RÉPUBLIQUE TUNISIENNE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Institut Supérieur d'Informatique
et des Technologies de Communication

Projet de Fin d'Études

Pour l'obtention du
Diplôme de Licence en Informatique

Système de Gestion des Fonds Bancaires

Amen Bank

Réalisé par :
Prénom NOM

Encadré par :
M./Mme. Encadreur

Année Universitaire 2024-2025

Dédicace

À mes parents,

Pour leur soutien inconditionnel et leurs encouragements

À mes enseignants,

Pour leur dévouement et leur patience

À tous ceux qui m'ont accompagné dans ce parcours

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce projet de fin d'études.

Mes remerciements s'adressent en premier lieu à mon encadreur académique, **M./Mme. [Nom]**, pour ses conseils précieux, sa disponibilité et son soutien tout au long de ce travail.

Je remercie également **Amen Bank**, et particulièrement l'équipe de la Direction des Systèmes d'Information, pour m'avoir accueilli au sein de leur établissement et pour la confiance qu'ils m'ont accordée en me confiant ce projet stratégique.

J'adresse également mes sincères remerciements à **[Nom du tuteur entreprise]**, mon encadreur professionnel chez Amen Bank, pour son accompagnement, ses orientations techniques et sa disponibilité malgré ses nombreuses responsabilités.

Je n'oublie pas de remercier le corps professoral de l'Institut Supérieur d'Informatique et des Technologies de Communication pour la qualité de l'enseignement dispensé tout au long de mon cursus.

Enfin, mes remerciements vont à ma famille et mes amis pour leur soutien moral et leurs encouragements constants qui m'ont permis de mener à bien ce projet.

Résumé

Dans le cadre de la transformation digitale des processus bancaires, ce projet de fin d'études présente la conception et la réalisation d'un **Système de Gestion des Fonds Bancaires** pour Amen Bank. Face aux défis de la gestion manuelle des mouvements de fonds inter-agences (provisionnement et versement), qui engendre des risques opérationnels, des délais prolongés et une traçabilité limitée, nous avons développé une solution web complète et sécurisée.

Le système mis en place permet d'automatiser l'intégralité du cycle de traitement des demandes de fonds : de la soumission par les agences, en passant par la validation de la Caisse Centrale, l'assignment des équipes de sécurité, jusqu'au dispatch et à la réception finale. L'application implémente un workflow multiniveaux avec un contrôle d'accès basé sur les rôles (RBAC), garantissant la sécurité et la traçabilité de chaque opération.

Sur le plan technique, nous avons adopté une architecture full-stack moderne basée sur **Next.js 15**, avec **PostgreSQL** comme système de gestion de base de données, **Prisma ORM** pour la couche d'accès aux données, et **NextAuth.js** pour l'authentification sécurisée. Le développement a suivi la méthodologie **Agile/Scrum**, réparti en quatre sprints de deux à trois semaines chacun.

Le système offre également un tableau de bord analytique permettant aux administrateurs de suivre les indicateurs clés de performance (KPIs), les tendances et les volumes de transactions, facilitant ainsi la prise de décision stratégique.

Mots-clés : Gestion des fonds bancaires, Workflow automatisé, RBAC, Next.js, PostgreSQL, Prisma, Authentification sécurisée, Scrum, Traçabilité, Analytics

Abstract

As part of the digital transformation of banking processes, this end-of-studies project presents the design and implementation of a **Banking Funds Management System** for Amen Bank. Facing the challenges of manual management of inter-agency fund movements (provisioning and remittance), which generates operational risks, prolonged delays and limited traceability, we have developed a complete and secure web solution.

The implemented system automates the entire fund request processing cycle : from submission by agencies, through validation by the Central Cash Department, security team assignment, to dispatch and final reception. The application implements a multi-level workflow with Role-Based Access Control (RBAC), ensuring security and traceability of each operation.

On the technical side, we adopted a modern full-stack architecture based on **Next.js 15**, with **PostgreSQL** as the database management system, **Prisma ORM** for the data access layer, and **NextAuth.js** for secure authentication. Development followed the **Agile/Scrum** methodology, divided into four sprints of two to three weeks each.

The system also provides an analytical dashboard allowing administrators to monitor key performance indicators (KPIs), trends and transaction volumes, thus facilitating strategic decision-making.

Keywords : Banking funds management, Automated workflow, RBAC, Next.js, PostgreSQL, Prisma, Secure authentication, Scrum, Traceability, Analytics

Table des matières

Table des figures

Liste des tableaux

Liste des Acronymes

Acronyme	Signification
API	Application Programming Interface (Interface de Programmation d'Application)
CRUD	Create, Read, Update, Delete (Créer, Lire, Mettre à jour, Supprimer)
JWT	JSON Web Token (Jeton Web JSON)
ORM	Object-Relational Mapping (Mappage Objet-Relationnel)
RBAC	Role-Based Access Control (Contrôle d'Accès Basé sur les Rôles)
SSR	Server-Side Rendering (Rendu Côté Serveur)
UML	Unified Modeling Language (Langage de Modélisation Unifié)
RGPD	Règlement Général sur la Protection des Données
KPI	Key Performance Indicator (Indicateur Clé de Performance)
MoSCoW	Must have, Should have, Could have, Won't have
HTTP	HyperText Transfer Protocol (Protocole de Transfert HyperTexte)
HTTPS	HTTP Secure (HTTP Sécurisé)
SQL	Structured Query Language (Langage de Requête Structuré)
REST	Representational State Transfer
JSON	JavaScript Object Notation
BD	Base de Données
SGBD	Système de Gestion de Base de Données
UI	User Interface (Interface Utilisateur)
UX	User Experience (Expérience Utilisateur)
CSS	Cascading Style Sheets (Feuilles de Style en Cascade)
HTML	HyperText Markup Language (Langage de Balisage HyperTexte)
JS	JavaScript
TS	TypeScript
npm	Node Package Manager (Gestionnaire de Paquets Node)
IDE	Integrated Development Environment (Environnement de Développement Intégré)
CI/CD	Continuous Integration/Continuous Deployment
ACID	Atomicity, Consistency, Isolation, Durability
PDF	Portable Document Format
PNG	Portable Network Graphics
URL	Uniform Resource Locator (Localisateur Uniforme de Ressource)

Introduction Générale

Contexte

La transformation digitale représente aujourd'hui un enjeu stratégique majeur pour le secteur bancaire tunisien. Dans un environnement économique en constante évolution, les établissements bancaires sont contraints d'optimiser leurs processus opérationnels pour améliorer leur efficacité, réduire les risques et offrir une meilleure qualité de service.

Amen Bank, l'une des principales banques tunisiennes avec un réseau étendu d'agences à travers le pays, fait face à des défis importants dans la gestion quotidienne des mouvements de fonds entre ses différentes entités. Les opérations de provisionnement (transfert de fonds des agences vers la Caisse Centrale) et de versement (transfert de fonds de la Caisse Centrale vers les agences) constituent des activités critiques qui nécessitent une coordination rigoureuse entre plusieurs départements : les agences, la Direction de la Caisse Centrale et la Direction de la Sécurité (Tunisie Sécurité).

Problématique

Actuellement, la gestion de ces mouvements de fonds repose largement sur des processus manuels et des documents papier. Cette approche traditionnelle présente plusieurs limitations significatives :

— **Risques opérationnels élevés** : Les processus manuels sont sujets aux erreurs humaines, aux pertes de documents et aux incohérences dans les informations.

— **Délais prolongés** : Le circuit de validation et d'approbation multiniveaux nécessite des échanges multiples de documents physiques, entraînant des retards considérables dans le traitement des demandes.

— **Traçabilité limitée** : L'absence de système centralisé rend difficile le suivi en temps réel des demandes et la constitution d'un historique complet des opérations.

— **Difficulté d'audit** : La reconstruction a posteriori du parcours d'une demande et l'identification des responsabilités sont complexes sans un système de logs automatisé.

— **Absence d'indicateurs de performance** : L'impossibilité d'extraire des statistiques et des KPIs limite la capacité de la direction à prendre des décisions éclairées et à optimiser les processus.

Ces problèmes s'amplifient avec l'augmentation du volume des transactions et la croissance du réseau d'agences, rendant impératif le développement d'une solution digitale adaptée.

Objectifs du Projet

Dans ce contexte, l'objectif principal de ce projet de fin d'études est de concevoir et développer un **Système de Gestion des Fonds Bancaires** moderne et sécurisé qui répond aux besoins d'Amen Bank. Les objectifs spécifiques sont les suivants :

1. **Automatiser le cycle complet de traitement** : Depuis la soumission d'une demande par une agence jusqu'à la confirmation de réception des fonds, en passant par toutes les étapes intermédiaires de validation, d'assignment et de dispatch.
2. **Mettre en place un workflow sécurisé** : Implémenter un système de contrôle d'accès basé sur les rôles (RBAC) garantissant que chaque acteur n'a accès qu'aux fonctionnalités pertinentes pour son rôle.
3. **Assurer la traçabilité complète** : Enregistrer automatiquement toutes les actions effectuées sur chaque demande, avec identification de l'acteur et horodatage précis.
4. **Faciliter la prise de décision** : Fournir aux administrateurs des tableaux de bord analytiques avec des indicateurs de performance, des tendances et des visualisations graphiques.
5. **Améliorer l'efficacité opérationnelle** : Réduire les délais de traitement, minimiser les erreurs et optimiser l'allocation des ressources logistiques.

Solution Proposée

Pour atteindre ces objectifs, nous proposons le développement d'une application web full-stack moderne basée sur les technologies suivantes :

— **Next.js 15** : Framework React pour le développement full-stack avec Server-Side Rendering

— **PostgreSQL** : Système de gestion de base de données relationnelle robuste

— **Prisma ORM** : Couche d'abstraction pour l'accès aux données avec type-safety

— **NextAuth.js** : Solution d'authentification sécurisée avec JWT

— **TypeScript** : Langage typé pour une meilleure maintenabilité

— **Tailwind CSS** : Framework CSS pour une interface moderne et responsive

L'application implémente un workflow complet avec quatre rôles principaux : Administrateur, Agence, Caisse Centrale et Tunisie Sécurité. Chaque rôle dispose d'interfaces et de permissions spécifiques, permettant une séparation claire des responsabilités.

Méthodologie de Travail

Le développement du projet suit la méthodologie **Agile/Scrum**, avec une organisation en quatre sprints de deux à trois semaines chacun :

— **Sprint 1** : Architecture du système et module d'authentification

— **Sprint 2** : Module de gestion des demandes (création et consultation)

— **Sprint 3** : Module de validation et d'assignment des équipes

— **Sprint 4** : Module de dispatch, réception et analytics

Cette approche itérative permet un développement progressif avec des validations régulières et une adaptation continue aux besoins identifiés.

Organisation du Rapport

Ce rapport est structuré en six chapitres principaux :

Chapitre 1 : Cadre Général du Projet présente l'organisme d'accueil, le cadrage du projet, l'étude de l'existant et la méthodologie de travail adoptée.

Chapitre 2 : État de l'Art expose les concepts métiers bancaires, les technologies utilisées et les choix architecturaux.

Chapitre 3 : Sprint 1 détaille la mise en place de l'architecture et du système d'authentification.

Chapitre 4 : Sprint 2 présente le développement du module de gestion des demandes.

Chapitre 5 : Sprint 3 décrit l'implémentation du workflow de validation et d'assignment.

Chapitre 6 : Sprint 4 expose la réalisation des modules de dispatch, réception et analytics.

Enfin, une conclusion générale synthétise les réalisations, les compétences acquises et les perspectives d'évolution du système.

Chapitre 1

Cadre Général du Projet

1.1 Introduction

Ce premier chapitre pose les fondations de notre projet en présentant son contexte général. Nous commençons par une présentation détaillée d'Amen Bank, l'organisme d'accueil qui constitue le cadre de ce travail. Nous exposons ensuite le cadrage précis du projet, les résultats de l'étude de l'existant, et nous justifions nos choix méthodologiques pour le développement de la solution.

1.2 Présentation d'Amen Bank

1.2.1 Historique et Position sur le Marché

Amen Bank est l'une des principales institutions bancaires tunisiennes, créée en 1967 sous la dénomination de *Banque de Développement Économique de Tunisie* (BDET). Après plusieurs évolutions et restructurations, elle a adopté sa dénomination actuelle en 2005.

Avec plus de 50 ans d'existence, Amen Bank s'est imposée comme un acteur majeur du secteur bancaire tunisien. La banque compte aujourd'hui :

— Plus de 180 agences réparties sur l'ensemble du territoire tunisien

— Un réseau de plus de 200 distributeurs automatiques de billets (DAB)

— Plus de 2000 collaborateurs

— Une base de clientèle diversifiée (particuliers, professionnels, entreprises)

1.2.2 Services Offerts

Amen Bank propose une gamme complète de produits et services bancaires :

— **Banque des particuliers** : Comptes courants, épargne, crédits à la consommation, prêts immobiliers

— **Banque des entreprises** : Crédits d'exploitation, financements d'investissements, commerce international

— **Services de trésorerie** : Gestion des mouvements de fonds, transferts inter-agences

— **Banque en ligne** : Services digitaux pour la consultation de comptes et les opérations à distance

1.2.3 Structure Organisationnelle

La structure organisationnelle d'Amen Bank suit un modèle hiérarchique classique adapté aux besoins du secteur bancaire. La figure 1.1 présente l'organigramme simplifié des entités concernées par notre projet.

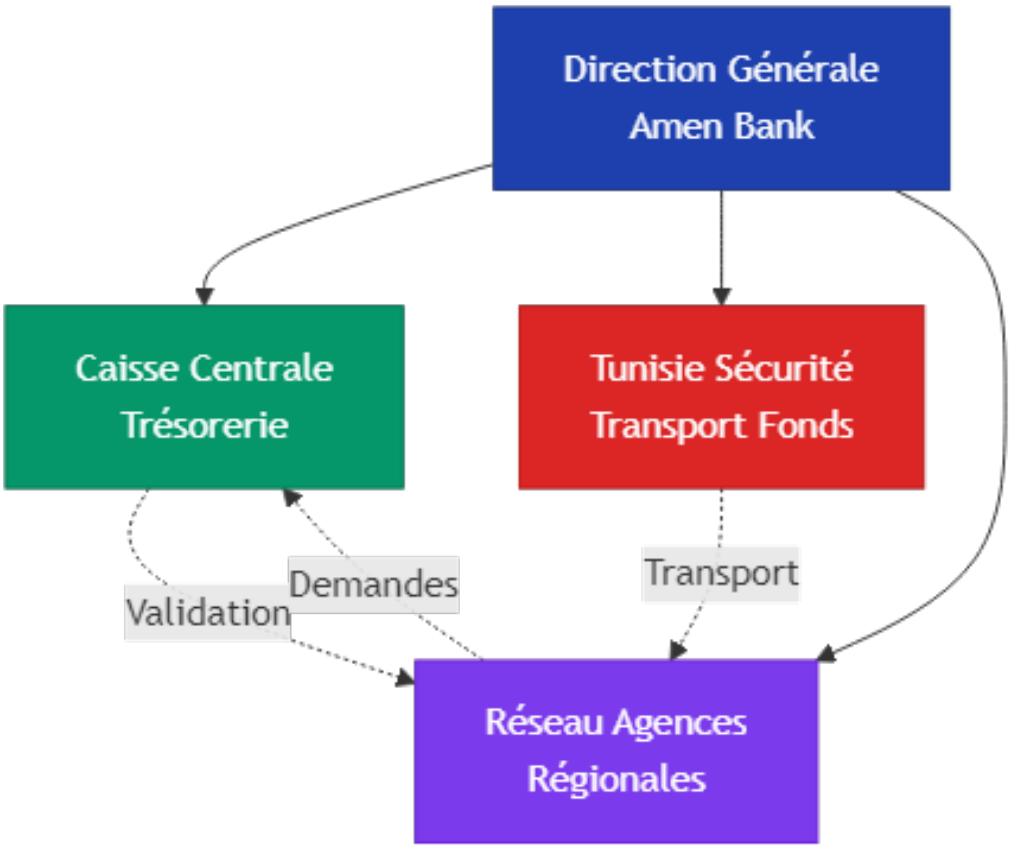

FIGURE 1.1 – Organigramme simplifié d'Amen Bank

Les principales entités impliquées dans le système de gestion des fonds sont :

Direction Générale : Définit la stratégie globale et supervise l'ensemble des opérations.

Caisse Centrale (Direction de la Trésorerie) : Gère les liquidités de la banque, valide les demandes de provisionnement et de versement, et coordonne les mouvements de fonds entre les agences.

Direction de la Sécurité (Tunisie Sécurité) : Assure le transport sécurisé des fonds, assigne les équipes (chauffeurs et gardes), et supervise les opérations de dispatch.

Réseau d'Agences : Points de contact avec la clientèle, les agences soumettent des demandes de fonds selon leurs besoins opérationnels et reçoivent les livraisons.

1.3 Cadrage du Projet

1.3.1 Problème Identifié

La gestion actuelle des mouvements de fonds chez Amen Bank repose sur un processus largement manuel qui présente plusieurs défaillances :

1. **Absence de système centralisé** : Les demandes sont gérées via des documents papier ou des emails non structurés, dispersés entre différents services.
2. **Circuit de validation inefficace** : Le processus d'approbation nécessite des signatures physiques multiples, entraînant des délais de plusieurs jours.
3. **Coordination difficile** : La communication entre les agences, la Caisse Centrale et Tunisie Sécurité manque de fluidité et génère des malentendus.
4. **Risques de sécurité** : Les informations sensibles (montants, coupures, équipes assignées) circulent sans protection adéquate.
5. **Traçabilité insuffisante** : Il est difficile de reconstituer l'historique complet d'une demande et d'identifier les responsabilités en cas de problème.
6. **Absence d'indicateurs** : La direction ne dispose pas de tableau de bord pour suivre les performances et identifier les goulots d'étranglement.

1.3.2 Solution Proposée

Pour résoudre ces problèmes, nous proposons le développement d'une **plateforme web centralisée** qui automatise l'intégralité du workflow de gestion des fonds. La solution comprend :

— Un système d'authentification sécurisé avec contrôle d'accès basé sur les rôles (RBAC)
— Des interfaces utilisateur spécialisées pour chaque acteur (Agence, Caisse Centrale, Tunisie Sécurité, Administrateur)
— Un workflow automatisé avec transitions d'états contrôlées
— Un système de traçabilité complet (audit logs) de toutes les actions
— Un tableau de bord analytique avec KPIs et visualisations
— Des notifications en temps réel pour les changements d'état

Les bénéfices attendus incluent :

— Réduction des délais de traitement de 70%
— Élimination des erreurs de saisie et de coordination
— Amélioration de la sécurité des informations
— Visibilité complète sur les opérations en cours
— Facilitation de l'audit et de la conformité réglementaire

1.4 Étude de l'Existant

1.4.1 Solutions Concurrentes sur le Marché

Plusieurs solutions commerciales existent pour la gestion bancaire, mais elles présentent des limitations pour notre cas d'usage spécifique :

SAP Banking Services

SAP propose un module de gestion de trésorerie intégré à sa suite ERP bancaire.

Avantages :

— Solution mature et éprouvée

— Intégration complète avec d'autres modules SAP

— Nombreuses fonctionnalités de reporting

Inconvénients :

— Coût de licence très élevé (plusieurs centaines de milliers d'euros)

— Complexité de déploiement et de paramétrage

— Nécessite des compétences spécialisées rares et coûteuses

— Workflows rigides difficiles à adapter aux processus spécifiques

Oracle FLEXCUBE

Oracle FLEXCUBE est une solution bancaire core complète incluant la gestion de trésorerie.

Avantages :

— Plateforme complète couvrant tous les besoins bancaires

— Scalabilité et haute disponibilité

— Support Oracle reconnu

Inconvénients :

— Infrastructure technique lourde et coûteuse

— Durée d'implémentation très longue (12-24 mois)

— Coûts de maintenance annuels élevés

— Sur-dimensionné pour notre besoin spécifique

Solutions Custom Développées en Interne

Certaines banques ont développé leurs propres solutions sur mesure.

Avantages :

— Adaptation parfaite aux processus internes

— Coûts de licence inexistants

— Maîtrise complète du code source

Inconvénients :

— Qualité variable selon les compétences de l'équipe

— Technologies souvent obsolètes (applications legacy)

— Maintenance coûteuse à long terme

— Documentation souvent insuffisante

1.4.2 Critique et Positionnement

Notre solution se positionne comme une alternative moderne et pragmatique :

— **Coût maîtrisé** : Développement sur mesure sans frais de licence
— **Technologies modernes** : Stack technique actuel (Next.js, PostgreSQL)
— **Adaptabilité** : Workflows configurables selon les besoins réels
— **Déploiement rapide** : Mise en production en 3-4 mois
— **Maintenabilité** : Code propre, documenté, avec tests automatisés
— **Évolutivité** : Architecture modulaire permettant des extensions futures

1.5 Méthodologie de Travail Adoptée

1.5.1 Comparaison des Méthodologies

Avant de choisir notre approche de développement, nous avons comparé les deux grandes familles de méthodologies.

Méthodologie en Cascade (Waterfall)

Le modèle en cascade est une approche séquentielle où chaque phase doit être complétée avant de passer à la suivante.

Phases : Analyse → Conception → Développement → Tests → Déploiement

Avantages :

— Structure claire et prévisible
— Documentation exhaustive
— Facile à gérer pour des projets à périmètre fixe

Inconvénients :

— Rigidité face aux changements de besoins
— Découverte tardive des problèmes
— Absence de livraisons intermédiaires
— Risque élevé si les spécifications initiales sont incorrectes

Méthodologie Agile

L'approche Agile favorise le développement itératif et incrémental avec des cycles courts et des validations fréquentes.

Principes clés :

— Individus et interactions plutôt que processus et outils
— Logiciel fonctionnel plutôt que documentation exhaustive
— Collaboration avec le client plutôt que négociation contractuelle
— Adaptation au changement plutôt que suivi d'un plan rigide

Avantages :

— Flexibilité et adaptation rapide aux changements

— Livraisons fréquentes de fonctionnalités utilisables
— Feedback continu et amélioration progressive
— Risques réduits grâce aux validations régulières

Inconvénients :

— Nécessite une implication forte du client
— Moins prévisible en termes de planning global
— Documentation parfois insuffisante

1.5.2 Choix Justifié : Scrum

Compte tenu des caractéristiques de notre projet, nous avons opté pour la méthodologie **Scrum**, un framework Agile structuré. Ce choix se justifie par :

— La complexité fonctionnelle du système nécessitant des validations fréquentes
— La disponibilité d'un Product Owner (représentant d'Amen Bank) impliqué
— La durée limitée du projet (3-4 mois) nécessitant une progression visible
— L'évolution possible des besoins au fur et à mesure de la compréhension du domaine

1.5.3 Cadre Scrum

Rôles

Product Owner : Représentant d'Amen Bank, définit les priorités et valide les fonctionnalités

Scrum Master : Facilite le processus Scrum, élimine les obstacles

Équipe de Développement : Développeur full-stack, responsable de la réalisation technique

Événements Scrum

Sprint Planning : Réunion de planification en début de sprint (2h) pour sélectionner les user stories et définir les tâches

Daily Standup : Point quotidien de 15 minutes pour synchroniser l'équipe et identifier les blocages

Sprint Review : Démonstration des fonctionnalités réalisées au Product Owner (1h)

Sprint Retrospective : Réunion d'amélioration continue identifiant ce qui fonctionne bien et ce qui peut être amélioré (1h)

Artefacts

Product Backlog : Liste ordonnée de toutes les fonctionnalités souhaitées, priorisées selon la méthode MoSCoW (Must have, Should have, Could have, Won't have)

Sprint Backlog : Ensemble des user stories et tâches sélectionnées pour le sprint en cours

Increment : Version fonctionnelle du produit à la fin de chaque sprint, potentiellement livrable

1.5.4 Planification Temporelle

Le projet a été planifié sur quatre sprints de deux à trois semaines chacun, pour une durée totale d'environ 10 semaines de développement.

