Institut Supérieur des Etudes Technologiques de Zaghouan

Département des Technologies de l'informatique

# Projet de fin d'Etudes

# Conception et développement d'une application de gestion des incédents

Organisme d'accueil

**Elaboré par :**

**MOKHTAR TOUMI , RAWEN AMIRI**

**Encadrante académique**
**RIM BRAHMI**

**Encadrant en entreprise**
**AMINE TAIEB**

# *Dédicases*

**À mes chers parents,**

Votre amour, votre soutien indéfectible et vos sacrifices m'ont porté tout au long de ce chemin. Merci pour tout ce que vous faites pour moi.

**À ma sœur,**

Ton sourire et ta présence sont une source constante d'inspiration. Merci d'être toujours là.

**À mes amis,**

Vous avez rendu ce parcours plus léger et joyeux. Merci pour vos encouragements et les moments partagés.

**À ma binôme**

Merci pour ta collaboration, ta patience et ton esprit d'équipe tout au long de ce projet. Ce travail est le fruit de nos efforts conjoints.

Avec toute ma gratitude

**Mokhtar.T**

# *Dédicaces*

*C'est avec une sincérité profonde et un respect infini que je dédie ce modeste travail à :*

***Ma chère mère Zaineb***

*Aucun dédicace ne serait exprimer mon grand respect, et ma reconnaissance pour les sacrifices que tu as consentis pour mon éducation. j'implorent Allah le tout pouissant de t'accorder bonne santé et langue vie.*

***Mon père Habib, mon héros silencieux***

*Ton amour est ma force, ton regard ma boussole.*

*Merci d'être ce pilier que rien n'ébranle.*

***Mon frère Chafik***

*Ton soutien et ton amour fraternel m'ont donné force et encouragement*

***Mes jolies soeurs Linda et Soujoud, mes yeux***

*Votre amour éclaire ma vie comme deux étoiles inséparables.*

*Avec vous, chaque battement de cœur devient un refuge.*

***Mes chères amies***

***Nawres et Rakia***

*mes meilleures amies, mes sœurs de cœur depuis l'enfance,*
*Votre amitié est un trésor qui a grandi avec moi, que le temps n'a jamais fané.*
*Dans chaque souvenir précieux, il y a votre rire, votre présence, votre tendresse…*
*Merci d'être ces âmes fidèles, cette bénédiction que je chéris du plus profond de mon cœur.*

***Hayfa, Nourhene, Sabeh, Sarra, Sana, Malek  Raya Maryem Hadj Salem et Maryem ben Taher***

*Votre présence adoucit mes peines et amplifie mes joies.*

*Je vous porte dans mon cœur, là où l'amitié ne meurt jamais.*

***À toute ma famille,***

*Votre force et votre unité m'ont guidé. Ce travail est aussi le fruit de votre amour et de votre confiance en moi.*

**Rawen.A**

# Remerciement

# Sommaire


Remerciement ..... 3
Introduction Générale ..... 1
Chapitre 1 ..... **Erreur ! Signet non défini.**
Introduction ..... 3
I. Cadre du projet ..... 3
I.1. Présentation de l'organisme ..... 3
I.1.a. Agil Energy S.A ..... 3
I.1.b.Activités ..... 4
II. Étude de l'existant ..... 5
II.1. Description de l'Existant ..... 5
II.2. Critique de l'Existant ..... 5
II.2.1. Problématiques ..... 5
II.2.2. Solution Proposée ..... 6
Conclusion ..... 6
Chapitre 2 ..... **Erreur ! Signet non défini.**
Introduction ..... 8
I. L'architecture des microservices ..... 8
II. Monolithique vs Microservice ..... 8
III. Caractéristique de l'architecture microservice ..... 10
IV. Conception et architecture logiciel ..... 11
IV.1. Choix des modèles d'integration ..... 12
IV.2. Organisation des Composants ..... 13
IV.3. Schéma de l'architecture ..... 13
V. Cadre de développement ..... 14
Conclusion ..... 16
Chapitre 3 ..... **Erreur ! Signet non défini.**
Introduction ..... 18
I. Analyse des besoins ..... 18
I.1. Exigences fonctionnelles ..... 18
I.1.a. Microservices ..... 18
I.1.b. Interactions entre microservices ..... 18
I.2. Exigences non fonctionnelles ..... 19
I.3. Identification des acteurs ..... 20
II. Diagramme de cas d'utilisation global ..... 20
III. Diagramme de classe global ..... 22
IV. Environnement de travail ..... 23
IV. 1.environnement matériel ..... 23
IV. 1.environnement logiciel ..... 23
V. Product Backlog global ..... 26
Conclusion ..... 27
Chapitre 4 ..... 28
I.Sprint 1: Gestion des utilisateurs ..... 29
Introduction ..... 29
I.1 Backlog du Microservice ..... 29
I.2. Conception ..... 30
I.1.a. Diagramme de Cas d'Utilisation ..... 30
I.1.b. Diagramme de Classes ..... 32
I.1.c. Diagramme de Séquence ..... 32
I.1.d. Diagramme de deployement ..... 34

I.2. Architecture ..... 35
I.2.a. API Gateway : ..... 35
I.2.b. Firebase Authentication : ..... 36
I.2.c. Firebase Firestore : ..... 37
I.2.d. Connexion avec d'autres services : ..... 37
I.3. Implementation ..... 37
I.3.a. Tableau Scrum ..... 37
I.3.b. Interface ..... 39
Conclusion ..... 41
Chapitre 5 ..... 42
Introduction ..... 43
I. Sprint 1: Déclaration et gestion des problèmes ..... 43
I.1. Backlog du premier Sprint ..... 43
I.2. Conception ..... 44
I.2.a. Diagramme de Cas d'Utilisation ..... 44
I.2.b. Diagramme de classes ..... 45
I.2.c. Diagramme de Séquence ..... 46
I.2.d. Diagramme de déploiement ..... 48
I.3. Architecture ..... 49
I.3. a. API Gateway : ..... 49
I.3. b. Firebase Firestore : ..... 50
I.3. c. Connexion avec d'autres services : ..... 50
I.4. Implémentation ..... 51
I.4.a. Tableau Scrum ..... 51
I.4.b. Interface ..... 52
Conclusion ..... 53
II. Sprint 2 : Suivi et Coordination ..... 53
II.1. Backlog du deuxième Sprint ..... 53
II.2. Conception ..... 54
II.2.a. Diagramme de Cas d'Utilisation ..... 54
II.2.b. Diagramme de classes ..... 55
II.2.c. Diagramme de Séquence ..... 56
II.3. Architecture ..... 58
II.3.a. API Gateway : ..... 58
II.2.b. Firebase Firestore : ..... 59
II.3.c. Connexion avec Autres services ..... 59
II.4.Implémentation ..... 60
II.4.a. Tableau Scrum ..... 60
II.4.b.Interface ..... 61
Conclusion ..... 62
Chapitre ..... 63
I. Sprint 1 : Dashboard et Notifications ..... 64
Introduction ..... 64
I.1.Backlog du premier Sprint ..... 64
I.2. Conception ..... 64
I.2.a. Diagramme de Cas d'Utilisation ..... 64
I.2.a. Diagramme de Séquence ..... 66
I.3. Architecture ..... 67
I.3.a. API Gateway : ..... 67
I.3.b. Connexion avec d'autres services : ..... 68
I.4. Implémentation ..... 68
I.4.a Tableau Scrum ..... 68

I.4.b. Interface ..... 69
II. Sprint 2 : ChatBot AI ..... 71
Introduction ..... 71
I.1.Backlog de deuxième Sprint ..... 71
I.2. Conception ..... 71
I.2.a. Diagramme de Cas d'Utilisation ..... 71
I.2.b. Diagramme de classe ..... 71
I.3. Architecture ..... 72
I.3.a. API Gateway : ..... 72
I.3.b. Connexion avec d'autres services : ..... 72
I.4. Implémentation ..... 73
I.4.a Tableau Scrum ..... 73
I.4.b. Interface ..... 74
Conclusion Générale ..... 75

# Table des figures

Chapitre 2 : Étude Préalable ..... 7
Figure1 :Monolithique vs Microservices ..... 10
Figure 2 : shema de l'architecture ..... 14
Figure 3 : Méthode agile ..... 14
Figure 4 : méthode scrum ..... 15
Chapitre 3 : Mise en ouevre ..... 17
Figure 5: Diagramme de cas d'utilisation global ..... 21
Figure 6 :Diagramme de classe global ..... 23
Release 1: Gestion des utilisateurs ..... 28
Figure 7 : Diagramme de cas d'utilisation de gestion des utilisateurs ..... 30
Figure 8 : Diagramme de classes de gestion des utilisateurs ..... 32
Figure 10 : Diagramme de séquence d'ajout d'un nouveau chef ..... 33
Figure 11 : Diagramme de séquence de modification d'un utilisateur ..... 34
Figure 12 : Diagramme de déploiement de gestion des utilisateurs ..... 35
Figure 13 : Firebase Authentification ..... 36
Figure 14 : Base de données de gestion des utilisateurs ..... 37
Figure 15 : communication entre microservices ..... 37
Figure 17 : Tableau scrum pour le dernier jour ..... 38
Release 2: Gestion des Problémes ..... 42
Figure 24 : Diagramme de cas d'utilisation Sprint 1 ..... 44
Table 10 : Tableau des informations clés sur la gestion des problèmes ..... 45
Figure 25 : Diagramme de classes sprint 1 ..... 46
Figure 27 : Diagramme de sequence signaler un probleme ..... 48
Figure 28 : Diagramme du déploiement sprint 1 ..... 49
Figure 29 : Base de données du gestion des problemes ..... 50
Figure 32 : Interface de creation d'un problème prédéfini ..... 52
Figure 33 : Interface d'affectation d'un problème ..... 52
Figure 34 : Interface de consultation des problémes ..... 53
Figure 35 : Diagramme de cas d'utilisation du sprint 2 ..... 55
Figure 36 : Diagramme Gestion des problèmes ..... 56
Figure 37 : Diagramme de consultation d'un problème ..... 57
Figure 38 : Diagramme d'acceptation d'un problème par le technicien ..... 58
Figure 40 : base de données sprint2 ..... 59
Figure 41 : base de données sprint2 ..... 59

Figure 42 : Tableau scrum pour le premier jour ..... 60
Figure 43 : Tableau scrum pour le dernier jour ..... 60
Figure 44 : interface des problèmes signaler ..... 61
Figure 45 : interface des problèmes assignées au technicien ..... 62
Release 3: Gestion des Notifications ..... 63
Figure 46: Diagramme de cas d'utilisation ..... 65
Figure 47: Diagramme de séquence d'accédation au dashboard ..... 66
Figure 48: Diagramme de séquence de consultation etat microservice ..... 67
Figure 49: Interface Dashboard ..... 69
Figure 50: Interface de tous les problème ..... 70
Figure 51: Interface Localisation ..... 70
Figure 52 : Diagramme de Cas d'Utilisation chatbot ..... 71
Figure 53 : Diagramme de classe chatbot ..... 72
Figure 54 : interface du chatbot ..... 74

# Table des tableaux

Chapitre 2 : Étude Préalable ..... 7
Table 1 : tableau de comparaison entre monolithique et microservice ..... 9
Table 2 :Tableau des modèles de conception ..... 13
Chapitre 3 : Mise en ouevre ..... 17
Table 3: Tableau des informations clés du système ..... 22
Table 4 : environnement matériel ..... 23
Table 5 : Product Backlog global ..... 27
Release 1: Gestion des utilisateurs ..... 28
Table 6: Backlog du microservice Gestion utilisateur ..... 29
Table 7 : Tableau des informations clés sur la gestion des utilisateurs ..... 31
Table 8: Routes et Méthodes de l'API gateway ..... 36
Release 2: Gestion des Problémes ..... 42
Table 9: Backlog du premier sprint ..... 43
Table 10 : Tableau des informations clés sur la gestion des problèmes ..... 45
Table 11 : Routes et Méthodes du scrum 1 ..... 50
Table 12: Backlog du deuxième sprint ..... 54
Table 13 : Routes et Méthodes de l’API Gateway du sprint 2 ..... 59
Release 3: Gestion des Notifications ..... 63
Table 14: Backlog du microservice Gestion des notification ..... 64
Table 15:Tableau récapitulatif  Gestion des notification ..... 66
Table 16:Routes et Méthodes de l'API Gateway ..... 68
Table 17: Tableau scrum du premier jour ..... 68
Table 18: Tableau scrum du dernier jour ..... 69
Table 19 : Api gateway du chatbot ..... 72
Table 20 : tableau scrum du premier jour chatbot ..... 73
Table 21 : tableau scrum du dernier jour chatbot ..... 73

# Liste des abréviations

- **ADM** – Administrateur
- **AI** – Artificial Intelligence (Intelligence artificielle)
- **API** – Application Programming Interface
- **API GW** – API Gateway
- **ASGI** – Asynchronous Server Gateway Interface
- **AST** – Assistant
- **Auth** – Authentication (Authentification)
- **CF** – Cloud Functions (Fonctions Cloud Firebase)
- **CHF** – Chef de station
- **CLI** – Command Line Interface
- **CRUD** – Create, Read, Update, Delete
- **CSV** – Comma-Separated Values
- **CSS** – Cascading Style Sheets
- **DB** – Database
- **DOM** – Document Object Model
- **ETL** – Extract, Transform, Load
- **FB** – Firebase
- **FCM** – Firebase Cloud Messaging
- **FS** – Firestore (Cloud Firestore)
- **HTML** – HyperText Markup Language
- **HTTP** – HyperText Transfer Protocol
- **ID** – Identifiant
- **IDE** – Integrated Development Environment
- **JSON** – JavaScript Object Notation

- **JS** – JavaScript
- **JSX** – JavaScript XML (extension React)
- **JWT** – JSON Web Token
- **KPI** – Key Performance Indicator
- **ML** – Machine Learning
- **MS** – Microservice
- **NPM** – Node Package Manager
- **ORM** – Object-Relational Mapping
- **Pydantic** – Python Data Validation Library
- **RTDB** – Realtime Database (Firebase)
- **REST** – Representational State Transfer
- **SDK** – Software Development Kit
- **SPA** – Single Page Application
- **TEC** – Technicien
- **TS** – TypeScript
- **UI** – User Interface
- **UX** – User Experience

# Introduction Générale

Dans un environnement technologique en constante évolution, l'optimisation et l'automatisation des processus métiers deviennent des enjeux majeurs pour les entreprises, en particulier dans le secteur industriel et énergétique. La gestion efficace des incidents techniques et des interventions est cruciale pour assurer la continuité des opérations et améliorer la productivité.

Face à ces défis, la Société Nationale de distribution de Pétrole cherche à moderniser son système de gestion des incidents rencontrés dans ses stations-service. Actuellement, le suivi des problèmes et leur résolution sont traités de manière non centralisée, ce qui entraîne une perte d'efficacité et des retards dans les interventions techniques. Une solution digitale adaptée est donc nécessaire pour fluidifier la communication entre les parties prenantes et assurer un suivi rigoureux des opérations de maintenance.

Ce projet vise ainsi à développer une application de gestion des incidents en s'appuyant sur une architecture microservices. En effet, il est organisé en 6 chapitres

Le premier chapitre, intitulé « Contexte du projet », présente l'environnement général ainsi que la motivation à l'origine de la réalisation de ce projet de stage.
Le deuxième chapitre, appelé « Étude préliminaire », se concentre sur les technologies, outils et choix architecturaux utilisés tout au long du projet.
Le troisième chapitre, intitulé « Mise en ouevre », décrit la structure générale de l'application, la méthodologie de travail adoptée ainsi que la répartition des tâches.
Le quatrième chapitre, intitulé « Release 1 : Gestion des utilisateurs », est dédié au développement du premier microservice, qui gère l'enregistrement des utilisateurs, l'authentification et la gestion des rôles.
Le cinquième chapitre, appelé « Release 2 : Gestion des problèmes », porte sur le second microservice, qui permet aux chefs de signaler et suivre les problèmes techniques dans les stations.
Le sixième chapitre, intitulé « Release 3 : Gestion des notifications », présente le troisième microservice, qui assure la communication avec les utilisateurs via des notifications et génère des rapports.

# Chapitre 1 : Contexte du projet

**Plan**

**Introduction**

**I.Cadre du projet**

**II.Étude de l'existant**

**Conclusion**

# Introduction

Avant de mettre en place une nouvelle solution, il est essentiel d'analyser le système existant afin d'identifier ses limites et ses points d'amélioration. Cette étude permettra de justifier la nécessité d'une nouvelle approche plus efficace et adaptée aux besoins actuels.

# I. Cadre du projet

## I.1. Présentation de l'organisme

### I.1.a. Agil Energy S.A

La Société Nationale de Distribution des Pétroles Agil Energy S.A. est une entreprise publique ayant pour mission la commercialisation des produits pétroliers et de leurs dérivés sous le label Agil Energy. Elle fait partie des grandes entreprises publiques tunisiennes qui, par leur dynamisme et la diversité de leurs activités, soutiennent l'économie nationale et lui assurent une croissance continue. Avec un chiffre d'affaires HT de 1 845 millions de dinars en 2020, Agil Energy S.A. joue un rôle d'avant-garde sur la voie du progrès et de l'excellence dans laquelle s'est engagée la Tunisie de l'ère nouvelle.

En développant ses activités, Agil Energy S.A. a fini par occuper la première place parmi les entreprises du secteur, tant par le volume de ses ventes que par l'importance de son chiffre d'affaires et le savoir faire de ses ressources humaines et s'emploie constamment à consolider cette position en offrant à ses clients la meilleure qualité de produit et de service.

Figure: Logo Agil Energy

### I.1.b.Activités

- **Réseau Agil:**

Agil Energy S.A. dispose de plus de 227 stations-service à travers la Tunisie et est présente dans 54 ports pour soutenir la pêche, le tourisme et la plaisance.

- **Aviation:**

AGILAIR, leader du ravitaillement aérien en Tunisie, dispose de la plus grande capacité de stockage de kérosène et applique les normes internationales pour assurer un service de qualité aux compagnies aériennes.

- **Carburants:**

Leader du marché des carburants, Agil Energy assure la qualité de ses produits grâce à des laboratoires mobiles et une large capacité de stockage. Elle propose aussi des solutions de ravitaillement flexibles via une flotte de camions et des cartes-bons utilisables dans son réseau de stations-service.

- **Gaz:**

Leader du marché du GPL en Tunisie, Agil Energy commercialise du GPL domestique, industriel et carburant. Elle dispose de trois centres d'emplissage, d'une flotte de camions spécialisés et d'un accord avec NAFTAL pour approvisionner le Nord-Ouest.

- **Lubrifiant:**

Agil Lubrifiants couvre une large part du marché tunisien grâce à une gamme de produits conformes aux normes internationales, une logistique efficace, une assistance client de proximité et un contrôle qualité rigoureux.

- **Marine:**

Agil Energy S.A. approvisionne tous types de navires dans les ports tunisiens, offrant un service fiable 7j/7 grâce à son expertise, sa flotte et ses dépôts de stockage.

- **Produits spéciaux:**

  Agil Energy S.A. excelle dans la commercialisation du white spirit, un solvant incolore et stable utilisé pour les peintures, vernis et nettoyages, offrant un produit conforme aux normes et adapté aux besoins industriels.

# II. Étude de l'existant

## II.1. Description de l'Existant

Actuellement, la gestion des incidents techniques dans les stations-service repose sur un système manuel et non centralisé. Les chefs de station signalent les problèmes par téléphone ou via des documents papier, ce qui entraîne des retards dans la transmission des informations et complique le suivi des interventions.

Le processus de gestion des incidents comprend plusieurs étapes :

- Identification du problème par le chef de station.
- Identification du problème par téléphone ou par e-mail à l'assistant.
- Transmission de la demande à un technicien compétent.
- Intervention du technicien et résolution du problème.
- Validation de la résolution par le chef de station.

Ce fonctionnement présente plusieurs limites qui freinent l'efficacité du traitement des incidents

## II.2. Critique de l'Existant

### II.2.1. Problématiques

L'analyse du système actuel met en évidence plusieurs problèmes majeurs :

- Manque de traçabilité : Aucune base de données centralisée ne permet d'enregistrer l'historique des incidents et des interventions.
- *Communication inefficace :* L'échange d'informations par téléphone ou e-mail entraîne des pertes de données et des retards.
- Difficulté d'attribution des tâches *:* L'assignation des techniciens repose sur des décisions humaines, ce qui peut conduire à des erreurs ou des déséquilibres de charge de travail.
- *Absence de notifications :* Les techniciens ne sont pas informés en temps réel des nouvelles interventions à effectuer.
- *Suivi limité :* Il est difficile pour les chefs de station et les administrateurs d'avoir une vue d'ensemble sur l'état des problèmes en cours et les délais de résolution.

### II.2.2. Solution Proposée

Afin de remédier aux limitations du système actuel, nous proposons la mise en place d'une **application web et mobile** basée sur une **architecture microservices**.

Cette solution permettra :

- Une déclaration centralisée des incidents via une interface intuitive.
- Une attribution automatique des techniciens selon le type de problème.
- Un suivi en temps réel de l'état des incidents et des interventions.
- L'envoi de notifications automatiques aux utilisateurs concernés.
- Une gestion centralisée des données pour assurer une traçabilité et une analyse efficace des incidents.

Cette solution garantira une meilleure organisation, une réduction des délais d'intervention et une amélioration de la communication entre les différents acteurs du processus.

## Conclusion

Cette étude de l'existant a mis en évidence les limites du système actuel et justifie la nécessité d'une solution plus moderne et efficace. La nouvelle application proposée répondra aux problématiques identifiées en offrant une gestion optimisée et automatisée des incidents techniques.

# Chapitre 2 : Étude Préalable

**Plan**

**Introduction**

- **I. L’architecture des microservices**
- **II. Monolithique vs Microservices**
- **III. Caractéristiques de l'architecture microservices**
- **IV. Conception et Architecture Logicielle**
- **V. Cadre de développement**

**Conclusion**

# Introduction

Dans le cadre de la modernisation de notre architecture logicielle, cette étude préalable explore les microservices comme solution pour améliorer la scalabilité, la maintenabilité et la résilience de notre système, en décomposant l'application en modules autonomes et spécialisés, offrent une alternative agile aux architectures monolithiques traditionnelles.

# I. L'architecture des microservices

Le terme « Microservice » est apparu ces dernières années pour décrire un style architectural très particulier. Cette approche consiste à développer une application unique sous la forme d'un ensemble de services petits, isolés, autonomes et déployés indépendamment. Ces services peuvent communiquer entre eux pour fournir les fonctionnalités nécessaires.

Les microservices sont, la plupart du temps, mis en œuvre et gérés par de petites équipes disposant d'une autonomie suffisante. Chacune d'elles peut modifier l'implémentation d'un microservice, ajouter ou supprimer des fonctionnalités de ce service avec un impact minimal sur les autres microservices.

Ce style d'architecture présente plusieurs avantages, tels que l'hétérogénéité technologique, la résilience face aux pannes, l'évolutivité adaptée, la facilité de déploiement, l'alignement organisationnel et la réutilisabilité.

# II. Monolithique vs Microservice

L'architecture monolithique regroupe toutes les fonctionnalités d'une application dans un seul bloc, ce qui la rend simple à développer au début mais difficile à maintenir et faire évoluer. À l'inverse, l'architecture microservices divise l'application en plusieurs services indépendants, ce qui permet une meilleure scalabilité, une maintenance plus facile, et le déploiement séparé de chaque service.

| Aspect | Architecture Monolithique | Architecture Microservices |
|---|---|---|
| **Structure** | Une seule base de code avec plusieurs modules fonctionnels ou techniques. | Plusieurs petits services indépendants, chacun dédié à une tâche spécifique. |
| **Déploiement** | Un seul fichier exécutable ou déployable (tout est compilé ensemble). | Chaque service peut être déployé indépendamment. |
| **Base de code** | Un seul système de build pour l'ensemble de l'application. | Plusieurs dépôts et builds distincts pour chaque service. |
| **Fonctionnement** | L'application fonctionne comme un seul bloc (tous les modules sont liés). | Chaque service fonctionne de manière autonome avec des dépendances minimales (faible couplage). |
| **Impact des modifications** | Toute mise à jour nécessite une intervention sur l'ensemble de l'application. | Les modifications d'un service n'affectent pas les autres (principe de responsabilité unique). |
| **Disponibilité** | Une panne affecte toute l'application. | Une défaillance d'un service n'affecte pas nécessairement les autres. |
| **Scalabilité** | Scalabilité verticale (ajout de ressources au serveur). | Scalabilité horizontale (chaque service peut être scalé indépendamment). |
| **Exemples d'entreprises** | Applications traditionnelles (ERP, CRM classiques). | Netflix, Amazon, Google, Spotify (adoptent les microservices pour flexibilité et performance). |
| **Avantages** | - Développement et déploiement simples (au début). | - Flexibilité, agilité, scalabilité.<br>- Réutilisabilité des services.<br>- Résilience. |
| **Inconvénients** | - Difficulté de maintenance à grande échelle.<br>- Déploiements lourds. | - Complexité accrue (gestion des services distribués).<br>- Nécessite une bonne orchestration. |

*Table 1 : tableau de comparaison entre monolithique et microservice*

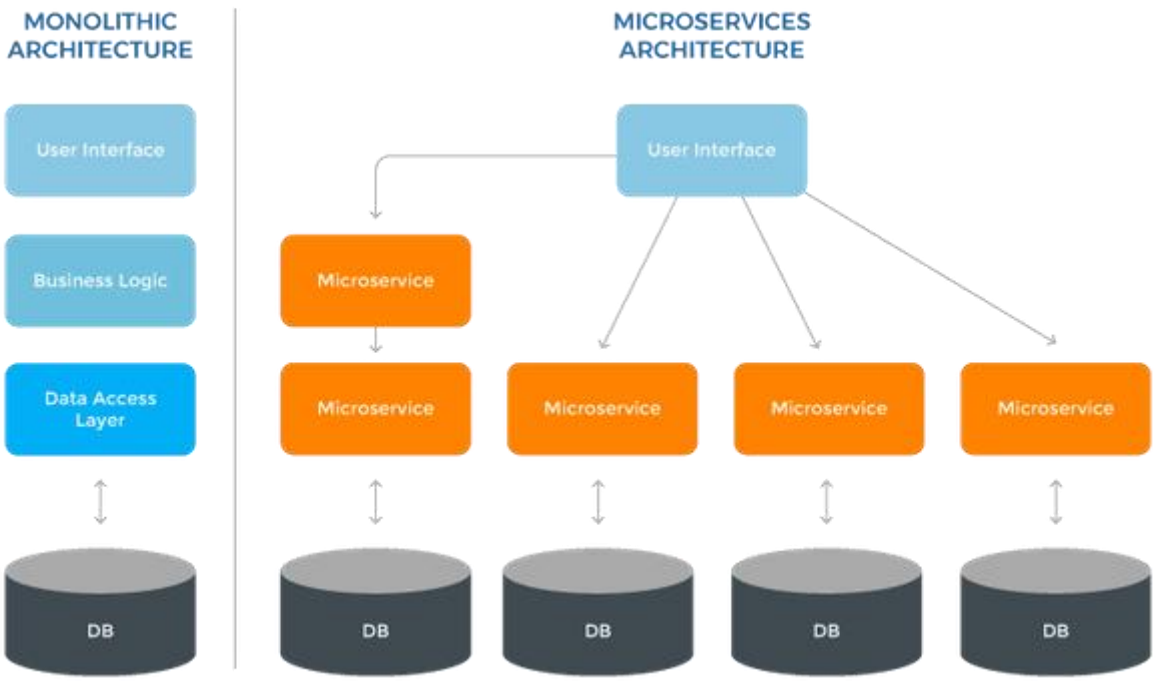



*Figure1 :Monolithique vs Microservices*

## III. Caractéristique de l'architecture microservice

- **Résilience et isolement des pannes**

Dans une architecture monolithique, une erreur va affecter toute l'application. Les microservices, en revanche, sont indépendants. Un échec ne va pas affecter les autres parties de l'application.

