

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Application de suivi et gestion des Tickets d’assistance**

Par

RANDRIANANTOANDRO My Ranto

Mémoire présenté  
en vue de l’obtention du grade de

En

Option web et design

Jury :

…  
 …  
 …

© ,

Table des matières

[Liste des tableaux 3](#_Toc150263087)

[Liste des figures 3](#_Toc150263088)

[Glossaire 4](#_Toc150263089)

[Avant-propos 5](#_Toc150263090)

[L’IT University 5](#_Toc150263091)

[L’Institution/Entreprise d’accueil 6](#_Toc150263092)

[Remerciements 6](#_Toc150263093)

[Introduction 6](#_Toc150263094)

[1 Présentation du projet 7](#_Toc150263095)

[1.1 Objectifs du projet 7](#_Toc150263096)

[1.2 Planning de réalisation 7](#_Toc150263097)

[1.3 Technologies utilisées 8](#_Toc150263098)

[o Système de gestion de base de données 8](#_Toc150263099)

[o Backend 10](#_Toc150263100)

[1.4 Méthode de gestion de projet 13](#_Toc150263103)

[2 Réalisation du projet 14](#_Toc150263104)

[2.1 Analyse et conception 14](#_Toc150263105)

[o Analyse de l’existant 14](#_Toc150263106)

[o Conception de l’application 15](#_Toc150263107)

[Affichage 15](#_Toc150263108)

[Métier 16](#_Toc150263109)

[2.2 Développement par fonctionnalité ou module 18](#_Toc150263110)

[2.2.1 Gestion des tickets 18](#_Toc150263111)

[2.2.2 Gestion des utilisateurs et des services 22](#_Toc150263112)

[2.2.3 Gestion des plannings 23](#_Toc150263113)

[2.2.4 Synchronisation 25](#_Toc150263114)

[2.2.5 Facturation 27](#_Toc150263115)

[2.3 État d’analyse et statistique 29](#_Toc150263116)

[o Analyse globale 29](#_Toc150263117)

[o Analyse de l’État des tickets 29](#_Toc150263118)

[o Performance temporelle 29](#_Toc150263119)

[o Évolution des tickets et des recettes 30](#_Toc150263120)

[2.4 Problèmes rencontrés et solutions 31](#_Toc150263121)

[3 Évaluation du projet et connaissances acquises 32](#_Toc150263122)

[3.1 Bilan de l’entreprise 32](#_Toc150263123)

[3.2 Bilan personnel 32](#_Toc150263124)

[3.3 Extension et évolution du projet 32](#_Toc150263125)

# Liste des tableaux

[Tableau 1 : Avantages et inconvénients de PostgreSQL 9](#_Toc151059032)

[Tableau 2 : Avantages et inconvénients de SQLite 10](#_Toc151059033)

[Tableau 3 : Comparaison des technologies backend 11](#_Toc151059034)

[Tableau 4 : Comparaison des technologies frontend (web) 12](#_Toc151059035)

[Tableau 5 : Comparaison des technologies pour le développement mobile 13](#_Toc151059036)

# Liste des figures

[Figure 1 : Résumé du planning de réalisation 7](#_Toc151021532)

[Figure 2 : Extrait du planning 7](#_Toc151021533)

[Figure 3 : Structure de l’application 8](#_Toc151021534)

[Figure 4 : Conception interface utilisateurs 15](#_Toc151021535)

[Figure 5 : Modèle conceptuel des données 16](#_Toc151021536)

[Figure 6 : Illustration de l’architecture utiliser dans spring boot 17](#_Toc151021537)

[Figure 7 : Configuration smtp Gmail 18](#_Toc151021538)

[Figure 8 : Formulaire de création de ticket 19](#_Toc151021539)

[Figure 9 : Liste des tickets dans l’application mobile 20](#_Toc151021540)

[Figure 10 : Liste des tickets 20](#_Toc151021541)

[Figure 11 : Modal sous-tâche 21](#_Toc151021542)

[Figure 12 : Détail ticket 22](#_Toc151021543)

[Figure 13 : Modal de création/modification service 23](#_Toc151021544)

[Figure 14 : Liste des personnes (profil admin) 23](#_Toc151021545)

[Figure 15 : Calendrier des tickets 24](#_Toc151021546)

[Figure 16 : Code de la fonction checkTicketDelay 25](#_Toc151021547)

[Figure 17 : Illustration de l’activité de checkTicketDelay 25](#_Toc151021548)

[Figure 18 : Synchronisation 26](#_Toc151021549)

[Figure 19 : Scénario de synchronisation 27](#_Toc151021550)

[Figure 20 : Liste des tickets à facturer selon son regroupement 28](#_Toc151021551)

[Figure 21 : Liste des tickets facturables 28](#_Toc151021552)

[Figure 22 : Détail facture 28](#_Toc151021553)

[Figure 23 : Pourcentage des tickets résolus avec détail agents 29](#_Toc151021554)

[Figure 24 : État et performance temporelle 30](#_Toc151021555)

[Figure 25 : Performance temporelle par agent 30](#_Toc151021556)

[Figure 26 : Évolutions d’entrée des recettes et des tickets 31](#_Toc151021557)

[Figure 27 : Assignation de la valeur de common 31](#_Toc151021558)

[Figure 28 : Planning de réalisation (1) 33](#_Toc151021559)

[Figure 29 : Planning de réalisation (2) 33](#_Toc151021560)

[Figure 30 : Planning de réalisation (3) 33](#_Toc151021561)

[Figure 31 : Planning de réalisation (4) 34](#_Toc151021562)

# Glossaire

**TIC :** Technologie de l’Information et de la Communication.

**ACID :** Ensemble de caractéristiques garantissant la fiabilité et la cohérence des transactions dans une base de données.

**SGBD :** Système de Gestion de Base de Données.

**Forum :** Communauté où on s’échange des informations.

**JPA :** Java Persistence API, un ensemble d’interfaces de programmation qui permet d’organiser les données relationnelles.

**API :** Application Programming Interface, ensemble de protocole permettant à différentes applications de communiquer entre elles.

**CRUD :** Create Read Update Delete, représente les opérations basiques dans une application

**SMTP :** Simple Mail Transfert Protocol, protocole de communication utiliséeé pour l’envoie d’e-mail.

**BLOB:** Binary Large Object, est un type de données qui est utilisé dans le contexte de la gestion de données.

# Avant-propos

Le présent mémoire présente les résultats du travail effectué lors de mon stage de fin d’études de Licence en Informatique de l’IT University; stage effectué au département informatique de Sunsoft de Monsieur RAOELISON Rado Andriamaharo durant 3 mois, d’août à novembre 2023.

Afin de poser clairement le contexte de ce mémoire, je vais présenter succinctement d’une part l’IT University et d’autre part mon institution/entreprise d’accueil.

## L’IT University

Fondée en 2011, l’IT University (ou ITU) est une université privée, spécialisée en informatique, formant les jeunes bacheliers, de préférence scientifiques :

* En trois ans, pour l‘obtention d’une :
  + Licence, option Développement, Réseaux et Bases de Données ou Web et Design
  + Licence, option Graphic Design ou Communication Digitale
* En cinq ans, pour l’obtention d’un :
  + Master MBDS en coopération avec l’Université Côte d’Azur à Nice Sophia Antipolis – France
  + Master BIHAR en coopération avec l’ESTIA du Pays Basque - France

Étant une formation professionnalisante, l’ITU a tissé des liens forts avec ses partenaires industriels, dont l’opérateur convergent TELMA et la plupart des entreprises et institutions du secteur des TIC[[1]](#footnote-1). Ces partenaires participent effectivement à la formation par la fourniture de connexion Internet à haut débit, l’envoi de conférenciers ou par l’accueil des étudiants en stage. Beaucoup de ces partenaires recrutent aussi les sortants dès leur sortie d'école.  
D’autre part, le corps enseignant de l’ITU est constitué intégralement de spécialistes de très haut niveau et obligatoirement actifs professionnellement dans leurs domaines respectifs.

## L’Institution/Entreprise d’accueil

## Remerciements

L'auteur tient à exprimer sa sincère reconnaissance à MM. les Professeurs Smith et Jones pour leur aide lors de la préparation de ce manuscrit.   
Il souhaite également remercier tout particulièrement le Dr. Elsa Leavitt dont la maîtrise du cours, tant sur le plan des besoins que des idées exposées, a été d'une grande aide pour la mise en route de ce projet.   
Enfin, merci également aux membres du comité étudiant pour leur soutien.

# Introduction

Le projet **‘suivi et gestion de ticket d’assistance’** consiste à développer des fonctionnalités permettant de centraliser le suivi des tickets d’assistance des clients et de Sunsoft. L’application est utilisable par les clients, les agents et l’administrateur.

Durant ce stage, j’ai développé l’application dans deux plateformes :

* Site web
* Application mobile (disponible en mode hors ligne)

Afin de bien étudier le sujet, nous allons suivre le plan suivant :

* En premier lieu, la présentation du projet : l’objectif du projet, le planning de réalisation et les technologies utilisées.
* Ensuite, la réalisation du projet: l’analyse et conception, analyse de l’existant, conception de l’application, développement des modules, présentation des problèmes rencontrés et solutions.
* Puis, l’évaluation du projet et les connaissances acquises : bilan de l’entreprise, bilan personnel, l’extension et évolution du projet.
* Enfin, nous verrons la conclusion.

# Présentation du projet

## Objectifs du projet

L’objectif du projet **‘suivi et gestion de ticket d’assistance’** est de développer des fonctionnalités permettant de faciliter la gestion des besoins, anomalies et pannes signalés par les clients utilisant des logiciels spécifiques de l’entreprise. Ce qui permettra une facilité de suivi détaillé des tickets clients.

## Planning de réalisation

Voici un diagramme de Gantt résumant le planning de réalisation de ce projet avec un extrait du planning.

J’ai réalisé ce diagramme sur GitHub car il est beaucoup plus lisible par rapport à celui de Gantt Project.

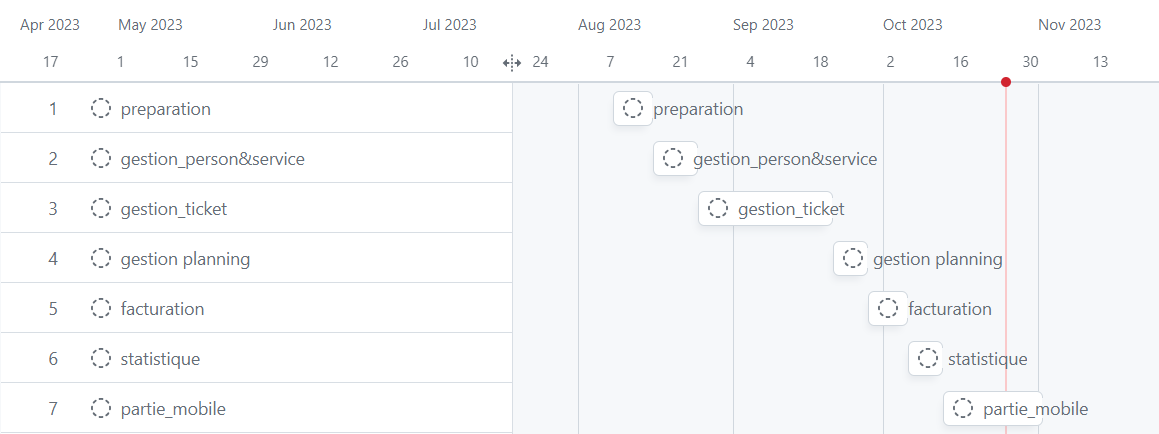


Figure 1 : Résumé du planning de réalisation

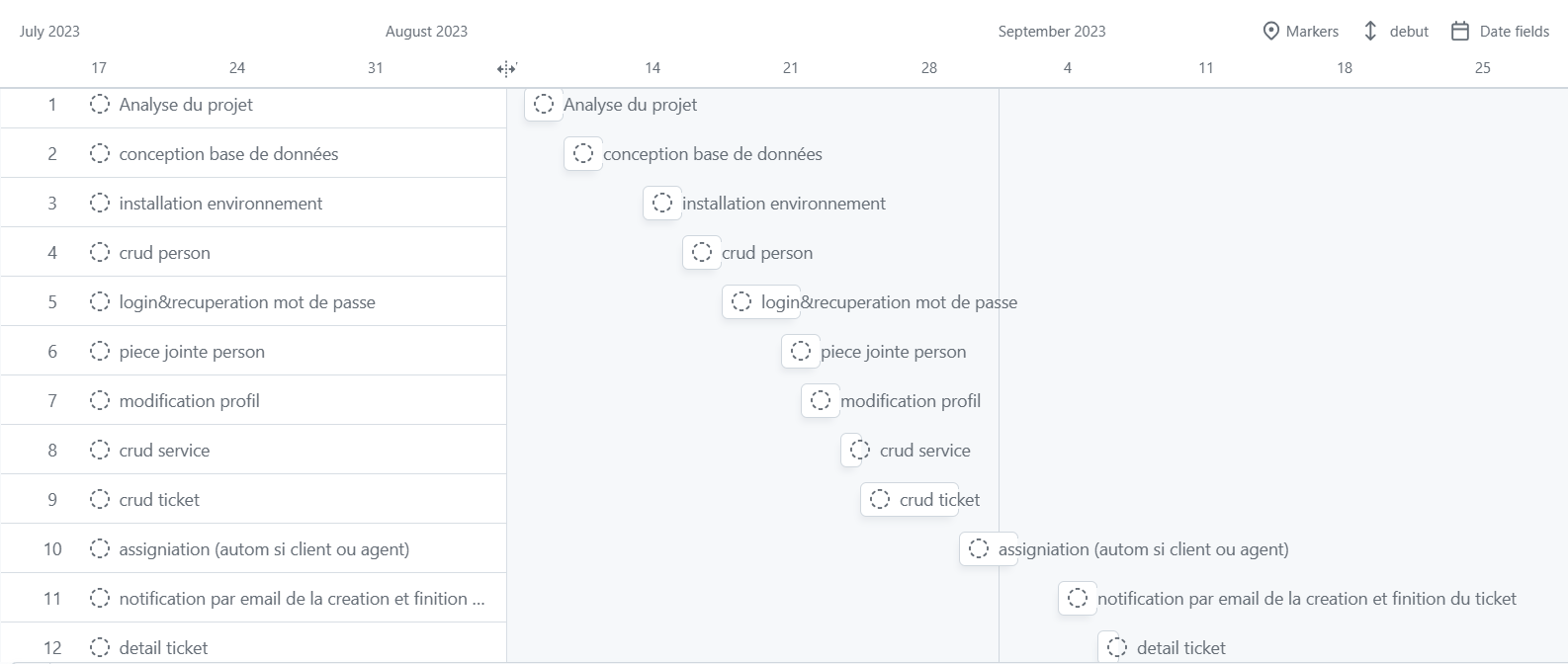


Figure 2 : Extrait du planning

## Technologies utilisées

Dans le projet, nous utilisons l'architecture multi-tiers. Cette architecture utilise le concept client-serveur, une structure d'application distribuée qui répartit les tâches ou les charges de travail entre les fournisseurs d'une ressource ou d'un service, appelés serveurs, et les demandeurs de services, appelés clients. Avec celle-ci, les fonctions de présentation, de traitement des applications et de gestion des données sont physiquement séparées. Un des avantages à utiliser cette architecture est la possibilité de modifier un niveau de l'application sur lequel on travaille sans avoir à la retravailler entièrement. De plus, chaque niveau peut être testé indépendamment des autres.

