

PROJET DE FIN D'ANNEE

5ème Année en Ingénierie Informatique et Réseaux

Système de Suivi Nutritionnel Assisté par l'IA

Réalisé par :

Hamza CHAFIQ

Nassim ZAHRI

Encadrant Académique :

M. MOURCHID MOHAMMED

Membre de jury:

M. MOURCHID MOHAMMED

M. TOUMI HICHAM

Remerciement

Nous tenons à exprimer nos plus sincères remerciements à **M. MOURCHID MOHAMMED**, notre encadrant académique, pour son accompagnement précieux tout au long de ce projet. Sa disponibilité, son écoute attentive et la pertinence de ses conseils ont constitué un appui inestimable qui a grandement contribué à la qualité et à la réussite de ce travail.

Nous lui sommes profondément reconnaissants pour la confiance qu'il nous a accordée, ainsi que pour sa bienveillance et son engagement constant à nous orienter et à nous encourager face aux différents défis rencontrés.

Note sur l'usage de l'intelligence artificielle

Dans le cadre de la rédaction de ce document, nous avons eu recours à des outils d'intelligence artificielle générative afin de nous assister dans la formulation, la structuration et l'amélioration stylistique du texte. **Ces outils n'ont pas remplacé notre travail de réflexion personnelle ni nos analyses**, mais ont servi d'appui pour optimiser la clarté, la cohérence et la qualité linguistique du rapport.

L'ensemble des idées, du contenu technique et des résultats présentés proviennent exclusivement de notre travail et de nos recherches, l'IA n'ayant été utilisée qu'en tant qu'outil d'assistance rédactionnelle.

Table des matières

Remerciement.....	2
Note sur l'usage de l'intelligence artificielle.....	2
Introduction Générale	6
Chapitre I - Contexte Général du Projet.....	7
1. Présentation générale du projet.....	7
1.1. Problématique.....	7
1.2. Client.....	7
2. Conduite de projet.....	8
2.1. Méthodologie de travail.....	8
2.2. Planification du projet.....	8
Conclusion	10
Chapitre II - Études Préliminaire et Fonctionnelle.....	11
1. Introduction.....	11
2. Étude Préliminaire	11
2.1. Présentation du projet.....	11
2.2. État de l'art.....	12
3. Étude de l'existant.....	13
4. Étude Fonctionnelle	13
4.1. Spécification des besoins fonctionnels.....	13
4.2. Spécification des besoins non fonctionnels.....	14
4.3. Identification des acteurs et cas d'utilisation.....	15
5. Diagramme de cas d'utilisation global	17
6. Conclusion.....	19
Chapitre III - Étude Conceptuelle.....	19
1. Méthode de Conception.....	19
2. Diagrammes.....	20
Conclusion	27
Chapitre IV - Étude Technique	28
1. Introduction.....	28
2. Architecture globale du système.....	28

2.1.	Couches logicielles.....	28
2.2.	Flux principal	30
2.3.	Rôle et architecture du Service IA.....	31
2.4.	Stratégie de fine-tuning pour la personnalisation du modèle.....	34
	Objectif et portée du fine-tuning.....	34
	Méthodologies de fine-tuning.....	34
	Constitution et annotation du jeu de données.....	35
3.	Outils et Framework de développement.....	36
3.1.	Environnement de développement	36
3.2.	Framework Backend.....	37
3.3.	Outils Frontend & API Testing	38
	Écosystème de Développement Frontend.....	38
	Stratégie de Test des API	38
	Conclusion	39
Chapitre V - Réalisation	40
1.	Introduction.....	40
2.	Présentation du résultat de l'application	40
2.1.	Interface Angular (Login \ Register).....	40
2.2.	Interface Settings préférence utilisateur	41
2.3.	Interface Agent et chatbot.....	42
2.4.	DashBoard.....	43
	Conclusion	44
	Synthèse des Réalisations d'Interface.....	44
Conclusion Générale et Perspectives	45
	Synthèse des réalisations.....	45
	Bénéfices et apports du projet.....	46
	Perspectives d'évolution.....	46
	Conclusion générale	47
	Références.....	47

Table des figures

Figure 1 - Diagramme de Gantt: Planification du projet.....	9
Figure 2 - Diagramme de cas d'utilisation global.....	18
Figure 3 - diagramme de classes.....	21
Figure 4 - Diagramme de séquence – Cas 1 : Consultation de l'Agent IA (Flux nominal)...	23
Figure 5 - Diagramme de séquence – Cas 2 : Erreur Azure OpenAI + gestion.....	24
Figure 6 - Diagramme de séquence – Cas 3 : Enregistrement d'un Repas (Tracking).....	26
Figure 7 - Architecture globale du system.....	30
Figure 8 - Pipeline RAG	33
Figure 9 - Logo de IntelliJ.....	36
Figure 10 - Logo de Git.....	37
Figure 11 - Logo de java.....	37
Figure 12 - Logo Angular.....	38
Figure 13 - Login page.....	40
Figure 14 - Register page.....	40
Figure 15 - 2.2. Interface Settings préférence utilisateur	41
Figure 16 - Chatbot.....	42
Figure 17 - Agent AI.....	42
Figure 18 - Dashboard	43

Liste des tableaux

Tableau 1 - Acteur: Utilisateur Final (Client / Patient)	15
Tableau 2 – Acteur: Coach Virtuel.....	15
Tableau 3 - Cas d'utilisation : « Authentification et Configuration du Profil » (CU1).....	16
Tableau 4 - Cas d'utilisation : « Interaction avec l'Agent IA » (CU2)	17

Introduction Générale

Dans un monde où la santé et le bien-être sont devenus des préoccupations centrales, la nutrition s'impose comme un levier fondamental pour prévenir les maladies et améliorer la qualité de vie. Cependant, malgré une prise de conscience croissante, la gestion quotidienne de l'alimentation reste une tâche complexe et souvent fastidieuse pour la majorité des individus. Les méthodes de suivi traditionnelles, basées sur la saisie manuelle et fastidieuse de chaque aliment consommé, constituent un frein majeur à une adoption durable de bonnes habitudes alimentaires.

C'est dans ce contexte que s'inscrit notre projet intitulé « **Système de Suivi Nutritionnel Assisté par l'IA** ». Notre ambition est de concevoir et de déployer une application web de nouvelle génération capable de transformer l'expérience du suivi nutritionnel. En mettant l'intelligence artificielle au service de la santé, nous visons à automatiser des tâches complexes telles que la reconnaissance alimentaire par vision par ordinateur, l'estimation des portions et la génération de recommandations personnalisées.

Sur le plan technique, ce projet représente un défi d'ingénierie logicielle complet basé sur une architecture moderne de microservices. Nous avons fait le choix stratégique de centraliser l'intégralité du développement backend et l'intégration des services d'intelligence artificielle sur l'écosystème Spring Boot. Cette approche garantit une cohérence technique totale et une sécurité accrue, tandis que l'interface utilisateur est portée par Angular 20 pour offrir une expérience fluide, réactive et intuitive.

L'objectif de ce rapport est de présenter de manière détaillée le fruit de notre travail, en parcourant les étapes de l'analyse des besoins, de la conception architecturale, de l'implémentation des modèles d'IA et du déploiement de la solution finale. À travers ce document, nous démontrerons comment l'innovation technologique peut simplifier le quotidien des utilisateurs tout en leur offrant des outils d'analyse nutritionnelle précis et engageants.

Chapitre I - Contexte Général du Projet

1. Présentation générale du projet

Ce projet consiste en la conception et le déploiement d'une application web de suivi nutritionnel moderne, intégrant l'intelligence artificielle pour faciliter la gestion de la santé au quotidien. L'objectif est de démontrer une expertise complète en développement full-stack, en traitement d'images par IA et en analyse de données nutritionnelles.

1.1. Problématique

Malgré une prise de conscience accrue de l'importance d'une alimentation saine, la majorité des utilisateurs se heurtent à plusieurs obstacles majeurs :

- **La complexité du suivi manuel:** Saisir manuellement chaque aliment consommé est une tâche fastidieuse qui mène souvent à l'abandon du suivi.
- **Le manque de précision :** Il est difficile pour un utilisateur non averti d'estimer précisément les calories et les macronutriments (protéines, glucides, lipides) d'un plat.
- **L'absence de recommandations personnalisées :** La plupart des outils existants se contentent de lister des données sans offrir de conseils adaptés au profil spécifique, aux allergies ou aux objectifs de l'utilisateur
- **La difficulté d'analyse des tendances :** Identifier les carences ou les excès sur le long terme sans outils de visualisation avancés est complexe pour le grand public.

1.2. Client

Nous avons conçu la solution pour répondre aux attentes de différents profils d'utilisateurs finaux.

Type d'utilisateur	Besoins principaux	Exemple concret
Profil en transformation physique	Suivi rigoureux des calories et des macronutriments pour atteindre un objectif de poids précis.	Un utilisateur souhaitant perdre 5 kg en 3 mois en suivant un déficit calorique contrôlé.

Utilisateur avec contraintes alimentaires	Gestion des allergies, des intolérances et des préférences spécifiques (végétarien, halal, etc.)	Une personne allergique au gluten ayant besoin d'un scanner IA pour vérifier la compatibilité des repas.
Utilisateur orienté bien-être	Suivi de l'hydratation, équilibre nutritionnel général et motivation par chatbot.	Un employé de bureau souhaitant s'assurer qu'il boit suffisamment d'eau et équilibre ses repas quotidiens.

2. Conduite de projet

2.1. Méthodologie de travail

Pour mener à bien ce projet complexe intégrant des microservices et de l'intelligence artificielle, nous avons mis en place une organisation rigoureuse basée sur une équipe recommandée de deux développeurs.

Approche générale :

- **Architecture Microservices :** Ce choix permet de découpler les fonctionnalités (Auth, Profil, IA, Repas) pour faciliter le développement parallèle et la maintenance.
- **Standardiser le développement :** Tous les microservices (authentification, profil, IA, suivi) partagent la même structure et les mêmes outils de build.
- **Standardisation technique :** Pour maximiser l'efficacité, nous avons centralisé la logique métier et l'orchestration de l'IA sur Spring Boot, tout en utilisant Angular 20 pour une interface utilisateur réactive et modulaire.

2.2. Planification du projet

Le projet est découpé en huit phases clés, totalisant 55 jours de travail. Cette planification permet de valider chaque brique technologique avant de passer à l'intégration globale.

Phase	Description
1. Analyse et conception	Analyse du marché (personas), diagrammes UML, schémas de base de données PostgreSQL/MongoDB et wireframes UI/UX.
2. Infrastructure de base	Configuration du Service Discovery (Eureka), de l'API Gateway et sécurisation par JWT.
3. Services Métier Backend	Développement des services de profil, de suivi des repas, d'hydratation et d'analyse nutritionnelle sous Spring Boot.
4. Frontend Angular	Développement de l'interface responsive, des tableaux de bord interactifs et intégration PWA.
5. Intelligence Artificielle	Mise en place d'un chatbot et moteur de recommandation aussi qu'un Agent IA.
6. Documentation et finalisation	Rédaction de la documentation technique et élaboration du rapport de projet.

Cette planification a servi de feuille de route stratégique et a permis de maintenir un équilibre constant entre le développement des microservices sous Java Spring Boot, la création d'une interface réactive avec Angular.

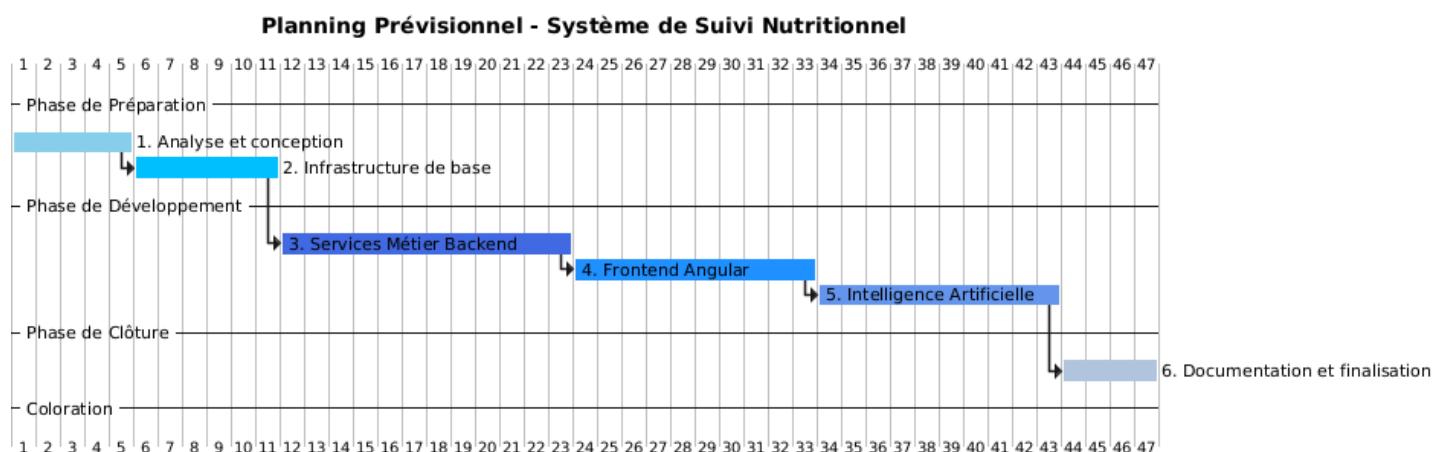


Figure 1 - Diagramme de Gantt: Planification du projet

Conclusion

Ce premier chapitre a permis de poser le cadre général du projet, en mettant en évidence la problématique de la gestion nutritionnelle moderne, le public cible visé ainsi que la démarche méthodologique adoptée pour sa réalisation. L'application développée vise à combler un besoin concret dans le domaine de la santé connectée et du bien-être, en automatisant le suivi des apports et la génération de plans alimentaires personnalisés grâce à un Agent IA intelligent intégré au système.

