Cahier des Charges Fonctionnel

1. Introduction

MyPMB est notre outil de gestion de tickets utilisé depuis 2013. Il est utilisé par les chefs de projets pour planifier, par les développeurs pour organiser leur développement et signaler la livraison d'un développement (roadmap), ainsi que par l'assistance pour échanger avec les clients, entre autres exemples. Avec plus de 1 200 projets actifs, totalisant 175 000 demandes traitées ou en cours de traitement, MyPMB représente une somme de connaissances et un historique importants. L'évolution des techniques, notamment l'arrivée de l'intelligence artificielle, justifie ce projet de concevoir un agent conversationnel basé sur l'IA, destiné à être utilisé en interne dans les différents pôles au sein de PMB Services.

L'objectif principal de cet agent est de résoudre les problématiques identifiées par les différents pôles de l'entreprise en exploitant la base de connaissance MyPMB. L'agent conversationnel sera accessible directement depuis un navigateur web, sans nécessiter d'installation spécifique. Cela garantira une accessibilité immédiate et une utilisation fluide pour tous les utilisateurs.

Les principaux utilisateurs identifiés pour cet agent sont les membres de PMB Services dans leur ensemble, permettant à chacun d'utiliser ses fonctionnalités selon ses besoins spécifiques.

En somme, ce cahier des charges fonctionnel permettra de garantir une vision partagée des attentes et des fonctionnalités, tout en facilitant la collaboration entre les différents acteurs et d'anticiper les contraintes.

2. Objectifs du projet

2.1 Objectifs primaires

L'objectif principal de l'agent conversationnel intelligent destiné à PMB Services est d'optimiser les gains de temps, la montée en compétence et le partage de connaissance, au sein de PMB Services à l'aide de la base de données MyPMB. L'IA apporte sa capacité de calcul capable d'utiliser une grande base de données pour mettre en relation les sujets traités et proposer des solutions clés en main. Des ateliers de recueil de besoins ont permis d'obtenir les différents besoins des futurs utilisateurs de l'agent. Les objectifs opérationnels incluent :

2.1.1 Réduction du temps de recherche d'informations

La recherche sur MyPMB est limitée à une recherche en texte intégral et les filtres sont quasiment inexistants, ce qui rend la recherche compliquée.

Selon les recueils d'usages :

- Rechercher les demandes similaires dans le même projet ou dans tout MyPMB.
- Rechercher des informations liées à un diagnostic d'erreur.
- Pouvoir synthétiser un projet ou une demande.

La fonctionnalité de recherche d'information vise à :

- Réduire le temps de recherche, afin d'améliorer la productivité et de limiter les déperditions de temps.
- Favoriser une meilleure exploitation des données au sein de la base MyPMB.

2.1.2 Aide au diagnostic

L'aide au diagnostic technique devra fournir un prédiagnostic de la demande concernée. Cet objectif répond particulièrement aux attentes du pôle Assistance.

Selon les recueils d'usages :

- Utiliser l'IA pour vérifier qu'une demande relève principalement d'un pôle spécifique, puis pré-rédiger la sous-demande destinée à ce pôle.
- Suggérer des solutions à partir des tests déjà effectués et des anciennes demandes MyPMB.

La fonctionnalité de diagnostic d'erreur vise à :

- Identifier rapidement les problèmes techniques et le pôle le plus adapté.
- Fournir des réponses construites à partir des solutions trouvées et décrites dans MyPMB, facilitant la résolution des incidents.

2.1.3 Aide à la rédaction

Fonctionnalités demandées :

- Pouvoir corriger ses textes.
- Pouvoir corriger ses requêtes SQL.
- Suggérer des réponses pour les clients.

La fonctionnalité d'aide à la rédaction vise à :

- Faciliter la création de contenus, en étant capable d'utiliser d'anciennes réponses MyPMB rédigées par l'équipe.
- Corriger et/ou reformuler des textes directement fournis par l'utilisateur de l'agent.

2.2 Source des données

Pour garantir une recherche exhaustive et pertinente, l'agent conversationnel s'appuie principalement sur les ressources suivantes :

- La base de données MyPMB, qui centralise les demandes des projets clients.
- Les fichiers internes couramment utilisés par les différents pôles (PDF, Excel, Word).

2.3 Fonctionnalités des demandes et recherches

L'agent conversationnel proposerait un mode de recherche avec la possibilité d'ajouter des filtres pour optimiser l'expérience utilisateur :

- **Recherche par prompt** : Cette méthode permet aux utilisateurs d'effectuer des recherches rapides en saisissant une question en langage naturel. L'agent est capable de prendre en compte le contexte des interactions précédentes. Cela inclut les conversations en cours et les préférences exprimées, afin d'affiner les résultats et de fournir des réponses plus adaptées.
- **Recherche avec filtres avancés** : Pour les utilisateurs ayant des besoins plus spécifiques, il est possible d'affiner les recherches en appliquant des filtres tels que :
- Personne assignée
- Groupe assigné
- Tracker
- Projet
- Date
- Limiter le nombre de demandes MyPMB lors des propositions de réponse.
- Version du PMB
- Inclure les demandes liées

Il sera également possible de combiner ces deux méthodes de filtrage, permettant ainsi des recherches plus précises et adaptées aux besoins spécifiques des différents pôles de PMB Services.