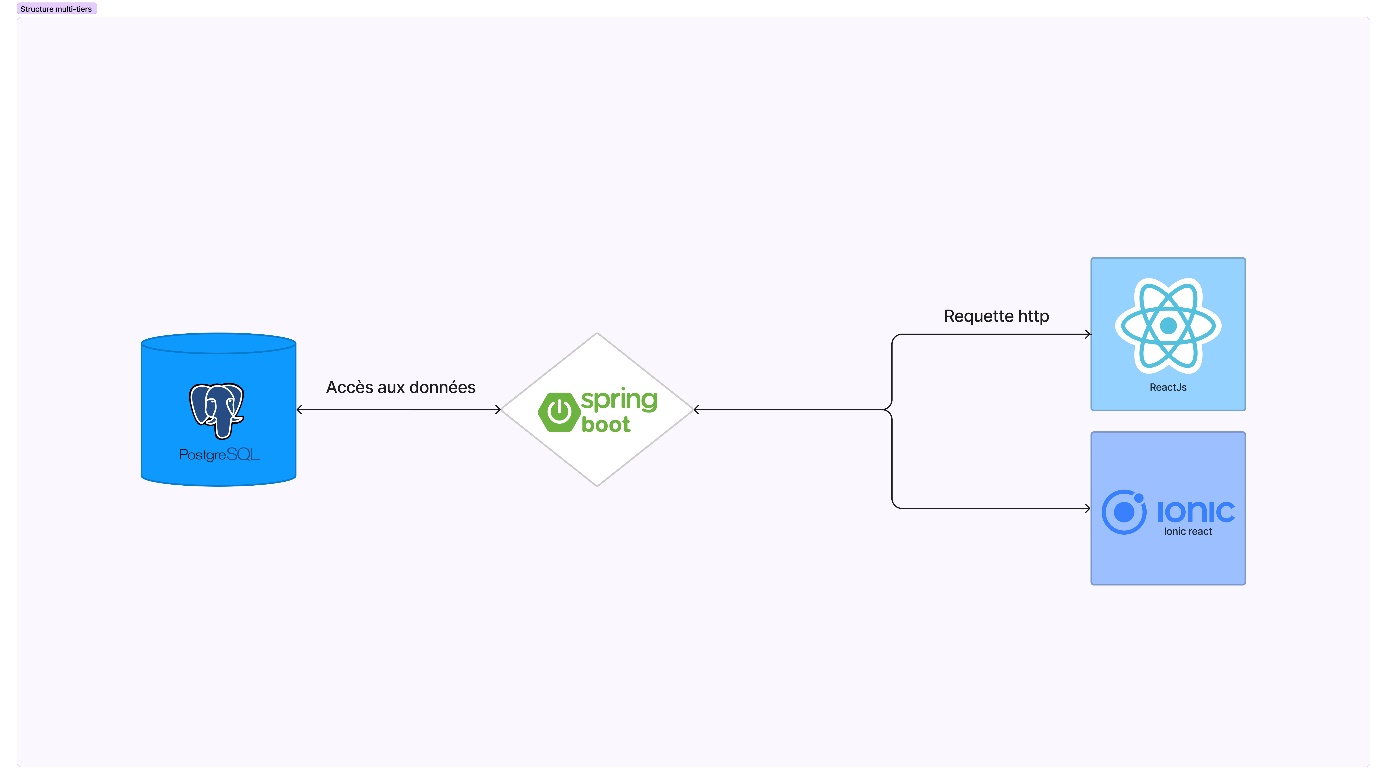


Figure 3 : Structure de l’application

L’application mobile dispose également d’un mode « hors ligne », ce qui signifie qu’elle peut être utilisée même en l’absence de connexion internet. Pour cela nous utilisons une base de données SQLite pour le stockage des données enregistrées hors connexion.

## Système de gestion de base de données

Dans la partie web, nous avons choisi d'utiliser PostgreSQL comme base de données, bien que MySQL aurait également pu convenir. Notre décision s'est principalement basée sur le fait que notre société disposait déjà d'un hébergement PostgreSQL, ce qui accélère le déploiement du projet.

Pour ce qui est de la partie mobile, nous avons opté pour SQLite en tant que base de données. Cette décision découle de sa grande disponibilité en termes de documentation et de support sur des plateformes telles que ‘Stack Overflow’. De plus, l'utilisation de SQLite permet à l'application mobile de fonctionner même en l'absence de connexion Internet."

Voici deux tableaux qui représente en quelque description les bases de données PostgreSQL et SQLite



|  |  |
| --- | --- |
| Avantages | Inconvénients |
| Garantit la conformité aux propriétés ACID [[2]](#footnote-2) assurant la fiabilité des transactions. | PostgreSQL peut avoir une empreinte mémoire relativement plus importante, ce qui peut nécessiter une gestion plus précise des ressources système. |
| Offre des mécanismes de réplication, de clustering et de basculement pour garantir une haute disponibilité et une tolérance aux pannes. | Par rapport à certains systèmes de gestion de bases de données commerciaux, PostgreSQL peut manquer de certaines fonctionnalités natives, mais celles-ci peuvent souvent être ajoutées via des extensions. |
| Possibilité de disposer de ‘vues' (vue normale, vue matérialisée), de fonction et bien d’autres | En raison de sa richesse en fonctionnalités, PostgreSQL peut demander plus d'efforts d'apprentissage pour les nouveaux utilisateurs. |

Tableau 1 : Avantages et inconvénients de PostgreSQL



|  |  |
| --- | --- |
| Avantages | Inconvénients |
| Faible consommation de mémoire | Les performances se dégradent lorsque la taille de la base de données devient volumineuse |
| La base de données est un fichier unique ce qui facilite sa manipulation | Ne permet pas de sous requête après la clause ‘from’ |
| Meilleure prise en main | Manque de certaines fonctionnalités avancées des SGBD [[3]](#footnote-3) relationnels |
| Grande disponibilité dans les documentations et les forum[[4]](#footnote-4) |  |

Tableau 2 : Avantages et inconvénients de SQLite

## Backend

Dans cette partie, nous avons opté pour Java en utilisant le Framework Spring Boot en utilisant JPA [[5]](#footnote-5) car nous avons besoin de créer plusieurs API [[6]](#footnote-6) avec une grande rapidité ainsi qu’une bonne sécurité. Spring boot possède aussi un aspect autonome, c’est-à-dire que l’application peut être exécutée indépendamment sans nécessiter un autre serveur d’application, cela favorise grandement le processus de déploiement et de distribution de l’application. Spring boot est donc idéal pour le bon développement du projet de gestion et suivi des tickets d’assistance.

Toutefois, on aurait aussi très bien pu utiliser d’autres technologies pour le développement de l’application comme :

* NodeJS car il est à la fois léger et rapide, la création de plusieurs API peuvent se faire rapidement grâce au Framework Express.js.
* C# avec Asp.net qui est aussi léger, ce qui permet une grande flexibilité dans le développement du projet

|  |  |  |
| --- | --- | --- |
| **Critère** | **Spring Boot** | **Express.js** |
| **Langage de programmation** | Java, langage mature et fortement typé | JavaScript, langage interprété |
| **Communauté et écosystème** | Grande communauté avec bibliothèques variées | Communauté active, NPM riche en module |
| **Performance** | Bonne performance, adapté aux applications lourdes | Bonne performance pour les applications légères |
| **Développement** | Structure rigide, configuration initiale plus complexe | Flexible, rapide à démarrer, moins de configuration |
| **Gestion de dépendance** | Maven ou Gradle, gestionnaire de dépendances robuste | NPM, gestionnaire de dépendances simple |
| **Sécurité** | Sécurisé, intégration avec Spring Security | Besoin d’ajouter des modules tiers pour la sécurité |
| **Maintenance** | Peut nécessiter une gestion plus complexe des dépendances | Moins de complexité dans la gestion des dépendances |
| **Déploiement** | Doit être déployé sur des serveurs Java | Peut-être déployé plus facilement dans des conteneurs Docker |

Tableau 3 : Comparaison des technologies backend

## Frontend

Pour cette partie, nous avons principalement utilisé ReactJS pour le développement de la partie web. Grâce à la modularité de ReactJS, celui-ci favorise la créativité et permet la création d’interfaces riches en fonctionnalités avec moins de lignes de code que d’autres bibliothèques.

De plus, le concept des composants permet la réutilisation de ces derniers, accélérant ainsi le développement et simplifiant la maintenance de l’interface utilisateur. La clarté du code est également un avantage majeur car on pourra s’y repérer plus facilement lors du débogage. En outre, ReactJS s’intègre très bien avec d’autres technologies et Framework.

|  |  |  |
| --- | --- | --- |
| **Critère** | **ReactJS** | **Angular** |
| **Langage de programmation** | JavaScript, TypeScript (optionnel) | TypeScript (obligatoire) |
| **Flexibilité et Modularité** | Léger et flexible | Plus restrictif en termes de structure, suit un ensemble de conventions strictes |
| **Bibliothèque** | Bibliothèque de base léger ce qui facilite le chargement de l’application | Bibliothèque de base volumineuse, peut entrainer des temps de chargement plus longs |
| **Performance** | Performances élevées | Performances élevées, mais nécessite une attention particulière à l’optimisation |
| **Intégration avec d’autres technologies** | Intégration facile | Intégration étroite avec des outils spécifique à Angular |
| **Communauté** | Grande communauté | Communauté active |
| **Maintenance** | Facilité de maintenance grâce au concept des composants | Complexe en raison de la structure rigide |
| **Réutilisabilité des composants** | Composants réutilisables, ce qui favorise l’efficacité du développement | Composant réutilisable mais peut nécessiter une configuration plus lourde |

Tableau 4 : Comparaison des technologies frontend (web)

En ce qui concerne la partie mobile, nous avons opté pour l’utilisation d’Ionic react.

En effet, nous aurions très bien pu simplement utiliser Java et Android studio mais en raison des besoins spécifique du projet ainsi que des inconvénients liés à l’utilisation de java pour cette partie, nous avons maintenu ce choix. De plus, étant donné que nous avions déjà utilisé ReactJS dans la première partie du développement frontend, l’utilisation d’Ionic react nous a permis de nous sentir très à l’aise et à éviter de perdre du temps à apprendre une nouvelle technologie tout en développant l’application.

|  |  |  |
| --- | --- | --- |
| **Critère** | **Ionic react** | **Java (avec android studio)** |
| Langage de programmation | JavaScipt, TypeScript (optionnel) | Java |
| Développement | Permet de développer des applications multiplateformes | Développement d’application réserver au système d’exploitation Android |
| Performance | Bonne performance | Performance solide |
| Intégration avec d’autres technologies | S’intègre bien avec d’autres technologies | S’intègre bien avec d’autres technologies |
| Maintenance | Facilité de maintenance grâce au concept des composants | Une maintenance plus complexe en raison de la gestion avancée de l’état et des spécificités d’Android |

Tableau 5 : Comparaison des technologies pour le développement mobile

## Méthode de gestion de projet

Nous avons adopté la méthode Merise pour la gestion et le suivi de notre projet.

Merise, avec son architecture structurée et séquentielle, offre une approche planifiée pour concevoir et réaliser efficacement un projet informatique. Cette méthode se distingue des méthodologies agiles qui sont plus adaptives et flexibles, en mettant l’accent sur une organisation rigoureuse.

Nous avons suivi les étapes suivantes:

* Etude du projet : nous avons commencé par comprendre les besoins généraux de l’entreprise, identifier les fonctionnalités requises pour l’application et déterminer qui seront les utilisateurs (agents, clients, administrateurs).
* Conception :

Nous avons ensuite travaillé sur la conception des interfaces pour chaque fonctionnalité du projet. Nous avons également élaboré l’architecture de la base de données pour répondre aux contraintes de conception des interfaces.

Ensuite, nous avons défini un plan de projet en priorisant les fonctionnalités essentielles.

* Réalisation :

C’est dans cette étape que nous allons développer l’application en suivant le plan établi.

* Test :

Après la réalisation, nous effectuons des tests pour s’assurer que l’application fonctionne correctement. Cependant, j’ai préféré faire des tests unitaires après chaque fonctionnalité terminer afin d’avoir une meilleure qualité

* Mise en production :

Une fois tous les tests effectués avec succès et est jugé prêt, l’application est déployée en production pour être utilisé par les utilisateurs finaux

# Réalisation du projet

# Analyse et conception

# Analyse de l’existant

Actuellement, pour accéder aux services (signalement de bug, maintenance, formation, installation logicielle…) de Sunsoft, les clients sont contraints de soit téléphoner, soit de se rendre directement auprès d’un agent responsable, cette procédure engendre une perte de temps significative. De plus, pour obtenir une vision globale des services rendus, y compris le nombre de services effectués par un client ou un agent, ainsi que d’autres détails pertinents, les données doivent être enregistrées manuellement dans Excel, et aussi les analysés de manière manuelle. C’est pourquoi nous avons lancé ce projet, visant à apporter une amélioration significative à au système actuel. Ce qui permettra de réduire considérablement la nécessité d’appels téléphoniques et d’interactions directes avec les agents. De plus, elle offrira une vue d’ensemble plus précise des statistiques, qui seront désormais calculées automatiquement par le système.

# Conception de l’application

Une bonne conception permet une analyse approfondie de l’ensemble du projet, prévoit d’éventuelles modifications pour minimiser les travaux, et contribue également au succès de l’application. Nous avons cependant mis en place plusieurs couches de conception qui se présentent comme suit :

# Affichage

Le projet est structuré de deux interfaces principales:

* L’interface web, qui comprend 20 pages dont une page d’erreur, ainsi que 6 modales
* L’interface mobile qui est un peu plus compacte avec un total de 4 modales et 16 pages, dont une page d’erreur.

Nous avons donc utilisé l’outil collaboratif de conception d’interface, ‘Figma’, pour concevoir et visualiser les écrans avant de les développer.

