ARCHITECTURE DEFINITION DOCUMENT

• Objet du document

Ce document permet de modéliser l'architecture qui sera réalisée par les équipes de développement. Au delà de la modélisation graphique, il s'agit également d'énoncer les principes sur lesquels l'architecture s'appuie, de justifier cette approche architecturale, mais également d'indiquer des transitions si nécessaire.

• Objectifs du projet

L'objectif du projet est de créer une application unique regroupant les fonctionnalités de plusieurs applications web actuellement utilisées par les clients et qui ne répondent plus pleinement à leurs besoins. Cette application doit permettre de :

- Créer et gérer leur compte
- Consulter les offres de location disponibles correspondant à des critères détaillés
- Réserver un véhicule
- Effectuer le paiement en ligne
- Consulter, modifier, ou annuler une réservation
- Communiquer avec le support en direct ou de manière asynchrone
- Recevoir des notifications

L'application devra être scalable en vue d'un déploiement international, et doit pouvoir communiquer avec les différentes applications internes par le biais d'une API permettant de consulter et de modifier des données.

• Principes d'architecture

L'architecture de cette nouvelle application sera basée sur différents principes choisis afin de répondre au mieux aux exigences de l'application.

Tout d'abord, on choisira une **architecture client/serveur en couches**, avec un **back-end en Java 21 avec Spring (6.2.8)** comprenant une **API REST** et une **API WebSocket**, et un **front-end en Angular (19.2.15)**, afin d'avoir une application **robuste**, **scalable**, **simple à maintenir**, et **sécurisée**. De plus, cette architecture pourra permettre d'**interfacer facilement la nouvelle application avec les applications existantes** utilisées en agence.

L'application sera placée dans un **conteneur Docker**, et hébergé sur un **cloud AWS** afin d'être **le plus scalable** et **résiliente** possible. Côté sécurité, l'**authentification sera basée sur des rôles**, et effectuée à l'aide d'un **token JWT**. La communication entre le backend et le frontend se fera via **HTTPS**.

Enfin, on s'attachera à ce que l'application soit conforme au **RGPD** en ne gardant que les données strictement nécessaires, et en supprimant le données des utilisateurs lorsque ceux-ci suppriment leurs comptes.

• Architecture existante

Le projet ne s'appuie pas sur un produit existant qui serait repris puis mis à jour. Il n'y a donc pas d'architecture existante à définir.

• Architectures

Métier

Voir annexe 1 – Diagramme des cas d'utilisation
Voir annexe 2 – Diagramme d'activité

De données

Voir annexe 3 – Schéma de base de données relationnelle

Technologiques

Voir annexe 4 – Diagramme de composants

Voir annexe 5 – Diagramme de déploiement

• Justification de l'approche architecturale

Étant destiné à une utilisation mondiale et un fort trafic, le projet se doit d'adopter une **architecture robuste**, **modulaire**, et **maintenable**. Grâce à l'**architecture client-serveur en couches** choisie ici, la modularité, l'évolutivité, et l'intégration avec les systèmes internes des agences, et des services externes (via des **APIs REST**), se trouvent facilitées. Cette structure facilite aussi la **séparation des responsabilités**, réduisant ainsi la complexité et la dépendance entre les composants, et améliorant la maintenabilité du code.

Pour la partie messagerie en temps réel, l'usage du protocole **WebSocket** permet une **communication bidirectionnelle** persistante avec une **faible latence**. Celle-ci est indispensable pour garantir une **expérience utilisateur fluide et réactive** lors de conversations asynchrones entre les utilisateurs et le support, et répond aux exigences de **performance** et de **scalabilité** de l'application.

Le déploiement prévu s'appuie sur une architecture conteneurisée (Docker), orchestrée sur un **Cloud AWS**, de manière à garantir une **grande disponibilité**, une **forte scalabilité**, mais aussi une grande **facilité pour l'intégration et la livraison continues**.

Côté sécurité, l'**authentification JWT**, le contrôle d'accès basé sur les **rôles**, et les **échanges chiffrés en HTTPS** assurent l'intégrité et la confidentialité des flux de données inter-composants.