En identifiant les besoins des sportifs, des personnes ayant des contraintes alimentaires et des utilisateurs soucieux de leur équilibre de vie, des objectifs clairs ont été définis, orientés vers l'efficacité, la précision nutritionnelle et l'interactivité. Sur le plan technique, la combinaison des technologies Spring Boot pour le backend et l'IA, de Angular 20 pour le frontend, et des bases de données PostgreSQL et MongoDB constitue un socle solide garantissant la performance et la fiabilité de la solution en environnement de développement local.

La méthodologie de travail adoptée, de nature itérative, assure un cycle de développement structuré, allant de la conception des microservices à l'intégration de l'interface utilisateur et des capacités décisionnelles de l'intelligence artificielle. Cette approche permet de réduire les risques techniques, d'optimiser les délais de réalisation sur une période de 11 semaines et de maintenir une qualité constante à chaque étape du développement sur nos postes de travail.

Ainsi, ce chapitre introductif prépare le terrain pour les sections suivantes, qui détailleront l'analyse fonctionnelle, les choix architecturaux et l'implémentation technique de l'Agent IA au sein d'une infrastructure de microservices robuste et cohérente.

Chapitre II - Études Préliminaire et Fonctionnelle

1. Introduction

Le deuxième chapitre a pour objectif de présenter les analyses préliminaires et fonctionnelles nécessaires à la conception du Système de Suivi Nutritionnel Assisté par l'IA. Avant d'entamer la phase de réalisation technique, il est essentiel de disposer d'une vision claire et précise du projet, tant sur le plan conceptuel que stratégique. Cette étape permet non seulement de cadrer les fonctionnalités à développer, mais également de s'assurer que la solution proposée répond efficacement aux besoins de santé et de bien-être identifiés dans le chapitre précédent.

Dans un premier temps, une étude préliminaire sera réalisée afin de présenter en détail la finalité du projet, son champ d'application et les choix technologiques retenus, incluant Spring Boot, Angular et l'intégration d'un Agent IA conversationnel. Cette partie intégrera également un état de l'art des solutions existantes dans le domaine de la « e-santé » et du coaching nutritionnel, en mettant en évidence leurs limites et en positionnant notre outil par rapport à la concurrence.

Ensuite, l'étude de l'existant permettra d'identifier les lacunes des outils de suivi traditionnels. Cette analyse fournira une évaluation claire de la valeur ajoutée apportée par notre Agent IA décisionnel. La partie centrale de ce chapitre sera consacrée à l'étude fonctionnelle. Elle définira de manière structurée les besoins fonctionnels (gestion de profil, planification des repas, interactions avec l'agent), les exigences non fonctionnelles (performance, sécurité des données), ainsi que les acteurs du système.

2. Étude Préliminaire

2.1. Présentation du projet

Le projet consiste à concevoir et développer une application web innovante dédiée au suivi nutritionnel et à la planification alimentaire personnalisée. Contrairement aux applications classiques qui se limitent à un enregistrement passif des données, notre solution intègre un Agent IA Intelligent capable d'analyser le profil de santé de l'utilisateur pour agir comme un véritable coach virtuel .

Ce projet cible principalement deux types d'utilisateurs :

- ✓ **Les profils sportifs ou esthétiques** : Souhaitant optimiser leurs performances ou leur composition corporelle grâce à un suivi rigoureux des macronutriments et des plans de repas adaptés à leur métabolisme.
- ✓ **Les profils « santé »** : Ayant des besoins spécifiques liés à des pathologies (diabète, hypertension) ou des contraintes alimentaires (allergies, intolérances) nécessitant une vigilance accrue et des recommandations fiables.

L'application repose sur une architecture moderne et modulaire, articulée autour de trois axes technologiques majeurs :

- **Backend Spring Boot** : Assurant la robustesse de la logique métier, la sécurité des données de santé et l'orchestration des microservices.
- **Frontend Angular** : Offrant une interface utilisateur réactive, fluide et ergonomique pour visualiser les progrès et interagir avec le système.
- **Intelligence Artificielle (Agent)** : Un module décisionnel basé sur le NLP (Traitement du Langage Naturel) capable de dialoguer avec l'utilisateur, de comprendre ses besoins en langage naturel et de générer des plans nutritionnels dynamiques.

2.2. État de l'art

Aujourd'hui, la nutrition numérique occupe une place centrale dans le secteur de la santé préventive. Les utilisateurs recherchent des solutions capables non seulement de stocker leurs données, mais aussi de les interpréter pour guider leurs décisions alimentaires.

Différentes approches existent sur le marché :

- **Les applications de « tracking » passif** : Des solutions populaires permettent aux utilisateurs de saisir manuellement leurs repas pour compter les calories. Bien qu'efficaces pour la collecte de données, elles demandent un effort constant de la part de l'utilisateur et n'offrent souvent que des statistiques brutes sans analyse contextuelle approfondie .
- **Les services de coaching humain** : Très performants car totalement personnalisés, ils restent cependant coûteux et difficilement accessibles au grand public sur le long terme.
- **L'émergence des coachs IA** : Les avancées récentes en intelligence artificielle (LLM, modèles génératifs) ouvrent la voie à une nouvelle génération d'outils. Ces systèmes peuvent désormais comprendre le contexte d'un utilisateur et fournir des réponses nuancées, simulant l'interaction avec un expert humain .

3. Étude de l'existant

Avant de concevoir notre solution, il est indispensable d'analyser les outils actuellement disponibles sur le marché afin de comprendre leurs fonctionnalités, mais surtout d'identifier les manques qui justifient le développement de notre système. Cette étude permet de situer notre projet par rapport à la concurrence et de définir précisément sa valeur ajoutée.

Actuellement, le marché de la « e-santé » et de la nutrition est dominé par deux grandes catégories d'applications, qui présentent chacune des avantages indéniables mais aussi des limites structurelles importantes.

D'un côté, nous trouvons les applications de suivi calorique traditionnelles (comme MyFitnessPal, Yazio ou FatSecret). Ces outils sont extrêmement robustes pour constituer des bases de données alimentaires et calculer des totaux caloriques. Cependant, leur modèle repose quasi exclusivement sur la saisie manuelle, une tâche répétitive et fastidieuse qui décourage une grande partie des utilisateurs après quelques semaines. De plus, ces applications restent passives : elles affichent des données brutes (graphiques, chiffres) mais n'offrent que rarement des conseils proactifs ou une véritable adaptation dynamique aux changements de vie de l'utilisateur .

D'un autre côté, l'émergence des chatbots généralistes (comme ChatGPT ou Claude) permet d'obtenir des conseils nutritionnels en langage naturel. Ces IA peuvent générer des menus ou répondre à des questions théoriques avec une grande fluidité. Toutefois, ces systèmes souffrent d'un manque de contexte critique : ils ne sont pas connectés en temps réel aux données de santé de l'utilisateur, à son historique de repas ou à ses objectifs précis stockés dans une base de données sécurisée. Leurs conseils restent donc génériques et parfois déconnectés de la réalité physiologique de la personne .

4. Étude Fonctionnelle

4.1. Spécification des besoins fonctionnels

La spécification des besoins fonctionnels définit précisément les capacités que le Système de Suivi Nutritionnel doit offrir pour répondre aux attentes des utilisateurs et aux objectifs fixés. Elle traduit les exigences métier en fonctionnalités concrètes implémentées au sein de l'application.

Fonctionnalités principales :

1. Gestion des profils et Authentification :

- ✓ Le système doit permettre la création de comptes utilisateurs sécurisés avec une gestion fine des données personnelles (âge, poids, taille, sexe, niveau d'activité).
- ✓ Il doit intégrer un calculateur automatique des besoins caloriques et des macronutriments (protéines, glucides, lipides) basé sur les objectifs de l'utilisateur (perte de poids, prise de masse, maintien).
- ✓ L'authentification doit être assurée par le protocole standard JWT (JSON Web Token) pour garantir la sécurité des sessions.

2. Suivi Nutritionnel Intelligent :

- ✓ L'application doit permettre l'enregistrement quotidien des repas et de l'hydratation. Contrairement à une saisie purement manuelle, ce module est assisté par l'IA pour simplifier l'estimation des portions et le calcul des apports.
- ✓ Un tableau de bord interactif doit présenter en temps réel la progression par rapport aux objectifs journaliers (calories consommées vs cibles).

3. Agent Conversationnel (IA Coach) :

- ✓ Le cœur du système repose sur un Agent IA capable d'interagir en langage naturel avec l'utilisateur.
- ✓ Il doit pouvoir répondre à des questions nutritionnelles, suggérer des alternatives alimentaires en cas d'allergies et fournir des explications sur les recommandations générées.
- ✓ L'agent doit agir de manière proactive en analysant l'historique de l'utilisateur pour détecter des tendances (carences, excès répétés) et proposer des ajustements.

4. Génération et Planification de Repas :

- ✓ Le système doit être capable de générer des plans de repas hebdomadaires personnalisés, respectant strictement les contraintes alimentaires (végétarien, sans gluten, halal) et les objectifs caloriques de l'utilisateur.

4.2. Spécification des besoins non fonctionnels

Les besoins non fonctionnels définissent les caractéristiques qualitatives de l'application, visant à garantir sa performance, sa fiabilité, sa sécurité et son ergonomie. Ces critères sont essentiels pour assurer une expérience utilisateur optimale et la robustesse de la solution dans un environnement de production.

1. Performance et Réactivité :

- ✓ L'interface utilisateur, développée avec Angular 20, doit offrir une expérience fluide (Single Page Application) avec des temps de chargement minimaux.

- ✓ Les microservices Spring Boot doivent traiter les requêtes de calcul nutritionnel et les interactions avec l'IA avec une latence réduite, optimisée pour un usage local efficace.

2. Sécurité et Confidentialité :

- ✓ Les données de santé étant sensibles, l'application doit assurer leur protection via un chiffrement des mots de passe et une sécurisation des échanges API.
- ✓ L'architecture doit respecter les principes de séparation des données, garantissant que les informations personnelles ne sont accessibles qu'aux utilisateurs autorisés.

3. Ergonomie et Accessibilité :

- ✓ L'application doit être "Responsive Design", s'adaptant aussi bien aux écrans de bureau qu'aux appareils mobiles pour un suivi en déplacement.
- ✓ Les visualisations graphiques (courbes de poids, répartition des macros) doivent être claires et lisibles pour tout type d'utilisateur.

4.3. Identification des acteurs et cas d'utilisation

L'identification des acteurs permet de cerner qui interagit avec le système et comment.

a) Acteurs principaux

i) Acteur 1

Acteur : Utilisateur Final (Client / Patient)
Rôle : Personne physique inscrite sur la plateforme, souhaitant améliorer son hygiène de vie ou atteindre des objectifs physiques précis (perte de poids, prise de masse, santé).
Besoin : Bénéficier d'un suivi nutritionnel personnalisé, enregistrer ses repas quotidiens et dialoguer avec le coach virtuel pour obtenir des conseils adaptés.

Tableau 1 - Acteur: Utilisateur Final (Client / Patient)

ii) Acteur 2

Acteur : Coach Virtuel
Rôle : Entité intelligente du système, agissant comme un expert nutritionnel virtuel capable d'analyser le contexte de l'utilisateur.
Besoin : Analyser les données de santé en temps réel, détecter les écarts alimentaires, générer des plans de repas et fournir des recommandations proactives.

Tableau 2 – Acteur: Coach Virtuel

b) Cas d'utilisation

i) Cas d'utilisation : « Authentification et Configuration du Profil » (CU1)

Elément	Description
Cas n°	CU1
Acteur(s)	Utilisateur
Objectif	Permettre à l'utilisateur de s'inscrire, de se connecter et de définir ses paramètres physiologiques pour le calcul des besoins.
Pré-condition(s)	L'application Angular est accessible et les microservices Spring Boot sont opérationnels.
Post-condition(s)	L'utilisateur est authentifié (Token JWT généré), son profil est stocké en base de données et ses objectifs caloriques sont calculés.
Scénario nominal	<ol style="list-style-type: none"> 1. L'utilisateur saisit ses identifiants (email/mot de passe) via l'interface Angular. 2. Le service d'authentification valide les données et renvoie un jeton JWT. 3. L'utilisateur remplit son profil santé (âge, poids, taille, niveau d'activité, objectif). 4. Le système calcule automatiquement les besoins caloriques journaliers (BMR/TDEE) via les algorithmes implémentés. 5. Le profil est sauvegardé et l'utilisateur est redirigé vers le tableau de bord.
Scénario alternatif	<ul style="list-style-type: none"> - Identifiants invalides : Le système renvoie une erreur 401 et invite à réessayer. - Données profil incohérentes : Si l'utilisateur saisit des valeurs impossibles (ex: poids négatif), le frontend bloque la validation via les validateurs de formulaire Angular.

