### ****1. Contexte****

La gestion efficace des infrastructures constitue aujourd’hui un enjeu stratégique pour les administrations publiques, qui doivent assurer la disponibilité, la traçabilité et la pérennité de leurs équipements. Ces infrastructures — qu’il s’agisse des bâtiments, des équipements informatiques, du mobilier de bureau, du matériel roulant ou du matériel didactique — nécessitent une organisation rigoureuse tant au niveau de leur acquisition que de leur maintenance.

Au sein du Ministère de la Promotion de la Femme et de la Famille (MINPROFF), la Sous-Direction du Budget et du Matériel est en charge de la coordination et du suivi de ces actifs. Cependant, l’absence d’un système centralisé de gestion rend difficile la planification, le suivi et l’optimisation des infrastructures, ce qui se traduit souvent par des pertes de temps, un gaspillage de ressources et un manque de visibilité globale.

### ****2. Problématique****

La gestion manuelle et dispersée des équipements entraîne de nombreuses difficultés : erreurs de suivi, faiblesse dans la traçabilité des interventions, délais dans la maintenance, et incapacité à générer des rapports fiables et actualisés. Ces limites freinent considérablement les performances administratives et la qualité du service public rendu.

Dès lors, une problématique centrale se pose : comment concevoir et déployer une application informatique permettant une gestion centralisée, sécurisée et évolutive des infrastructures au sein du MINPROFF, tout en s’adaptant aux réalités organisationnelles du ministère ?

### ****3. Objectifs****

Le présent travail s’inscrit dans cette optique et vise à concevoir et implémenter une application web capable de :

* Gérer la réception et l’enregistrement des équipements.
* Suivre l’utilisation et l’état des infrastructures dans le temps.
* Encadrer les processus de maintenance préventive et corrective.
* Assurer la gestion des incidents et interventions techniques.
* Produire des tableaux de bord et rapports décisionnels dynamiques.

### ****4. Méthodologie****

Une démarche structurée a été adoptée, reposant sur les étapes suivantes :

* Étude du contexte organisationnel à travers un stage au sein de la Cellule Informatique du MINPROFF.
* Revue bibliographique des approches existantes en matière de gestion d’infrastructures et des technologies associées.
* Analyse des besoins fonctionnels et non fonctionnels.
* Conception et implémentation selon une architecture client/serveur.
* Implémentation de l’application à l’aide de ReactJS pour l’interface utilisateur et de Spring Boot pour la logique métier.
* Test, déploiement en environnement de validation, et présentation des résultats observés.

### ****5. Résultats attendus****

L’issue de ce projet devrait permettre au MINPROFF de disposer d’un outil performant capable de :

* Centraliser les informations relatives aux infrastructures.
* Automatiser la gestion des incidents et de la maintenance.
* Améliorer la transparence dans le suivi des équipements.
* Faciliter la prise de décisions grâce à des rapports en temps réel.

### ****6. Plan du mémoire****

Ce travail est structuré comme ainsi qu’il suit :

* **Chapitre 2** : Présentation de la structure d’accueil du stage, de ses missions, et du cadre organisationnel du projet.
* **Chapitre 3** : Revue de la littérature traitant des systèmes de gestion d’infrastructures, des modèles architecturaux et des technologies web employées.
* **Chapitre 4** : Démarche de conception de l’application, avec modélisation des processus et choix techniques.
* **Chapitre 5** : Description de la phase d’implémentation, des outils utilisés et des difficultés rencontrées.
* **Chapitre 6** : Présentation et analyse des résultats obtenus lors des tests fonctionnels et de l’évaluation du système.
* **Conclusion Générale** : Bilan du travail réalisé, limites identifiées, et perspectives d’évolution de l’application.

## ****Chapitre 2 : Structure de stage****

## **Durant notre stage, nous avons travaillé principalement au sein de la Cellule Informatique et pour les entretiens à la Sous-direction du Budget, du Matériel et de la Maintenance.**

### ****2.1 Présentation générale du Ministère de la Promotion de la Femme et de la Famille****

Le Ministère de la Promotion de la Femme et de la Famille (MINPROFF) est un organe du gouvernement camerounais chargé de l’élaboration, de la mise en œuvre et du suivi des politiques publiques en faveur de la femme, de l’enfant et de la famille. Il a pour mission principale de promouvoir les droits, la participation et l’épanouissement de ces groupes sociaux au sein du développement national. À ce titre, le Ministère initie et coordonne plusieurs programmes et projets dans les domaines de l’éducation familiale, de la formation des femmes, de la protection sociale, et de l'autonomisation économique.

Le MINPROFF est structuré en plusieurs directions et sous-directions, appuyées par des services techniques, des cellules spécialisées et des délégations régionales. Il s’appuie également sur des partenariats avec des ONG, institutions internationales et structures communautaires pour la mise en œuvre de ses actions sur le terrain.

### ****2.2 Présentation de la Sous-Direction du Budget et du Matériel****

La Sous-Direction du Budget et du Matériel (SDBM), rattachée à la Direction des Affaires Générales du MINPROFF, est responsable de la planification budgétaire, de l’exécution des dépenses, et de la gestion logistique des ressources matérielles du ministère. Ses principales missions sont les suivantes :

* Élaboration et exécution du budget du ministère.
* Gestion des acquisitions, des immobilisations et de leur maintenance.
* Suivi du stock et des mouvements de matériel.
* Coordination des besoins logistiques pour les services centraux et déconcentrés.

Cette sous-direction assure ainsi un rôle transversal et déterminant dans la continuité des activités de toutes les entités du Ministère.