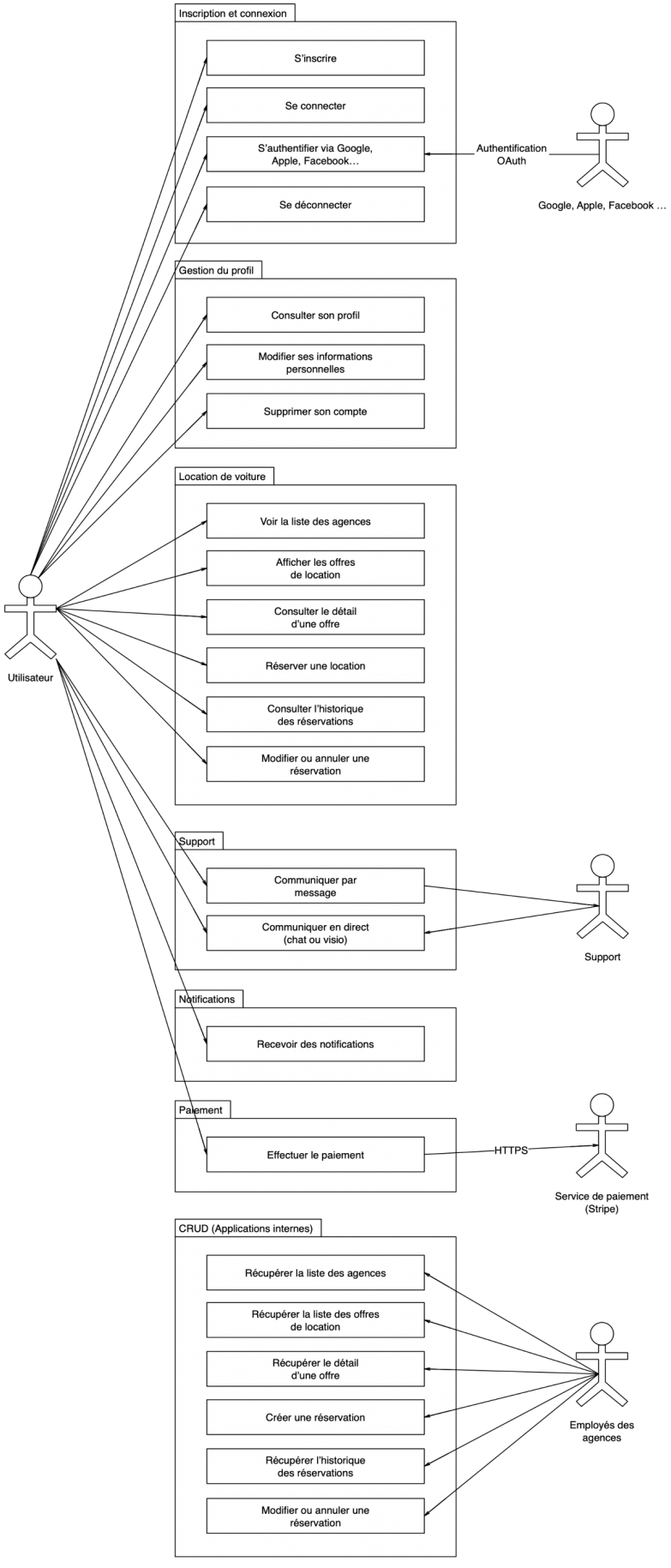
• Architectures de transition

La complète mise en place de l'architecture pouvant représenter une charge importante de travail, on peut choisir, dans un soucis de **livraison rapide**, de ne **livrer qu'une partie des fonctionnalités dans un premier temps**. On priorisera les **fonctionnalités essentielles** de l'application, à savoir, la **réservation de véhicules**. La partie contact avec le support pourra se faire par mail en attendant le déploiement des fonctionnalités de message et de chat.