Répartition des Sprints

Sprint 1 (Semaines 1-2) : Architecture et Authentification

- Configuration de l'environnement de développement
- Mise en place de l'architecture Next.js full-stack
- Système d'authentification et RBAC
- Gestion des utilisateurs et des rôles

Sprint 2 (Semaines 3-5) : Module Gestion des Demandes

- Création de demandes avec spécification des coupures
- Liste et filtrage des demandes
- Détails d'une demande
- Validation des montants

Sprint 3 (Semaines 6-8) : Module Validation et Assignment

- Validation/rejet des demandes par la Caisse Centrale
- Assignment des équipes de sécurité
- Système de logs et traçabilité
- Notifications de changement d'état

Sprint 4 (Semaines 9-10) : Module Dispatch et Analytics

- Confirmation de dispatch par Tunisie Sécurité
- Réception des fonds par les agences
- Gestion des non-conformités
- Tableau de bord analytics avec KPIs

Diagramme de Gantt

La figure 1.2 présente la planification temporelle du projet sous forme de diagramme de Gantt.

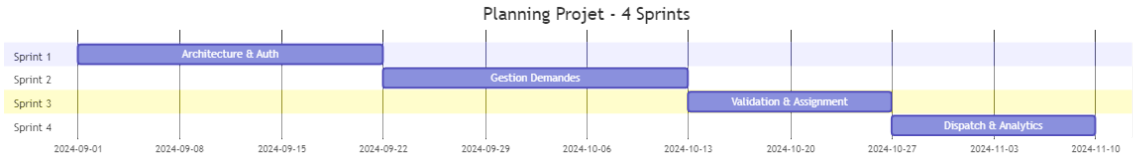

FIGURE 1.2 – Diagramme de Gantt du projet

Ce diagramme illustre la progression séquentielle des sprints avec leurs jalons respectifs. Chaque sprint se termine par une revue et une rétrospective avant d'entamer le suivant.

1.6 Conclusion

Dans ce chapitre, nous avons présenté le cadre général de notre projet. Nous avons d'abord exposé Amen Bank, son historique et sa structure organisationnelle, avant de définir précisément la problématique à résoudre et la solution que nous proposons. L'étude de l'existant nous a permis de positionner notre approche par rapport aux solutions commerciales disponibles. Enfin, nous avons justifié notre choix méthodologique en faveur de Scrum et présenté la planification détaillée du projet en quatre sprints.

Le chapitre suivant sera consacré à l'état de l'art, où nous approfondirons les concepts métiers bancaires et présenterons en détail la pile technologique retenue pour le développement de la solution.

Chapitre 2

État de l'Art et Technologies

2.1 Introduction

Ce chapitre présente le contexte académique et technologique de notre projet. Nous commençons par exposer les concepts métiers essentiels à la compréhension du domaine bancaire, notamment la gestion de trésorerie et les workflows d'approbation. Nous détaillons ensuite la pile technologique retenue, en justifiant chaque choix technique par rapport aux alternatives disponibles.

2.2 Concepts Métiers Bancaires

2.2.1 Gestion de Trésorerie

La gestion de trésorerie (ou Cash Management) est une fonction critique dans le secteur bancaire. Elle consiste à optimiser les flux de liquidités pour garantir que chaque point de vente dispose des fonds nécessaires à son activité quotidienne, tout en minimisant les coûts de détention de cash et les risques associés.

Enjeux de la Trésorerie Bancaire

Les principaux enjeux incluent :

— **Liquidité** : Assurer la disponibilité des fonds pour répondre aux demandes de retrait des clients

— **Sécurité** : Minimiser les risques de vol, perte ou erreur lors des transports de fonds

— **Optimisation** : Réduire les coûts liés au transport, à la détention et à la gestion du cash

— **Conformité** : Respecter les réglementations bancaires en matière de traçabilité et de reporting

Mouvements de Fonds Inter-Agences

Dans un réseau bancaire multi-agences, les mouvements de fonds entre entités sont quotidiens. Une agence peut avoir un excédent de liquidités qu'elle doit reverser à la Caisse Centrale, ou au contraire un besoin de provisionnement pour faire face aux demandes de ses clients.

2.2.2 Types de Mouvements

Notre système gère deux types principaux de mouvements de fonds :

Provisionnement (Remittance)

Le provisionnement correspond à l'envoi de fonds d'une agence vers la Caisse Centrale. Ce type de mouvement intervient lorsque :

— L'agence accumule un excédent de liquidités supérieur à son seuil de sécurité

— L'agence souhaite sécuriser les fonds collectés en fin de journée

— La Caisse Centrale demande une remontée de fonds pour rééquilibrer la trésorerie globale

Exemple : L'Agence Tunis Centre a collecté 500,000 DT suite à des dépôts clients importants. Elle crée une demande de provisionnement pour transférer ce montant à la Caisse Centrale.

Versement (Cash Supply)

Le versement correspond à l'envoi de fonds de la Caisse Centrale vers une agence. Ce type de mouvement intervient lorsque :

— L'agence anticipe des besoins importants (période de fêtes, salaires)

— Le niveau de liquidité de l'agence est tombé en dessous du seuil minimal

— L'agence a besoin de coupures spécifiques (billets de petite dénomination)

Exemple : L'Agence Sousse prévoit des retraits massifs en période d'Aïd. Elle demande un versement de 300,000 DT avec une répartition spécifique en billets de 20 DT et 50 DT.

2.2.3 Workflow d'Approbation Multiniveaux

Le traitement d'une demande de fonds suit un workflow complexe impliquant plusieurs acteurs et étapes de validation. Ce processus garantit la sécurité et la traçabilité des opérations.

Étapes du Workflow

Le cycle complet comprend cinq étapes principales :

1. **SUBMITTED** : L'agence soumet une demande avec les détails (montant, coupures)
2. **VALIDATED/REJECTED** : La Caisse Centrale examine et valide ou rejette la demande
3. **ASSIGNED** : Tunisie Sécurité assigne une équipe (chauffeur + garde)
4. **DISPATCHED** : Tunisie Sécurité confirme le départ de l'équipe avec les fonds
5. **RECEIVED/COMPLETED** : L'agence confirme la réception des fonds

Acteurs et Responsabilités

Agence : Initie la demande, spécifie les besoins, confirme la réception

Caisse Centrale : Valide la pertinence de la demande, vérifie la disponibilité des fonds

Tunisie Sécurité : Assigne les ressources logistiques, coordonne le transport

Système : Enregistre chaque action, notifie les parties prenantes, assure la traçabilité

Diagramme de Flux des États

La figure 2.1 illustre les transitions d'états possibles d'une demande dans le système.

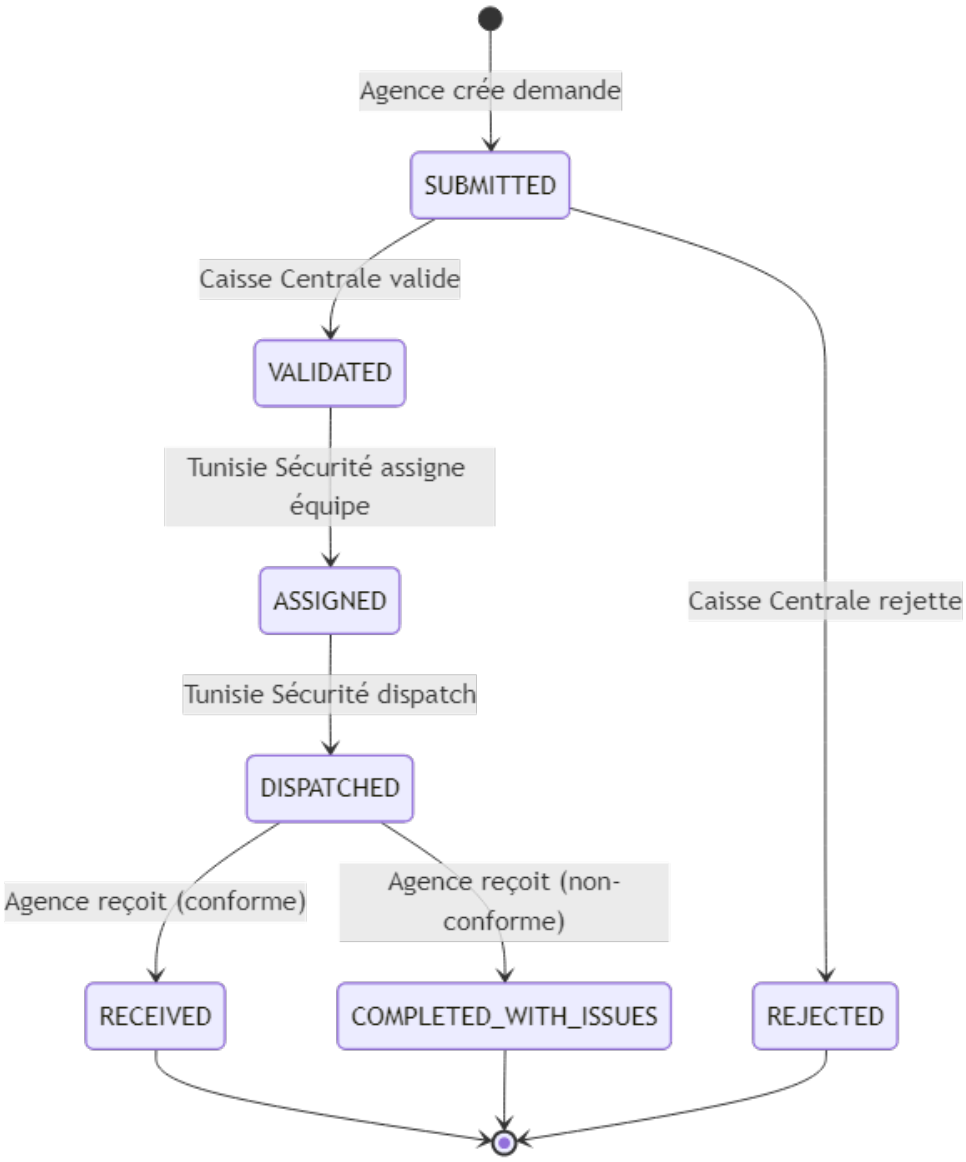

FIGURE 2.1 – Diagramme des transitions d'états d'une demande

Ce diagramme montre clairement les chemins possibles : une demande peut être rejetée à l'étape de validation, ou suivre le parcours complet jusqu'à la réception. Chaque transition est tracée avec identification de l'acteur et horodatage.

2.2.4 Traçabilité et Audit Trail

La traçabilité est un requirement critique pour les systèmes bancaires. Elle permet de :

— Reconstituer l'historique complet de chaque opération

— Identifier les responsabilités en cas d'incident

— Répondre aux exigences réglementaires d'audit

— Détecter les anomalies et les tentatives de fraude

Éléments Tracés

Pour chaque action effectuée sur une demande, le système enregistre :

— **Qui ?** Identification de l'utilisateur ayant effectué l'action
— **Quoi ?** Type d'action (validation, rejet, assignment, dispatch, réception)
— **Quand ?** Horodatage précis (date et heure)
— **Détails ?** Informations spécifiques (motif de rejet, équipe assignée, notes)

2.3 Pile Technologique

2.3.1 Architecture Générale

Pattern Architectural

Nous avons opté pour une architecture **full-stack JavaScript/TypeScript** basée sur Next.js, qui unifie le frontend et le backend dans un seul framework. Cette approche présente plusieurs avantages :

— Réduction de la complexité (un seul langage, une seule codebase)
— Partage de code entre client et serveur (types, validations, utilitaires)
— Performances optimisées grâce au Server-Side Rendering (SSR)
— Déploiement simplifié sur plateformes serverless

Architecture Trois Tiers

Malgré l'unification du frontend et backend, l'application respecte une séparation claire en trois couches :

Couche Présentation : Composants React (Server Components et Client Components), interfaces utilisateur, gestion de l'état client

Couche Logique Métier : API Routes Next.js, contrôleurs, services métier, validations, authentification/autorisation

Couche Données : PostgreSQL via Prisma ORM, gestion des transactions, migrations de schéma

Architecture Serverless

Le déploiement cible une architecture serverless sur Vercel pour le frontend et les API Routes, avec PostgreSQL hébergé sur Neon (cloud database). Les avantages incluent :

— Scalabilité automatique selon la charge
— Pas de gestion d'infrastructure
— Coûts réduits (facturation à l'usage)
— Haute disponibilité par défaut

2.3.2 Technologies Frontend

Next.js 15

Next.js est un framework React full-stack développé par Vercel. La version 15 apporte plusieurs innovations majeures :

App Router : Nouveau système de routing basé sur le système de fichiers avec support natif des layouts et des Server Components.

Server Components : Composants React exécutés côté serveur, réduisant la taille du bundle JavaScript envoyé au client et améliorant les performances.

API Routes intégrées : Possibilité de créer des endpoints API directement dans le projet Next.js, sans serveur backend séparé.

Optimisations automatiques : Minification, tree-shaking, code splitting, prefetching automatique.

TypeScript

TypeScript est un sur-ensemble typé de JavaScript qui apporte :

— **Type-safety** : Détection d'erreurs à la compilation plutôt qu'à l'exécution

— **IntelliSense** : Autocomplétion et documentation inline dans l'IDE

— **Refactoring sûr** : Les modifications sont propagées avec garantie de cohérence

— **Maintenabilité** : Code plus explicite et auto-documenté

Tailwind CSS et Radix UI

Tailwind CSS est un framework CSS utility-first qui permet de construire rapidement des interfaces modernes sans écrire de CSS personnalisé. Les classes utilitaires sont composables et offrent une cohérence de design.

Radix UI (via shadcn/ui) fournit des composants UI accessibles et non stylisés de base, que nous personnalisons avec Tailwind. Les composants incluent : Button, Dialog, Select, Table, Input, etc.

Zustand

Zustand est une bibliothèque de gestion d'état client minimaliste. Comparé à Redux :

— API plus simple (moins de boilerplate)

— Taille réduite (1KB vs 3KB pour Redux)

— Pas de Provider wrapper nécessaire

— Performance optimisée avec re-renders minimaux

Zod

Zod est une bibliothèque de validation de schémas TypeScript-first. Elle permet de :

— Définir des schémas de validation réutilisables

— Générer automatiquement des types TypeScript à partir des schémas

— Valider les données entrantes (formulaires, API requests)
— Produire des messages d'erreur clairs et personnalisables

2.3.3 Technologies Backend

Node.js 18+

Node.js est l'environnement d'exécution JavaScript côté serveur. La version 18 LTS apporte :

— Support natif de Fetch API
— Amélioration des performances V8
— Stabilité et support à long terme

Next.js API Routes

Au lieu d'utiliser Express.js ou un autre framework backend séparé, nous utilisons les API Routes de Next.js. Chaque fichier dans le dossier `app/api/` devient automatiquement un endpoint HTTP.

Exemple :

```
// app/api/requests/route.ts
export async function GET(request: Request) {
  const requests = await prisma.request.findMany();
  return Response.json(requests);
}
```

PostgreSQL

PostgreSQL est un SGBD relationnel open-source reconnu pour :

— **Robustesse** : Support complet des transactions ACID
— **Richesse fonctionnelle** : JSON, full-text search, contraintes complexes
— **Performance** : Optimisations avancées, indexation efficace
— **Extensibilité** : Types personnalisés, fonctions, triggers

Pour notre projet, PostgreSQL garantit l'intégrité des données critiques (montants, statuts, équipes) et supporte les transactions nécessaires pour les opérations composées.

Prisma ORM

Prisma est un ORM moderne qui révolutionne l'accès aux données en TypeScript. Ses caractéristiques clés :

Schema-first : Le schéma de base de données est défini dans un fichier `schema.prisma` déclaratif.

Génération de types : Prisma génère automatiquement des types TypeScript correspondant exactement au schéma.

Migrations : Système de migrations automatisé pour faire évoluer le schéma en production.

Prisma Studio : Interface graphique pour explorer et modifier les données en développement.

Query API : API fluide et type-safe pour construire des requêtes complexes.
Exemple :

```
const request = await prisma.request.create({
  data: {
    type: "PROVISIONING",
    totalAmount: 500000,
    agencyId: 1,
    denominations: {
      create: [
        { denomination: 100, quantity: 3000 },
        { denomination: 50, quantity: 4000 }
      ]
    }
  },
  include: { denominations: true }
});
```

NextAuth.js

NextAuth.js est la solution d'authentification standard pour Next.js. Elle fournit :

— Support de multiples providers (OAuth, credentials, email magic links)

— Gestion sécurisée des sessions (JWT ou database sessions)

— Callbacks personnalisables pour enrichir les tokens

— Protection CSRF intégrée

— Hooks React pour accéder à la session

Pour notre projet, nous utilisons le **Credentials Provider** avec authentification par email/mot de passe. Les sessions sont gérées via JWT stockés dans des cookies HTTP-only sécurisés.

2.3.4 Sécurité

Hashing des Mots de Passe : bcrypt

Les mots de passe ne sont jamais stockés en clair. Nous utilisons **bcrypt** pour le hashing, avec un salt factor de 12 rounds. Bcrypt est résistant aux attaques par force brute grâce à sa lenteur intentionnelle.

```
import bcrypt from 'bcryptjs';

// Lors de la creation d'utilisateur
const hashedPassword = await bcrypt.hash(password, 12);

// Lors de la connexion
const isValid = await bcrypt.compare(password, hashedPassword);
```

JSON Web Tokens (JWT)

Les sessions utilisateur sont matérialisées par des JWT signés cryptographiquement. Le token contient :

— ID utilisateur

— Rôle utilisateur
— Agence (si applicable)
— Date d'expiration

Les tokens ont une durée de vie limitée (24h) et sont automatiquement rafraîchis lors de l'activité utilisateur.

Middleware de Protection

Un middleware Next.js vérifie l'authentification et les autorisations sur chaque requête vers une route protégée :

```
export function middleware(request: NextRequest) {
  const token = request.cookies.get('session-token');

  if (!token) {
    return NextResponse.redirect('/login');
  }

  // Verification du token et des permissions
  // ...
}
```

RBAC : Role-Based Access Control

Le système implémente un contrôle d'accès fin basé sur quatre rôles :

— **ADMIN** : Accès complet, gestion des utilisateurs, analytics
— **AGENCY** : Création et consultation de demandes, confirmation réception
— **CENTRAL_CASH** : Validation/rejet des demandes, vue globale
— **SECURITY** : Assignment équipes, confirmation dispatch

Chaque endpoint API vérifie que l'utilisateur authentifié possède le rôle requis.

2.3.5 Outils de Développement

Version Control : Git + GitHub

Git est utilisé pour le versionnement du code avec une stratégie de branching :

— `main` : Branche de production, toujours stable
— `develop` : Branche d'intégration des fonctionnalités
— `feature/*` : Branches de développement pour chaque user story

GitHub héberge le repository et facilite la revue de code et la documentation.

IDE : Visual Studio Code

VS Code est l'éditeur principal avec extensions essentielles :

— Prisma (syntax highlighting pour schema.prisma)
— ESLint (linting JavaScript/TypeScript)
— Prettier (formatage automatique du code)
— Tailwind CSS IntelliSense (autocomplétion classes CSS)

Gestion BD : Prisma Studio

Prisma Studio est une interface web pour visualiser et manipuler les données pendant le développement. Lancé via `npx prisma studio`, il offre une vue tabulaire de toutes les tables avec possibilité de CRUD manuel.

API Testing : Postman

Postman est utilisé pour tester les endpoints API indépendamment du frontend. Nous maintenons une collection Postman documentée avec :

— Exemples de requêtes pour chaque endpoint

— Variables d'environnement (dev, staging, prod)

— Tests automatisés vérifiant les codes de statut et la structure des réponses

Formatage et Linting

Prettier assure un formatage cohérent du code (indentation, guillemets, points-virgules).

ESLint détecte les erreurs potentielles et enforce les bonnes pratiques :

— Variables non utilisées

— Dépendances manquantes dans useEffect

— Import de hooks React incorrects

Package Manager

Nous utilisons **npm** (Node Package Manager) pour gérer les dépendances du projet. Le fichier `package.json` définit :

— Les dépendances de production (Next.js, Prisma, NextAuth, etc.)

— Les dépendances de développement (TypeScript, ESLint, types)

— Les scripts de build, développement et déploiement

2.3.6 Tableau Comparatif des Technologies

Le tableau 2.1 compare nos choix technologiques avec des alternatives courantes.

TABLE 2.1 – Comparaison des choix technologiques

Aspect	**Notre Choix**	**Alternative**	**Justification**
Framework	Next.js 15	Express + React	SSR natif, API Routes intégrées Réduction complexité architecture
Base de données	PostgreSQL	MongoDB	Transactionnalité ACID requise Relations complexes entre entités
ORM	Prisma	TypeORM Sequelize	Type-safety excellente Migrations plus simples Prisma Studio très pratique
Authentification	NextAuth.js	Passport.js JWT manual	Intégration parfaite Next.js Session management inclus
État Client	Zustand	Redux Context API	Légèreté (1KB vs 3KB) API plus simple, moins verbose
Styling	Tailwind CSS	Material-UI Styled Comp.	Flexibilité design Performance (pas de runtime)
Validation	Zod	Yup Joi	Intégration TypeScript native Inférence de types automatique

2.4 Conclusion

Dans ce chapitre, nous avons présenté les fondements théoriques et techniques du projet. L'exploration des concepts métiers bancaires (gestion de trésorerie, workflows d'approbation, traçabilité) nous a permis de comprendre les exigences fonctionnelles du système. La présentation détaillée de la pile technologique a justifié nos choix architecturaux et techniques, en soulignant les avantages de chaque technologie retenue.

Le prochain chapitre entame la phase de réalisation en décrivant le Sprint 1, consacré à la mise en place de l'architecture du système et du module d'authentification.

Chapitre 3

Sprint 1 - Architecture et Authentification

3.1 Introduction

Le premier sprint constitue la fondation technique du projet. L'objectif principal est de mettre en place l'infrastructure de base : l'architecture Next.js full-stack, le modèle de données initial, et surtout le système d'authentification sécurisé avec contrôle d'accès basé sur les rôles (RBAC). Ce sprint établit les standards de sécurité et de qualité qui seront maintenus tout au long du développement.

3.2 Objectifs du Sprint

Les objectifs spécifiques du Sprint 1 sont :

1. Configurer l'environnement de développement complet (Next.js, PostgreSQL, Prisma)
2. Définir et implémenter le schéma de base de données initial
3. Développer le système d'authentification avec NextAuth.js
4. Implémenter le contrôle d'accès basé sur les rôles (RBAC)
5. Créer les interfaces de gestion des utilisateurs (CRUD)
6. Développer les dashboards de base pour chaque rôle
7. Mettre en place les composants UI réutilisables

3.3 Spécification des Besoins

3.3.1 Besoins Fonctionnels

Authentification

— **F1.1** : Un utilisateur doit pouvoir se connecter avec email et mot de passe

— **F1.2** : Le système doit créer une session sécurisée après connexion réussie

— **F1.3** : Un utilisateur authentifié doit pouvoir se déconnecter

— **F1.4** : Les utilisateurs non authentifiés doivent être automatiquement redirigés vers la page de login

— **F1.5** : Le système doit afficher des messages d'erreur clairs en cas d'échec d'authentification

Gestion des Utilisateurs

— **F1.6** : Un administrateur doit pouvoir créer un nouvel utilisateur
— **F1.7** : Un administrateur doit pouvoir consulter la liste de tous les utilisateurs
— **F1.8** : Un administrateur doit pouvoir modifier les informations d'un utilisateur
— **F1.9** : Un administrateur doit pouvoir supprimer un utilisateur
— **F1.10** : Chaque utilisateur doit être associé à un rôle unique
— **F1.11** : Les utilisateurs de type Agence doivent être rattachés à une agence spécifique

Gestion des Rôles et Agences

— **F1.12** : Le système doit supporter quatre rôles prédéfinis : Admin, Agence, CaisseCentrale, TunisieSécurité
— **F1.13** : Un administrateur doit pouvoir consulter la liste des agences
— **F1.14** : Un administrateur doit pouvoir créer de nouvelles agences

Tableau de Bord

— **F1.15** : Chaque utilisateur doit accéder à un dashboard personnalisé selon son rôle
— **F1.16** : Le dashboard doit afficher les informations pertinentes (nom, rôle, agence)
— **F1.17** : La navigation doit être adaptée aux permissions de l'utilisateur

3.3.2 Besoins Non-Fonctionnels

Sécurité

— **NF1.1** : Les mots de passe doivent être hashés avec bcrypt (12 rounds)
— **NF1.2** : Les sessions doivent utiliser des JWT avec expiration (24h)
— **NF1.3** : Les cookies de session doivent être HTTP-only et Secure
— **NF1.4** : Chaque route sensible doit être protégée par un middleware d'authentification
— **NF1.5** : La validation des données doit être effectuée côté serveur avec Zod

Performance

— **NF1.6** : Le temps de chargement initial de la page de login ne doit pas excéder 1 seconde
— **NF1.7** : L'authentification doit être réalisée en moins de 500ms
— **NF1.8** : Les pages doivent utiliser le SSR de Next.js pour un chargement rapide

Ergonomie

— **NF1.9** : L'interface doit être responsive (mobile, tablette, desktop)
— **NF1.10** : Les actions utilisateur doivent avoir un feedback visuel immédiat (toasts, loaders)
— **NF1.11** : Les messages d'erreur doivent être explicites et orientés solution

Maintenabilité

— **NF1.12** : Le code doit être écrit en TypeScript avec types stricts

— **NF1.13** : L'architecture doit suivre le principe de séparation des responsabilités

— **NF1.14** : Les composants UI doivent être réutilisables et bien documentés

3.3.3 Acteurs Identifiés

Le système distingue quatre types d'acteurs principaux :

Administrateur : Responsable de la gestion globale du système. Crée et gère les utilisateurs, les rôles et les agences. Accède aux statistiques et analytics.