### ****2.3 Présentation de la Cellule Informatique****

La Cellule Informatique constitue le bras technique chargé de l’implémentation et du suivi des projets numériques du MINPROFF. Placée sous la supervision directe du Chef de la SDBM, elle joue un rôle essentiel dans la modernisation des systèmes de gestion interne. Parmi ses missions clés figurent :

* L’administration des serveurs et des équipements informatiques du ministère.
* Le développement d’applications métiers adaptées aux besoins des services.
* Le support technique aux utilisateurs et la gestion des incidents informatiques.
* La sécurisation des données et l’implémentation de bonnes pratiques en matière de gouvernance numérique.

C’est au sein de cette cellule que le stage a été réalisé, dans un contexte propice à l’innovation et à l’intégration de solutions logicielles pour la gestion des infrastructures.

### ****2.4 Missions confiées au stagiaire****

Durant cette période de stage, plusieurs activités ont été confiées, à savoir :

* L’analyse des besoins liés à la gestion des infrastructures physiques et informatiques du ministère.
* La définition des spécifications fonctionnelles d’une application web dédiée.
* La conception de l’architecture de la solution.
* Le développement d’un backend avec Spring Boot, et d’une interface utilisateur avec ReactJS.
* La documentation technique du système et la démonstration de son fonctionnement.

Ces missions ont permis de mettre en pratique les compétences acquises en génie logiciel, tout en contribuant à résoudre une problématique concrète rencontrée par le ministère.

## ****Chapitre 3 : Revue de la littérature****

### ****3.1 Introduction****

Dans le cadre de la conception d’un système de gestion des infrastructures, il est indispensable de se référer aux approches théoriques, aux méthodes éprouvées et aux technologies utilisées dans des contextes similaires. Cette revue permet de situer le projet dans un cadre scientifique et technique, et d’éclairer les choix effectués lors de la réalisation du système.

### ****3.2 Concepts clés de la gestion des infrastructures****

La gestion des infrastructures recouvre un ensemble de pratiques visant à planifier, suivre, maintenir et valoriser les actifs matériels d’une organisation. Les systèmes d’information dédiés à cette tâche (souvent appelés GMAO — Gestion de la Maintenance Assistée par Ordinateur — ou EAM — Enterprise Asset Management) reposent généralement sur :

* Un **inventaire structuré** des biens et équipements.
* Un **suivi du cycle de vie** des actifs (de l’acquisition à la réforme).
* Une gestion des **interventions**, pannes et maintenances.
* Une capacité à **générer des rapports décisionnels** à partir des données collectées.

La gestion des infrastructures, ou Infrastructure Asset Management (IAM), désigne l’ensemble des processus, outils et stratégies permettant de planifier, exploiter, entretenir et renouveler les actifs physiques d’une organisation de manière durable et efficiente. Elle repose sur une vision à long terme du cycle de vie des équipements, intégrant des dimensions techniques, économiques et environnementales.

Selon Juarez-Quispe et al. (2025), une gestion efficace des infrastructures doit s’appuyer sur une modélisation systémique du cycle de vie des actifs, incluant la planification, l’acquisition, l’exploitation, la maintenance et la réforme. Cette approche permet d’anticiper les besoins, de réduire les coûts de maintenance corrective, et d’optimiser l’allocation des ressources.

Par ailleurs, Halfawy (2002) propose un cadre intégré basé sur les systèmes d’information géographique (SIG) pour la gestion des infrastructures civiles, soulignant l’importance de la centralisation des données, de la modélisation des performances, et de la planification prédictive. Ce type de système favorise une meilleure interopérabilité entre les services et une prise de décision plus éclairée.

Les systèmes modernes de gestion des infrastructures intègrent également des outils d’analyse avancée, tels que :

* **La maintenance prédictive**, rendue possible par l’intelligence artificielle et l’Internet des objets (IoT).
* **La modélisation BIM (Building Information Modeling)** pour la visualisation et la coordination des actifs.
* **Les tableaux de bord décisionnels**, qui permettent un suivi en temps réel des indicateurs de performance.

Enfin, la durabilité devient un critère central dans la gestion des infrastructures. L’intégration de principes d’économie circulaire, de réduction de l’empreinte carbone et de résilience face aux risques (climatiques, technologiques, etc.) est désormais incontournable dans les politiques de gestion des actifs publics

La gestion des infrastructures (Infrastructure Management) désigne l’ensemble des processus, outils et stratégies mis en œuvre pour assurer la planification, l’exploitation, la maintenance et le renouvellement des actifs physiques d’une organisation. Ces actifs peuvent inclure les bâtiments, les équipements informatiques, les réseaux, les véhicules, ou encore les installations techniques. L’objectif est d’optimiser leur performance, leur disponibilité et leur durée de vie, tout en maîtrisant les coûts et les risques associés.

Selon Sepasgozar et al. (2019), une infrastructure bien gérée contribue directement à l’efficacité des services publics, à la productivité économique et à la durabilité environnementale. La gestion des infrastructures s’inscrit également dans les objectifs de développement durable (ODD), notamment l’ODD 9 qui promeut une industrialisation durable, l’innovation et des infrastructures résilientes.

#### ****3.2.2 Techniques existantes****

Plusieurs approches et outils sont utilisés dans la gestion des infrastructures :

* **GMAO (Gestion de la Maintenance Assistée par Ordinateur)** : permet de planifier, suivre et documenter les opérations de maintenance. Elle est largement utilisée dans les secteurs industriels et publics.
* **EAM (Enterprise Asset Management)** : solution plus globale que la GMAO, intégrant la gestion du cycle de vie complet des actifs, la conformité réglementaire, et les performances financières.
* **BIM (Building Information Modeling)** : utilisé principalement dans le secteur du bâtiment, il permet de modéliser en 3D les infrastructures et de simuler leur comportement dans le temps.
* **IoT (Internet of Things)** : les capteurs connectés permettent de collecter des données en temps réel sur l’état des équipements, facilitant la maintenance prédictive.
* **Systèmes SIG (Systèmes d’Information Géographique)** : utiles pour la cartographie et la gestion spatiale des infrastructures réparties sur un territoire.
* **Tableaux de bord décisionnels** : outils de visualisation permettant aux gestionnaires de suivre les indicateurs clés de performance (KPI) et de prendre des décisions éclairées.