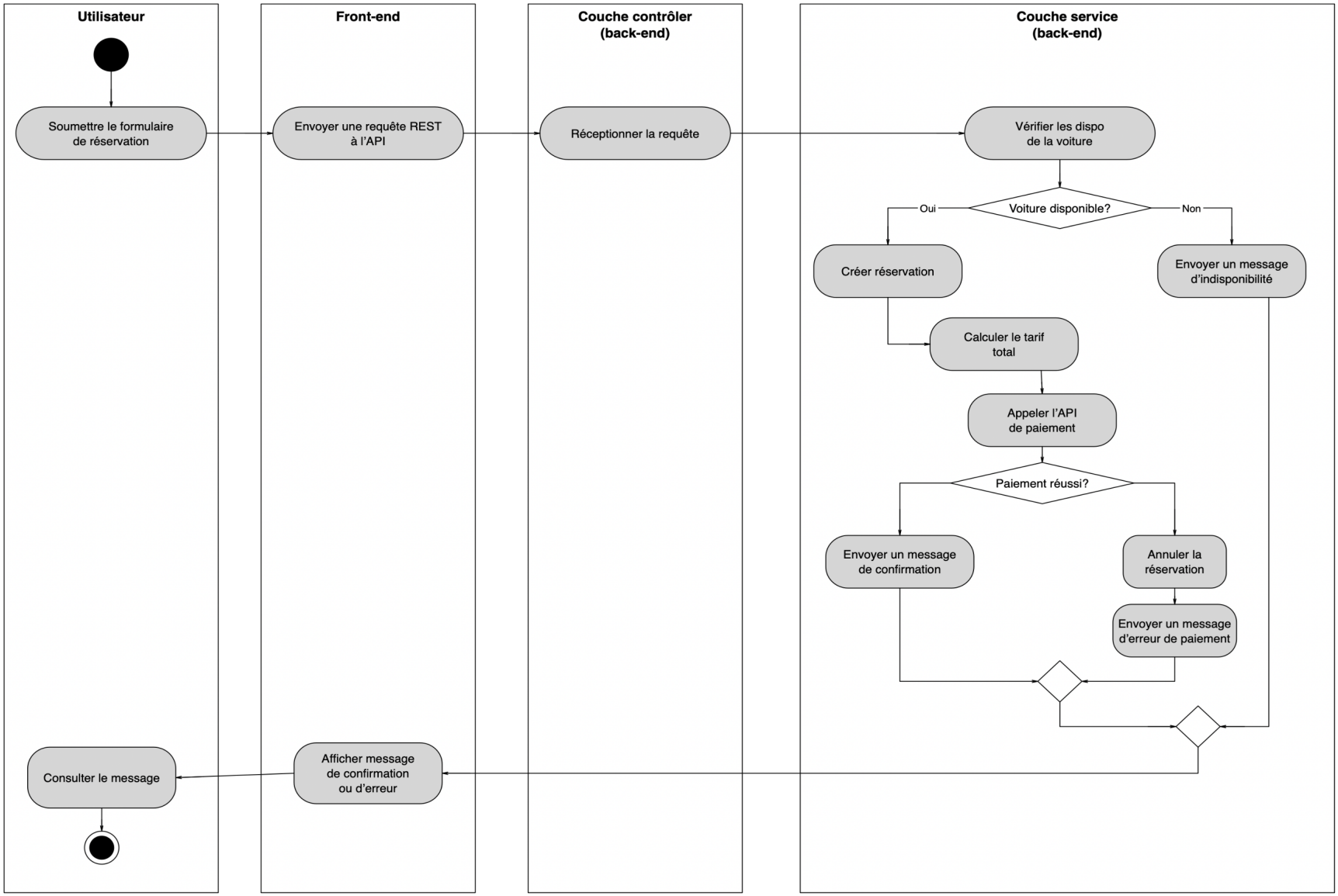
On pourra alors s'occuper dans un **second temps** de déployer les composants de la couche contrôleur utilisant **WebSocket**, ainsi que les composants front-end concernant ces fonctionnalités, et les tables de la base de données concernées.

Ce déploiement en deux temps permettra de s'assurer dans un premier temps du bon fonctionnement des APIs REST avant de se concentrer, dans un second temps, sur le bon fonctionnement des développements basés sur WebSocket.