- **Agilité**

Le fait de scinder les applications en différents éléments plus réduits permet d'accélérer le développement. Les équipes peuvent ainsi créer, tester et déployer des logiciels plus rapidement.

- **Diversité technologique**

Les microservices permettent aux développeurs de choisir les outils et les technologies les mieux adaptés à leurs tâches. L'efficacité et la productivité en sont ainsi augmentées.

- **Maintenance améliorée**

La possibilité de tester des composants individuels permet de détecter et de corriger les bugs plus facilement, sans devoir mettre l'ensemble de l'application hors connexion.

# IV. Conception et architecture logiciel

Une architecture de microservices apporte certains défis. Les modèles de conception peuvent aider à atténuer ces défis. Il est important de savoir qu'une architecture de microservices peut être utilisée avec différents modèles de conception, de manière à mettre en avant deux modèles ou plus dans la même application. Tout cela dépend des besoins et des résultats souhaités. On peut citer quelques exemples :

❖ **Modèles de Décomposition :**

Décomposer l'application en services indépendants:

- Décomposition par capacité métier : Séparation basée sur les fonctions métier (ex: service de paiement, service d'authentification).
- Décomposition par sous-domaine : Alignement avec les sous-domaines du domaine métier (ex: gestion des commandes, gestion des stocks).

❖ **Modèles d'Intégration :**

Gérer la communication entre service

- API Gateway : Point d'entrée unique pour les clients, gère le routage, l'authentification, et l'agrégation de réponses
- Aggregator Pattern: Combine les données de plusieurs services avant de les renvoyer au client (ex: un tableau de bord utilisateur)

❖ **Modèles de Base de Données :**

Gérer le stockage et la cohérence des données :

- Database per Service : Chaque service possède sa propre base de données (isolation complète).
- Saga Pattern : Gère les transactions distribuées via une séquence de compensations (ex: annulation d'une commande si un service échoue).

❖ **Modèles d'Observabilité :**

Surveiller et diagnostiquer le système :

- Log Aggregation : Centralise les logs de tous les services pour un débogage simplifié (ex: outils comme ELK Stack).
- Performance Metrics : Collecte et analyse des métriques en temps réel (ex: latence, taux d'erreur).

**Modèles de Résilience :**

Gérer les pannes et les erreurs :

- Circuit Break : Bloque les appels vers un service défaillant pour éviter les cascades d'erreurs.
- Retrempe Pattern : Réessaie automatiquement les requêtes en cas d'échec temporaire.

## IV.1. Choix des modèles d'integration

| Modèle | Description | Fonctionnalités Clés |
|---|---|---|
| 1. API Gateway | Point d'entrée unique pour les requêtes vers les microservices. Gère la sécurité, le routage et l'agrégation des réponses. | - Sécurité des API (authentification, autorisation).<br>- Connectivité multiple-protocole (REST, WebSocket, etc.).<br>- Virtualisation des API (masquage des services back-end).<br>- Évolutivité/HA (gestion de la charge).<br>- Gestion simplifiée (versioning, monitoring). |
| 2. Firestore Functions | Fonctions serverless exécutées dans Google Cloud, déclenchées par des événements Firestore (sans gestion de serveur). | - Exécution serverless (pas d'infrastructure à gérer).<br>- Intégration Firebase (Firestore, Auth, Messaging).<br>- Automatisation (validation, notifications, calculs).<br>- Sécurité (via Firebase Auth et Security Rules).<br>- Scalabilité automatique. |
| 3. Firestore Triggers | Déclencheurs automatisés qui exécutent du code en réponse à des modifications | - Temps réel (réaction immédiate aux changements).<br>- Types d'événements (onCreate, onUpdate, onDelete, onWrit |

| Modèle | Description | Fonctionnalités Clés |
|---|---|---|
| | de données dans Firestore (ajout/modification/suppression). | e).<br>- Interopérabilité (APIs tierces, notifications).<br>- Scalabilité/Fiabilité |

*Table 2 :Tableau des modèles de conception*

## IV.2. Organisation des Composants

L'architecture articule quatre éléments centraux :

- API Gateway :
  - Joue le rôle d'un point d'entrée unique pour toutes les requêtes.
  - Gère l'authentification et redirige les requêtes vers les microservices appropriés.
- Microservices :
  - Décomposés par domaine métier (ex: gestion des techniciens, notifications).
- Triggers :
  - Automatisation des processus métier (ex: validation des données, réactions en temps réel) ,
  - Réactions aux changements dans Firestore (ex. : assignation automatique d'un technicien après la création d'un problème)
- Firebase Firestore
  - Base de données NoSQL pour stocker toutes les informations (Utilisateurs, problèmes, notifications).
  - Utilise des collections et documents pour organiser les données.

## IV.3. Schéma de l'architecture

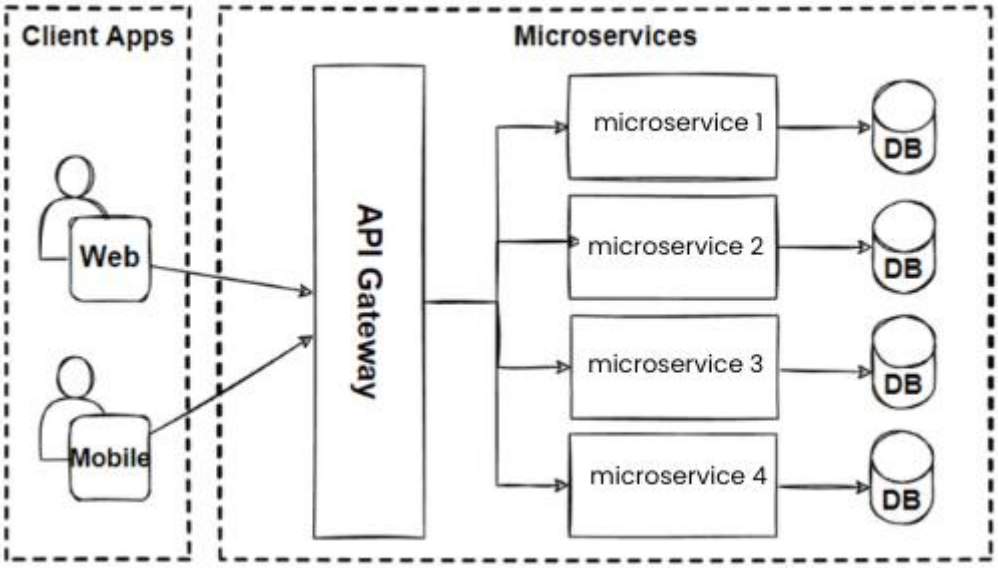



Figure 2 : shema de l'architecture

# V. Cadre de développement

Pour réussir à produire une solution informatique performante, il est important d'adopter une méthodologie de travail bien définie. Bien qu'il existe plusieurs méthodologies à suivre, nous avons adopté les méthodes agiles les plus utilisées et les plus adaptées à la réalisation de notre application.

## 1. Méthode Agile

Pour atteindre nos objectifs, nous avons adopté la méthode agile, car elle repose sur une approche itérative et incrémentale.

- **Itérative** : consiste à diviser le projet en tâches.
- **Incrémentale** : chaque itération doit être développée et validée par le client avant de passer à l'itération suivante.

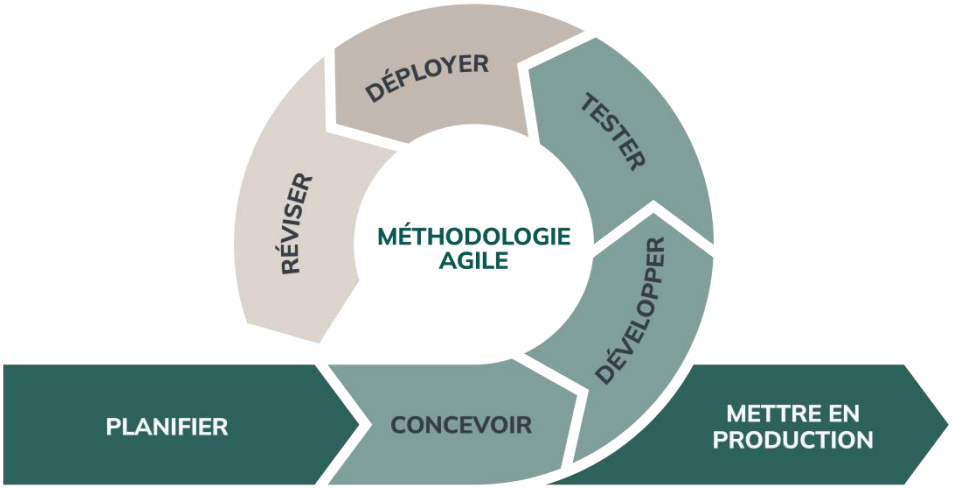



*Figure 3 : Méthode agile*

## 2.Méthode Scrum

Le principe de la méthodologie SCRUM est de développer un logiciel de manière incrémentale tout en maintenant une liste totalement transparente des demandes de modifications ou de corrections à implémenter. Grâce à des livraisons très fréquentes, le client reçoit un logiciel fonctionnel à chaque itération. Plus le projet avance, plus le logiciel se complète et intègre de nouvelles fonctionnalités.

### a) Rôles dans Scrum

La méthodologie **SCRUM** implique trois rôles principaux :

- **Product Owner** : Dans la plupart des projets, le Product Owner est le responsable de l'équipe projet côté client. Il est chargé de définir et de prioriser la liste des fonctionnalités du produit, ainsi que de choisir la date et le contenu de chaque sprint en fonction des estimations (charges) communiquées par l'équipe.
- **Scrum Master** : Véritable facilitateur du projet, il veille à ce que chacun puisse travailler dans les meilleures conditions en supprimant les obstacles et en protégeant l'équipe des perturbations externes.
- **Équipe** : L'équipe s'organise elle-même et reste inchangée tout au long d'un sprint. Elle doit tout mettre en œuvre pour livrer le produit.

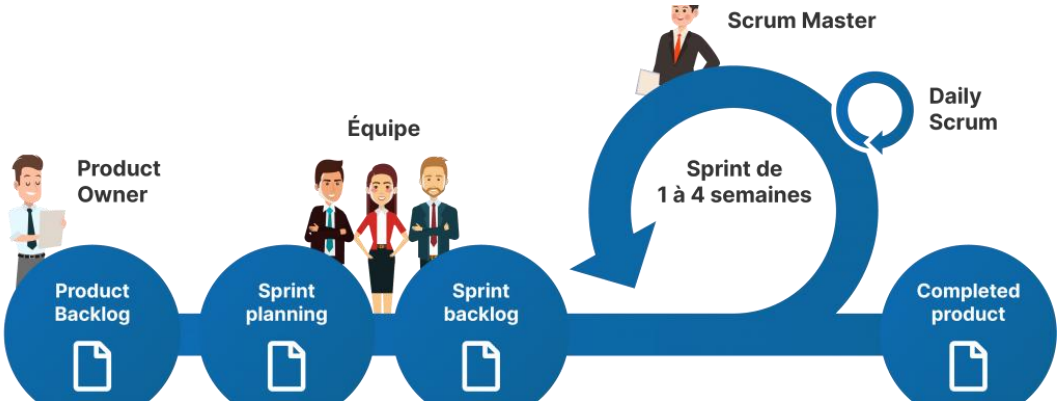



*Figure 4 : méthode scrum*

### b) Le Sprint Agile

Le sprint agile est le cœur de la méthode Scrum. Cette qualification lui convient parfaitement, car tous les développements incrémentaux menant progressivement au produit final du projet sont réalisés au sein des sprints.

Un périmètre de développement est défini au début d'un sprint et doit être entièrement accompli à sa fin. Chaque sprint doit apporter une fonctionnalité supplémentaire à l'application en cours de développement et être livré une fois terminé. Le Product Owner est responsable de la

définition des sprints et de l'organisation du product backlog afin de faciliter la construction du produit.

**c) Product Backlog**

Le backlog a pour objectif de rassembler tous les besoins du client que l'équipe projet doit réaliser. Il contient la liste des fonctionnalités nécessaires à la construction du produit, ainsi que tous les éléments nécessitant l'intervention de l'équipe projet. Tous les éléments du backlog sont classés par ordre de priorité, indiquant l'ordre de leur réalisation.

La création du backlog commence par la formalisation et la définition des objectifs du produit, des utilisateurs ciblés et des différents acteurs du projet. Une liste d'exigences, pouvant être fonctionnelles ou non fonctionnelles, est ensuite établie. Chaque exigence est analysée par l'équipe de développement, et son coût de réalisation est évalué lors d'une session de planning poker.

Une liste de fonctionnalités est ensuite extraite de cette liste d'exigences et classée par ordre de priorité d'exécution.

Cette planification est réalisée en fonction de la valeur ajoutée que chaque fonctionnalité apporte au produit. C'est cette liste ordonnée des fonctionnalités qui constitue le product backlog.

# Conclusion

Dans ce chapitre, nous avons pu détailler l'architecture des microservices, clarifier les concepts de base qui facilitent la compréhension et la réalisation de notre projet. Nous allons maintenant passer à l'étape suivante, qui consiste en l'identification et l'analyse des exigences.

# Chapitre 3 : Mise en ouevre

Plan

Introduction

I. Analyse des besoins

II. Diagramme de cas d'utilisation global

III. Diagramme de classe global

IV. Environnement de travail

V. Product Backlog global

Conclusion

# Introduction

# I. Analyse des besoins

## I.1. Exigences fonctionnelles

Les exigences fonctionnelles décrivent les fonctionnalités que le système doit offrir pour répondre aux besoins des utilisateurs. Dans notre projet, ces exigences sont principalement centrées autour des microservices et de leurs interactions.

### I.1.a. Microservices

Les microservices sont au cœur de notre architecture, et chaque service est conçu pour répondre à une fonctionnalité spécifique.

Voici les principales exigences fonctionnelles liées aux microservices :

- **Service Utilisateur (User Service)** :
  - Gestion des utilisateurs (création, modification, suppression).
  - Authentification et autorisation des utilisateurs.
  - Gestion des rôles et des permissions.
- **Service Problème (Problem Service)** :
  - Création, suivi et mise à jour, documentation des problèmes signalés par les utilisateurs
  - Attribution automatique des problèmes aux techniciens disponibles.
  - Historique des problèmes résolus.
- **Service Notification (Notification Service)** :
  - Envoi de notifications en temps réel aux utilisateurs (par exemple, confirmation de création de problème, mise à jour de statut).
  - Gestion des préférences de notification (e-mail, notifications push).

### I.1.b. Interactions entre microservices

- **Communication via API** : Les microservices communiquent entre eux via des API REST ou GraphQL pour échanger des données et coordonner les actions.
- **Gestion des événements** : Les microservices utilisent un système de messagerie asynchrone (comme Kafka ou RabbitMQ) pour gérer les événements tels que la création d'un problème ou l'attribution d'un technicien.

## I.2. Exigences non fonctionnelles

Les exigences non fonctionnelles décrivent les caractéristiques du système qui ne sont pas directement liées à des fonctionnalités spécifiques, mais qui sont essentielles pour garantir la performance, la sécurité, la scalabilité et la maintenabilité du système.

- **Performance**

  Temps de réponse : Le système doit répondre aux requêtes des utilisateurs en moins de 2 secondes.

  Scalabilité : Le système doit être capable de gérer une augmentation du nombre d'utilisateurs et de requêtes sans dégradation des performances.

- **Sécurité**

  Authentification et autorisation : Le système doit garantir que seuls les utilisateurs authentifiés et autorisés peuvent accéder aux fonctionnalités et aux données.

  Chiffrement des données : Les données sensibles (comme les mots de passe et les informations personnelles) doivent être chiffrées lors du stockage et de la transmission.

- **Disponibilité**

  Haute disponibilité : Le système doit être disponible 99,9% du temps, avec des mécanismes de redondance et de reprise après sinistre en place.

  Résilience : Le système doit être capable de gérer les pannes de manière transparente, sans impact sur les utilisateurs finaux.

- **Maintenabilité**

Modularité : Le système doit être conçu de manière modulaire pour faciliter les mises à jour et les modifications.

Documentation : Une documentation complète doit être fournie pour chaque microservice, y compris les API, les schémas de base de données.

### I.3. Identification des acteurs

Les acteurs sont les entités (personnes, systèmes ) qui interagissent avec le système. Dans notre projet, les acteurs principaux sont :

- Chef : Les utilisateurs qui signalent des problèmes et reçoivent des notifications.
- Assistant : Les utilisateurs qui supervisent les problèmes signalés, coordonnent l'attribution aux techniciens et assurent le suivi des interventions.
- Techniciens : Les utilisateurs responsables de la résolution des problèmes.
- Administrateurs : Les utilisateurs qui gèrent les rôles, les permissions et les configurations du système.

## II. Diagramme de cas d'utilisation global

Ce diagramme montre comment les acteurs du système (Administrateur, Chef, Assistant, Technicien) interagissent avec les différents modules. L'administrateur gère les utilisateurs, les rôles et les problèmes prédéfinis. Le chef signale des problèmes, l'assistant consulte ses chefs, et le technicien résout les problèmes. Tous passent par l'authentification et peuvent recevoir des notifications.

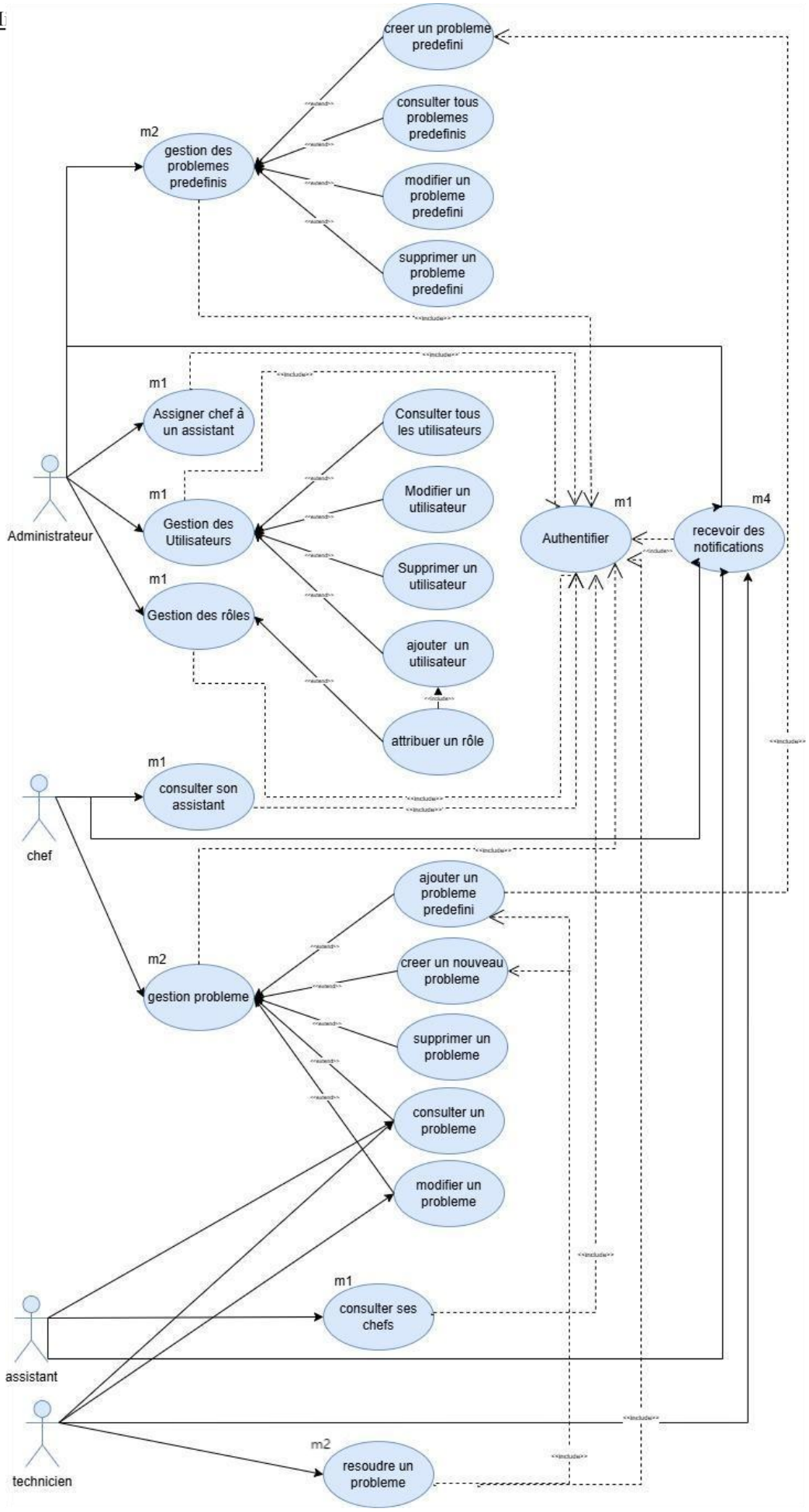



*Figure 5: Diagramme de cas d'utilisation global*

| | |
|---|---|
| **Résumé** | Ce système gère les problèmes (préexistants ou nouveaux), la gestion des utilisateurs et des rôles, ainsi que l'assignation et la résolution des problèmes par différents rôles (Administrateur, Chef, Assistant, Technicien). Il inclut aussi l'authentification et les notifications. |
| **Acteurs** | - Administrateur<br>- Chef<br>- Assistant<br>- Technicien |
| **Préconditions** | - L'utilisateur doit être authentifié.<br>- Les rôles doivent être attribués au préalable.<br>- La base de données doit contenir des utilisateurs, des rôles et éventuellement des problèmes prédéfinis. |
| **Postconditions** | - Problème créé, modifié ou résolu.<br>- Utilisateur ou rôle ajouté, modifié ou supprimé.<br>- Notifications reçues en temps réel.<br>- Assignation correcte des chefs à assistants. |
| **Scénario nominal** | 1. Un administrateur s'authentifie.<br>2. Il crée des utilisateurs et leur attribue des rôles.<br>3. Il ajoute des problèmes prédéfinis.<br>4. Un chef déclare un problème (nouveau ou prédéfini).<br>5. L'assistant consulte ses chefs, puis assigne le problème à un technicien.<br>6. Le technicien consulte ses problèmes assignés et les résout.<br>7. Le système envoie des notifications aux parties concernées. |
| **Scénarios alternatifs** | - Un utilisateur tente d'accéder à une fonctionnalité sans rôle ou permission suffisante → accès refusé.<br>- Un problème est supprimé alors qu'il est en cours de traitement → action bloquée ou message d'erreur.<br>- Échec d'authentification → message d'erreur.<br>- Tentative de création d'un utilisateur existant → message d'erreur. |
| **Exigences non fonctionnelles (NFN)** | - Sécurité (authentification obligatoire).<br>- Accessibilité multiplateforme (web, mobile).<br>- Notifications en temps réel.<br>- Interface intuitive pour chaque rôle.<br>- Historique des actions (traçabilité).<br>- Disponibilité élevée du système. |

*Table 3: Tableau des informations clés du système*

# III. Diagramme de classe global

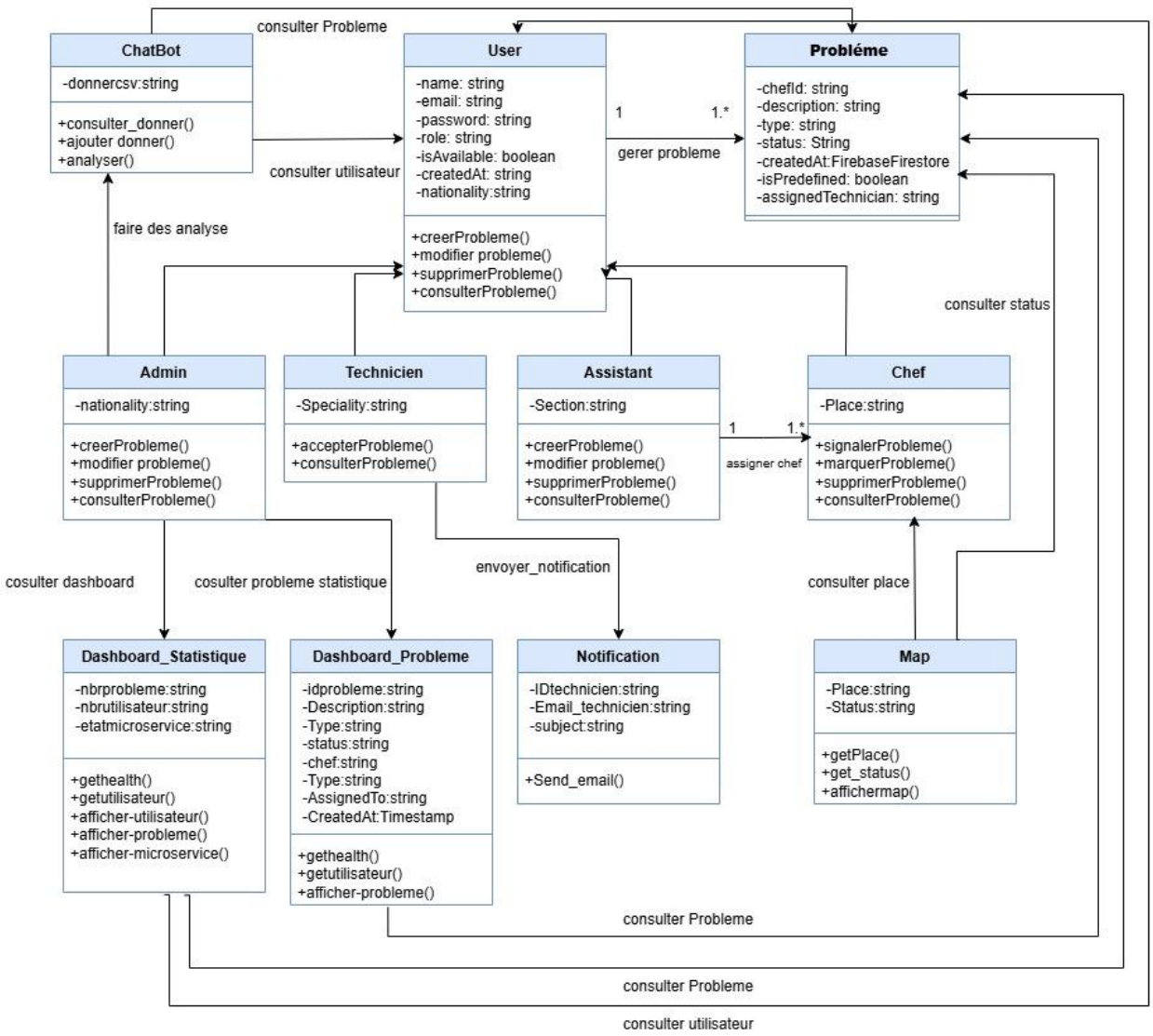



*Figure 6 :Diagramme de classe global*

# IV. Environnement de travail

**Dans cette section, nous présenterons brièvement l'environnement matériel (hardware) et logiciel (software) dans lequel l'application a été développée**

## IV. 1.environnement matériel

| Composant | Spécifications |
|---|---|
| **Modèle** | Dell G15 |
| **Processeur (CPU)** | Intel Core i5-13xxxHX (13e génération, série HX) |
| **Cœurs / Threads** | 10 cœurs , avec hyper-threading |
| **Mémoire Vive (RAM)** | 32 Go DDR5 |
| **Carte Graphique (GPU)** | NVIDIA GeForce RTX 3050 |
| **Stockage** | SSD NVMe 512 Go |

*Table 4 : environnement matériel*

## IV. 1.environnement logiciel

**React.js** est une bibliothèque JavaScript développée par Meta (Facebook) permettant de créer des interfaces utilisateur dynamiques et réactives. Elle est idéale pour construire des applications web modernes grâce à son système de composants réutilisables et sa gestion efficace de l'état.

