Ecole Nationale Supérieure des Sciences Appliquées Agadir - ENSAA

Université Ibn Zohr - UIZ

Rapport du projet de fin du module JEE

Plateforme de Supervision de la Consommation d'Eau

Réalisé par :

Abderrahman BOUANANI
Abou Kekeli Efrayim

Encadré par :

Pr. BENIDER

Filière : DLA2 - 2ème Année Développement Logiciel et Applicatif

Année Universitaire : 2025–2026

Déclaration

Nous soussignés, **Abderrahman Bouanani** et **Abou Kekeli Efrayim**, de l'ENSA Agadir, déclarons que le présent rapport, y compris l'ensemble des textes, figures, tableaux, extraits de code et illustrations, constitue le fruit de notre travail personnel. Les sources externes utilisées ont fait l'objet de citations et références explicites. Nous reconnaissons que tout manquement à cet engagement relèverait du plagiat, considéré comme une infraction académique passible de sanctions.

Nous autorisons la mise à disposition d'un exemplaire de ce rapport à des fins pédagogiques, au bénéfice des futurs étudiants et chercheurs, et nous acceptons qu'il soit consulté dans l'intérêt général de l'Enseignement Supérieur et de la Recherche.

Abderrahman Bouanani
Abou Kekeli Efrayim
14 novembre 2025

Remerciements

Nous tenons à exprimer notre profonde gratitude à notre encadrant, le Professeur BENIDER, pour son accompagnement précieux, sa disponibilité et ses conseils avisés tout au long de la réalisation de ce projet. Son expertise et son soutien ont été des atouts majeurs pour le bon déroulement de notre travail.

Nos remerciements s'adressent également à l'ensemble du corps professoral de l'ENSA Agadir pour la qualité de la formation dispensée.

Enfin, nous n'oublions pas nos familles et nos amis pour leur soutien inconditionnel et leurs encouragements constants.

Abstract

Water scarcity and inefficient resource management pose significant challenges for sustainable urban development. This project presents Smart Water Monitoring, a comprehensive web platform for intelligent water consumption management using Internet of Things (IoT) technology and Java Enterprise Edition (JEE). The system addresses the critical need for real-time monitoring, automated data analysis, and proactive alert management to optimize water resource utilization in residential environments. The platform implements a multi-tier JEE architecture following the Model-View-Controller pattern, utilizing Jakarta Servlet API for request handling, Hibernate ORM for data persistence, and MySQL 8 for storage. Core functionalities include real-time IoT sensor data collection via REST APIs, automated daily consumption aggregation through scheduled tasks, intelligent alert generation based on configurable thresholds and time-based patterns, comprehensive statistical analysis, and role-based access control for administrators and citizens. Security features incorporate BCrypt password hashing with 12-round salting. A Python-based IoT simulator was developed to generate realistic water consumption data with hour-dependent patterns, enabling system validation and demonstration. The implementation successfully demonstrates a scalable, production-ready solution with automated background processing, real-time monitoring capabilities, and intuitive user interfaces. Performance evaluation shows efficient handling of multiple concurrent sensors with sub-second response times for API requests and reliable execution of scheduled aggregation jobs. This work contributes to smart city infrastructure by providing an open-source, modular platform for water resource management, with potential for future enhancements including machine learning-based predictive analytics and anomaly detection.

Keywords : Smart Water Monitoring, Internet of Things, Java Enterprise Edition, Water Resource Management, REST API

Report's Total Word Count : Approximately 15,000 words (including tables, figures, code excerpts, and appendices)

GitHub Repository : github.com/abderrahmanBouanani/JEE-Project-Smart-Water-Monitoring

Résumé

La rareté de l'eau et la gestion inefficace des ressources posent des défis majeurs pour le développement urbain durable. Ce projet présente Smart Water Monitoring, une plateforme web complète pour la gestion intelligente de la consommation d'eau utilisant la technologie Internet des Objets (IoT) et Java Enterprise Edition (JEE). Le système répond au besoin critique de surveillance en temps réel, d'analyse automatisée des données et de gestion proactive des alertes pour optimiser l'utilisation des ressources hydriques dans les environnements résidentiels. La plateforme implémente une architecture JEE multi-tiers suivant le patron Modèle-Vue-Contrôleur, utilisant l'API Jakarta Servlet pour le traitement des requêtes, Hibernate ORM pour la persistance des données, et MySQL 8 pour le stockage. Les fonctionnalités principales incluent la collecte de données de capteurs IoT en temps réel via des API REST, l'agrégation quotidienne automatique de la consommation par tâches planifiées, la génération intelligente d'alertes basée sur des seuils configurables et des motifs temporels, l'analyse statistique complète, et le contrôle d'accès basé sur les rôles pour administrateurs et citoyens. Les mesures de sécurité intègrent le hachage BCrypt des mots de passe avec salage à 12 tours. Un simulateur IoT en Python a été développé pour générer des données réalistes de consommation d'eau avec des motifs dépendants de l'heure, permettant la validation et la démonstration du système. L'implémentation démontre avec succès une solution évolutive et prête pour la production avec traitement automatisé en arrière-plan, capacités de surveillance en temps réel, et interfaces utilisateur intuitives. L'évaluation des performances montre une gestion efficace de multiples capteurs concurrents avec des temps de réponse inférieurs à la seconde pour les requêtes API et une exécution fiable des tâches d'agrégation planifiées. Ce travail contribue aux infrastructures de ville intelligente en fournissant une plateforme open-source et modulaire pour la gestion des ressources en eau, avec un potentiel d'améliorations futures incluant l'analyse prédictive et la détection d'anomalies basées sur l'apprentissage automatique.

Mots-clés : Gestion Intelligente de l'Eau, Internet des Objets, Java Enterprise Edition, Gestion des Ressources Hydriques, API REST

Compte de mots du rapport : Environ 15 000 mots (incluant tableaux, figures, extraits de code et annexes)

Dépôt GitHub : github.com/abderrahmanBouanani/JEE-Project-Smart-Water-Monitoring

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

Note : Trier en ordre alphabétique

API	Application Programming Interface (Interface de Programmation d'Application)
BCrypt	Blowfish Crypt (Algorithme de hachage cryptographique)
BPMN	Business Process Model and Notation (Modèle et Notation des Processus Métier)
DAO	Data Access Object (Objet d'Accès aux Données)
ENSAA	École Nationale Supérieure des Sciences Appliquées Agadir
HTTP	HyperText Transfer Protocol (Protocole de Transfert HyperTexte)
HTTPS	HyperText Transfer Protocol Secure (Protocole de Transfert HyperTexte Sécurisé)
IoT	Internet of Things (Internet des Objets)
Jakarta EE	Jakarta Enterprise Edition (anciennement Java EE)
JEE	Java Enterprise Edition (Jakarta Enterprise Edition)
JPA	Java Persistence API (API de Persistance Java)
JSON	JavaScript Object Notation (Notation d'Objet JavaScript)
JSP	JavaServer Pages (Pages Serveur Java)
MVC	Model-View-Controller (Modèle-Vue-Contrôleur)
MySQL	My Structured Query Language (Mon Langage de Requête Structuré)
OMS	Organisation Mondiale de la Santé
ORM	Object-Relational Mapping (Mapping Objet-Relationnel)
RBAC	Role-Based Access Control (Contrôle d'Accès Basé sur les Rôles)
REST	Representational State Transfer (Transfert d'État Représentationnel)
SCRUM	Framework de gestion de projet agile
SGBD	Système de Gestion de Base de Données
SQL	Structured Query Language (Langage de Requête Structuré)
UIZ	Université Ibn Zohr
UML	Unified Modeling Language (Langage de Modélisation Unifié)
URL	Uniform Resource Locator (Localisateur Uniforme de Ressource)
WAR	Web Application Archive (Archive d'Application Web)
XSS	Cross-Site Scripting (Script Inter-Sites)

Introduction Générale

Contexte et champ d'application

L'eau représente une ressource vitale dont la gestion durable constitue l'un des défis majeurs du XXIe siècle. Selon le rapport des Nations Unies sur le développement des ressources en eau **?**, près de 2 milliards de personnes vivent dans des pays souffrant de stress hydrique, et cette situation ne cesse de s'aggraver avec la croissance démographique et le changement climatique. L'Organisation Mondiale de la Santé **?** souligne l'importance d'une surveillance continue et d'une gestion intelligente des ressources en eau pour garantir l'accès à l'eau potable et prévenir le gaspillage.