Tableau 3 - Cas d'utilisation : « Authentification et Configuration du Profil » (CU1)

ii) Cas d'utilisation : « Interaction avec l'Agent IA » (CU2)

Elément	Description
Cas n°	CU2
Acteur(s)	Utilisateur, Agent IA
Objectif	Obtenir des conseils nutritionnels personnalisés ou un plan de repas via une interface conversationnelle.
Pré-condition(s)	L'utilisateur est connecté et possède un profil nutritionnel complet.
Post-condition(s)	Une réponse contextuelle est générée par l'IA et affichée dans le module de chat ; l'historique est sauvegardé.
Scénario nominal	<ol style="list-style-type: none"> 1. L'utilisateur pose une question en langage naturel (ex: "Que manger après mon sport ?") via le module de Chatbot. 2. Le service Spring Boot récupère le contexte de l'utilisateur (objectifs, allergies, historique récent). 3. Le système construit un prompt enrichi et l'envoie au modèle de langage (LLM) via l'API intégrée. 4. L'Agent IA génère une réponse personnalisée et motivante. 5. La réponse est affichée instantanément dans l'interface Angular.
Scénario alternatif	<ul style="list-style-type: none"> - Service IA indisponible : Si l'API externe ne répond pas, le système affiche un message d'attente ou une réponse par défaut. - Demande hors sujet : Si l'utilisateur pose une question non liée à la nutrition, l'Agent recadre poliment la conversation sur son domaine d'expertise

Tableau 4 - Cas d'utilisation : « Interaction avec l'Agent IA » (CU2)

5. Diagramme de cas d'utilisation global

Ce diagramme synthétise les interactions majeures identifiées dans votre étude fonctionnelle : la gestion autonome par l'utilisateur (profil, suivi), l'administration technique, et le cœur du système : la collaboration avec l'Agent IA pour le coaching et la planification.

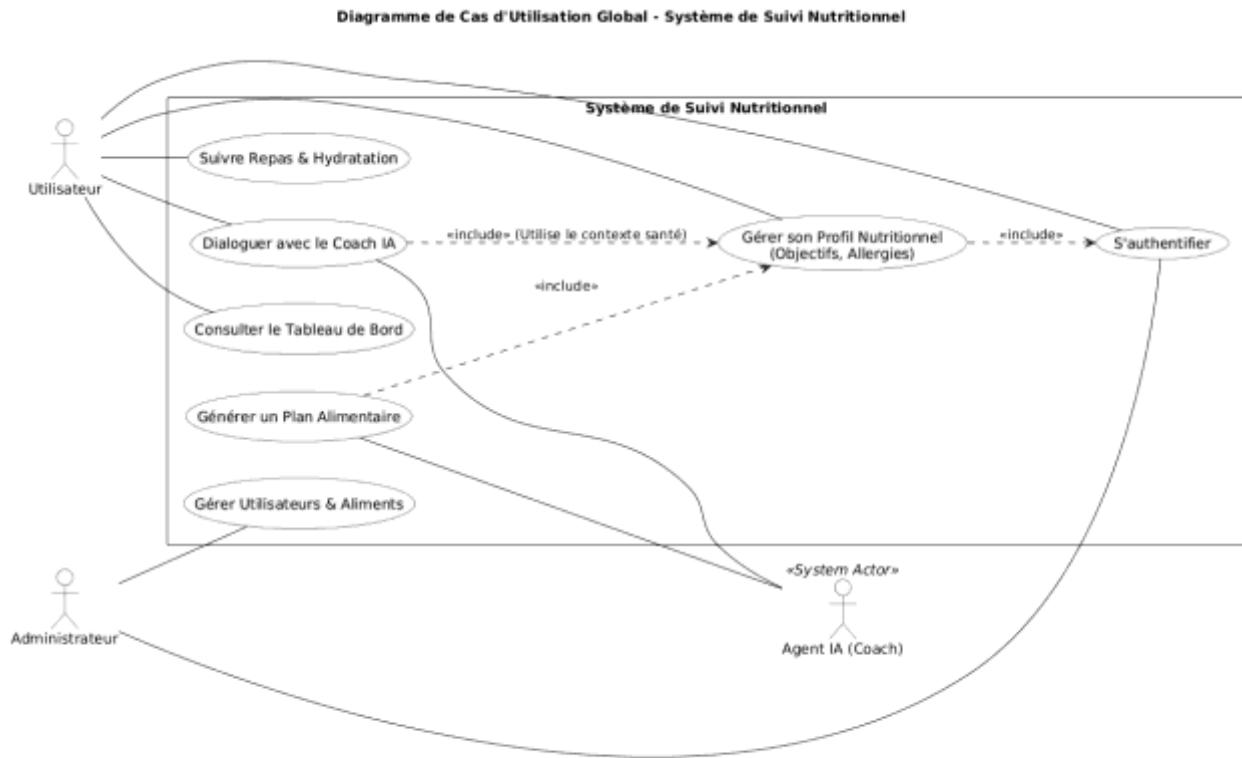


Figure 2 - Diagramme de cas d'utilisation global

Ce diagramme met en évidence la structure tripartite de l'interaction :

- ❖ **L'Utilisateur (à gauche)** : Il accède aux fonctions classiques de "Tracking" (Repas, Hydratation, Dashboard) et gère ses données personnelles.
- ❖ **L'Agent IA (à droite)** : Il est représenté comme un acteur système actif. Il intervient spécifiquement dans les modules "intelligents" : le dialogue (Chatbot) et la génération complexe de plans alimentaires, qui nécessitent une puissance de calcul et d'analyse (NLP).
- ❖ **L'Administrateur** : Il conserve un rôle de supervision technique (gestion des comptes et de la base de données alimentaire).

Les relations <<include>> (en pointillés) soulignent que l'intelligence de l'IA (Chat et Planification) repose directement sur la qualité des données du Profil Nutritionnel renseigné par l'utilisateur.

6. Conclusion

Ce deuxième chapitre nous a permis de poser les bases analytiques et fonctionnelles nécessaires à la réalisation du Système de Suivi Nutritionnel Assisté par l'IA.

Dans un premier temps, l'étude préliminaire a clarifié les objectifs, le périmètre et les choix technologiques stratégiques (Spring Boot, Angular), tout en mettant en lumière l'état de l'art du marché de la « e-santé ». Cette analyse a montré que, malgré la popularité des applications de tracking calorique et l'émergence des chatbots, il subsiste un besoin non comblé pour une solution hybride alliant la rigueur des données de santé à la proactivité d'un coach virtuel intelligent.

L'étude fonctionnelle a ensuite détaillé les besoins du système, en distinguant les exigences fonctionnelles critiques (authentification, gestion de profil, interaction avec l'Agent IA) des exigences non fonctionnelles (performance, sécurité des données sensibles). De plus, l'identification des acteurs et la modélisation via le diagramme de cas d'utilisation ont apporté une vision claire des rôles et des interactions au sein de l'application, soulignant la place centrale de l'Agent IA comme acteur système.

Ainsi, ce chapitre prépare directement la transition vers la conception technique et architecturale qui sera développée dans le chapitre suivant. Les éléments présentés ici constituent le socle de référence pour assurer une implémentation alignée aux besoins réels des utilisateurs et conforme aux bonnes pratiques de développement moderne .

Chapitre III – Étude Conceptuelle

1. Méthode de Conception

Pour concevoir ce projet, nous avons retenu UML (Unified Modeling Language) comme méthode de modélisation. Il s'agit d'un langage visuel normalisé par l'Object Management Group (OMG, 2017) qui permet de représenter à la fois la structure et le comportement d'un système logiciel . UML met à disposition différents types de diagrammes, tels que les diagrammes de classes ou de composants pour la partie structurelle, et les diagrammes de cas d'utilisation, de séquence ou d'activités pour la partie comportementale (Fowler, 2004).

L'un des principaux atouts d'UML réside dans sa capacité à apporter une vision claire et partagée du projet, que ce soit pour des développeurs, des concepteurs ou des utilisateurs non techniques. Cette clarté et cette standardisation favorisent la communication et réduisent les ambiguïtés dans l'interprétation des spécifications (Booch, Rumbaugh & Jacobson, 2005) .

L'utilisation d'UML présente plusieurs avantages :

- **Clarté et standardisation** : UML est reconnu internationalement (normalisé par l'OMG – Object Management Group) et permet une communication claire entre développeurs, chefs de projet et autres parties prenantes.
- **Vision globale du système** : il facilite la compréhension des besoins fonctionnels et techniques en offrant une représentation visuelle accessible même aux non-techniciens.
- **Réduction des ambiguïtés** : grâce à ses diagrammes variés, UML aide à formaliser les spécifications et à éviter les malentendus entre l'équipe technique et les utilisateurs.
- **Adaptabilité** : UML peut être utilisé pour des projets de petite ou grande envergure et s'adapte aussi bien aux approches classiques qu'aux méthodologies agiles.

Dans le cadre de notre projet, UML se justifie par la nécessité de représenter de manière rigoureuse les cas d'utilisation identifiés, les interactions entre les utilisateurs, l'administrateur et l'Agent IA, ainsi que les flux techniques qui relient notre architecture backend aux services d'intelligence artificielle. En résumé, il constitue un outil efficace pour formaliser les besoins et assurer une transition fluide entre l'étude fonctionnelle et la phase de développement technique.

2. Diagrammes

Afin de traduire les besoins fonctionnels et techniques définis précédemment en représentations claires et structurées, nous avons recours à plusieurs types de diagrammes UML. Ces diagrammes permettent d'illustrer différents aspects complémentaires du système.

Le diagramme de cas d'utilisation met en évidence les interactions entre les acteurs (utilisateurs, administrateurs, Agent IA) et les principales fonctionnalités de l'application. Le diagramme de séquence décrit la dynamique des échanges entre les composants du système, notamment lors des dialogues avec l'Agent IA et du traitement des données nutritionnelles, en mettant en avant l'ordre temporel des interactions. Enfin, le diagramme de classes offre une vision structurelle de l'application en représentant les entités principales (Profil, Repas, Aliments), leurs attributs et les relations qui les unissent.

En combinant ces représentations, il devient possible d'obtenir une vision globale et cohérente du système, facilitant ainsi la compréhension, la conception et l'implémentation du projet.

a) Diagramme de classe

Le diagramme de classes est un outil fondamental de la modélisation UML. Il offre une vision statique du système en représentant les entités principales (classes, interfaces, objets de configuration, modèles de données) ainsi que les relations entre elles. L'objectif est de clarifier l'architecture logicielle avant la mise en œuvre, en montrant à la fois la structure interne des classes et leurs interactions.

Le diagramme de classes ci-dessous illustre l'architecture de données du système. Contrairement à une application monolithique classique, cette architecture est conçue en microservices. Chaque module (Auth, User, Food, Meal, Water) fonctionne de manière indépendante avec sa propre persistance de données. Les relations entre les entités ne sont donc pas des clés étrangères SQL rigides, mais des références logiques (stockage de l'ID d'un objet appartenant à un autre service), garantissant un couplage faible et une haute résilience.