Agence : Utilisateur rattaché à une agence bancaire spécifique. Peut créer des demandes de fonds et consulter l'historique de son agence.

Caisse Centrale : Responsable de la validation des demandes. Examine les demandes soumises et décide de leur approbation ou rejet.

Tunisie Sécurité : Responsable de la logistique. Assigne les équipes de transport et confirme les dispatches.

3.3.4 Diagramme de Cas d'Utilisation - Sprint 1

La figure 3.1 présente les cas d'utilisation du premier sprint.

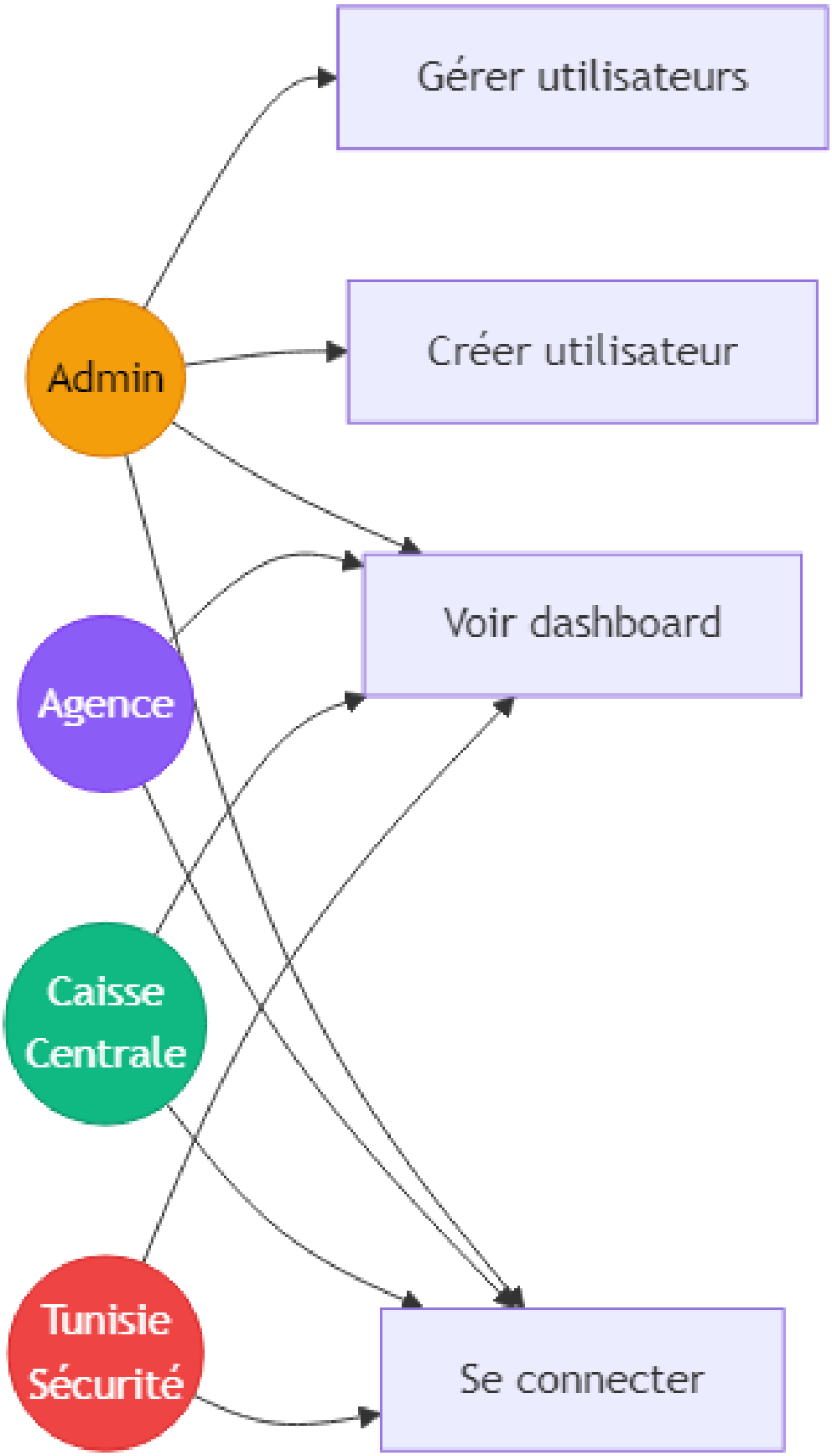

FIGURE 3.1 – Diagramme de cas d'utilisation - Sprint 1

Ce diagramme montre les interactions principales entre les acteurs et le système pendant cette première phase. L'administrateur peut gérer les utilisateurs, tandis que tous les acteurs peuvent s'authentifier et accéder à leur dashboard respectif.

3.4 Conception du Sprint

3.4.1 Diagramme de Classes - Entités Principales

La figure 3.2 présente le modèle de classes des entités principales du Sprint 1.

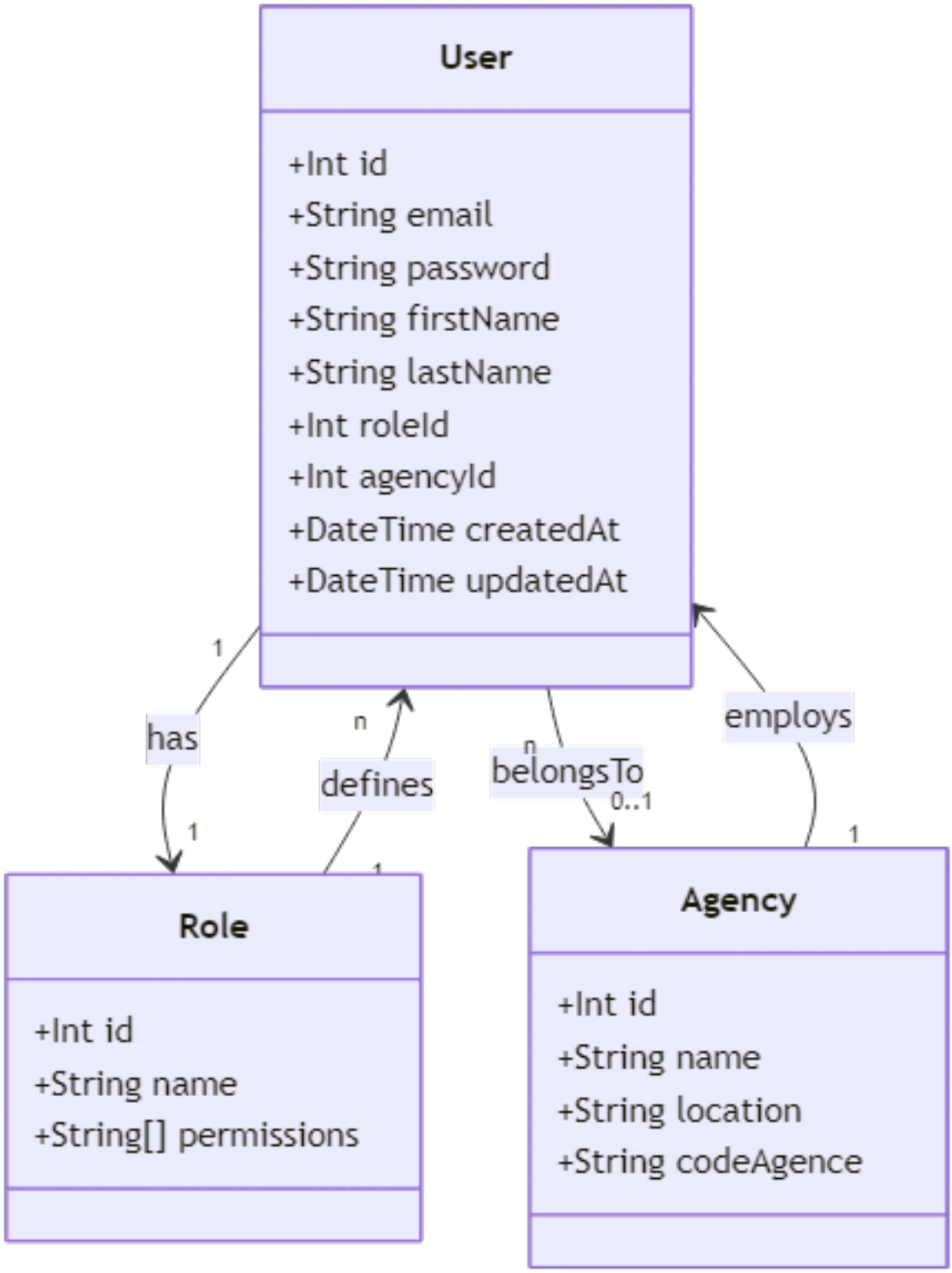

FIGURE 3.2 – Diagramme de classes - Sprint 1

Les entités principales sont :

User : Représente un utilisateur du système avec ses identifiants, son rôle et son agence éventuelle.

Role : Définit un rôle avec un nom et des permissions associées.

Agency : Représente une agence bancaire avec son nom, sa localisation et son code unique.

Les relations sont :

— Un User appartient à un Role (many-to-one)

— Un User peut appartenir à une Agency (many-to-one, optionnel)

— Une Agency peut avoir plusieurs Users (one-to-many)

3.4.2 Diagramme de Séquence - Processus de Connexion

La figure 3.3 illustre le processus d'authentification d'un utilisateur.

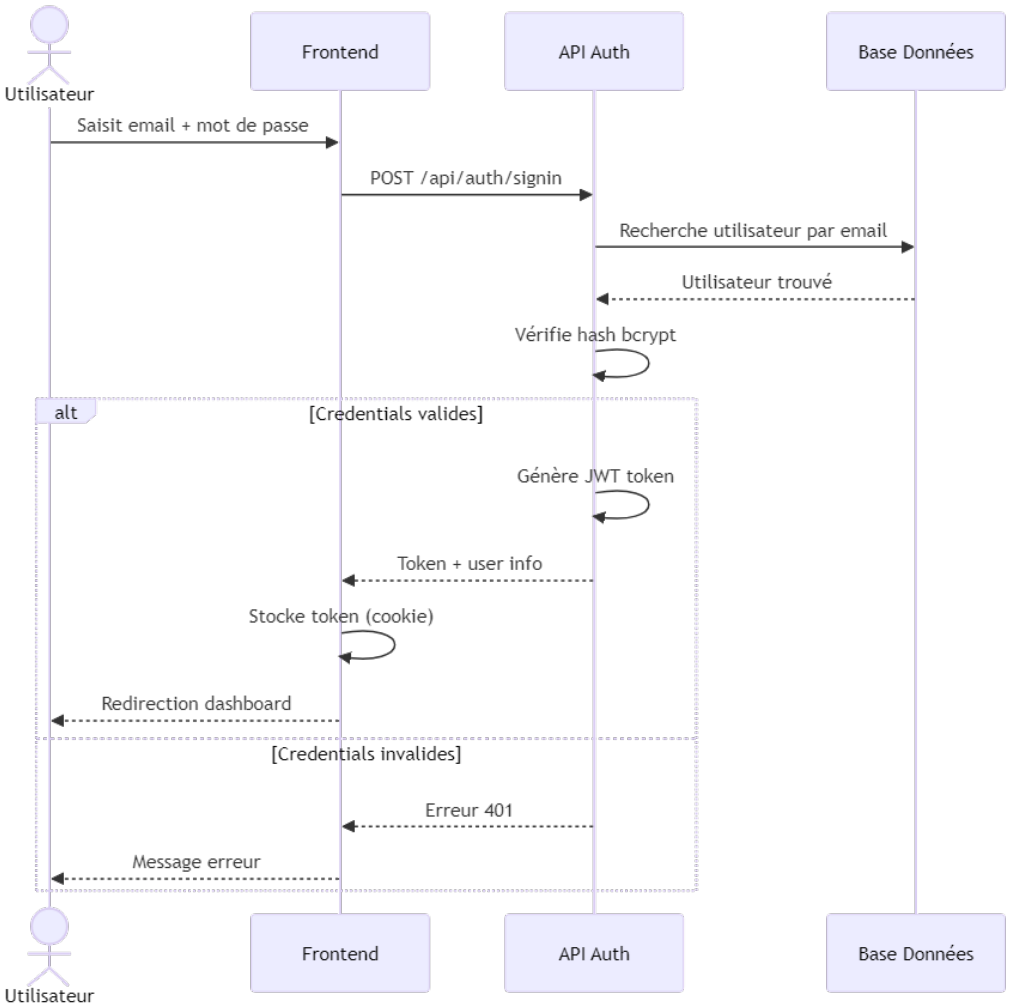

FIGURE 3.3 – Diagramme de séquence - Processus de connexion

Le processus suit les étapes suivantes :

1. L'utilisateur saisit son email et mot de passe dans le formulaire
2. Le frontend envoie une requête POST à l'API d'authentification
3. L'API recherche l'utilisateur dans la base de données par email
4. Si l'utilisateur existe, l'API compare le hash du mot de passe avec bcrypt
5. Si la validation réussit, l'API génère un JWT contenant l'ID, le rôle et l'agence

6. Le JWT est stocké dans un cookie HTTP-only sécurisé
7. L'API retourne les informations utilisateur au frontend
8. Le frontend redirige l'utilisateur vers son dashboard approprié

3.4.3 Diagramme de Séquence - Création d'Utilisateur

La figure 3.4 montre le processus de création d'un utilisateur par l'administrateur.

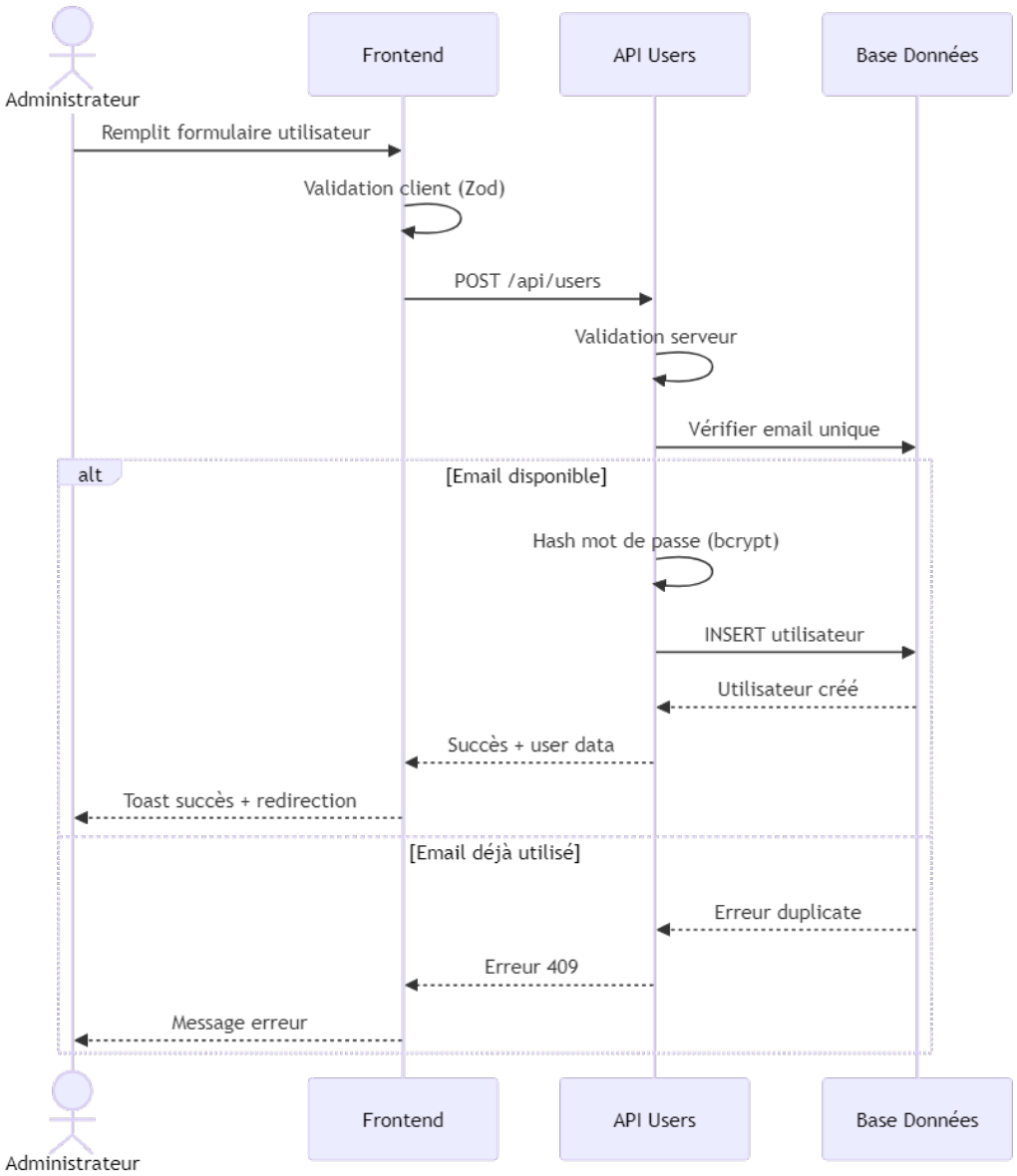

FIGURE 3.4 – Diagramme de séquence - Création d'utilisateur

Les étapes détaillées sont :

1. L'administrateur accède au formulaire de création d'utilisateur
2. Il saisit les informations requises (nom, prénom, email, mot de passe, rôle, agence)
3. Le frontend valide les données côté client avec Zod
4. Une requête POST est envoyée à `/api/users` avec les données

5. L'API vérifie que l'utilisateur courant est bien administrateur
6. L'API valide à nouveau les données côté serveur (Zod schema)
7. L'API vérifie l'unicité de l'email dans la base de données
8. Le mot de passe est hashé avec bcrypt
9. L'utilisateur est créé dans la base via Prisma
10. L'API retourne l'utilisateur créé (sans le mot de passe)
11. Le frontend affiche un message de succès et actualise la liste

3.4.4 Diagramme d'Activités - Flux d'Authentification

La figure 3.5 présente le flux décisionnel du processus d'authentification.

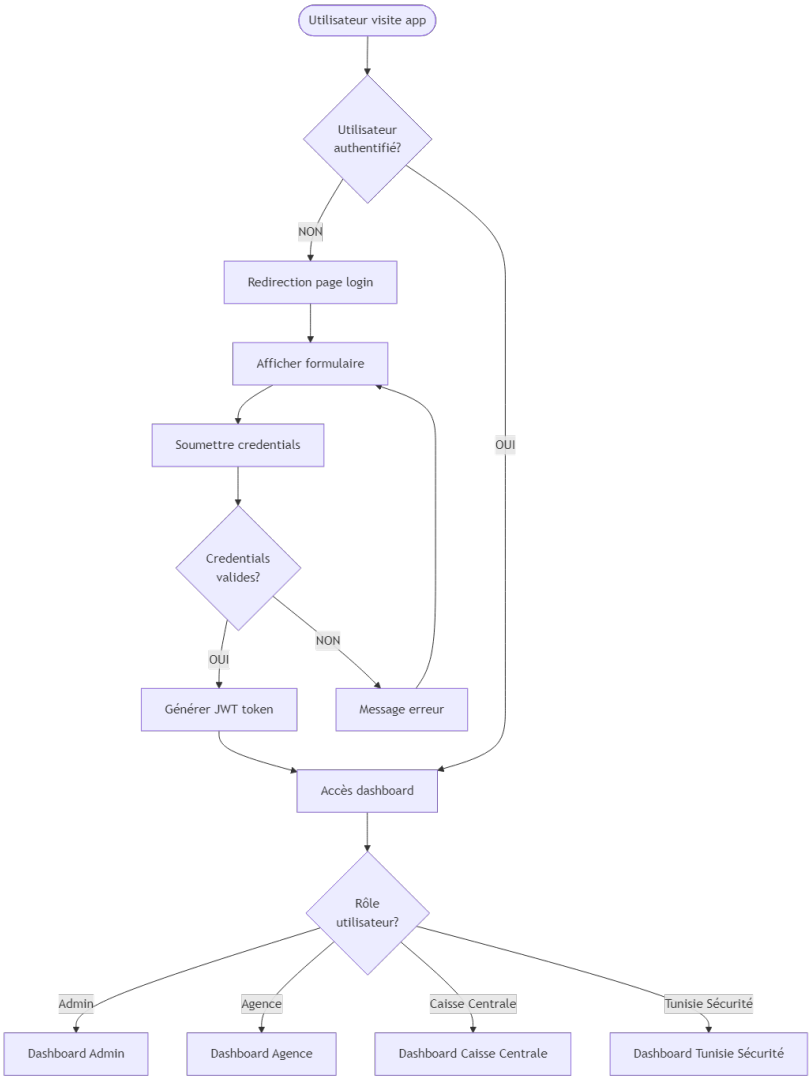

FIGURE 3.5 – Diagramme d'activités - Flux d'authentification

Ce diagramme illustre les différents chemins possibles :

— Si l'utilisateur est déjà authentifié, il accède directement à son dashboard

— Sinon, il est redirigé vers la page de login

— Après soumission du formulaire, les credentials sont vérifiés

— En cas de succès, un JWT est généré et l'utilisateur accède à l'application

— En cas d'échec, un message d'erreur est affiché et l'utilisateur reste sur la page de login

3.4.5 Modèle de Données - Schema Prisma

Le schéma Prisma définit la structure de la base de données pour le Sprint 1 :

```
model User {
  id         Int      @id @default(autoincrement())
  email      String   @unique
  password   String
  firstName  String
  lastName   String
  role       Role     @relation(fields: [roleId], references: [id])
  roleId     Int
  agency     Agency?  @relation(fields: [agencyId], references: [id])
  agencyId   Int?
  createdAt  DateTime @default(now())
  updatedAt  DateTime @updatedAt

  @@index([email])
  @@index([roleId])
}

model Role {
  id    Int    @id @default(autoincrement())
  name  String @unique
  users User[]
}

model Agency {
  id          Int      @id @default(autoincrement())
  name        String
  location    String
  codeAgence  String   @unique
  users       User[]
  createdAt   DateTime @default(now())

  @@index([codeAgence])
}
```

Les contraintes importantes :

— Email unique pour chaque utilisateur

— Code agence unique pour chaque agence

— Index sur les colonnes fréquemment interrogées (email, roleId, codeAgence)

— Timestamps automatiques pour audit (createdAt, updatedAt)

3.5 Réalisation du Sprint

3.5.1 Composants Frontend Développés

Page de Login (`app/login/page.tsx`)

Interface d'authentification avec :

— Formulaire email/password avec validation en temps réel
— Affichage des erreurs de validation
— Loader pendant la soumission
— Redirection automatique après connexion réussie

Dashboard Layout (`components/layout/dashboard-layout.tsx`)

Layout principal de l'application incluant :

— Sidebar avec navigation adaptée au rôle
— Header avec informations utilisateur et bouton de déconnexion
— Zone de contenu principale
— Breadcrumb pour la navigation contextuelle

Protected Route (`components/auth/protected-route.tsx`)

Composant wrapper qui :

— Vérifie l'authentification de l'utilisateur
— Redirige vers login si non authentifié
— Affiche un loader pendant la vérification
— Vérifie optionnellement les permissions par rôle

User Management (Admin)

Interface complète de gestion des utilisateurs avec :

— Tableau listant tous les utilisateurs avec pagination
— Filtres par rôle et agence
— Bouton d'ajout ouvrant un dialog modal
— Actions d'édition et suppression sur chaque ligne
— Confirmation avant suppression

3.5.2 Routes API Implémentées

Authentification

`POST /api/auth/signin` : Authentification avec email/password, retourne JWT

`POST /api/auth/signout` : Déconnexion, suppression du cookie de session

`GET /api/auth/session` : Récupération des informations de session courante

Gestion des Utilisateurs

`POST /api/users` : Création d'un nouvel utilisateur (admin only)

`GET /api/users` : Liste de tous les utilisateurs avec pagination (admin only)

`GET /api/users/[id]` : Détails d'un utilisateur spécifique

`PUT /api/users/[id]` : Modification d'un utilisateur (admin only)

`DELETE /api/users/[id]` : Suppression d'un utilisateur (admin only)

Rôles et Agences

`GET /api/roles` : Liste des rôles disponibles

`GET /api/agencies` : Liste de toutes les agences

`POST /api/agencies` : Création d'une nouvelle agence (admin only)

3.5.3 Composants Logique (Services)

Configuration NextAuth (`lib/auth.ts`)

Configuration principale de NextAuth.js incluant :

— Credentials Provider pour authentification email/password

— Callbacks pour enrichir le JWT avec rôle et agence

— Pages personnalisées (login, error)

— Options de session (stratégie JWT, durée de vie)

Utilitaires Auth (`lib/auth-utils.ts`)

Fonctions réutilisables :

— `getServerSession()` : Récupération de la session côté serveur

— `requireAuth()` : Middleware pour routes protégées

— `requireRole(role)` : Vérification de rôle spécifique

— `hashPassword(password)` : Hashing bcrypt

— `verifyPassword(password, hash)` : Vérification bcrypt

Database Client (`lib/db.ts`)

Initialisation et export du client Prisma :

```
import { PrismaClient } from '@prisma/client';

const globalForPrisma = global as unknown as {
  prisma: PrismaClient | undefined;
};

export const prisma =
  globalForPrisma.prisma ??
  new PrismaClient({
    log: ['query', 'error', 'warn'],
  });
```

```
if (process.env.NODE_ENV !== 'production')
  globalForPrisma.prisma = prisma;
```

Cette approche évite la multiplication des connexions en développement avec hot-reload.