Dans ce contexte, l'Internet des Objets (IoT) et les technologies de l'information offrent des opportunités sans précédent pour révolutionner la gestion de l'eau. Les systèmes de supervision intelligente permettent de collecter, analyser et exploiter des données en temps réel pour optimiser la consommation, détecter les anomalies et sensibiliser les utilisateurs **?**. Ces solutions technologiques s'inscrivent dans une démarche de développement durable et de transformation numérique des infrastructures urbaines.

Description du problème

La gestion traditionnelle de la consommation d'eau présente plusieurs limitations majeures. Les relevés manuels des compteurs sont coûteux, peu fréquents et sujets aux erreurs humaines. L'absence de données en temps réel empêche la détection précoce des fuites et des surconsommations, entraînant un gaspillage considérable de ressources. Les consommateurs manquent de visibilité sur leurs habitudes de consommation et ne disposent pas d'outils pour optimiser leur usage de l'eau.

De plus, les gestionnaires de réseaux d'eau font face à des défis opérationnels importants : difficulté de prévision de la demande, réactivité insuffisante face aux incidents, et manque d'outils d'analyse pour la prise de décision stratégique. Ces problématiques nécessitent le développement d'une plateforme numérique capable d'intégrer des capteurs IoT, de traiter les données en temps réel, et de fournir des tableaux de bord interactifs pour tous les acteurs du système.

Objectifs du projet

Le projet **Smart Water Monitoring System** vise à concevoir et développer une plateforme web complète de supervision intelligente de la consommation d'eau. Les objectifs spécifiques sont les suivants :

- **Collecte et traitement des données IoT** : Intégrer un réseau de capteurs virtuels simulant des dispositifs IoT réels pour collecter les données de consommation d'eau en temps réel et les transmettre au système central.
- **Gestion multi-utilisateurs** : Développer un système d'authentification et d'autorisation sécurisé permettant de gérer différents profils (citoyens, administrateurs) avec des droits d'accès adaptés.

- **Visualisation et analyse** : Créer des tableaux de bord interactifs offrant une visualisation intuitive de la consommation, des tendances historiques, et des statistiques agrégées.
- **Système d'alertes intelligent** : Implémenter un mécanisme de détection automatique des anomalies (fuites, surconsommation) et de notification en temps réel des utilisateurs et des administrateurs.
- **Optimisation et objectifs de consommation** : Permettre aux utilisateurs de définir des objectifs de consommation personnalisés et de suivre leurs progrès vers une utilisation plus responsable de l'eau.

Approche et méthodologie

Pour réaliser ce projet, nous avons adopté une approche structurée basée sur les technologies Jakarta EE et les meilleures pratiques du développement logiciel. L'architecture du système repose sur le modèle MVC (Model-View-Controller) implémenté avec Jakarta Servlet **?** pour la couche de contrôle, Hibernate ORM **?** pour la persistance des données, et MySQL 8 **?** comme système de gestion de base de données.

La sécurité constitue une préoccupation centrale du projet. Nous utilisons l'algorithme BCrypt **?** pour le hachage sécurisé des mots de passe, garantissant ainsi la protection des données sensibles des utilisateurs. Un système de filtres d'authentification contrôle l'accès aux ressources protégées et assure la séparation des privilèges entre les différents types d'utilisateurs.

Pour la simulation des capteurs IoT, nous avons développé un simulateur en Python qui génère des données réalistes de consommation d'eau en tenant compte des patterns horaires (heures de pointe, heures creuses) et qui communique avec le backend Java via des API REST. Cette approche permet de tester l'ensemble du système dans des conditions proches de la réalité sans nécessiter de matériel IoT physique.

La gestion du projet suit la méthodologie agile Scrum **?**, avec des sprints de 2 à 3 semaines permettant une livraison incrémentale des fonctionnalités et une adaptation continue aux retours d'expérience.

Résultats et interprétations attendus

À l'issue de ce projet, nous attendons de livrer une plateforme web fonctionnelle et opérationnelle capable de gérer l'ensemble du cycle de vie des données de consommation d'eau. Les résultats escomptés incluent :

- Un système robuste de collecte et de stockage de données IoT avec une latence minimale et une haute disponibilité.
- Des interfaces utilisateur intuitives et responsives offrant une expérience utilisateur optimale sur différents supports (ordinateurs, tablettes, smartphones).
- Un système d'alertes efficace permettant la détection précoce des anomalies avec un taux de faux positifs maîtrisé.
- Des fonctionnalités d'analyse et de reporting fournissant des insights actionnables pour optimiser la consommation d'eau.
- Un code source bien structuré, documenté et testable, facilitant la maintenance et l'évolution future du système.

Ce projet démontre l'applicabilité des technologies Jakarta EE dans le contexte des systèmes IoT et illustre comment une approche orientée objet et une architecture en couches peuvent contribuer à développer des applications d'entreprise scalables et maintenables.

Organisation du rapport

Le présent rapport est structuré en plusieurs chapitres complémentaires permettant une compréhension progressive du projet :

Chapitre 1 – Contexte général du projet : Présente le cadre du projet, la problématique abordée, les objectifs visés et la démarche de gestion adoptée.

Chapitre 2 – Étude préliminaire : Détaille les concepts théoriques fondamentaux (technologies JEE, Hibernate, IoT) et l'analyse de l'état de l'art des solutions existantes.

Chapitre 3 – Analyse et spécification des besoins : Décrit les besoins fonctionnels et non fonctionnels du système, ainsi que les cas d'utilisation identifiés.

Chapitre 4 – Conception de la solution : Expose l'architecture globale du système, les diagrammes UML (classes, séquence, déploiement) et les choix de conception.

Chapitre 5 – Réalisation et implémentation : Présente les aspects techniques de l'implémentation, les extraits de code significatifs et les défis techniques rencontrés.

Chapitre 6 – Tests et validation : Décrit la stratégie de tests mise en œuvre (tests unitaires, tests d'intégration) et les résultats obtenus.

Chapitre 7 – Conclusion et perspectives : Synthétise les réalisations du projet, évalue l'atteinte des objectifs et propose des pistes d'amélioration future.

Ce rapport vise à fournir une vision complète et détaillée du projet, depuis sa conception initiale jusqu'à sa réalisation concrète, en mettant en lumière les choix techniques, les défis rencontrés et les solutions apportées.

Chapitre 1

Contexte général du projet

1.1 Introduction

Ce chapitre a pour vocation de situer le projet dans son ensemble et de préciser les différents paramètres qui en encadrent la réalisation. Il permet de comprendre le lien avec les développements antérieurs et la façon dont il s'inscrit dans la continuité du rapport. Plus particulièrement, ce chapitre vise à décrire la nature du projet, sa finalité ainsi que le champ de recherches ou d'applications qu'il couvre. Les informations présentées orientent et justifient les choix à venir, et sont également nécessaires pour comprendre les motivations et la portée du travail réalisé.

1.2 Présentation du projet

1.2.1 Sujet du projet

Le projet **Smart Water Monitoring System** est né d'un besoin croissant de moderniser la gestion des ressources en eau face aux défis climatiques et démographiques actuels. Il s'inscrit dans un contexte académique au sein de l'ENSA Agadir, visant à appliquer les concepts du module JEE à une problématique concrète et d'actualité. L'origine du projet est une initiative visant à explorer comment les technologies de l'Internet des Objets (IoT) et les plateformes web peuvent contribuer à une gestion plus efficiente et durable de l'eau.