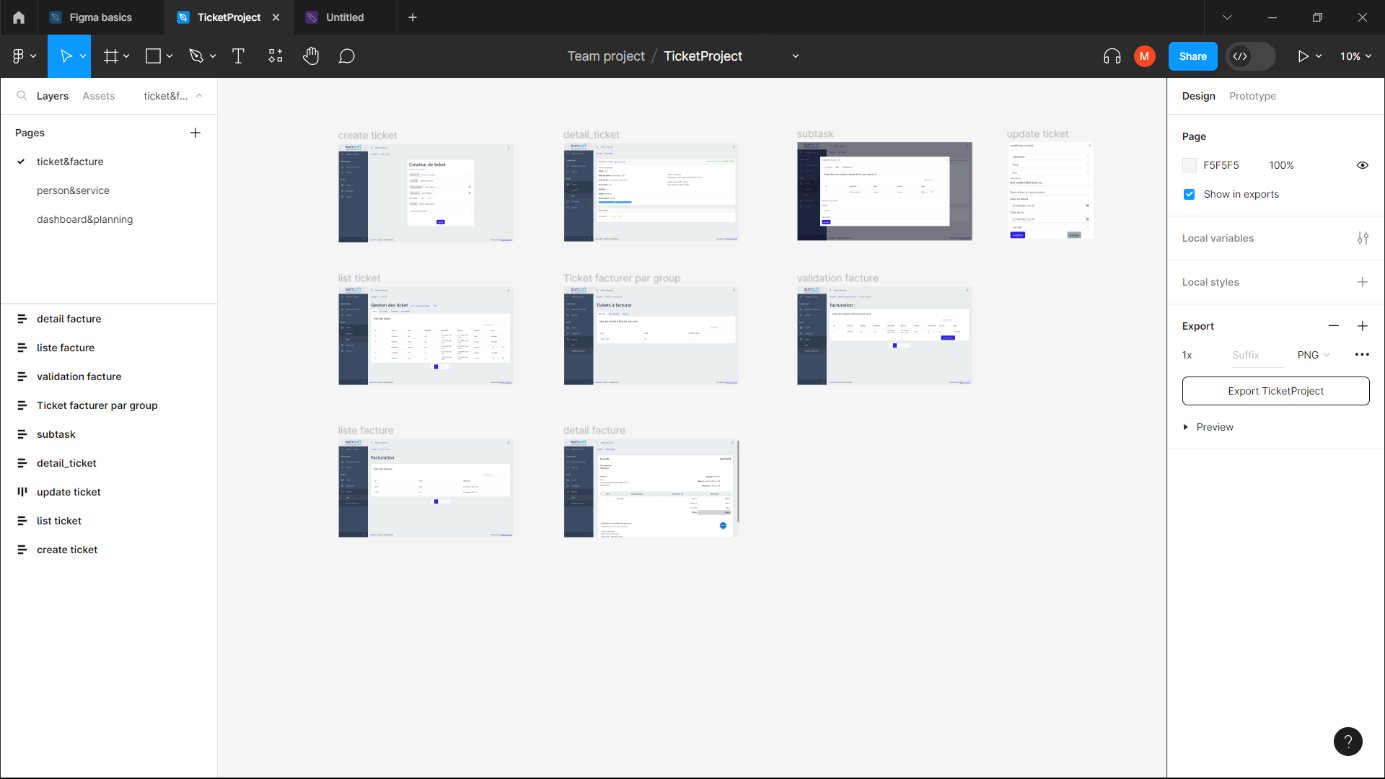


Figure 4 : Conception interface utilisateurs

La base de données

Étant donné que le projet nécessite une synchronisation des données entre l’application mobile en mode hors ligne et le site web, les deux bases de données (PostgreSQL et SQLite) partagent une conception commune. Au total, nous avons 13 tables. Trois profils sont attribués à la table ‘person’, avec une colonne ‘rôle’ qui définit le type de profil connecté. La table ‘execSql’ est utilisée pour stocker les scripts à synchroniser lors de chaque sauvegarde, modification ou suppression de données.

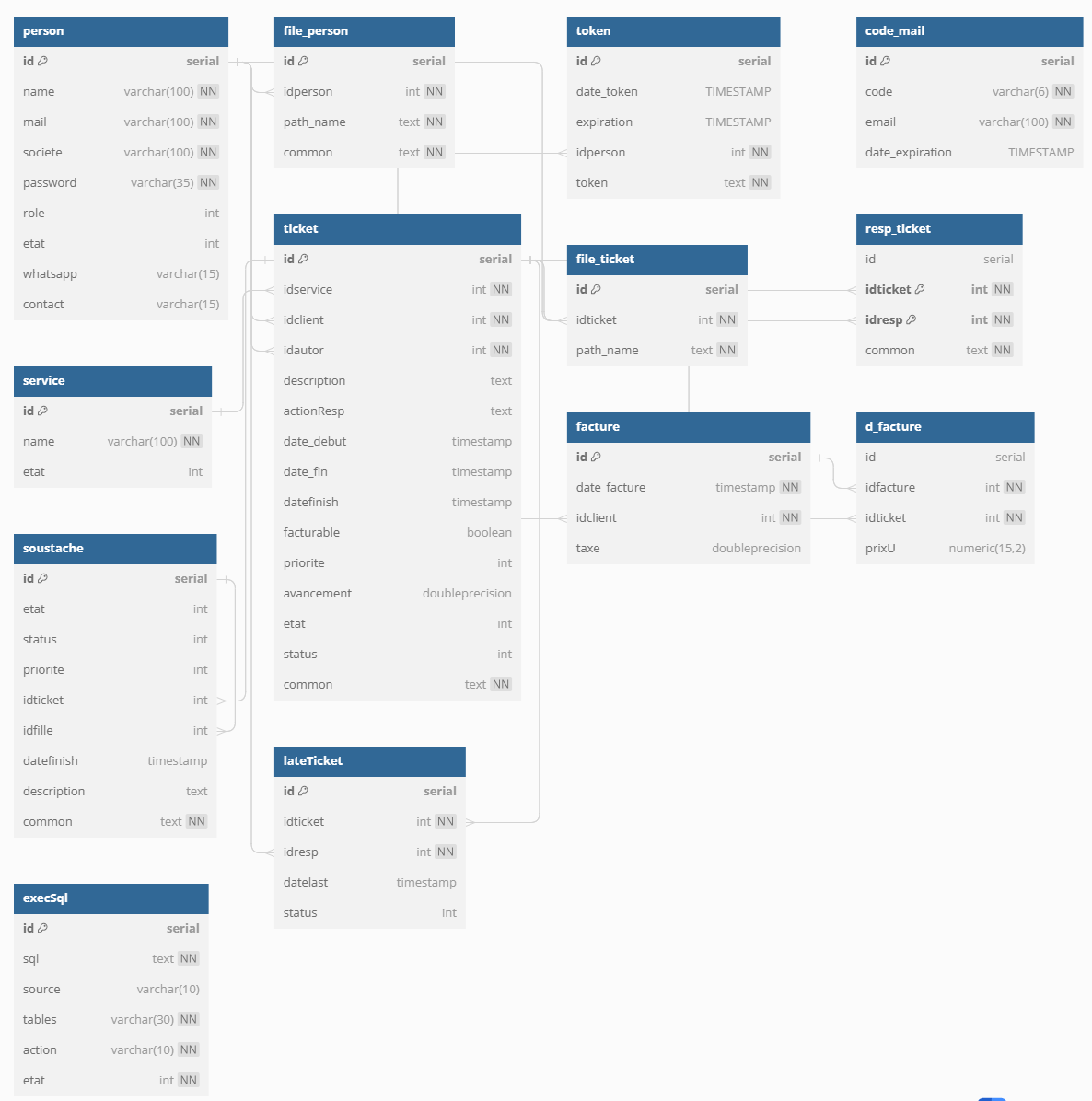


Figure 5 : Modèle conceptuel des données

# Métier

Dans cette partie, nous avons défini trois couches principales : les modèles, les contrôleurs et les services.

Les modèles sont des classes qui permettent d’accéder aux données et de les manipuler, par l’intermédiaire des repository (ce sont des interfaces où on effectuera les requêtes sql).

Les contrôleurs sont des classes qui gèrent les requêtes entrantes et sortantes. Ils font le lien entre les services et les vues.

Les services sont des classes qui implémentent la logique métier du système. Ils effectuent les contrôles, les calculs, etc.

Nous avons créé 20 interfaces dont 16 sont des ‘repository’ (pour les modèles et les vues utilisée), et 4 sont pour d’autres utilisation, puis 46 classes, réparties comme suit:

* 11 contrôleurs
* 13 modèles
* 18 services
* 3 classes d’exception
* Une classe ‘ToJsonData’ qui sert à formater les données renvoyées en JSON

Nous avons également utilisé le principe de l’héritage pour simplifier le code. Ainsi, nous avons créé une classe ‘CrudController’ qui contient les méthodes de base pour le CRUD [[7]](#footnote-7)ainsi que d’autres fonction comme la pagination. Chaque contrôleur spécifique hérite de cette classe et n’a pas besoin de redéfinir ces méthodes. De même, nous avons créé une classe ‘CrudService’ qui regroupe les fonctionnalités communes aux services CRUD. Chaque service spécifique hérite de cette classe et peut la compléter selon ses besoins.

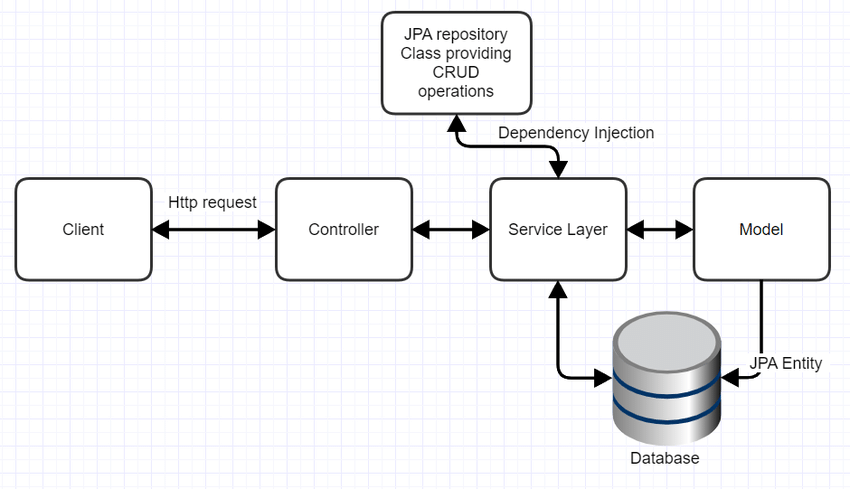


Figure 6 : Illustration de l’architecture utilisé dans spring boot

* Prérequis : Notification par email

Pour pouvoir envoyer un email via spring boot, nous avons besoin d’un serveur smtp[[8]](#footnote-8), cependant dans la phase de test nous utiliserons celui de Gmail en utilisant mon compte personnel, on utilisera celui de l’entreprise lorsque le projet devra être mis en production.

Voici l’extrait de configuration pour l’utilisation de Gmail dans spring boot, on le configurera dans application.properties :

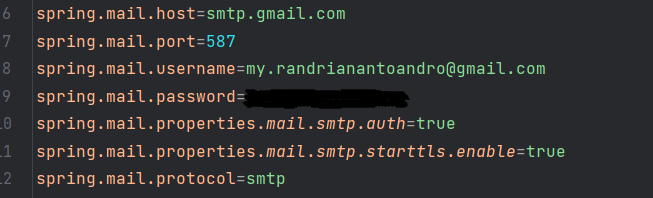


Figure 7 : Configuration smtp Gmail

# Développement par fonctionnalité ou module

Le projet comporte 3 profils : client, agent et administrateur.

# Gestion des tickets

Au sein du projet, le module de gestion des tickets est une composante clé permettant de gérer efficacement les demandes des clients, de suivre leur progression et de garantir une communication transparente. Ce module offre un ensemble de fonctionnalités cruciales pour assurer une gestion professionnelle des tickets, voici une description plus claire de ses fonctionnalités :

* Création de Ticket :

Lorsqu'un ticket est créé, que l'auteur soit un client, un agent ou un administrateur, un email de notification est automatiquement envoyé au client concerné. Cela assure une communication immédiate et un accusé de réception de la demande.

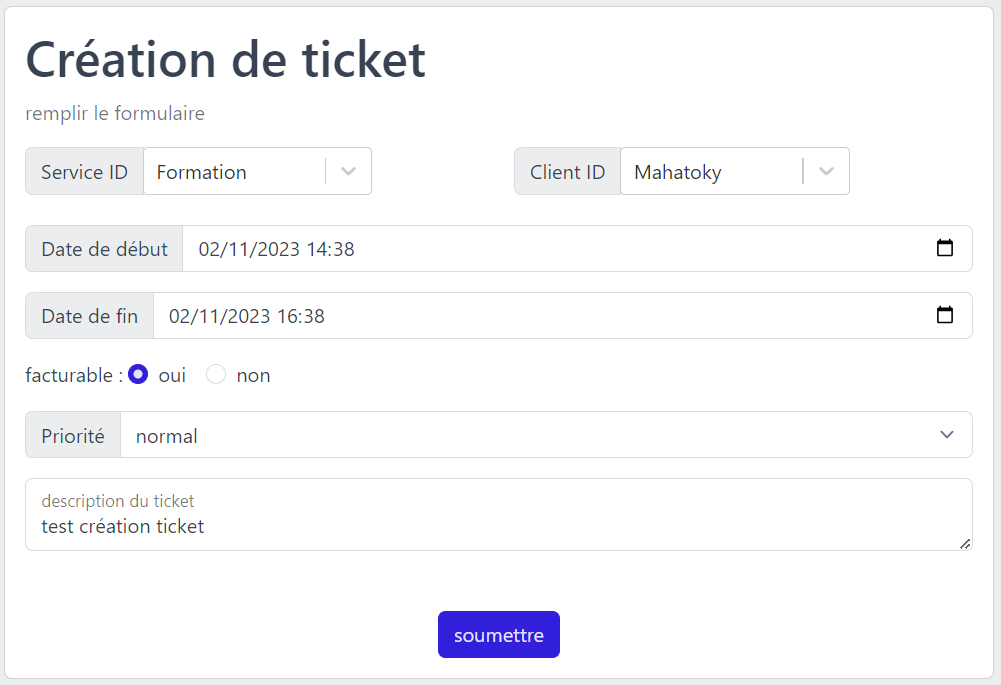


Figure 8 : Formulaire de création de ticket

* Validation :

Après la création d’un ticket, l’administrateur doit valider ce dernier pour qu’il soit traitable mais si c'est lui qui le crée, alors le ticket sera automatiquement validé, ce qui permet de commencer le processus de résolution sans délai.

* Assignation de Tickets :

L'administrateur a la possibilité d'assigner un ticket à un agent spécifique, garantissant que chaque demande est traitée par le membre le plus approprié de l'équipe. Partant du même système que la validation, si l’auteur est un agent, celui-ci sera le responsable d’office du ticket créé. De plus, un agent peut décider de prendre en responsabilité un ticket qui n’a pas encore été assigner.

Entre outre, lorsque le client créera un ticket, ce dernier sera assigné automatiquement à un agent selon la disponibilité de celui-ci.