**Vite** est un outil de build moderne extrêmement rapide conçu pour le développement frontend. Il permet de lancer des applications React ou Vue.js en quelques millisecondes grâce à un serveur de développement rapide basé sur ES Modules.

**Python** est un langage de programmation polyvalent, simple à apprendre et largement utilisé dans le développement web, la science des données, l'intelligence artificielle et l'automatisation. Sa syntaxe claire et sa vaste collection de bibliothèques en font un choix idéal pour les projets rapides et complexes.

**Postman** est un outil populaire utilisé par les développeurs pour tester, documenter et automatiser les requêtes API. Il permet d'envoyer facilement des requêtes HTTP (GET, POST, PUT, DELETE) vers un serveur, d'analyser les réponses, et de simuler l'interaction avec une API sans avoir besoin de créer une interface utilisateur.

**Ollama** est un environnement local qui permet de faire tourner des modèles de langage (comme LLaMA 3) directement sur une machine, sans connexion à un service cloud. Il est utilisé pour rendre ton chatbot IA autonome et fonctionnel hors ligne

**JavaScript** est un langage de programmation essentiel du web, permettant d'ajouter de l'interactivité et du dynamisme aux pages web. Il s'exécute principalement côté client dans le navigateur, mais peut aussi être utilisé côté serveur grâce à des environnements comme **Node.js**.

**HTML** est le langage de balisage standard utilisé pour structurer le contenu d'une page web. Il définit les éléments de base tels que les titres, paragraphes, liens, images, formulaires

**CSS** est un langage de style utilisé pour décrire l'apparence et la mise en forme des pages HTML. Il permet de contrôler les couleurs, les polices, les marges, les animations et la disposition des éléments.

**FastAPI** est un framework web moderne et rapide pour créer des APIs REST en Python. Il est utilisé dans ton microservice Notifications & Rapports, notamment pour gérer les envois de notifications et générer des rapports grâce à sa rapidité et sa compatibilité avec les annotations type de Python.

**Flutter** est un framework open-source développé par Google pour créer des applications mobiles multiplateformes. Il est utilisé dans ton application "Grand Taxi" pour permettre aux utilisateurs de réserver un taxi facilement, tout en maintenant une taille d'application légère.

**GitHub** est une plateforme de développement collaboratif basée sur le système de contrôle de version **Git**. Elle permet aux développeurs de stocker, partager, suivre les modifications de leur code source, et collaborer efficacement via des dépôts (repositories).

**Vercel** est une plateforme de déploiement optimisée pour les applications React et Next.js. Elle est simple à utiliser, propose un déploiement instantané via **Git**, et fournit des performances optimales grâce à son infrastructure mondiale. Vercel peut héberger le frontend de ton application .

**Render** est une plateforme d'hébergement tout-en-un qui permet de déployer aussi bien des applications web (frontend) que des services backend (APIs, bases de données, workers). Elle est souvent utilisée pour héberger des serveurs Node.js ou FastAPI. Dans ton projet, Render peut être utilisée pour déployer tes microservices backend (ex. Problem Service ou Notification Service).

**Netlify** est une plateforme d'hébergement moderne principalement utilisée pour déployer des sites web statiques ou des applications frontend. Elle prend en charge les frameworks comme **React**, **Vue** ou **Vite**, avec un déploiement automatisé depuis **GitHub**. Dans ton projet, Netlify peut être utilisé pour héberger l'interface frontend React.js de manière rapide, sécurisée et gratuite.

## V. Product Backlog global

| ID | User Story | Priorité |
|---|---|---|
| 1 | En tant qu'administrateur, je voudrais créer un utilisateur | Élevée |
| 2 | En tant qu'administrateur, je voudrais modifier les données d'un utilisateur | Élevée |
| 3 | En tant qu'administrateur, je voudrais supprimer un utilisateur | Moyenne |
| 4 | En tant qu'administrateur, je voudrais assigner un chef a un assistant | Élevée |
| 5 | En tant qu'administrateur, je voudrais consulter tous les utilisateurs | Élevée |
| 6 | En tant qu'administrateur, je peux créer des problèmes prédéfinis | Moyenne |
| 7 | En tant qu'administrateur, je peux modifier des problèmes prédéfinis | Moyenne |
| 8 | En tant qu'administrateur , je peux supprimer des problèmes prédéfinis | Moyenne |
| 9 | En tant que Administrateur , je peux accéder au tableau de board | Haute |
| 10 | En tant que Administrateur , je peux consulter les statistiques des utilisateurs | Haute |
| 11 | En tant que Administrateur , je peux consulter les statistique des problèmes | Haute |
| 12 | En tant que Administrateur , je peux consulter l'état du micoservices | Haute |
| 13 | En tant que Administrateur , je peux consulter la l'état des stations | Moyenne |
| 14 | En tant qu' Administrateur, je veux poser des questions sur les stations. | Haute |

| | | |
|---|---|---|
| 15 | En tant qu'Administrateur, je veux charger les données CSV | Haute |
| 16 | En tant que chef, je peux marquer un problème comme résolu | Haute |
| 17 | En tant que chef, je peux créer un problème personnalisé | Moyenne |
| 18 | En tant que technicien, je peux accepter un problème | Haute |
| 19 | En tant que technicien, je peux consulter les problèmes qui me sont assignés | Haute |
| 20 | En tant que système, je peux attribuer automatiquement un technicien selon le type | Haute |

*Table 5 : Product Backlog global*

## Conclusion

Dans ce chapitre, nous avons analysé les besoins fonctionnels et non fonctionnels de notre système, en mettant l'accent sur les microservices et leurs interactions. Nous avons également identifié les acteurs clés qui interagissent avec le système. Cette analyse nous permet de mieux comprendre les exigences du projet et de préparer la phase de conception et de développement. Dans le prochain chapitre, nous aborderons la conception détaillée de l'architecture et des composants du système.

# Chapitre 4

## Release 1: Gestion des utilisateurs

**Plan**

**Introduction**

**I.Sprint 1: Gestion des utilisateurs**

**Conclusion**

# I.Sprint 1: Gestion des utilisateurs

## Introduction

Le microservice **gestion des utilisateurs** joue un rôle fondamental dans l'architecture de notre système. Il est responsable de la gestion des utilisateurs, de l'authentification et de l'attribution des rôles. Ce microservice permet aux administrateurs de gérer les comptes utilisateurs et aux autres microservices de vérifier les permissions via l'API Gateway.

### I.1 Backlog du Microservice

| ID | User Story | Priorité |
|---|---|---|
| 1 | En tant qu'administrateur, je voudrais créer un utilisateur | Élevée |
| 2 | En tant qu'administrateur, je voudrais modifier les données d'un utilisateur | Élevée |
| 3 | En tant qu'administrateur, je voudrais supprimer un utilisateur | Moyenne |
| 4 | En tant qu'administrateur, je voudrais assigner un chef a un assistant | Élevée |
| 5 | En tant qu'administrateur, je voudrais consulter tous les utilisateurs | Élevée |
| 6 | En tant qu'utilisateur je voudrais m'authentifier avec mon e-mail et mot de passe | Moyenne |
| 7 | En tant qu'utilisateur je voudrais consulter mes données | Faible |
| 8 | En tant qu'assistant je voudrais consulter mes chefs | Faible |
| 9 | En tant que chef je voudrais consulter mon assistant | Moyenne |

*Table 6: Backlog du microservice Gestion utilisateur*

## I.1. Conception

### I.1.a. Diagramme de Cas d'Utilisation

Ce diagramme illustre les interactions entre les quatre acteurs du système (Administrateur, Chef, Assistant et Technicien) et les fonctionnalités centrales du microservice. Il met en évidence les opérations spécifiques à chaque rôle : la gestion complète des utilisateurs (création, modification, suppression) pour l'Administrateur, l'assignation de chefs aux assistants, ainsi que les fonctionnalités de consultation (données personnelles, chefs assignés ou assistant référent). L'authentification sécurisée est un prérequis commun à toutes ces opération

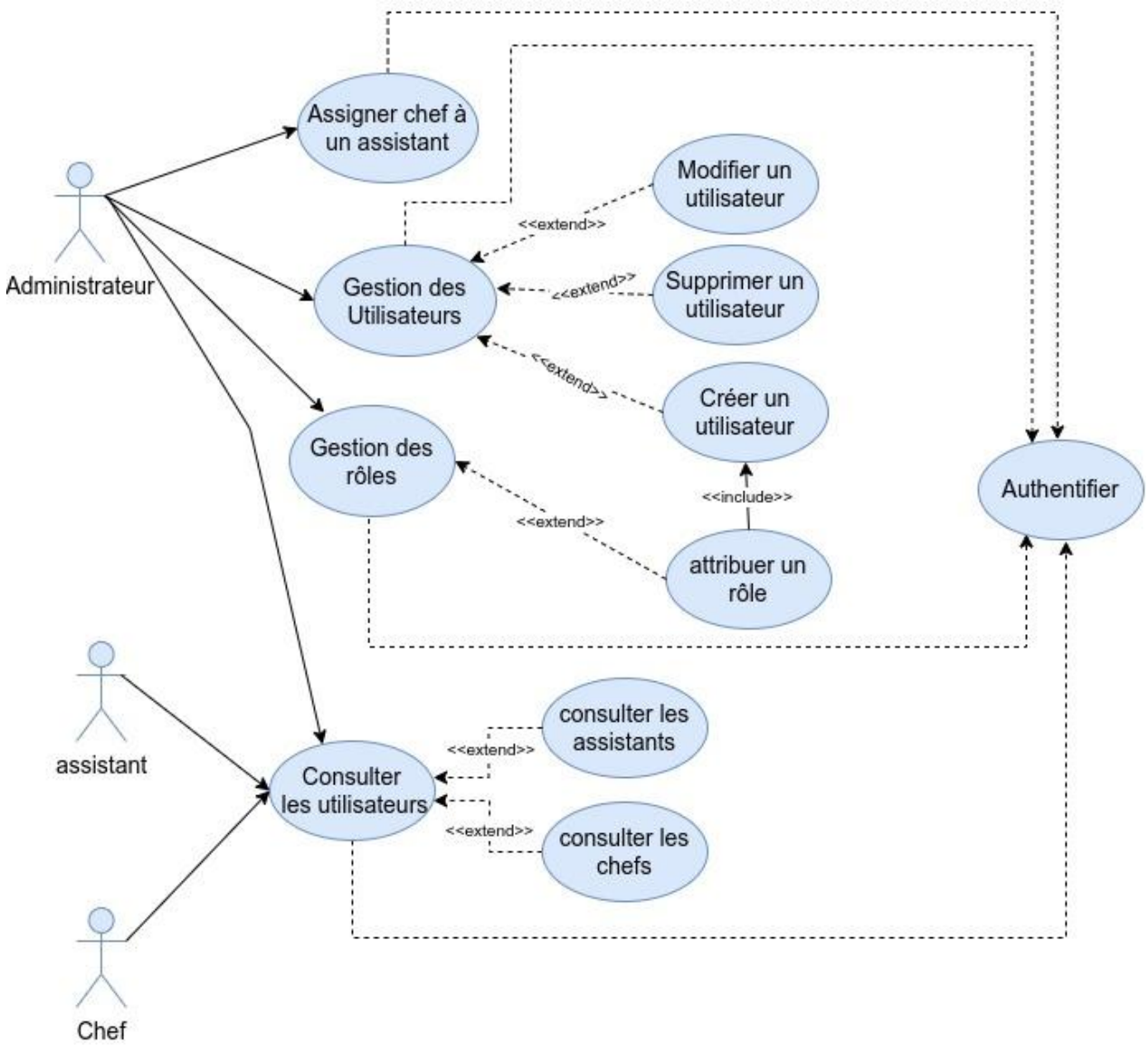



*Figure 7 : Diagramme de cas d'utilisation de gestion des utilisateurs*

| Gestions des utulisateurs | |
|---|---|
| **Résumé** | Le diagramme illustre un système de gestion des utilisateurs avec différentes fonctionnalités comme la création, la modification, la suppression des utilisateurs, l'attribution de rôles, et l'authentification. Il inclut également des scénarios pour consulter les données et gérer les relations hiérarchiques (chef/assistant). |
| **Acteurs** | - Administrateur : Gère les utilisateurs, les rôles et les données.<br>- Assistant : Consulte ses chefs.<br>- Chef : Consulte son assistant.<br>- Technicien : Consulte les données. |
| **Préconditions** | - Le système doit être accessible et opérationnel.<br>- L'utilisateur doit être authentifié.<br>- Les rôles et permissions doivent être définis. |
| **Postconditions** | - Les utilisateurs sont créés/modifiés/supprimés avec succès.<br>- Les rôles sont attribués.<br>- Les données sont consultables.<br>- Les relations hiérarchiques sont mises à jour. |
| **Scénario nominal** | 1. L'administrateur se connecte.<br>2. Il crée/modifie/supprime un utilisateur.<br>3. Il attribue un rôle ou assigne un chef à un assistant.<br>4. Les autres acteurs consultent les données selon leurs permissions. |
| **Scénarios alternatifs** | - Échec d'authentification : Accès refusé.<br>- Données manquantes : Opération annulée.<br>- Conflit de rôles : Notification d'erreur. |
| **Exigences non fonctionnelles (NFN)** | - Sécurité : Authentification requise.<br>- Performance : Réponse rapide aux requêtes.<br>- Disponibilité : Accès 24/7.<br>- Ergonomie : Interface intuitive. |

*Table 7 : Tableau des informations clés sur la gestion des utilisateurs*

### I.1.b. Diagramme de Classes

Ce diagramme de classes représente la structure centrale du microservice de gestion des utilisateurs. Il repose sur une hiérarchie de classes avec une classe de base *User* et quatre sous-classes spécialisées (*Admin*, *Assistant*, *Chef*, *Technicien*), chacune disposant d'attributs et de méthodes spécifiques à leur rôle.

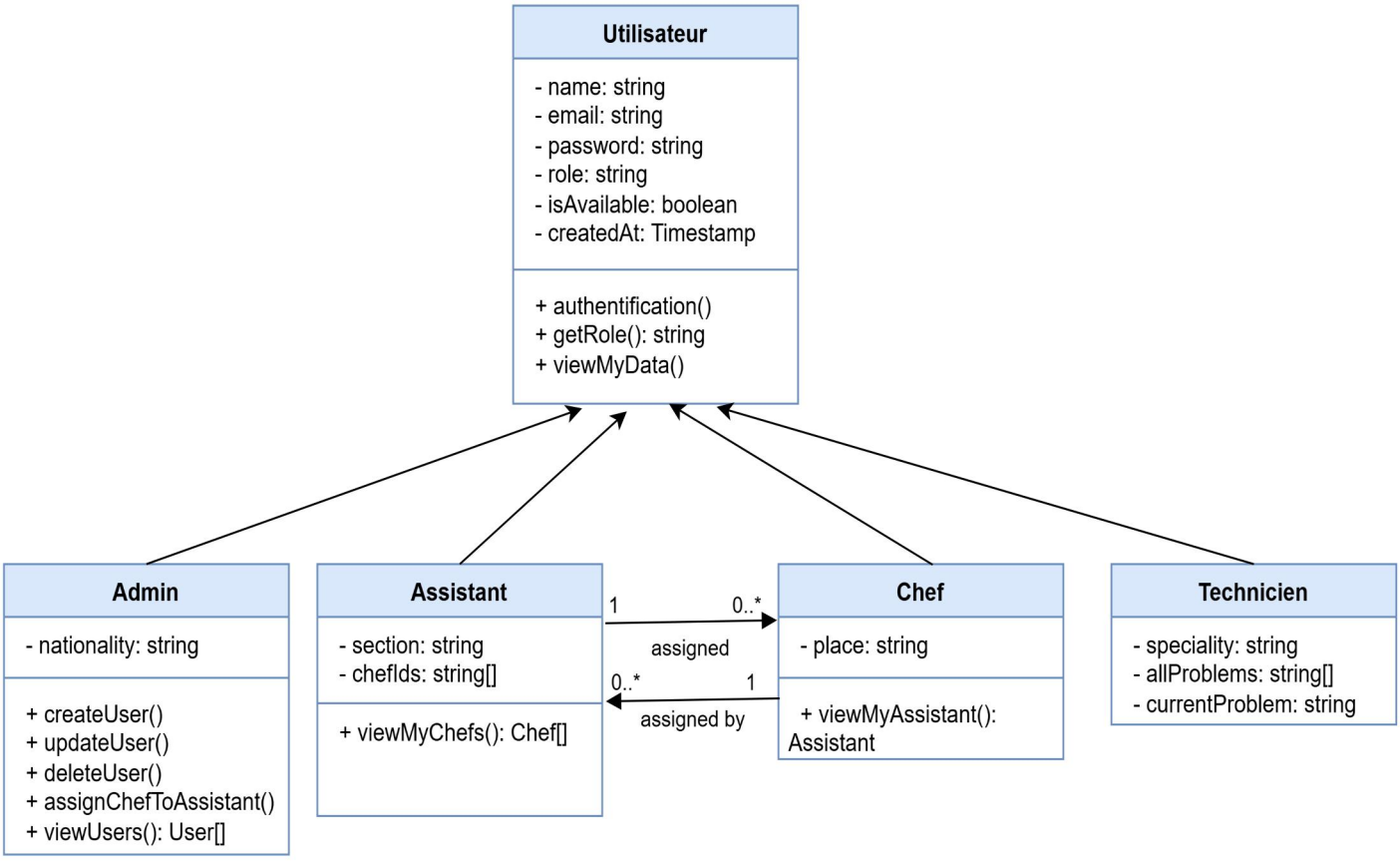



*Figure 8 : Diagramme de classes de gestion des utilisateurs*

### I.1.c. Diagramme de Séquence

- **Authentification**

L'authentification est une étape essentielle pour sécuriser l'accès au système. Elle permet de vérifier l'identité des utilisateurs et d'assurer qu'ils disposent des permissions nécessaires pour accéder aux fonctionnalités de l'application. Le diagramme suivant illustre le processus d'authentification.

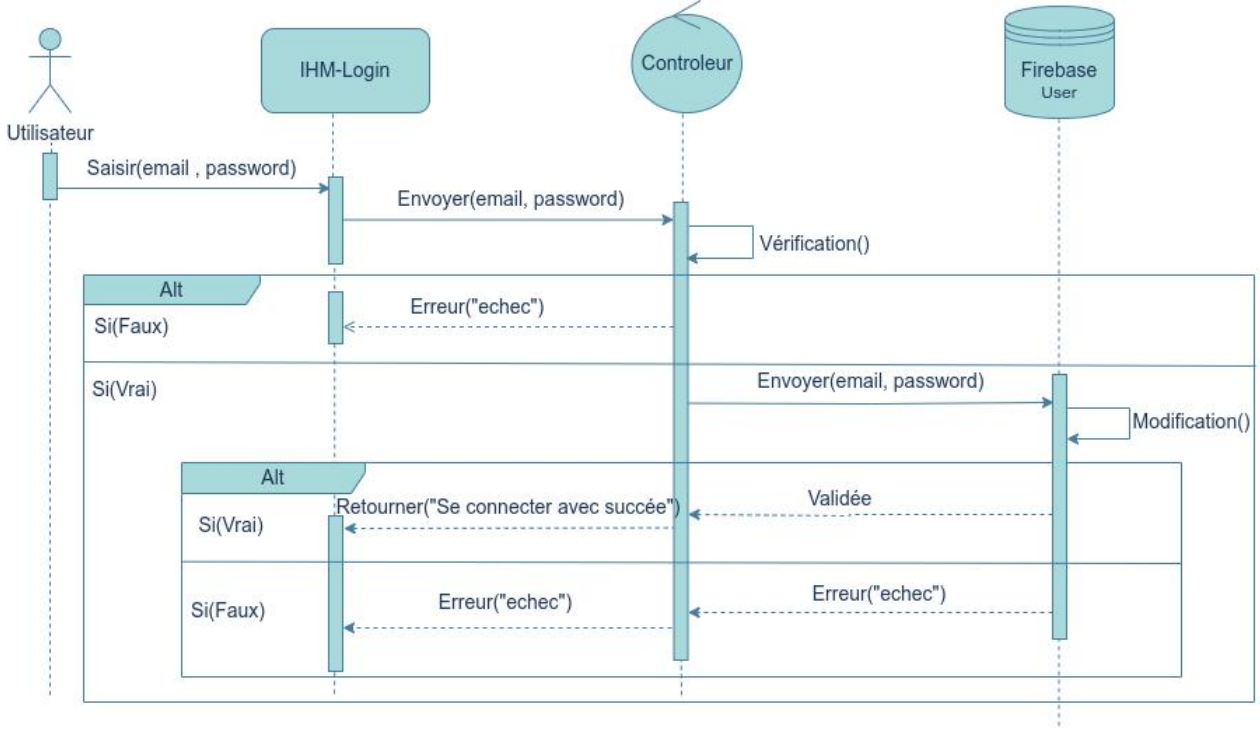



Figure 9: Diagramme de séquence d'Authentification des utilisateurs

- **Création d'un chef**

L'ajout d'un nouvel utilisateur est une fonctionnalité clé du User Service, permettant aux administrateurs de gérer les comptes. Ce processus comprend la saisie des informations, la validation et l'enregistrement dans la base de données. Le diagramme ci-dessous détaille ce processus.

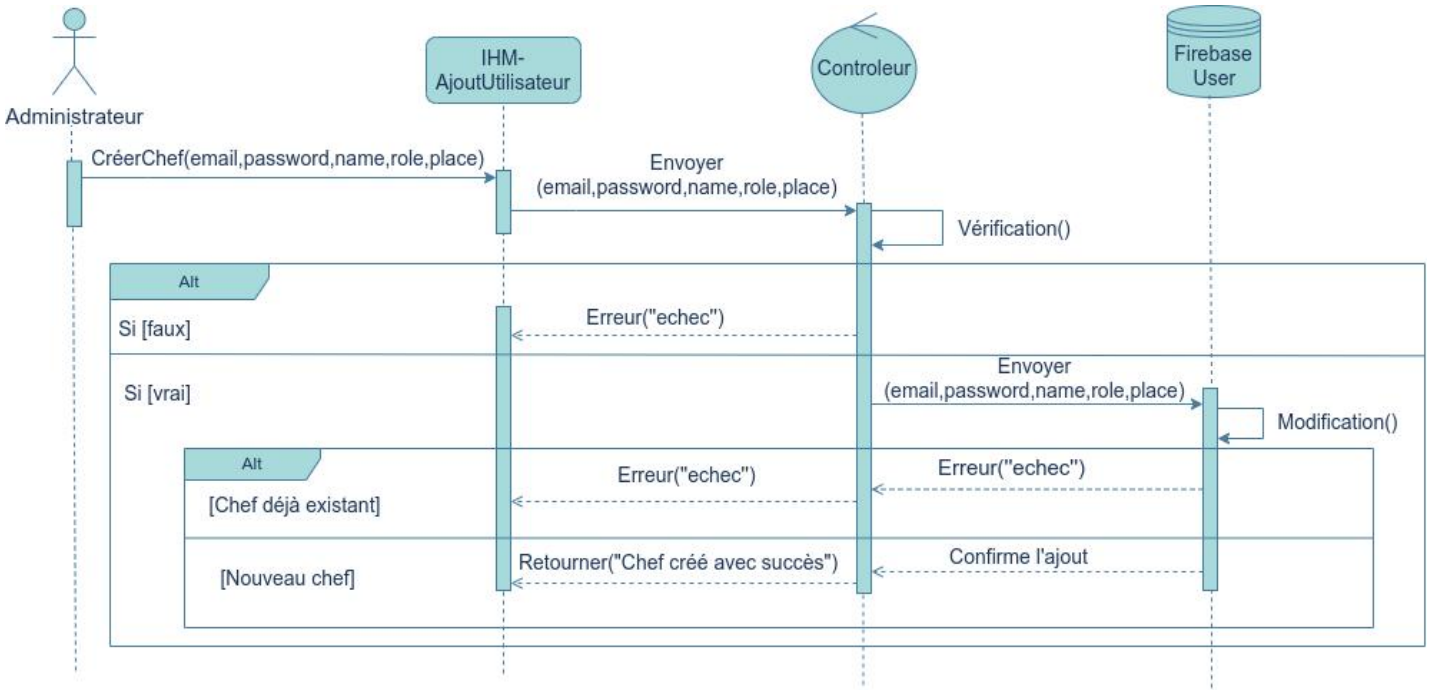



*Figure 10 : Diagramme de séquence d'ajout d'un nouveau chef*

- **Modifier les données d'un Utilisateur**

Pour assurer une gestion efficace des utilisateurs, il est crucial de pouvoir modifier leurs informations et leurs rôles. Cette fonctionnalité permet aux administrateurs d'adapter les accès et les permissions en fonction des besoins. Le diagramme suivant illustre ce processus.