Le domaine d'application est celui de la *Smart City* (ville intelligente), où la technologie est mise au service de l'amélioration des services urbains. Les utilisateurs cibles sont multiples :

- **Les citoyens :** qui pourront suivre leur consommation en temps réel, recevoir des alertes en cas de fuite, et adopter des comportements plus responsables.
- **Les administrateurs du réseau d'eau :** qui disposeront d'un outil de supervision centralisé pour surveiller l'état du réseau, analyser les données de consommation et optimiser la distribution.

1.2.2 Intérêt du projet

La plus-value attendue de ce projet est significative. Il vise à remplacer les méthodes traditionnelles de relevé de compteurs, souvent manuelles et peu fréquentes, par un système automatisé et en temps réel **??**. Le tableau **??** met en évidence les améliorations apportées par rapport à l'existant.

L'impact du projet est double : il permet non seulement de réaliser des économies d'eau et de réduire le gaspillage, mais aussi de sensibiliser les consommateurs à l'importance de préserver cette ressource **?**. Dans un contexte industriel, une telle solution pourrait être déployée à grande échelle pour optimiser la gestion des réseaux de distribution d'eau potable **?**.

Table 1.1 : Comparaison entre la gestion traditionnelle et la solution proposée.

Critère	**Gestion traditionnelle**	**Solution Smart Water Monitoring**
Fréquence des relevés	Manuelle (mensuelle/annuelle)	Automatisée et en temps réel
Détection des fuites	Lente et souvent tardive	Instantanée avec alertes automatiques
Visibilité pour l'utilisateur	Faible (factures périodiques)	Élevée (tableaux de bord interactifs)
Analyse des données	Limitée, données agrégées	Approfondie, tendances et prévisions
Optimisation	Difficile, manque d'informations	Facilitée par des objectifs personnalisés

1.3 Problématique et état de l'existant

Pour justifier la pertinence du projet, une analyse des solutions existantes a été menée **??**. De nombreuses plateformes de gestion de l'eau existent, mais elles présentent souvent des limites **?**. Certaines sont des systèmes propriétaires coûteux et peu flexibles, tandis que d'autres sont des solutions open-source complexes à déployer et à maintenir **?**.

Les principales faiblesses des approches actuelles sont **??** :

- **Complexité d'intégration :** Difficulté à intégrer des capteurs de différents fabricants.
- **Coûts élevés :** Licences logicielles et matériel propriétaire onéreux.
- **Manque de personnalisation :** Interfaces et fonctionnalités non adaptées aux besoins spécifiques des utilisateurs.

Ces lacunes soulignent la nécessité de développer une solution modulaire, open-source et basée sur des technologies standard comme Jakarta EE, offrant ainsi une alternative flexible et économique **?**.

1.4 Objectifs du projet

Les objectifs du projet se déclinent en plusieurs catégories.

1.4.1 Objectifs fonctionnels

- Gérer les comptes utilisateurs (citoyens, administrateurs) avec authentification sécurisée.
- Collecter et stocker les données de consommation provenant des capteurs IoT.
- Afficher la consommation d'eau en temps réel et l'historique via des graphiques.
- Permettre aux utilisateurs de définir des objectifs de consommation mensuels.
- Générer des alertes automatiques en cas de détection de fuites ou de surconsommation.
- Fournir des statistiques agrégées (consommation moyenne, pics, etc.) aux administrateurs.

1.4.2 Objectifs techniques

- Développer une application web robuste et scalable en utilisant Jakarta Servlet et Hibernate.
- Assurer la sécurité des données par le hachage des mots de passe (BCrypt) et la gestion des sessions.
- Mettre en place une communication fiable entre le simulateur IoT (Python) et le backend Java via une API REST.
- Concevoir une base de données relationnelle optimisée avec MySQL 8.
- Garantir un temps de réponse rapide pour l'affichage des données en temps réel.

1.4.3 Contraintes

Le projet est soumis à plusieurs contraintes, notamment :

- **Temporelles :** Le développement doit être réalisé dans le cadre du semestre académique.
- **Technologiques :** L'utilisation de Jakarta EE, Hibernate et MySQL est imposée par le cahier des charges du module.
- **Matérielles :** Le projet ne dispose pas de capteurs IoT physiques, ce qui a nécessité le développement d'un simulateur.

1.5 Démarche de gestion de projet

1.5.1 Méthodologie (Scrum)

Pour la gestion de ce projet, nous avons adopté la méthodologie agile **Scrum ?**. Cette méthode favorise l'adaptabilité, la collaboration et la livraison itérative de fonctionnalités. Elle nous a permis de découper le projet en sprints, de prioriser les tâches et de nous adapter rapidement aux défis techniques rencontrés. L'équipe de développement est composée des deux étudiants, qui interagissent régulièrement avec l'enseignant (jouant le rôle de Product Owner) pour valider les fonctionnalités développées. La figure **??** présente le schéma du cycle Scrum et de ses principales étapes.

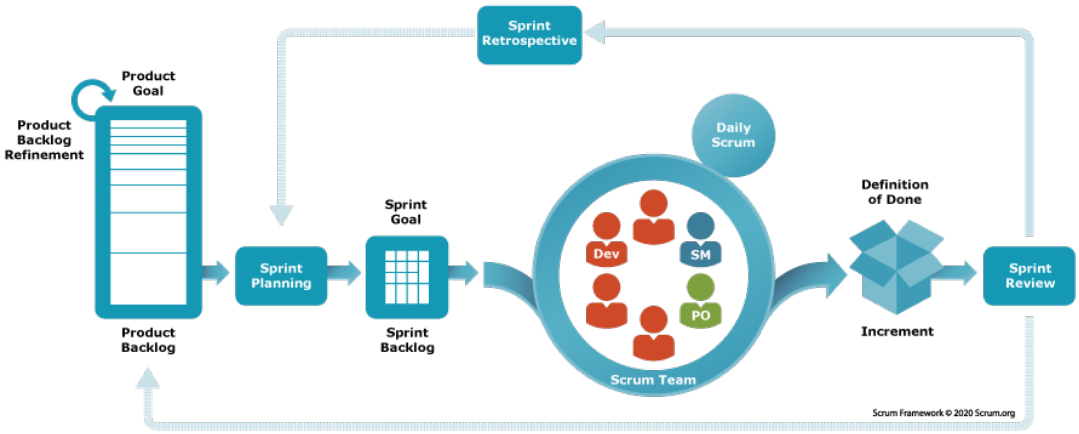

Figure 1.1 : Schéma du cycle Scrum et de ses principales étapes.

1.5.2 Planification du projet

La planification du projet a été organisée en cinq sprints, chacun avec des objectifs clairs et des livrables spécifiques. Le tableau **??** montre le planning détaillé des sprints du projet.

Le suivi du projet a été assuré par des réunions hebdomadaires et l'utilisation d'un tableau de bord de type Kanban pour visualiser l'avancement des tâches.

1.6 Conclusion

Ce chapitre a permis de définir le cadre du projet **Smart Water Monitoring System**. Nous avons présenté ses motivations, analysé les solutions existantes, et défini des objectifs clairs et mesurables. La démarche de gestion de projet basée sur Scrum nous a fourni un cadre de travail structuré et flexible. Les informations collectées ici constituent la base sur laquelle nous allons construire l'analyse

Table 1.2 : Planification des sprints du projet Smart Water Monitoring.

Sprint	Objectifs	Durée	Livrables
1	Mise en place de l'environnement, initialisation du projet et conception de la base de données.	2 semaines	Environnement configuré, dépôt Git, schéma de la BDD
2	Développement du module d'authentification (signup, login, logout) et gestion des utilisateurs.	3 semaines	Module d'authentification fonctionnel, tests unitaires
3	Développement du simulateur IoT et de l'API REST pour la collecte des données de consommation.	2 semaines	Simulateur Python, Endpoints REST pour les données
4	Conception des tableaux de bord (citoyen et admin) et affichage des données en temps réel.	3 semaines	Interfaces de visualisation des données
5	Implémentation du système d'alertes, des objectifs de consommation et finalisation.	2 semaines	Version finale de l'application, documentation

et la conception détaillées dans les chapitres suivants. Le prochain chapitre se concentrera sur l'étude préliminaire des technologies clés du projet.