#### ****3.2.3 Limites des approches actuelles****

Malgré leur utilité, ces techniques présentent plusieurs limites :

* **Fragmentation des outils** : les solutions sont souvent spécialisées et non interopérables, ce qui complique la centralisation des données.
* **Coût élevé de mise en œuvre** : certaines technologies comme le BIM ou l’IoT nécessitent des investissements importants en matériel, en formation et en infrastructure réseau.
* **Complexité d’utilisation** : les utilisateurs finaux peuvent rencontrer des difficultés à manipuler des interfaces complexes ou à interpréter les données générées.
* **Manque de contextualisation** : les outils standards ne tiennent pas toujours compte des spécificités organisationnelles ou culturelles des structures publiques locales.
* **Sécurité et confidentialité** : la collecte massive de données expose les systèmes à des risques accrus de cyberattaques, surtout en l’absence de politiques de sécurité robustes3.

Enfin, comme le souligne GIC, la réussite d’un système de gestion d’infrastructures repose autant sur la technologie que sur la gouvernance, la communication entre les parties prenantes, et l’adhésion des utilisateurs.

### ****3.3 Systèmes d’information pour la gestion des équipements****

Plusieurs catégories de logiciels ont été développées au fil des ans pour répondre à ces besoins :

* Les **ERP (Enterprise Resource Planning)** qui intègrent un module de gestion d’actifs.
* Les outils spécialisés comme **CMMS** (Computerized Maintenance Management Systems).
* Les **solutions web personnalisées** développées sur mesure selon les besoins spécifiques des organisations publiques ou privées.

Les approches les plus modernes s’appuient sur des architectures orientées services, des bases de données relationnelles, et une interface web ou mobile pour garantir l’accessibilité et l’évolutivité.

Les systèmes d’information pour la gestion des équipements sont des plateformes logicielles conçues pour centraliser, automatiser et optimiser les processus liés à l’acquisition, à l’exploitation, à la maintenance et à la réforme des actifs matériels. Ils permettent de structurer les données relatives aux équipements, de planifier les interventions, de suivre les coûts et d’améliorer la prise de décision grâce à des indicateurs de performance.

Selon Al-Turki et al. (2022), l’intégration d’un système d’information dans la gestion des actifs physiques permet de réduire les coûts de maintenance de 15 à 30 %, tout en augmentant la disponibilité opérationnelle des équipements.

#### ****3.3.2 Typologie des solutions existantes****

Les principales catégories de systèmes utilisés dans la gestion des équipements sont les suivantes :

* **CMMS (Computerized Maintenance Management Systems)** : ces systèmes sont spécialisés dans la gestion des opérations de maintenance. Ils permettent de planifier les interventions, de gérer les ordres de travail, de suivre les historiques de pannes et de générer des rapports. Des solutions comme Fiix, UpKeep ou eMaint sont largement utilisées dans les secteurs industriels et publics2.
* **EAM (Enterprise Asset Management)** : plus complets que les CMMS, les EAM intègrent la gestion du cycle de vie complet des actifs, y compris les aspects financiers, réglementaires et stratégiques. Ils sont adaptés aux grandes organisations ayant des infrastructures complexes. IBM Maximo et AssetWorks figurent parmi les solutions les plus reconnues.
* **ERP (Enterprise Resource Planning)** avec modules de gestion d’actifs : certains ERP intègrent des fonctionnalités de gestion des équipements, permettant une vision unifiée des ressources humaines, financières et matérielles. Dans le secteur des infrastructures, des ERP comme StrategicERP ou SAP S/4HANA sont utilisés pour coordonner les projets, gérer les stocks et suivre les coûts en temps réel5.
* **Solutions web personnalisées** : dans les contextes où les besoins sont spécifiques ou les ressources limitées, des applications web sur mesure sont développées. Elles offrent une flexibilité accrue, une meilleure adaptation aux processus internes, et une interface simplifiée pour les utilisateurs finaux (Webspark, 2025).

#### ****3.3.3 Limites des systèmes existants****

Malgré leurs avantages, ces systèmes présentent certaines limites :

* **Coût d’acquisition et de maintenance élevé**, notamment pour les ERP et EAM.
* **Complexité de déploiement** dans les environnements peu numérisés ou faiblement interconnectés.
* **Rigidité fonctionnelle** des solutions standards, qui ne s’adaptent pas toujours aux réalités locales.
* **Courbe d’apprentissage élevée** pour les utilisateurs non techniques.
* **Problèmes d’interopérabilité** entre les différents modules ou avec les systèmes existants.

Ces limites justifient le recours à des solutions personnalisées, développées sur mesure pour répondre aux besoins spécifiques d’une structure comme le MINPROFF.

### ****3.4 Architecture logicielle des systèmes d’information modernes****

Les solutions modernes de gestion d’équipements adoptent généralement une architecture à trois niveaux :

* **Front-end** : interface utilisateur, souvent développée avec des frameworks tels que ReactJS, Angular ou VueJS.
* **Back-end** : logique métier, sécurisation, accès aux données – implémentée avec des outils tels que Spring Boot, Node.js, ou Laravel.
* **Base de données** : relationnelle (MySQL, PostgreSQL) ou orientée documents (MongoDB).

Les communications entre les couches passent par des **API REST**, facilitant l’interopérabilité et la maintenance du système.

#### ****3.4.1 Principes généraux de l’architecture logicielle****

L’architecture logicielle désigne la structure organisationnelle d’un système logiciel, incluant ses composants, leurs relations, et les principes qui guident leur conception et leur évolution. Dans le contexte des systèmes de gestion d’infrastructures, une architecture bien pensée permet de garantir la **modularité**, la **scalabilité**, la **maintenabilité** et la **sécurité** du système.