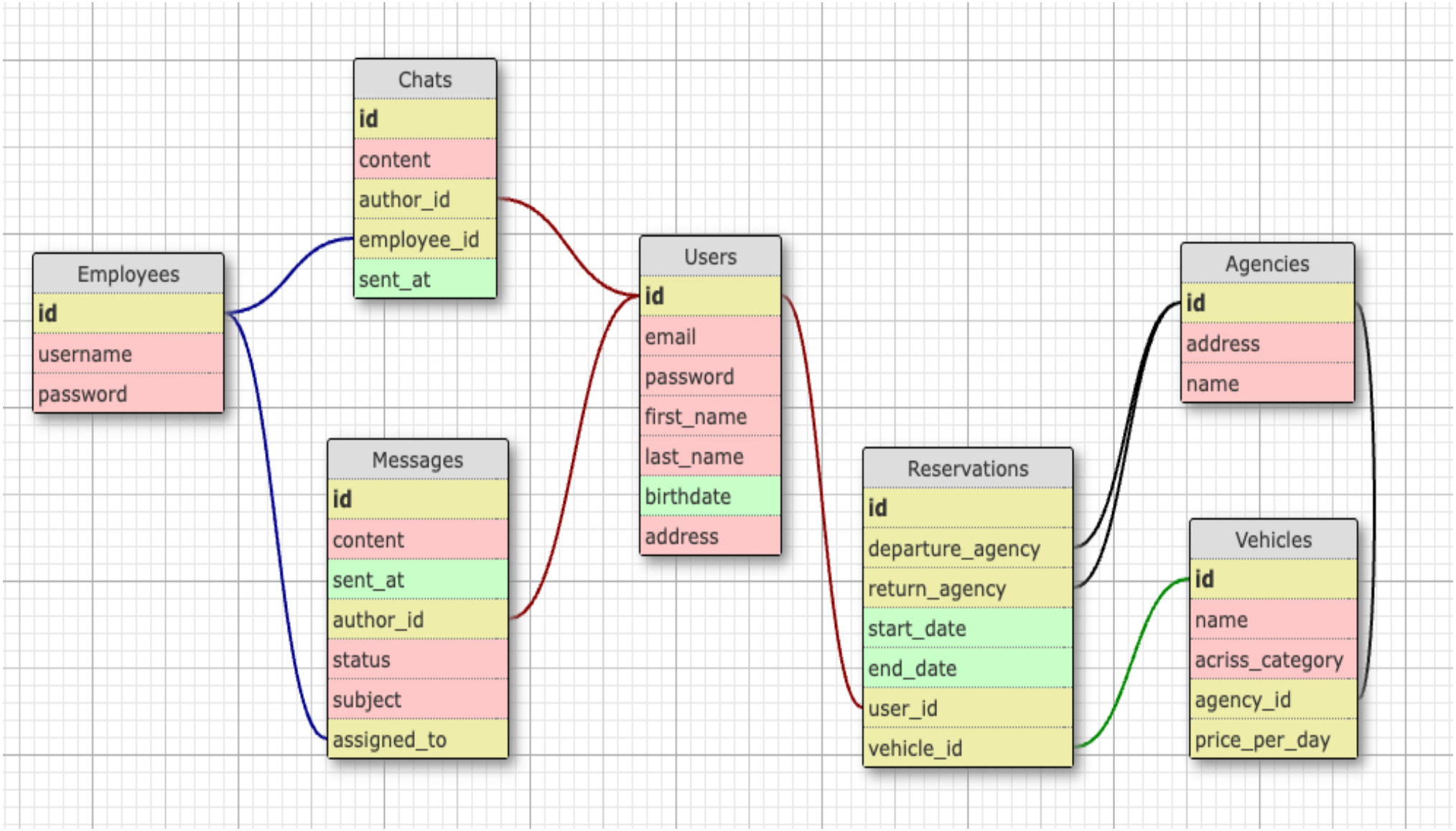
Annexe 1 – Diagramme des cas d'utilisation