3.5.4 Composants UI Réutilisables

Nous avons créé une bibliothèque de composants UI de base avec shadcn/ui et Tailwind :

— **Button** : Bouton avec variantes (primary, secondary, danger, ghost)

— **Input** : Champ de saisie avec label et message d'erreur

— **Label** : Label de formulaire

— **Card** : Conteneur de contenu avec header/body/footer

— **Dialog** : Modal dialog pour formulaires et confirmations

— **Table** : Tableau avec tri, pagination et sélection

— **Select** : Liste déroulante avec recherche

— **Skeleton** : Placeholder de chargement

Ces composants sont stylisés avec Tailwind CSS et respectent les principes d'accessibilité (ARIA labels, navigation au clavier).

3.6 Tests et Validation

3.6.1 Tests Fonctionnels

Nous avons effectué les tests suivants :

Authentification

✓ Login avec credentials valides réussit

✓ Login avec email invalide échoue avec message approprié

✓ Login avec mot de passe incorrect échoue

✓ Session créée après login réussi

✓ Logout supprime la session correctement

✓ Redirection automatique vers login si non authentifié

Gestion des Utilisateurs

✓ Création d'utilisateur avec données valides réussit

✓ Email dupliqué provoque une erreur

✓ Mot de passe trop court refusé par validation

✓ Liste des utilisateurs s'affiche correctement

✓ Modification d'utilisateur sauvegarde les changements

✓ Suppression d'utilisateur retire de la base

Permissions par Rôle

✓ Admin peut accéder à la gestion des utilisateurs
✓ Agence ne peut pas accéder à la gestion des utilisateurs
✓ Chaque rôle voit son dashboard approprié
✓ Navigation adaptée selon le rôle utilisateur

3.6.2 Tests de Sécurité

✓ Mots de passe hashés dans la base (bcrypt)
✓ JWT signé correctement et non falsifiable
✓ Cookies de session HTTP-only et Secure
✓ Routes API protégées refusent accès non authentifié
✓ Validation serveur empêche injection SQL

3.7 Revue de Sprint

3.7.1 Démonstration

À la fin du Sprint 1, nous avons présenté au Product Owner les fonctionnalités suivantes :

1. Processus de connexion complet avec validation
2. Dashboard administrateur avec navigation
3. Interface de gestion des utilisateurs (CRUD complet)
4. Différents dashboards par rôle (Admin, Agence, Caisse Centrale, Tunisie Sécurité)
5. Protection des routes et contrôle d'accès par rôle

3.7.2 Validation Product Owner

Le Product Owner a validé :

— L'approche architecture (Next.js full-stack)
— La sécurité du système d'authentification
— L'ergonomie des interfaces (design moderne, responsive)
— La couverture des besoins fonctionnels du Sprint 1

3.7.3 Feedback et Ajustements

Quelques suggestions ont été formulées pour les sprints suivants :

— Ajouter un système de recherche dans la liste des utilisateurs
— Améliorer les messages de feedback utilisateur (toasts plus visibles)
— Prévoir l'export de la liste des utilisateurs en Excel

3.7.4 Métriques du Sprint

— **Durée** : 2 semaines
— **User Stories complétées** : 8/8 (100%)
— **Points de complexité** : 21/21
— **Bugs identifiés** : 3 (tous résolus)
— **Couverture de tests** : Tests manuels complets

3.8 Conclusion du Chapitre

Le Sprint 1 a permis de poser des fondations solides pour le projet. Le système d'authentification sécurisé avec RBAC garantit que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités sensibles. Le modèle de données initial est extensible et prêt à accueillir les entités plus complexes des sprints suivants (demandes, logs, équipes).

L'architecture Next.js full-stack s'est révélée pertinente, offrant un développement fluide avec partage de code entre frontend et backend. Les composants UI réutilisables créés dans ce sprint accéléreront le développement des interfaces des sprints suivants.

Le chapitre suivant décrit le Sprint 2, consacré au développement du module core du système : la gestion des demandes de fonds avec spécification des coupures et consultation de l'historique.

Chapitre 4

Sprint 2 - Module Gestion des Demandes

4.1 Introduction

Le Sprint 2 se concentre sur le développement du module central du système : la gestion des demandes de fonds. Ce module permet aux agences de créer des demandes détaillées spécifiant les coupures nécessaires, et offre aux différents acteurs (Agence, Caisse Centrale) une vue complète sur l'état des demandes. Ce sprint constitue le cœur fonctionnel de l'application.

4.2 Vue d'Ensemble

Le module de gestion des demandes couvre l'intégralité du cycle de vie d'une demande, de sa création à sa consultation. Les fonctionnalités principales incluent :

— Création de demandes avec spécification détaillée des coupures et pièces

— Validation automatique des montants (correspondance total vs détails)

— Liste des demandes avec filtrage avancé (statut, date, agence, montant)

— Consultation des détails complets d'une demande

— Historique des actions effectuées sur chaque demande

4.3 Objectif Principal

L'objectif principal du Sprint 2 est de permettre aux agences de soumettre des demandes de fonds précises et détaillées, tout en offrant à la Caisse Centrale une visibilité complète sur toutes les demandes en attente de traitement. Le système doit garantir la cohérence des données et faciliter la prise de décision.

4.4 User Stories Détaillées

Le tableau 4.1 présente les user stories du Sprint 2 avec leurs critères d'acceptation et priorités.

TABLE 4.1 – User Stories - Sprint 2

ID	En tant que	Je veux	Critères d'acceptation	Priorité
US-01	Agence	Créer une demande de provisionnement	Formulaire complet, validation montants, confirmation	Must
US-02	Agence	Spécifier les coupures détaillées	Ajouter plusieurs dénominations, calcul automatique	Must
US-03	Agence	Consulter mes demandes avec filtres	Liste paginée, filtres statut/date	Must
US-04	Caisse Centrale	Voir toutes les demandes	Vue globale, tri agence/date/statut	Must
US-05	Agence	Télécharger un PDF de ma demande	Export PDF formaté avec logo	Should

4.5 Analyse et Spécifications

4.5.1 Diagramme de Cas d'Utilisation - Sprint 2

La figure 4.1 présente les cas d'utilisation du Sprint 2.

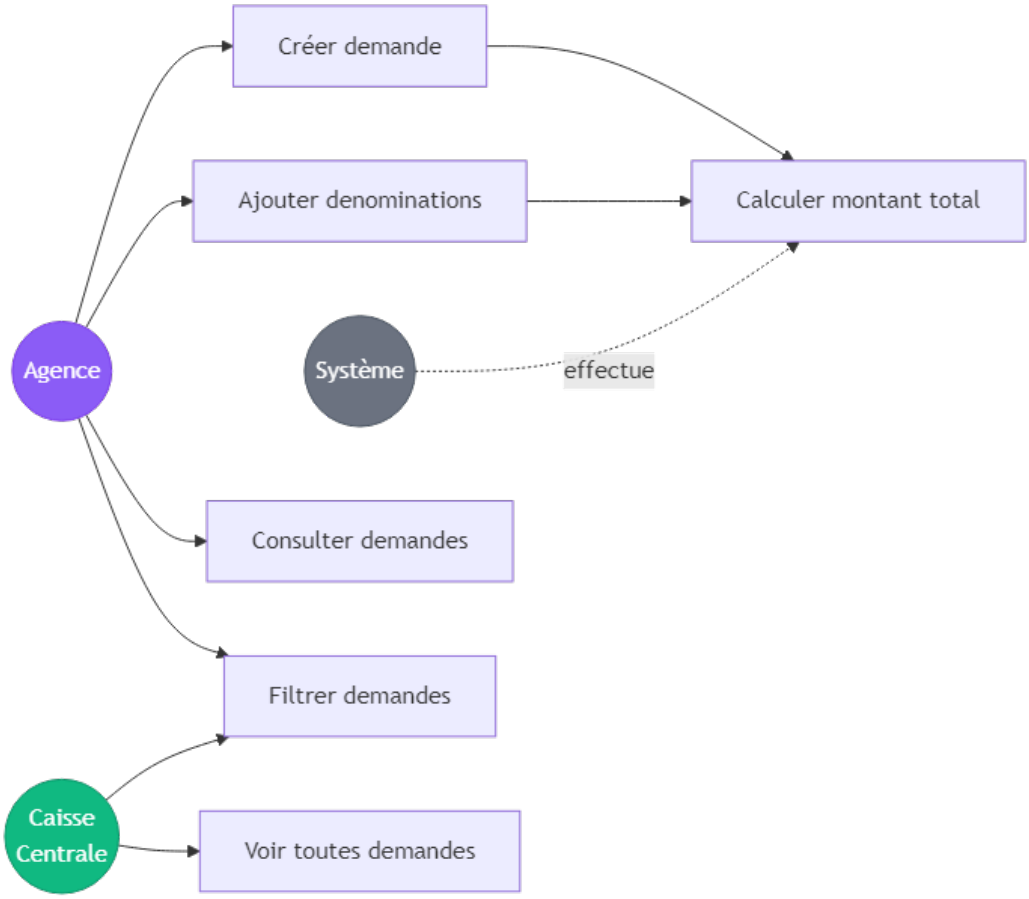

FIGURE 4.1 – Diagramme de cas d'utilisation - Sprint 2

Les cas d'utilisation principaux sont :

— **Créer demande** : L'agence soumet une nouvelle demande avec détails
— **Ajouter denominations** : Spécification des coupures et quantités
— **Consulter mes demandes** : L'agence visualise ses demandes avec filtres
— **Voir toutes demandes** : La Caisse Centrale a une vue globale
— **Calculer montant total** : Le système valide la cohérence

4.5.2 Raffinements Textuels - Scénario Nominal

Cas d'utilisation : Créer une Demande

Acteur Principal : Utilisateur Agence
Préconditions :

— L'utilisateur est authentifié avec le rôle Agence
— L'utilisateur est rattaché à une agence spécifique

Scénario nominal :

1. L'utilisateur accède à la page "Créer une demande"
2. Le système affiche un formulaire vide
3. L'utilisateur sélectionne le type de mouvement (Provisionnement ou Versement)

4. L'utilisateur saisit le montant total souhaité (ex : 500,000 DT)
5. L'utilisateur saisit une description optionnelle
6. L'utilisateur ajoute les détails des coupures :
   - Sélectionne une coupure (100 DT)
   - Saisit la quantité (3000 billets)
   - Le système calcule le subtotal (300,000 DT)
   - Répète pour d'autres coupures (50 DT × 4000 = 200,000 DT)
7. Le système calcule le montant total des denominations (500,000 DT)
8. Le système vérifie que le total correspond au montant saisi
9. L'utilisateur clique sur "Soumettre la demande"
10. Le système valide les données (schéma Zod)
11. Le système génère un numéro unique de demande (ex : REQ-2024-001)
12. Le système sauvegarde la demande avec statut "SUBMITTED"
13. Le système affiche une confirmation avec le numéro de demande
14. Le système envoie une notification à la Caisse Centrale

Postconditions :

- Une nouvelle demande est créée dans la base de données
- La demande est visible dans la liste des demandes de l'agence
- La Caisse Centrale peut consulter cette demande

Scénario Alternatif : Montants Incohérents

Point de divergence : Étape 8 du scénario nominal

1. L'utilisateur saisit un montant total de 1,000,000 DT
2. L'utilisateur ajoute des denominations pour un total de 950,000 DT
3. Le système détecte une différence de 50,000 DT
4. L'utilisateur tente de soumettre le formulaire
5. Le système affiche un message d'erreur : "Le montant total ne correspond pas à la somme des denominations (différence : 50,000 DT)"
6. L'utilisateur corrige soit le montant total, soit les denominations
7. Retour à l'étape 9 du scénario nominal

4.5.3 Diagramme de Classes - Entités Demandes

La figure 4.2 présente le modèle de classes étendu incluant les entités de demandes.

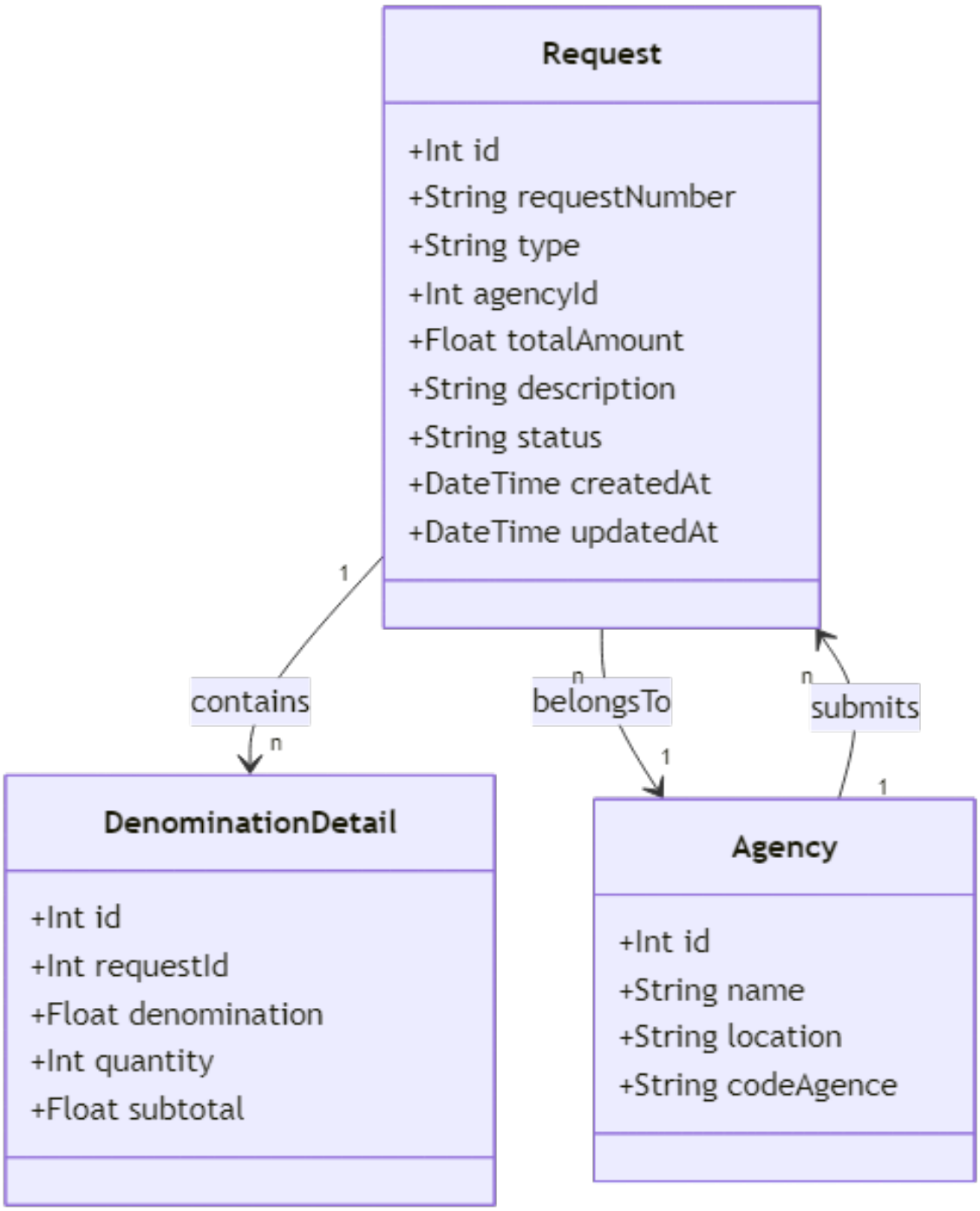

FIGURE 4.2 – Diagramme de classes - Sprint 2

Les nouvelles entités sont :

Request : Représente une demande de fonds avec son type, montant, statut et agence émettrice

DenominationDetail : Détaille une ligne de coupure (dénomination, quantité, subtotal)

Le modèle Prisma correspondant :

```
model Request {
```

```
  id               Int      @id @default(autoincrement())
  requestNumber    String   @unique
  type             String   // "PROVISIONING" | "REMITTANCE"
  agency           Agency   @relation(fields: [agencyId], references: [
     id])
  agencyId         Int
  totalAmount      Float
  description      String?
  status           String   @default("SUBMITTED")
  denominations    DenominationDetail[]
  createdBy        User     @relation(fields: [createdById], references
     : [id])
  createdById      Int
  createdAt        DateTime @default(now())
  updatedAt        DateTime @updatedAt

  @@index([requestNumber])
  @@index([status])
  @@index([agencyId])
}

model DenominationDetail {
  id           Int     @id @default(autoincrement())
  request      Request @relation(fields: [requestId], references: [id
     ], onDelete: Cascade)
  requestId    Int
  denomination Float   // 100, 50, 20, 10, 5, 2, 1
  quantity     Int
  subtotal     Float   // denomination * quantity

  @@index([requestId])
}
```

4.5.4 Diagramme de Séquence - Création Demande

La figure 4.3 illustre le processus complet de création d'une demande.

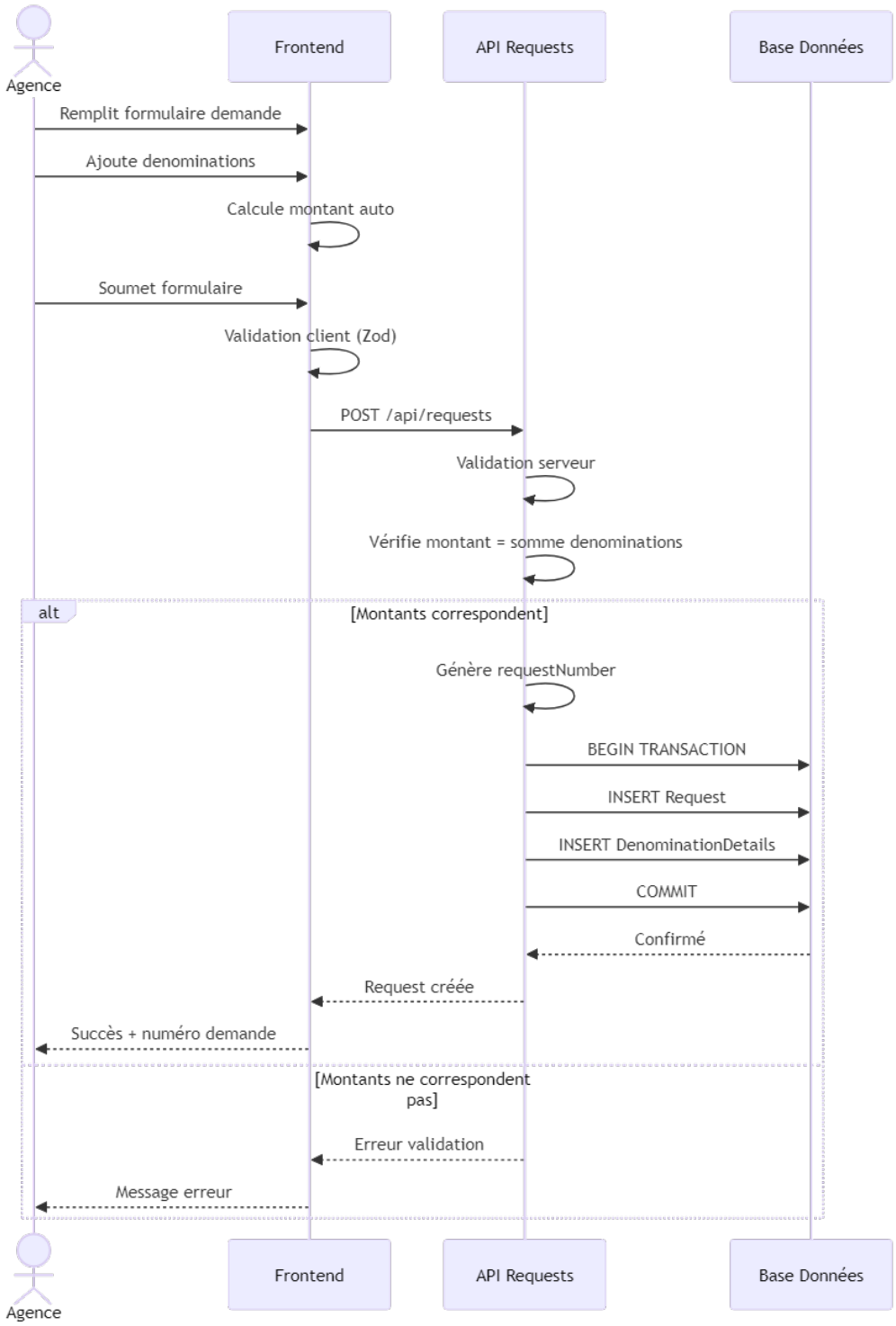

FIGURE 4.3 – Diagramme de séquence - Création de demande

Le processus détaillé :

1. L'agence remplit le formulaire avec type, montant et denominations
2. Le frontend valide les données côté client (Zod schema)
3. Une requête POST est envoyée à `/api/requests`
4. L'API vérifie l'authentification et le rôle (doit être Agence)
5. L'API valide à nouveau les données côté serveur

6. L'API vérifie que le montant total = somme des subtotals
7. L'API génère un requestNumber unique (format : REQ-YYYY-XXX)
8. L'API démarre une transaction Prisma
9. L'API crée l'enregistrement Request
10. L'API crée les enregistrements DenominationDetail associés
11. L'API commit la transaction
12. L'API retourne la demande créée avec son numéro
13. Le frontend affiche un message de succès
14. Le frontend redirige vers la page de détails de la demande

4.5.5 Diagramme d'Activités - Consultation Demandes

La figure 4.4 présente le flux de consultation des demandes avec filtres.