* Classification des Tickets :

Les tickets peuvent être listé en fonction de leur statut, avec des catégories telles que "Terminé", "Ticket non valider", "En cours" ou "Tous les tickets". Cette classification simplifie la gestion et le suivi des demandes. De plus, les tickets à haute priorité sont mis en avant dans cette classification. Cette mise en évidence assure que les demandes critiques sont traitées en priorité, améliorant ainsi la réactivité et l'efficacité de la résolution des problèmes.



Figure 9 : Liste des tickets dans l’application mobile

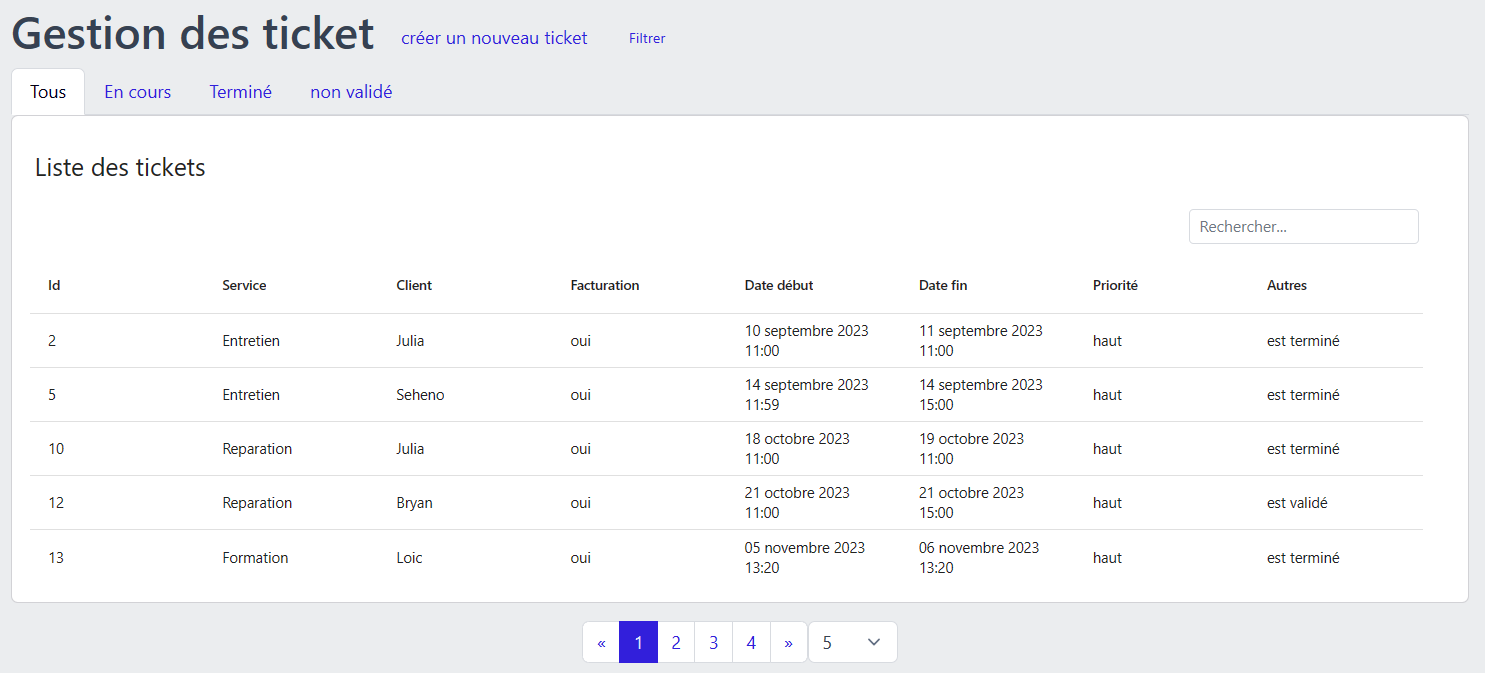


Figure 10 : Liste des tickets

* Gestion des Sous-Tâches :

Le module permet d'ajouter des sous-tâches à un ticket, permettant de diviser les demandes complexes en étapes plus gérables. La progression du ticket est calculée automatiquement en fonction de l'accomplissement des sous-tâches ou des changements de statut.

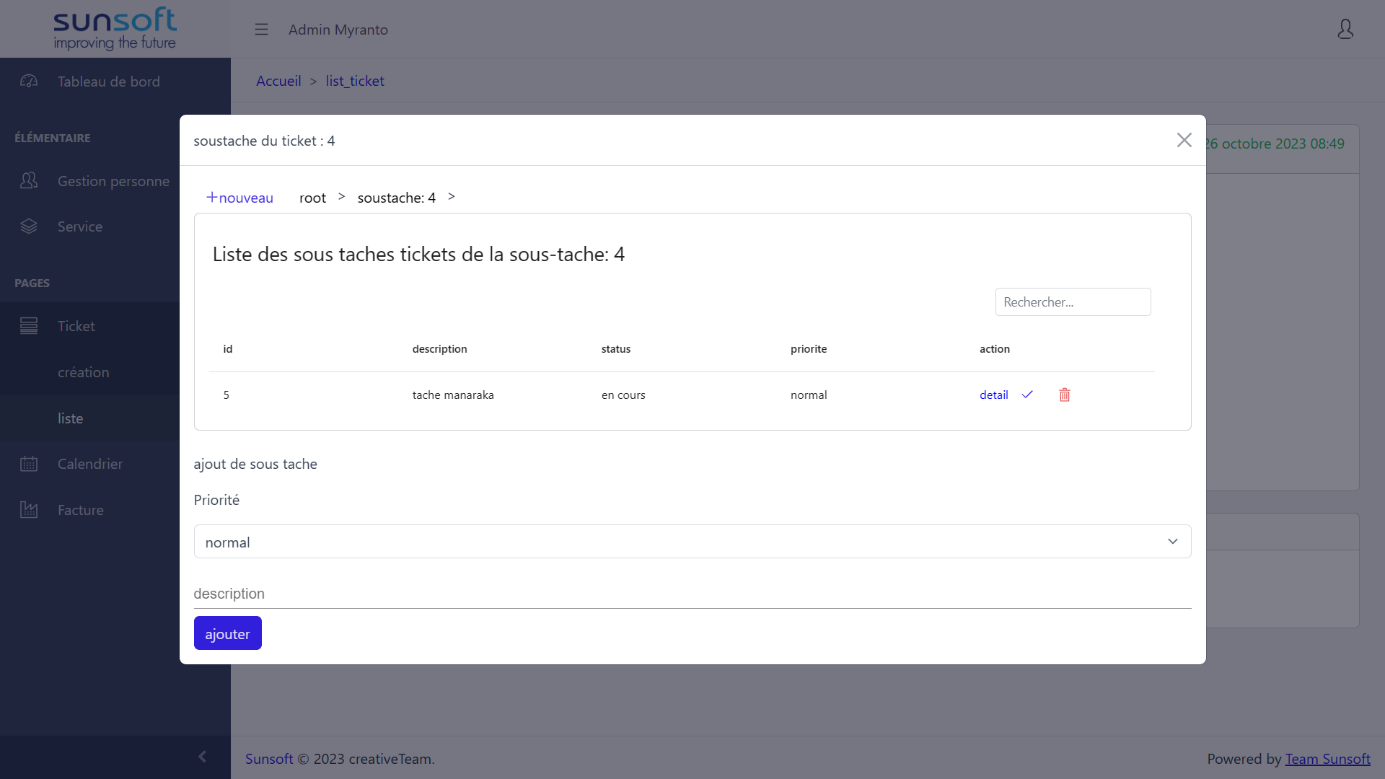


Figure 11 : Modal sous-tâche

* Pièces Jointes :

Les utilisateurs ont la possibilité d'attacher des fichiers pertinents aux tickets, facilitant ainsi le partage d'informations pour la résolution des demandes.

* Notifications de Progrès :

Les clients reçoivent des notifications par email à chaque modification de statut de leur ticket ou à la finition d'une sous-tâche, garantissant une communication en temps réel sur l'état de leur demande.

* Suppression Sécurisée :

Pour éviter toute suppression accidentelle, une confirmation est requise avant de supprimer un ticket. Cette mesure de sécurité protège les données importantes et prévient toute perte involontaire d'informations.

En résumé, ce module de gestion des tickets est essentiel à notre projet, permettant une gestion professionnelle et efficace des demandes des clients, une communication transparente et un suivi précis de l'avancement. Il contribue à améliorer l'efficacité globale de notre projet en offrant une expérience utilisateur fluide et organisée.

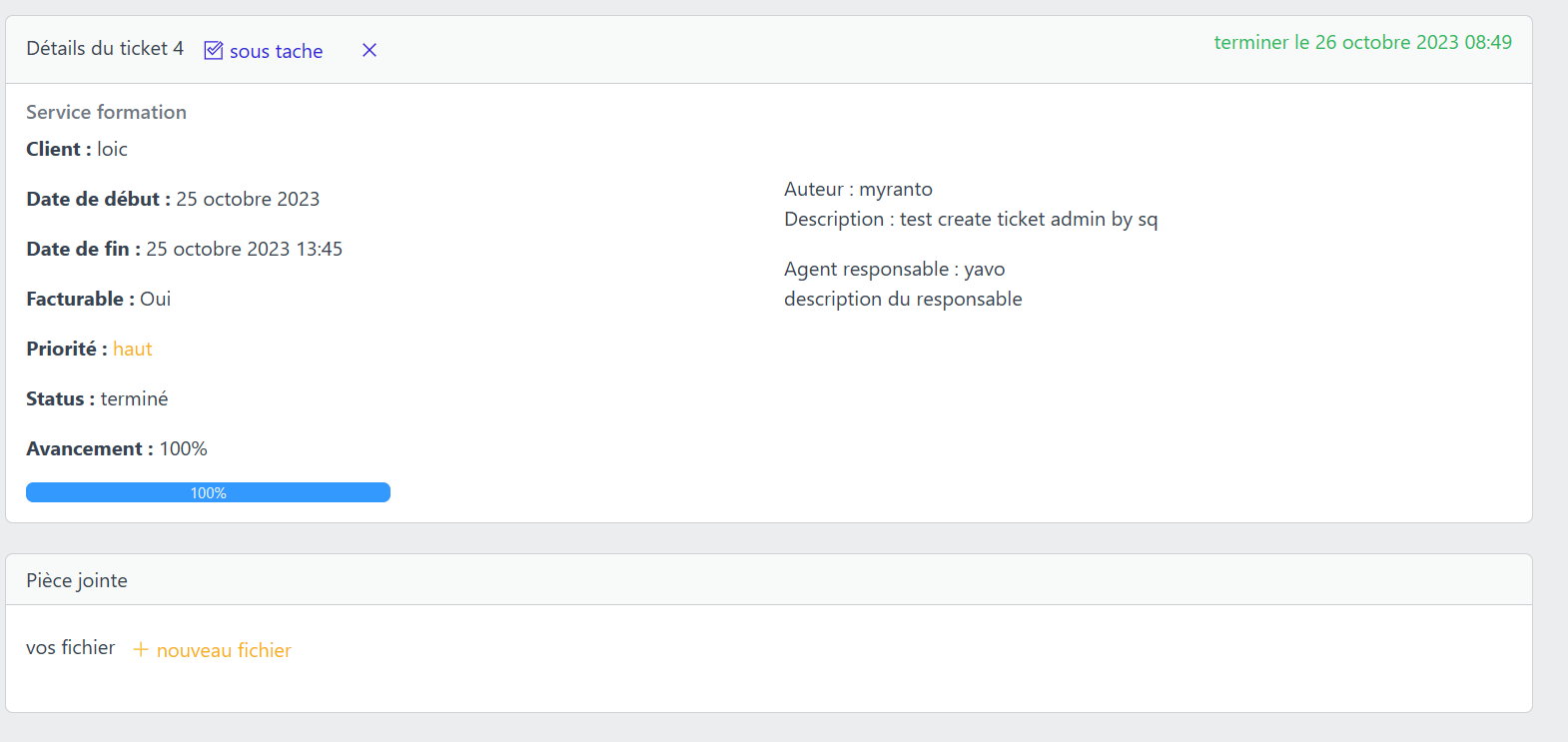


Figure 12 : Détail ticket

# Gestion des utilisateurs et des services

Le module utilisateur et le module service offrent tous deux les fonctionnalités de créer, modifier, supprimer et lister des éléments.

* Les utilisateurs créés peuvent avoir le rôle de client, d’agent ou d’administrateur.
  + Client : c’est le propriétaire des tickets qu’il a créés ou qui ont été créés par un agent ou l’administrateur en son nom. Il peut consulter et modifier ses tickets.
  + Agent : c’est la personne responsable de la résolution des tickets. Il peut prendre en charge les tickets encore non assignés.
  + Administrateur : c’est la personne qui gère tout le système. Il peut créer, modifier et supprimer des utilisateurs et des services. Il peut également créer, modifier, supprimer et valider des tickets pour n’importe quel client.
* La création, la modification et la suppression de service sont des actions réservées à l’administrateur. Un service sert à déterminer ce que le client souhaite obtenir dans la création de ticket.