2.4 Format des résultats

Les résultats de recherche sont présentés de manière claire et facilement exploitable par les utilisateurs, notamment à travers :

- Une zone de réponse générée par l'agent conversationnel, affichant directement les informations les plus pertinentes sous forme de texte clair et concis. Elle peut inclure des suggestions, des recommandations ou des explications détaillées, selon la nature de la demande, tout en tenant compte du contexte de la conversation en cours.
- Une liste de demandes MyPMB cliquables, accompagnés de titres et descriptions pour un aperçu rapide du contenu (+ popup qui ouvre la demande).

2.4 Objectifs secondaires

Outre les objectifs primaires déjà mentionnés, plusieurs objectifs secondaires ont été identifiés tels que :

2.4.1 Autres sources de données

L'agent conversationnel pourra se connecter à différents outils tiers tels que :

- Git
- Knowledge
- Dolibarr
- PMB
- Beesbusy
- Webikéo
- Documentation en ligne du logiciel
- YouTube

2.4.2 Sauvegarde de questions et historique de la conversation

La sauvegarde des questions dans notre agent conversationnel sera une fonctionnalité permettant d'améliorer et d'optimiser les interactions. En enregistrant les questions posées, l'utilisateur pourra les réutiliser plus tard.

3. Contraintes et exigences

3.1 Contraintes techniques

3.1.1 Utilisation de logiciels libres

L'agent conversationnel devra être développé en utilisant des outils libres pour garantir la flexibilité et la transparence, en accord avec les valeurs de PMB Services. Cela inclut l'utilisation de bibliothèques open-source, de frameworks et de technologies qui permettent une intégration fluide avec les systèmes de PMB Services.

3.1.2 Optimisation de l'expérience utilisateur

- Amélioration de l’ergonomie de l’interface pour garantir un accès intuitif et rapide aux fonctionnalités principales. Cet objectif inclut la réduction de la charge cognitive des utilisateurs afin de faciliter l’utilisation quotidienne de l’agent.

3.2 Contraintes de délai

3.2.1 Planning serré

Le projet doit être réalisé dans des délais précis pour pouvoir être présenté en tant que projet d’alternance dans le cadre du BTS de Gabriel et du stage de Tom. Cela nécessite une planification rigoureuse et une gestion efficace des ressources pour respecter les échéances.