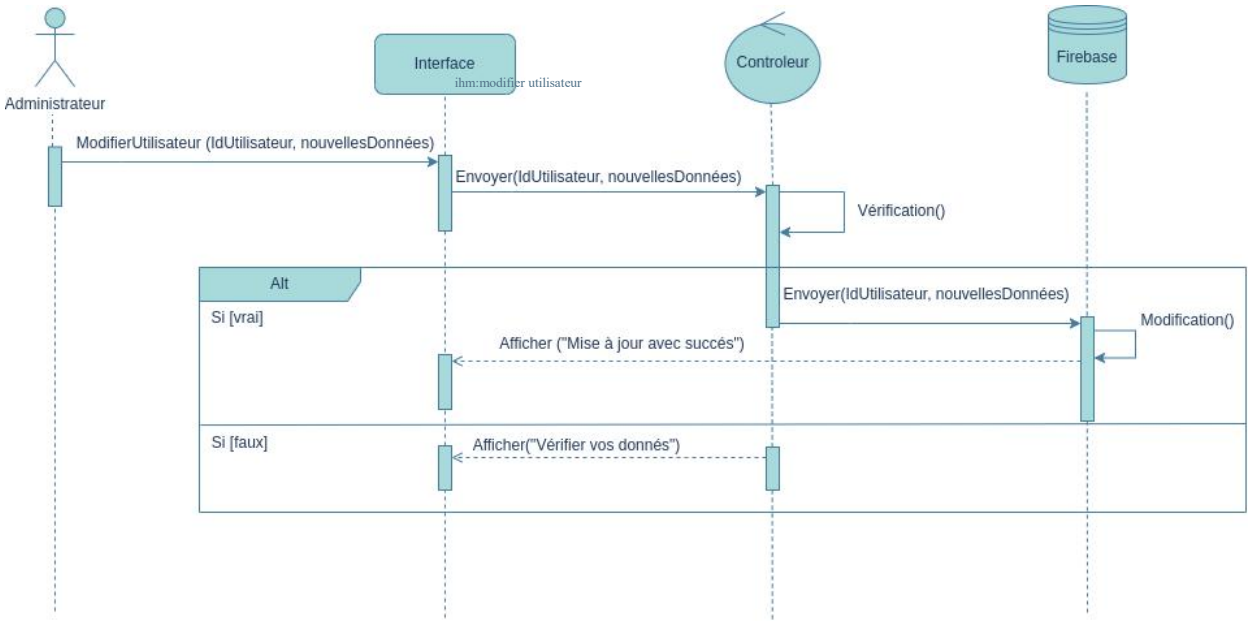



*Figure 11 : Diagramme de séquence de modification d'un utilisateur*

### I.1.d. Diagramme de deployement

Ce diagramme montre comment une demande utilisateur est traitée :

- **Interface** : L'utilisateur envoie une demande.
- **API Gateway** (Express.js) : Reçoit et dirige la demande.
- **Firebase Auth** : Vérifie l'identité de l'utilisateur.
- **Firestore** : Récupère les données de l'utilisateur.

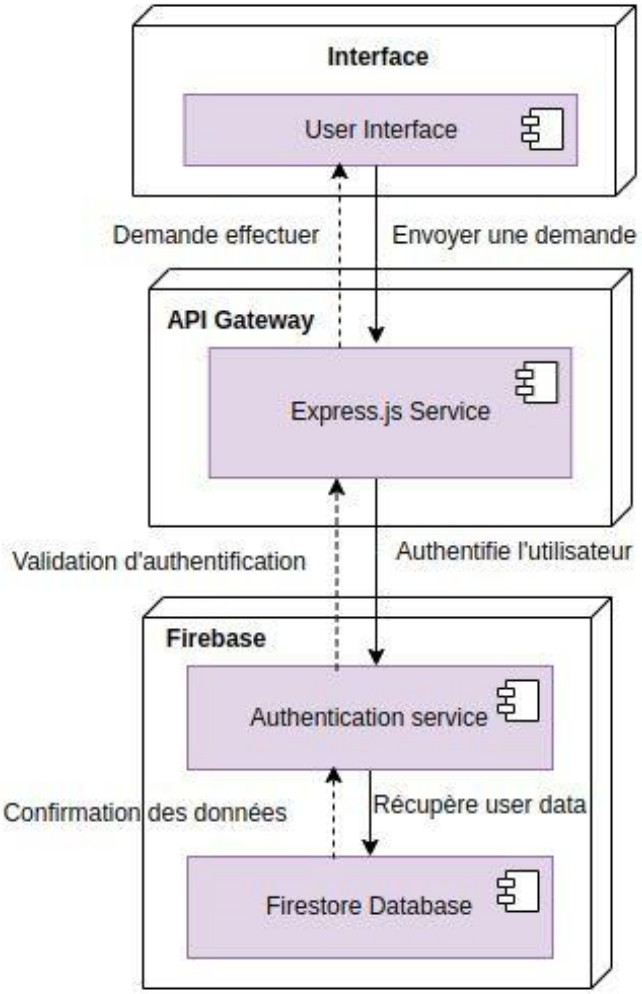



*Figure 12 : Diagramme de déploiement de gestion des utilisateurs*

## I.2. Architecture

Le service de gestion des utilisateurs s'intègre parfaitement dans notre architecture microservices, fonctionnant comme un module autonome mais connecté qui échange en permanence et de manière sécurisée avec tous les autres composants du système, formant ainsi un élément central et indispensable de notre écosystème applicatif.

### I.2.a. API Gateway :

- Point d'entrée unique pour toutes les requêtes
- Filtrage et routage vers les services concernés

| Method | Route | Description |
|---|---|---|
| GET | / | Welcome route |
| POST | /users | Create a new user |
| GET | /users | List all users or filtered |
| GET | /users/:id | Get user by ID |
| PUT | /users/:id | Update user |
| PUT | /users/:id/availability | Update technician availability |
| POST | /login | Login and get token |
| PUT | /users/assign-chef/:assistantId | Assign a chef to assistant |
| GET | /users/:id/assigned-chefs | Get assigned chefs for assistant |
| GET | /assigned-chefs | Get all assigned chef IDs |

*Table 8: Routes et Méthodes de l'API gateway*

### I.2.b. Firebase Authentication :

- Vérification des identités utilisateurs

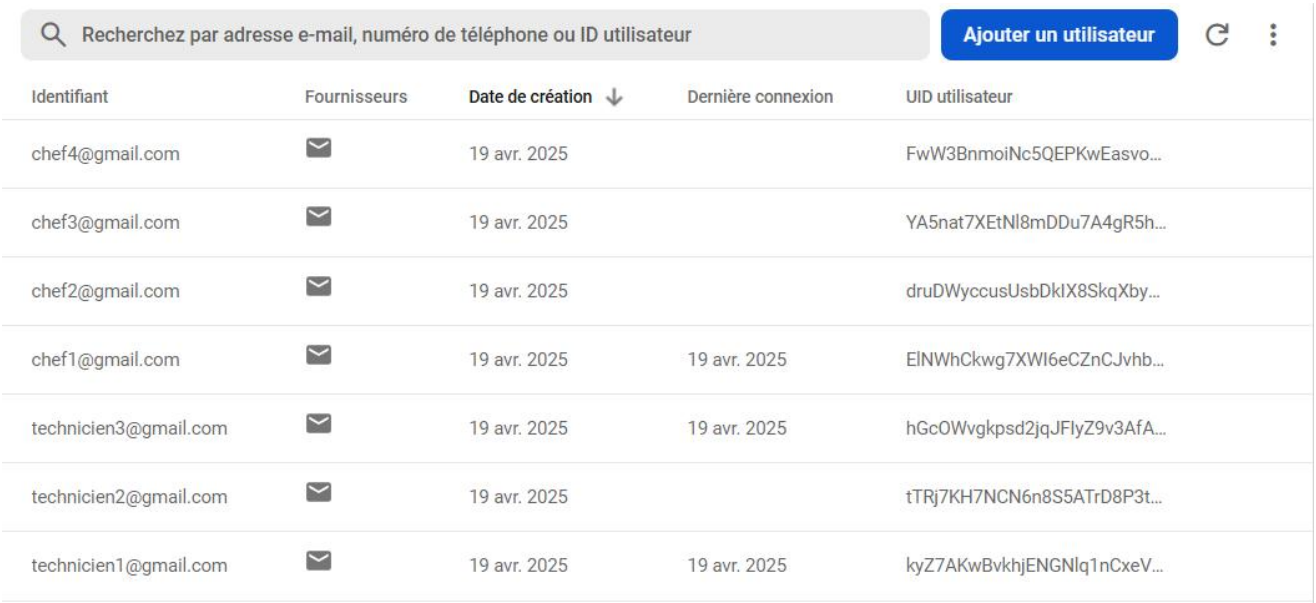



*Figure 13 : Firebase Authentification*

### I.2.c. Firebase Firestore :

- Base de données Temps réel pour le stockage .