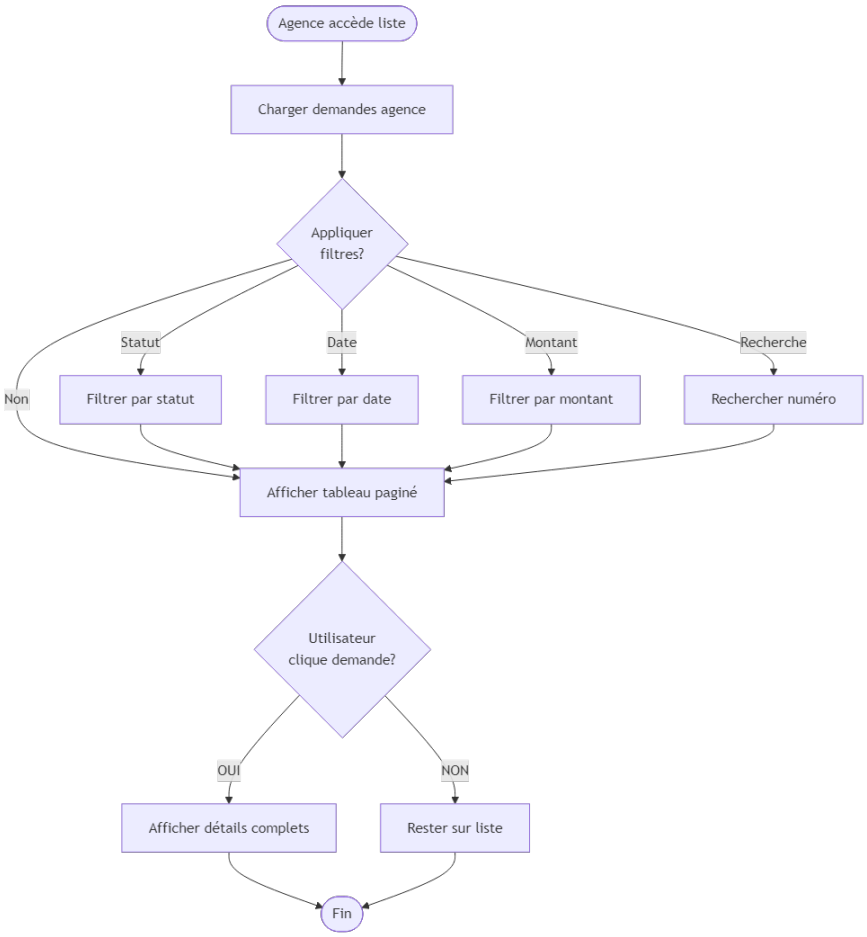

FIGURE 4.4 – Diagramme d'activités - Consultation des demandes

Le flux décisionnel inclut :

— Chargement initial des demandes selon le rôle (Agence = ses demandes, Caisse Centrale = toutes)

— Application optionnelle de filtres (statut, date, montant)

— Recherche par numéro de demande
— Tri des résultats
— Pagination (10, 25 ou 50 par page)
— Navigation vers détails si l'utilisateur clique sur une demande

4.6 Conception Détaillée

4.6.1 Architecture Composants Frontend

La structure des composants pour le module demandes :

```
app/requests/
 page.tsx                    # Liste des demandes
 create/
    page.tsx                # Création de demande
 [id]/
     page.tsx                # Détails d'une demande

components/requests/
 request-filters.tsx         # Filtres et recherche
 denomination-input.tsx      # Input coupures dynamique
 request-table.tsx           # Tableau des demandes
 request-details.tsx         # Vue détails complète
```

4.6.2 Routes API

`GET /api/requests` : Liste des demandes avec filtres
  — Query params : agencyId, status, startDate, endDate, page, limit
  — Retourne : `{ requests: Request[], total: number, page: number }`

`POST /api/requests` : Création d'une demande
  — Body : `{ type, agencyId, totalAmount, description?, denominations[] }`
  — Validation : Schema Zod
  — Retourne : Request créée avec requestNumber

`GET /api/requests/[id]` : Détails d'une demande
  — Include : denominations, agency, createdBy
  — Retourne : Request complète

`PUT /api/requests/[id]` : Modification (description uniquement si statut = SUBMITTED)

`GET /api/requests/[id]/pdf` : Export PDF de la demande

4.6.3 Schémas de Validation Zod

```
import { z } from 'zod';

const DenominationSchema = z.object({
  denomination: z.enum([100, 50, 20, 10, 5, 2, 1]),
  quantity: z.number().int().min(1).max(100000),
});

const CreateRequestSchema = z.object({
  type: z.enum(['PROVISIONING', 'REMITTANCE']),
  totalAmount: z.number().min(100).max(10000000),
  description: z.string().max(500).optional(),
  agencyId: z.number().int().positive(),
  denominations: z.array(DenominationSchema).min(1),
}).refine((data) => {
  const calculatedTotal = data.denominations.reduce(
    (sum, d) => sum + (d.denomination * d.quantity),
    0
  );
  return calculatedTotal === data.totalAmount;
}, {
  message: "Le montant total ne correspond pas aux denominations",
});
```

4.7 Réalisation - Implémentation

4.7.1 Composant Création Demande

Le composant de création de demande (`app/requests/create/page.tsx`) implémente :

Formulaire Principal

- **Section Informations Générales**
  - Radio buttons pour type (Provisionnement / Versement)
  - Input numérique pour montant total
  - Textarea pour description (optionnel)
- **Section Détails Denominations**
  - Liste dynamique de lignes denomination
  - Chaque ligne : Select (coupure) + Input (quantité) + Display (subtotal)
  - Bouton "Ajouter une ligne" pour nouvelles coupures
  - Bouton "Supprimer" sur chaque ligne
  - Calcul automatique du total en temps réel
- **Section Validation**
  - Affichage du montant calculé vs montant saisi
  - Indicateur visuel (vert si match, rouge si différence)
  - Message d'erreur si incohérence
- **Actions**
  - Bouton "Soumettre" (disabled si validation échoue)

— Bouton "Annuler" (retour à la liste)
— Loader pendant la soumission

Gestion de l'État Local

Le composant utilise React hooks pour gérer l'état :

```
const [type, setType] = useState('PROVISIONING');
const [totalAmount, setTotalAmount] = useState(0);
const [description, setDescription] = useState('');
const [denominations, setDenominations] = useState([
  { denomination: 100, quantity: 0 }
]);

const calculatedTotal = denominations.reduce(
  (sum, d) => sum + (d.denomination * d.quantity),
  0
);

const isValid = calculatedTotal === totalAmount && totalAmount > 0;
```

4.7.2 Composant Liste Demandes

Le composant de liste (`app/requests/page.tsx`) offre :

Tableau des Demandes

Colonnes affichées :
— Numéro de demande (cliquable, lien vers détails)
— Type (badge "Provisionnement" ou "Versement")
— Agence (nom complet)
— Montant (formaté avec séparateurs de milliers)
— Statut (badge coloré selon l'état)
— Date de création (format JJ/MM/AAAA HH :mm)
— Actions (icône "Voir détails")

Système de Filtrage

— **Filtre par statut** : Dropdown multi-select (SUBMITTED, VALIDATED, REJECTED, etc.)
— **Filtre par date** : Date picker range (de - à)
— **Filtre par montant** : Input range (min - max)
— **Recherche** : Input text recherchant dans requestNumber
— **Bouton "Réinitialiser"** : Efface tous les filtres

Pagination

— Sélecteur de nombre par page (10, 25, 50)
— Boutons Précédent/Suivant
— Affichage "Résultats 1-10 sur 45"
— Navigation directe vers une page spécifique

4.7.3 Composant Détails Demande

Le composant de détails (`app/requests/[id]/page.tsx`) structure l'information en sections :

Header

— Numéro de demande (titre principal)
— Badge de statut (couleur selon l'état)
— Informations agence (nom, code, localisation)

Section Informations Générales

Tableau récapitulatif :
— Type de mouvement
— Montant total
— Date de création
— Créé par (nom utilisateur)
— Description (si présente)

Section Détails Denominations

Tableau des coupures :
— Colonne "Dénomination" (100 DT, 50 DT, etc.)
— Colonne "Quantité" (nombre de billets/pièces)
— Colonne "Subtotal" (dénomination × quantité)
— Ligne de total (somme des subtotals)

Section Timeline / Historique

Affichage chronologique des événements :
— Icône selon le type d'action
— Date et heure
— Type d'action (Création, Validation, Rejet, etc.)
— Acteur (utilisateur ayant effectué l'action)
— Détails supplémentaires (motif de rejet, équipe assignée)

Actions Disponibles

Selon le statut et le rôle :

— Bouton "Télécharger PDF" (toujours visible)

— Bouton "Modifier" (si statut = SUBMITTED et rôle = Agence)

— Actions de validation (Sprint 3)

4.7.4 État Zustand - Request Store

Le store Zustand centralise l'état des demandes :

```
import create from 'zustand';

interface RequestState {
  requests: Request[];
  filters: {
    status: string[];
    dateFrom: Date | null;
    dateTo: Date | null;
    minAmount: number | null;
    maxAmount: number | null;
    search: string;
  };
  pagination: {
    page: number;
    limit: number;
    total: number;
  };
  setRequests: (requests: Request[]) => void;
  setFilters: (filters: Partial<FilterState>) => void;
  setPagination: (pagination: Partial<PaginationState>) => void;
  addRequest: (request: Request) => void;
  updateRequest: (id: number, updates: Partial<Request>) => void;
}

export const useRequestStore = create<RequestState>((set) => ({
  requests: [],
  filters: {
    status: [],
    dateFrom: null,
    dateTo: null,
    minAmount: null,
    maxAmount: null,
    search: '',
  },
  pagination: { page: 1, limit: 25, total: 0 },
  setRequests: (requests) => set({ requests }),
  setFilters: (newFilters) =>
    set((state) => ({
      filters: { ...state.filters, ...newFilters },
      pagination: { ...state.pagination, page: 1 } // Reset page
    })),
  setPagination: (newPagination) =>
    set((state) => ({
      pagination: { ...state.pagination, ...newPagination }
    })),
  addRequest: (request) =>
```

```
    set((state) => ({
      requests: [request, ...state.requests]
    })),
  updateRequest: (id, updates) =>
    set((state) => ({
      requests: state.requests.map(r =>
        r.id === id ? { ...r, ...updates } : r
      )
    })),
}));
```

4.8 Tests et Validation

4.8.1 Tests Fonctionnels

Création de Demandes

✓ Création avec montant et denominations valides réussit
✓ Numéro de demande unique généré automatiquement
✓ Montants incohérents provoquent erreur de validation
✓ Denomination dupliquée refusée
✓ Montant négatif ou zéro refusé
✓ Message de succès affiché après création

Consultation

✓ Agence voit uniquement ses demandes
✓ Caisse Centrale voit toutes les demandes
✓ Filtres par statut fonctionnent correctement
✓ Filtres par date limitent résultats
✓ Recherche par requestNumber trouve la demande
✓ Pagination affiche le bon nombre de résultats

Détails

✓ Toutes les informations affichées correctement
✓ Denominations listées avec calculs exacts
✓ Timeline vide pour demande nouvellement créée

4.8.2 Tests de Performance

✓ Liste de 1000 demandes charge en $<$ 2 secondes
✓ Filtrage réactif ($<$ 300ms)
✓ Création de demande avec 7 denominations $<$ 1 seconde

4.8.3 Tests d'Intégration

✓ Transaction Prisma garantit atomicité (Request + DenominationDetails)
✓ Rollback en cas d'erreur ne laisse pas d'orphelins
✓ Contraintes d'intégrité référentielle respectées

4.9 Revue de Sprint

4.9.1 Démonstration

Démonstration au Product Owner :

1. Création d'une demande de provisionnement (500,000 DT)
2. Ajout de denominations multiples avec calcul automatique
3. Tentative de soumission avec montants incohérents (erreur affichée)
4. Correction et soumission réussie
5. Consultation de la liste avec différents filtres
6. Navigation vers détails d'une demande
7. Export PDF (aperçu)

4.9.2 Validation Product Owner

Points validés :

— Ergonomie de création de demande (intuitive)
— Système de validation des montants (fiable)
— Filtrage et recherche (complets et performants)
— Présentation des détails (claire et structurée)

4.9.3 Feedback

Suggestions retenues :

— Ajouter indicateur visuel du statut dans la liste (badges colorés) [✓ Implémenté]
— Permettre tri des colonnes du tableau [*Ajouté au backlog*]
— Ajouter statistiques en haut de liste (nombre par statut) [*Sprint 4*]

4.9.4 Métriques du Sprint

— **Durée** : 3 semaines
— **User Stories complétées** : 5/5 (100%)
— **Points de complexité** : 34/34
— **Bugs identifiés** : 5 (4 résolus, 1 mineur reporté)
— **Lignes de code** : 2,500 (frontend + backend)

4.10 Conclusion du Chapitre

Le Sprint 2 a permis de développer le module central du système : la gestion des demandes de fonds. Les agences peuvent désormais soumettre des demandes détaillées avec spécification précise des coupures nécessaires. Le système de validation garantit la cohérence des données, évitant les erreurs de saisie.

La Caisse Centrale dispose d'une vue complète sur toutes les demandes soumises, avec des outils de filtrage et de recherche performants facilitant la priorisation et le traitement. L'interface intuitive et responsive permet une adoption rapide par les utilisateurs.

Le chapitre suivant décrit le Sprint 3, qui implémente le workflow de validation par la Caisse Centrale et d'assignment des équipes de sécurité par Tunisie Sécurité, complétant ainsi la phase de préparation avant le dispatch.

Chapitre 5

Sprint 3 - Module Validation et Assignment

5.1 Introduction

Le Sprint 3 se concentre sur l'implémentation du workflow multiniveaux qui transforme une demande soumise en demande prête à être dispatchée. Ce sprint est crucial car il met en place les mécanismes d'approbation et d'allocation des ressources logistiques, deux étapes essentielles dans le processus de gestion des fonds.

5.2 Vue d'Ensemble

Le module de validation et d'assignment couvre deux processus distincts mais séquentiels :

1. **Validation par la Caisse Centrale** : Examen des demandes soumises et décision d'approbation ou de rejet
2. **Assignment par Tunisie Sécurité** : Attribution d'une équipe de transport (chauffeur + garde) aux demandes validées

Ces processus garantissent que seules les demandes légitimes sont traitées et que les ressources humaines et logistiques sont correctement allouées.

5.3 Objectif Principal

L'objectif principal du Sprint 3 est d'automatiser le processus d'approbation et d'attribution des ressources logistiques, tout en assurant une traçabilité complète de chaque décision et action. Le système doit permettre aux différents acteurs de prendre des décisions éclairées rapidement.

5.4 User Stories Détaillées

Le tableau 5.1 présente les user stories du Sprint 3.

TABLE 5.1 – User Stories - Sprint 3

ID	En tant que	Je veux	Critères d'acceptation	Priorité
US-06	Caisse Centrale	Valider une demande soumise	Transition SUBMITTED → VALIDATED	Must
US-07	Caisse Centrale	Rejeter une demande avec motif	Ajouter justification obligatoire	Must
US-08	Tunisie Sécurité	Assigner une équipe à une demande	Sélectionner chauffeur + garde	Must
US-09	Tunisie Sécurité	Voir liste équipes disponibles	CIN + noms + disponibilité	Must
US-10	Système	Tracer chaque action	Logs utilisateur + timestamp	Should
US-11	Système	Envoyer notifications	Notifier changements statut	Could

5.5 Analyse et Spécifications

5.5.1 Diagramme de Cas d'Utilisation - Sprint 3

La figure 5.1 présente les cas d'utilisation du Sprint 3.

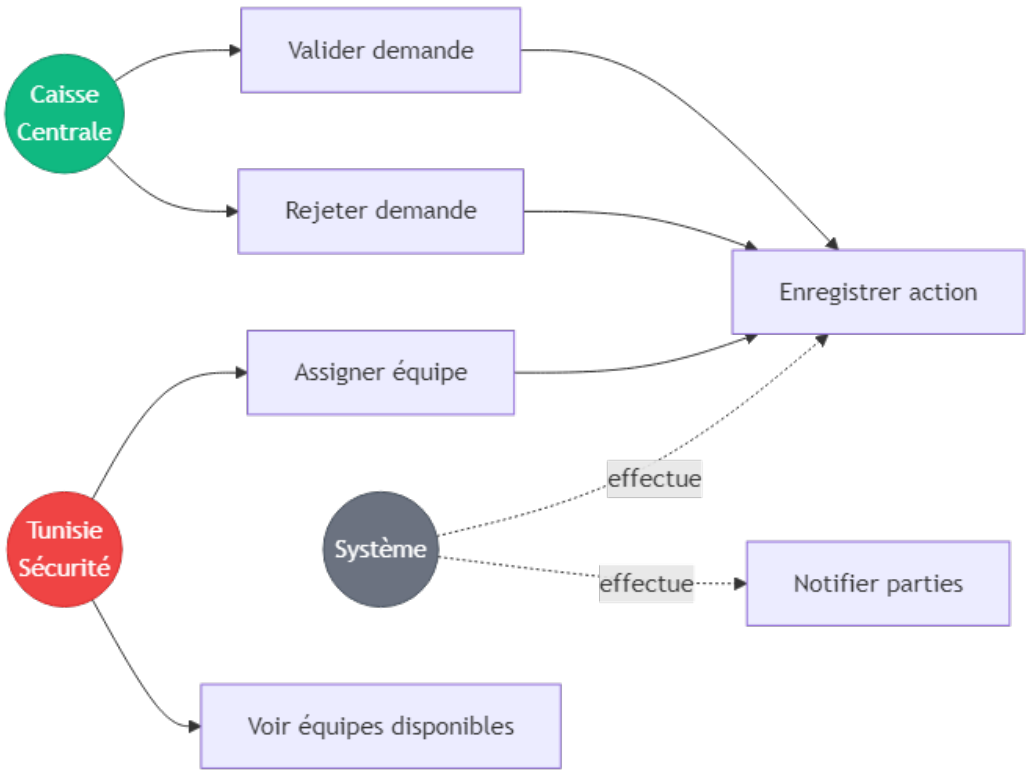

FIGURE 5.1 – Diagramme de cas d'utilisation - Sprint 3

Les cas d'utilisation principaux sont :

— **Valider demande** : Approbation par la Caisse Centrale
— **Rejeter demande** : Refus avec justification
— **Assigner équipe** : Allocation des ressources logistiques
— **Consulter équipes disponibles** : Vue sur les ressources
— **Enregistrer action** : Traçabilité automatique

5.5.2 Raffinements Textuels - Scénario Validation

Cas d'utilisation : Valider une Demande

Acteur Principal : Utilisateur Caisse Centrale
Préconditions :

— L'utilisateur est authentifié avec le rôle Caisse Centrale
— Une demande existe avec statut SUBMITTED
— Les fonds nécessaires sont disponibles

Scénario nominal :

1. L'utilisateur accède à la liste des demandes en attente (statut = SUBMITTED)
2. L'utilisateur clique sur une demande pour voir les détails
3. Le système affiche toutes les informations (montant, agence, denominations)
4. L'utilisateur examine la demande (vérification de la pertinence)
5. L'utilisateur clique sur le bouton "Valider"
6. Le système affiche un dialog de confirmation
7. L'utilisateur confirme la validation
8. Le système met à jour le statut de la demande → VALIDATED
9. Le système enregistre la date et l'utilisateur ayant validé
10. Le système crée un log d'action (action : VALIDATED, actor : userId)
11. Le système envoie une notification à Tunisie Sécurité
12. Le système affiche un message de succès
13. La vue est rafraîchie, la demande n'apparaît plus dans la liste SUBMITTED

Postconditions :

— La demande passe au statut VALIDATED
— Tunisie Sécurité peut maintenant assigner une équipe
— Un log d'action est créé pour audit

Scénario Alternatif : Rejet avec Motif

Point de divergence : Étape 5 du scénario nominal

1. L'utilisateur identifie un problème (montant excessif, demande non justifiée)
2. L'utilisateur clique sur le bouton "Rejeter"
3. Le système affiche un dialog avec un champ texte "Motif du rejet"

4. L'utilisateur saisit une justification détaillée (minimum 10 caractères)
5. L'utilisateur confirme le rejet
6. Le système valide la présence du motif
7. Le système met à jour le statut → REJECTED
8. Le système enregistre le motif, la date et l'utilisateur
9. Le système crée un log d'action (action : REJECTED, details : motif)
10. Le système envoie une notification à l'agence émettrice
11. Le système affiche un message de succès
12. La vue est rafraîchie

5.5.3 Raffinements Textuels - Scénario Assignment

Cas d'utilisation : Assigner une Équipe

Acteur Principal : Utilisateur Tunisie Sécurité
Préconditions :

— L'utilisateur est authentifié avec le rôle Tunisie Sécurité
— Une demande existe avec statut VALIDATED
— Des agents de sécurité sont disponibles

Scénario nominal :

1. L'utilisateur accède à la liste des demandes validées (statut = VALIDATED)
2. L'utilisateur sélectionne une demande à traiter
3. L'utilisateur clique sur "Assigner une équipe"
4. Le système affiche un dialog modal avec un formulaire
5. Le formulaire contient :
   — Dropdown "Chauffeur" avec liste des chauffeurs disponibles
   — Dropdown "Garde" avec liste des gardes disponibles
   — Champ "Notes" optionnel
6. L'utilisateur sélectionne un chauffeur dans la liste
7. L'utilisateur sélectionne un garde dans la liste (différent du chauffeur)
8. L'utilisateur ajoute des notes si nécessaire
9. L'utilisateur clique sur "Assigner"
10. Le système valide que chauffeur garde
11. Le système vérifie la disponibilité des deux agents
12. Le système met à jour le statut → ASSIGNED
13. Le système enregistre les IDs des agents, la date et l'utilisateur
14. Le système crée un log d'action avec détails de l'équipe
15. Le système envoie des notifications (Agence + Caisse Centrale)
16. Le système affiche un message de succès
17. La vue est rafraîchie

Postconditions :

— La demande passe au statut ASSIGNED
— Une équipe est assignée (chauffeur + garde identifiés)
— La demande est prête pour dispatch

5.5.4 Diagramme de Classes - Entités Validation/Assignment

La figure 5.2 présente le modèle de classes étendu avec les entités d'audit.