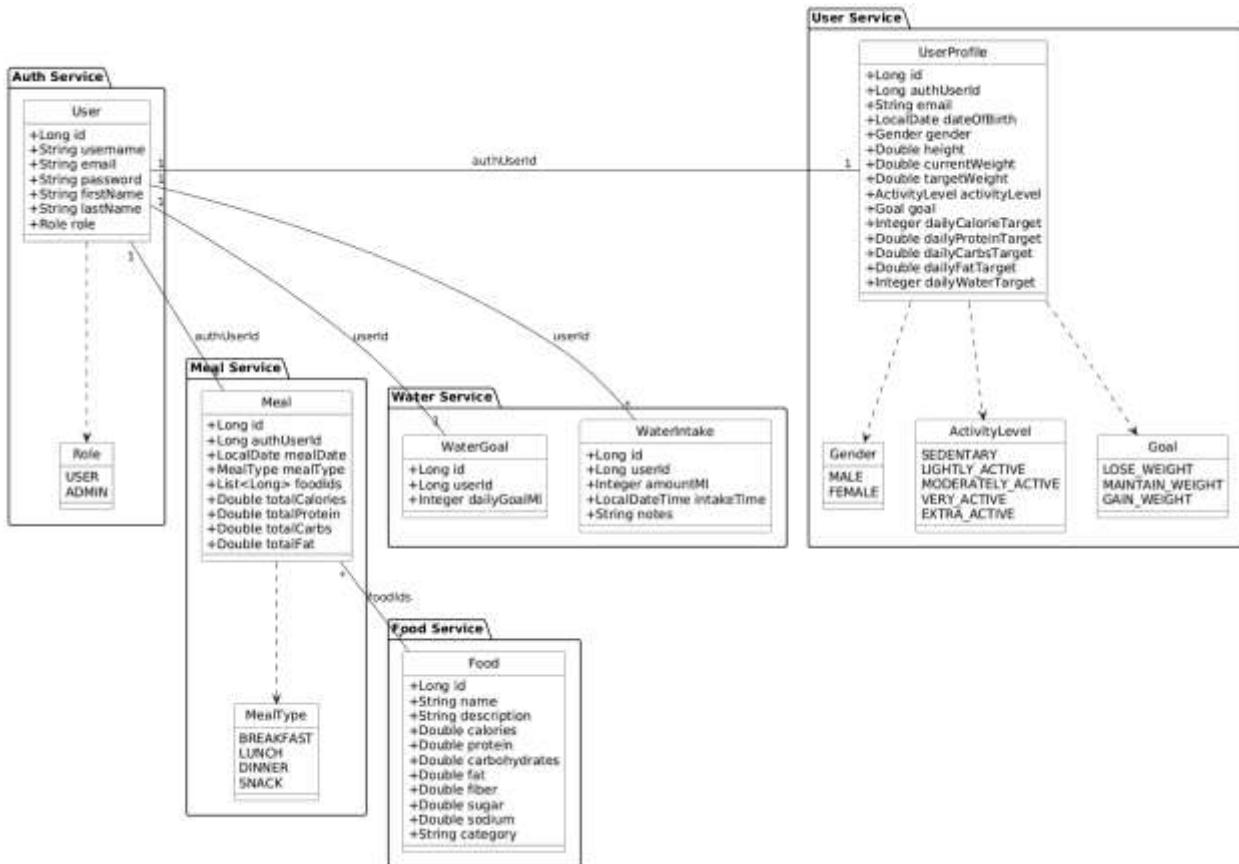


Figure 3 - diagramme de classes

L'architecture s'articule autour de cinq services interconnectés logiquement :

- 1. Auth Service (Gestion de l'Identité)** Ce service constitue le point d'entrée sécuritaire du système.
 - **Entité User** : Elle centralise les informations d'authentification essentielles (username, email, password chiffré) et les droits d'accès (Role). L'identifiant unique (id) de cette entité sert de pivot central ("colonne vertébrale") auquel tous les autres services font référence pour identifier l'utilisateur actif.
- 2. User Service (Gestion du Profil Santé)** Ce service est découplé de l'authentification pour alléger le processus de connexion et isoler les données sensibles de santé.
 - **Entité UserProfile** : Elle stocke les données anthropométriques (height, currentWeight, dateOfBirth) ainsi que les préférences (activityLevel, goal).
 - **Calculs Automatisés** : Le service persiste également les objectifs nutritionnels calculés par l'IA ou les algorithmes métier (dailyCalorieTarget, dailyProteinTarget).
 - **Relation** : La liaison avec le compte utilisateur est de type 1-pour-1, assurée par le champ authUserId.
- 3. Food Service (Catalogue Alimentaire)** Ce service agit comme une bibliothèque de référence, totalement indépendante des utilisateurs.
 - **Entité Food** : Représente un aliment standardisé avec ses valeurs macro-nutritionnelles (calories, protéines, glucides, lipides) pour une portion donnée. Ce service ne gère aucune donnée personnelle, garantissant ainsi la générnicité du catalogue.
- 4. Meal Service (Suivi Nutritionnel)** C'est le service d'agrégation qui matérialise le journal alimentaire de l'utilisateur.
 - **Entité Meal** : Elle enregistre la consommation d'aliments à un moment précis (mealDate, MealType).
 - **Optimisation** : Pour des raisons de performance, l'entité Meal stocke directement les totaux calculés (totalCalories, totalProtein), évitant de recalculer la somme des nutriments à chaque consultation.
 - **Relations** : Elle lie l'utilisateur (authUserId) aux aliments consommés via une liste d'identifiants (List<Long> foodIds), sans jointure SQL directe avec le Food Service.
- 5. Water Service (Suivi de l'Hydratation)** Un microservice dédié assurant le suivi hydrique.
 - **Entité WaterGoal** : Définit l'objectif quotidien personnalisé (Relation 1-1 avec l'utilisateur).
 - **Entité WaterIntake** : Historique des consommations d'eau (Relation 1-N), permettant de suivre la progression au fil de la journée.

Cette architecture découpée assure une robustesse accrue : une indisponibilité du service d'hydratation n'impacte pas, par exemple, la capacité de l'utilisateur à s'authentifier ou à enregistrer ses repas.

b) Diagrammes de séquence

i) Cas 1 : Consultation de l'Agent IA (Flux nominal)

Ce premier diagramme décrit le scénario idéal où un utilisateur sollicite l'assistant virtuel. Il met en évidence une spécificité technique majeure de notre projet : l'enrichissement contextuel (RAG - Retrieval Augmented Generation). L'API ne se contente pas de relayer la question ; elle l'enrichit avec les données de santé de l'utilisateur avant d'interroger le modèle LLM.

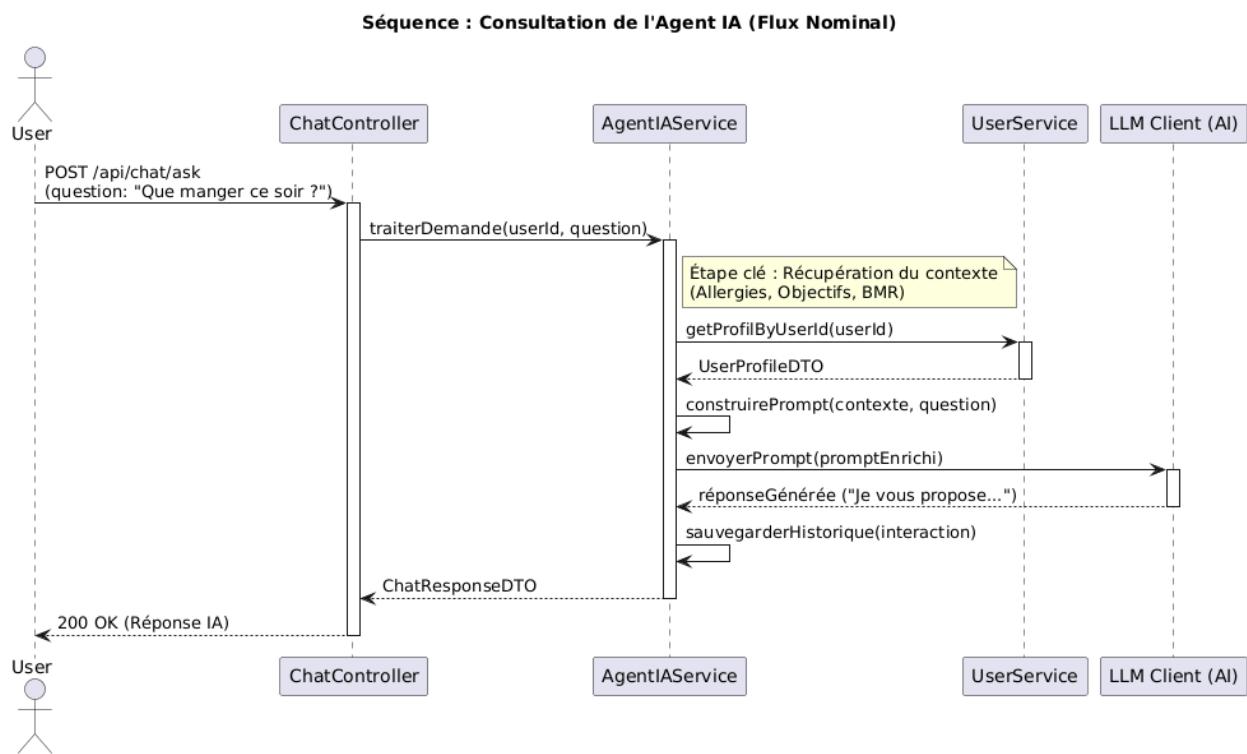


Figure 4 - Diagramme de séquence – Cas 1 : Consultation de l'Agent IA (Flux nominal)

Dans le cas d'une consultation de l'assistant virtuel, le scénario nominal décrit le flux attendu lorsqu'un utilisateur souhaite obtenir un conseil nutritionnel personnalisé. L'acteur principal, User, envoie une requête POST au ChatController contenant sa question formulée en langage naturel. Le contrôleur délègue ensuite le traitement de la demande à AgentIAService, le cœur logique du module. Avant d'interroger l'intelligence artificielle, ce service sollicite le UserService pour récupérer le profil de santé de l'utilisateur (objectifs, allergies, métabolisme), permettant ainsi d'enrichir le contexte de la requête. Une fois le prompt construit avec ces données physiologiques, le service envoie une requête au client LLM (modèle d'IA). L'IA traite les informations et retourne une réponse générée, qui est sauvegardée pour l'historique des conversations avant d'être encapsulée dans un objet de réponse (DTO). Enfin, le contrôleur retourne le conseil à l'utilisateur sous la forme d'un code HTTP 200 et du message textuel.

Ce diagramme met en évidence plusieurs points clés : la séquence précise des interactions, l'enrichissement contextuel des données (RAG) indispensable à la personnalisation, ainsi que la séparation des responsabilités entre la gestion du profil utilisateur et la logique conversationnelle. Il reflète également l'intégration fluide avec le modèle d'IA générative, assurant que chaque réponse est mathématiquement adaptée aux besoins de l'utilisateur.

En résumé, ce diagramme de séquence illustre de manière claire et détaillée le cheminement complet d'une interaction avec le coach virtuel, depuis la question initiale jusqu'à la réception d'un conseil sur mesure, permettant ainsi de comprendre comment le backend Spring Boot orchestre intelligemment les données de santé et l'IA.

ii) Cas 2 : Gestion des erreurs (Service IA indisponible)

Ce second diagramme illustre la robustesse du système face aux aléas externes. Si le service d'IA (API externe) est inaccessible ou rencontre un délai d'attente (timeout), l'application doit gérer l'incident proprement sans "crasher".

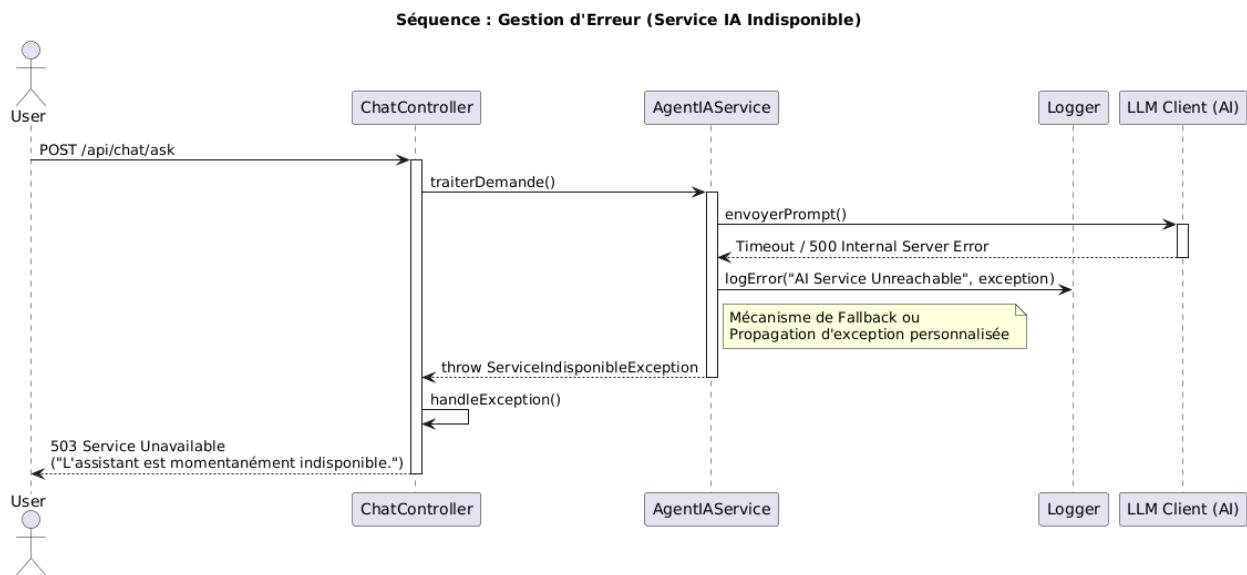


Figure 5 - Diagramme de séquence – Cas 2 : Erreur Azure OpenAI + gestion

Dans le cas d'une défaillance du service d'intelligence artificielle, le scénario alternatif décrit le comportement robuste du système face à une indisponibilité externe. Lorsqu'un utilisateur soumet une requête via l'API, le ChatController transmet la demande à AgentIAService qui tente d'interroger le client LLM. Si cet appel échoue (par exemple, en cas de Timeout ou d'erreur interne 500 du fournisseur d'IA), le service capture

immédiatement l'exception technique. Il consigne alors l'incident via le composant Logger pour assurer la traçabilité et le diagnostic ultérieur. Au lieu de provoquer un arrêt brutal de l'application, le service propage une exception métier personnalisée (ServiceIndisponibleException) vers le contrôleur. Ce dernier intercepte l'erreur grâce à un gestionnaire global et retourne une réponse structurée à l'utilisateur avec un code HTTP 503 et un message explicatif adapté.

Ce diagramme met en évidence plusieurs points clés : la résilience de l'architecture face aux pannes externes, la gestion centralisée des exceptions et l'importance de la traçabilité des erreurs techniques. Il reflète également la volonté de protéger l'expérience utilisateur en fournissant un feedback clair et rassurant, même lorsque le service fonctionne en mode dégradé.