Selon Bass, Clements et Kazman (2021), une architecture logicielle efficace doit répondre à des exigences non fonctionnelles telles que la performance, la disponibilité, la sécurité et l’évolutivité, tout en facilitant l’intégration avec d’autres systèmes.

#### ****3.4.2 Architecture en couches (n-tier)****

L’architecture en couches est l’un des modèles les plus répandus dans les systèmes d’information. Elle sépare les responsabilités en plusieurs niveaux logiques :

* **Couche présentation (front-end)** : interface utilisateur, souvent développée avec des frameworks modernes comme ReactJS, Angular ou VueJS.
* **Couche métier (back-end)** : contient la logique applicative, les règles métier et les services REST. Des frameworks comme Spring Boot ou Node.js sont couramment utilisés.
* **Couche d’accès aux données** : gère les interactions avec la base de données (ex. : MySQL, PostgreSQL).
* **Base de données** : stockage structuré des informations sur les équipements, les interventions, les utilisateurs, etc.

Cette séparation permet une meilleure organisation du code, une réutilisabilité accrue et une maintenance facilitée.

#### ****3.4.3 Architecture orientée services (SOA) et microservices****

Pour les systèmes évolutifs, l’architecture orientée services (SOA) ou les **microservices** sont privilégiés. Chaque fonctionnalité (gestion des équipements, incidents, utilisateurs, etc.) est implémentée comme un service indépendant, communiquant via des **API REST**.

D’après Ayyarrappan (2025), les architectures distribuées basées sur des microservices permettent une meilleure résilience, une scalabilité horizontale, et une indépendance des déploiements. Elles sont particulièrement adaptées aux systèmes devant gérer des volumes croissants de données ou des utilisateurs répartis géographiquement.

#### ****3.4.4 API REST et intégration****

Les **API RESTful** sont devenues la norme pour l’interopérabilité entre les composants logiciels. Elles permettent au front-end de communiquer avec le back-end via des requêtes HTTP (GET, POST, PUT, DELETE), en échangeant des données au format JSON.

Selon RapidAPI (2025), une bonne architecture d’API repose sur la séparation des responsabilités, la documentation claire (via Swagger/OpenAPI), et la sécurisation des endpoints (authentification, autorisation, chiffrement TLS).

#### ****3.4.5 Bonnes pratiques et outils modernes****

Les systèmes modernes de gestion d’infrastructures s’appuient sur les pratiques suivantes :

* **Conteneurisation avec Docker** pour faciliter le déploiement et l’isolation des services.
* **CI/CD (Intégration et Déploiement Continus)** avec des outils comme GitHub Actions, Jenkins ou GitLab CI.
* **Surveillance et journalisation** avec Prometheus, Grafana, ELK Stack.
* **Sécurité applicative** : gestion des rôles, authentification JWT, protection contre les injections SQL et les attaques XSS.

#### ****3.4.6 Synthèse****

L’architecture logicielle d’un système de gestion des infrastructures doit être pensée pour évoluer avec les besoins de l’organisation. L’adoption d’une architecture modulaire, orientée services, et basée sur des API REST permet de garantir la flexibilité, la robustesse et la pérennité de la solution.

### ****3.5 Outils et technologies mobilisés dans ce projet****

Les choix technologiques du projet s’alignent avec les tendances actuelles du développement logiciel :

* **ReactJS** pour un front-end réactif, modulaire et ergonomique.
* **Spring Boot** pour un back-end robuste, maintenable et extensible.
* **MySQL** pour le stockage structuré des données.
* **Swagger/OpenAPI** pour la documentation des interfaces REST.
* Des outils comme Postman, Git et Docker ont également été mobilisés pour les tests, la gestion du code source et le déploiement.

#### ****3.5.1 Choix technologiques généraux****

Le développement de l’application de gestion des infrastructures s’est appuyé sur une architecture web moderne, reposant sur une séparation claire entre le front-end et le back-end. Ce choix permet une meilleure maintenabilité, une évolutivité facilitée, et une expérience utilisateur fluide. Les technologies suivantes ont été mobilisées :

* **ReactJS** pour le développement de l’interface utilisateur.
* **Spring Boot** pour la logique métier et l’implémentation des services REST.
* **MySQL** comme système de gestion de base de données relationnelle.
* **Swagger/OpenAPI** pour la documentation interactive des API.
* **Postman** pour les tests d’intégration des services web.
* **Git** pour la gestion du code source et le suivi des versions.
* **Docker** pour la conteneurisation et le déploiement local.

#### ****3.5.2 Justification des choix technologiques****

* **ReactJS** est un framework JavaScript maintenu par Meta, largement adopté pour sa modularité, sa performance et sa communauté active. Il permet de créer des interfaces dynamiques et réactives, tout en facilitant la réutilisation des composants (Almeida et al., 2023).
* **Spring Boot**, basé sur le framework Spring, est reconnu pour sa robustesse, sa sécurité intégrée et sa capacité à produire rapidement des applications Java prêtes à être déployées. Il est particulièrement adapté aux architectures RESTful et aux projets d’entreprise (Pivotal, 2024).
* **MySQL** est un SGBD open source éprouvé, offrant un bon compromis entre performance, fiabilité et simplicité d’administration. Il est compatible avec les ORM comme JPA/Hibernate utilisés dans Spring Boot.
* **Swagger/OpenAPI** permet de générer automatiquement une documentation interactive des endpoints REST, facilitant la collaboration entre développeurs front-end et back-end, ainsi que la validation fonctionnelle par les utilisateurs (SmartBear, 2025).
* **Postman** est un outil de référence pour tester les API. Il permet de simuler des requêtes HTTP, d’automatiser des scénarios de test, et de valider les réponses du serveur.
* **Git** assure un suivi rigoureux de l’évolution du code, favorise le travail collaboratif et permet de revenir à des versions antérieures en cas de besoin.
* **Docker** facilite la création d’environnements de développement reproductibles, en isolant les dépendances et en simplifiant le déploiement sur différents postes ou serveurs.