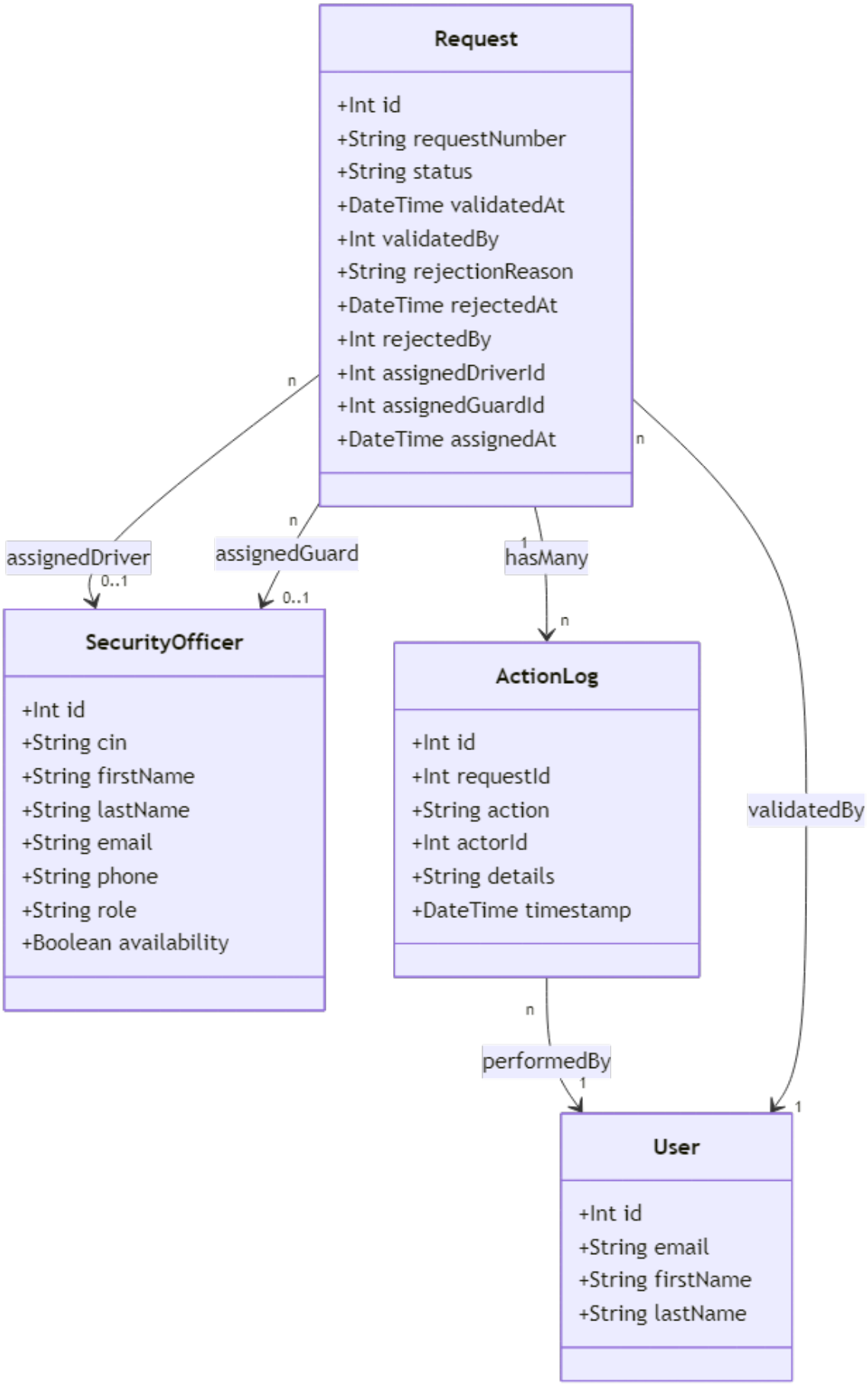

FIGURE 5.2 – Diagramme de classes - Sprint 3

Les nouvelles entités et attributs :

Request (étendu) :

- validatedAt, validatedBy (références User)
- rejectionReason, rejectedAt, rejectedBy
- assignedDriverId, assignedGuardId, assignedAt

SecurityOfficer : Représente un agent de sécurité

- cin, firstName, lastName, email, phone
- role (DRIVER | GUARD)
- availability (boolean)

ActionLog : Enregistre chaque action sur une demande

- requestId, action, actorId, details, timestamp

Le modèle Prisma étendu :

```
model Request {
  // ... champs existants

  validatedAt     DateTime?
  validatedBy     User?     @relation("ValidatedBy", fields: [
     validatedById], references: [id])
  validatedById   Int?

  rejectionReason String?
  rejectedAt      DateTime?
  rejectedBy      User?     @relation("RejectedBy", fields: [
     rejectedById], references: [id])
  rejectedById    Int?

  assignedDriver   SecurityOfficer? @relation("Driver", fields: [
     assignedDriverId], references: [id])
  assignedDriverId Int?
  assignedGuard    SecurityOfficer? @relation("Guard", fields: [
     assignedGuardId], references: [id])
  assignedGuardId  Int?
  assignedAt       DateTime?

  actionLogs ActionLog[]
}

model SecurityOfficer {
  id           Int     @id @default(autoincrement())
  cin          String  @unique
  firstName    String
  lastName     String
  email        String  @unique
  phone        String
  role         String  // "DRIVER" | "GUARD"
  availability Boolean @default(true)

  driverAssignments Request[] @relation("Driver")
  guardAssignments  Request[] @relation("Guard")

  @@index([role, availability])
}
```

```
model ActionLog {
  id        Int      @id @default(autoincrement())
  request   Request  @relation(fields: [requestId], references: [id],
     onDelete: Cascade)
  requestId Int
  action    String   // "VALIDATED", "REJECTED", "ASSIGNED", etc.
  actor     User     @relation(fields: [actorId], references: [id])
  actorId   Int
  details   String?  // JSON string with additional info
  timestamp DateTime @default(now())

  @@index([requestId])
  @@index([timestamp])
}
```

5.5.5 Diagramme de Séquence - Validation

La figure 5.3 illustre le processus de validation d'une demande.

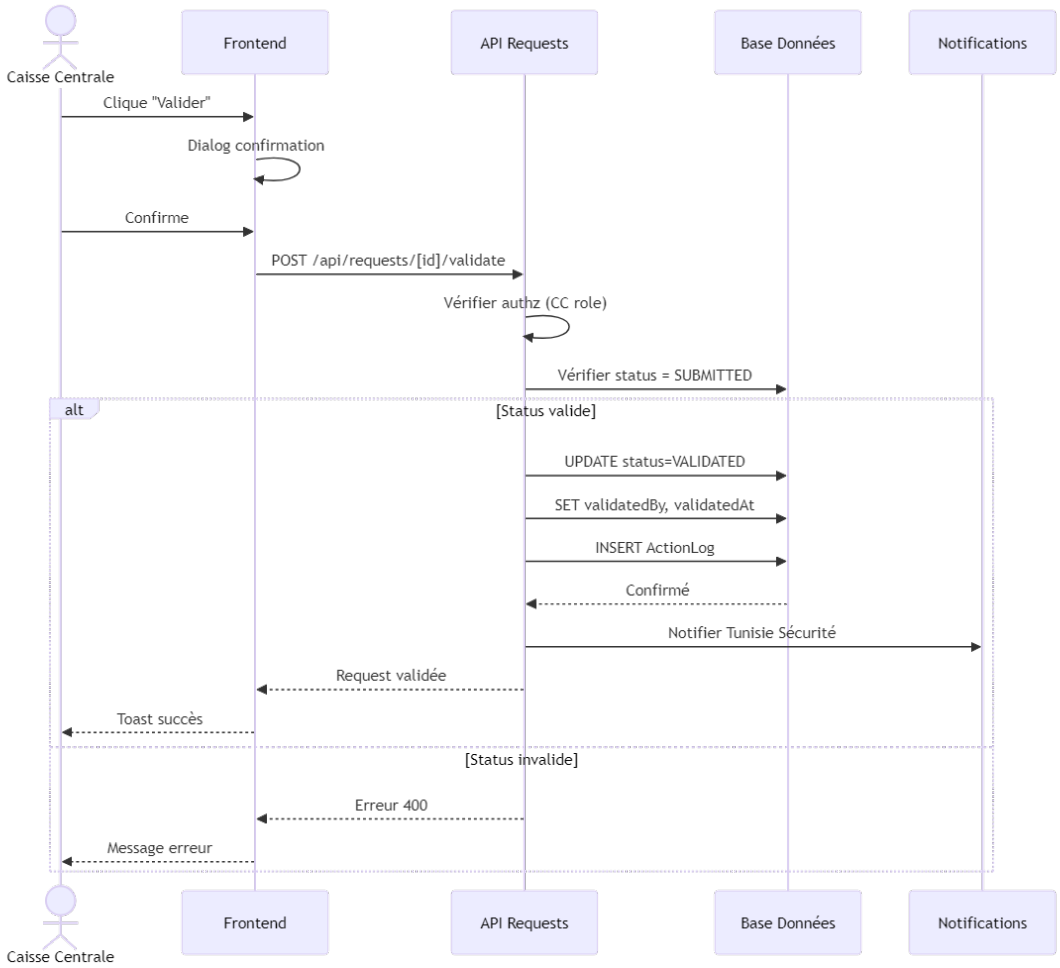

FIGURE 5.3 – Diagramme de séquence - Validation de demande

Le processus détaillé :

1. L'utilisateur Caisse Centrale clique sur "Valider"

2. Le frontend affiche un dialog de confirmation
3. L'utilisateur confirme
4. Requête POST envoyée à `/api/requests/[id]/validate`
5. L'API vérifie l'authentification et le rôle (CaisseCentrale)
6. L'API vérifie que le statut actuel = SUBMITTED
7. L'API démarre une transaction Prisma
8. L'API met à jour la demande : status=VALIDATED, validatedBy, validatedAt
9. L'API crée un ActionLog
10. L'API commit la transaction
11. L'API déclenche une notification (queue ou email)
12. L'API retourne la demande mise à jour
13. Le frontend affiche un toast de succès
14. Le frontend met à jour la vue locale (optimistic update)

5.5.6 Diagramme de Séquence - Assignment Équipe

La figure 5.4 montre le processus d'assignment d'une équipe.

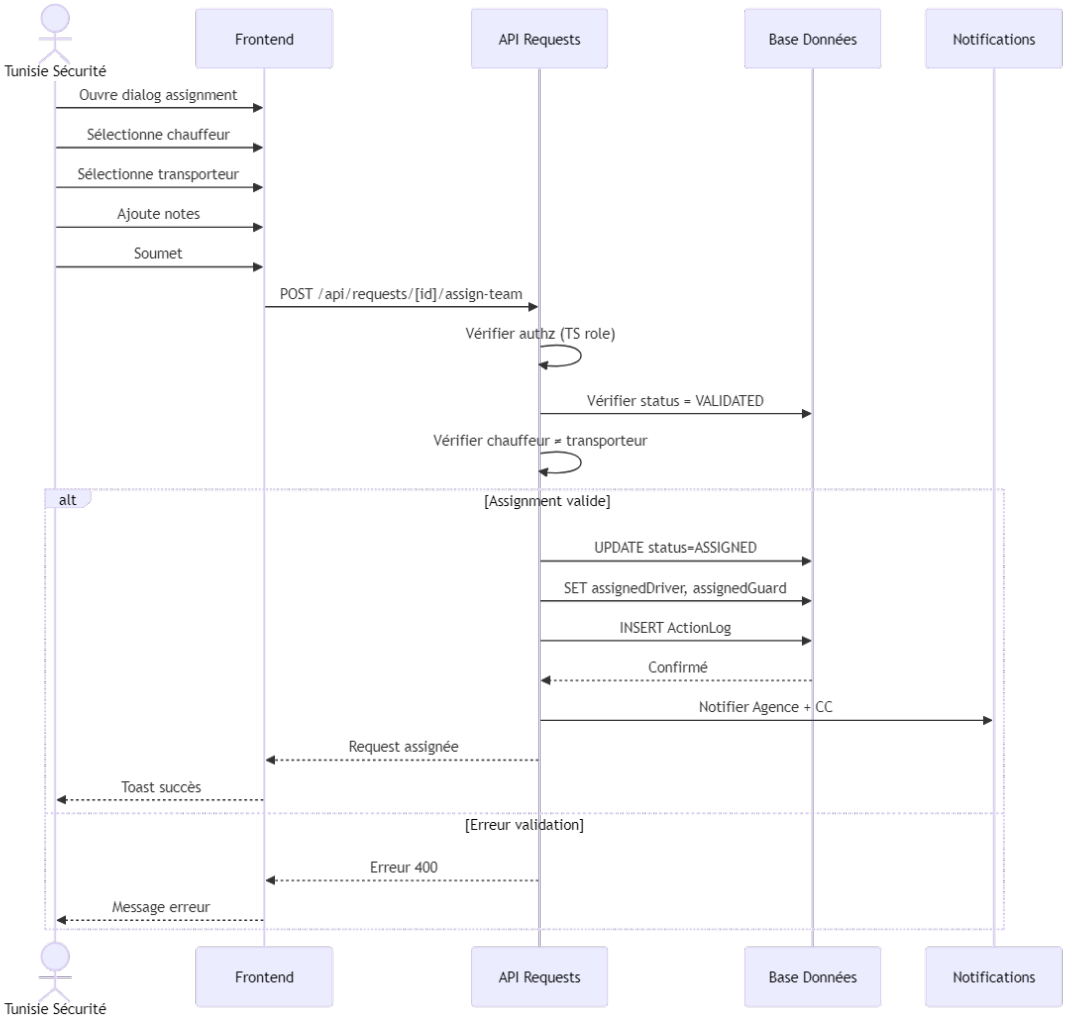

FIGURE 5.4 – Diagramme de séquence - Assignment d'équipe

Les étapes détaillées :

1. Tunisie Sécurité remplit le formulaire (driverId, guardId, notes)
2. Le frontend valide : driverId guardId
3. Requête POST à `/api/requests/[id]/assign-team`
4. L'API vérifie l'authentification et le rôle
5. L'API vérifie statut = VALIDATED
6. L'API vérifie l'existence et la disponibilité du chauffeur
7. L'API vérifie l'existence et la disponibilité du garde
8. L'API démarre une transaction
9. L'API met à jour la demande avec assignedDriverId, assignedGuardId, status=ASSIGNED
10. L'API crée un ActionLog avec détails de l'équipe
11. L'API commit la transaction
12. L'API déclenche des notifications multiples
13. L'API retourne la demande
14. Le frontend affiche succès

5.5.7 Diagramme d'Activités - Validation Workflow

La figure 5.5 présente le flux décisionnel de validation.

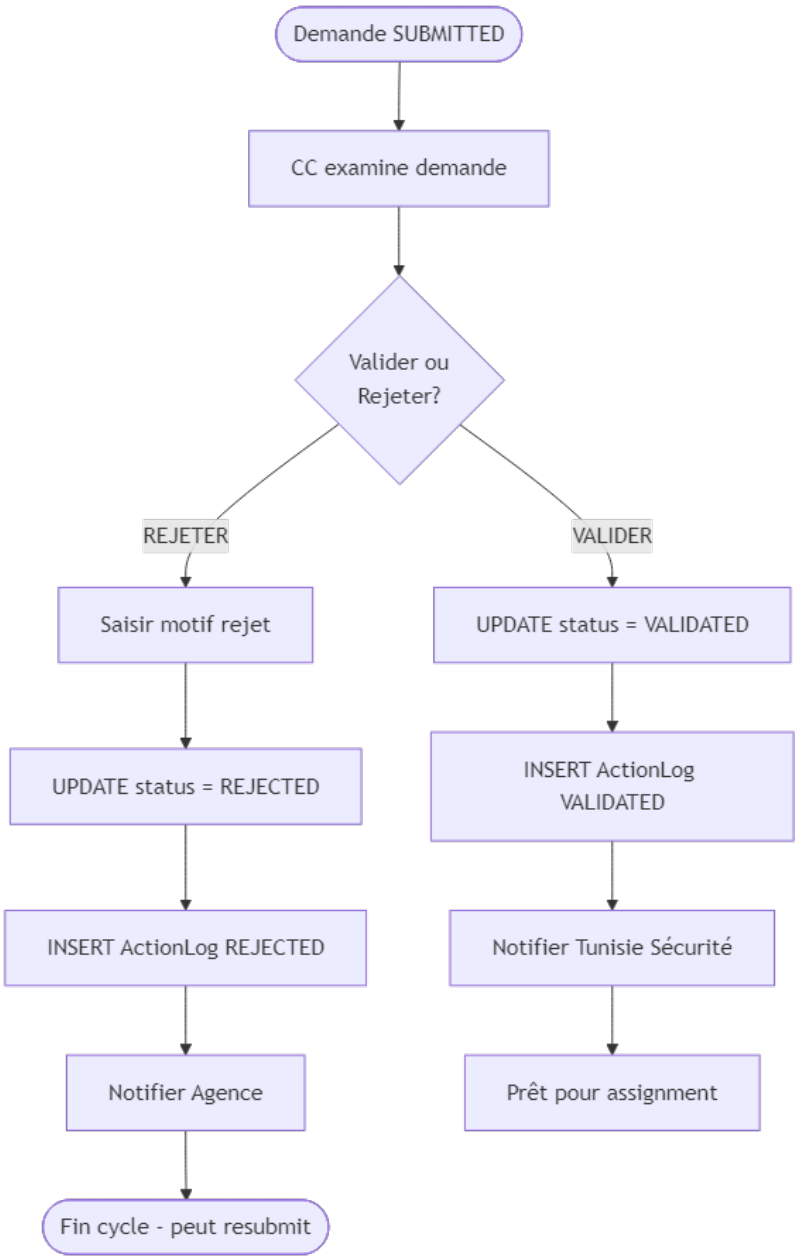

FIGURE 5.5 – Diagramme d'activités - Workflow de validation

Le flux inclut :

— Point de décision : La demande est-elle acceptable ?
— Si OUI : Validation → statut VALIDATED → notifier Tunisie Sécurité
— Si NON : Demander motif → Rejet → statut REJECTED → notifier Agence
— Chaque chemin enregistre un log d'action

5.6 Conception Détaillée

5.6.1 Architecture Composants Frontend

Structure des composants :

```
components/
 validate-dialog.tsx        # Dialog validation simple
 reject-dialog.tsx          # Dialog rejet avec motif
 assign-team-dialog.tsx     # Dialog assignment équipe
 team-selector.tsx          # Sélecteur chauffeur/garde
 action-timeline.tsx        # Timeline des actions

app/requests/[id]/
 page.tsx                   # Vue détails avec actions conditionnelles
```

5.6.2 Routes API

`POST /api/requests/[id]/validate` : Validation de demande

- Authorization : Rôle CaisseCentrale uniquement
- Body : `{}` (pas de données supplémentaires)
- Retourne : Request avec status=VALIDATED

`POST /api/requests/[id]/reject` : Rejet de demande

- Authorization : Rôle CaisseCentrale uniquement
- Body : `{ rejectionReason: string }`
- Validation : rejectionReason min 10 caractères
- Retourne : Request avec status=REJECTED

`POST /api/requests/[id]/assign-team` : Assignment équipe

- Authorization : Rôle TunisieSécurité uniquement
- Body : `{ driverId: number, guardId: number, notes?: string }`
- Validation : driverId guardId, disponibilité des deux
- Retourne : Request avec status=ASSIGNED

`GET /api/requests/[id]/history` : Historique des actions

- Retourne : ActionLog[] ordonnés par timestamp DESC

`GET /api/security-officers` : Liste des agents

- Query params : role (DRIVER|GUARD), available (true|false)
- Retourne : SecurityOfficer[]

5.6.3 Middleware - Vérification Permissions

Protection des endpoints par rôle :

```
// lib/auth-middleware.ts
export function requireRole(allowedRoles: string[]) {
  return async (req: Request) => {
    const session = await getServerSession();

    if (!session) {
      return Response.json(
        { error: 'Non authentifie' },
        { status: 401 }
      );
    }
```

```
    if (!allowedRoles.includes(session.user.role)) {
      return Response.json(
        { error: 'Permission refusee' },
        { status: 403 }
      );
    }

    return null; // Permission OK
  };
}

// Usage dans API route
export async function POST(request: Request, { params }) {
  const authError = await requireRole(['CAISSE_CENTRALE'])(request);
  if (authError) return authError;

  // ... logique de validation
}
```

5.7 Réalisation - Implémentation

5.7.1 Dialog Validation

Composant simple de confirmation :

— Header : "Valider la demande #REQ-2024-001"
— Body : Résumé (montant, agence, type)
— Question : "Confirmez-vous la validation de cette demande ?"
— Actions : "Valider" (vert), "Annuler" (gris)
— Loader pendant l'appel API
— Toast de succès après validation

5.7.2 Dialog Rejet

Composant avec formulaire :

— Header : "Rejeter la demande"
— Body : Résumé de la demande
— Form :
  — Label : "Motif du rejet *"
  — Textarea (min 10 chars, max 500)
  — Helper text : "Veuillez expliquer la raison du rejet"
  — Validation en temps réel (compte de caractères)
— Actions : "Rejeter" (rouge, disabled si motif invalide), "Annuler"
— Toast après succès avec redirection

5.7.3 Dialog Assignment Équipe

Composant complexe avec sélecteurs :

- Header : "Assigner une équipe"
- Résumé demande (numéro, agence, montant)
- Form :
  - Select "Chauffeur" :
    - Options : CIN - Nom Prénom
    - Filtrage par disponibilité (disponibles en premier)
    - Recherche textuelle
  - Select "Garde" :
    - Options : CIN - Nom Prénom
    - Exclut le chauffeur sélectionné
    - Filtrage disponibilité
  - Textarea "Notes" (optionnel)
- Validation :
  - Les deux sélections obligatoires
  - Chauffeur Garde
  - Message d'erreur si même personne
- Actions : "Assigner" (disabled si invalide), "Annuler"

5.7.4 Component Timeline Actions

Affichage chronologique des événements :

- Structure verticale avec ligne de connexion
- Pour chaque action :
  - Icône (validation, × rejet, assignment, dispatch, réception)
  - Date et heure formatées
  - Type d'action (badge coloré)
  - Acteur : "Par Nom Prénom (Rôle)"
  - Détails supplémentaires :
    - Si rejet : affichage du motif
    - Si assignment : affichage des noms chauffeur/garde
    - Si non-conformité : raison
- Animation au chargement (fade-in séquentiel)

5.7.5 Affichage Conditionnel des Actions

Dans la vue détails, les actions disponibles dépendent du statut et du rôle :

```
// Logique d'affichage des boutons
const canValidate =
  userRole === 'CAISSE_CENTRALE' &&
  request.status === 'SUBMITTED';

const canReject =
  userRole === 'CAISSE_CENTRALE' &&
  request.status === 'SUBMITTED';

const canAssign =
  userRole === 'TUNISIE_SECURITE' &&
  request.status === 'VALIDATED';

return (
  <div className="actions">
    {canValidate && <ValidateButton />}
    {canReject && <RejectButton />}
    {canAssign && <AssignTeamButton />}
  </div>
);
```

5.8 Tests et Validation

5.8.1 Tests Fonctionnels

Validation

✓ Validation change statut SUBMITTED → VALIDATED
✓ Utilisateur et date enregistrés correctement
✓ ActionLog créé avec détails
✓ Seul CaisseCentrale peut valider
✓ Impossible de valider deux fois

Rejet

✓ Rejet avec motif fonctionne
✓ Motif obligatoire (erreur si vide)
✓ Motif stocké correctement
✓ Notification envoyée à agence
✓ Statut passe à REJECTED

Assignment

✓ Assignment avec chauffeur et garde différents réussit
✓ Erreur si même personne pour les deux rôles
✓ Erreur si agent indisponible
✓ Statut passe à ASSIGNED
✓ Seul TunisieSécurité peut assigner

Traçabilité

✓ Chaque action crée un log
✓ Timeline affiche tous les logs ordonnés
✓ Détails spécifiques présents (motif, équipe)
✓ Timestamp précis enregistré

5.8.2 Tests de Sécurité

✓ Agence ne peut pas valider
✓ Agence ne peut pas assigner équipe
✓ CaisseCentrale ne peut pas assigner
✓ TunisieSécurité ne peut pas valider/rejeter
✓ Requêtes non authentifiées refusées (401)
✓ Requêtes sans permission refusées (403)

5.9 Revue de Sprint

5.9.1 Démonstration

Démonstration complète du workflow :

1. Connexion en tant que Caisse Centrale
2. Consultation liste demandes SUBMITTED
3. Examen détails d'une demande
4. Validation de la demande
5. Vérification du changement de statut et log
6. Déconnexion et reconnexion en Tunisie Sécurité
7. Consultation demandes VALIDATED
8. Assignment d'une équipe (sélection chauffeur + garde)
9. Vérification statut ASSIGNED et timeline complète
10. Test de rejet : création demande, tentative rejet sans motif (erreur), rejet avec motif (succès)

5.9.2 Validation Product Owner

Points validés :

— Workflow clair et intuitif
— Sécurité des permissions robuste
— Traçabilité complète (audit trail)
— Ergonomie des dialogs (simples et efficaces)
— Performance (actions instantanées)

5.9.3 Feedback

Suggestions :

— Ajouter filtres sur disponibilité agents dans liste [✓ Implémenté]

— Permettre modification de l'équipe si encore en ASSIGNED [*Backlog*]

— Afficher statistiques de validation (taux, délai moyen) [*Sprint 4*]

5.9.4 Métriques du Sprint

— **Durée** : 3 semaines

— **User Stories complétées** : 6/6 (100%)

— **Points de complexité** : 34/34

— **Bugs identifiés** : 4 (tous résolus)

— **Couverture permissions** : 100% (tous rôles testés)

5.10 Conclusion du Chapitre

Le Sprint 3 a permis d'implémenter le workflow critique de validation et d'assignment. Les mécanismes d'approbation multiniveaux garantissent que seules les demandes légitimes sont traitées, tandis que le système d'assignment optimise l'allocation des ressources logistiques.

La traçabilité complète via les ActionLogs constitue un atout majeur pour l'audit et la conformité réglementaire. Chaque décision est documentée avec son contexte, son acteur et son horodatage, permettant une reconstitution fidèle de l'historique.

Le système est maintenant prêt pour les étapes finales : le dispatch des fonds par Tunisie Sécurité et leur réception par les agences, qui seront développés dans le Sprint 4 avec le module analytics.

Chapitre 6

Sprint 4 - Module Dispatch et Analytics

6.1 Introduction

Le Sprint 4 constitue la phase finale du développement, avec deux objectifs majeurs : finaliser le cycle de traitement des demandes (dispatch et réception) et développer le module analytics pour fournir une visibilité stratégique sur les opérations. Ce sprint transforme le système en une solution complète et opérationnelle.

6.2 Vue d'Ensemble

Le Sprint 4 couvre :

1. **Module Dispatch et Réception** :
   - Confirmation du dispatch par Tunisie Sécurité
   - Réception des fonds par les agences
   - Gestion des non-conformités
2. **Module Analytics** :
   - Tableau de bord avec KPIs
   - Graphiques et visualisations
   - Filtrage temporel et par critères
   - Export des données

6.3 Objectif Principal

L'objectif principal est de compléter le cycle de vie des demandes avec traçabilité jusqu'à la livraison finale, tout en fournissant aux décideurs les outils d'analyse nécessaires pour optimiser les processus et détecter les anomalies.