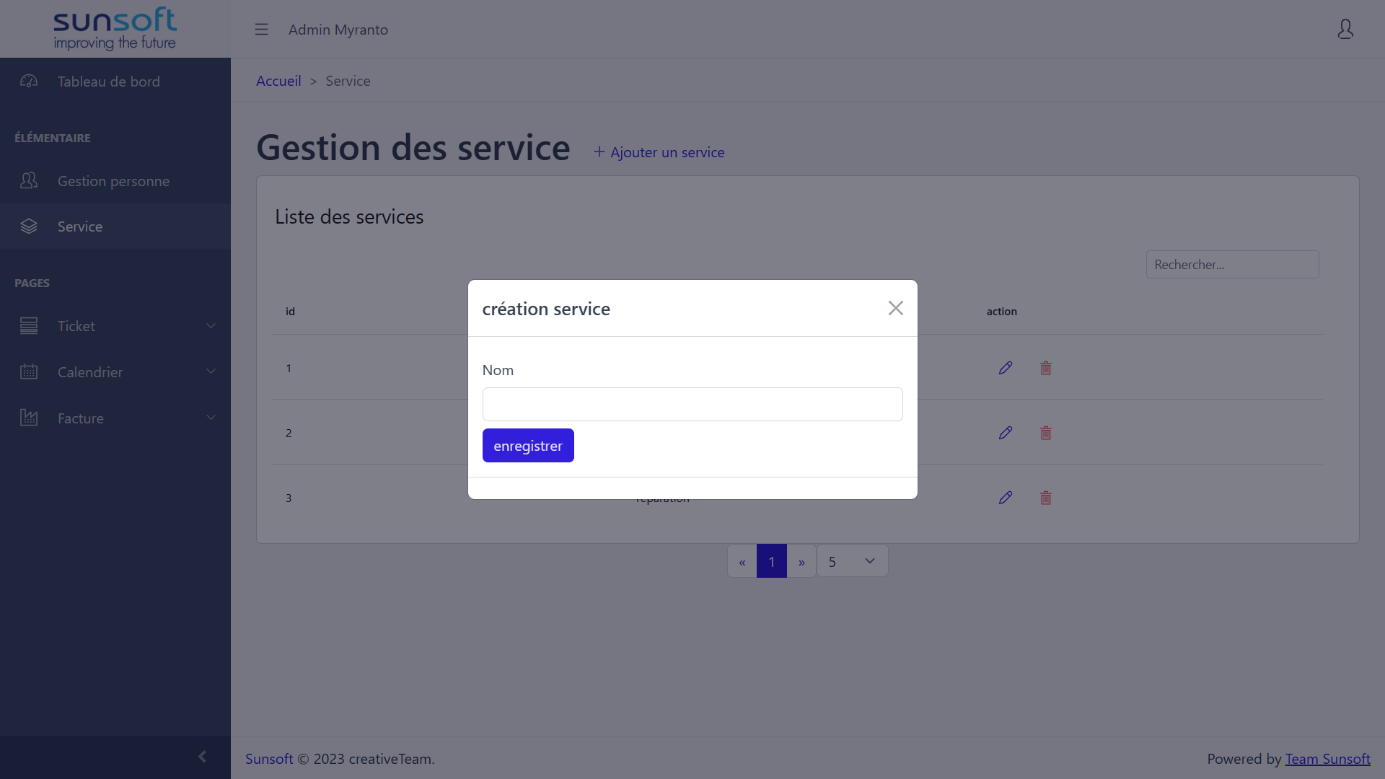


Figure 13 : Modal de création/modification service

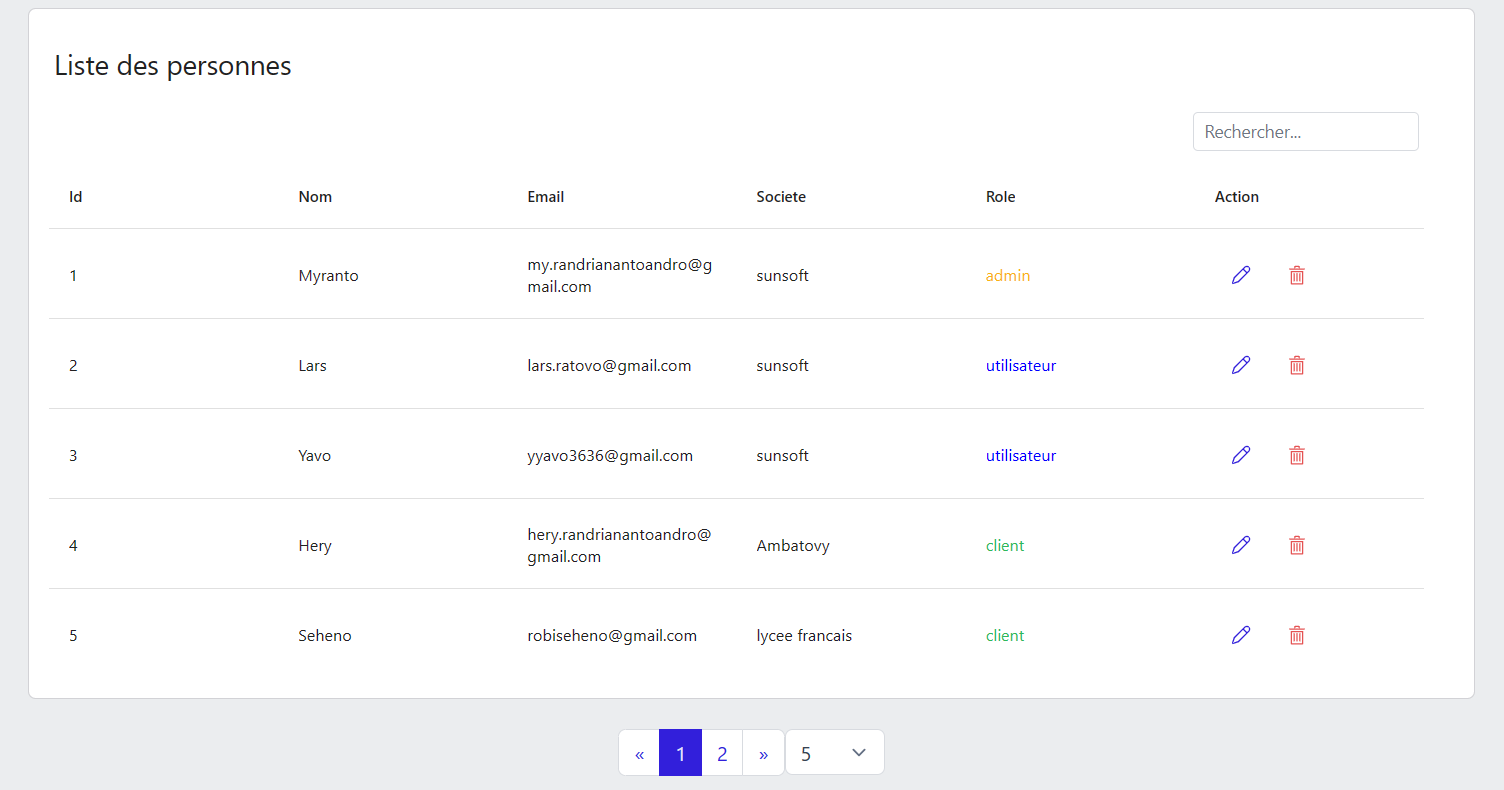


Figure 14 : Liste des personnes (profil admin)

# Gestion des plannings

Au sein de l’application, le module de gestion du planning joue un rôle fondamental pour garantir un bon suivi des opérations. Lors de la création d'un ticket, les utilisateurs ont la possibilité de définir une plage de dates (début et fin), ce qui permet de planifier et organiser nos activités de manière optimale.

Un calendrier intégré affiche les événements, c'est-à-dire les tickets à résoudre, que ce soit sur une base quotidienne, hebdomadaire ou mensuelle. Cela permet un suivi rigoureux et une planification précise des tickets. Chaque événement est accessible en un simple clic, offrant un accès rapide aux détails du ticket. Cette fonctionnalité permet aux responsables et aux agents de rester informés en permanence sur l'état d'avancement de chaque ticket, contribuant ainsi à une gestion plus proactive.

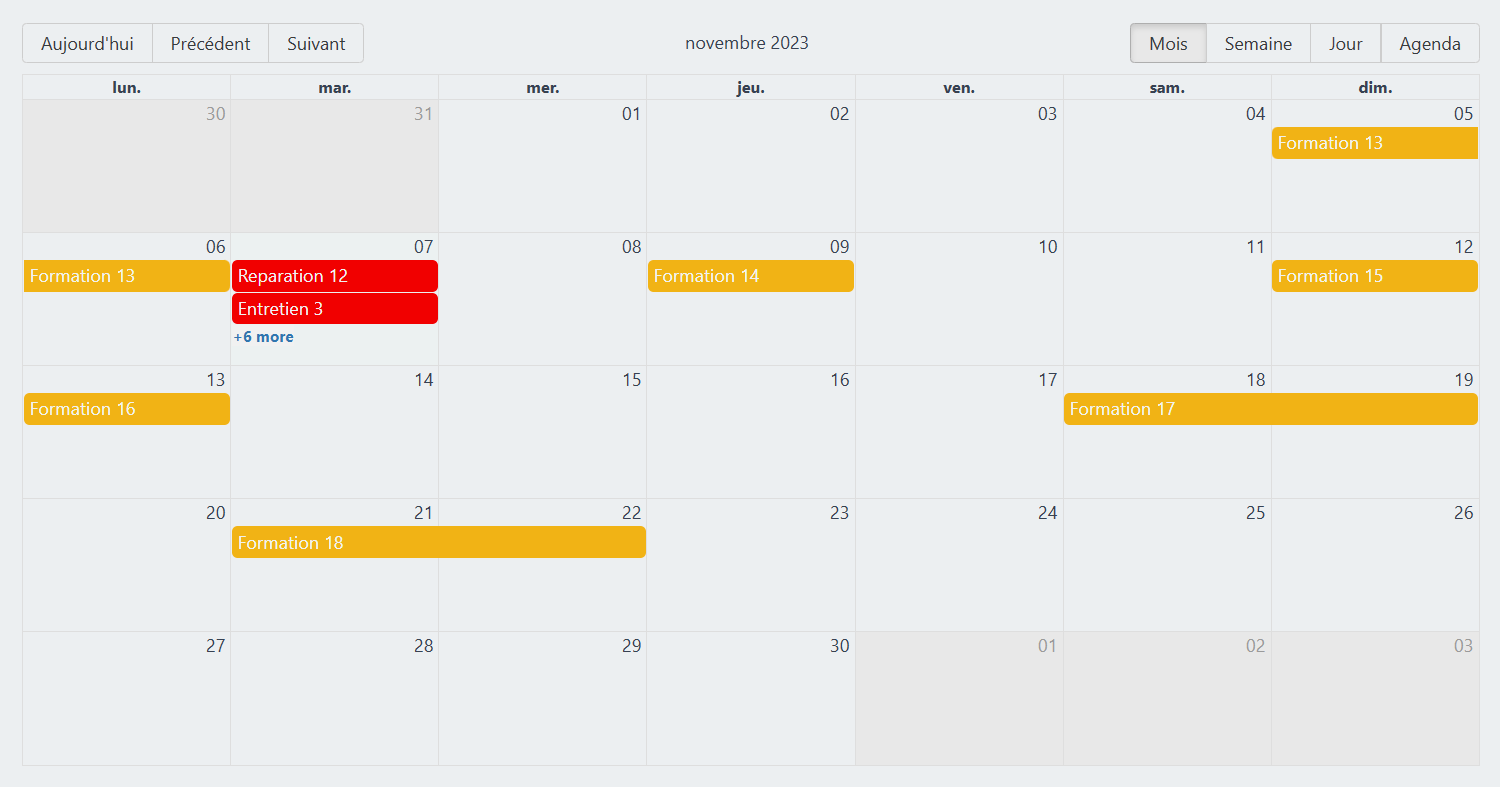


Figure 15 : Calendrier des tickets

Dans un souci de ponctualité et de non-respect des échéances, le système est équipé d'une fonction automatisée, "checkTicketDelay", qui surveille constamment les délais des tickets. En cas de retard constaté, le système réaffecte automatiquement le ticket à la date actuelle, alertant ainsi le responsable de la tâche qu'elle n'a pas encore été achevée. De plus, si un agent n'est pas connecté à l'application au moment de cette réaffectation, les tickets en retard seront alors représentés en **rouge** dans le calendrier. Un email d'alerte lui est automatiquement envoyé. Cette double approche garantit que les tâches sont traitées en temps voulu.

Pour une gestion plus précise, le calendrier permet également de filtrer les événements en fonction de leur type de service, offrant la possibilité d'afficher uniquement les événements liés à des services spécifiques. Cette fonctionnalité s'avère particulièrement utile pour une organisation plus ciblée des activités.

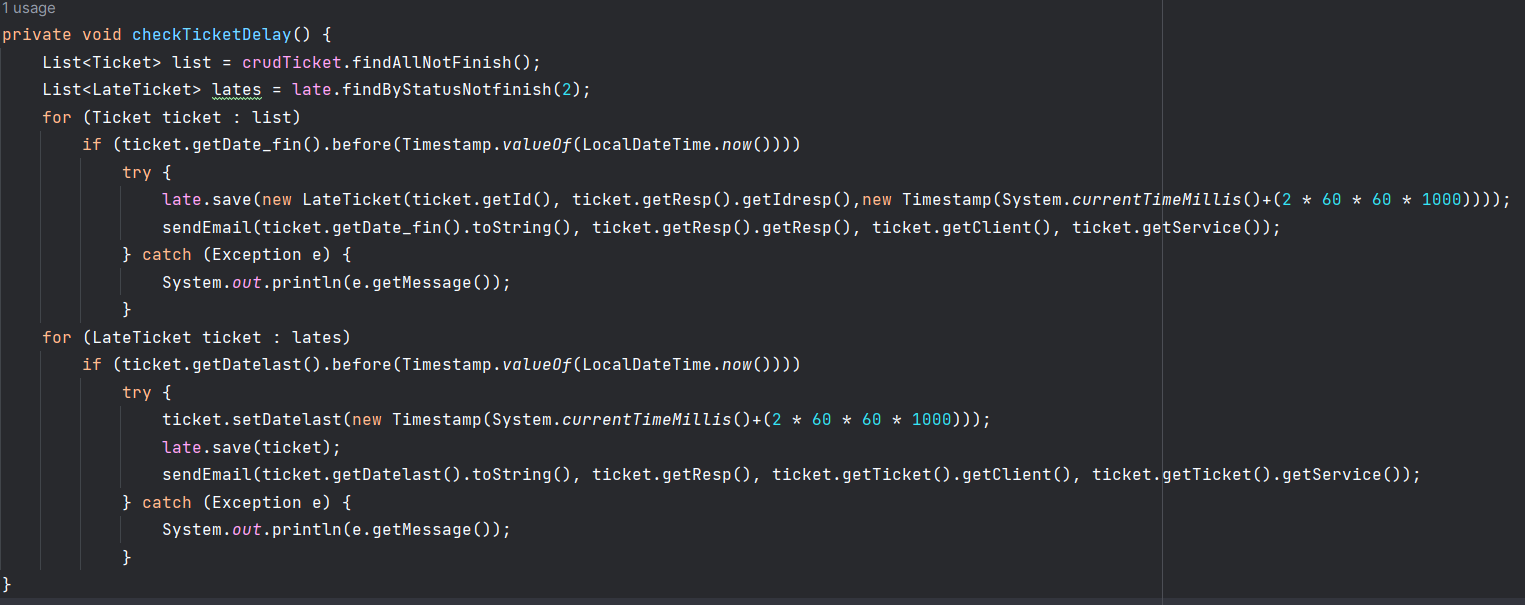


Figure 16 : Code de la fonction checkTicketDelay

Ici, nous avons deux types de liste, une List de Ticket nommé liste qui est la liste des tickets qui ne sont pas encore fini et qui ne figurent pas dans la table lateticket (qui est la table où sont stocker tous les tickets en retard encore non résolu) et une de LateTicket nommé lates (qui est la liste des tickets qui sont en retard depuis quelque temps déjà),

Dans les deux boucles, nous vérifions si la date d’échéance de chaque élément est passé par rapport a la date actuelle, si c’est le cas :

* Pour list, on sauvegardera un nouvel enregistrement LateTicket dans la table lateticket, puis enverra un mail d’alerte a l’agent responsable correspondant.
* Pour lates, on modifiera juste la date d’échéance en y ajoutant deux heures de plus pour que l’agent ait le temps d’agir, puis en lui envoie un mail d’alerte.