En résumé, ce diagramme de séquence illustre les mécanismes de défense du système, garantissant que l'application reste stable, sécurisée et informative même lorsque les dépendances externes critiques, comme le moteur d'IA, rencontrent des dysfonctionnements majeurs.

c) *Diagrammes de séquence – Flux Métier*

Cette section détaille les interactions internes du système pour les fonctionnalités de suivi quotidien, mettant en lumière l'architecture en couches de Spring Boot (Controller, Service, Repository) et la gestion de la base de données.

i) *Cas 3 : Enregistrement d'un Repas (Tracking)*

Ce diagramme illustre le processus complet lorsqu'un utilisateur enregistre ce qu'il a mangé. Contrairement aux interactions avec l'IA, ce flux est déterministe et transactionnel : il implique la vérification des aliments, le calcul précis des macronutriments et la sauvegarde durable dans la base de données PostgreSQL.

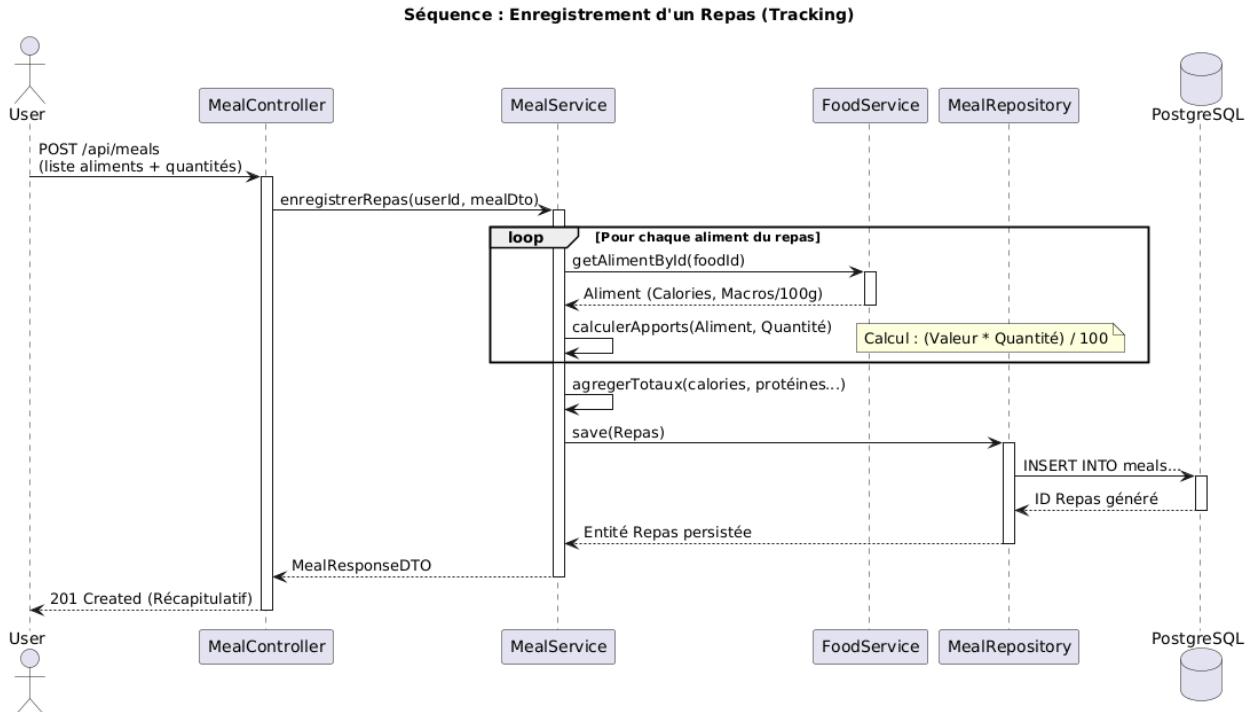


Figure 6 - Diagramme de séquence – Cas 3 : Enregistrement d'un Repas (Tracking)

Dans le cas de l'enregistrement d'un repas, le scénario nominal décrit le flux transactionnel permettant d'ajouter des données au journal alimentaire. L'acteur principal, User, envoie une requête POST au MealController contenant la liste des aliments consommés et leurs quantités respectives. Le contrôleur délègue la logique métier au MealService. Ce dernier itère sur chaque élément de la liste et interroge le FoodService pour récupérer les valeurs nutritionnelles de référence (calories, protéines, glucides, lipides pour 100g). Pour chaque aliment, le service effectue un calcul précis des apports réels en fonction de la quantité ingérée, puis agrège ces valeurs pour obtenir les totaux du repas. Une fois les données consolidées, l'objet Repas est transmis au MealRepository qui exécute l'insertion SQL dans la base de données PostgreSQL. La base de données confirme la transaction et renvoie l'identifiant généré. Enfin, le service convertit l'entité persistée en un objet de réponse (DTO) que le contrôleur retourne à l'utilisateur avec un code HTTP 201 Created, confirmant la prise en compte du suivi.

Ce diagramme met en évidence la rigueur de la gestion des données : contrairement au chat avec l'IA qui est probabiliste, le module de tracking assure une exactitude mathématique des calculs et une intégrité référentielle des données. Il illustre également le découpage propre entre le service de catalogue (FoodService) et le service de journal (MealService).

En résumé, ce diagramme de séquence montre comment l'application garantit un suivi nutritionnel fiable et performant, en s'appuyant sur la robustesse du framework Spring Boot pour gérer les règles métier et la persistance des données.

Conclusion

Ce chapitre a permis de formaliser la conception conceptuelle du projet en s'appuyant sur la méthodologie UML, offrant une représentation claire et structurée de l'architecture des données et des interactions dynamiques du système.

L'utilisation de diagrammes variés – de classes et de séquence – a facilité la visualisation des relations entre les entités, des flux de données et des comportements attendus, tout en identifiant les responsabilités de chaque microservice.

Le diagramme de classes a mis en évidence la structure interne de l'application, détaillant les entités critiques telles que le profil utilisateur, les repas et les aliments, ainsi que leurs relations logiques, garantissant une architecture de données cohérente et découplée.

Les diagrammes de séquence ont illustré le déroulement des processus critiques. D'une part, ils ont clarifié la logique complexe de l'Agent IA, notamment l'enrichissement contextuel du prompt et la gestion robuste des erreurs externes. D'autre part, ils ont détaillé le flux métier transactionnel de l'enregistrement des repas, démontrant la rigueur des calculs nutritionnels implémentés dans le backend.

Cette étape conceptuelle constitue une base solide pour la phase de réalisation, garantissant que le développement respectera les besoins fonctionnels identifiés, tout en facilitant la maintenance et l'évolution future du système.

En synthèse, ce chapitre illustre la cohérence entre l'architecture logicielle, les flux de données et les exigences du projet, préparant efficacement la transition vers le développement pratique et la mise en œuvre technique de l'application qui sera abordée dans la suite de ce rapport.

Chapitre IV – Étude Technique

1. Introduction

Ce chapitre présente l'analyse technique approfondie du système Makla, décrivant minutieusement l'architecture logicielle, les choix technologiques, les modalités de déploiement et les contraintes opérationnelles. L'objectif est de fournir une vision exhaustive de l'infrastructure technique permettant d'assurer le développement, le déploiement et la maintenance optimale de la solution.

2. Architecture globale du système

L'architecture globale du système Makla a été conçue selon une approche microservices modernes visant à garantir la modularité, les performances élevées et la scalabilité horizontale, tout en facilitant l'intégration transparente avec les services cloud avancés et les modèles d'intelligence artificielle générative. Cette architecture repose sur plusieurs couches logicielles distinctes mais parfaitement intégrées, permettant une séparation claire des responsabilités et assurant ainsi la maintenabilité optimale du code sur le long terme.

2.1. Couches logicielles

Couche Présentation (Frontend)

Rôle : interface utilisateur, collecte des entrées (inscription, connexion, saisie de repas, affichage de recommandations), validation côté client et rendu.

Technologies : Angular (TypeScript), serveur de développement Node.js/NPM.

Composants : pages de connexion, tableau de bord, formulaires de saisie des repas, affichage des historiques et recommandations IA.

Remarque : communique exclusivement avec la couche API via l'API Gateway (proxy côté frontend).

Couche API / Gateway

Rôle : point d'entrée unique pour le frontend et les clients externes, routage vers les microservices, centralisation des politiques transverses (authentification, throttling, logging).

Technologies : Spring Cloud Gateway (ou équivalent), config gérée par le Config Server.

Fonctionnalités : routage des requêtes REST, gestion des erreurs globales, intégration JWT pour valider les tokens d'authentification.

Couche Authentification & Autorisation

Rôle : gestion des utilisateurs, création / validation des tokens JWT, contrôle d'accès aux ressources.

Technologies : service Spring Boot dédié (Auth Service), base de données (PostgreSQL) pour les comptes, chiffrement des mots de passe.

Composants : endpoints login/register, génération/renouvellement des tokens, filtres de sécurité côté gateway ou service.

Couche Métier (Microservices applicatifs)

Rôle : implémentation des règles métier réparties en microservices indépendants.

Principaux services :

User Service : gestion des profils utilisateurs.

Food Service : catalogue d'aliments et données nutritionnelles.

Meal Service : gestion des repas (création, calcul nutriments).

Water Service : suivi de la consommation d'eau.

Nutrition Service : calculs et règles de nutrition (agrégation, objectifs).

Notification Service : envoi de notifications (email/push).

Technologies : Spring Boot (Java 17), services packagés via Maven.

Communication : REST synchrone via le Gateway ou appels internes, découverte via Eureka.

Couche Persistance (Data)

Rôle : stockage durable des données (utilisateurs, aliments, repas, journaux).

Technologies : PostgreSQL (pour Auth et autres services nécessitant SQL), éventuellement bases NoSQL pour caches ou logs si requis.

Remarque : chaque microservice possède son propre schéma/BDD (principe de découplage des données).

Couche Découverte et Configuration

Rôle : service discovery pour l'auto-découverte des instances (Eureka) et centralisation de la configuration (Config Server).

Technologies : Spring Cloud Eureka, Spring Cloud Config.

Avantages : orchestration plus facile des instances, modification centralisée des propriétés sans redéploiement global.

Couche IA / Recommandation

Rôle : fourniture de recommandations nutritionnelles personnalisées en utilisant un modèle IA et une stratégie RAG (Retrieval-Augmented Generation) pour exploiter la base documentaire du projet.

Technologies : service IA dédié (Ollama + modèle Phi3 ou équivalent), mécanisme de recherche documentaire pour RAG, microservice "AI Service".

Fonctionnalités : traitement des requêtes de recommandation, enrichissement par documents pertinents, réponses textuelles et suggestions de repas.

2.2. Flux principal

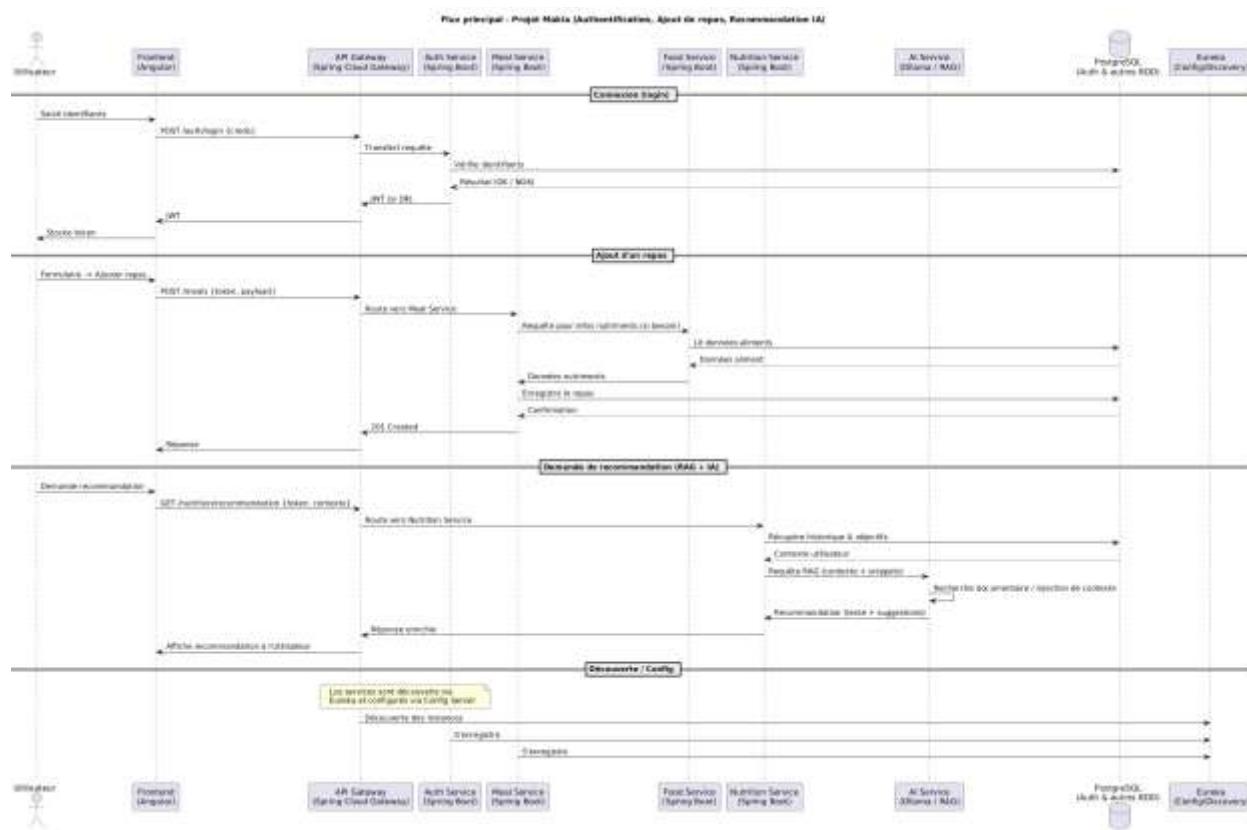


Figure 7 - Architecture globale du système

Explications et points d'attention

Le diagramme montre trois cas clés : authentification, création d'un repas et obtention d'une recommandation IA (RAG).