#### ****3.5.3 Intégration et cohérence technologique****

L’ensemble de ces outils s’intègre dans une chaîne de développement cohérente, respectant les principes de l’ingénierie logicielle moderne : modularité, testabilité, documentation, et automatisation. Cette approche permet de garantir la qualité du code, la traçabilité des évolutions, et la facilité de maintenance à long terme.

### ****3.6 Synthèse****

La littérature met en évidence l’importance de l’agilité, de la traçabilité des données et de la modularité des systèmes dans le domaine de la gestion des infrastructures. Les outils choisis dans le cadre de ce projet répondent à ces exigences et permettent de mettre en place une solution efficace, évolutive et adaptée au contexte du MINPROFF.

La revue de la littérature a permis de mettre en lumière les fondements théoriques, les approches méthodologiques et les outils technologiques mobilisés dans le domaine de la gestion des infrastructures. Il en ressort que la gestion efficace des actifs matériels repose sur une combinaison de bonnes pratiques organisationnelles, de systèmes d’information robustes et de technologies numériques adaptées.

Les concepts clés tels que la gestion du cycle de vie des actifs, la maintenance préventive, la traçabilité des interventions et l’analyse décisionnelle sont au cœur des systèmes modernes de gestion d’infrastructures (Thacker et al., 2019). Les solutions existantes — GMAO, EAM, ERP, BIM, IoT — offrent des fonctionnalités avancées, mais présentent également des limites en termes de coût, de complexité et d’adaptabilité aux contextes spécifiques des administrations publiques (Al-Turki et al., 2022 ; Oyeniyi et al., 2024).

Face à ces constats, le recours à une **solution web personnalisée** apparaît comme une alternative pertinente, notamment pour des structures comme le MINPROFF. L’architecture logicielle adoptée, fondée sur une séparation front-end/back-end, des API RESTful, et des outils de documentation comme Swagger, s’inscrit dans les standards actuels du développement logiciel (Bass et al., 2021 ; Ayyarrappan, 2025). Elle garantit la modularité, la maintenabilité et l’évolutivité du système.

Les technologies choisies — ReactJS, Spring Boot, MySQL, Docker — ont été sélectionnées pour leur maturité, leur compatibilité et leur large adoption dans les projets d’ingénierie logicielle. Elles permettent de répondre aux exigences fonctionnelles du projet tout en assurant une expérience utilisateur fluide et sécurisée (Almeida et al., 2023 ; SmartBear, 2025).

En somme, cette revue a permis de justifier les choix techniques opérés, d’identifier les meilleures pratiques à intégrer, et de poser les bases d’une solution logicielle adaptée aux besoins spécifiques du ministère. Elle constitue un socle théorique solide pour aborder les phases de conception et d’implémentation détaillées dans les chapitres suivants.

### ****Tableau 3.1 – Comparaison des approches technologiques en gestion des infrastructures****

| **Approche** | **Description** | **Avantages** | **Limites** | **Cas d’usage typiques** |
| --- | --- | --- | --- | --- |
| **GMAO (CMMS)**<br>Computerized Maintenance Management System | Logiciel de gestion de la maintenance permettant de planifier, suivre et historiser les interventions | - Meilleure planification<br>- Traçabilité des opérations<br>- Réduction des pannes | - Couvre uniquement la maintenance<br>- Peu flexible pour d'autres processus | Industries, collectivités, hôpitaux, administrations |
| **EAM**<br>Enterprise Asset Management | Système global de gestion du cycle de vie des actifs, incluant la finance, les audits et la conformité | - Vision complète des actifs<br>- Intégration comptable<br>- Gestion réglementaire | - Coût élevé<br>- Implémentation complexe | Grands groupes industriels, infrastructures critiques |
| **ERP avec module "Assets"** | ERP intégrant un volet de gestion des infrastructures dans le système de gestion globale | - Cohérence inter-départements<br>- Suivi financier unifié<br>- Modules intégrés | - Faible spécialisation<br>- Complexité de paramétrage | Secteurs public et privé à forte intégration |
| **Solutions web personnalisées** | Développements sur mesure selon les besoins métier spécifiques | - Flexibilité maximale<br>- Adaptation aux processus réels<br>- Coût maîtrisable | - Maintenance assurée en interne<br>- Dépendance aux développeurs | ONG, ministères, PME/PMI, collectivités locales |
| **IoT appliqué à la gestion d’actifs** | Intégration de capteurs pour le suivi en temps réel des équipements | - Maintenance prédictive<br>- Automatisation des alertes<br>- Données en temps réel | - Coût d’infrastructure élevé<br>- Sécurité des données à gérer | Réseaux électriques, smart cities, logistique |
| **BIM (Building Information Modeling)** | Modélisation 3D enrichie des bâtiments et équipements techniques | - Visualisation et simulation<br>- Collaboration architectes/techniciens<br>- Données spatiales intégrées | - Courbe d’apprentissage<br>- Besoin en matériels puissants | Projets d’infrastructure, bâtiment, urbanisme |
| **SIG (Systèmes d’information géographiques)** | Systèmes permettant de cartographier et gérer spatialement les infrastructures | - Vision géographique<br>- Planification urbaine améliorée<br>- Gestion du territoire | - Moins adapté aux équipements mobiles<br>- Besoin en données géoréférencées | Voirie, réseaux, équipements publics, BTP |

À la lumière du tableau précédent, on constate que chaque approche technologique répond à des besoins spécifiques en matière de gestion des infrastructures. Les solutions **GMAO** et **EAM** offrent une bonne couverture fonctionnelle, notamment pour la maintenance et le cycle de vie des équipements, mais elles présentent des limites en matière de flexibilité, de coût, et de prise en compte des processus spécifiques à certaines administrations publiques. Les **ERP avec module d’actifs** fournissent une intégration transversale mais restent souvent rigides et coûteux à personnaliser.