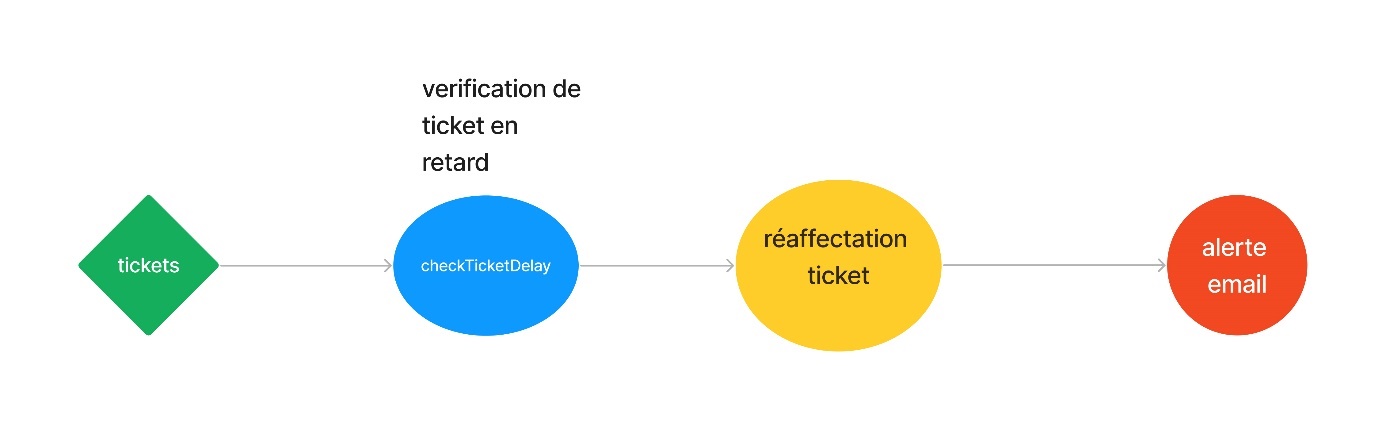


Figure 17 : Illustration de l’activité de checkTicketDelay

# Synchronisation

L’application offre deux modes de synchronisation, à savoir la **synchronisation automatique** et la **synchronisation manuelle**. L'application mobile est principalement conçue pour être utilisée en mode en ligne. Cependant, même en cas de problème de connexion, les utilisateurs peuvent toujours bénéficier de la pleine fonctionnalité de l'application en mode hors ligne., garantissant ainsi une expérience fluide, quelles que soient les conditions de connectivité.

La synchronisation dans notre application repose sur un système bidirectionnel. Cela signifie que l'application mobile est capable d'envoyer des données vers l'application Web, mais elle est également en mesure de recevoir des données de cette dernière. Pour garantir l'intégrité des données et éviter toute ambiguïté, nous avons adopté une stratégie d'indexation basée sur un identifiant universel unique.

Dans le cadre de la synchronisation automatique, notre application effectuera automatiquement des vérifications à chaque action entreprise par l'utilisateur pour déterminer si une connexion Internet est disponible. Si c'est le cas, la synchronisation des données sera immédiatement déclenchée, sinon rien ne se passe.

Pour la synchronisation manuelle, l'utilisateur dispose de la liberté de déclencher le processus en appuyant sur le bouton de synchronisation. Si, à ce moment-là, une connexion Internet est accessible, la synchronisation aura lieu comme prévu. Dans le cas contraire, un message d'erreur sera affiché pour informer l'utilisateur de la nécessité d'une connexion Internet pour mener à bien cette opération.

Cette approche de synchronisation bidirectionnelle, avec une gestion flexible de la synchronisation automatique et manuelle, vise à offrir une expérience utilisateur transparente et à s'assurer que les données sont toujours à jour et cohérentes, quelles que soient les circonstances

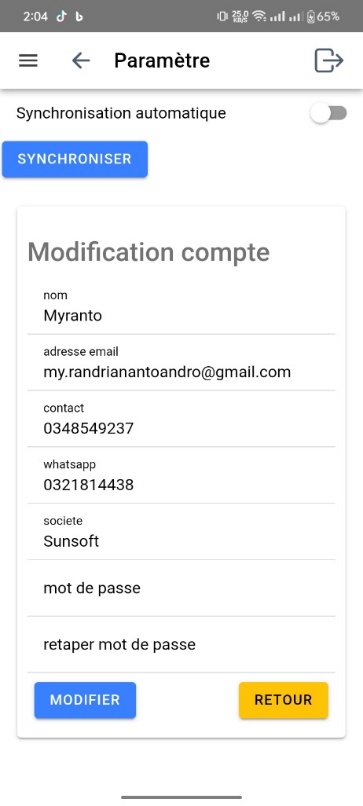


Figure 18 : Synchronisation

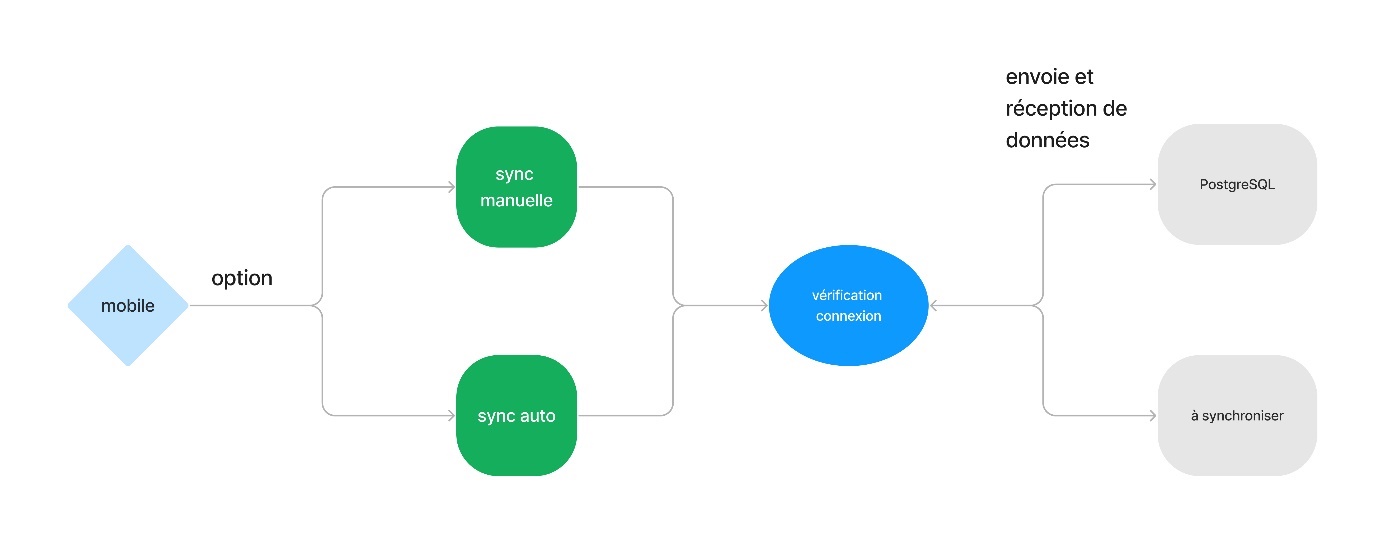


Figure 19 : Scénario de synchronisation

# Facturation

Ce module a été conçu pour rationaliser le processus de gestion des factures au sein du système. Il offre un ensemble de fonctionnalités avancées pour simplifier la création, le suivi et la gestion des factures clients. Ce module permet de regrouper les tickets terminés qui n'ont pas encore été facturés, d'organiser ces données par mois, semaine ou jour, et de générer des factures de manière efficace.

* Pour générer des factures, il suffit de sélectionner le groupe de tickets à facturer, puis de définir la date de facturation et les tarifs des services (pour des raisons de confidentialité). Cette approche offre une flexibilité exceptionnelle en permettant aux administrateurs de personnaliser chaque facture en fonction des services rendu.
* Si nécessaire, le module permet également de créer une facture pour un seul ticket. Cela donne à l'utilisateur un contrôle total sur le processus de facturation.

Une fois les factures créées, elles sont stockées dans une liste accessible depuis une autre page. L'utilisateur que ce soit un client, un agent ou un administrateur peut accéder aux détails de chaque facture (le client n’aura accès qu’à ces propres factures) et générer des fichiers PDF.