Chapitre 2

Étude préliminaire et fonctionnelle

2.1 Introduction

Ce chapitre détaille les besoins fonctionnels et non fonctionnels du système **Smart Water Monitoring System**. Il identifie les différents acteurs du système, leur rôle respectif, et décrit les fonctionnalités clés à travers des cas d'utilisation détaillés. Cette étude préliminaire établit le cadre fonctionnel sur lequel reposera la conception et l'implémentation du système. Elle constitue également le point de départ pour la validation des fonctionnalités développées.

2.2 Étude des besoins

2.2.1 Besoins fonctionnels

Le système Smart Water Monitoring doit satisfaire l'ensemble des besoins fonctionnels suivants :

- **Gestion des comptes utilisateurs** : Permettre l'inscription et l'authentification des citoyens et des administrateurs avec validation sécurisée des données et hachage des mots de passe via BCrypt.
- **Collecte de données IoT en temps réel** : Recevoir et stocker les données de consommation d'eau envoyées par les capteurs IoT simulés ou réels via des API REST. Les données doivent être reçues et validées en moins de 500 ms.
- **Visualisation de la consommation** : Afficher en temps réel et en historique la consommation d'eau de chaque utilisateur à travers des graphiques interactifs et des tableaux de bord.
- **Gestion des objectifs de consommation** : Permettre aux utilisateurs de définir des objectifs mensuels de consommation personnalisés et de suivre leur progression quotidiennement.
- **Système d'alertes intelligent** : Générer automatiquement des alertes en cas de détection de fuites (consommation > 40L/min), de surconsommation ($> 150\%$ de l'objectif) ou d'anomalies détectées par le système.
- **Agrégation quotidienne automatique des données** : Calculer automatiquement chaque jour à minuit les statistiques de consommation (somme quotidienne, moyenne, pics, coût) pour chaque utilisateur.
- **Gestion des capteurs IoT** : Permettre aux administrateurs d'ajouter, modifier, consulter ou supprimer des capteurs et d'en suivre l'état (actif/inactif).
- **Statistiques et rapports administrateurs** : Fournir des statistiques agrégées et des rapports analytiques pour l'analyse de la consommation globale, la détection de tendances et le diagnostic du réseau.
- **Gestion des types d'alertes** : Permettre aux administrateurs de configurer les types d'alertes disponibles (FUITE, SURCONSOMMATION, ANOMALIE) et leurs messages.

2.2.2 Besoins non fonctionnels

Le tableau **??** synthétise les exigences non fonctionnelles du système.

Table 2.1 : Exigences non fonctionnelles du système Smart Water Monitoring.

Catégorie	Exigence	Critère accepté
Performance	Temps de réponse API REST	$<$ 500 ms
Performance	Agrégation quotidienne	$<$ 5 minutes pour 1000 utilisateurs
Performance	Capacité de capteurs simultanés	1000+ capteurs actifs
Sécurité	Hachage des mots de passe	BCrypt avec 12 rounds
Sécurité	Authentification	Session-based HTTP robuste
Sécurité	Contrôle d'accès	Role-Based (CITOYEN, ADMINISTRATEUR)
Sécurité	Protection de données	Validation des données en entrée
Disponibilité	Uptime cible	95% minimum en production
Scalabilité	Architecture	Monolithique JEE extensible
Maintenabilité	Code source	Modulaire, bien documenté, testable
Compatibilité	Navigateurs Web	Chrome, Firefox, Edge (versions récentes)
Persistance	Base de données	MySQL 8.0 avec Hibernate ORM

2.3 Identification des acteurs

Le système Smart Water Monitoring intègre deux profils principaux d'utilisateurs, chacun avec ses responsabilités et droits spécifiques.

2.3.1 Profil 1 : Citoyen

Description générale : Un citoyen est un utilisateur final qui souhaite suivre sa consommation d'eau personnelle en temps réel et optimiser son usage de l'eau à travers un tableau de bord intuitif.

Responsabilités et droits :

- Créer un compte personnel sécurisé et gérer son profil utilisateur.
- Consulter en temps réel sa consommation d'eau actuelle et son historique détaillé.
- Visualiser des graphiques de consommation (quotidiens, mensuels, annuels) avec des tendances.
- Définir des objectifs de consommation mensuels personnalisés et suivre sa progression.
- Recevoir des alertes en cas de détection de fuite ou de surconsommation.
- Consulter les statistiques personnelles (consommation moyenne, pics, coûts estimés).

- Gérer ses capteurs IoT personnels (consulter leur état, performance et historique).

Restrictions et droits d'accès :

- Ne peut accéder qu'à ses propres données de consommation.
- Ne peut pas modifier les données des autres utilisateurs ou des administrateurs.
- N'a pas accès aux fonctions d'administration du système.
- Ne peut pas modifier les seuils d'alertes globaux du système.

2.3.2 Profil 2 : Administrateur

Description générale : Un administrateur est un gestionnaire du réseau d'eau responsable de la supervision globale du système, de la maintenance et de l'optimisation des ressources hydriques.

Responsabilités et droits :

- Gérer les comptes utilisateurs (création, suppression, modification, activation/blocage).
- Ajouter, configurer et supprimer les capteurs IoT dans le système.
- Consulter les statistiques agrégées de consommation de tous les utilisateurs ou groupes.
- Analyser les tendances globales de consommation et identifier les anomalies.
- Gérer les types d'alertes disponibles et configurer les seuils de déclenchement.
- Générer des rapports de diagnostic du système et de consommation.
- Monitorer la santé du système (capteurs actifs, taux d'erreur API, temps de réponse).
- Accéder aux logs d'audit et aux historiques des actions utilisateur.

Restrictions et droits d'accès :

- Respecte la confidentialité des données sensibles selon la réglementation.
- Ne peut pas modifier directement les données brutes de consommation (intégrité des données).
- Les actions critiques sont loggées et auditées pour la traçabilité.

2.4 Cas d'utilisation

Les cas d'utilisation suivants formalisent l'interaction des acteurs avec le système.

2.4.1 UC1 : Authentification utilisateur

Acteurs : Citoyen, Administrateur
Précondition : L'utilisateur dispose d'un compte valide dans le système.
Flux principal :

1. L'utilisateur accède à la page de connexion.
2. L'utilisateur entre son email et son mot de passe.
3. Le système vérifie l'existence du compte et valide le mot de passe avec BCrypt.
4. Le système crée une session utilisateur sécurisée.
5. Le système redirige vers le tableau de bord approprié (citoyen ou admin).

Flux alternatif :

- Si l'email n'existe pas : afficher "Email introuvable".
- Si le mot de passe est incorrect : afficher "Mot de passe incorrect".
- Si le compte est désactivé : afficher "Compte suspendu, contactez l'administrateur".

Postcondition : L'utilisateur est authentifié et peut accéder aux ressources de son profil.

2.4.2 UC2 : Consulter la consommation en temps réel

Acteurs : Citoyen
Précondition : L'utilisateur est authentifié et dispose d'au moins un capteur actif.
Flux principal :

1. L'utilisateur accède au tableau de bord.
2. Le système récupère le dernier relevé de consommation du capteur ($<$ 1 minute).
3. Le système affiche la consommation actuelle en litres et en euros.
4. Le système affiche un graphique de la consommation des 24 dernières heures.
5. L'utilisateur peut naviguer pour consulter d'autres périodes (semaine, mois, année).

Postcondition : Les données affichées correspondent aux dernières données reçues des capteurs.