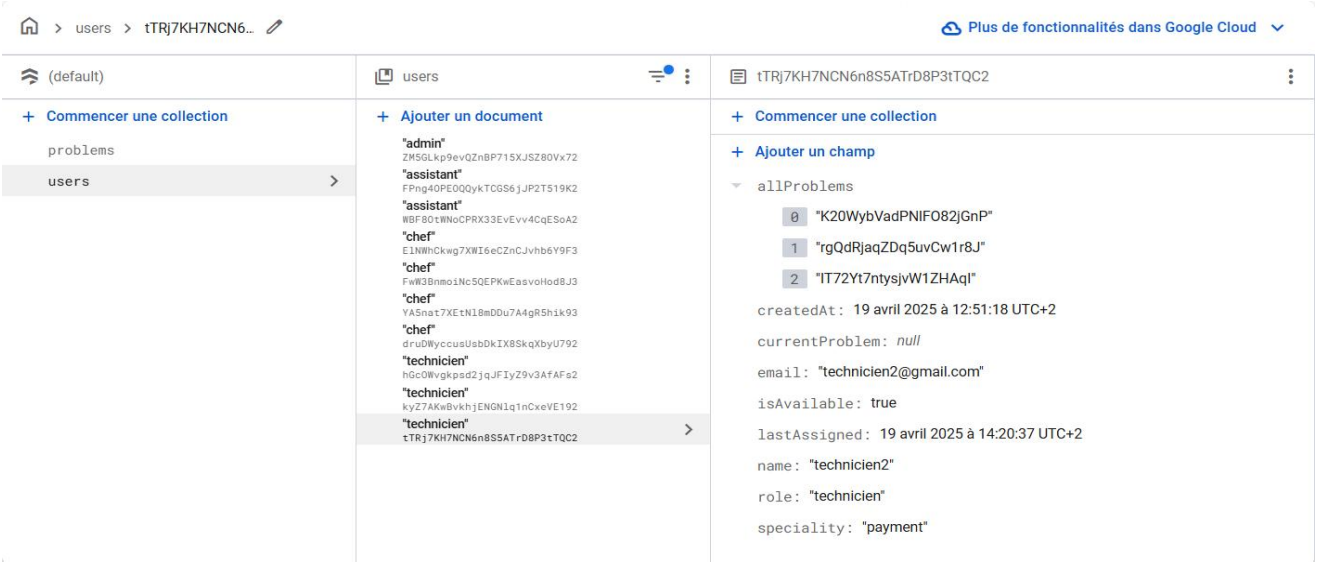



*Figure 14 : Base de données de gestion des utilisateurs*

### I.2.d. Connexion avec d'autres services :

- Service de gestion des problèmes
- Service de notifications

*Figure 15 : communication entre microservices*

## I.3. Implementation

### I.3.a. Tableau Scrum

- **Premier jour**

**Release 1: Gestion des utilisateurs**

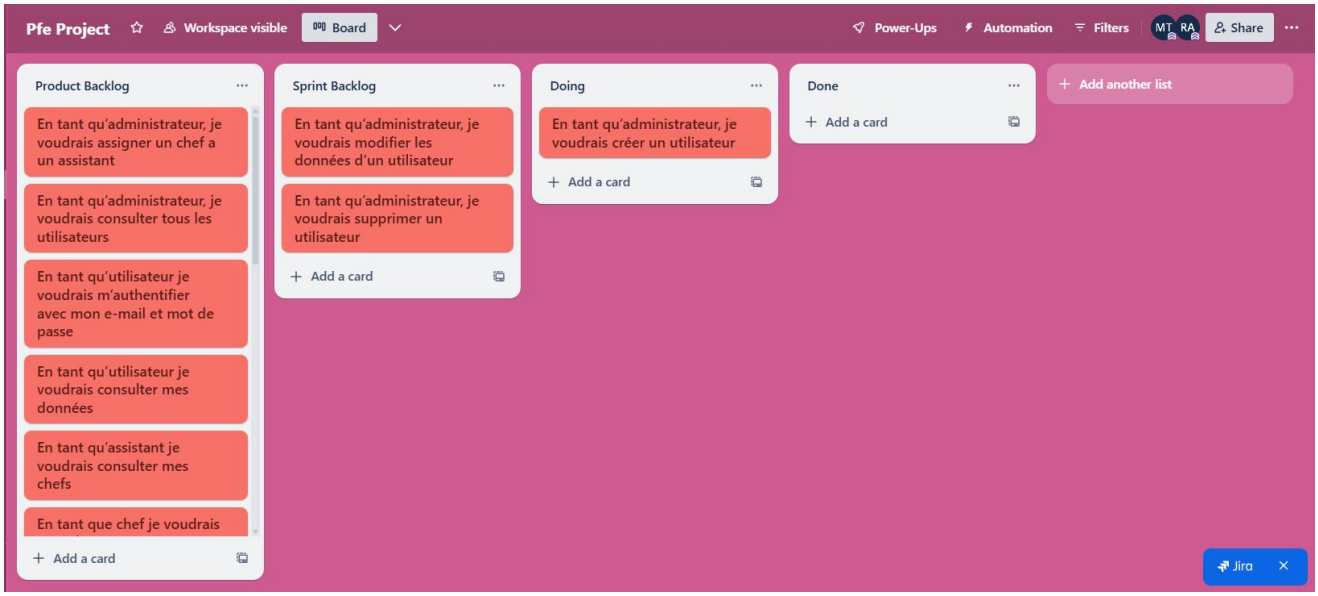



Figure 16 : Tableau scrum pour le premier jour

- **Dernier jour**

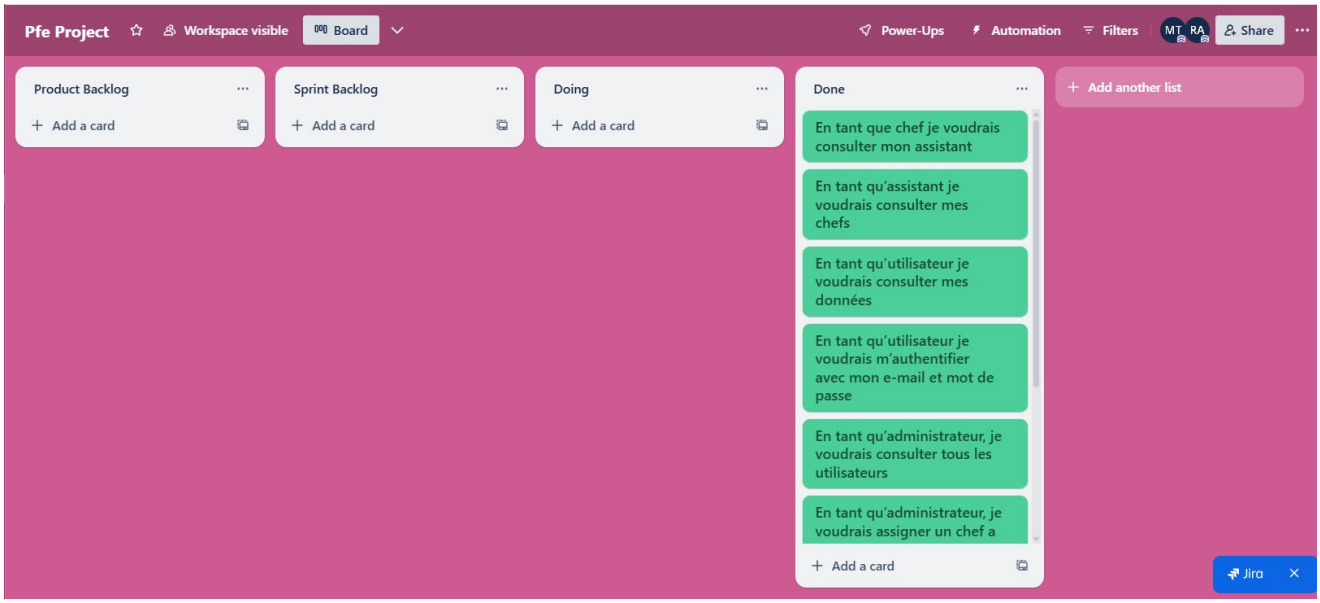



*Figure 17 : Tableau scrum pour le dernier jour*

### I.3.b. Interface

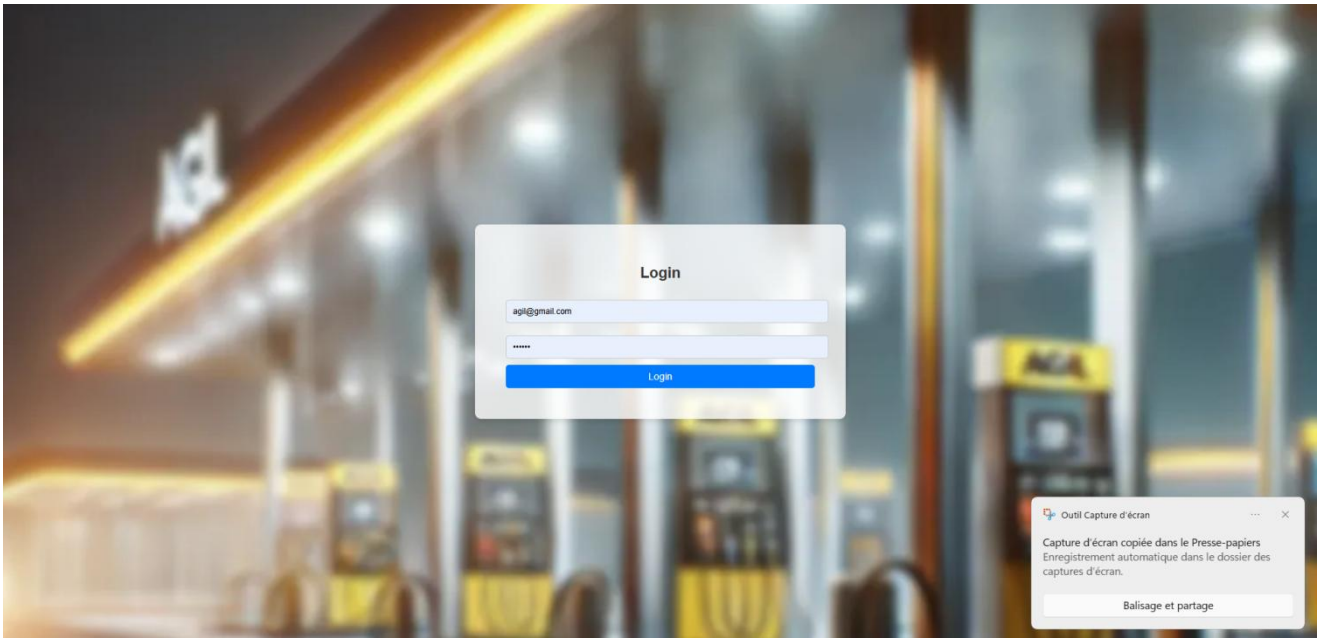



Figure 18 : Interface d'authentification

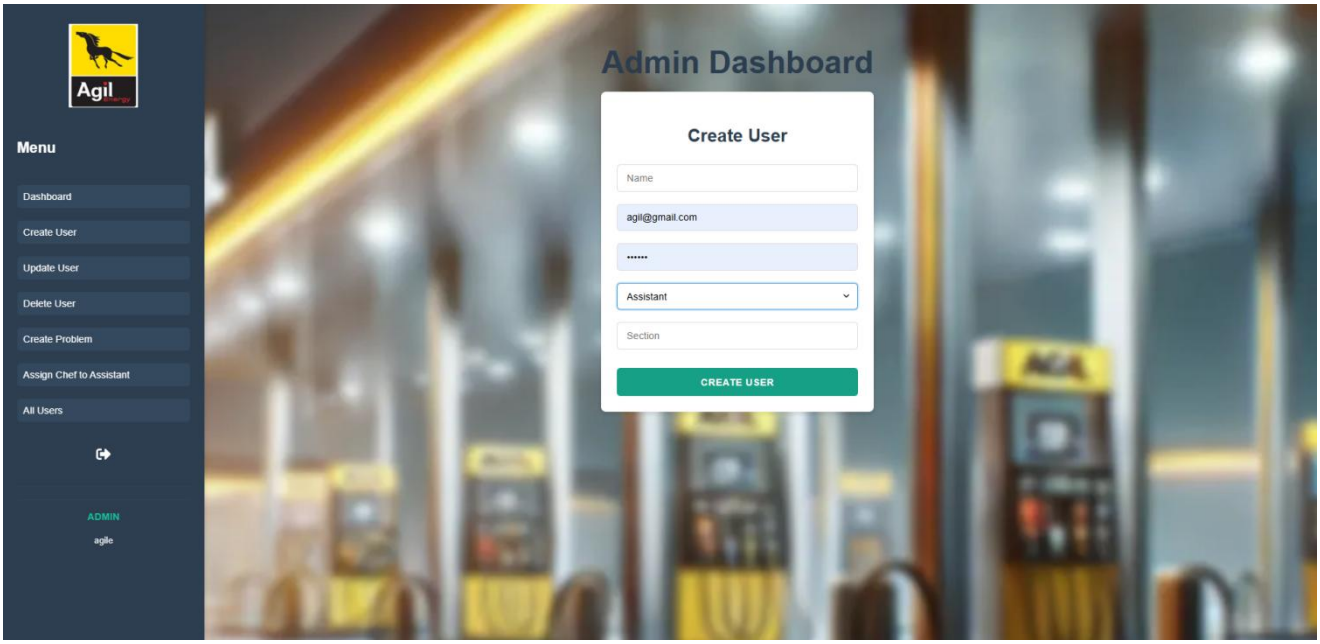



Figure 19 : Interface de création d'un utilisateur

**Release 1: Gestion des utilisateurs**

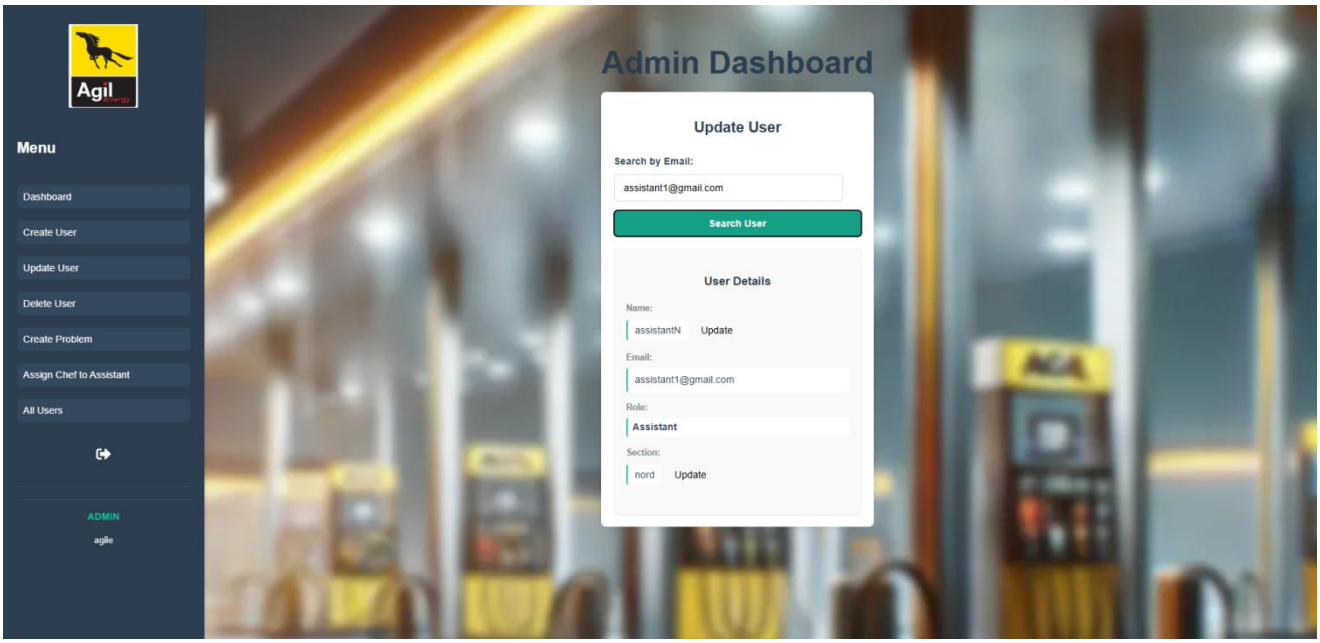



Figure 20 : Interface modifier un utilisateur

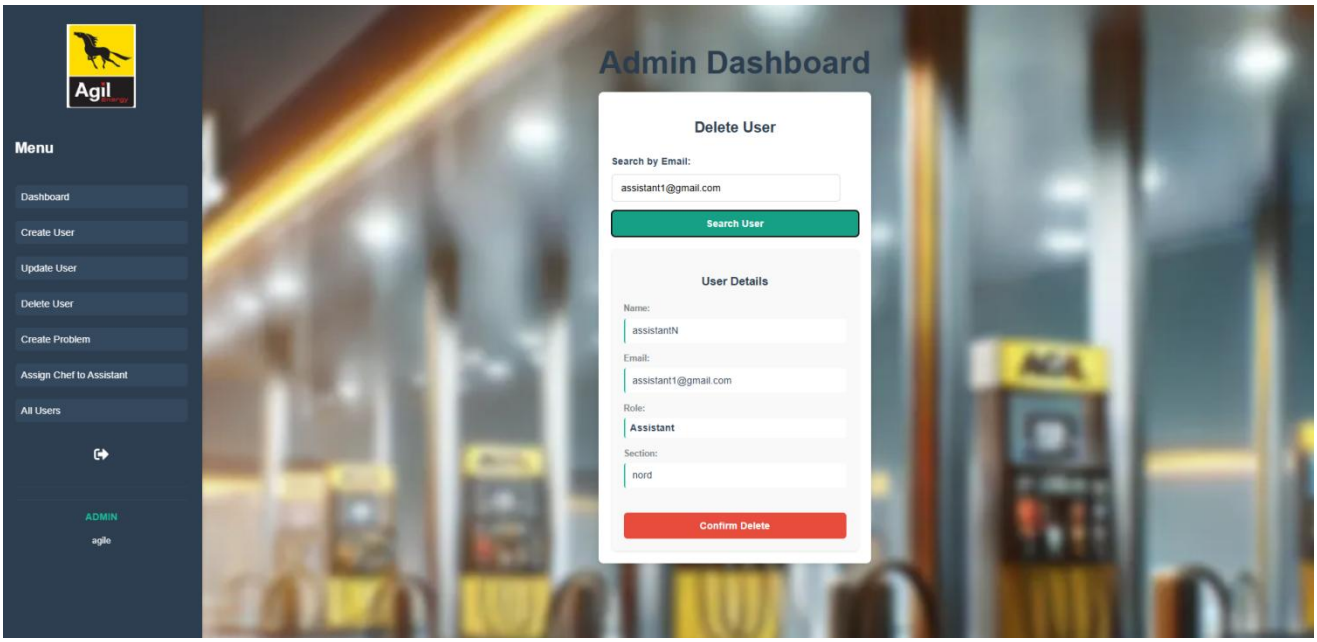



Figure 21 : Interface supprimer un utilisateur

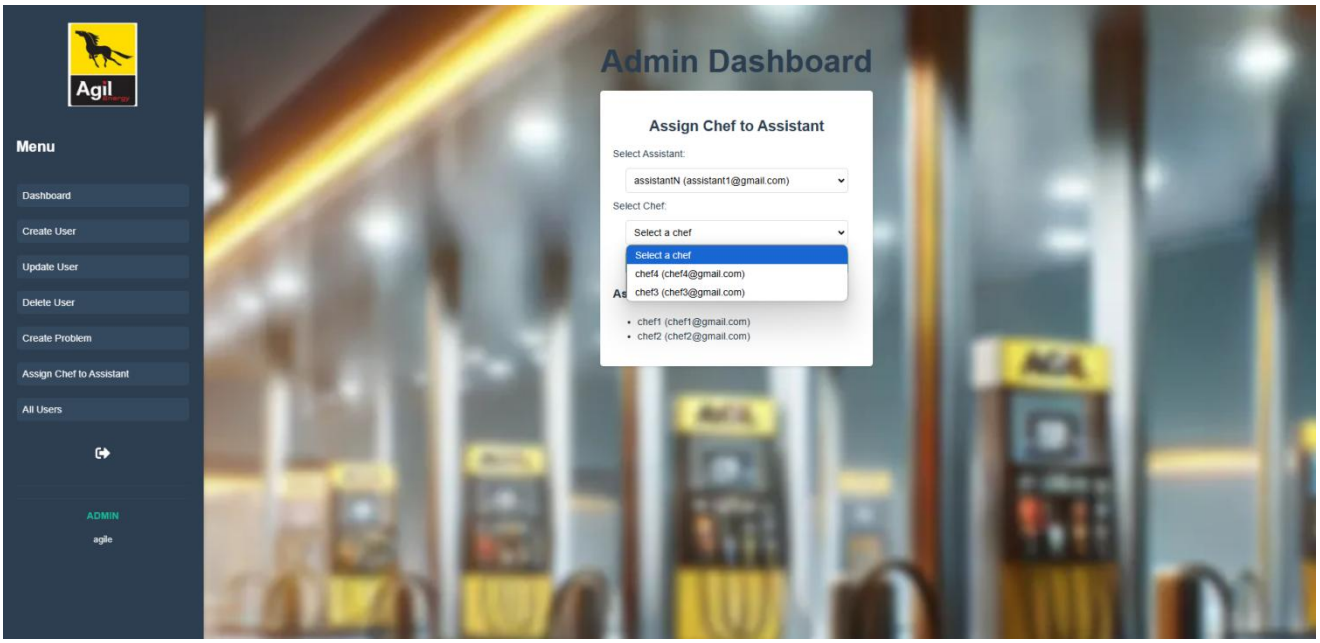



Figure 22 : Interface dassignation d'un chef a un assistant

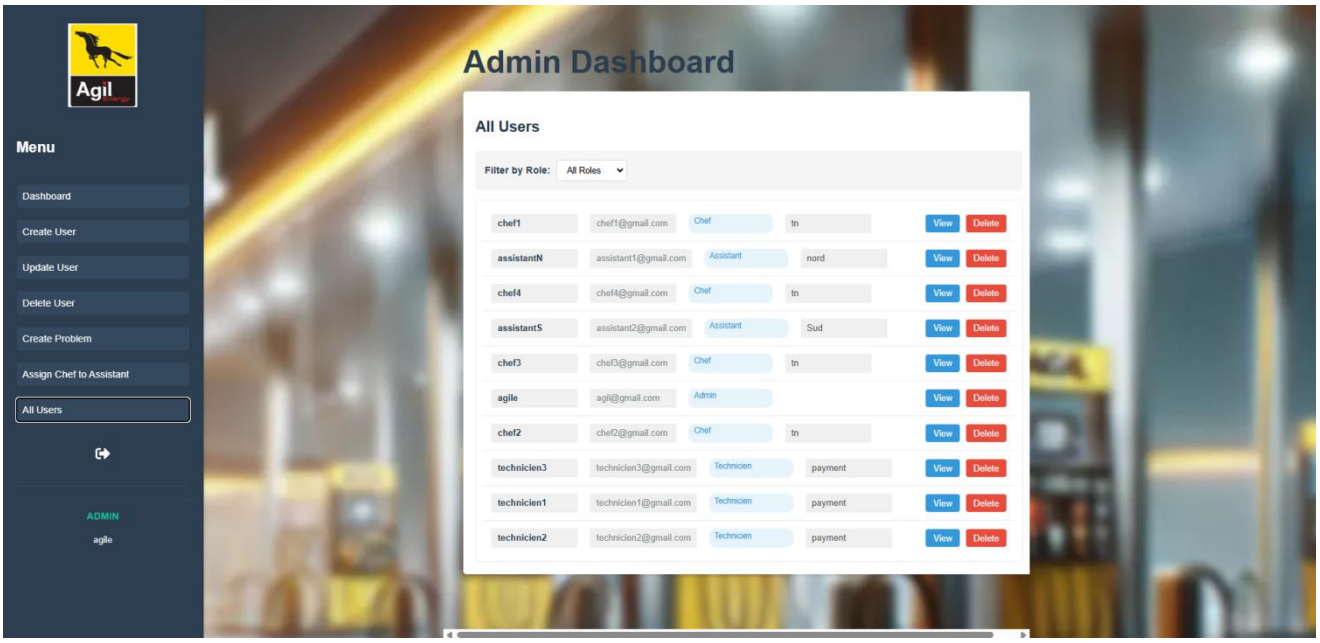



Figure 23 : Interface consulter tous les utilisateurs

# Conclusion

Ce microservice offre une solution clé-en-main pour gérer les utilisateurs en toute sécurité. Intégré naturellement à notre architecture, il combine simplicité d'utilisation et robustesse technique. Son design modulaire permet des évolutions sans remettre en cause l'existant, tandis que ses connexions fluides avec les autres services en font un maillon essentiel de notre écosystème. Une base solide, pensée pour  nous adapter à nos besoins actuels et futurs.

# Chapitre 5

## Release 2: Gestion des Problémes

**PLAN**

**Introduction**

I. **Sprint 1 : Déclaration et gestion des problèmes**
II. **Sprint 2 : suivi et coordination**

**Conclusion**

# Introduction

Ce chapitre présente l'évolution de notre application avec l'ajout d'un service dédié à la gestion des problèmes dans les stations-service. Il détaille les fonctionnalités développées lors de deux sprints successifs : la déclaration initiale des problèmes par les chefs, puis le suivi et la coordination entre techniciens, assistants et administrateurs.

# I. Sprint 1: Déclaration et gestion des problèmes

Ce sprint pose les bases du service. Il permet au chef de station de signaler un problème, de suivre son statut et de le marquer comme résolu. L'administrateur peut quant à lui définir une liste de problèmes types réutilisables.

## I.1. Backlog du premier Sprint

| ID | Histoire utilisateur | Priorité |
|---|---|---|
| 1 | En tant que chef, je peux signaler un problème prédéfini | Haute |
| 2 | En tant que chef, je peux consulter la liste de mes problèmes | Haute |
| 3 | En tant que chef, je peux marquer un problème comme résolu | Haute |
| 4 | En tant qu'administrateur, je peux créer des problèmes prédéfinis | Moyenne |
| 5 | En tant qu'administrateur, je peux modifier des problèmes prédéfinis | Moyenne |
| 6 | En tant qu'administrateur , je peux supprimer des problèmes prédéfinis | Moyenne |
| 7 | En tant que chef, je peux créer un problème personnalisé | Moyenne |

*Table 9: Backlog du premier sprint*

## I.2. Conception

### I.2.a. Diagramme de Cas d'Utilisation

Ce diagramme de cas d'utilisation présente les interactions principales entre les chefs et les administrateurs au sein du microservice "Problem Service". Il illustre les fonctionnalités offertes aux chefs pour déclarer un problème, qu'il soit prédéfini ou libre, ainsi que la possibilité pour l'administrateur d'ajouter, modifier ou supprimer des problèmes prédéfinis. Chaque problème est associé à un type et à une description, et peut déclencher automatiquement une assignation à un technicien disponible en fonction de sa spécialité. L'accès sécurisé à ces fonctionnalités repose sur une authentification préalable, garantissant que seules les personnes autorisées peuvent effectuer des opérations sensibles.

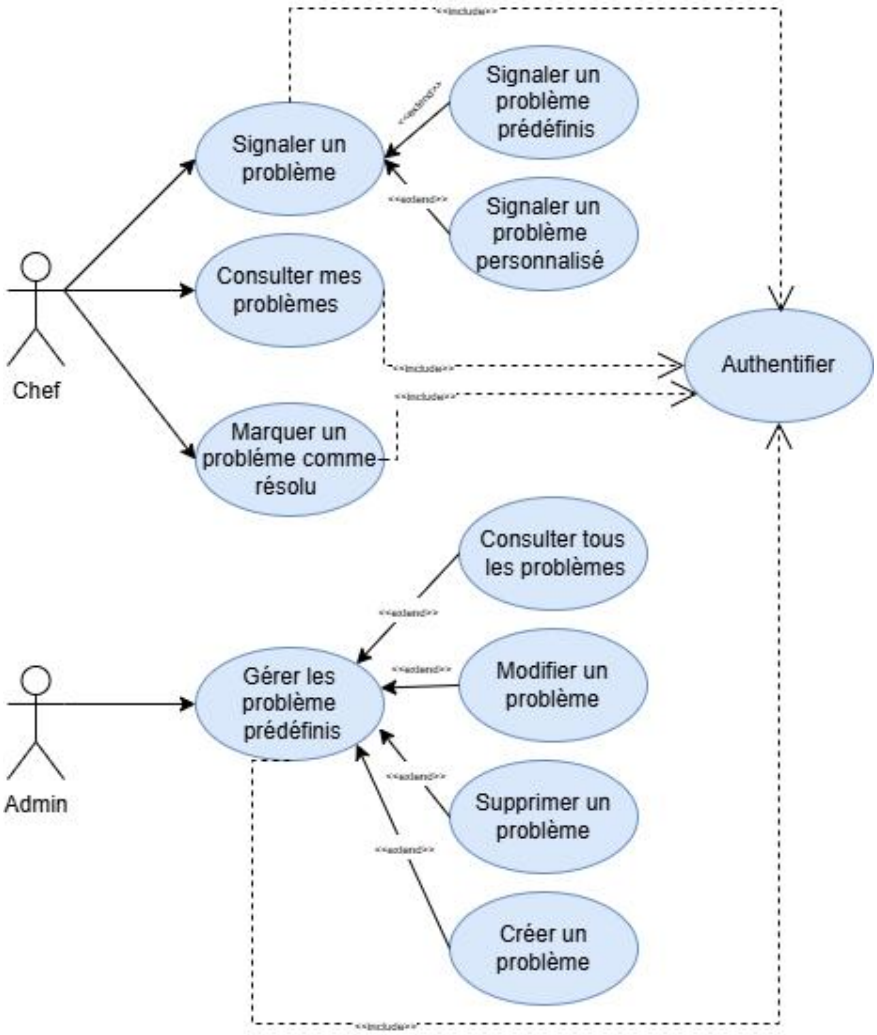



*Figure 24 : Diagramme de cas d'utilisation Sprint 1*

| Gestion des problemes | |
|---|---|
| **Résumé** | Le diagramme de cas d'utilisation illustre la manière dont les chefs peuvent signaler des problèmes dans les stations-service, en choisissant parmi une liste de problèmes prédéfinis ou en saisissant un problème libre. Il décrit également les fonctionnalités offertes à l'administrateur pour gérer cette liste (ajout, modification, suppression). |
| **Acteurs** | - Chef : Déclare un problème.<br>- Administrateur : Gère la liste des problèmes prédéfinis. |
| **Préconditions** | - L'utilisateur doit être connecté (authentifié).<br>- La station doit être sélectionnée ou identifiée.<br>- Les problèmes prédéfinis doivent être disponibles. |
| **Postconditions** | - Le problème est enregistré avec succès dans le système.<br>- Le type de problème est identifié.<br>- Le problème peut être assigné à un technicien. |
| **Scénario nominal** | 1. Le chef s'authentifie et accède à l'interface.<br>2. Il sélectionne ou écrit un problème.<br>3. Le problème est enregistré avec son type et l'ID du chef.<br>4. L'administrateur peut mettre à jour la liste des problèmes. |
| **Scénarios alternatifs** | - Problème non soumis : Données manquantes ou invalides.<br>- L'accès est refusé en cas d'échec d'authentification.<br>- Tentative de création d'un doublon de problème : rejet de la requête. |
| **Exigences non fonctionnelles (NFN)** | - Sécurité : Authentification obligatoire.<br>- Performance : Enregistrement rapide des problèmes.<br>- Fiabilité : Les problèmes doivent être sauvegardés même en cas de coupure.<br>- Ergonomie : Interface simple pour signaler un problème. |

*Table 10 : Tableau des informations clés sur la gestion des problèmes*

### I.2.b. Diagramme de classes

Ce diagramme met en évidence les principales entités du système ainsi que leurs relations. On retrouve les classes suivantes(probléme prédéfini,probléme personalisé,Admin,Chef).

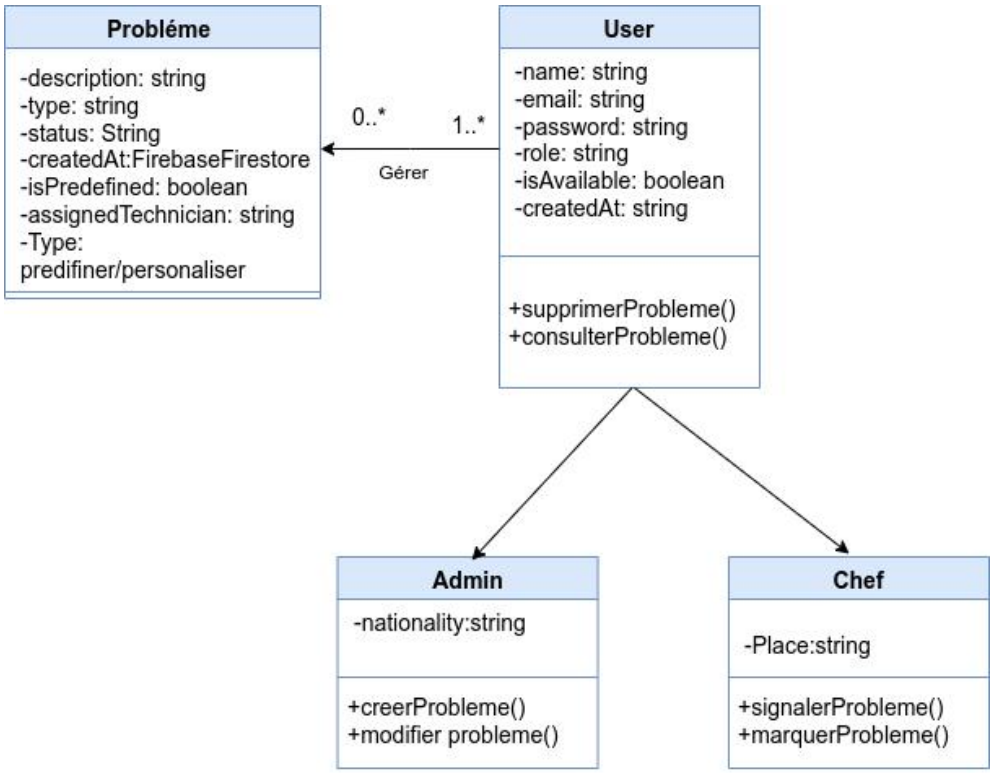



*Figure 25 : Diagramme de classes sprint 1*

### I.2.c. Diagramme de Séquence

**Création d'un probleme prédéfini**

Ce diagramme de séquence illustre le processus de création d'un problème prédéfini par un administrateur au sein du microservice de gestion des problèmes. L'interaction commence lorsque l'administrateur utilise l'interface frontend pour saisir les détails d'un nouveau problème . Le frontend envoie ensuite une requête POST à l'API backend contenant ces informations. L'API traite la demande et communique avec la base de données Firestore pour ajouter un nouveau document dans la collection Problem. Une fois l'opération réussie, Firestore renvoie une confirmation au backend, qui à son tour informe le frontend du succès de la création. L'interface affiche ensuite un message de confirmation à l'administrateur. Ce processus garantit une gestion centralisée et structurée des types de problèmes récurrents signalés dans le système.

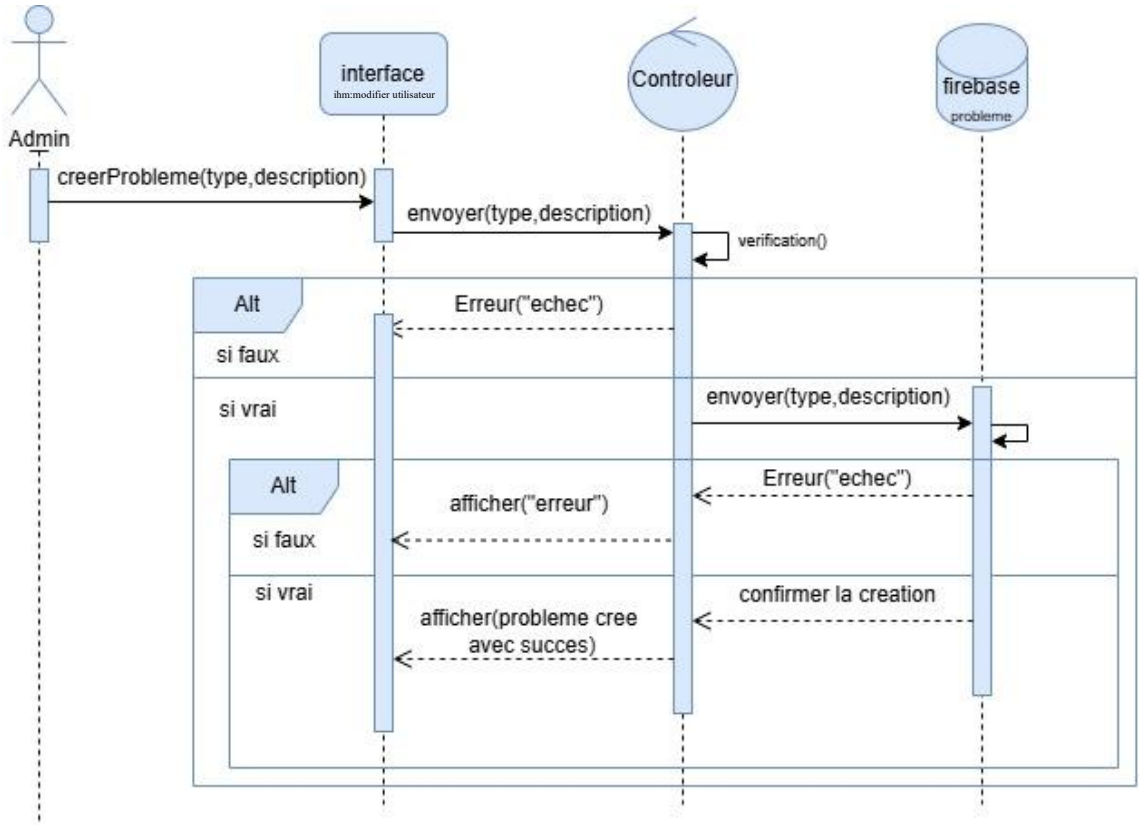



Figure 26 : Diagramme de séquence créer un problème prédéfini

- **Signaler un probléme prédéfini**

Ce diagramme de séquence décrit le scénario dans lequel un chef signale un problème prédéfini via l'application. L'interaction commence lorsque le chef sélectionne un problème prédéfini à partir d'une liste affichée sur l'interface frontend. Après validation, une requête POST est envoyée au backend contenant l'identifiant du problème prédéfini, l'identifiant du chef. Le backend reçoit cette requête, vérifie l'authenticité et la validité des données, puis enregistre un nouveau document dans la collection Problem de Firestore, en liant le problème prédéfini à l'utilisateur concerné. Une fois l'enregistrement effectué, une réponse de succès est transmise au frontend, qui affiche une confirmation à l'utilisateur. Ce processus assure une remontée rapide et structurée des incidents tout en s'appuyant sur une base de problèmes standardisés.

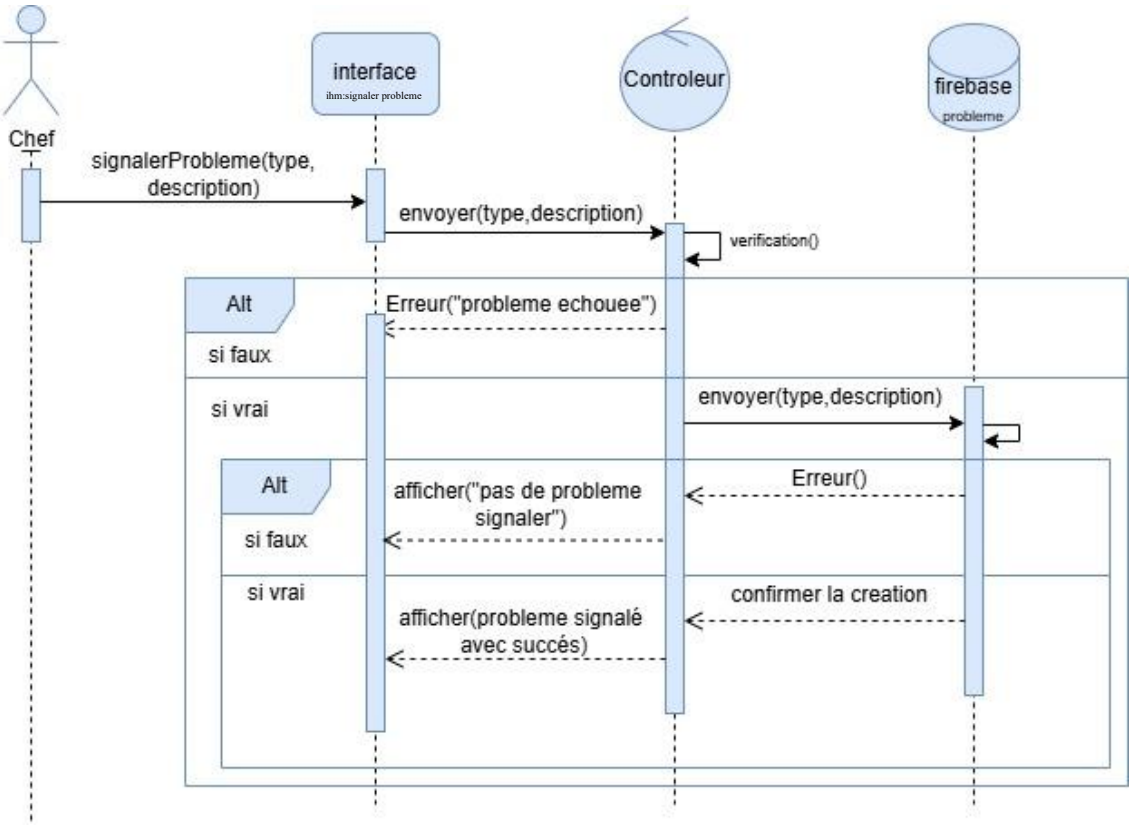



*Figure 27 : Diagramme de sequence signaler un probleme*

### I.2.d. Diagramme de déploiement

Le diagramme de déploiement illustre l'architecture technique du système basé sur une application web développée en React.js pour le front-end, et un backend en Node.js (Express) déployé sur un serveur d'application. Les utilisateurs finaux (administrateur, chef, assistant, technicien) interagissent via le navigateur avec l'interface React, qui communique avec le backend via des requêtes HTTP (API REST). Le backend utilise Firebase pour la gestion des données (Firestore), l'authentification sécurisée des utilisateurs. Firebase Storage peut également être utilisé pour le stockage de fichiers tels que des images liées aux problèmes signalés. Cette architecture assure une séparation claire entre l'interface utilisateur, la logique métier, et la persistance des données, tout en exploitant la scalabilité et la haute disponibilité des services cloud Firebase.

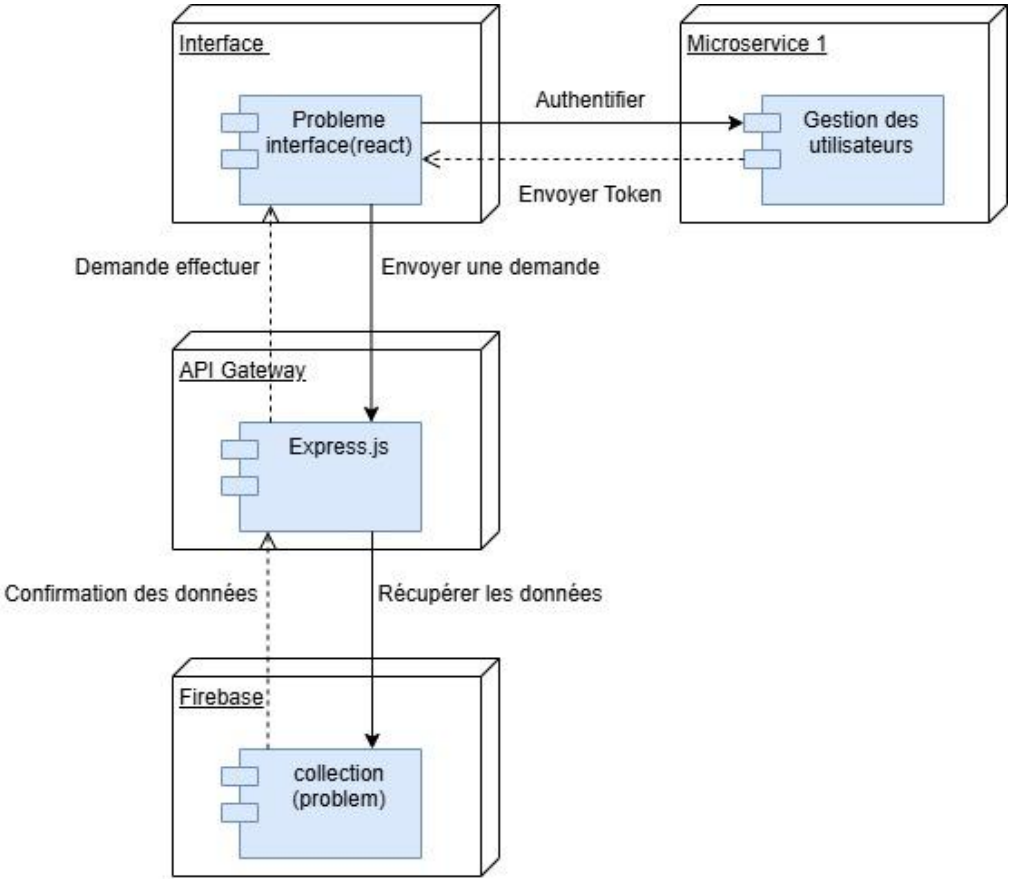



*Figure 28 : Diagramme du déploiement sprint 1*

## I.3. Architecture

Le service de gestion des problèmes est un module clé de notre système microservices. Il permet aux chefs de station de signaler et suivre facilement les incidents techniques, avec des modèles prédéfinis ou personnalisés. L'administrateur gère ces modèles de manière centralisée.
Ce service autonome facilite la communication entre les utilisateurs et les équipes techniques, renforçant l'efficacité du système.

