired
    private ArticleService articleService;

    @Tool("method to load list of all articles")
    public List<Article> loadArticles() {
        return articleService.findAll();
    }

    @Tool("method to load article by id")
    public Article loadArticleById(Long id) {
        return articleService.findById(id);
    }

    @Tool("method to add new article")
    public Article addArticle(Article article) {
        return articleService.save(article);
    }

    @Tool("method to update article")
    public Article updateArticle(Article article, Long id) {
        article.setId(id);
        return articleService.update(article, id);
    }

    @Tool("method to delete article")
    public void deleteArticle(Long id) {
        articleService.delete(id);
    }
}
```

A copier :

Workflow 3 : Angular app

Création : `ng new frontend`

Installations :

bootstrap : `npm i bootstrap`

markdown : `npm i ngx-markdown@19`

Importation de bootstrap dans `style.css` : `@import "bootstrap/dist/css/bootstrap.min.css";`

Configurations :

ficher `app.config.ts` :

```
import { provideHttpClient } from '@angular/common/http';
```

```
import { provideMarkdown } from 'ngx-markdown';
```

```
export const appConfig: ApplicationConfig = {
```

```
  providers: [
```

```
    provideZoneChangeDetection({ eventCoalescing: true }),
```

```
    provideRouter(routes),
```

```
    provideHttpClient(),
```

```
    provideMarkdown(),
```

```
  ]
```

```
};
```

A copier :

Workflow 3 : Angular app

Configuration des routes :

deux components : ng g c Article et ng g c Chatbot

Dans le fichier app.routes.ts :

```
export const routes: Routes = [
  {
    path: 'chatbot',
    component: ChatbotComponent,
  },
  {
    path: 'article',
    component: ArticleComponent,
  },
  {
    path: '',
    component: ArticleComponent,
  }
];
```

A copier :

Workflow 3 : Angular app

Chargement des articles :

1. Model article (article.model.ts):

```
export interface Article {  
    id: number;  
    titre: string;  
    resume: string;  
    contenu: string;  
    auteur: Date;  
    note: string;  
}
```

2. Méthode de chargement (article.component.ts)

```
articles : Article[] = [];  
load(){  
    this.http.get<Article[]>("http://localhost:8096/article").subscribe({  
        next : (data) => { this.articles = data;},  
        error : (err) => console.log("erreur lors du chargement des articles")  
    })  
}
```

A copier :

Workflow 3 : Angular app

Notation des articles : (article.component.ts)

```
noter(id:any){
  this.http.get("http://localhost:8096/agent/note/"+id, {responseType: 'text'}).subscribe({
    next : (data) => {
      console.log(data);
    },
    error : (err) => console.log("erreur lors de la notation"),
    complete : () => {
      this.load();
    }
  })
}
```

A copier :

Workflow 3 : Angular app

Résumé des articles des articles : (article.component.ts)

```
resume(id:any){
  this.http.get("http://localhost:8096/agent/generate/resume/"+id, {responseType: 'text'}).subscribe({
    next : (data) => {
      console.log(data);
    },
    error : (err) => console.log("erreur lors de la mise à jour du résumé"),
    complete : () => {
      this.load();
    }
  })
}
```

NB : ne pas oublier d'importer CommonModule si non,

A copier :

Workflow 3 : Angular app

Article.component.html :

```
<div class="row">
  <div class="col-12">
    <h1 class="mb-4">Liste des Articles</h1>

    <div class="table-responsive">
      <table class="table table-striped table-hover table-bordered">
        <thead class="table-dark">
          <tr>
            <th scope="col">ID</th>
            <th scope="col">Auteur</th>
            <th scope="col">Résumé</th>
            <th scope="col">Contenu</th>
            <th scope="col">Note</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let item of articles">
            <td>{{item.id}}</td>
            <td>{{item.auteur}}</td>
            <td>{{item.resume}}</td>
            <td>{{item.contenu}}</td>
            <td>{{item.note}}</td>
            <td>
              <button class="btn btn-outline-success" (click)="noter(item.id)">noter</button>
              <button class="btn btn-outline-success" (click)="resume(item.id)">resumer</button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</div>
```

Nb : importer CommonModule dans article.component.ts pour que ngFor fonctionne

A copier :

Workflow 3 : Angular app

Article.component.html :

```
<div class="row">
  <div class="col-12">
    <h1 class="mb-4">Liste des Articles</h1>

    <div class="table-responsive">
      <table class="table table-striped table-hover table-bordered">
        <thead class="table-dark">
          <tr>
            <th scope="col">ID</th>
            <th scope="col">Auteur</th>
            <th scope="col">Résumé</th>
            <th scope="col">Contenu</th>
            <th scope="col">Note</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let item of articles">
            <td>{{item.id}}</td>
            <td>{{item.auteur}}</td>
            <td>{{item.resume}}</td>
            <td>{{item.contenu}}</td>
            <td>{{item.note}}</td>
            <td>
              <button class="btn btn-outline-success" (click)="noter(item.id)">noter</button>
              <button class="btn btn-outline-success" (click)="resume(item.id)">resumer</button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</div>
```

Nb : importer CommonModule dans article.component.ts pour que ngFor fonctionne

A copier :

Workflow 3 : Angular app

Chatbot : (chatbot.component.ts)

```
question : any;
response : any;
state : string = '';
constructor(private http: HttpClient){}

//Méthode pour une réponse en stream
askstream(){
  this.http.get("http://localhost:8096/agent/stream?question="+this.question, {responseType: 'text', observe:'events', reportProgress:true}).subscribe({
    next : evt => {
      this.state = "loading";
      if(evt.type === HttpEventType.DownloadProgress){
        this.response = (evt as HttpDownloadProgressEvent).partialText;
      }
    },
    error : (error) => {
      console.log(error);
      this.state = "failed";
    },
    complete : () => {
      this.state = "complete";
    }
  })
}

//Méthode pour une string brut
ask(){
  this.http.get("http://localhost:8096/agent?question="+this.question, {responseType: 'text'}).subscribe({
    next : (data) => {
      this.response = data;
    },
    error : (err) => console.log("Erreur lors de la réponse de l'agent IA"),
    complete : () => {
    }
  })
}
```

A copier :

Workflow 3 : Angular app

Chatbot : (chatbot.component.html)

```
<div class="row">
  <div class="col-12">
    <div class="card">
      <div class="card-header">
        <input type="text" [(ngModel)]="question" class="form-control"/>
        <button (click)="askstream()" class="btn btn-outline-primary mt-2">Poser une question</button>
      </div>
      <div class="card-body">
        <markdown [data]="response"></markdown>
      </div>
      <div class="footer">
        {{state}}
      </div>
    </div>
  </div>
</div>
```

Nb : importer FormsModule dans chatbot.component.ts pour que NgModel fonctionne