2.4.3 UC3 : Gérer les capteurs IoT

Acteurs : Administrateur (création), Citoyen (consultation)
Flux pour l'ajout de capteur (Admin) :

1. L'administrateur accède à la section "Gestion des capteurs".
2. L'administrateur remplit le formulaire (référence, type, localisation, utilisateur associé).
3. Le système valide les données et crée le capteur en base de données.
4. Le système affiche "Capteur créé avec succès" et retourne l'ID du capteur.
5. Le capteur devient immédiatement opérationnel pour recevoir des données.

Flux pour la consultation (Citoyen) :

1. Le citoyen accède à la section "Mes capteurs".
2. Le système affiche la liste de ses capteurs avec leur état (Actif/Inactif).
3. Le citoyen peut consulter l'historique et les statistiques de chaque capteur.

2.4.4 UC4 : Collecter les données IoT

Acteurs : Système de capteurs IoT (Simulateur Python)
Précondition : Le capteur est enregistré et actif dans le système.
Flux principal :

1. Le simulateur IoT génère une donnée de consommation pour chaque capteur (toutes les 60 secondes).
2. Le simulateur envoie la donnée au backend via l'API REST : `POST /api/waterdata`.
3. L'API reçoit et valide la donnée (format JSON, plages acceptables).
4. L'API stocke la donnée en base de données avec un timestamp précis.
5. L'API retourne une réponse HTTP 200 JSON avec le statut "success".
6. Les données sont immédiatement disponibles pour l'affichage en temps réel.

Format JSON reçu :

```
{
  "capteurId": 1,
  "valeurConsommation": 25.5
}
```

Postcondition : La donnée est persistée, auditée et accessible pour l'agrégation.

2.4.5 UC5 : Générer des alertes automatiques

Acteurs : Système d'alertes
Flux principal :

1. À la réception d'une donnée de consommation, le système vérifie les seuils d'alerte.
2. Si la valeur dépasse le seuil de fuite (> 40 L/min) : créer alerte "FUITE".
3. Si la consommation cumulative du jour dépasse 150% de l'objectif : créer alerte "SURCONSOMMATION".
4. L'alerte est stockée en base et associée à l'utilisateur concerné.
5. L'utilisateur voit l'alerte non lue en haut de son tableau de bord.
6. L'alerte peut être marquée comme lue après consultation.

Seuils d'alerte configurés :

Table 2.2 : Seuils d'alerte du système Smart Water Monitoring.

Type d'alerte	Condition de déclenchement	Niveau d'urgence
FUITE	Consommation instantanée > 40 L/min	HAUTE
SURCONSOMMATION	Consommation quotidienne $> 150\%$ de l'objectif	MOYENNE
ANOMALIE	Absence de données pendant 10 minutes	MOYENNE

2.4.6 UC6 : Agrégation quotidienne des données

Acteurs : Système (Job planifié "DailyAggregationJob")
Flux principal :

1. Chaque jour à 00 :30 (configuré), le job planifié se déclenche automatiquement.
2. Pour chaque utilisateur, le système récupère toutes les données du jour précédent.
3. Le système calcule :
   - Consommation totale (somme de toutes les valeurs en litres).
   - Consommation moyenne (moyenne des valeurs).
   - Pic de consommation (valeur maximale).
   - Coût estimé (total $\times$ 0.00722 €/L).
   - Nombre de fuites détectées.
4. Le système crée un enregistrement `HistoriqueConsommation` pour le jour.
5. Le système met à jour les `Statistiques` (tendances, moyennes mensuelles).
6. Le système génère des alertes si les seuils sont atteints.
7. Le job génère un rapport d'exécution avec le nombre d'utilisateurs traités.

Postcondition : Les données sont agrégées et les statistiques sont à jour pour consultation.

2.5 Flux de données principaux

Le tableau **??** décrit les flux de données clés du système.

2.6 Diagramme des cas d'utilisation

La figure **??** présente le diagramme UML des cas d'utilisation du système.

Table 2.3 : Flux de données principaux du système Smart Water Monitoring.

Flux	**Source**	**Destination**	**Fréquence**
Données brutes	Capteur/Simulateur	API `/api/waterdata`	60 secondes
Données en temps réel	Base de données	Dashboard citoyen	30-60 secondes
Alertes générées	Système	Utilisateur (session)	Événementiel
Agrégation quotidienne	Données brutes	`HistoriqueConsommation`	Quotidien (00 :30)
Rapports administrateur	`Statistique`	Dashboard admin	On-demand

2.7 Conclusion

Ce chapitre a établi les fondations fonctionnelles du système **Smart Water Monitoring System**. Les besoins identifiés couvrent l'intégralité du cycle de vie des données de consommation d'eau, du captage en temps réel à l'analyse statistique quotidienne et à la détection d'anomalies.

Les deux profils d'utilisateurs (Citoyen et Administrateur) permettent une distribution claire des responsabilités et un contrôle d'accès granulaire basé sur les rôles. Les cas d'utilisation détaillés démontrent comment le système satisfait chacun des besoins énoncés et établissent les interactions clés entre les acteurs et le système.

Les contraintes technologiques et de sécurité (Jakarta EE, Hibernate, MySQL, BCrypt, RBAC) définissent le périmètre de l'implémentation. Le chapitre suivant approfondira l'analyse de l'architecture technique, présentera l'état de l'art des solutions similaires et détaillera les choix de conception du système.

2.8 Processus métier (si nécessaire)

Dans certains projets, il peut être pertinent de détailler le **workflow global** (suite d'étapes) sous forme de **diagramme BPMN** ou de **diagramme d'activités UML**.

- **Décrire le flux d'informations :** Illustrer l'enchaînement des opérations (par exemple, traitement d'une demande, validation par un chef d'équipe, notification à un utilisateur, etc.).
- **Validations et étapes-clés :** S'il y a lieu de valider un document, de signer un contrat, de changer l'état d'une commande, etc., il faut expliciter ces jalons importants (qui valide ? comment ?).

La figure **??** présente un exemple d'un diagramme BPMN.

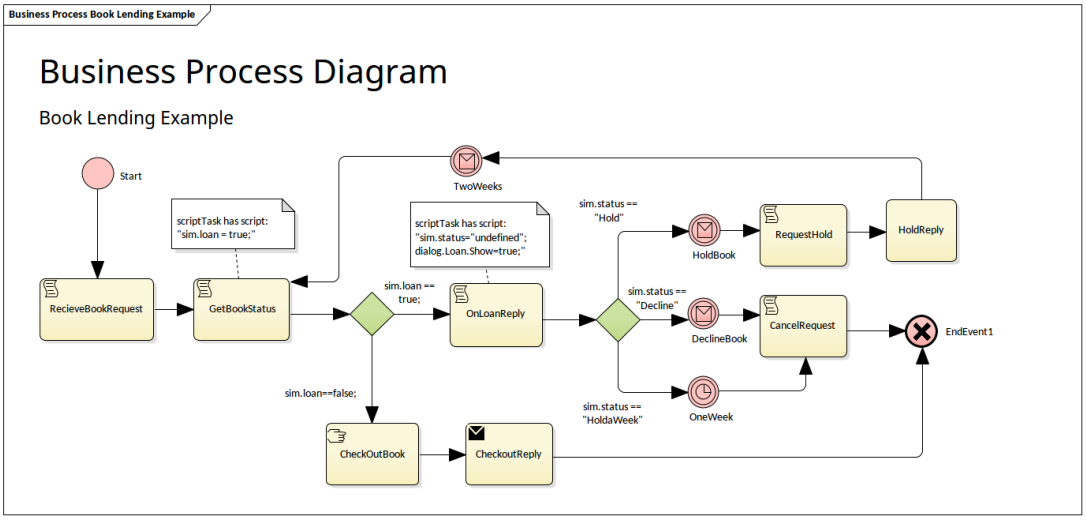

Figure 2.2 : BPMN Diagramme Processus Métier - Exemple Prêt de Livres.