### I.3. a. API Gateway :

- Point d'entrée unique pour toutes les requêtes
- Filtrage et routage vers les services concernés

| Méthode | Route | Description |
|---|---|---|
| POST | /problems | Créer un nouveau problème (non prédéfini) par un chef.<br>Affectation automatique à un technicien disponible. |

| Méthode | Route | Description |
|---|---|---|
| POST | /problems/predefined | Ajouter un problème prédéfini (par l'admin), sans affectation automatique. |
| GET | /problems/predefined | Récupérer la liste de tous les problèmes prédéfinis. |
| GET | /problems/regular | Récupérer tous les problèmes créés par les chefs (non prédéfinis). |
| GET | /problems | Récupérer les problèmes filtrés par chefId ou assignedTechnician. |
| GET | /problems/:id | Obtenir les détails d'un problème via son ID. |
| DELETE | /problems/:id | Supprimer un problème (par l'admin). |

*Table 11 : Routes et Méthodes du scrum 1*

### I.3. b. Firebase Firestore :

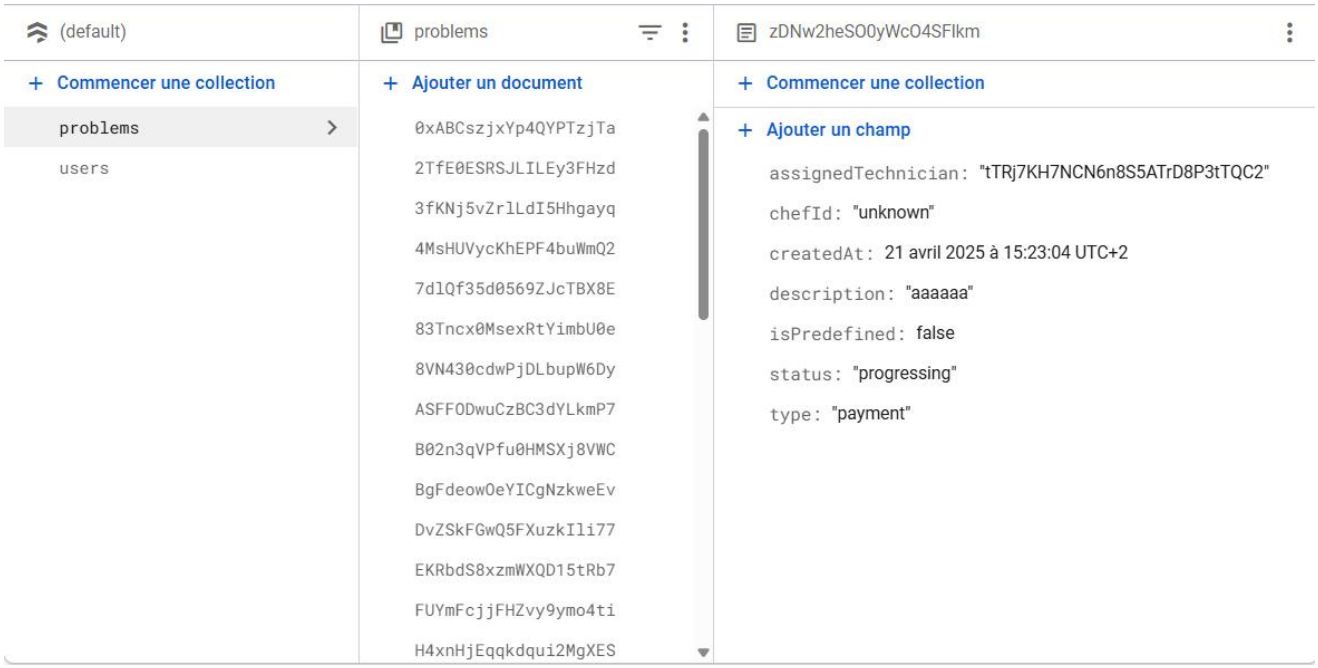



*Figure 29 : Base de données du gestion des problemes*

### I.3. c. Connexion avec d'autres services :

- Service de gestion des utilisateurs
- Service de notifications

# I.4. Implémentation

## I.4.a. Tableau Scrum

- **Premier jour**

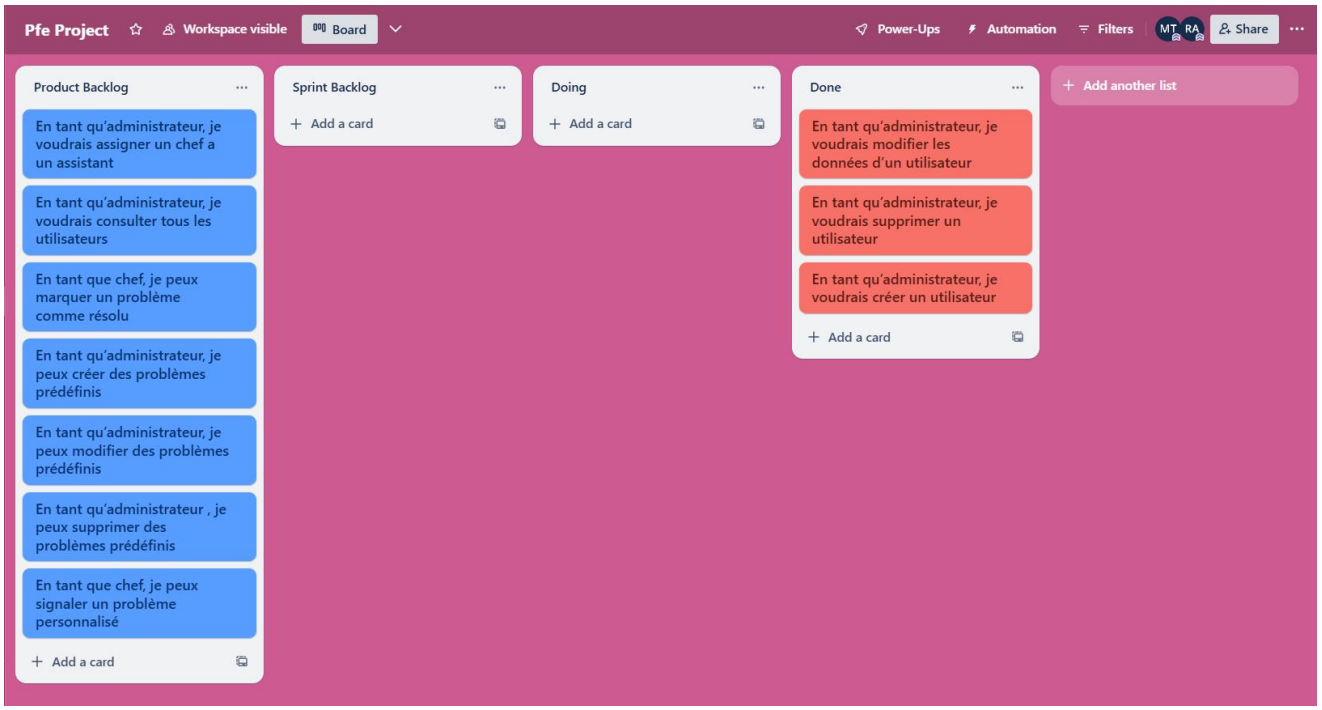



*Figure 30 : Tableau scrum pour le premier jour*

- **Dernier jour**

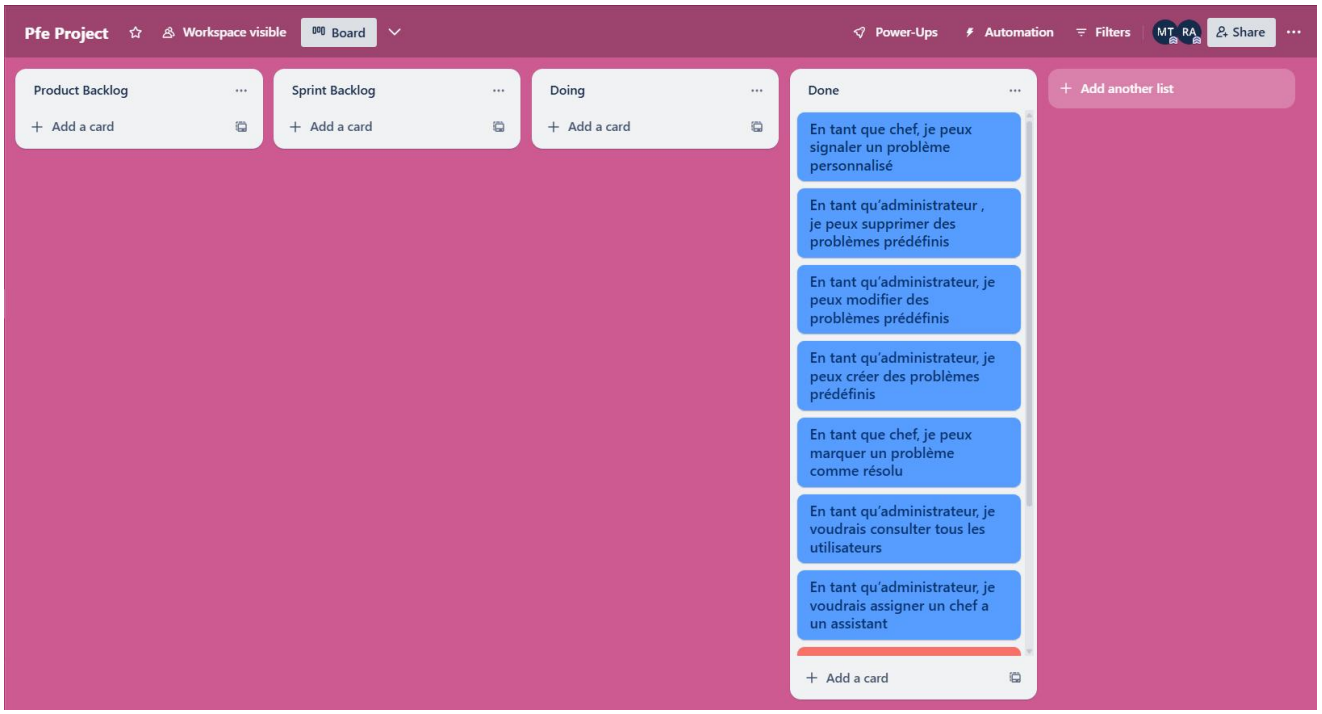



*Figure 31: Tableau scrum pour le dernier jour*

### I.4.b. Interface

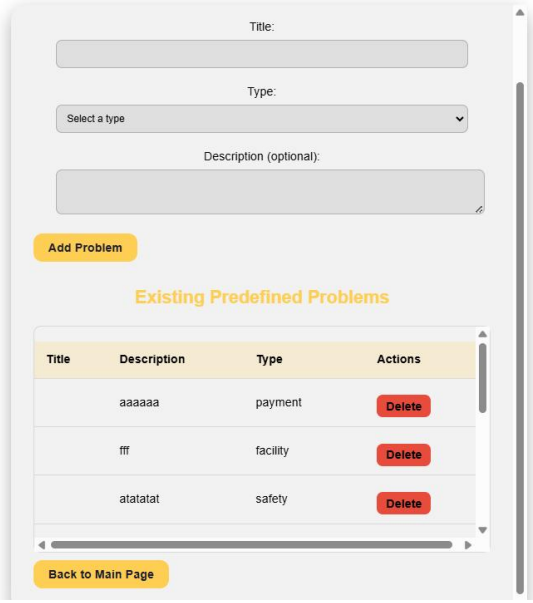



*Figure 32 : Interface de creation d'un problème prédéfini*

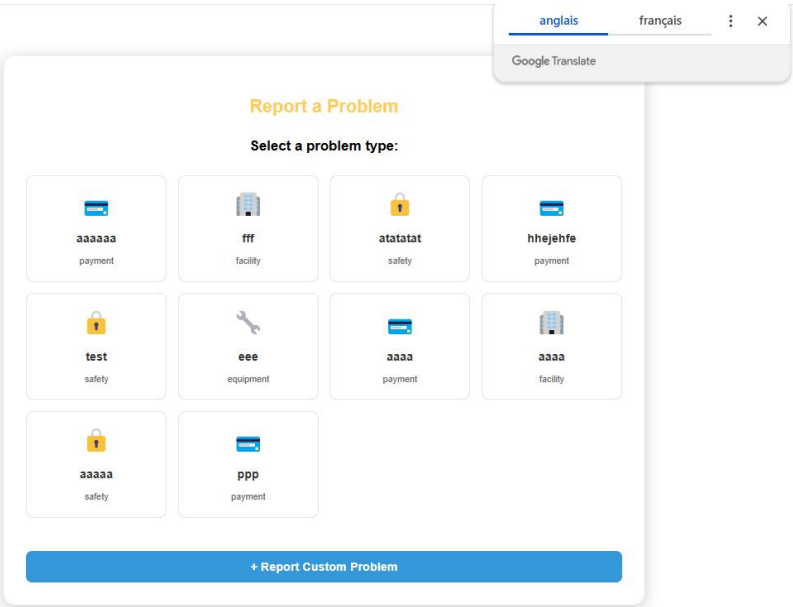
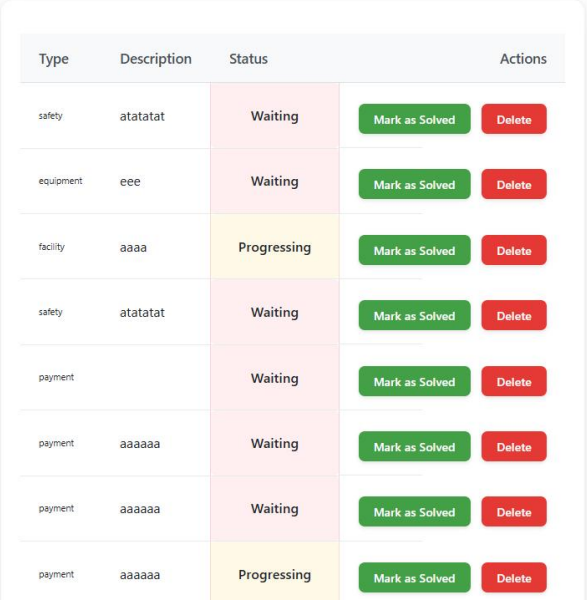



*Figure 33 : Interface d'affectation d'un problème*



| Type | Description | Status | Actions |
|---|---|---|---|
| safety | atatatat | Waiting | Mark as Solved Delete |
| equipment | eee | Waiting | Mark as Solved Delete |
| facility | aaaa | Progressing | Mark as Solved Delete |
| safety | atatatat | Waiting | Mark as Solved Delete |
| payment | | Waiting | Mark as Solved Delete |
| payment | aaaaaa | Waiting | Mark as Solved Delete |
| payment | aaaaaa | Waiting | Mark as Solved Delete |
| payment | aaaaaa | Progressing | Mark as Solved Delete |

*Figure 34 : Interface de consultation des problémes*

## Conclusion

Ce sprint a permis d'implémenter les fonctionnalités clés pour la déclaration, la consultation et la gestion des problèmes. Les chefs peuvent signaler efficacement les incidents, tandis que les administrateurs assurent la gestion des problèmes prédéfinis. L'architecture mise en place garantit sécurité, fiabilité et évolutivité. Ce socle fonctionnel ouvre désormais la voie au sprint suivant, dédié au suivi et à la coordination entre les différents acteurs du système.

# II. Sprint 2 : Suivi et Coordination

Dans ce sprint, nous avons intégré les fonctionnalités de suivi et de coordination entre les utilisateurs, telles que l'attribution automatique des techniciens, la consultation des problèmes, et la supervision des chefs par les assistants. L'objectif est d'améliorer la réactivité et la gestion collaborative des incidents.

## II.1. Backlog du deuxième Sprint

| ID | Histoire utilisateur | Priorité |
|---|---|---|
| 1 | En tant que technicien, je peux consulter les problèmes qui me sont assignés | Haute |

| ID | Histoire utilisateur | Priorité |
|---|---|---|
| 2 | En tant que technicien, je peux accepter un problème | Haute |
| 3 | En tant que système, je peux attribuer automatiquement un technicien selon le type | Haute |
| 4 | En tant qu'assistant, je peux consulter les problèmes des chefs qui me sont assignés | Moyenne |
| 5 | En tant qu'assistant, je peux voir la liste des chefs qui me sont assignés | Moyenne |

*Table 12: Backlog du deuxième sprint*

## II.2. Conception

### II.2.a. Diagramme de Cas d'Utilisation

Le système de gestion des problèmes techniques est conçu pour faciliter le suivi et le traitement des incidents dans les stations de service, en se concentrant sur les rôles du technicien, de l'assistant et du système lui-même. Le technicien a la capacité de consulter les problèmes qui lui ont été assignés, de les accepter pour intervention, puis de les marquer comme résolus une fois le travail effectué. L'assistant, quant à lui, peut visualiser la liste des chefs sous sa supervision ainsi que les problèmes qu'ils ont signalés, ce qui lui permet de suivre l'état d'avancement des interventions sur le terrain. Enfin, le système joue un rôle central en automatisant l'affectation des problèmes aux techniciens, en tenant compte de leur spécialité et de leur disponibilité, afin d'optimiser les délais de traitement et de garantir une gestion efficace des ressources humaines.

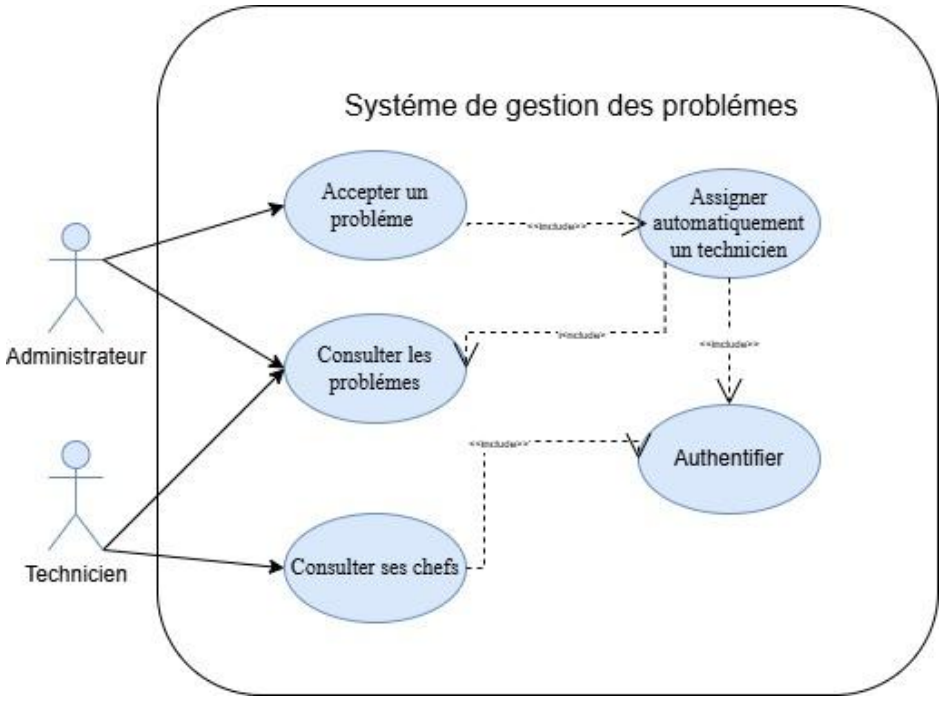



*Figure 35 : Diagramme de cas d'utilisation du sprint 2*

### II.2.b. Diagramme de classes

Le diagramme de classe représente la structure statique du système de gestion des problèmes, en modélisant les principales entités et leurs relations. La classe Problème constitue l'élément central, contenant des attributs tels que l'identifiant, le titre, la description, le type, le statut, la date de création, et l'identifiant du chef ayant signalé le problème. Chaque problème est automatiquement assigné à un Technicien, une classe qui possède des attributs comme l'identifiant, le nom, la spécialité et la disponibilité. La relation entre les deux classes permet d'assurer une affectation ciblée selon les compétences techniques. Les Assistants sont liés à plusieurs Chefs, qu'ils supervisent, et peuvent ainsi accéder indirectement aux problèmes rapportés par ces derniers. Ce diagramme met en évidence les associations entre les entités du système et prépare le terrain pour l'implémentation des règles métier, notamment l'automatisation de l'assignation basée sur la spécialité du technicien.

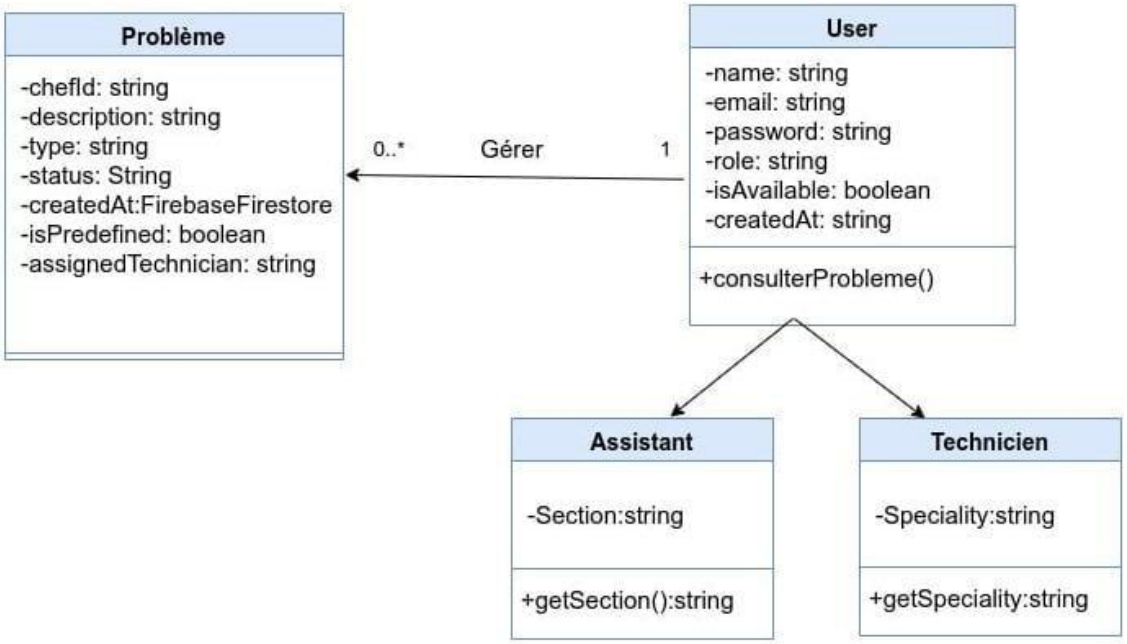



*Figure 36 : Diagramme Gestion des problèmes*

### II.2.c. Diagramme de Séquence

**Consulter les problèmes assignés**

Ce diagramme illustre le processus par lequel un technicien consulte les problèmes qui lui ont été assignés. Lorsqu'il accède à son tableau de bord via l'interface frontend, une requête est envoyée au backend avec son identifiant. Le backend interroge alors la base de données Firestore pour récupérer tous les problèmes liés à cet identifiant de technicien. Une fois les données obtenues, elles sont retournées au frontend, qui les affiche à l'utilisateur. Ce scénario met en évidence l'interaction en lecture entre les composants du système et le rôle du technicien dans le suivi de ses tâches.

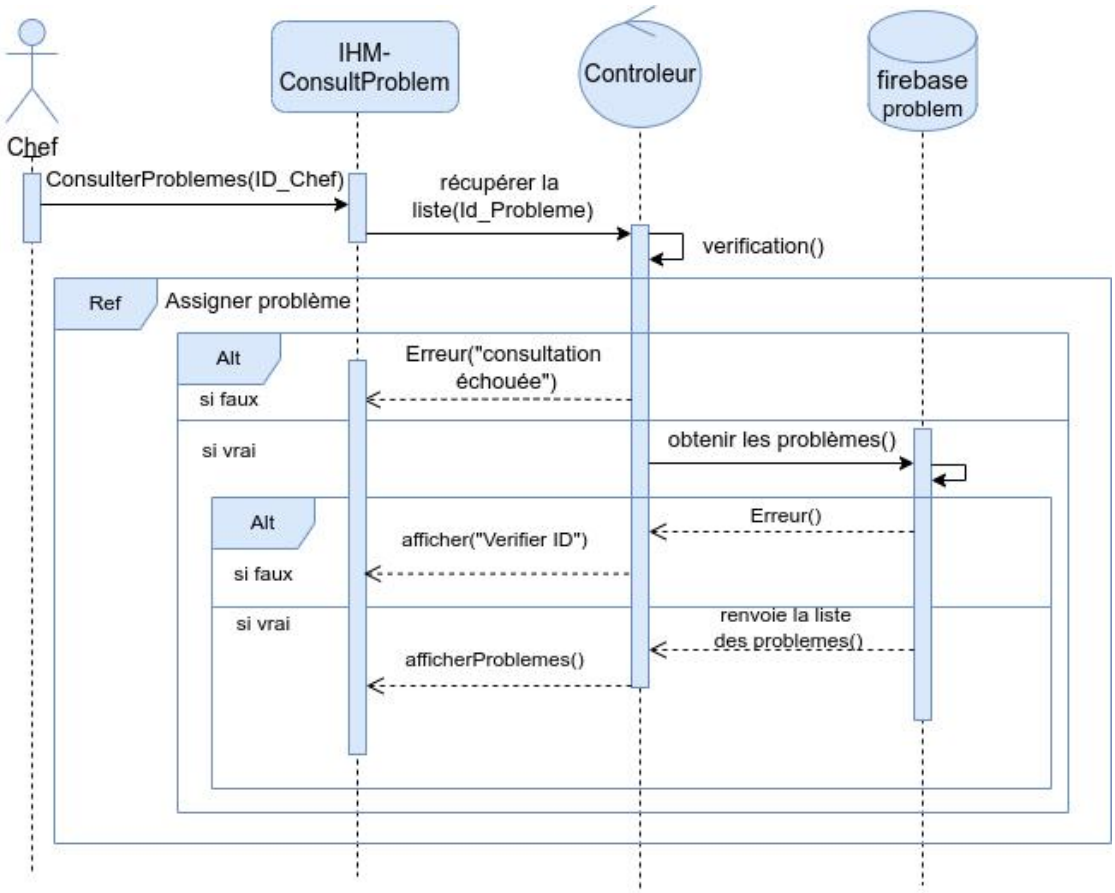



*Figure 37 : Diagramme de consultation d'un problème*

**Accepter un problème**

Ce diagramme décrit l'action qu'effectue un technicien lorsqu'il décide de prendre en charge un problème. À travers l'interface utilisateur, il clique sur un bouton "Accepter", ce qui déclenche une requête PUT vers le backend. Le backend met alors à jour le statut du problème dans Firestore, passant de "Waiting" à "Progress". Une fois la mise à jour confirmée, une réponse de succès est renvoyée à l'interface, qui informe le technicien que le problème est désormais en cours de traitement. Ce scénario reflète un changement d'état dans le cycle de vie d'un problème.