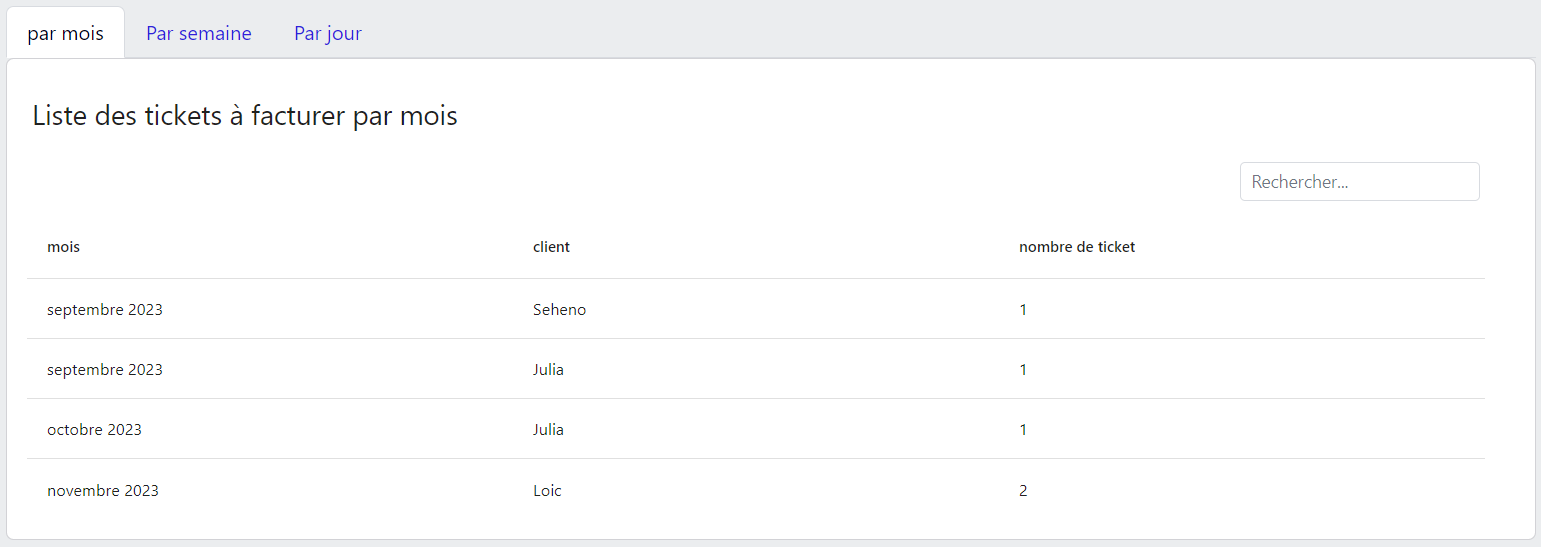


Figure 20 : Liste des tickets à facturer selon son regroupement

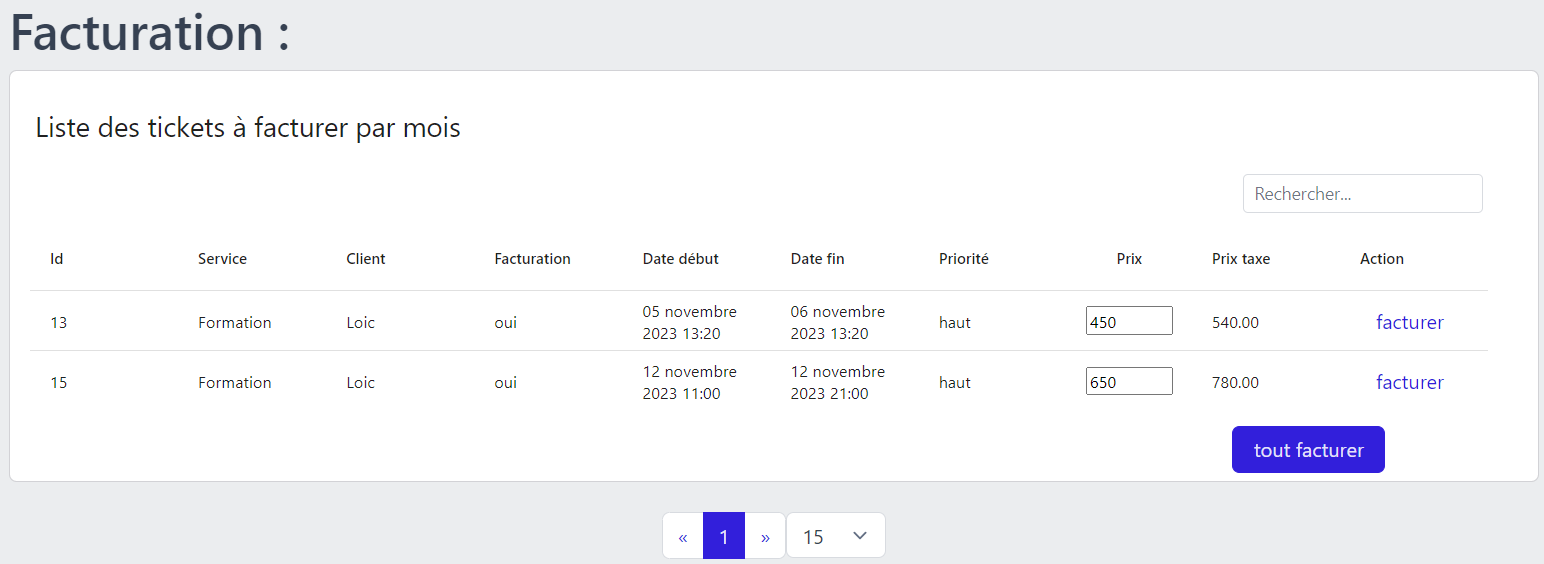


Figure 21 : Liste des tickets facturables

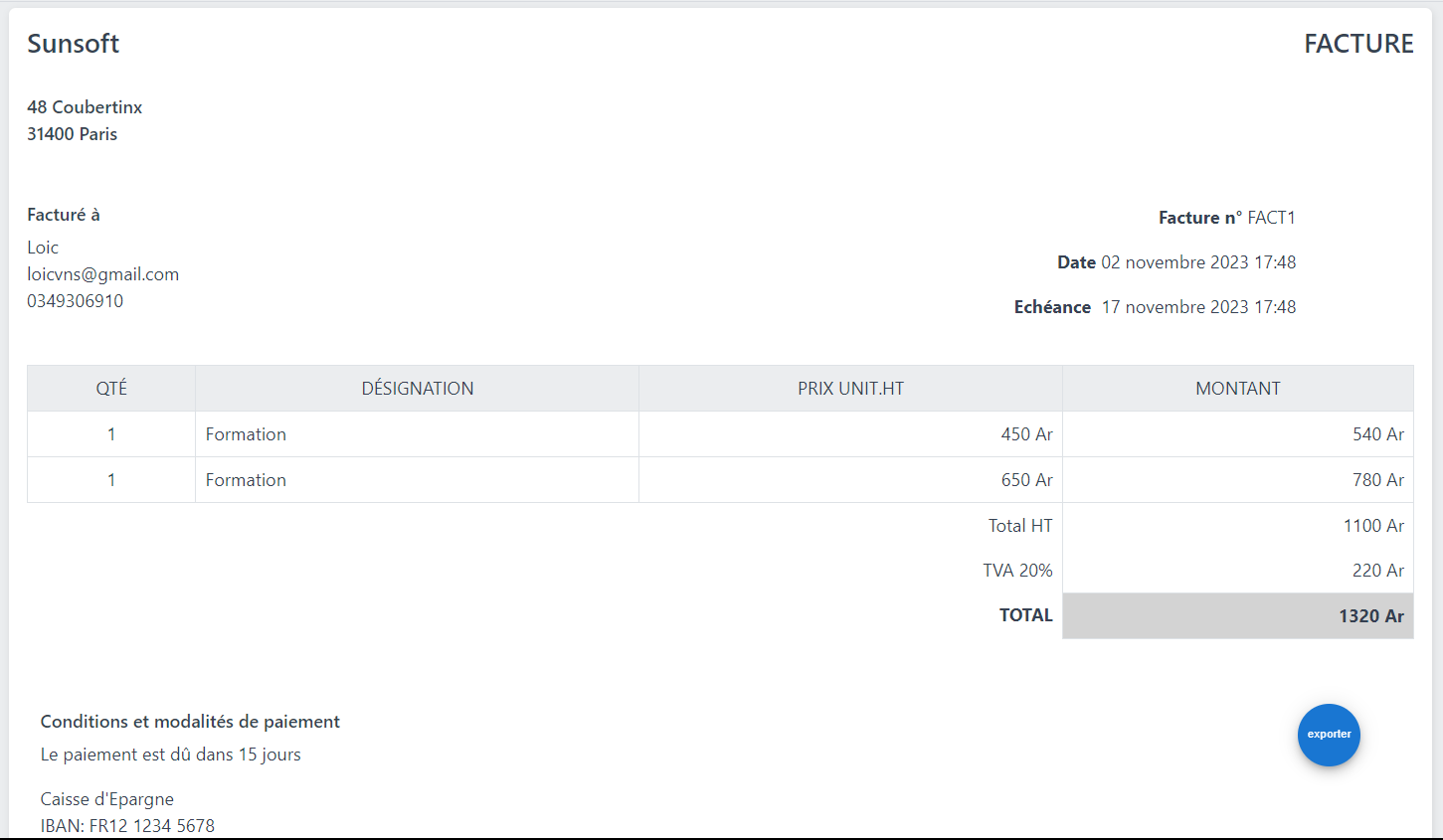


Figure 22 : Détail facture

# État d’analyse et statistique

Dans le tableau de bord de notre application, nous avons quelques statistiques pouvant influencer les prochaines décisions de l’utilisateur concernant les tickets.

# Analyse globale

L’application commence par présenter un aperçu du pourcentage total de tickets réalisés, permettant une ventilation détaillée de la ponctualité de leur réalisation. Vous avez la possibilité de distinguer les tickets réalisés en temps opportun de ceux ayant subi des retards. En outre, les administrateurs peuvent examiner les performances individuelles des agents responsables des tickets. Vous pouvez accéder au pourcentage de réalisation des tickets par agent, au nombre total de tickets réalisés, et comparer leur ponctualité par rapport aux retards éventuels. Ces statistiques sont facilement filtrables par mois et par année, vous permettant de prendre des décisions éclairées.

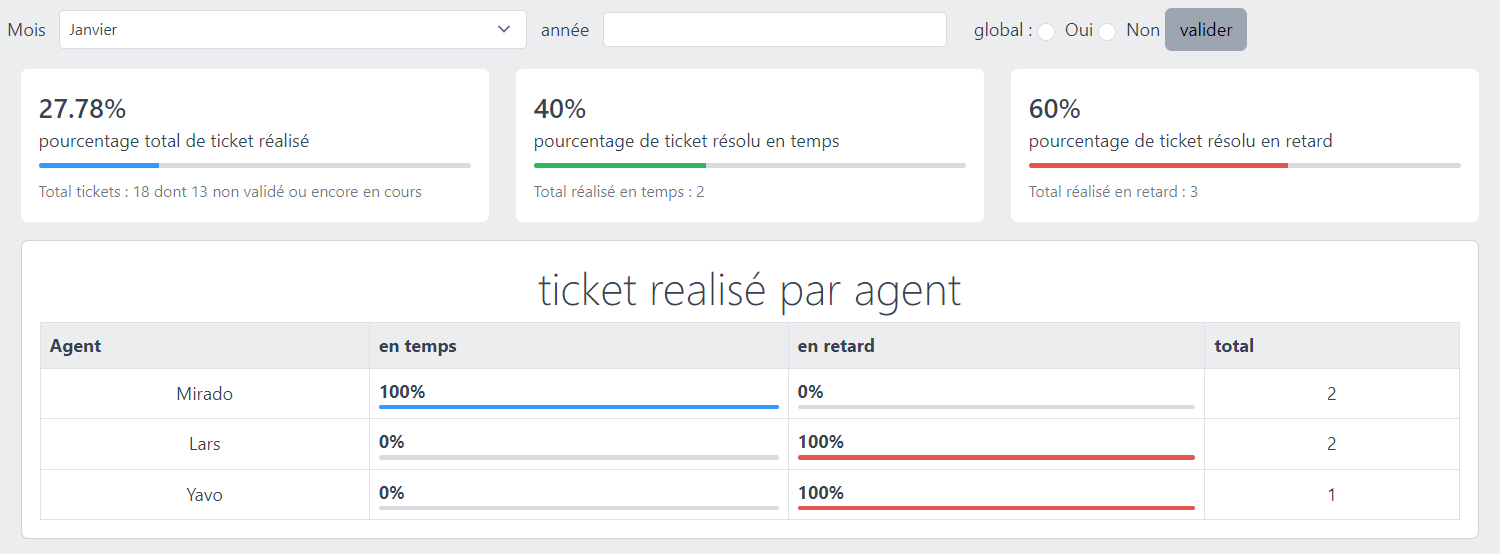


Figure 23 : Pourcentage des tickets résolus avec détail agents

# Analyse de l’État des tickets

Une analyse du nombre de tickets en cours, en attente et terminés par mois est également disponible. Cette analyse permet de suivre l'évolution de la charge de travail au fil du temps. Par exemple, une augmentation du nombre de tickets en attente peut signaler des problèmes de traitement ou de ressources, incitant à des mesures correctives.

# Performance temporelle

L'application fournit des informations essentielles sur la performance temporelle de votre équipe en charge de la réalisation des tickets. Vous pouvez observer le pourcentage de tickets réalisés le matin et l'après-midi, ce qui permet de déterminer les moments de la journée où l'équipe est la plus productive. De plus, cette fonctionnalité offre une vue détaillée des performances individuelles des agents, vous permettant d'ajuster leurs heures de travail en fonction de leurs performances respectives.



Figure 24 : État et performance temporelle

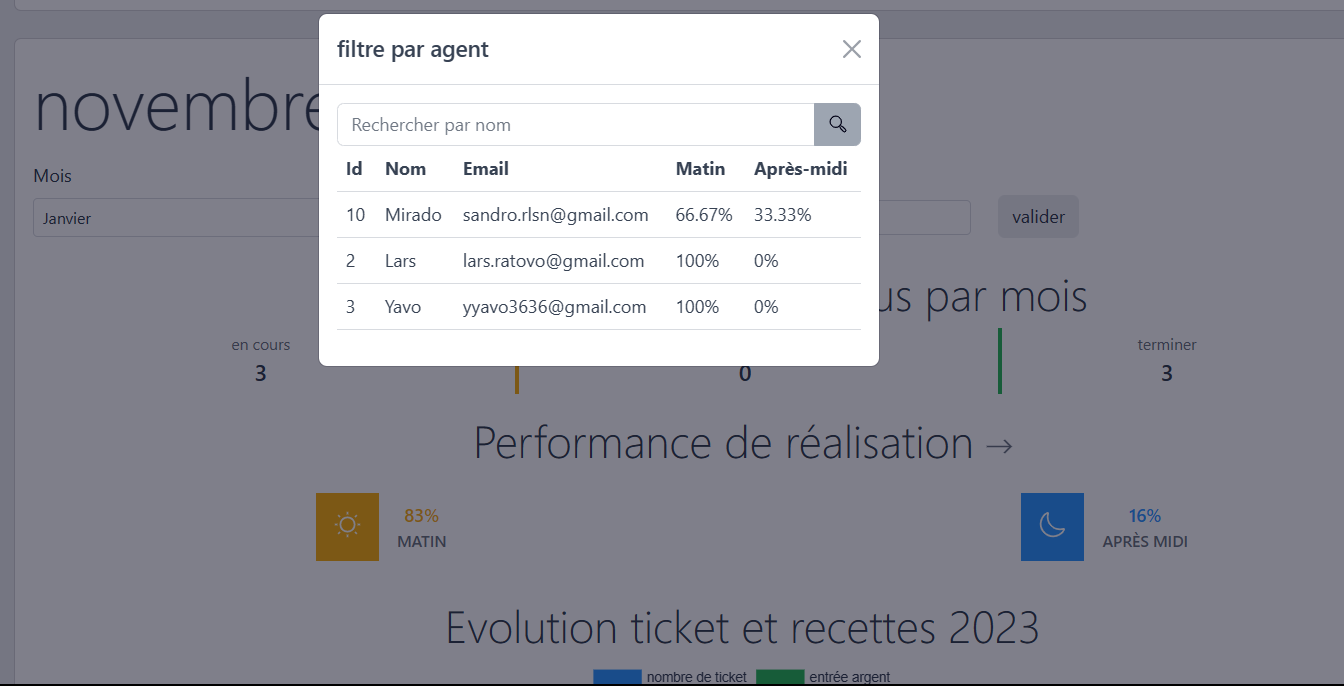


Figure 25 : Performance temporelle par agent

# Évolution des tickets et des recettes

Notre application met à disposition un graphique illustrant l'évolution du nombre de tickets reçus chaque mois et chaque année. Cela vous permet de repérer les périodes de hausse ou de baisse des demandes, ce qui peut orienter la mise en place de stratégies commerciales. De plus, vous pouvez suivre l'évolution des recettes générées par le traitement des tickets. Ces informations vous offrent une vue globale de votre activité et vous aident à décider, par exemple, quand intensifier la promotion de vos services.

Notre tableau de bord offre une analyse professionnelle complète de vos opérations, permettant des décisions éclairées et une gestion plus efficace de votre activité.

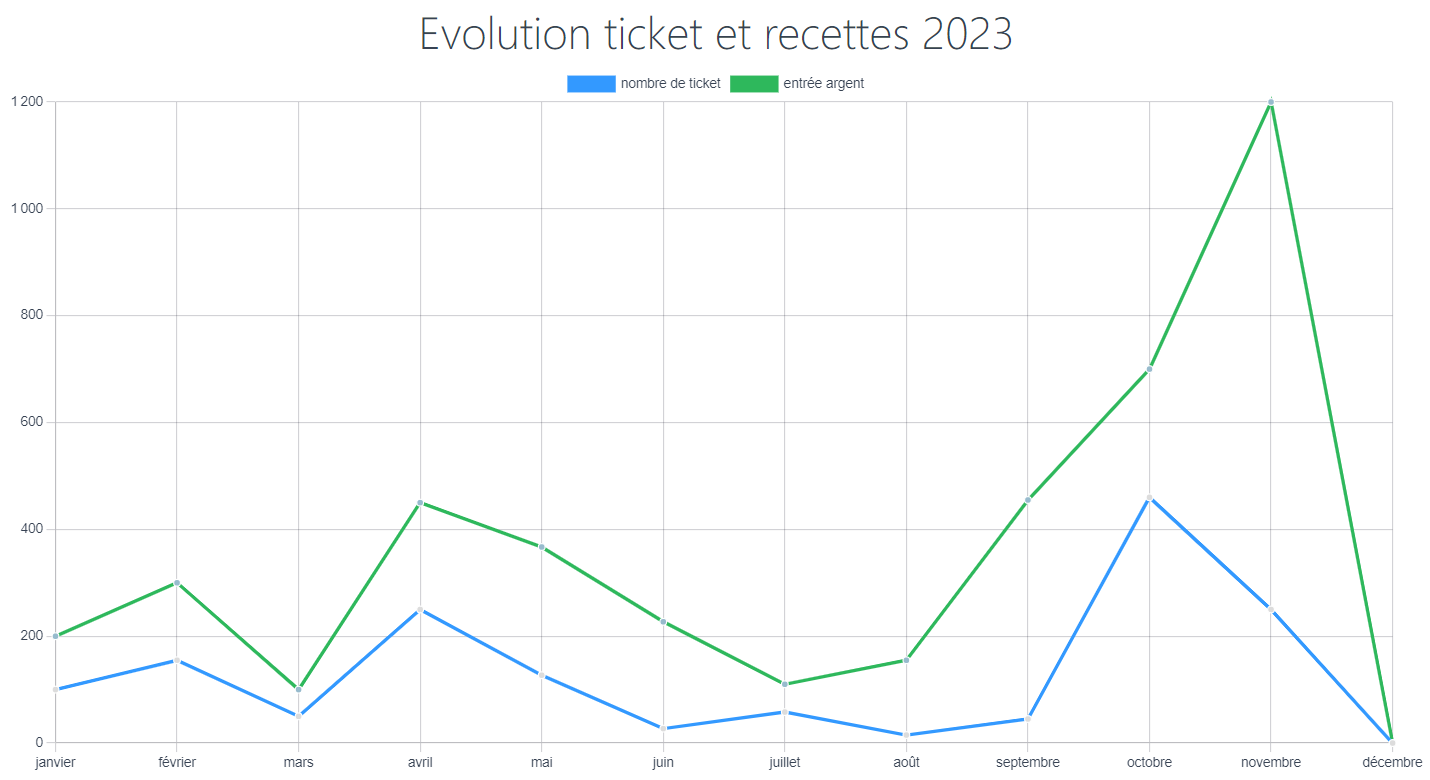


Figure 26 : Évolutions d’entrée des recettes et des tickets

# Problèmes rencontrés et solutions

Synchronisation table :

Lors de la conception de la base de données, je n'ai pas pris en compte les clés primaires de chaque table pour qu'elles soient universelles dans les deux bases de données, PostgreSQL et SQLite. Cela aurait permis d'éviter les conflits entre les données lors de la synchronisation. J'ai donc dû modifier chaque table à synchroniser en ajoutant une colonne 'common', unique.

Pour PostgreSQL, cette colonne est simplement une colonne de type varchar générée automatiquement par concaténation via une séquence.