6.4 User Stories Détaillées

Le tableau 6.1 présente les user stories du Sprint 4.

TABLE 6.1 – User Stories - Sprint 4

ID	En tant que	Je veux	Critères d'acceptation	Priorité
US-12	Tunisie Sécurité	Confirmer le dispatch	Transition ASSIGNED → DISPATCHED	Must
US-13	Agence	Confirmer la réception	Transition DISPATCHED → RECEIVED	Must
US-14	Agence	Signaler non-conformité	Ajouter raison + statut spécial	Should
US-15	Administrateur	Voir analytics globales	Charts, KPIs, tables	Must
US-16	Administrateur	Exporter données	Export Excel/PDF	Should

6.5 Analyse et Spécifications

6.5.1 Diagramme de Cas d'Utilisation - Sprint 4

La figure 6.1 présente les cas d'utilisation du Sprint 4.

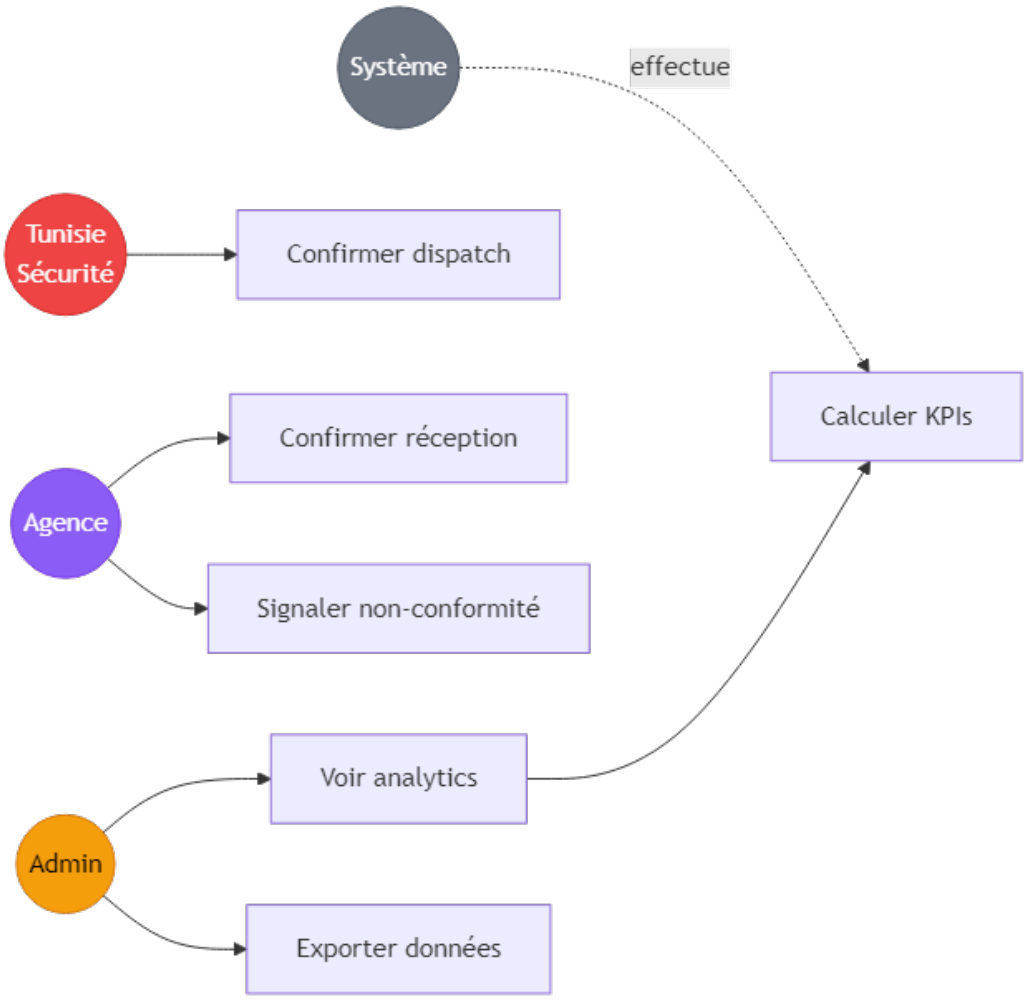

FIGURE 6.1 – Diagramme de cas d'utilisation - Sprint 4

Les cas d'utilisation incluent :

— **Dispatcher demande** : Confirmation départ équipe
— **Recevoir demande** : Confirmation arrivée fonds
— **Déclarer non-conformité** : Signalement problème
— **Consulter analytics** : Vue KPIs et graphiques
— **Exporter données** : Génération rapports

6.5.2 Raffinements Textuels - Scénario Dispatch

Cas d'utilisation : Confirmer Dispatch

Acteur Principal : Utilisateur Tunisie Sécurité
Préconditions :

— L'utilisateur est authentifié avec le rôle Tunisie Sécurité
— Une demande existe avec statut ASSIGNED
— Une équipe est assignée

Scénario nominal :

1. L'utilisateur accède à la liste des demandes assignées (statut = ASSIGNED)
2. L'utilisateur vérifie les détails de la demande et de l'équipe
3. L'équipe est prête à partir avec les fonds
4. L'utilisateur clique sur "Confirmer Dispatch"
5. Le système affiche un dialog de confirmation
6. Dialog affiche : numéro demande, agence destination, équipe assignée
7. L'utilisateur confirme le dispatch
8. Le système met à jour status → DISPATCHED
9. Le système enregistre dispatchedAt (timestamp) et dispatchedBy (userId)
10. Le système crée un ActionLog
11. Le système envoie une notification à l'agence destinataire
12. Le système affiche un message de succès
13. La demande disparaît de la liste ASSIGNED

Postconditions :

— La demande est en transit (status = DISPATCHED)
— L'agence est notifiée de l'arrivée imminente
— Les fonds sont en route

6.5.3 Raffinements Textuels - Scénario Réception

Cas d'utilisation : Recevoir Demande

Acteur Principal : Utilisateur Agence
Préconditions :

— L'utilisateur est authentifié avec le rôle Agence
— Une demande de son agence existe avec statut DISPATCHED
— Les fonds sont physiquement arrivés

Scénario nominal :

1. L'utilisateur accède à la liste des demandes en attente de réception
2. L'utilisateur reçoit physiquement les fonds de l'équipe
3. L'utilisateur vérifie la conformité (montant, coupures)
4. L'utilisateur clique sur "Confirmer Réception"
5. Le système affiche un dialog avec formulaire
6. Formulaire contient :
   — Checkbox "Conforme ?" (checked par défaut)
   — Si décoché : Textarea "Raison de non-conformité" (obligatoire)
   — Textarea "Notes additionnelles" (optionnel)
7. L'utilisateur confirme que tout est conforme
8. L'utilisateur clique sur "Recevoir"
9. Le système met à jour status → RECEIVED

10. Le système enregistre receivedAt, receivedBy
11. Le système crée un ActionLog
12. Le système envoie notifications (Caisse Centrale, Tunisie Sécurité)
13. Le système affiche un message de succès
14. Le cycle est terminé

Postconditions :

— La demande est complétée (status = RECEIVED ou COMPLETED)
— Les fonds sont entre les mains de l'agence
— Tous les acteurs sont notifiés

Scénario Alternatif : Non-Conformité

Point de divergence : Étape 3 du scénario nominal

1. L'utilisateur constate une différence (montant manquant, coupures incorrectes)
2. L'utilisateur décoche "Conforme ?"
3. Le champ "Raison de non-conformité" devient obligatoire
4. L'utilisateur saisit une description détaillée du problème
5. L'utilisateur ajoute des notes si nécessaire
6. L'utilisateur clique sur "Recevoir avec réserve"
7. Le système valide la présence de la raison
8. Le système met à jour status → COMPLETED_WITH_ISSUES
9. Le système enregistre nonConformityReason, receivedAt, receivedBy
10. Le système crée un ActionLog avec flag problème
11. Le système envoie des alertes (Caisse Centrale, Tunisie Sécurité, Admin)
12. Le système affiche un message d'avertissement
13. Une investigation peut être déclenchée

6.5.4 Diagramme de Séquence - Dispatch

La figure 6.2 illustre le processus de dispatch.

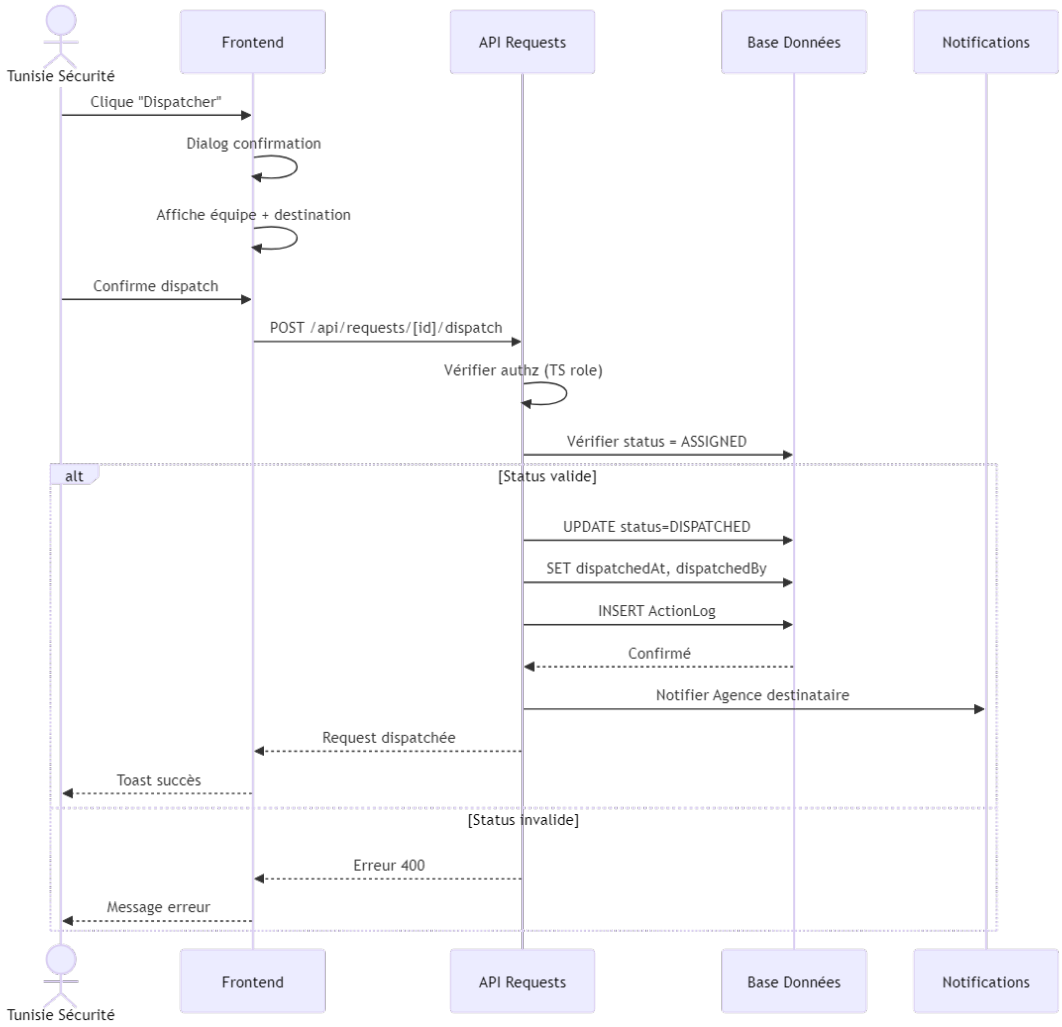

FIGURE 6.2 – Diagramme de séquence - Dispatch

6.5.5 Diagramme de Séquence - Réception

La figure 6.3 montre le processus de réception avec gestion de conformité.

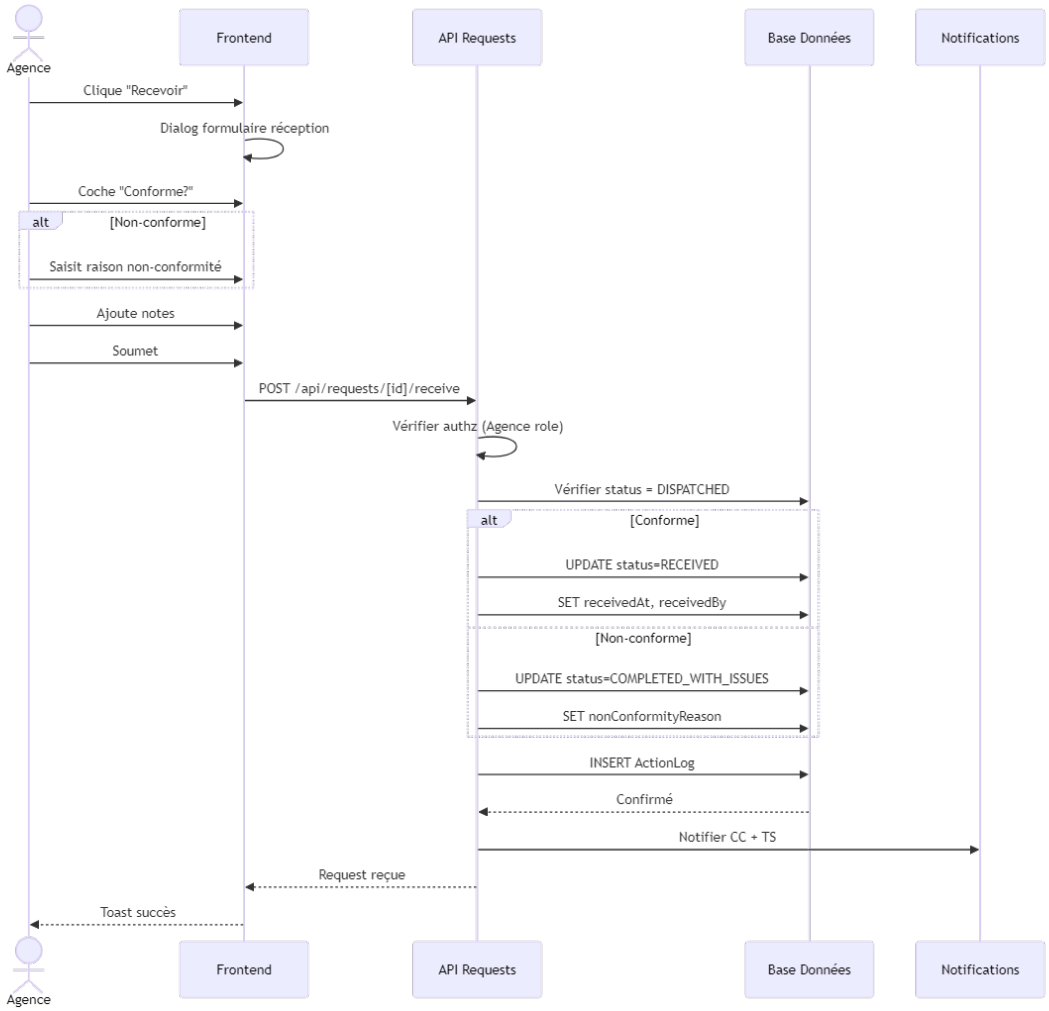

FIGURE 6.3 – Diagramme de séquence - Réception

6.5.6 Diagramme d'Activités - Flux Réception

La figure 6.4 présente le flux décisionnel de réception.

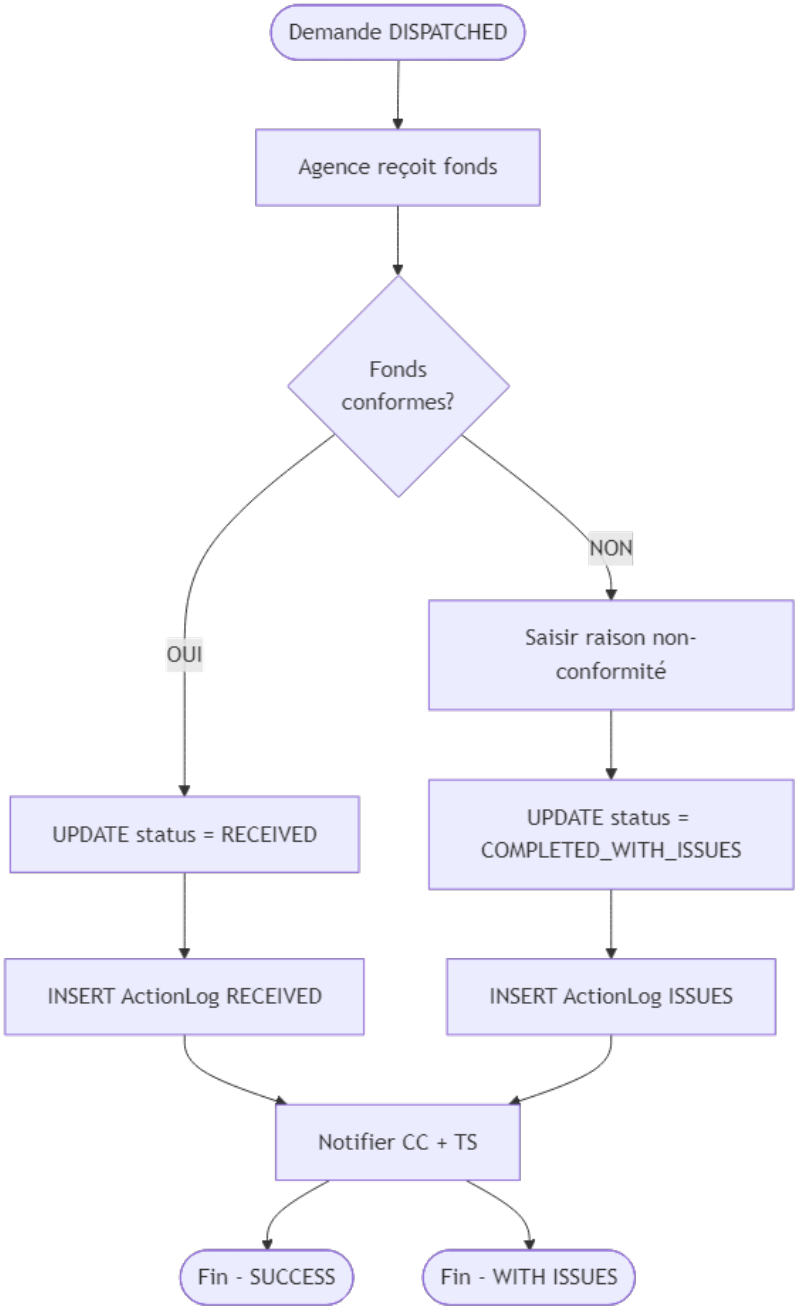

FIGURE 6.4 – Diagramme d'activités - Flux de réception

Le flux inclut la vérification de conformité et les chemins alternatifs selon le résultat.

6.5.7 Spécifications Analytics

KPIs à Afficher

1. **Nombre total de demandes** : Compteur global avec trend
2. **Montant total transité** : Somme en DT avec trend
3. **Demandes en cours** : Nombre de demandes non finalisées
4. **Délai moyen de traitement** : Temps moyen de SUBMITTED à RECEIVED (en jours)

5. **Taux de rejet** : Pourcentage de demandes rejetées
6. **Taux de non-conformité** : Pourcentage de réceptions avec problèmes

Graphiques

1. **Pie Chart - Demandes par Statut** :
   - Segments colorés par statut
   - Labels avec nombres et pourcentages
   - Légende interactive
2. **Bar Chart - Demandes par Agence** :
   - Top 10 agences par volume
   - Axe secondaire pour montant total
   - Tri décroissant
3. **Line Chart - Évolution Temporelle** :
   - Nombre de demandes par jour/semaine/mois
   - Ligne de tendance
   - Zoom et pan
4. **Bar Chart - Délais par Étape** :
   - Temps moyen : SUBMITTED→VALIDATED, VALIDATED→ASSIGNED, etc.
   - Identification des goulots d'étranglement

Tables Agrégées

1. **Top Agences** :
   - Colonnes : Nom, Nombre demandes, Montant total, Taux rejet
   - Tri par volume
2. **Demandes Récentes** :
   - Dernières 10 demandes
   - Colonnes : Numéro, Agence, Montant, Statut, Date

Filtres Analytics

- **Plage de dates** : Date picker range (de - à)
- **Agences** : Multi-select agences
- **Statuts** : Multi-select statuts
- **Type** : Provisionnement / Versement / Tous
- **Bouton "Actualiser"** : Recharge les données

6.6 Conception Détaillée

6.6.1 Architecture Composants Frontend

Structure des composants analytics :

```
components/dashboard/
 request-stats-chart.tsx        # Pie chart statuts
 agency-volume-chart.tsx        # Bar chart agences
 timeline-chart.tsx             # Line chart évolution
 analytics-kpi-cards.tsx        # Cartes KPIs
 analytics-table.tsx            # Tables données
 analytics-filters.tsx          # Filtres

app/admin/analytics/
 page.tsx                       # Page dashboard admin
```

6.6.2 Routes API

`POST /api/requests/[id]/dispatch` : Confirmation dispatch

- Authorization : TunisieSécurité uniquement
- Body : `{}`
- Retourne : Request avec status=DISPATCHED

`POST /api/requests/[id]/receive` : Confirmation réception

- Authorization : Agence uniquement (sa propre demande)
- Body : `{ conforme: boolean, nonConformityReason?: string, notes?: string }`
- Validation : Si conforme=false, raison obligatoire
- Retourne : Request avec status=RECEIVED ou COMPLETED_WITH_ISSUES

`GET /api/analytics/summary` : KPIs globaux

- Query : dateFrom, dateTo, agencyIds[], statuses[]
- Retourne : `{ totalRequests, totalAmount, averageDelay, rejectionRate, nonConformityRate }`

`GET /api/analytics/by-status` : Distribution par statut

- Retourne : `[{ status, count }]`

`GET /api/analytics/by-agency` : Agrégation par agence

- Retourne : `[{ agencyName, count, totalAmount, rejectionRate }]`

`GET /api/analytics/timeline` : Évolution temporelle

- Query : granularity (day|week|month)
- Retourne : `[{ date, count, amount }]`

`GET /api/analytics/export` : Export données

- Query : format (excel|pdf), filters
- Retourne : File download

6.6.3 Calculs KPIs (Backend)

Requêtes Prisma pour les KPIs :

```
// Nombre total
const totalRequests = await prisma.request.count({
  where: { createdAt: { gte: dateFrom, lte: dateTo } }
});

// Montant total
const { _sum } = await prisma.request.aggregate({
  _sum: { totalAmount: true },
  where: { createdAt: { gte: dateFrom, lte: dateTo } }
});

// Delai moyen (en jours)
const completedRequests = await prisma.request.findMany({
  where: {
    status: { in: ['RECEIVED', 'COMPLETED_WITH_ISSUES'] },
    receivedAt: { not: null }
  },
  select: { createdAt: true, receivedAt: true }
});

const averageDelay = completedRequests.reduce((sum, req) => {
  const delay = (req.receivedAt - req.createdAt) / (1000 * 60 * 60 *
     24);
  return sum + delay;
}, 0) / completedRequests.length;

// Taux de rejet
const rejectedCount = await prisma.request.count({
  where: { status: 'REJECTED' }
});
const rejectionRate = (rejectedCount / totalRequests) * 100;

// Taux non-conformite
const nonConformCount = await prisma.request.count({
  where: { status: 'COMPLETED_WITH_ISSUES' }
});
const nonConformityRate = (nonConformCount / completedRequests.length)
    * 100;
```

6.7 Réalisation - Implémentation

6.7.1 Dialog Dispatch

Simple dialog de confirmation :

- Header : "Confirmer le dispatch"
- Body :
  - Numéro de demande
  - Agence destination
  - Équipe assignée (Chauffeur + Garde)

  - Montant à transporter
- Question : "Confirmez-vous le départ de l'équipe ?"
- Actions : "Confirmer" (bleu), "Annuler"
- Toast succès après dispatch

6.7.2 Dialog Réception

Dialog avec formulaire conditionnel :

- Header : "Recevoir la demande"
- Body : Résumé (numéro, équipe, montant attendu)
- Form :
  - Checkbox "La livraison est conforme" (checked)
  - Textarea "Raison de non-conformité" (hidden si conforme, required si non-conforme)
  - Textarea "Notes additionnelles" (optionnel, toujours visible)
- Logic : onChange du checkbox toggle required sur raison
- Actions : "Recevoir" (vert si conforme, orange si non-conforme), "Annuler"
- Toast succès (vert) ou warning (orange) selon conformité