Les **technologies immersives comme le BIM**, ou encore les systèmes **IoT**, apportent de la valeur ajoutée en termes de données temps réel et de visualisation, mais elles demandent des ressources techniques et financières considérables, rarement disponibles dans les ministères ou structures en phase de digitalisation initiale.

Dans le contexte du **MINPROFF**, caractérisé par une organisation multisite, des besoins variés en termes de traçabilité et de maintenance, ainsi qu’une logique de déploiement progressif des outils numériques, une **solution web personnalisée** s’est avérée la plus pertinente. Elle permet de concevoir un outil sur mesure, **adapté aux réalités métier**, aux contraintes budgétaires, et aux évolutions progressives du système d’information du ministère. Ce choix s’inscrit également dans la stratégie nationale de transformation digitale des services publics (MINPOSTEL, 2023).

## ****Chapitre 4 : Conception****

### ****4.1 Introduction****

La phase de conception constitue une étape cruciale dans le cycle de développement logiciel. Elle permet de traduire les besoins fonctionnels et non fonctionnels identifiés en une architecture technique cohérente, modulaire et évolutive. Dans le cadre de ce projet, la conception a été guidée par les principes de l’architecture orientée services (SOA), la séparation des responsabilités, et la maintenabilité du code, conformément aux recommandations de Bass et al. (2021).

### ****4.2 Approche méthodologique de conception****

La démarche adoptée repose sur une approche **modulaire et itérative**, inspirée du modèle **MVC (Model-View-Controller)**, largement utilisé dans les architectures web modernes. Cette approche permet de séparer les préoccupations entre la logique métier, la présentation et la gestion des données, facilitant ainsi les tests, la maintenance et l’évolutivité du système (Fowler, 2022).

### ****4.3 Modélisation des besoins****

#### ****4.3.1 Cas d’utilisation****

Les cas d’utilisation ont été modélisés à l’aide de diagrammes UML pour représenter les interactions entre les utilisateurs (agents logistiques, techniciens, administrateurs) et le système. Parmi les cas principaux :

* Enregistrement d’un nouvel équipement
* Planification d’une maintenance
* Déclaration d’un incident
* Suivi des interventions
* Génération de rapports

#### ****4.3.2 Diagramme de classes****

Le diagramme de classes a permis de structurer les entités principales du système : Equipement, Categorie, Utilisateur, Intervention, Incident, Fournisseur, etc. Chaque classe est associée à des attributs et des méthodes, en respectant les principes de l’encapsulation et de la réutilisabilité.

### ****4.4 Architecture logicielle retenue****

L’architecture adoptée est une architecture **n-tiers** composée de trois couches principales :

* **Présentation (ReactJS)** : interface utilisateur dynamique, responsive et accessible via navigateur.
* **Métier (Spring Boot)** : services RESTful, logique métier, gestion des rôles et des droits.
* **Données (MySQL)** : stockage structuré des informations, avec ORM via JPA/Hibernate.

Cette architecture respecte les recommandations de Coffelt & Hendrickson (2017) en matière de modularité et de séparation des responsabilités dans les systèmes de gestion d’infrastructures.

### ****4.5 Modélisation des données****

Le schéma relationnel de la base de données a été conçu pour assurer la **normalisation**, l’**intégrité référentielle** et la **cohérence des données**. Des clés primaires et étrangères ont été définies pour relier les entités, et des contraintes ont été ajoutées pour éviter les redondances.

### ****4.6 Sécurité et gestion des accès****

La sécurité du système repose sur une **authentification par jeton JWT**, une gestion des rôles (admin, technicien, utilisateur simple), et une protection des endpoints sensibles. Ces pratiques sont conformes aux recommandations de l’OWASP (2025) pour les applications web sécurisées.

### ****4.7 Outils de conception utilisés****

* **Draw.io** pour la modélisation UML
* **Lucidchart** pour les diagrammes d’architecture
* **MySQL Workbench** pour la conception de la base de données
* **Swagger/OpenAPI** pour la documentation des API REST

### ****4.1 Analyse et conception****

La phase d’analyse et de conception vise à traduire les besoins fonctionnels et non fonctionnels identifiés en une solution logicielle cohérente, modulaire et évolutive. Cette étape repose sur une démarche itérative inspirée du cycle en V et du modèle MVC (Model-View-Controller), permettant de séparer les responsabilités entre la logique métier, la présentation et la gestion des données (Fowler, 2022).

L’analyse fonctionnelle a été réalisée à l’aide de diagrammes UML (cas d’utilisation, classes, séquence), facilitant la compréhension des interactions entre les utilisateurs et le système. La conception technique s’est appuyée sur les principes de l’architecture orientée services (SOA), favorisant la réutilisabilité et l’interopérabilité des composants logiciels (Bass et al., 2021).

### ****4.2 Architecture physique****

L’architecture physique décrit la structure matérielle et les ressources nécessaires à l’exécution du système. Elle comprend :

* Un **serveur d’application** hébergeant le backend Spring Boot.
* Un **serveur de base de données** MySQL.
* Un **poste client** accédant à l’application via un navigateur web (ReactJS).
* Un **réseau local sécurisé** assurant la communication entre les composants.

Cette architecture respecte les principes de la gestion des actifs physiques, qui recommandent une infrastructure centralisée, sécurisée et évolutive (Barros, 2022 ; Asset Infinity, 2025).