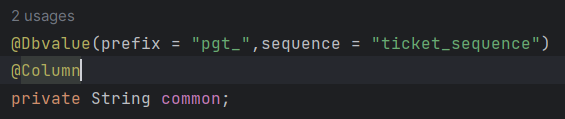


Figure 27 : Assignation de la valeur de common

Cependant, étant donné que SQLite est présent sur chaque appareil mobile, appliquer le même concept que pour PostgreSQL pourrait entraîner des conflits de données provenant de sources SQLite. C'est pourquoi j'ai décidé de générer un identifiant unique en utilisant la formule suivante : « 'sqt' || ${now} || lower(hex(randomblob(10))) ».

* 'sqt' est juste un préfixe concaténé aux autres variables.
* 'now' est la date actuelle convertie en secondes.
* 'randomblob(10)' génère un blob de 10 octets contenant des bytes pseudo-aléatoires.
* 'hex()' interprète son argument comme un BLOB [[9]](#footnote-9)et renvoie une chaîne qui est la représentation hexadécimale en majuscules du contenu de ce blob.

Ainsi, "hex(randomblob(10))" génère une chaîne hexadécimale unique à partir d'un blob pseudo-aléatoire de 10 octets. De cette manière, je m'assure que la valeur de "common" sera toujours unique parmi plusieurs enregistrements de données.

# Évaluation du projet et connaissances acquises

# Bilan de l’entreprise

# Bilan personnel

Mon stage au sein de Sunsoft m'a ouvert les portes du monde professionnel et m'a offert de précieuses expériences qui ont renforcé mes compétences techniques. Travailler sur ce projet dans un environnement réel en dehors de l'IT University a été une opportunité de croissance personnelle, m'incitant à porter une attention minutieuse aux moindres détails du projet sur lequel j'ai travaillé.

En outre, cette expérience m'a permis d'approfondir et d'améliorer mes connaissances tout au long du stage. J'ai su faire preuve de maturité dans mon approche du travail, comparativement à ma manière de gérer les projets à l'université.

# Extension et évolution du projet

**Conclusion**

**Bibliographie**

**Annexe**

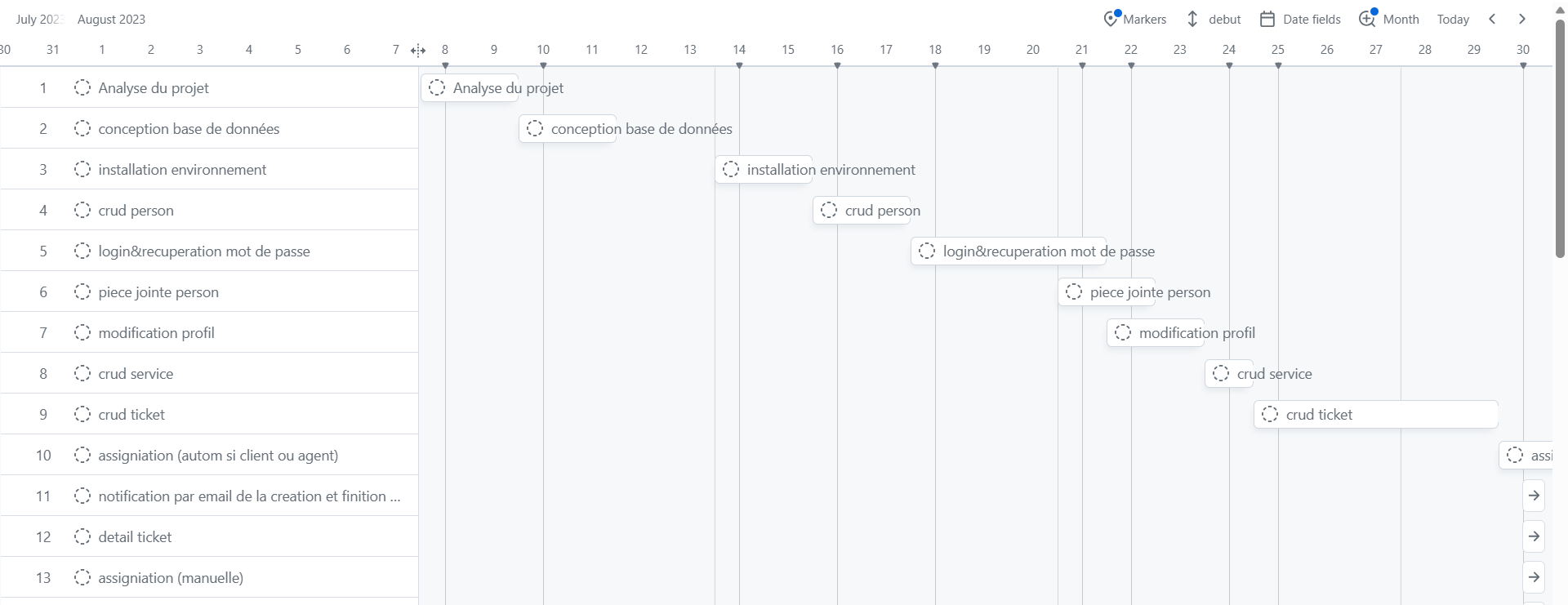


Figure 28 : Planning de réalisation (1)

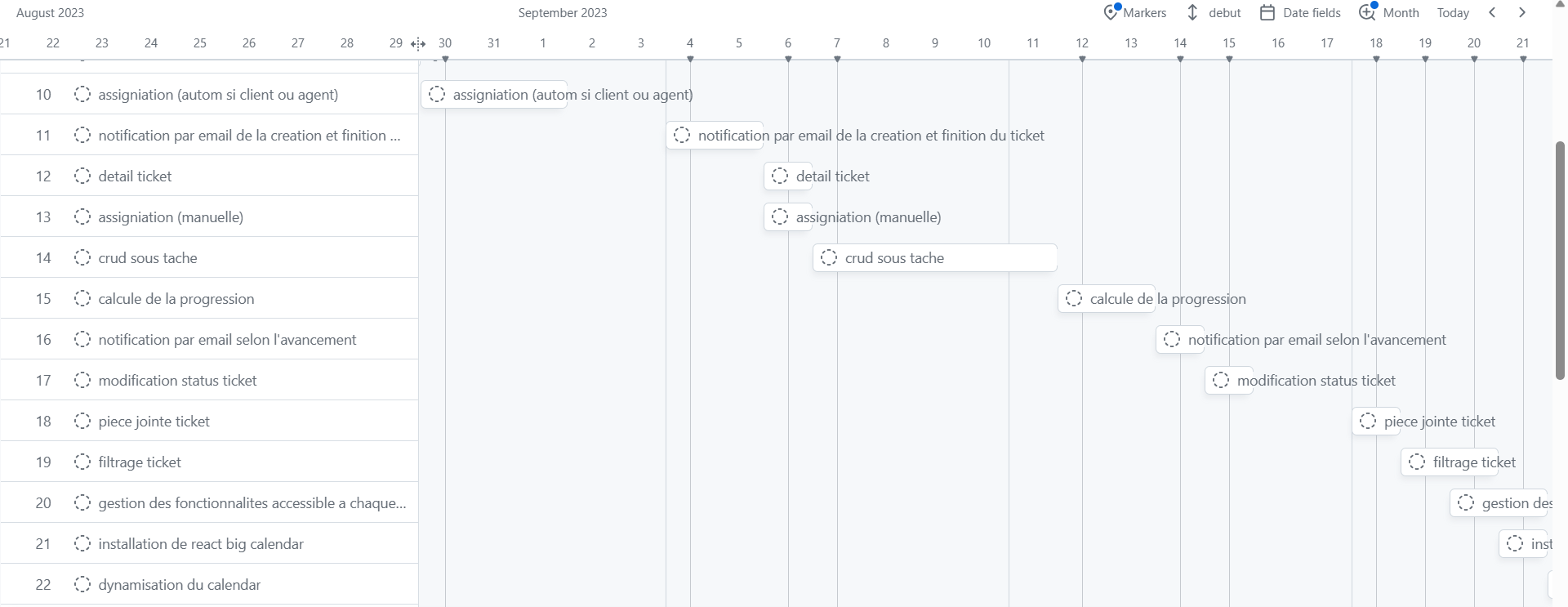


Figure 29 : Planning de réalisation (2)

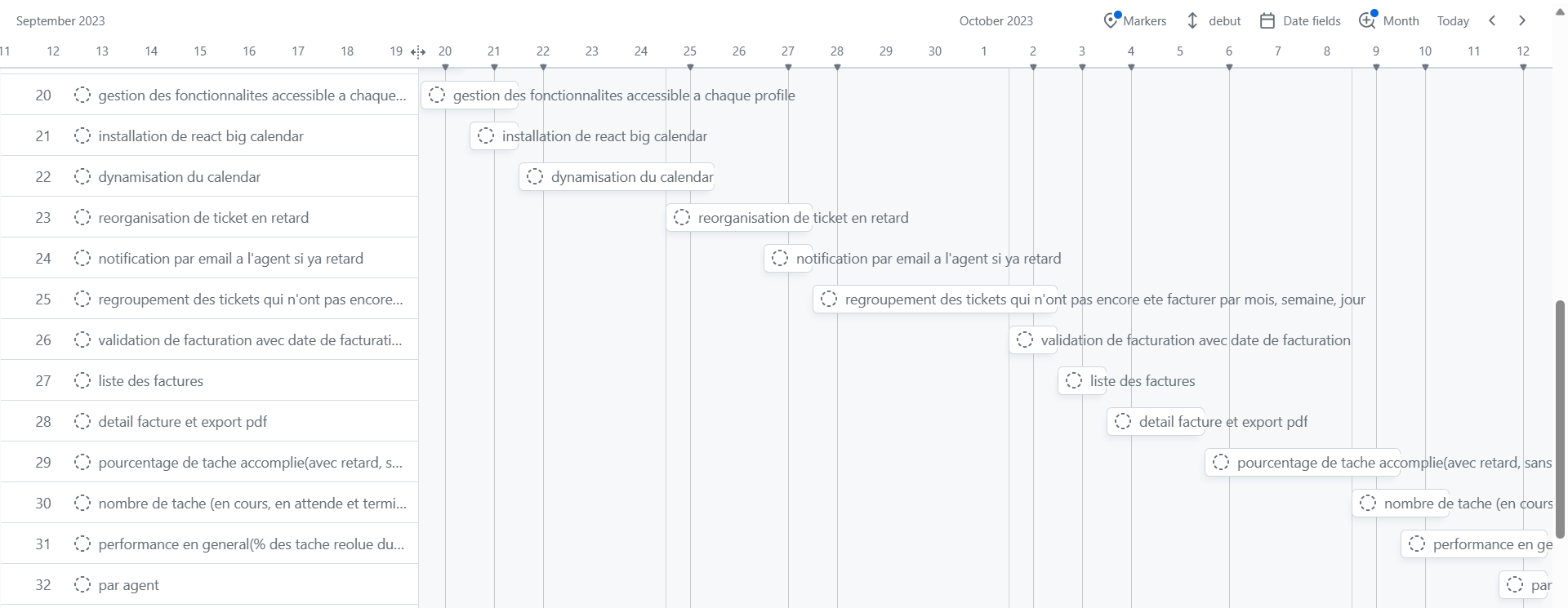


Figure 30 : Planning de réalisation (3)

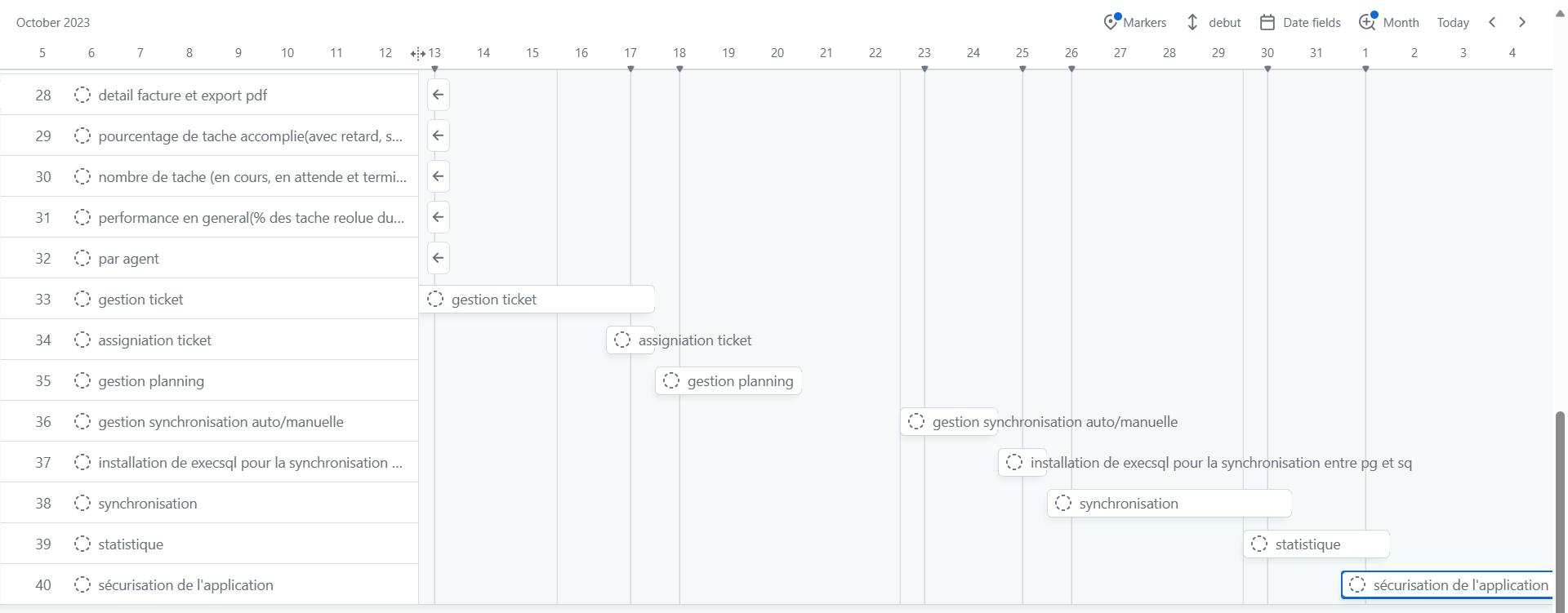


Figure 31 : Planning de réalisation (4)

1. TIC : Technologies de l’Information et de la Communication [↑](#footnote-ref-1)
2. ACID, Atomicité, Cohérence, Isolation, Durabilité [↑](#footnote-ref-2)
3. SGBD, Système de Gestion de Base de Données [↑](#footnote-ref-3)
4. Forum, Communauté où on s’échange des informations [↑](#footnote-ref-4)
5. JPA, Java Persistence API [↑](#footnote-ref-5)
6. API, Application Programming Interface [↑](#footnote-ref-6)
7. CRUD, Create Read Update Delete [↑](#footnote-ref-7)
8. SMTP, Simple Mail Transfert Protocol [↑](#footnote-ref-8)
9. BLOB, Binary Large Object [↑](#footnote-ref-9)