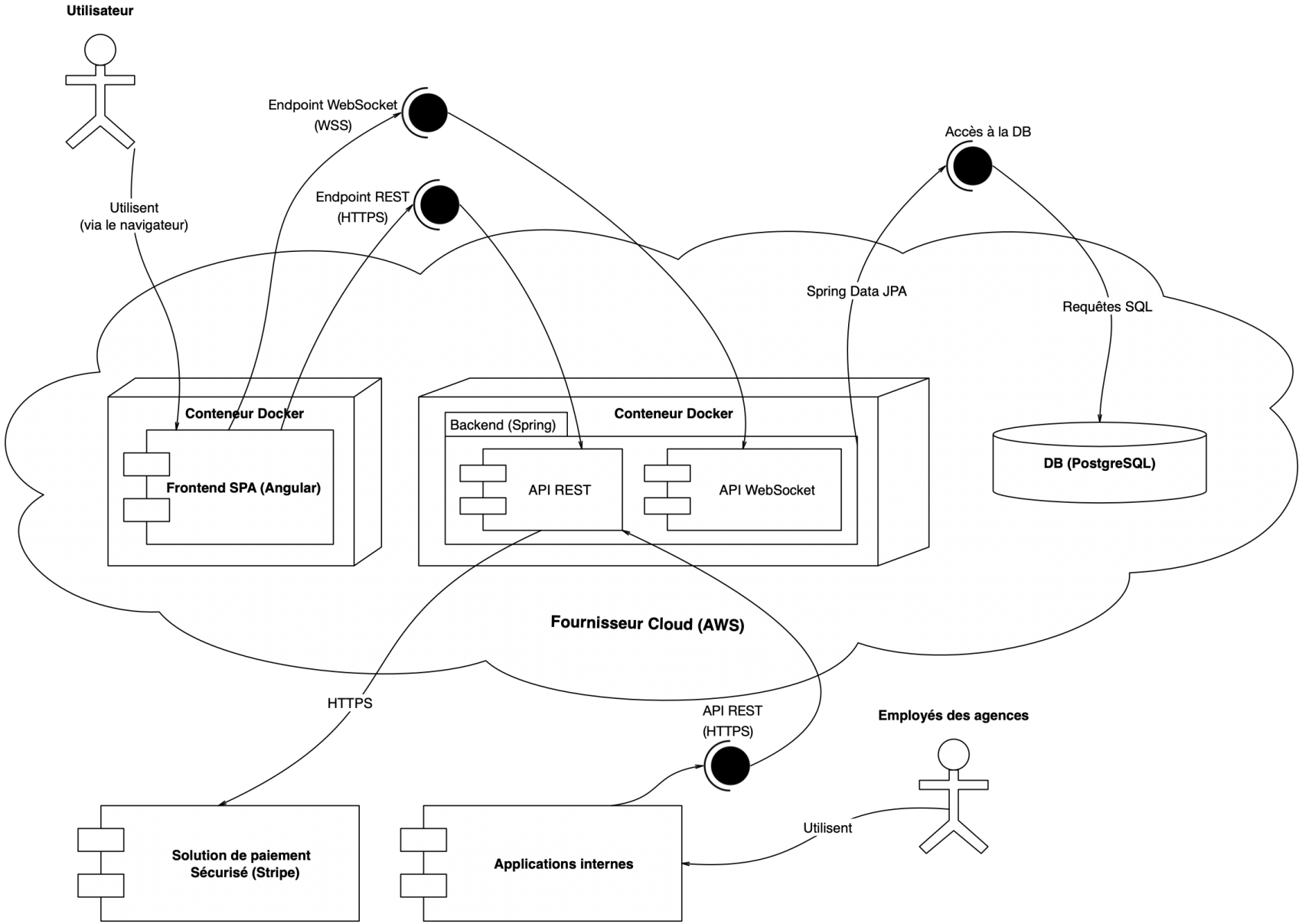
Annexe 2 – Diagramme d’activité



Annexe 3 – Schéma de base de données relationnelle



Annexe 4 – Diagramme de composants



Annexe 5 – Diagramme de déploiement