6.7.3 Page Dashboard Analytics

Layout de la page :

1. **Header** :
   - Titre "Analytics - Tableau de Bord"
   - Bouton "Exporter" avec dropdown (Excel, PDF)
2. **Section Filtres** :
   - Date range picker (par défaut : 30 derniers jours)
   - Multi-select agences
   - Multi-select statuts
   - Bouton "Appliquer filtres"
3. **Section KPIs** (grille 3 colonnes) :
   - Card "Demandes totales" avec icône et trend
   - Card "Montant total" formaté
   - Card "En cours" avec badge
   - Card "Délai moyen" en jours
   - Card "Taux rejet" en pourcentage
   - Card "Non-conformité" en pourcentage
4. **Section Graphiques** (grille 2 colonnes) :
   - Pie chart demandes par statut
   - Bar chart demandes par agence
   - Line chart évolution temporelle (pleine largeur)
5. **Section Tables** :
   - Table "Top 10 Agences"
   - Table "Demandes Récentes"

6.7.4 Composant KPI Card

Structure d'une carte KPI :

```
interface KPICardProps {
  title: string;
  value: number | string;
  icon: ReactNode;
  trend?: { value: number; direction: 'up' | 'down' };
  format?: 'number' | 'currency' | 'percentage';
}

function KPICard({ title, value, icon, trend, format }: KPICardProps)
   {
  const formattedValue = formatValue(value, format);

  return (
    <Card>
      <div className="flex items-center justify-between">
        <div>
          <p className="text-sm text-gray-500">{title}</p>
          <p className="text-3xl font-bold">{formattedValue}</p>
          {trend && (
            <p className={`text-sm ${trend.direction === 'up' ? 'text-
               green-600' : 'text-red-600'}`}>
              {trend.direction === 'up' ? '   ' : '   '} {Math.abs(
                 trend.value)}%
            </p>
          )}
        </div>
        <div className="text-4xl text-blue-500">{icon}</div>
      </div>
    </Card>
  );
}
```

6.7.5 Charts avec Recharts

Implémentation des graphiques avec la bibliothèque Recharts :

```
import { PieChart, Pie, Cell, BarChart, Bar, LineChart, Line } from '
   recharts';

// Pie Chart Statuts
function RequestStatsChart({ data }) {
  const COLORS = {
    SUBMITTED: '#3b82f6',
    VALIDATED: '#10b981',
    REJECTED: '#ef4444',
    ASSIGNED: '#f59e0b',
    DISPATCHED: '#8b5cf6',
    RECEIVED: '#06b6d4',
  };

  return (
    <PieChart width={400} height={300}>
      <Pie
        data={data}
```

```
        dataKey="count"
        nameKey="status"
        cx="50%"
        cy="50%"
        outerRadius={100}
        label
      >
        {data.map((entry, index) => (
          <Cell key={index} fill={COLORS[entry.status]} />
        ))}
      </Pie>
      <Tooltip />
      <Legend />
    </PieChart>
  );
}
```

6.7.6 État Zustand - Analytics Store

```
interface AnalyticsState {
  stats: AnalyticsSummary | null;
  filters: {
    dateFrom: Date;
    dateTo: Date;
    agencyIds: number[];
    statuses: string[];
  };
  loading: boolean;
  setStats: (stats: AnalyticsSummary) => void;
  setFilters: (filters: Partial<FilterState>) => void;
  setLoading: (loading: boolean) => void;
}

export const useAnalyticsStore = create<AnalyticsState>((set) => ({
  stats: null,
  filters: {
    dateFrom: new Date(Date.now() - 30 * 24 * 60 * 60 * 1000), // 30
       jours
    dateTo: new Date(),
    agencyIds: [],
    statuses: [],
  },
  loading: false,
  setStats: (stats) => set({ stats }),
  setFilters: (newFilters) =>
    set((state) => ({
      filters: { ...state.filters, ...newFilters }
    })),
  setLoading: (loading) => set({ loading }),
}));
```

6.8 Tests et Validation

6.8.1 Tests Fonctionnels

Dispatch

✓ Dispatch change statut ASSIGNED → DISPATCHED
✓ Timestamp et utilisateur enregistrés
✓ Seul TunisieSécurité peut dispatcher
✓ Notification envoyée à agence

Réception

✓ Réception conforme crée RECEIVED
✓ Réception non-conforme crée COMPLETED_WITH_ISSUES
✓ Raison obligatoire si non-conforme
✓ Seule l'agence concernée peut recevoir
✓ Alertes envoyées si problème

Analytics

✓ KPIs calculés correctement
✓ Graphiques affichent données réelles
✓ Filtres modifient les résultats
✓ Export Excel génère fichier
✓ Performance acceptable (<2s chargement)

6.8.2 Tests d'Intégration End-to-End

Test du cycle complet :

1. Agence crée demande → SUBMITTED
2. Caisse Centrale valide → VALIDATED
3. Tunisie Sécurité assigne équipe → ASSIGNED
4. Tunisie Sécurité dispatche → DISPATCHED
5. Agence reçoit → RECEIVED
6. Vérification : Tous les logs présents, timeline complète
7. Analytics : Demande apparaît dans statistiques

✓ Cycle complet sans erreur
✓ Toutes les transitions valides
✓ Logs chronologiques corrects
✓ Analytics à jour

6.9 Revue de Sprint

6.9.1 Démonstration Finale

Démonstration complète au Product Owner :

1. Workflow end-to-end : Création → Validation → Assignment → Dispatch → Réception
2. Gestion de non-conformité : Réception avec problème
3. Dashboard analytics : Présentation de tous les KPIs et graphiques
4. Filtrage analytics par période et agence
5. Export Excel des données
6. Timeline complète d'une demande avec tous les logs

6.9.2 Validation Product Owner

Points validés :

— Système complet et opérationnel
— Cycle de vie des demandes fluide
— Analytics riches et pertinents
— Interface intuitive pour tous les rôles
— Performance satisfaisante
— Sécurité et traçabilité assurées

6.9.3 Feedback

Suggestions pour évolutions futures :

— Notifications push en temps réel (WebSockets)
— Export PDF avec graphiques inclus
— Module de reporting avancé (rapports programmés)
— Application mobile pour consultation
— Signature électronique pour réception

6.9.4 Métriques du Sprint

— **Durée** : 2 semaines
— **User Stories complétées** : 5/5 (100%)
— **Points de complexité** : 28/28
— **Bugs identifiés** : 3 (tous résolus)
— **Performance** : $<$ 2s chargement analytics

6.9.5 Métriques Globales du Projet

— **Durée totale** : 10 semaines (4 sprints)
— **User Stories totales** : 24/24 (100%)
— **Points de complexité** : 117/117
— **Lignes de code** : 8,500 (frontend + backend)
— **Tests** : Coverage manuel complet
— **Bugs résolus** : 15/15

6.10 Conclusion du Chapitre

Le Sprint 4 a permis de finaliser le système avec l'implémentation des étapes finales du cycle de traitement (dispatch et réception) et le développement d'un module analytics complet. Le système est désormais opérationnel de bout en bout, couvrant l'intégralité du processus de gestion des fonds bancaires.

Le module analytics offre aux décideurs une visibilité stratégique sur les opérations, avec des KPIs pertinents, des visualisations graphiques et des capacités de filtrage et d'export. Cette transparence facilite l'identification des goulots d'étranglement et l'optimisation continue des processus.

La gestion des non-conformités garantit que les problèmes sont tracés et remontés aux bonnes personnes, permettant une résolution rapide et une amélioration continue de la qualité de service.

Le projet atteint maintenant sa phase de conclusion, où nous synthétisons les réalisations, les compétences acquises et les perspectives d'évolution du système.

Conclusion Générale

Synthèse du Projet

Ce projet de fin d'études a permis de concevoir et développer un **Système de Gestion des Fonds Bancaires** complet et opérationnel pour Amen Bank. Le système répond à une problématique concrète : la nécessité d'automatiser et de sécuriser les processus de gestion des mouvements de fonds entre les agences et la Caisse Centrale.

À travers quatre sprints de développement suivant la méthodologie Scrum, nous avons progressivement construit une solution moderne qui couvre l'intégralité du cycle de vie des demandes de fonds :

1. **Sprint 1 - Architecture et Authentification** : Mise en place des fondations techniques avec un système d'authentification sécurisé et un contrôle d'accès basé sur les rôles (RBAC).
2. **Sprint 2 - Gestion des Demandes** : Développement du module central permettant aux agences de créer des demandes détaillées avec spécification des coupures, et aux différents acteurs de consulter et filtrer les demandes.
3. **Sprint 3 - Validation et Assignment** : Implémentation du workflow multiniveaux avec validation par la Caisse Centrale et assignment des équipes de sécurité par Tunisie Sécurité, incluant un système complet de traçabilité.
4. **Sprint 4 - Dispatch et Analytics** : Finalisation du cycle avec les modules de dispatch, réception et gestion des non-conformités, ainsi que le développement d'un tableau de bord analytics pour la prise de décision stratégique.

Le système développé offre des bénéfices tangibles :

— **Réduction des délais** : Le processus automatisé réduit le temps de traitement d'une demande de plusieurs jours à quelques heures.

— **Élimination des erreurs** : La validation automatique des montants et des coupures élimine les erreurs de saisie et de calcul.

— **Traçabilité complète** : Chaque action est enregistrée avec identification de l'acteur et horodatage, facilitant l'audit et la conformité réglementaire.

— **Sécurité renforcée** : Le contrôle d'accès granulaire et le chiffrement des données sensibles protègent contre les accès non autorisés.

— **Visibilité stratégique** : Le module analytics fournit aux décideurs les indicateurs nécessaires pour optimiser les processus et allouer les ressources efficacement.

Compétences Acquises

Ce projet a été l'occasion de développer et consolider un large éventail de compétences techniques et méthodologiques :

Compétences Techniques

Développement Full-Stack : Maîtrise de Next.js 15 pour le développement d'applications web modernes, avec compréhension approfondie de l'App Router, des Server Components et du Server-Side Rendering.

TypeScript : Utilisation avancée de TypeScript pour garantir la type-safety du code, améliorer la maintenabilité et faciliter le refactoring.

Gestion de Bases de Données : Conception de schémas relationnels complexes avec PostgreSQL, utilisation de Prisma ORM pour l'accès aux données, gestion des migrations et optimisation des requêtes.

Sécurité Applicative : Implémentation de l'authentification avec NextAuth.js, hashing des mots de passe avec bcrypt, gestion des sessions JWT, mise en place du contrôle d'accès basé sur les rôles.

Architecture Logicielle : Application des principes de séparation des responsabilités, conception de workflows complexes avec transitions d'états, gestion des transactions pour garantir l'intégrité des données.

UI/UX Design : Création d'interfaces utilisateur modernes et responsives avec Tailwind CSS, implémentation de composants réutilisables, attention portée à l'accessibilité et à l'ergonomie.

Validation de Données : Utilisation de Zod pour la validation côté client et serveur, définition de schémas réutilisables, génération de messages d'erreur explicites.

Visualisation de Données : Intégration de graphiques interactifs avec Recharts, calcul de KPIs, agrégation de données pour l'analytics.

Compétences Méthodologiques

Méthodologie Agile/Scrum : Application concrète du framework Scrum avec planification de sprints, daily standups, reviews et rétrospectives. Compréhension de l'importance de la flexibilité et du feedback continu.

Gestion de Projet : Priorisation des fonctionnalités avec la méthode MoSCoW, estimation de la complexité, gestion du backlog produit, respect des deadlines.

Modélisation UML : Création de diagrammes de cas d'utilisation, de classes, de séquence et d'activités pour documenter l'architecture et les processus métier.

: Élaboration de scénarios de tests fonctionnels et de sécurité, validation des permissions par rôle, tests d'intégration end-to-end.

: Rédaction de documentation technique claire, commentaires de code pertinents, création de ce rapport détaillé.

Compétences Métiers

Compréhension du Domaine Bancaire : Acquisition de connaissances sur la gestion de trésorerie bancaire, les processus de provisionnement et de versement, les contraintes de sécurité et de conformité.

Analyse des Besoins : Capacité à traduire des besoins métier en spécifications techniques, à identifier les cas d'usage et les scénarios alternatifs.

Communication : Présentation des livrables au Product Owner, ajustement des fonctionnalités selon le feedback, vulgarisation technique pour les parties prenantes non techniques.

Difficultés Rencontrées et Solutions Apportées

Comme tout projet de développement, nous avons été confrontés à plusieurs défis techniques et organisationnels :

Gestion des États Complexes

Difficulté : Le workflow des demandes implique de nombreuses transitions d'états avec des règles métier strictes. Gérer ces transitions tout en garantissant l'intégrité des données représentait un défi.

Solution : Nous avons défini une machine d'états claire avec des transitions explicites. Chaque endpoint API vérifie le statut actuel avant d'autoriser une transition. L'utilisation de transactions Prisma garantit l'atomicité des opérations composées (mise à jour du statut + création du log).

Validation Transactionnelle

Difficulté : Lors de la création d'une demande, il faut créer à la fois l'enregistrement Request et les multiples DenominationDetail associés. Un échec partiel pourrait laisser la base de données dans un état incohérent.

Solution : Utilisation systématique des transactions Prisma pour les opérations composées. En cas d'erreur, un rollback automatique annule toutes les modifications, préservant l'intégrité référentielle.

```
await prisma.$transaction(async (tx) => {
  const request = await tx.request.create({ data: requestData });
  await tx.denominationDetail.createMany({
    data: denominations.map(d => ({ ...d, requestId: request.id }))
  });
  await tx.actionLog.create({ data: logData });
});
```

Sécurité des Sessions JWT

Difficulté : Gérer correctement les tokens JWT (expiration, rafraîchissement, stockage sécurisé) tout en maintenant une expérience utilisateur fluide.

Solution : Configuration de NextAuth.js avec des durées de session adaptées (24h), utilisation de cookies HTTP-only et Secure pour prévenir les attaques XSS et CSRF, vérification systématique de la validité du token dans le middleware.

Performance des Requêtes

Difficulté : Les requêtes de listing des demandes avec joins multiples (agency, user, denominations) pouvaient être lentes avec un grand volume de données.

Solution : Optimisation avec :

— Index sur les colonnes fréquemment filtrées (status, agencyId, requestNumber)
— Utilisation de `select` Prisma pour ne récupérer que les champs nécessaires
— Pagination côté serveur pour limiter le nombre de résultats
— Caching des données de référence (rôles, agences) avec stratégie de revalidation

Gestion des Permissions Granulaires

Difficulté : Certaines actions nécessitent des vérifications complexes (ex : une agence ne peut recevoir que ses propres demandes).

Solution : Création de fonctions d'autorisation réutilisables qui vérifient non seulement le rôle, mais aussi le contexte (agence de l'utilisateur vs agence de la demande). Middleware personnalisé pour chaque endpoint sensible.

Perspectives d'Évolution

Bien que le système soit pleinement fonctionnel et réponde aux besoins identifiés, plusieurs axes d'amélioration et d'extension sont envisageables :

Court Terme (3-6 mois)

Notifications en Temps Réel : Intégration de WebSockets ou Server-Sent Events pour des notifications push instantanées, éliminant le besoin de rafraîchir manuellement la page.

Export Avancé : Génération de rapports PDF formatés incluant les graphiques, avec possibilité de programmation d'exports récurrents (hebdomadaires, mensuels).

Gestion Avancée des Équipes : Interface complète de gestion des SecurityOfficers avec planning de disponibilité, historique des missions, évaluation de performance.

Dashboards Personnalisés : Permettre à chaque utilisateur de configurer son dashboard avec les widgets et KPIs qui l'intéressent.

Recherche Avancée : Moteur de recherche full-text permettant de rechercher dans tous les champs (description, notes, motifs de rejet).

Moyen Terme (6-12 mois)

Application Mobile : Développement d'une application React Native pour consultation des demandes et notifications sur smartphone, particulièrement utile pour les équipes de sécurité en déplacement.

Signature Électronique : Intégration d'une solution de signature électronique pour valider formellement les réceptions, avec valeur légale.

Versionnage des Demandes : Permettre la modification de demandes en attente avec historique des versions, traçabilité des changements.

Workflow Configurable : Permettre à l'administrateur de configurer les étapes du workflow selon les besoins (ajout d'étapes de validation supplémentaires, approbations parallèles).

Intégration API Swift : Connexion au système Swift de la banque pour réconciliation automatique avec les mouvements comptables.

Module de Prévisions : Utilisation de l'historique pour prédire les besoins futurs en liquidités de chaque agence (Machine Learning).

Long Terme (12+ mois)

Intelligence Artificielle :

- Détection automatique d'anomalies dans les demandes (montants inhabituels, patterns suspects)
- Prédiction des délais de traitement selon les périodes
- Optimisation de l'allocation des équipes par algorithmes génétiques

Blockchain pour Traçabilité : Utilisation d'une blockchain privée pour garantir l'immuabilité des logs d'actions critiques, renforçant la confiance et la conformité.

Architecture Microservices : Découplage du monolithe en microservices indépendants (service demandes, service authentification, service analytics) pour améliorer la scalabilité et la résilience.

Scaling Horizontal : Mise en place de load balancing, caching distribué (Redis), et réplication de base de données pour supporter un volume élevé de transactions.

Multi-Tenant : Adaptation du système pour servir plusieurs institutions bancaires avec isolation des données, permettant une commercialisation du produit.

Apport Personnel et Professionnel

Ce projet de fin d'études a représenté bien plus qu'un simple exercice académique. Il m'a permis de :

- **Appliquer concrètement les connaissances théoriques** acquises tout au long de mon cursus en informatique, dans un contexte réel avec des contraintes métier authentiques.
- **Développer une vision globale** d'un projet informatique, de l'analyse des besoins à la livraison finale, en passant par la conception, le développement, les tests et la documentation.
- **Renforcer mon autonomie** dans la résolution de problèmes techniques complexes, en recherchant et en assimilant de nouvelles technologies rapidement.
- **Améliorer mes compétences en communication**, en présentant régulièrement les avancées au Product Owner et en ajustant le développement selon le feedback.
- **Comprendre les enjeux du secteur bancaire**, un domaine exigeant en termes de sécurité, de conformité et de fiabilité.
- **Me préparer au monde professionnel** en travaillant selon des méthodologies modernes (Agile/Scrum) et avec des outils utilisés dans l'industrie.

Mot de Fin

Le développement de ce Système de Gestion des Fonds Bancaires pour Amen Bank a été une expérience enrichissante et formatrice. Le système livré est opérationnel, sécurisé et évolutif, répondant pleinement aux objectifs fixés en début de projet.

Au-delà du produit technique, ce projet m'a permis de développer une rigueur professionnelle et une capacité à mener un projet complexe de bout en bout. Les compétences acquises, tant techniques que méthodologiques, constituent une base solide pour ma future carrière d'ingénieur en informatique.

Je suis convaincu que ce système apportera une réelle valeur ajoutée à Amen Bank en modernisant ses processus de gestion des fonds, en réduisant les risques opérationnels et en améliorant l'efficacité globale. Les perspectives d'évolution identifiées offrent de nombreuses opportunités d'enrichissement futur du système.

Ce projet marque la fin de mon parcours académique et le début de mon aventure professionnelle, avec la satisfaction d'avoir contribué à la transformation digitale d'une institution bancaire majeure.

Bibliographie

[1] Vercel Inc. *Next.js Documentation - The React Framework for Production*. https://nextjs.org/docs, 2024.

[2] Prisma Data Inc. *Prisma - Next-generation ORM for Node.js and TypeScript*. https://www.prisma.io/docs, 2024.

[3] NextAuth.js Contributors. *NextAuth.js Documentation - Authentication for Next.js*. https://next-auth.js.org, 2024.

[4] Meta Platforms Inc. *React - A JavaScript Library for Building User Interfaces*. https://react.dev, 2024.

[5] Microsoft Corporation. *TypeScript Documentation - JavaScript with Syntax for Types*. https://www.typescriptlang.org/docs, 2024.

[6] The PostgreSQL Global Development Group. *PostgreSQL Documentation - The World's Most Advanced Open Source Relational Database*. https://www.postgresql.org/docs, 2024.

[7] Tailwind Labs Inc. *Tailwind CSS Documentation - Rapidly Build Modern Websites*. https://tailwindcss.com/docs, 2024.

[8] Colinhacks. *Zod - TypeScript-first Schema Validation*. https://zod.dev, 2024.

[9] Schwaber, Ken and Sutherland, Jeff. *The Scrum Guide - The Definitive Guide to Scrum : The Rules of the Game*. Scrum.org, November 2020.

[10] Beck, Kent et al. *Manifesto for Agile Software Development*. https://agilemanifesto.org, 2001.

[11] Martin, Robert C. *Clean Code : A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[12] Hunt, Andrew and Thomas, David. *The Pragmatic Programmer : From Journeyman to Master*. Addison-Wesley Professional, 2019. 20th Anniversary Edition.

[13] Gamma, Erich et al. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[14] OpenJS Foundation. *Node.js Documentation*. https://nodejs.org/docs, 2024.

[15] Jones, M., Bradley, J., and Sakimura, N. *JSON Web Token (JWT) - RFC 7519*. Internet Engineering Task Force (IETF), May 2015.

[16] OWASP Foundation. *OWASP Top Ten - Web Application Security Risks*. https://owasp.org/www-project-top-ten, 2021.

[17] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[18] Object Management Group (OMG). *Unified Modeling Language (UML) Specification*. https://www.omg.org/spec/UML, 2017.

[19] Sommerville, Ian. *Software Engineering*, 10th Edition. Pearson, 2015.

[20] Elmasri, Ramez and Navathe, Shamkant B. *Fundamentals of Database Systems*, 7th Edition. Pearson, 2015.

[21] Duckett, Jon. *JavaScript and jQuery : Interactive Front-End Web Development*. Wiley, 2014.

[22] Stoyan, Stefanov. *React Design Patterns and Best Practices*, 2nd Edition. Packt Publishing, 2019.

[23] Newman, Sam. *Building Microservices : Designing Fine-Grained Systems*, 2nd Edition. O'Reilly Media, 2021.

[24] Kim, Gene et al. *The DevOps Handbook : How to Create World-Class Agility, Reliability, and Security*. IT Revolution Press, 2016.

[25] Anderson, Ross. *Security Engineering : A Guide to Building Dependable Distributed Systems*, 3rd Edition. Wiley, 2020.

[26] Chacon, Scott and Straub, Ben. *Pro Git*, 2nd Edition. Apress, 2014. https://git-scm.com/book, Libre accès.

[27] Mozilla Developer Network (MDN). *Web Technology Documentation*. https://developer.mozilla.org, 2024.

[28] Provos, Niels and Mazières, David. *A Future-Adaptable Password Scheme*. Proceedings of USENIX Annual Technical Conference, 1999.

[29] Gray, Jim and Reuter, Andreas. *Transaction Processing : Concepts and Techniques*. Morgan Kaufmann, 1992.

[30] Basel Committee on Banking Supervision. *Principles for the Sound Management of Operational Risk*. Bank for International Settlements, June 2011.

[31] European Parliament and Council. *Regulation (EU) 2016/679 - General Data Protection Regulation (GDPR)*. Official Journal of the European Union, April 2016.

[32] Stack Overflow. *Developer Community and Q&A Platform*. https://stackoverflow.com, 2024.

[33] GitHub Inc. *GitHub Documentation - Code Hosting and Collaboration Platform*. https://docs.github.com, 2024.

[34] Vercel Inc. *Vercel Documentation - Platform for Frontend Frameworks*. https://vercel.com/docs, 2024.

[35] Recharts Contributors. *Recharts - Redefined Chart Library Built with React and D3*. https://recharts.org, 2024.

[36] Poimandres Contributors. *Zustand - A Small, Fast and Scalable State-Management Solution*. https://github.com/pmndrs/zustand, 2024.

[37] WorkOS Inc. *Radix UI - Unstyled, Accessible Components for Building High-Quality Design Systems*. https://www.radix-ui.com, 2024.

[38] shadcn. *shadcn/ui - Beautifully Designed Components Built with Radix UI and Tailwind CSS*. https://ui.shadcn.com, 2024.

[39] Medium. *Articles et Retours d'Expérience Techniques*. https://medium.com/tag/programming, 2024.

[40] CSS-Tricks. *Articles on Web Design and Development*. https://css-tricks.com, 2024.