Le flux RAG : le service Nutrition envoie au service AI le contexte utilisateur + extraits documentaires pour que le modèle génère une réponse contextualisée.

Chaque microservice peut interroger sa propre base (principe BDD par service) — ici simplifié via un PostgreSQL commun pour Auth et autres exemples.

Eureka et Config Server gèrent la découverte et la configuration (mentionnés en note).

2.3. Rôle et architecture du Service IA

But : fournir des recommandations nutritionnelles personnalisées en combinant (a) un modèle de génération de texte (LLM) et (b) une base documentaire (documents nutritionnels, fiches aliments, règles métier) pour ancrer les réponses et éviter les hallucinations.

Position dans l'architecture : appelé par le Nutrition Service (ou directement via Gateway) ; reçoit contexte utilisateur (historique, objectifs) et renvoie une recommandation textuelle structurée + metadata (sources, score).

Pipeline RAG

étapes détaillées Étape 1 — Ingestion des documents

Sources : fiches nutritionnelles (Food Service), documentation projet, articles, règles métier, FAQ.

Prétraitement : nettoyage HTML/Markdown, normalisation (unicodes), suppression des parties non pertinentes (scripts), et extraction des métadonnées (titre, source, date).

Étape 2 — Segmentation / chunking

On découpe chaque document en "chunks" (par ex. 200–800 tokens) en veillant à:
garder le contexte logique (ne pas couper une phrase au milieu),
superposer légèrement (overlap ~20–30 %) pour préserver la continuité.
Stocker pour chaque chunk : texte, doc_id, position, métadonnées.

Étape 3 — Calcul des embeddings

Modèle d'embedding recommandé : modèle dense de type sentence-transformers (ex : all-mpnet-base-v2) ou embeddings fournis si Ollama propose un encodeur local.

Représentation : vecteur flottant (dim 384–1536).

Normalisation : L2/float32 ou float16 pour économie mémoire.

Étape 4 — Indexation (vector store)

Choix d'implémentation : FAISS (local), Milvus, Weaviate, Pinecone (cloud) selon contrainte infra.

Stratégie : index HNSW ou IVF+PQ pour latence/compromis mémoire. Persister l' index + mapping chunk→métadonnées.

Étape 5— Retrieval (au moment de la requête)

Input : prompt utilisateur + contexte (tokenisé) + JWT → accès.

Création d'une requête d'embedding à partir du contexte (history + question).

Récupération top-k chunks (k typiquement 3–10).

Optionnel : reranking sémantique (cross-encoder) pour améliorer précision.

Étape 6 — Construction du prompt (injection)

Pattern : System prompt (instr. modèle) + Instruction utilisateur + Contextual evidence (chunks) + Constraints (format attendu).

Limiter la taille insérée (ce que tient la fenêtre contextuelle)— choisir les chunks les plus pertinents.

Exemple de template (voir plus bas).

Étape 7 — Génération et post-traitement

Model génère texte. Post-traitement : extraction de la structure (recommandations, raisons, sources citées).

Joindre un champ sources (liste des doc_id et extraits utilisés) pour traçabilité.

Prompt engineering — template et bonnes pratiques Template simple (en français) :

System (contexte): "Vous êtes un assistant nutritionnel qui fournit des recommandations basées sur les documents fournis. Répondez clairement, citez la source (id) pour chaque affirmation importante et retournez le résultat en JSON avec champs {summary, actions, sources}."

User: "Contexte utilisateur: {objectif, allergies, historique}. Question: {question}. Documents: {chunks...}" Bonnes pratiques :

Donner un format de sortie clair (JSON) pour parsing automatique.

Réduire ambiguïté : dire explicitement au modèle d'utiliser uniquement les documents fournis pour les faits.

Insérer un "safety check" : si info manquante, dire "insuffisant".

Exemple de prompt : System: Vous êtes un assistant nutritionnel... User: Contexte utilisateur: "femme 30 ans, 65kg, végétarienne"... Documents: [DOC 23] "Valeur nutritive du quinoa..." [DOC 47] "Règles apport protéique pour végétariens..." Question: "Proposez un menu journalier avec apport protéique 60g."

Stratégies de fusion / variantes RAG

Retrieve-and-Generate (RAG simple) : injecter chunks puis générer.

Two-step (retrieve + rerank + generate) : retrieve large set → rerank → generate (meilleure précision).

Retrieval-only + Template fill : extraire faits puis remplir un template (utile si vous voulez citations strictes).

Closed-book + citation via retrieval : forcer le modèle à "se baser uniquement sur les chunks".

Pipeline RAG Vertical - Makla

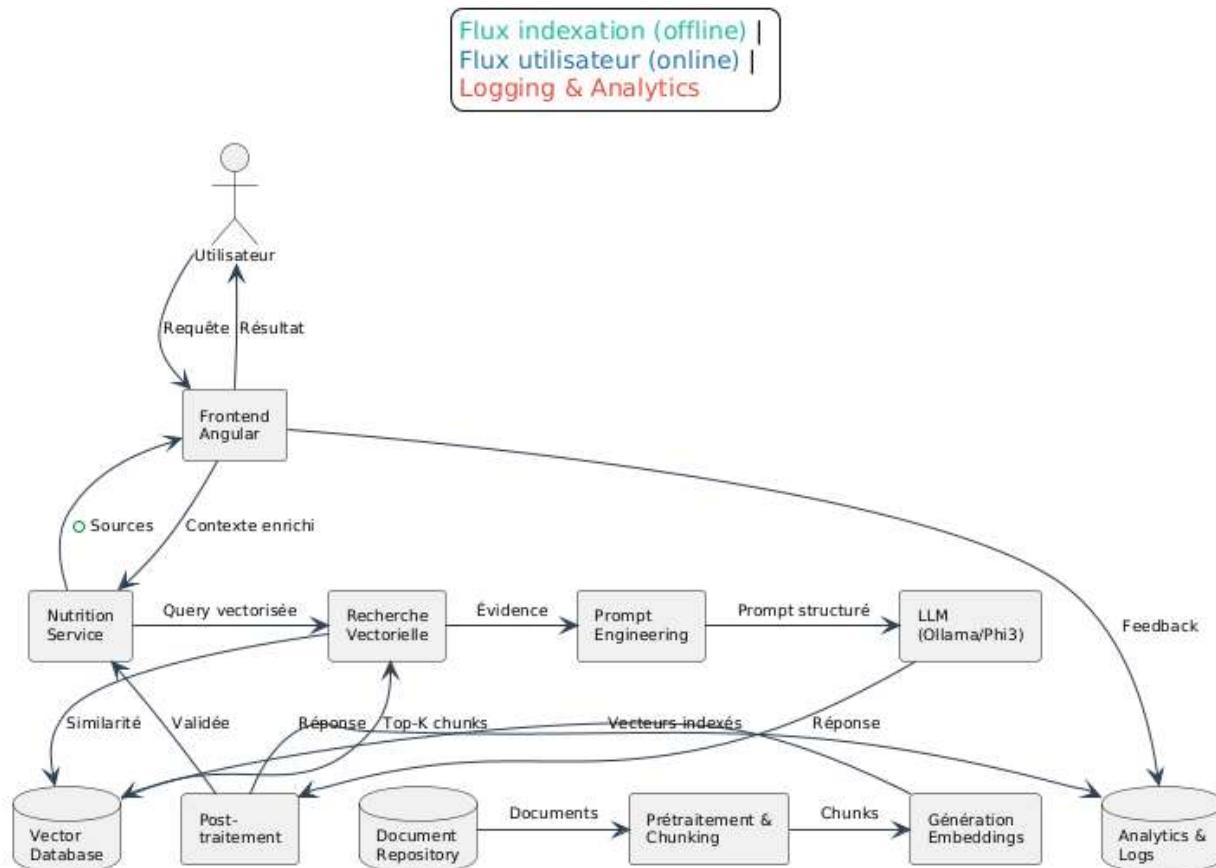


Figure 8 - Pipeline RAG

2.4. Stratégie de fine-tuning pour la personnalisation du modèle

Objectif et portée du fine-tuning

Le processus de fine-tuning a pour objectif de spécialiser le modèle de langage Phi3 sur les spécificités du domaine nutritionnel du projet Makla. Cette spécialisation vise à aligner le comportement du modèle sur trois dimensions essentielles : l'adoption d'un style de communication adapté aux applications de santé, la maîtrise des règles métier nutritionnelles spécifiques, et la production systématique de réponses structurées selon un format JSON prédéfini avec un haut degré de précision nutritionnelle. Cette adaptation permet de transformer un modèle générique en assistant nutritionnel spécialisé, réduisant ainsi les hallucinations et améliorant la fiabilité des recommandations fournies aux utilisateurs.

Méthodologies de fine-tuning

a) Supervised fine-tuning (SFT) complet

Le supervised fine-tuning complet représente l'approche la plus exhaustive pour adapter le modèle. Cette méthode nécessite la préparation d'un jeu de données structuré contenant des paires instruction-complétion au format JSONL. Chaque entrée du dataset inclut une instruction simulant une requête utilisateur et une réponse correspondant à la sortie attendue, intégrant des citations vers les sources documentaires et respectant le format de réponse JSON imposé par les spécifications techniques.

La répartition du dataset suit une distribution standard avec 80 % des données allouées à l'entraînement, 10 % à la validation et 10 % aux tests finaux. L'infrastructure matérielle requiert des unités de traitement graphique de type A100 ou RTX 4090, avec possibilité d'extension en configuration multi-GPU pour optimiser les temps de traitement. La durée d'entraînement varie de quelques heures à plusieurs jours selon la taille du dataset et la configuration matérielle.

Les hyperparamètres d'entraînement sont calibrés spécifiquement pour les modèles de langage de grande taille. La taille des lots (batch_size) est fixée entre 8 et 32 selon les capacités mémoire du GPU. Le taux d'apprentissage (learning rate) est configuré entre 1e-5 et 5e-5. Le nombre d'époques est limité à 2-5 pour éviter le sur-apprentissage, avec une phase de préchauffage couvrant les 500 premières étapes d'entraînement. L'implémentation technique s'appuie sur les bibliothèques Hugging Face Transformers et Accelerate. Une stratégie de sauvegarde de points de contrôle est mise en place à intervalles réguliers, couplée à un mécanisme d'arrêt anticipé basé sur la perte de validation.

b) Fine-tuning efficace via LoRA (low-rank adaptation)

La méthode LoRA représente une alternative plus légère au fine-tuning complet, particulièrement adaptée aux contraintes de ressources du projet Makla. Cette technique opère par l'ajout de matrices de faible rang aux couches d'attention du modèle,

permettant une adaptation ciblée sans modifier la totalité des poids pré-entraînés. Les avantages principaux incluent une réduction substantielle des besoins en mémoire GPU, une taille de stockage réduite et des temps d'entraînement accélérés.

Le processus d'implémentation LoRA commence par la conversion du jeu de données au format attendu par la bibliothèque, établissant une correspondance claire entre instructions et sorties attendues. L'utilisation conjointe des bibliothèques peft, transformers et accelerate permet une intégration transparente dans le pipeline d'entraînement. Les hyperparamètres spécifiques à LoRA sont configurés avec attention : le rang (r) est défini entre 4 et 16 selon la complexité de l'adaptation souhaitée, le paramètre alpha est positionné entre 16 et 32 pour contrôler l'échelle de l'adaptation, et un taux d'abandon (dropout) de 0,05 à 0,2 est appliqué pour régulariser l'apprentissage.

La séquence d'exécution technique suit un pattern établi : initialisation du tokenizer et du modèle de base à partir des poids pré-entraînés de Phi3, application de la configuration LoRA via peft.LoraConfig, transformation du modèle avec get_peft_model, puis lancement de l'entraînement via l'interface standard Trainer. Concernant la compatibilité avec l'écosystème Ollama, une vérification préalable détermine si la plateforme supporte l'importation de poids fine-tunés ou l'application directe d'adaptateurs LoRA. En cas de limitation, l'approche retenue consiste à effectuer le fine-tuning via l'écosystème Hugging Face puis à procéder à une exportation et conversion des poids selon les spécifications de compatibilité d'Ollama.