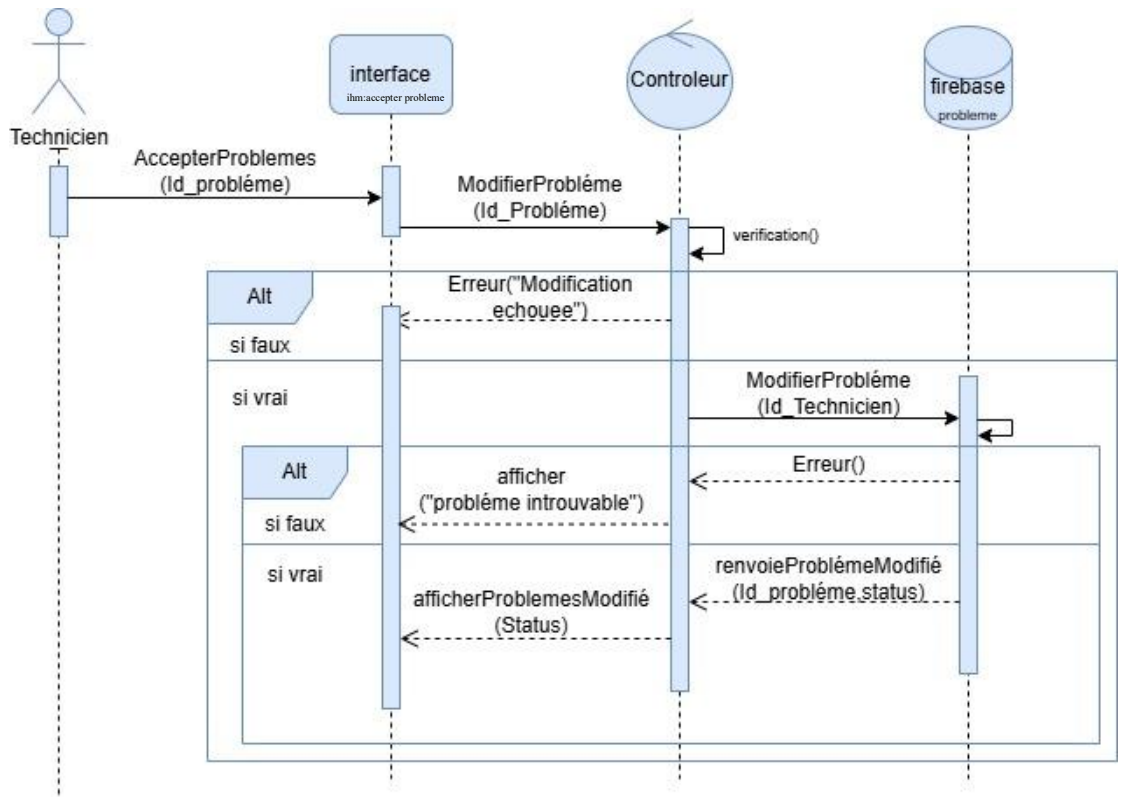



*Figure 38 : Diagramme d'acceptation d'un problème par le technicien*

## II.3. Architecture

### II.3.a. API Gateway :

- Point d'entrée unique pour toutes les requêtes
- Filtrage et routage vers les services concernés

| Méthode | Route | Description |
|---|---|---|
| PUT | /problems/:id | Mettre à jour le statut d'un problème (ex. : en "solved"). Met à jour aussi la disponibilité du technicien. |
| GET | /problems?assignedTechnician=ID | Récupérer la liste des problèmes assignés à un technicien spécifique (vue technicien). |
| GET | /problems?chefId=ID | Obtenir tous les problèmes rapportés par un chef (vue assistant). |

*Table 13 : Routes et Méthodes de l'API Gateway du sprint 2*

### II.2.b. Firebase Firestore :

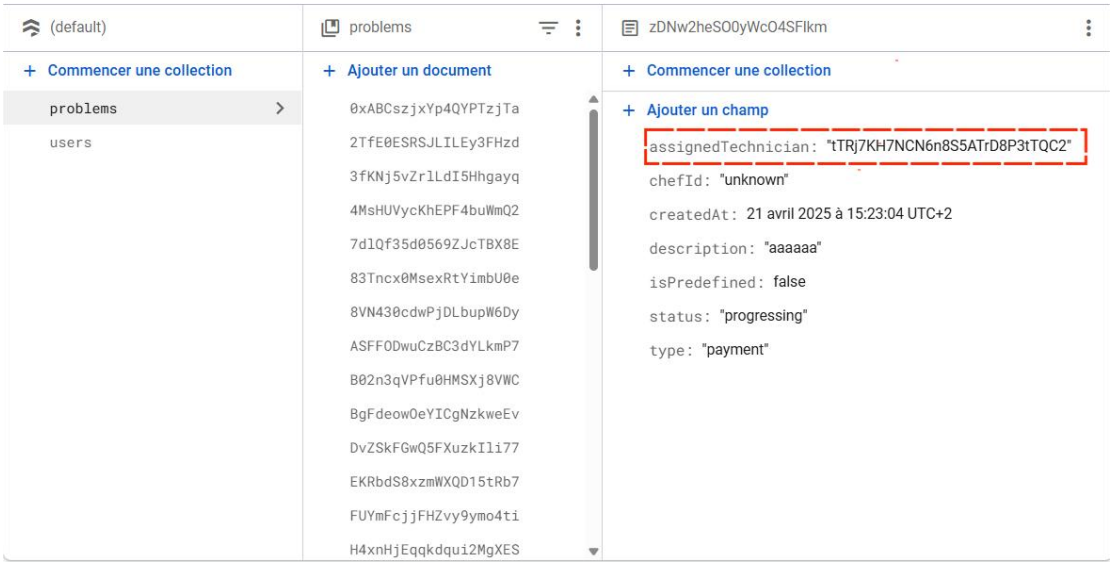



*Figure 40 : base de données sprint2*

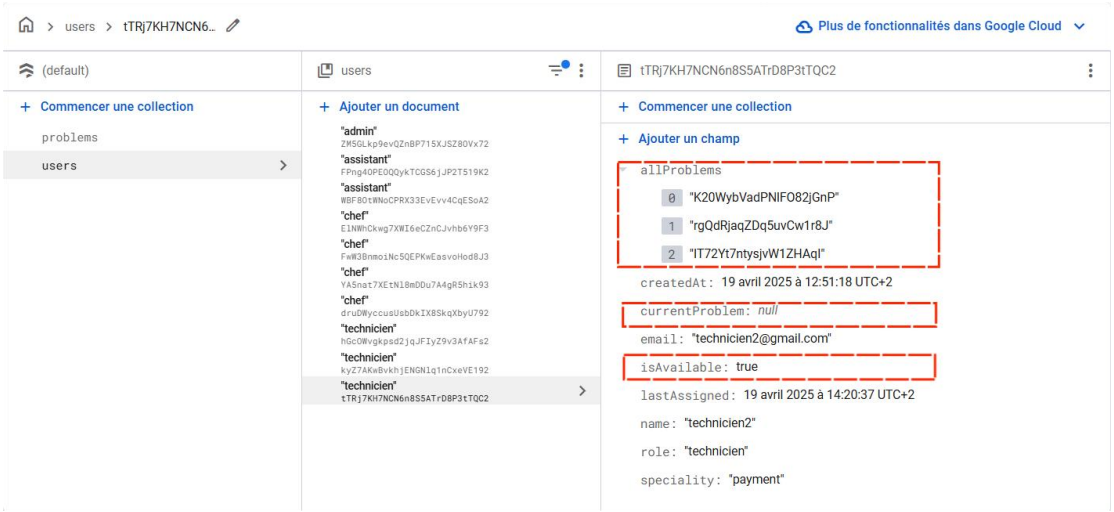



*Figure 41 : base de données sprint2*

### II.3.c. Connexion avec Autres services

- Service de gestion des utilisateurs
- Service de notifications

# II.4.Implémentation

## II.4.a. Tableau Scrum

- **Premier jour**

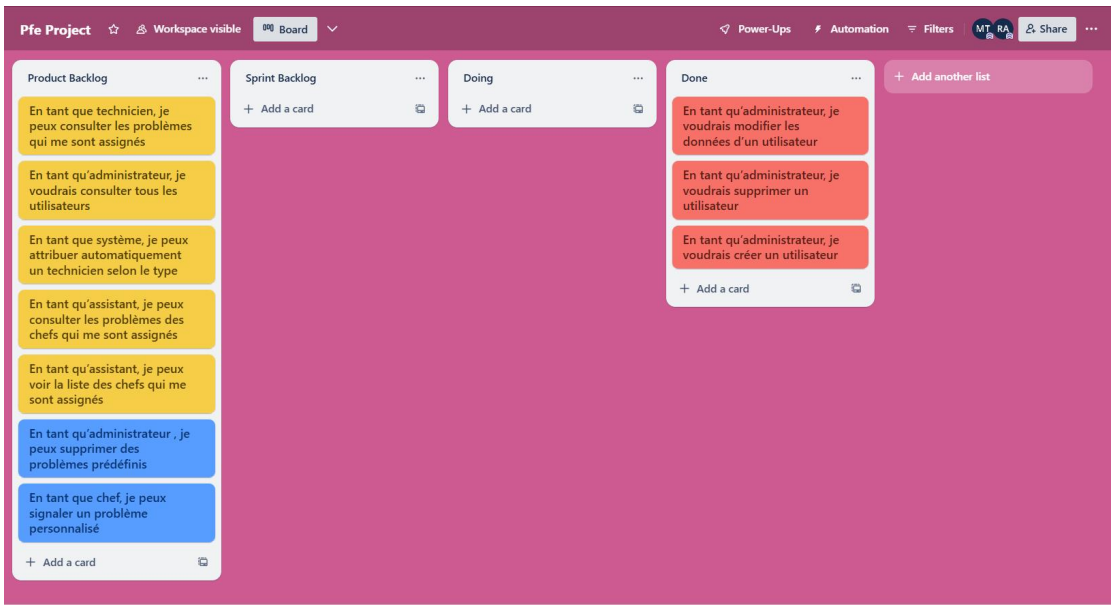



*Figure 42 : Tableau scrum pour le premier jour*

- **Dernier jour**

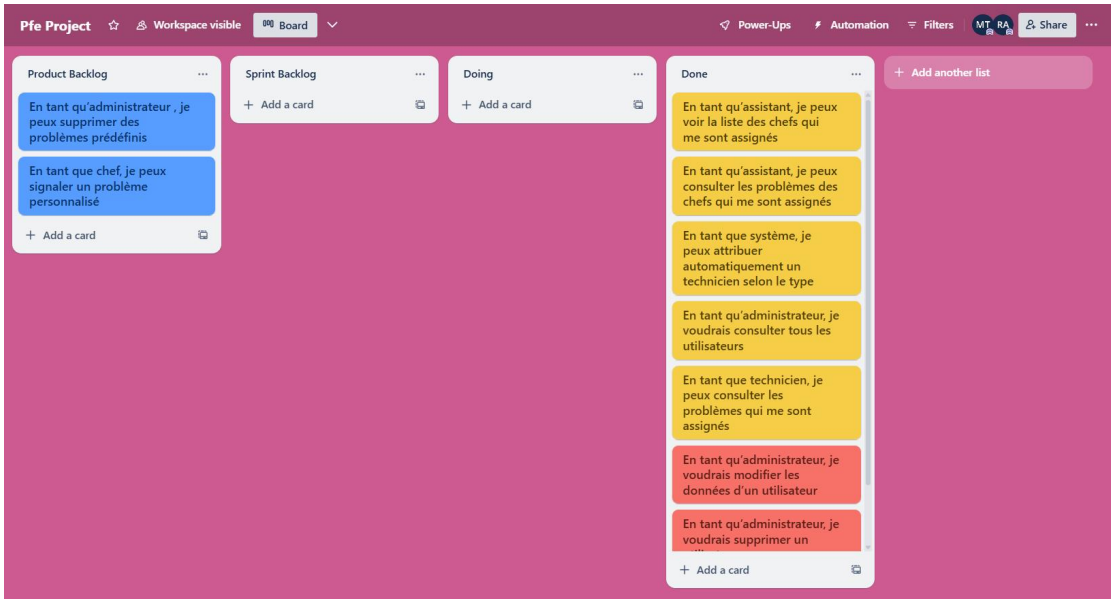



*Figure 43 : Tableau scrum pour le dernier jour*

## II.4.b.Interface

| Type | Description | Status | Actions |
|---|---|---|---|
| equipment | Pump Malfunction | Solved | Delete |
| equipment | Pump Malfunction | Solved | Delete |
| safety | Fuel Spill | Solved | Delete |
| equipment | pomp perdu | Waiting | Mark as Solved Delete |
| safety | Nozzle Stuck | Solved | Delete |
| equipment | Pump Malfunction | Solved | Delete |
| equipment | Pump Malfunction | Solved | Delete |

*Figure 44 : interface des problèmes signaler*

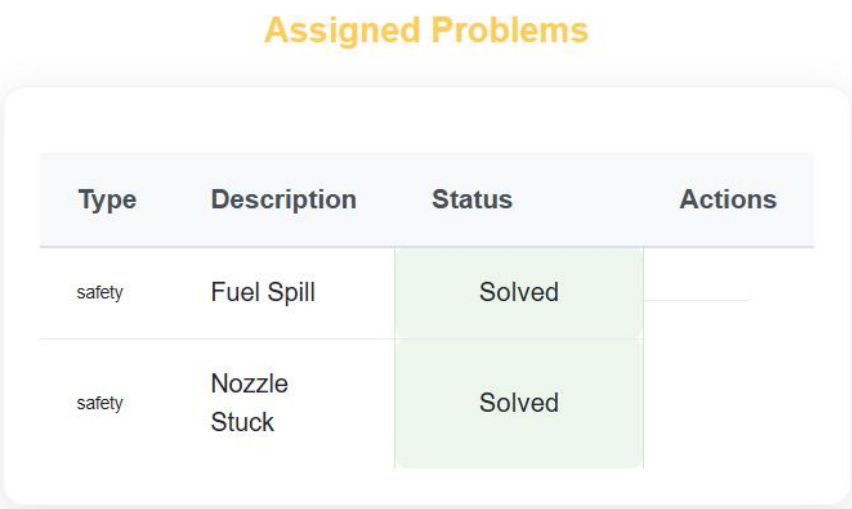



*Figure 45 : interface des problèmes assignées au technicien*

## Conclusion

Les rôles de technicien et d'assistant sont désormais fonctionnels. Le service de problèmes couvre maintenant tout le cycle de vie d'un incident, du signalement à la résolution, en passant par l'attribution et le suivi.

# Chapitre 6

## Release 3: Gestion des Notifications

**PLAN**

**Introduction**

**I. Sprint 1 : Dashboard et Notifications**

**II. Sprint 2 : ChatBot AI**

**Conclusion**

## Introduction

Ce microservice fait partie d'un système distribué basé sur une architecture microservices. Il a pour objectif de fournir aux administrateurs une vue globale du système à travers un tableau de bord interactif, tout en assurant l'envoi de notifications par email de manière automatique.

# I. Sprint 1 : Dashboard et Notifications

Ce sprint initie le développement du microservice de supervision, en fournissant à l'administrateur un tableau de bord centralisé et des notifications automatisées pour un suivi en temps réel du système.

## I.1.Backlog du premier Sprint

| ID | Histoire utilisateur | Priorité |
|---|---|---|
| 1 | En tant que Administrateur , je peux accéder au tableau de board | Haute |
| 2 | En tant que Administrateur , je peux consulter les statistiques des utilisateurs | Haute |
| 3 | En tant que Administrateur , je peux consulter les statistique des problèmes | Haute |
| 4 | En tant que Administrateur , je peux consulter l'état du micoservices | Haute |
| 5 | En tant que Administrateur , je peux consulter la l'état des stations | Moyenne |

*Table 14: Backlog du microservice Gestion des notification*

## I.2. Conception

### I.2.a. Diagramme de Cas d'Utilisation

Le diagramme de cas d'utilisation représente les différentes fonctionnalités accessibles à l'administrateur dans le cadre du microservice de Notifications et Dashboard. L'acteur principal, l'Administrateur, interagit avec le système en accédant au dashboard principal. Une fois connecté, il peut exécuter plusieurs cas d'utilisation spécifiques :

- Consulter les statistiques des utilisateurs, afin de visualiser des données comme le nombre total d'utilisateurs, leur répartition par rôle et leur disponibilité ;
- Consulter les statistiques des problèmes, permettant de suivre les incidents signalés, leur état (en attente, en cours, résolus) et leur type ;
- Consulter l'état des microservices, pour vérifier si les différents services du système (utilisateurs, problèmes, notifications) sont fonctionnels ;
- Consulter l'état des stations, déduit à partir des incidents signalés, permettant d'avoir une vision en temps réel de l'état des stations sur le réseau.

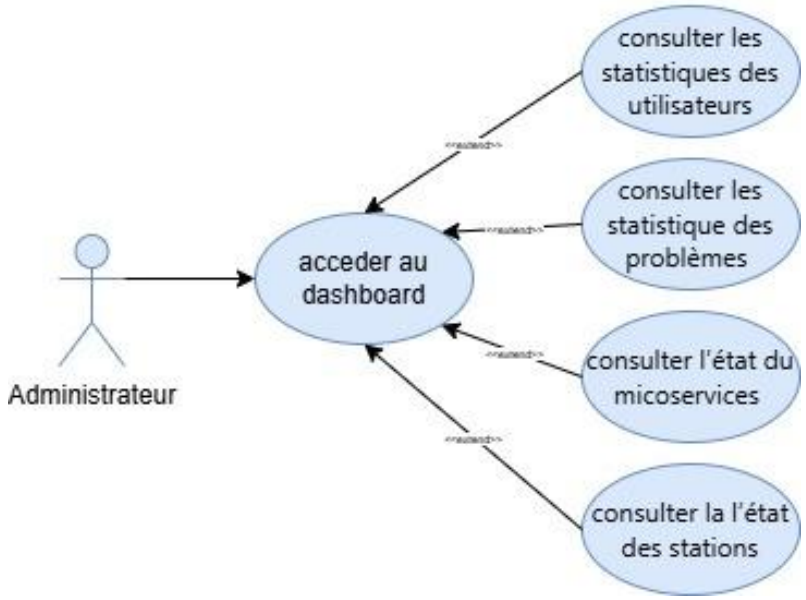



*Figure 46: Diagramme de cas d'utilisation*

| | |
|---|---|
| **Résumé** | L'administrateur accède au tableau de bord pour consulter diverses statistiques et états liés au système. |
| **Acteurs** | Administrateur |
| **Préconditions** | - L'administrateur est authentifié- Le token d'accès est valide |
| **Postconditions** | - Le tableau de bord est affiché avec les informations mises à jour |
| **Scénario nominal** | 1.L'administrateur demande à accéder au dashboard.<br>2.Le système vérifie le token.<br>3.Le système valide que l'utilisateur est un administrateur.<br>4. Le système affiche les statistiques des utilisateurs, des problèmes, l'état des microservices et celui des stations. |
| **Scénarios alternatifs** | - Token invalide → message "Validation échouée"- Rôle non administrateur → "Accès refusé"- Erreur de récupération des données → messages d'erreur correspondants |
| **Exigences non** | - Réactivité de l'interface en moins de 2 secondes- Sécurité |

| | |
|---|---|
| **fonctionnelles (NFN)** | (JWT, HTTPS)- Disponibilité 24/7- Tolérance aux pannes (retries, fallback)- Visualisation responsive (tablette, PC) |

*Table 15:Tableau récapitulatif Gestion des notification*

### I.2.a. Diagramme de Séquence

- **Accéder au dashboard**

Ce diagramme de séquence décrit le processus d'accès au dashboard par un administrateur. Le scénario débute lorsque l'Administrateur initie la requête Accéder au dashboard(id) via l'interface. L'interface transmet cette requête au contrôleur en appelant la fonction verifier-Token(id), afin de s'assurer que l'utilisateur est bien authentifié.

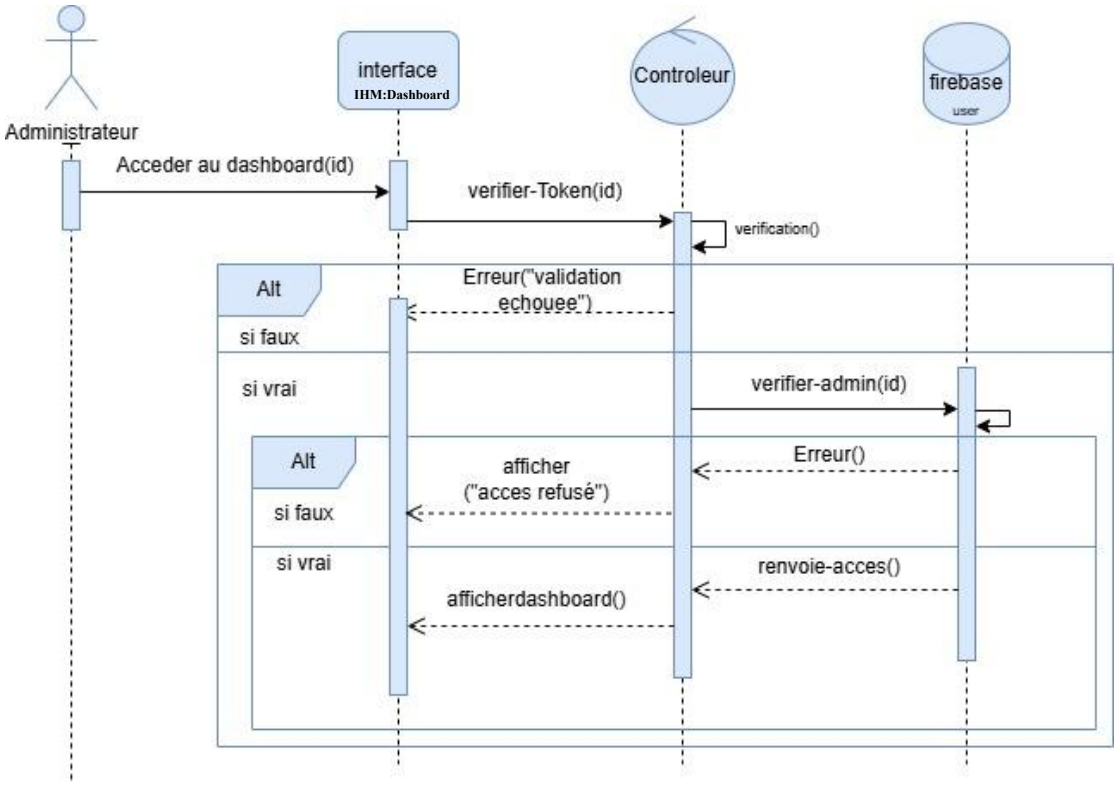



*Figure 47: Diagramme de séquence d'accédation au dashboard*

- **Consulter l'état des microservice**

Ce diagramme de séquence illustre le processus par lequel un Administrateur peut consulter l'état de fonctionnement d'un microservice via une interface. Le scénario commence lorsque l'administrateur déclenche l'action consulter-etat-microservice() à travers l'interface utilisateur.

L'interface envoie une requête recuperer-etat() au contrôleur, qui à son tour contacte le service externe API Health via la méthode verification(). Cette dernière interroge l'état actuel du microservice concerné à l'aide de la fonction retourner-etat().

Si la récupération échoue (par exemple, si l'API Health ne répond pas), une erreur "récupération échouée" est renvoyée et l'interface ne peut afficher l'état. Si la récupération réussit, une deuxième condition vérifie l'état retourné :

- Si le microservice est en panne, l'interface affiche le message "microservice en panne"
- Si le microservice est fonctionnel, l'interface affiche "microservice en marche".