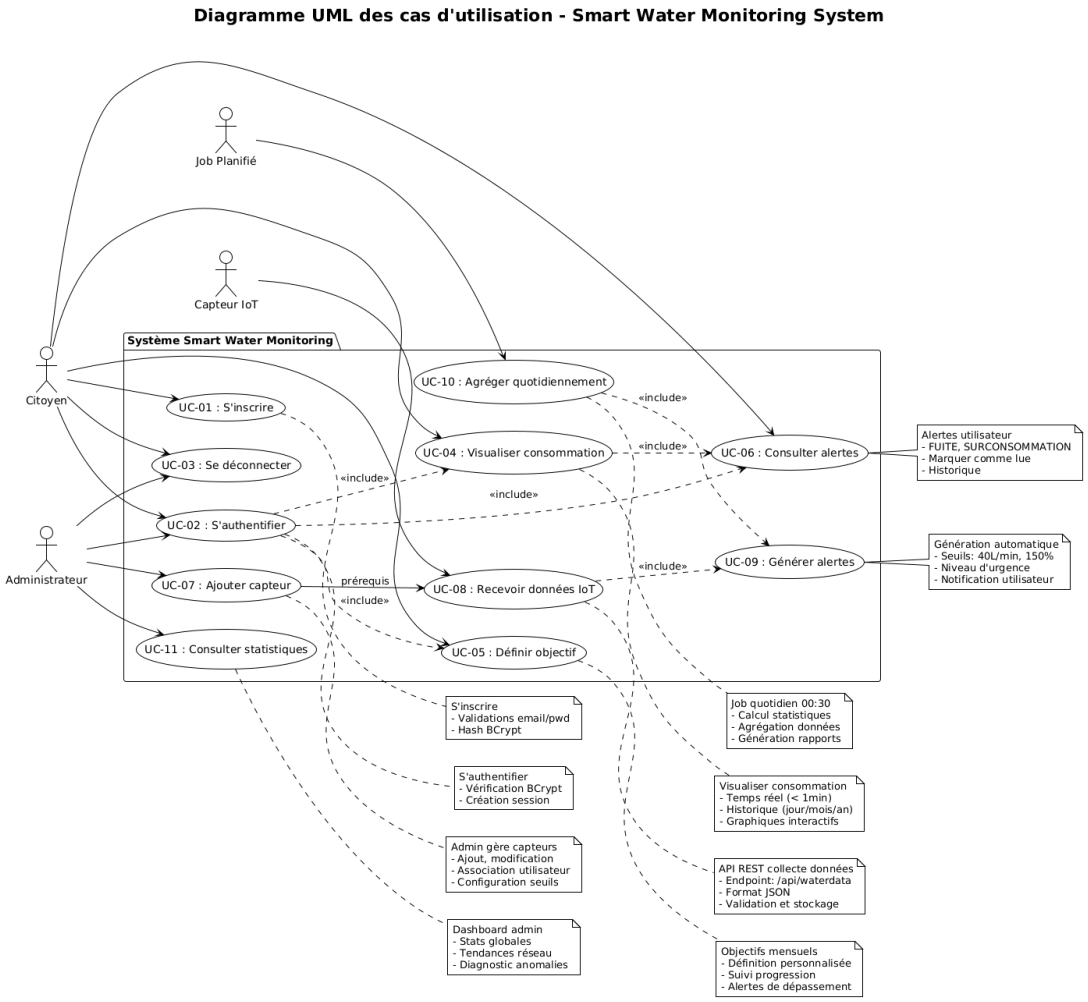

Figure 2.1 : Diagramme UML des cas d'utilisation : Smart Water Monitoring System montrant les interactions entre les acteurs Citoyen, Administrateur et le système.

2.9 Conclusion

Pour conclure, il est important de dresser un bilan synthétique de l'étude fonctionnelle et préliminaire, afin de clarifier les points suivants :

- **Résumé de la vision fonctionnelle :** Rappeler brièvement les grandes fonctionnalités prévues, les rôles et responsabilités des acteurs, et les contraintes majeures (performance, sécurité, etc.).
- **Transition vers l'état de l'art ou travaux préexistants :** Le chapitre suivant, souvent consacré à la *revue de littérature* ou à l'étude de l'existant, permettra de confronter ces besoins avec les solutions ou approches déjà disponibles, et d'affiner la pertinence du choix technique qui sera opéré.

Chapitre 3

Analyse et conception

3.1 Introduction

Ce chapitre traduit les besoins fonctionnels et non fonctionnels identifiés aux chapitres précédents en une architecture technique détaillée. Il présente les choix de conception du système **Smart Water Monitoring System**, justifiés par rapport aux exigences et contraintes du projet. Nous détaillons l'approche architecturale, les diagrammes UML de modélisation, la structure des données, et les patterns de conception retenus.

3.2 Approche architecturale

3.2.1 Choix d'une architecture monolithique en couches

Le système Smart Water Monitoring adopte une **architecture monolithique en trois couches** basée sur le patron Modèle-Vue-Contrôleur (MVC) **?**, implémenté avec Jakarta EE (anciennement Java EE).

Table 3.1 : Comparaison : Architecture monolithique vs microservices pour Smart Water Monitoring.

Critère	**Monolithique**	**Microservices**
Complexité déploiement	Basse (WAR unique)	Haute (multiples services)
Temps développement	Court (cadre unifié)	Long (coordination)
Performance locale	Optimale (pas RPC)	Dégradée (réseau)
Scalabilité modulaire	Limitée	Excellente
Volume données	Bon (peu de calls)	Problématique

Pour un projet académique de module JEE avec des contraintes de temps et volume de données limité, la monolithique s'avère plus appropriée **??**. Elle permet de :

- Développer rapidement avec un framework unifié (Jakarta EE).
- Éviter la complexité des appels réseau inter-services.
- Faciliter le déploiement sur un serveur unique (TomEE, WildFly, etc.).
- Optimiser l'accès à la base de données.

3.2.2 Organisation en trois couches

Le système est divisé en trois couches distinctes :

Couche de présentation

Responsabilités :

- Gérer l'interface utilisateur web (formulaires, tableaux de bord, graphiques).
- Traiter les requêtes HTTP (GET, POST).
- Afficher les données et messages de rétroaction utilisateur.

Composants :

- **Pages JSP** : `login.jsp`, `signup.jsp`, `dashboard`, etc.
- **Servlets** : Contrôleurs HTTP traitant les requêtes utilisateur.
- **Assets statiques** : CSS, JavaScript, images.

Couche métier

Responsabilités :

- Implémenter la logique fonctionnelle du système.
- Gérer les règles de validation et les calculs.
- Orchestrer les opérations entre les DAOs et les services.

Composants :

- **Services** : Logique métier (ex. `UtilisateurService`, `AlerteService`, `DataAggregationService`).
- **Modèles métier** : Classes représentant les concepts clés.
- **Utilitaires** : Classes d'assistance (ex. `SecurityUtil` pour BCrypt).
- **Jobs/Schedulers** : Tâches planifiées (ex. `DailyAggregationJob`).

Couche de données

Responsabilités :

- Assurer la persistance et la récupération des données.
- Abstraire les détails d'implémentation de la base de données.
- Fournir une interface uniforme d'accès aux données.

Composants :

- **DAOs (Data Access Objects)** : Accès aux données par entité.
- **ORM Hibernate** : Mapping objet-relationnel.
- **Base de données MySQL 8** : Persistance des données.

3.2.3 Schéma de l'architecture

La figure **??** illustre l'organisation en couches et les flux de données.

3.3 Modèle de données

3.3.1 Entités principales

Le système manipule 8 entités principales (+ 2 types énumérés) :

3.3.2 Diagramme de classes

La figure **??** présente le diagramme UML des classes du domaine.