Constitution et annotation du jeu de données

La qualité du jeu de données constitue le facteur déterminant pour le succès du fine-tuning. Les sources de données privilégiées incluent les conversations réelles anonymisées issues des phases de test utilisateur, les questions fréquemment posées documentées dans la FAQ du système, et les corrections manuelles appliquées aux exemples où le modèle initial présente des hallucinations ou des imprécisions. Le format de stockage retenu est le JSONL, avec une structure incluant les champs instruction, input, output et metadata pour assurer une traçabilité complète.

Des techniques d'augmentation de données sont systématiquement appliquées pour enrichir le dataset et améliorer la robustesse du modèle. Ces techniques incluent la génération de paraphrases variées pour une même instruction, la permutation du contexte fourni, et l'inclusion délibérée de cas limites représentatifs des situations complexes rencontrées en nutrition, telles que les allergies multiples, les conditions médicales spécifiques ou les régimes alimentaires contraignants. Concernant le volume de données, un minimum de quelques milliers d'exemples de haute qualité est nécessaire pour obtenir des résultats significatifs avec un fine-tuning complet. Pour l'approche LoRA, quelques centaines à un millier d'exemples peuvent suffire pour une adaptation stylistique et structurelle.

c) ***Évaluation et métriques de performance***

L'évaluation du modèle fine-tuné repose sur une combinaison de métriques automatiques et de tests humains. Les métriques automatiques incluent la perplexité pour évaluer la cohérence linguistique du modèle après fine-tuning SFT, les scores BLEU et ROUGE pour mesurer la similarité superficielle avec les réponses de référence, la similarité sémantique calculée via les embeddings pour évaluer la préservation du sens, et le taux de correspondance exacte pour les sorties structurées en JSON.

Les tests humains restent indispensables pour évaluer des dimensions qualitatives non capturables par des métriques automatiques. Ces tests évaluent la pertinence contextuelle des réponses, leur exactitude factuelle par rapport aux connaissances nutritionnelles établies, leur sécurité en évitant tout conseil potentiellement dangereux, et leur utilité pratique pour l'utilisateur final. Des scénarios d'usage réalistes sont conçus pour simuler des interactions complètes, permettant une évaluation holistique des performances dans des conditions proches de la production. Des tests de robustesse complémentaires sont exécutés, incluant des injections de prompts adverses, des requêtes mal formées intentionnellement, et des scénarios d'absence d'information dans les sources disponibles.

3. Outils et Framework de développement

3.1. Environnement de développement

IntelliJ IDEA Ultimate Edition constitue l'environnement de développement intégré (IDE) principal pour le développement du backend Java/Spring du projet Makla. Cet IDE, développé par JetBrains, offre un ensemble complet d'outils et de fonctionnalités spécialement conçus pour le développement d'applications Java d'entreprise, avec un support exceptionnel pour l'écosystème Spring et les architectures microservices.



Figure 9 - Logo de IntelliJ

Pour la gestion du code source, le système Git a été utilisé. Git est un gestionnaire de versions distribué qui permet de suivre l'évolution du projet, de gérer les branches et de collaborer efficacement tout en garantissant l'intégrité du code .



Figure 10 - Logo de Git

3.2. Framework Backend

Le backend est développé en Java 17 avec Spring Boot 3.2.0 et Spring Cloud 2023.0.0, suivant une architecture microservices. La gestion des dépendances utilise Maven via le wrapper mvnw pour garantir la reproductibilité des builds. La persistance des données est assurée par PostgreSQL pour le service d'authentification, choisi pour sa fiabilité et son respect des propriétés ACID.

La découverte des services est gérée par Eureka Server, permettant l'enregistrement dynamique des microservices. L'API Gateway centralise le routage des requêtes et implémente des mécanismes de sécurité et de résilience. Les services communiquent via HTTP/REST avec JSON, et l'authentification repose sur des tokens JWT validés au niveau de la gateway.

La configuration est externalisée via Spring Cloud Config Server, et l'observabilité est assurée par des métriques exposées via Actuator. Les tests couvrent unitaires, intégration et contrats d'API. Le déploiement utilise Docker pour la conteneurisation et Kubernetes pour l'orchestration en production.

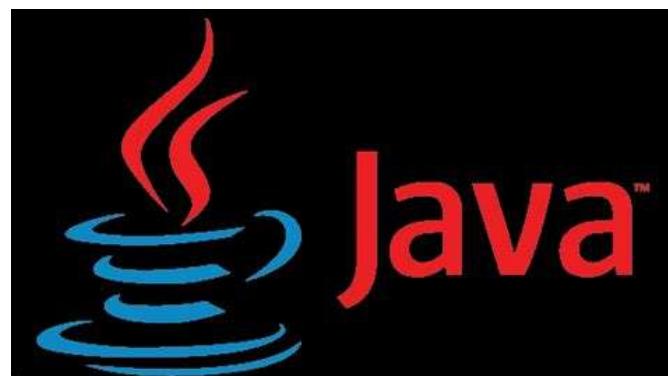


Figure 11 - Logo de java

3.3. Outils Frontend & API Testing

Bien que ce projet ne dispose pas d'une interface graphique dédiée, deux outils principaux ont été utilisés pour assurer l'interaction avec l'API et la validation de son bon fonctionnement : **Swagger (OpenAPI)** et Postman.



Figure 12 - Logo Angular

Cette section présente les outils et méthodologies employés pour le développement de l'interface utilisateur et la validation des interfaces de programmation applicatives. L'objectif est de garantir la qualité des interfaces graphiques et la fiabilité des points d'extrémité backend au travers de procédures de test structurées, combinant approches manuelles et automatisation.

Écosystème de Développement Frontend

L'environnement de développement frontend repose sur Angular 20, framework permettant la création d'applications web monopages évolutives. TypeScript 5.8 apporte un typage statique renforçant la robustesse du code. Node.js fournit l'environnement d'exécution nécessaire, tandis que npm gère les dépendances logicielles et les scripts d'automatisation.

Stratégie de Test des API

La validation des interfaces backend s'appuie sur une approche multi-niveaux utilisant des outils complémentaires. Postman constitue l'outil principal pour les tests interactifs et la création de collections d'API réutilisables. Newman, version en ligne de commande de Postman, permet l'exécution automatisée de ces collections dans les pipelines.

d'intégration continue. L'utilitaire curl sert aux vérifications rapides et aux scripts de diagnostic simples.

Conclusion

L'étude technique complète du projet Makla a permis d'élaborer une architecture cohérente et robuste, parfaitement adaptée aux exigences spécifiques d'une plateforme de nutrition personnalisée. La démarche d'analyse a systématiquement privilégié des solutions techniques matures tout en intégrant des innovations pertinentes dans les domaines de l'intelligence artificielle et des architectures distribuées.

Synthèse de l'Architecture Globale

L'architecture microservices retenue s'avère particulièrement adaptée au caractère modulaire des fonctionnalités nutritionnelles. La décomposition en services spécialisés (authentification, gestion des utilisateurs, suivi nutritionnel, intelligence artificielle) permet une évolution indépendante de chaque composant, facilitant ainsi les itérations rapides et les mises à jour ciblées. L'API Gateway centralisée constitue un point d'entrée unique qui simplifie la gestion de la sécurité et du routage, tout en offrant une façade cohérente aux clients frontaux.

Évaluation des Choix Technologiques

Les technologies sélectionnées présentent un équilibre judicieux entre maturité et modernité. Le couple Java 17 / Spring Boot 3.2 offre une base solide pour le backend, bénéficiant d'un écosystème riche et d'une communauté active. L'intégration de Spring Cloud pour les fonctionnalités de microservices (découverte, configuration, résilience) garantit une cohérence technique et réduit la complexité d'intégration.

Le choix d'Angular pour le frontend assure la construction d'une interface utilisateur performante et maintenable, avec des mécanismes de réactivité adaptés aux mises à jour fréquentes des données nutritionnelles. L'adoption de TypeScript renforce la robustesse du code frontend tout en améliorant la productivité des développeurs.

Chapitre V – Réalisation

1. Introduction

Ce chapitre présente la matérialisation concrète du projet Makla à travers son implémentation fonctionnelle. Il détaille la réalisation effective de l'application, depuis le développement des microservices backend jusqu'à l'interface utilisateur frontale, en passant par l'intégration du système d'intelligence artificielle. L'objectif est de démontrer comment les choix architecturaux et technologiques définis dans les chapitres précédents se sont traduits en une application opérationnelle, tout en mettant en lumière les fonctionnalités clés développées pour répondre aux besoins métier identifiés.

La démonstration de l'application s'organise autour de trois volets complémentaires : l'exposition des interfaces utilisateur et leur ergonomie, la présentation des fonctionnalités métier principales, et la validation technique des composants sous-jacents. Cette approche permet d'appréhender l'application tant du point de vue de l'utilisateur final que sous l'angle technique de son implémentation.

Ce chapitre vise également à illustrer le parcours utilisateur type au sein de l'application, depuis l'authentification jusqu'à l'obtention de recommandations nutritionnelles personnalisées, en passant par la saisie et le suivi des données alimentaires. La démonstration concrète des fonctionnalités permettra d'évaluer l'adéquation entre les besoins initiaux et la solution technique déployée.

2. Présentation du résultat de l'application

2.1. Interface Angular (Login \ Register)

The screenshot shows the 'Créer un compte' (Create account) form. It features a header with the Makla logo and a sub-header 'Répondez à quelques questions pour créer votre profil'. The form includes fields for 'Prénom' (First name), 'Nom' (Last name), 'Nom d'utilisateur' (Username), 'Email' (Email), 'Mot de passe' (Password), 'Confirmer le mot de passe' (Confirm password), and 'Confirmez votre mot de passe' (Confirm your password). A purple 'Créer mon compte' (Create my account) button is at the bottom.

Figure 14 - Register page

The screenshot shows the 'Connexion' (Login) form. It features a header with the Makla logo and a sub-header 'Répondez à quelques questions pour créer votre profil'. The form includes fields for 'Nom d'utilisateur' (Username) and 'Mot de passe' (Password). A purple 'Se connecter' (Connect) button is at the bottom. Below the button, a note says 'Une fois sur votre tableau de bord, vous pouvez modifier vos paramètres.' (Once on your dashboard, you can modify your settings.)

Figure 13 - Login page

Résumé du fonctionnement (haut niveau)

Enregistrement (`register`):

Le frontend envoie un POST vers `/api/auth/register` avec email, mot de passe, etc.

Le `Auth Service` vérifie l'unicité, hache le mot de passe (BCrypt) et enregistre l'utilisateur dans `PostgreSQL`.

Retourne `201 Created` ou `409` si l'email existe.

Connexion (`login`):

Le frontend envoie un POST vers `/api/auth/login` avec email + mot de passe.

Le service authentifie (AuthenticationManager / UserDetailsService) en comparant le mot de passe haché.

Si succès, génère un JWT signé et le renvoie (plus info utilisateur).

Le frontend stocke le token (ex: `localStorage`) et ajoute l'en-tête `Authorization: Bearer <token>` aux requêtes suivantes.</token>

L'API Gateway / chaque microservice valide le JWT sur les requêtes protégées.

2.2. Interface Settings préférence utilisateur

The screenshot shows a user profile settings page with the following sections:

- Your Profile:** Fields for Age (74), Weight (kg) (106), Height (cm) (170), and Gender (Male). A large blue bar displays "Your BMI: 36.7" with a "Change" button.
- Activity & Goals:** Activity Level (Moderately Active (3-5 days/week)) and Your Goal (Radio buttons for Lose Weight, Maintain, or Gain Weight). A yellow "Calculate Recommended Goals" button is present.
- Health Conditions:** A section to "Select any conditions that apply to you for personalized recommendations." Buttons for Diabetes Type 1, Diabetes Type 2, High Blood Pressure, High Cholesterol, Celiac Disease, Food Allergies, Heart Disease, and Kidney Disease are shown.
- Dietary Preferences:** A section titled "Your dietary restrictions and requirements".

Figure 15 - 2.2. Interface Settings préférence utilisateur

Le système de préférences utilisateur de Makla permet une expérience hautement personnalisée, adaptée aux besoins individuels de chaque utilisateur dans son parcours nutritionnel. Les paramètres configurables s'articulent autour de plusieurs dimensions essentielles.

- **Unités de Mesure**

Pour accomoder les utilisateurs du monde entier, le système permet de basculer entre le système métrique (grammes, millilitres) et le système impérial (onces, cups). Cette conversion s'applique de manière cohérente à travers toute l'application, notamment dans les recettes, les quantités d'ingrédients et les objectifs nutritionnels.