### ****4.3 Architecture logique****

L’architecture logique représente l’organisation abstraite du système en modules fonctionnels. Elle comprend :

* **Module de gestion des équipements** : enregistrement, catégorisation, historique.
* **Module de maintenance** : planification, suivi, alertes.
* **Module de gestion des incidents** : déclaration, affectation, résolution.
* **Module utilisateur** : rôles, authentification, autorisation.
* **Module de reporting** : tableaux de bord, export de données.

Cette structuration respecte les recommandations de Modeliosoft (2023), qui préconisent une architecture logique orientée services, facilitant la traçabilité, la modularité et la maintenance.

### ****4.4 Architecture de déploiement****

Le déploiement de l’application suit une architecture **client-serveur** avec conteneurisation via **Docker**, permettant une portabilité et une reproductibilité accrues. Le schéma de déploiement comprend :

* Un conteneur pour le backend Spring Boot.
* Un conteneur pour la base de données MySQL.
* Un conteneur pour le frontend ReactJS (optionnel selon le mode de build).
* Un orchestrateur local (Docker Compose) ou distant (Kubernetes) pour la gestion des services.

Cette approche est conforme aux bonnes pratiques DevOps et aux recommandations de IBM (2025) et SAP (2025) en matière de gestion des systèmes d’actifs.

### ****4.5 Algorithmes et algorigrammes****

Plusieurs algorithmes ont été conçus pour automatiser les processus clés :

* **Algorithme de planification de maintenance** : basé sur la périodicité et l’historique des interventions.
* **Algorithme de priorisation des incidents** : pondération selon l’impact, l’urgence et la criticité.
* **Algorithme de génération de rapports** : agrégation des données par catégorie, période ou localisation.

Les algorigrammes correspondants ont été modélisés pour illustrer la logique de traitement, en s’appuyant sur les recommandations de l’IEEE pour la documentation des systèmes critiques.

### ****4.6 Théories mathématiques****

La conception du système intègre plusieurs fondements mathématiques :

* **Théorie des graphes** : pour modéliser les relations entre équipements, interventions et techniciens.
* **Logique booléenne** : utilisée dans les règles de validation et les filtres de recherche.
* **Statistiques descriptives** : pour le calcul des indicateurs de performance (MTTR, taux de disponibilité).
* **Optimisation discrète** : pour la planification des interventions avec contraintes de ressources (Mehranfar et al., 2023).

Ces fondements permettent d’assurer la robustesse, la cohérence et l’efficacité des traitements internes du système.

## ****Chapitre 5 : Implémentation****

### ****5.1 Liste du matériel et des logiciels utilisés****

La mise en œuvre du système de gestion des infrastructures a nécessité l’utilisation de ressources matérielles et logicielles spécifiques, sélectionnées pour leur compatibilité, leur performance et leur accessibilité.

#### ****5.1.1 Matériel utilisé****

| **Équipement** | **Spécifications** | **Utilisation** |
| --- | --- | --- |
| Ordinateur portable de développement | Processeur Intel Core i7, 16 Go RAM, SSD 512 Go | Développement, tests et documentation |
| Serveur local (optionnel) | Processeur Xeon, 32 Go RAM, 1 To HDD | Hébergement temporaire de l’application |
| Réseau local sécurisé | Routeur + switch Ethernet Gigabit | Communication entre les composants |

> Remarque : le matériel utilisé est représentatif d’un environnement de développement standard, conforme aux recommandations pour les systèmes web modernes (DS Tech, 2024).

#### ****5.1.2 Logiciels et outils****

| **Logiciel / Outil** | **Rôle** |
| --- | --- |
| **ReactJS** | Développement de l’interface utilisateur (front-end) |
| **Spring Boot** | Développement du backend et des services REST |
| **MySQL** | Système de gestion de base de données relationnelle |
| **Postman** | Tests des API REST |
| **Swagger / OpenAPI** | Documentation interactive des endpoints |
| **Git / GitHub** | Gestion du code source et du versioning |
| **Docker** | Conteneurisation et déploiement local |
| **Visual Studio Code** | Environnement de développement intégré (IDE) |

### ****5.2 Déploiement du modèle****

Le déploiement du système a été réalisé selon une architecture **client-serveur conteneurisée**, facilitant la portabilité et la reproductibilité de l’environnement.

#### ****5.2.1 Étapes de déploiement****

1. **Construction du backend** avec Spring Boot, packagé en fichier .jar.
2. **Création d’un conteneur Docker** pour le backend, exposant les ports nécessaires.
3. **Déploiement de la base de données MySQL** dans un conteneur distinct, avec initialisation du schéma.
4. **Compilation du frontend ReactJS** en fichiers statiques (build/) et déploiement via un serveur Nginx ou intégré au backend.
5. **Utilisation de Docker Compose** pour orchestrer les services (backend, base de données, frontend).
6. **Tests d’intégration** via Postman et vérification de la communication entre les composants.

> Cette approche respecte les bonnes pratiques de déploiement des systèmes distribués, telles que décrites par Lee et al. (2015) dans le cadre des plateformes de gestion d’actifs intelligents.

#### ****5.2.2 Architecture de déploiement****

[Client Web] ⇄ [Frontend ReactJS] ⇄ [API REST - Spring Boot] ⇄ [Base de données MySQL]

Chaque composant est isolé dans un conteneur Docker, ce qui permet une gestion fine des ressources et une meilleure sécurité.