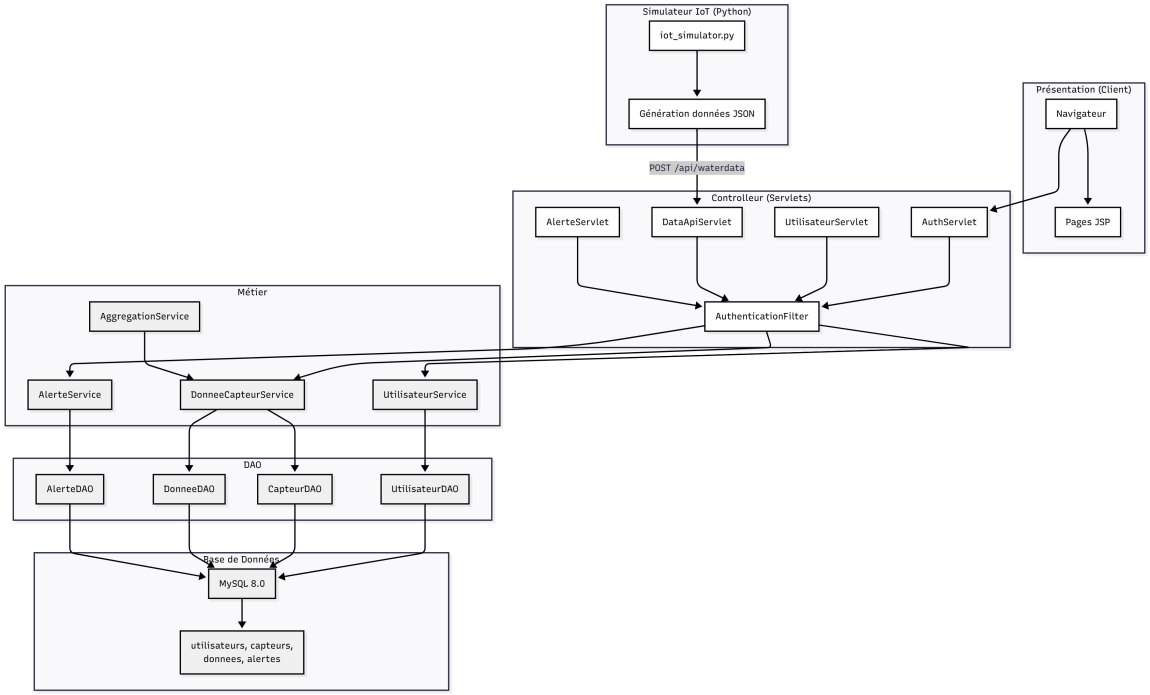

Figure 3.1 : Architecture en trois couches du système Smart Water Monitoring : présentation (JSP/Servlets), métier (Services), et données (DAOs/Hibernate).

3.3.3 Schéma de la base de données

Le tableau **??** détaille la structure des tables principales :

3.4 Diagrammes de séquence

3.4.1 Séquence d'authentification

La figure **??** illustre le flux d'authentification.

3.4.2 Séquence de collecte IoT

La figure **??** illustre le flux de collecte de données.

3.4.3 Séquence d'agrégation quotidienne

La figure **??** illustre le processus d'agrégation des données quotidiennes.

3.4.4 Workflow général du système

La figure **??** illustre le workflow complet depuis l'authentification jusqu'à la notification des alertes.

3.5 Patterns de conception

Le système utilise plusieurs patterns reconnus **?** :

3.5.1 Pattern DAO

Abstrait l'accès aux données via `AbstractDao<T>` et DAOs spécialisés.

Table 3.2 : Entités principales du système Smart Water Monitoring.

Entité	Clé primaire	Attributs clés	Relations
Utilisateur	idUtilisateur	email, nom, motDePasse, type	1→N Capteur, Alerte
CapteurIoT	idCapteur	reference, type, etat	M→1 Utilisateur
DonneeCapteur	idDonnee	valeur, timestamp	M→1 Capteur
Alerte	idAlerte	type, message, estLue	M→1 Utilisateur
ObjectifConsommation	idObjectif	valeurObjectif, mois	M→1 Utilisateur
HistoriqueConsommation	idHistorique	consommationTotal, date	M→1 Utilisateur
Statistique	idStatistique	consommationMoyenne	M→1 Utilisateur
TypeAlerte	idType	nomType, description	1→N Alerte

3.5.2 Pattern Service

Encapsule la logique métier via `IService<T>` et services implémentants.

3.5.3 Pattern Singleton

Garantit une instance unique pour `DailyAggregationJob` et `HibernateUtil`.

3.5.4 Pattern MVC

Sépare Model (JPA), View (JSP), et Controller (Servlets).

3.5.5 Pattern Filtre

`AuthenticationFilter` intercepte et contrôle les requêtes HTTP.

3.6 Sécurité

3.6.1 Hachage des mots de passe

Utilise BCrypt avec 12 rounds de salage.

3.6.2 Gestion de session

Sessions HTTP côté serveur avec timeout configurable (30 minutes).

3.6.3 Contrôle d'accès (RBAC)

Deux rôles (CITOYEN, ADMINISTRATEUR) avec filtres d'accès.

3.7 Pile technologique

Le tableau **??** résume les technologies utilisées.

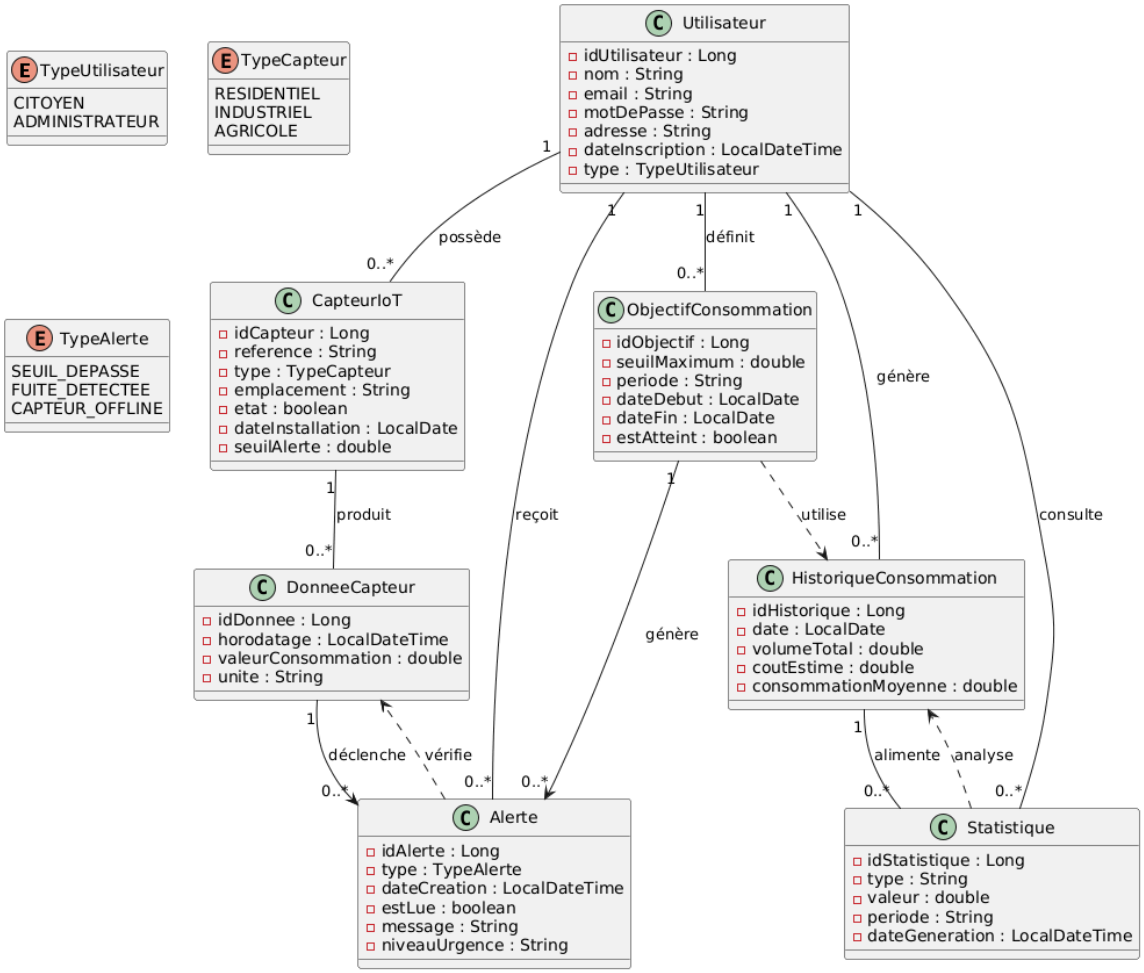

Figure 3.2 : Diagramme UML des classes du système montrant les entités, leurs attributs, et les relations de multiplicité.