- **Objectifs Nutritionnels Quotidiens**

Chaque utilisateur peut définir ses objectifs personnels, incluant :

- **Apport calorique cible**: Basé sur le métabolisme basal, le niveau d'activité et les objectifs de poids
- **Consommation hydrique**: Quantité d'eau recommandée en millilitres, personnalisée selon le poids et l'activité
- **Répartition des macronutriments**: Pourcentages cibles de protéines, glucides et lipides

2.3. Interface Agent et chatbot

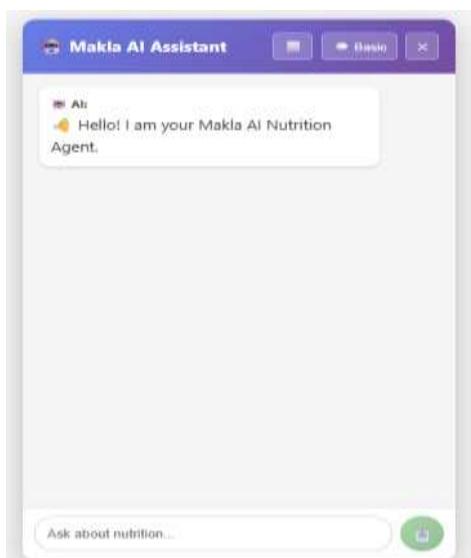


Figure 16 - Chatbot

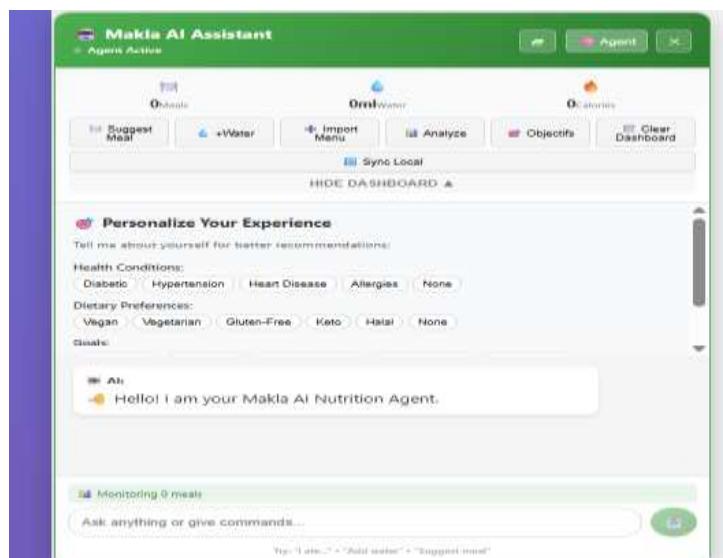


Figure 17 - Agent AI

Chatbot « normal » vs « agent mode »

Chatbot normal

Définition : modèle de langage qui répond aux entrées utilisateur en se basant sur le contexte de la conversation et ses connaissances internes (prompt + historique).

Capacités : génération de texte, Q/A, reformulation, assistance conversationnelle.

Accès externe : généralement aucun accès direct à des services/outils externes (ou via étapes séparées : RAG pour documents).

Flux typique : User → Prompt → Modèle → Réponse.

Usage : FAQ, support utilisateur, conversation générale, réponses rapides.

Agent mode

Définition : modèle qui, en plus de générer du texte, peut planifier et invoquer des outils/exécutables (API, base de données, recherche, exécution de commandes) puis réagir en fonction des résultats.

Capacités : actions multi-étapes (planifier, appeler outil, interpréter sortie), boucles « think/act/observe », intégration de plugins ou microservices.

Accès externe : accès contrôlé à des outils (ex : recherche, exécution SQL, services internes, génération d'images).

Flux typique : User → Modèle planifie → Modèle appelle outil(s) → Observation → Modèle finalise réponse.

Usage : automatisation de workflows, recherches en temps réel, réponses nécessitant accès à données fraîches ou opérations (exécution de recette, réservation, génération IA personnalisée).

2.4. DashBoard

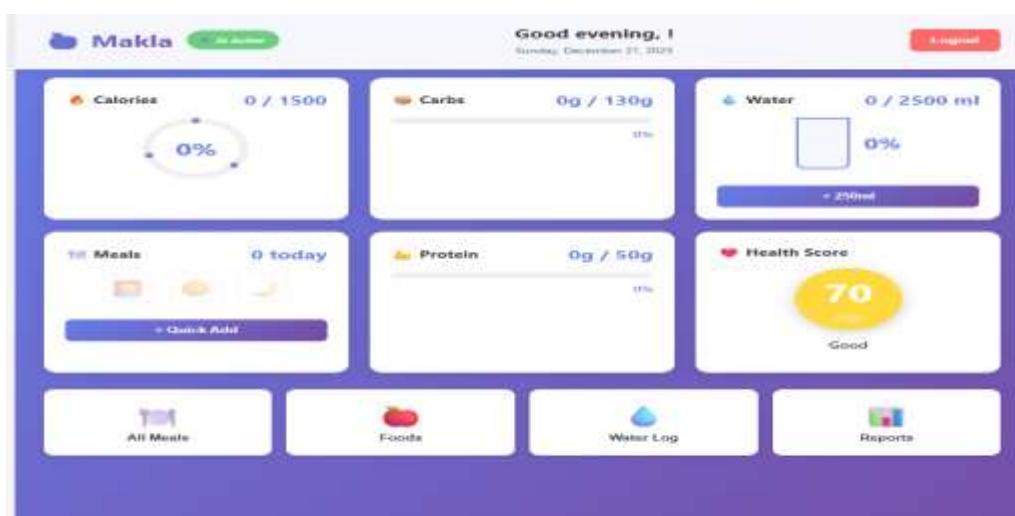


Figure 18 - Dashboard

Le dashboard principal centralise l'état quotidien de l'utilisateur et propose actions rapides. Sections typiques :

En-tête : nom utilisateur, bouton paramètres, résumé rapide (calories consommées / objectif).

Carte "Aujourd'hui" : liste des repas du jour, total calories/macros, bouton 'Ajouter un repas'.

Hydratation : total eau consommée, progression vers l'objectif, bouton 'Ajouter eau'.

Recommandations IA : suggestions de repas/plans générés par le service IA.

Historique / Activités récentes : modifications, synchronisations.

Widgets visuels : graphique calories/journées, breakdown macros, indicateurs d'objectif.

Flux de données et appels API

Chargement initial : au `ngOnInit`, récupérer en parallèle :

```
'GET /api/meals/today'  
'GET /api/water/today'  
'GET /api/user/preferences'  
'GET /api/ai/recommendation?date=today' (optionnel)
```

Conclusion

Synthèse des Réalisations d'Interface

Les interfaces développées pour l'application Makla démontrent une conception centrée utilisateur qui allie simplicité d'utilisation et fonctionnalité avancée. L'analyse des écrans de connexion et d'inscription révèle une approche méthodique de l'expérience utilisateur, où chaque élément d'interface sert un objectif précis dans le parcours utilisateur.

Cohérence et Identité Visuelle

L'application présente une identité visuelle cohérente à travers toutes ses interfaces. La palette de couleurs, la typographie et les principes de composition graphique sont appliqués uniformément, créant une expérience unifiée. Cette cohérence est particulièrement visible dans la transition fluide entre l'écran de connexion et celui d'inscription, où les utilisateurs reconnaissent immédiatement qu'ils interagissent avec la même application.

Conclusion Générale et Perspectives

Le projet Makla a abouti à la conception et au développement d'une plateforme complète de suivi nutritionnel intelligent, intégrant des technologies modernes et des approches innovantes. Les principales réalisations techniques comprennent :

Architecture Microservices Robustes : Implémentation d'un système distribué basé sur Spring Boot et Spring Cloud, avec une séparation claire des responsabilités entre les différents services (authentification, gestion utilisateur, nutrition, IA).

Pipeline d'Intelligence Artificielle Avancé : Développement d'un système RAG (Retrieval-Augmented Generation) combinant recherche vectorielle et modèles de langage pour fournir des recommandations nutritionnelles personnalisées et précises.

Interface Utilisateur Moderne et Accessible : Création d'une application Angular responsive, respectant les principes d'accessibilité et offrant une expérience utilisateur fluide sur tous les dispositifs.

Système de Préférences Personnalisables : Implémentation d'un mécanisme sophistiqué de gestion des préférences utilisateur, permettant une adaptation fine de l'expérience à chaque individu.

Infrastructure DevOps Complète : Mise en place de pipelines CI/CD, de conteneurisation Docker, et d'outils d'observabilité pour garantir la qualité et la maintenabilité du système.

Synthèse des réalisations

La première étape du projet a consisté en une étude préliminaire et fonctionnelle approfondie, permettant d'identifier les besoins fonctionnels et non fonctionnels, les acteurs du système ainsi que les contraintes techniques et opérationnelles. Cette étape a servi de fondation pour la conception d'une architecture cohérente et modulaire. Le choix d'UML comme méthode de modélisation a facilité la formalisation des différents composants et flux du système, à travers des diagrammes de classes, de séquences et de cas d'utilisation, assurant une compréhension claire et partagée des interactions entre les acteurs et l'application.

La phase de conception technique a conduit à la mise en place d'une architecture cloud sécurisée et robuste. L'API, développée en ASP.NET Core, interagit avec le service Azure OpenAI pour analyser les images et générer les prompts détaillés. L'infrastructure cloud, reposant sur Azure App Service et le support optionnel de Docker, assure scalabilité, disponibilité et maintenance facilitée. Les secrets et clés API sont gérés via les variables de pipeline et les App Settings, renforçant la sécurité. Les pipelines CI/CD implémentés garantissent la continuité des déploiements, la cohérence des versions et une intégration fluide des nouvelles fonctionnalités.

Les tests réalisés, notamment via Swagger UI et des requêtes POST multipart, ont confirmé la capacité de l'application à répondre aux besoins initiaux : générer des prompts détaillés, fidèles aux visuels originaux, et fournir un service rapide et stable. L'application est ainsi prête à être utilisée par des créateurs de contenu, des agences marketing ou des entreprises souhaitant automatiser la création de textes descriptifs pour des images.

Bénéfices et apports du projet

Modularité facilitant développement parallèle et évolutivité.

- Séparation claire des responsabilités permettant tests et maintenance.
- Expérience utilisateur améliorée par dashboard, préférences et recommandations IA.
- Base prête pour CI/CD, conteneurisation et déploiement en production.
- Fondations pour recherche personnalisée (RAG) et automatisation via agent IA.

Perspectives d'évolution

- Renforcer la sécurité : stockage sécurisé des tokens, rotation, audits et hardening CORS.
- Observabilité : métriques, tracing distribué (Prometheus / Jaeger) et logs centralisés.
- Scalabilité : orchestration Kubernetes et autoscaling des services critiques.
- IA : pipeline RAG complet, personnalisation des modèles, audits éthiques et confidentialité des données.
- Améliorations UX : offline-first, PWA / application mobile, accessibilité renforcée.
- Tests et automatisation : suites E2E, tests de charge et intégration continue/déploiement continu.

Conclusion générale

Le projet fournit une base robuste et extensible pour un produit de suivi nutritionnel moderne : architecture microservices, frontend réactif et intégration IA. Les choix techniques permettent une montée en charge progressive et des itérations rapides. Les priorités immédiates sont la consolidation de la sécurité, l'observabilité et l'amélioration des capacités IA pour transformer la preuve de concept en une solution opérationnelle et fiable.

Références

1. UML et Modélisation

- [1] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, 2nd ed., Addison-Wesley, 2005.
- [2] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed., Addison-Wesley, 2004.
- [3] UML.org, "UML resources." [En ligne]. Disponible : <https://www.uml.org/>

2. Développement Backend & Microservices (Spring Boot)

- [4] C. Walls, *Spring Boot in Action*, Manning Publications, 2016.
- [5] Spring.io, "Spring Boot Documentation." [En ligne]. Disponible : <https://docs.spring.io/spring-boot/docs/current/reference/html/>
- [6] C. Richardson, *Microservices Patterns: With examples in Java*, Manning Publications, 2018.
- [7] Baeldung, "REST with Spring Tutorial." [En ligne]. Disponible : <https://www.baeldung.com/rest-with-spring-series>

3. Développement Frontend (Angular)

- [8] Google, "Angular Documentation - Introduction to the Angular Docs." [En ligne]. Disponible : <https://angular.io/docs>
- [9] N. Murray, *ng-book: The Complete Book on Angular*, Fullstack.io, 2020.
- [10] Mozilla Developer Network (MDN), "TypeScript Documentation." [En ligne]. Disponible : <https://www.typescriptlang.org/docs/>

4. Intelligence Artificielle & NLP (Agent IA)

- [11] OpenAI, "GPT-4 Technical Report," 2023. [En ligne]. Disponible : <https://openai.com/research/gpt-4>
- [12] Lewis, P., et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks (RAG)," *Advances in Neural Information Processing Systems*, 2020.
- [13] LangChain, "Introduction to LLM Agents." [En ligne]. Disponible : <https://python.langchain.com/docs/modules/agents/>
- [14] Vaswani, A., et al., "Attention Is All You Need," *Advances in Neural Information Processing Systems*, 2017. (Papier fondateur des Transformers).

5. Nutrition & Santé Numérique (Contexte)

- [15] Organisation Mondiale de la Santé (OMS), "Digital Health Strategy 2020-2025." [En ligne]. Disponible : <https://www.who.int/>
- [16] Higgins, J. P., "Smartphone Applications for Patients' Health and Fitness," *The American Journal of Medicine*, 2016.

6. Bases de données & Sécurité

- [17] PostgreSQL Global Development Group, "PostgreSQL Documentation." [En ligne]. Disponible : <https://www.postgresql.org/docs/>
- [18] IETF, "JSON Web Token (JWT) - RFC 7519." [En ligne]. Disponible : <https://tools.ietf.org/html/rfc7519>