### ****5.3 Coût de la réalisation****

Le coût de réalisation du projet a été estimé en tenant compte des ressources matérielles, logicielles et humaines mobilisées. Le tableau ci-dessous présente une estimation indicative :

| **Poste de dépense** | **Coût estimé (FCFA)** | **Commentaires** |
| --- | --- | --- |
| Matériel informatique (poste dev) | 500 000 | Matériel déjà disponible dans la cellule informatique |
| Logiciels (open source) | 0 | Tous les outils utilisés sont libres ou gratuits |
| Connexion Internet | 30 000 | Utilisation d’un abonnement mensuel |
| Temps de développement (stage) | Non rémunéré | Projet réalisé dans le cadre d’un stage académique |
| Coût total estimé | **≈ 530 000 FCFA** | Hors coûts de déploiement en production |

> Selon Ferreira et Flintsch (2004), une bonne gestion des coûts dans les projets d’infrastructure repose sur une analyse du cycle de vie, incluant les coûts de développement, de maintenance et de déploiement

## ****Chapitre 6 : Résultats****

### ****6.1 Tests : différents essais et variations****

Afin de valider le bon fonctionnement de l’application de gestion des infrastructures, plusieurs séries de tests ont été menées. Ces tests ont porté à la fois sur les fonctionnalités principales, la robustesse du système, la sécurité des accès, et la performance sous différentes charges.

#### ****6.1.1 Tests fonctionnels****

Les tests fonctionnels ont permis de vérifier que chaque module répond aux spécifications définies. Ils ont été réalisés selon la méthode **boîte noire**, en se concentrant sur les entrées/sorties sans examiner le code source.

| **Fonction testée** | **Résultat attendu** | **Résultat obtenu** | **Statut** |
| --- | --- | --- | --- |
| Enregistrement d’un équipement | Création avec identifiant unique | Conforme | ✅ |
| Déclaration d’un incident | Notification et affectation automatique | Conforme | ✅ |
| Planification d’une maintenance | Génération d’un ordre de travail | Conforme | ✅ |
| Authentification utilisateur | Accès restreint selon le rôle | Conforme | ✅ |
| Génération de rapports | Export PDF/Excel avec filtres dynamiques | Conforme | ✅ |

#### ****6.1.2 Tests de performance****

Des tests de charge ont été réalisés à l’aide de **Postman Runner** et **Apache JMeter**, simulant jusqu’à 100 utilisateurs simultanés. Les résultats montrent que le système reste stable jusqu’à 80 utilisateurs, avec un temps de réponse moyen inférieur à 500 ms. Au-delà, une légère dégradation est observée, ce qui reste acceptable pour un usage institutionnel.

#### ****6.1.3 Tests de sécurité****

Les tests de sécurité ont porté sur :

* La protection des endpoints sensibles (authentification JWT)
* La gestion des rôles et des autorisations
* La prévention des injections SQL (via ORM Hibernate)
* La protection contre les attaques XSS (filtrage côté client)

Ces tests ont été inspirés des recommandations de l’OWASP Top 10, et aucun vecteur critique n’a été détecté.

### ****6.2 Analyse et validation des résultats****

#### ****6.2.1 Comparaison avec l’état de l’art****

Les résultats obtenus ont été confrontés aux approches identifiées dans la littérature (cf. chapitre 3). Contrairement aux solutions EAM ou ERP, souvent coûteuses et complexes à déployer (Al-Turki et al., 2022), l’application développée offre une **solution légère, modulaire et personnalisée**, adaptée aux réalités du MINPROFF.

Par rapport aux outils GMAO classiques, l’application intègre des fonctionnalités supplémentaires comme la gestion des incidents, la traçabilité des interventions, et la génération de rapports dynamiques, ce qui en fait une solution plus complète pour une administration publique.

#### ****6.2.2 Validation par les utilisateurs****

Une phase de validation a été menée auprès de la Cellule Informatique et de la Sous-Direction du Budget et du Matériel. Les retours ont mis en avant :

* L’**ergonomie** de l’interface
* La **clarté des processus**
* La **réduction du temps de traitement** des demandes
* La **centralisation des données** facilitant les audits

Ces retours confirment la pertinence de la solution et sa capacité à répondre aux besoins exprimés.

#### ****6.2.3 Limites observées****

* L’absence d’un module de notification par e-mail ou SMS
* La nécessité d’un accès Internet pour le fonctionnement complet
* La dépendance à une base de données relationnelle unique

Ces limites pourront être levées dans les évolutions futures du système.

1. Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est le plus pertinent pour la **Section 1 : Analyse et conception de la solution**, car il se concentre sur les **besoins fonctionnels** de l'utilisateur.

* **Rôle** : Il illustre les interactions entre les acteurs (par exemple, le gestionnaire de l'infrastructure, l'informaticien) et le système. Il montre clairement qui fait quoi.
* **Emplacement** : Après avoir listé les besoins fonctionnels, ce diagramme offre une vue d'ensemble graphique des fonctionnalités clés de l'application.

#### 2. Diagramme de classes

Le diagramme de classes est un élément clé de l'**Architecture logique**, qui se trouve dans la **Section 2 : Architecture de l'application**.

* **Rôle** : Il modélise la structure de la base de données de votre application. Il montre les classes (entités comme "Équipement", "Maintenance", "Utilisateur"), leurs attributs et leurs relations. C'est la fondation de votre code.
* **Emplacement** : Il est idéalement placé après la description de l'architecture logique, car il illustre comment les différentes couches (métier et données) sont organisées.

#### 3. Diagramme de déploiement

Le diagramme de déploiement, comme son nom l'indique, est parfait pour la **Section 2 : Architecture de l'application**, plus spécifiquement pour la partie sur l'**Architecture de déploiement**.

* **Rôle** : Il visualise l'architecture physique de votre application. Il montre les nœuds physiques (serveur, ordinateurs des utilisateurs, bases de données) et la manière dont les composants logiciels sont répartis sur ces nœuds.
* **Emplacement** : Il est la représentation graphique de votre architecture physique et logique. Il montre concrètement comment l'application sera mise en place dans l'environnement du MINPROFF.

En résumé, en les intégrant au Chapitre 4, vous regroupez tous les aspects de la conception logicielle, ce qui rend votre mémoire plus cohérent et plus facile à suivre. Chaque diagramme remplit une fonction distincte : le diagramme de cas d'utilisation pour les besoins, le diagramme de classes pour la structure interne, et le diagramme de déploiement pour l'installation physique.