4. Diagramme de class

```
@startuml
interface Chunker{
        {static} chunk(string text) string[]
}

class ChunkTokenOverLap implements Chunker{
        {static} chunk(string text,int nbToken,int nbOverlap) string[]
}

class ChunkUnit implements Chunker{
        {static} chunk(string text) string[]
}

class ChunkLineOverlap implements Chunker{
        {static} chunk(string text) string[]
}

interface Entity{
        + updateDate string
        + createDate string
        + id int
        + fromJson(string json) Entity
        + getID() int
        + getJson() string
        + getText() string
        + getUpdateDate() string
}
note right: type de donnée

class Demande implements Entity{
        + updateDate string
        + createDate string
        + id int
        - tracker int
        - project int
        - status int
        - description string
        - priority int
        - assigned int
        + __construct(string updateDate, string createDate, int id, int tracker, int project, int status, string description, int pr
        + fromJson(string json) Demande
        + getID() int
        + getJson() string
        + getText() string
        + getUpdateDate() string
        - normalizeText(string text) string
}

interface IaService {
        + systemPromptTemplate string
        + __construct(string systemPromptTemplate)
        + getEmbedding(string question): float[]|bool
        + response(Reference[] references,string question) : Response
        + vectorize(Extract[] extraits) Extract[]
}

class AiServiceMistral implements IaService{
        + systemPromptTemplate string
        - mistralKey string
        + __construct(string systemPromptTemplate, string apiKey)
        + getEmbedding(string question): float[]|bool
        + response(Reference[] references,string question) : Response
        + vectorize(Extract[] extraits) Extract[]
        - getEmbeddings(Extract[] extracts): Extract[]
}
```

```
interface Source{
        + source string
        + entities SourceEntity[]
        + __construct(string source)
        + countAllEntities(string type): int
        + countDateAllEntities(string type, SyncDate lastSyncDate): int
        + getAllEntities(string type, SyncDate lastSyncDate): Entity[]
        + getChunker(string type): string
        + getDateLimitEntities(string type, SyncDate lastSyncDate, int offset, int limit): Entity[]
        + getEntity(string type, int id): bool|Entity
        + getIdLimitEntities(string type, int offset, int limit): int[]
        + getLimitEntities(string type, int offset, int limit): Entity[]
        + getTypes(): string[]
        + getExtractsForResponse(string type): bool
        + hasType(type): string
}
note left : base externe

class MyPmb implements Source{
        + source string
        + entities SourceEntity[]
        - db PDO
        + __construct(string source, SourceEntity[] entities)
        + countAllEntities(string type): int
        + countDateAllEntities(string type, SyncDate lastSyncDate): int
        + getAllEntities(string type, SyncDate lastSyncDate): Entity[]
        + getChunker(string type): string
        + getDateLimitEntities(string type, SyncDate lastSyncDate, int offset, int limit): Entity[]
        + getEntity(string type, int id): bool|Entity
        + getIdLimitEntities(string type, int offset, int limit): int[]
        + getLimitEntities(string type, int offset, int limit): Entity[]
        + getTypes(): string[]
        + getExtractsForResponse(string type): bool
        + hasType(type): string
}

class Config{
        - data []
        + __construct(string path)
        + getDbParam(): []
        + getIaService(): IaService|bool
        + getLastSyncDates(): SyncDate[]
        + getSources(): Source[]
        + setLastSyncDates(SyncDate[] syncDates): bool
}
note bottom : lire le fichier de config en json

class Extract{
        - updateDate string
        - source string
        - type string
        - idEntity int
        - rank int
        - extract string
        - embedding float[]
        + __construct(string updateDate, string source, string type, int idEntity, int rank, string extract, float[]embedding)
        + getUpdateDate(): string
        + getSource(): string
        + getType(): string
        + getIdEntity(): int
        + getRank(): int
        + getExtract(): string
        + getEmbedding(): float[]
        + setEmbedding(float[] embedding): void
}
```

```
class Reference{
        - source string
        - type string
        - idEntity int
        - rankExtract int
        + __construct(string source, string type, int idEntity, int rankExtract)
        + getText():string
        - getEntityText():string|bool
        - getExtractText():string|bool
}

class Response{
        - answer string
        - references Reference[]
        + __construct(string answer, Reference[] references)
        + getAnswer(): string
}

class SourceEntity{
        - source string
        - type string
        - properties string[]
        - links string[]
        - chunker string
        - extractsForResponse bool
        - chunkParameters []
        + __construct(string source, string type, string baseType, string[] properties, string[] links, string chunker, bool extracts
        + getBaseType(): string
        + getChunker(): string
        + getExtractsForResponse(): bool
        + getProperties(): string[]
        + getType(): string
}
note left: description type de donnée

class SyncDate{
        - source string
        - type string
        - lastSyncDate string
        + __construct(string source, string type, string lastSyncDate)
        + getDate(): string
        + getSource(): string
        + getType(): string
}

class Store{
        __Constantes__
        + {static} {final} LIMIT_SOURCE = 50
        + {static} {final} LIMIT_INTELLIGATOR = 50
        + {static} {final} LIMIT_DELETE = 50
        __Variables__
        - {static} instance Store
        - iaService IaService
        - db PDO
        __
        - __construct(string configPath)
        + getInstance(string configPath)
        + getStoreEntity(string source, string type, int id): Entity|bool
        + getStoreExtractText(string source, string type, int idEntity, int rank): string|bool
        + shouldGetExtract(string source, string type): bool
        __Synchronisation__
        + chunkAll(): bool
        + syncAll(): bool
        + syncDateAll(): bool
        + syncDate(string source, string type): bool
```

```
        + sync(string source, string type): bool
        - chunk(string source, string type, Entity[] entities): bool
        - deleteEntities(Entity[] entities, string source, string type)
        - deleteLimitEntities(Entity[] entities, string source, string type): bool
        - insertExtracts(Extract[] extracts): bool
        - insertLimitExtracts(Extract[] extracts): bool
        - replaceEntities(Entity[] entities, string source, string type): bool
        - replaceLimitEntities(Entity[] entities, string source, string type): bool
        - synchDelete(string source, string type): bool
        __Recherche__
        + search(string question): string
        - calculeCosim(string requete, array extractsEntree): array
        - cosineSimilarity(array vectorA, array vectorB): float
        - creerExtraits(): array
        - getExtraits(): array
}
note top : gere la base de données

SourceEntity o-- Source : privided by
Store *-- SyncDate : synchronised at
Store *-- Config : has
Store *-- IaService : uses
Store --> Response : spawn
Chunker *-- Entity
Chunker *-- Extract
Response *-- Reference : consist of
@enduml
```

5. Dictionnaire de données

tables

Tables_in_intelligator
entity
extract

entity

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	NULL	
type	varchar(255)	NO	PRI	NULL	
creationDate	date	YES		NULL	
majDate	date	YES		NULL	
content	longtext	YES		NULL	

extracts

Field	Type	Null	Key	Default	Extra
updateDate	date	YES		NULL	
source	int(11)	YES		NULL	
type	varchar(255)	NO	PRI	NULL	
idEntity	bigint(20)	NO	PRI	NULL	
rank	int(11)	NO	PRI	NULL	
extract	text	YES		NULL	
embedding	longtext	YES		NULL	