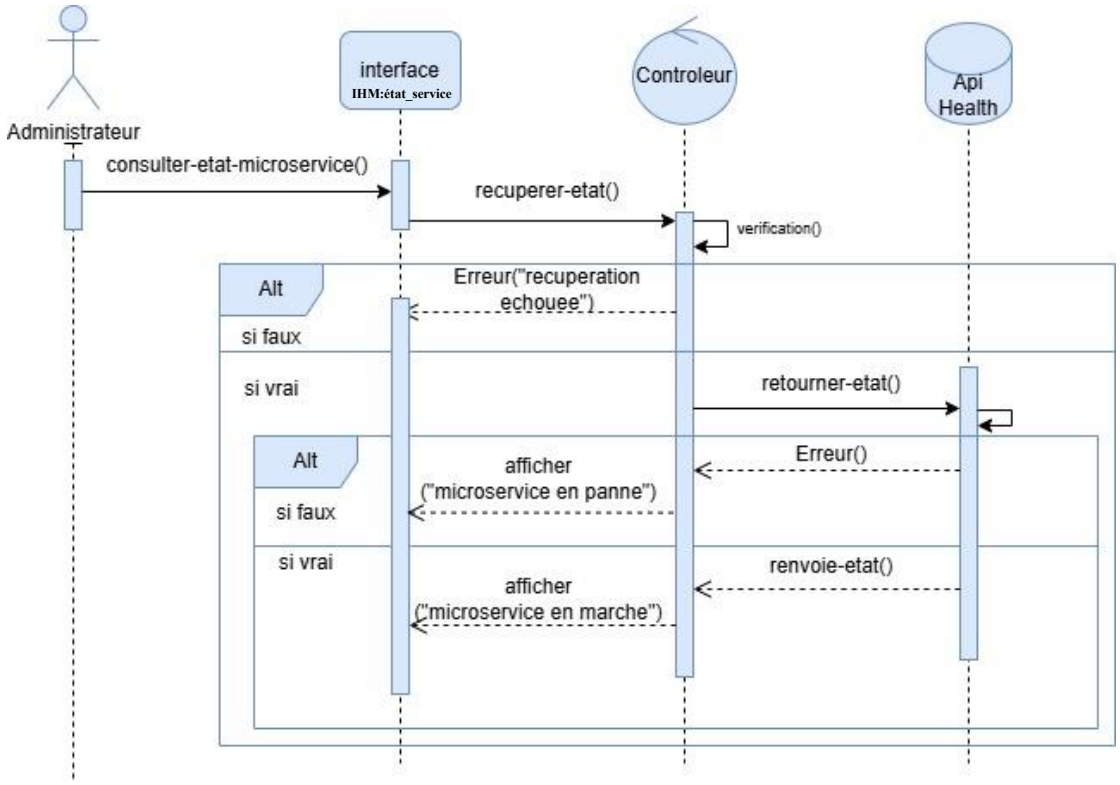



*Figure 48: Diagramme de séquence de consultation etat microservice*

## I.3. Architecture

### I.3.a. API Gateway :

- Point d'entrée unique pour toutes les requêtes
- Filtrage et routage vers les services concernés

| Méthode | Route | Description |
|---|---|---|
| GET | /api/dashboard | Récupère toutes les statistiques agrégées : utilisateurs, problèmes, état des services. |
| GET | /api/problems | Récupère tous les problèmes avec les noms des utilisateurs et techniciens associés (fusionne avec User Service). |
| POST | /send-email/ | Envoie un e-mail personnalisé |
| POST | /notify/email/ | Envoie un e-mail de notification standard |

*Table 16:Routes et Méthodes de l'API Gateway*

### I.3.b. Connexion avec d'autres services :

- Service gestion des utilisateurs
- Service gestion gestion des problémes

## I.4. Implémentation

### I.4.a Tableau Scrum

- **Premier jour**

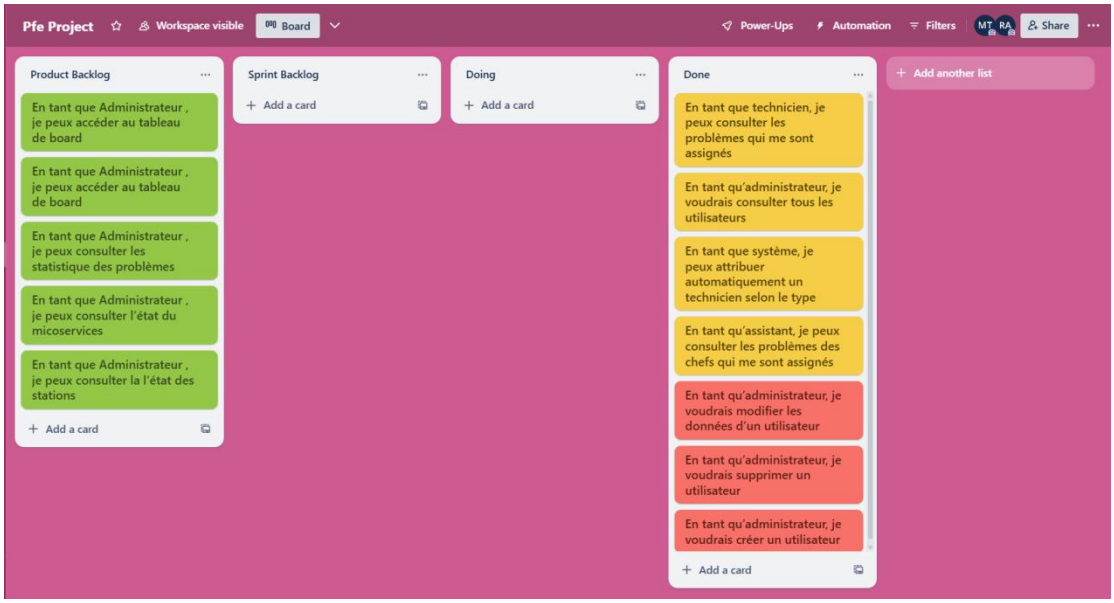



*Table 17: Tableau scrum du premier jour*

- **Dernier jour**

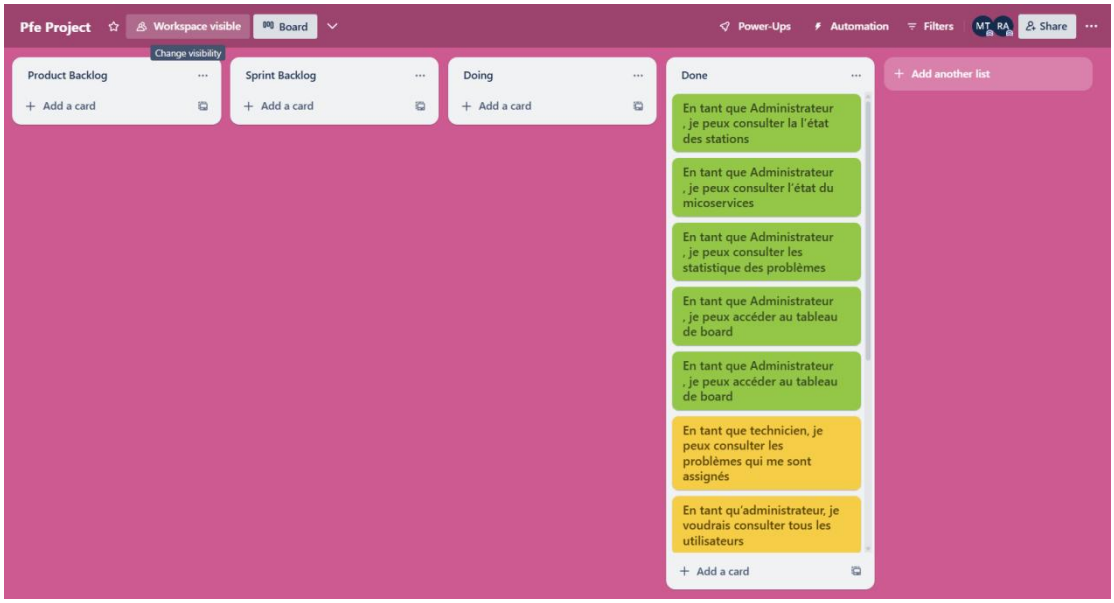



*Table 18: Tableau scrum du dernier jour*

## I.4.b. Interface

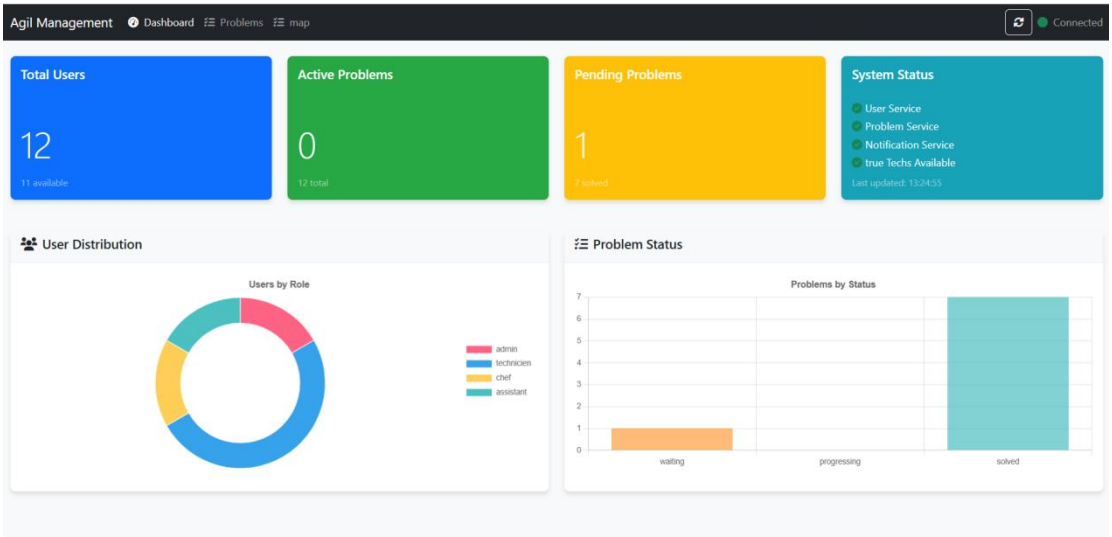



*Figure 49: Interface Dashboard*

All Problems

Refresh

| ID | Description | Type | Status | Chef | Assigned To | Created At | Actions |
|---|---|---|---|---|---|---|---|
| qpKSzvcN | Pump Malfunction | equipment | solved | chef1... | rawen11... | 02/05/2025 16:01:40 | View |
| yPoPeNdM | pomp perdu | equipment | solved | chef4... | rawen11... | 29/04/2025 14:34:50 | View |
| SYFh66BW | pomp perdu | equipment | waiting | Unknown ... | Unassign... | 29/04/2025 14:33:34 | View Assign |
| B67gFBV5 | Pump Malfunction | equipment | solved | chef1... | Unassign... | 28/04/2025 16:01:05 | View |
| r1c0xv3i | Pump Malfunction | equipment | solved | chef1... | Unassign... | 28/04/2025 15:58:43 | View |
| xvDaEUtS | Pump Malfunction | equipment | solved | chef1... | Unassign... | 28/04/2025 15:51:34 | View |
| ePG22fjF | Fuel Spill | safety | solved | chef1... | firas... | 28/04/2025 15:44:56 | View |
| mLhBCR5Y | Nozzle Stuck | safety | solved | chef1... | firas... | 28/04/2025 15:44:52 | View |
| Gl0YXf8c | Nozzle Stuck | safety | | Unknown ... | Unassign... | 28/04/2025 15:35:21 | View |
| NqINRUYi | Fuel Spill | safety | | Unknown ... | Unassign... | 28/04/2025 15:35:04 | View |
| EFTCt96M | Pump Malfunction | equipment | | Unknown ... | Unassign... | 28/04/2025 15:34:44 | View |
| EKRbdS8x | hhejehfe | payment | | Unknown ... | Unassign... | 18/04/2025 12:06:58 | View |

*Figure 50: Interface de tous les problème*

**Tunisian Gas Station Status**

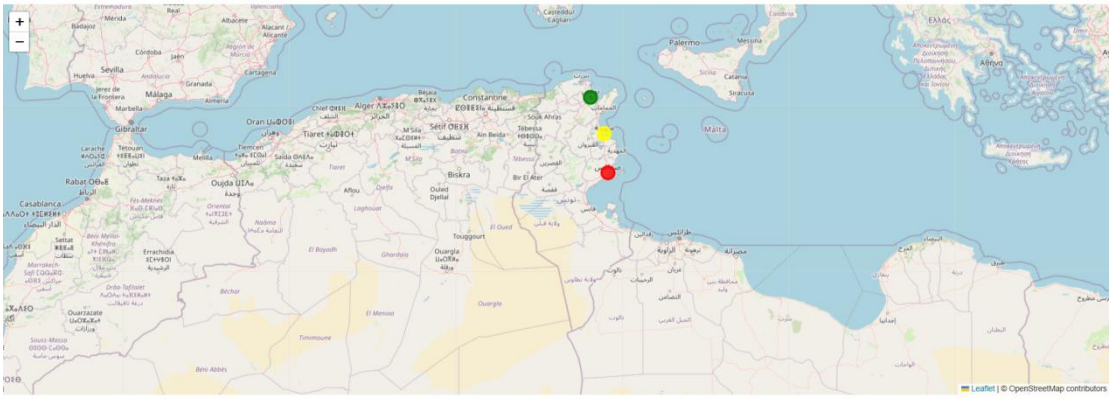



*Figure 51: Interface Localisation*

## Conclusion

Le microservice de supervision joue un rôle essentiel dans l'écosystème global du système multi-microservices. Il permet à l'administrateur d'avoir une vue centralisée et en temps réel sur l'état des utilisateurs, des incidents signalés, des stations-service ainsi que la santé des autres microservices Grâce à son tableau de bord interactif.

# II. Sprint 2 : ChatBot AI

## Introduction

Dans ce sprint, nous avons développé un Chatbot IA intelligent capable d'interagir avec des données structurées liées aux stations-service. Basé sur le modèle LLaMA3 (8B) via Ollama et intégré avec l'agent ReAct, ce chatbot comprend les requêtes en langage naturel, sélectionne automatiquement l'outil approprié selon le besoin de l'utilisateur, et répond avec précision. Il peut à la fois interroger des données à partir d'un fichier CSV, et enregistrer des notes sur demande explicite. Cette solution offre une interface conversationnelle puissante et intuitive pour explorer et exploiter efficacement les données disponibles.

### I.1.Backlog de deuxième Sprint

| ID | Histoire utilisateur | Priorité |
|---|---|---|
| 1 | En tant qu' Administrateur, je veux poser des questions sur les stations. | Haute |
| 2 | En tant qu'Administrateur, je veux charger les données CSV | Haute |

### I.2. Conception

#### I.2.a. Diagramme de Cas d'Utilisation

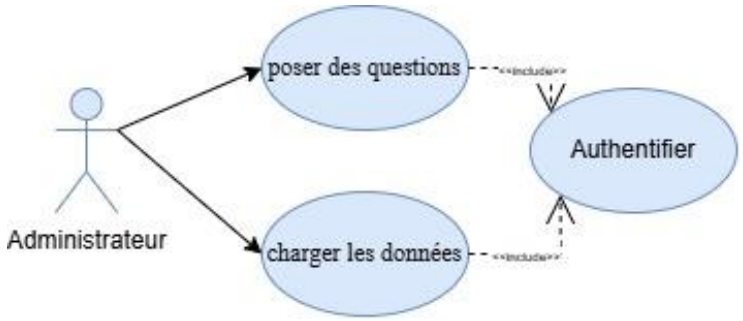



*Figure 52 : Diagramme de Cas d'Utilisation chatbot*

### I.2.b. Diagramme de classe

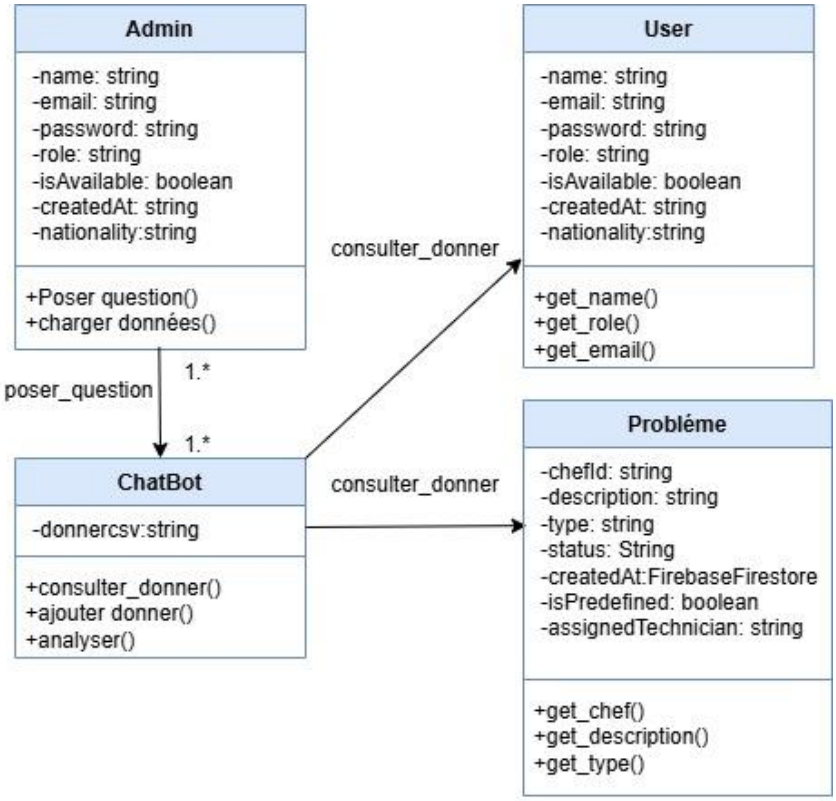



*Figure 53 : Diagramme de classe chatbot*

## I.3. Architecture

### I.3.a. API Gateway :

| Méthode | Route | Description |
|---|---|---|
| GET | /chatbot/query | Chatbot ia service |
| POST | /problem/report | Problem service |

*Table 19 : Api gateway du chatbot*

### I.3.b. Connexion avec d'autres services :

Dans ce sprint, le chatbot interagit directement avec la base de données structurée, représentée sous forme de fichier CSV chargé dans un moteur de requêtes. Grâce à cette intégration, l'utilisateur peut poser des questions en langage naturel, et le système interroge instantanément les données sans passer par une couche intermédiaire, garantissant des réponses rapides, précises et à jour.

## I.4. Implémentation

### I.4.a Tableau Scrum

- **Premier jour**

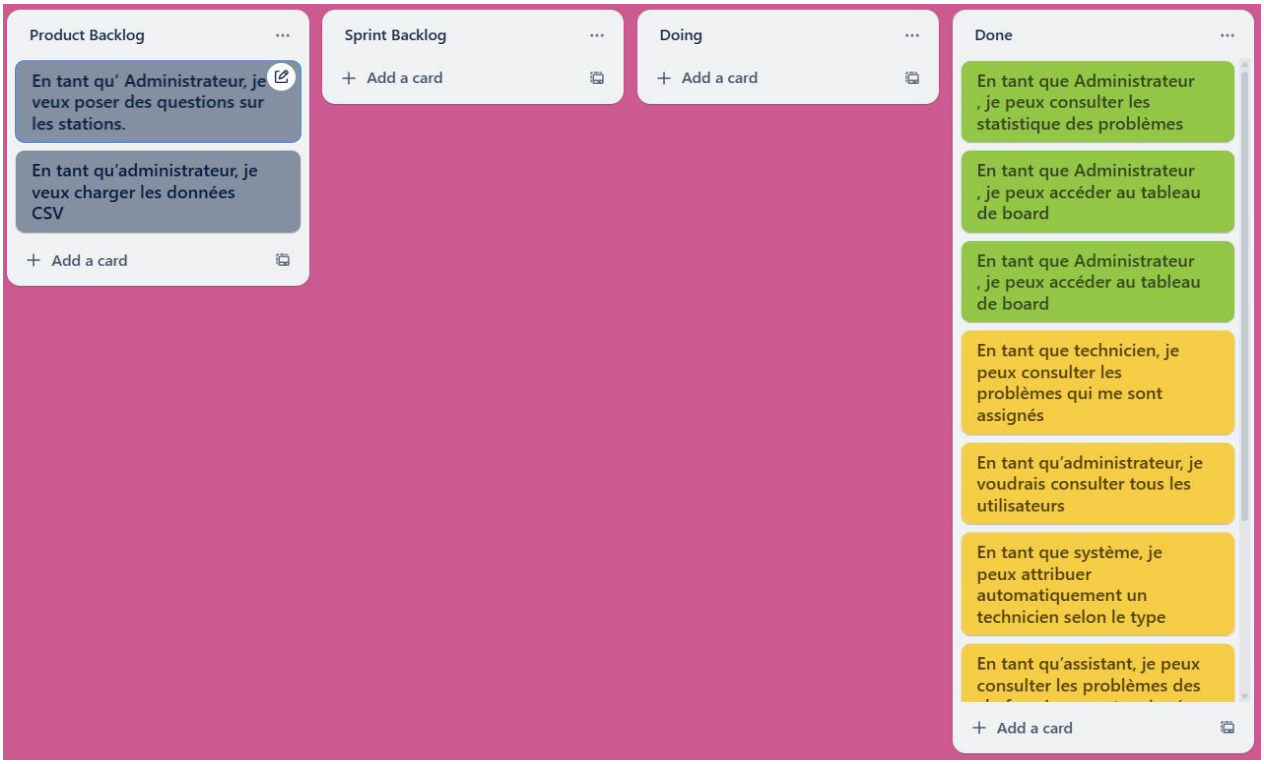



*Table 20 : tableau scrum du premier jour chatbot*

- **Dernier jour**

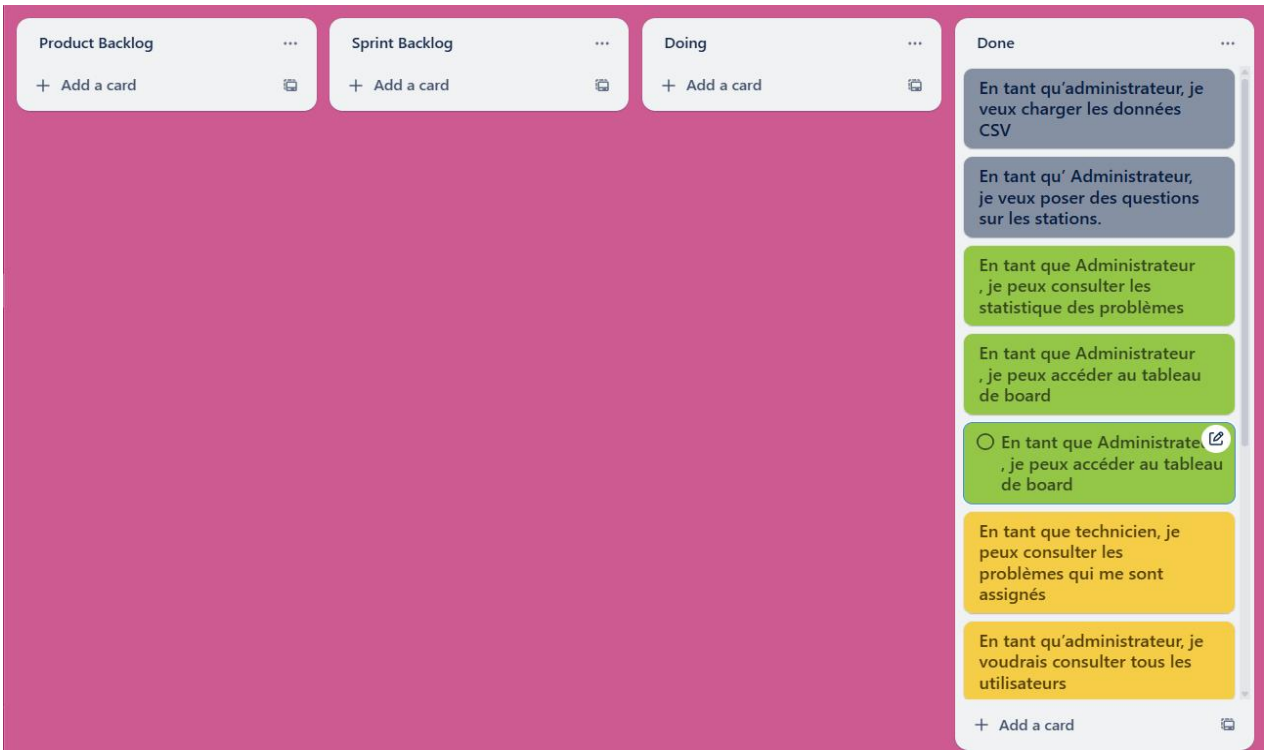



*Table 21 : tableau scrum du dernier jour chatbot*

### I.4.b. Interface

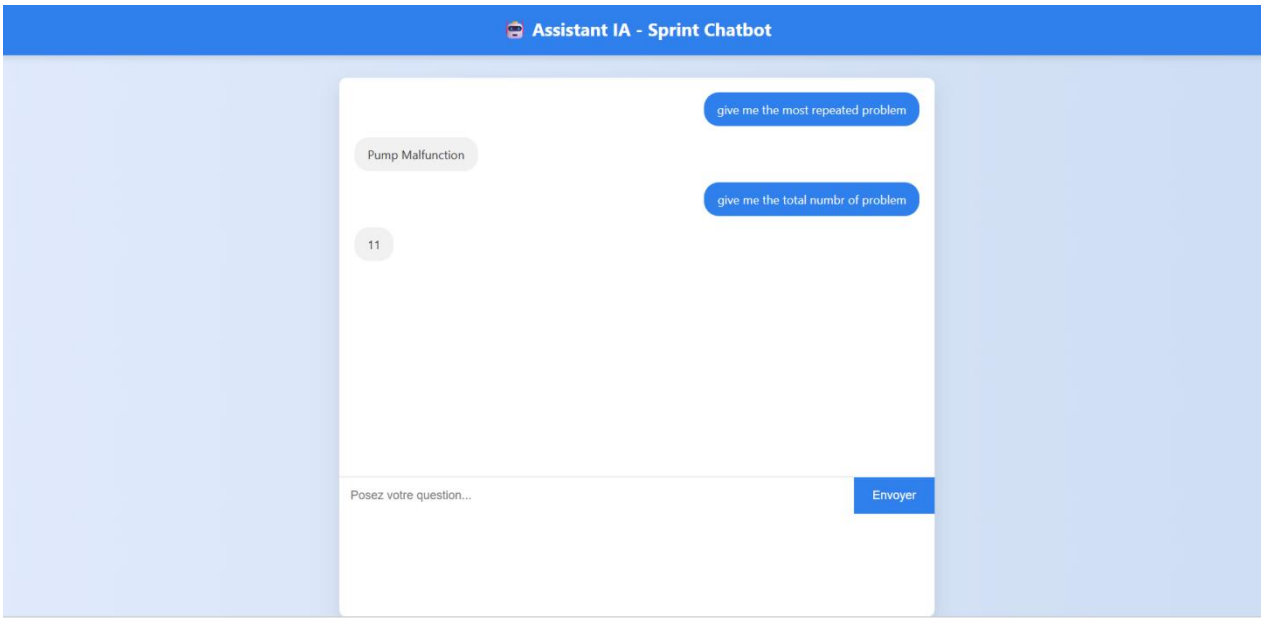



*Figure 54 : interface du chatbot*

## Conclusion

En conclusion, ce sprint a permis la mise en place d'un Chatbot IA intelligent capable d'interagir directement avec des données structurées via un moteur de requêtes performant. Grâce à l'intégration du modèle LLaMA3 et de l'agent ReAct

# Conclusion Générale

Ce projet a permis de concevoir une application web complète et moderne, fondée sur une architecture microservices. Grâce à des technologies variées et complémentaires telles que React.js, Node.js, FastAPI, Firebase, et l'intégration d'une intelligence artificielle via LlamaIndex, l'application répond aux besoins spécifiques de gestion, de traitement de problèmes, et de communication intelligente dans un environnement distribué. Chaque microservice a été pensé de manière autonome, scalable et facile à maintenir. De plus, le déploiement via des plateformes comme Netlify, Render, ou Vercel, ainsi que l'utilisation de GitHub pour le versionnage, ont renforcé la fiabilité et la portabilité du système. Ce travail marque une étape importante dans la maîtrise des technologies modernes du développement logiciel, tout en proposant une solution concrète et efficace pour des besoins réels.

Le suivi d'une méthodologie de travail bien structurée, ainsi que l'utilisation de techniques innovantes et en constante évolution, figurent parmi les acquis majeurs de ce stage. Il a permis la mise en pratique et le renforcement des connaissances académiques acquises à l'Institut Supérieur des Études Technologiques de Zaghouan, tout en les enrichissant grâce à une application concrète. Au-delà des compétences techniques, ce stage chez AGIL a représenté une véritable initiation à la vie professionnelle dans un environnement réel.

# References

https://firebase.google.com/docs

https://fastapi.tiangolo.com/

https://reactjs.org/docs/getting-started.html

https://nodejs.org/en/docs

https://owasp.org/www-project-json-web-token/

https://www.youtube.com/watch?v=0iB5IPoTDts

https://www.youtube.com/watch?v=GWB9ApTPTv4

https://www.youtube.com/watch?v=Cy9fAvsXGZA&t=209s

https://github.com/ollama/ollama/tree/main/docs

https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction

· Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

· Richardson, C. (2018). *Microservices Patterns: With examples in Java*. Manning Publications.

· Fowler, M., & Lewis, J. (2014). *Microservices – a definition of this new architectural term*. martinfowler.com.

· Pautasso, C., Zimmermann, O., & Leymann, F. (2017). *RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision*. Springer.