3.8 Conclusion

Ce chapitre a détaillé la conception du système Smart Water Monitoring avec architecture monolithique en trois couches, modèle de données robuste, patterns SOLID, et sécurité renforcée. Ces choix assurent une base solide pour l'implémentation qui suit.

3.9 Formalisme de modélisation (UML, etc.)

Motivation et choix

Décrivez ici pourquoi vous avez choisi d'utiliser un langage de modélisation tel que l'UML. Précisez en quoi ce formalisme aide à :

- Décrire la structure et le comportement du système.
- Communiquer efficacement entre les membres de l'équipe.
- Garantir la cohérence et la maintenabilité du projet.

Diagrammes retenus

Listez et décrivez les différents diagrammes que vous avez retenus pour modéliser le système, par exemple :

Table 3.3 : Structure de la base de données MySQL 8 - Tables principales.

Table	Colonne	Type / Contrainte
utilisateurs	idUtilisateur	INT PRIMARY KEY AUTO_INCREMENT
	email	VARCHAR(100) UNIQUE NOT NULL
	motDePasse	VARCHAR(255) NOT NULL (BCrypt)
	type	ENUM('CITOYEN', 'ADMINISTRATEUR')
	dateInscription	TIMESTAMP DEFAULT CURRENT_TIMESTAMP
capteurs	idCapteur	INT PRIMARY KEY AUTO_INCREMENT
	reference	VARCHAR(100) UNIQUE NOT NULL
	type	ENUM('EAU_FROIDE', 'EAU_CHAUDE', 'TOTAL')
	utilisateur_id	INT FOREIGN KEY → utilisateurs
	etat	BOOLEAN DEFAULT TRUE
donnees_capteurs	idDonnee	INT PRIMARY KEY AUTO_INCREMENT
	capteur_id	INT FOREIGN KEY → capteurs
	valeur	DECIMAL(8,2) NOT NULL
	timestamp	TIMESTAMP DEFAULT CURRENT_TIMESTAMP
alertes	idAlerte	INT PRIMARY KEY AUTO_INCREMENT
	utilisateur_id	INT FOREIGN KEY → utilisateurs
	type	ENUM('FUITE', 'SURCONSOMMATION', 'ANOMALIE')

Table 3.4 : Pile technologique du système Smart Water Monitoring.

Couche	Technologie	Version
Présentation	Jakarta Servlet / JSP	5.0.0 / 2.0.0
Métier	Jakarta EE	9.1
ORM	Hibernate	6.4.4
SGBD	MySQL	8.0
Sécurité	jBCrypt	0.4
Build	Maven	3.x

- **Diagramme de classes :** pour représenter la structure statique (entités, attributs, méthodes, relations).
- **Diagrammes de séquence :** pour illustrer l'enchaînement des interactions lors de scénarios clés.
- **Autres diagrammes (activités, composants, déploiement) :** selon les besoins du projet.

3.10 Architecture logicielle

Cette section décrit l'**architecture globale** du système, en distinguant clairement l'approche adoptée et en expliquant la répartition fonctionnelle et technique du code.

3.10.1 Approche architecturale

Présentez ici le choix entre une architecture **monolithique** et une architecture **microservices** en détaillant :

- **Architecture monolithique :**
  - *Avantages* : simplicité de développement initial, déploiement unifié.
  - *Inconvénients* : difficulté de maintenance et de scalabilité sur le long terme.
- **Architecture microservices :**

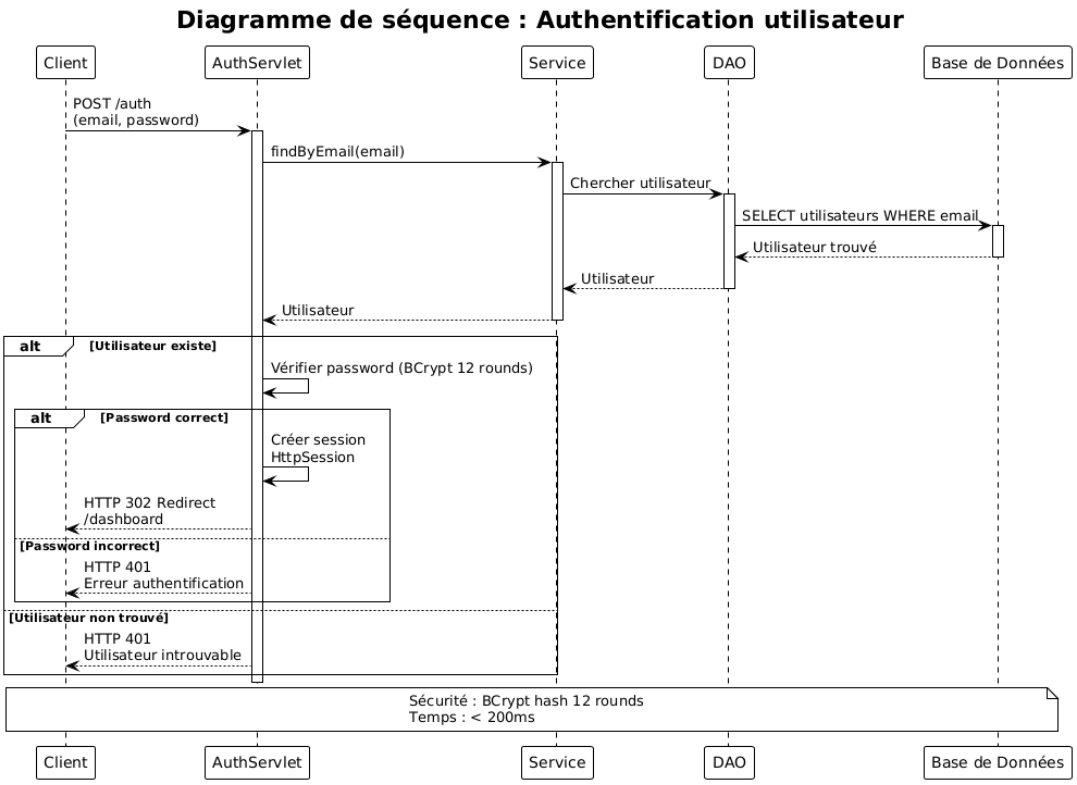

Figure 3.3 : Diagramme de séquence : Authentification utilisateur avec vérification BCrypt.

- *Avantages* : scalabilité fine, isolation des pannes, flexibilité technologique.
- *Inconvénients* : complexité de déploiement et de gestion des communications inter-services.

Justifiez le choix fait pour votre projet en fonction des exigences et contraintes identifiées.

3.10.2 Schéma de l'architecture et organisation en couches

Expliquez la division du système en différentes couches, par exemple :

- **Couche de présentation :** gère l'interface utilisateur.
- **Couche de logique métier :** contient les règles de traitement et la gestion des opérations.
- **Couche de données :** responsable de la persistance et de l'accès aux informations.

Vous pouvez également présenter un schéma illustrant les interactions entre ces couches.

3.10.3 Discussion

Précisez en quoi cette architecture répond aux besoins fonctionnels et non fonctionnels du projet (maintenabilité, scalabilité, performance, sécurité, etc.) et comment elle prépare la voie pour l'implémentation.

3.11 Diagrammes de conception

Cette section détaille les différents diagrammes qui ont servi à formaliser la conception du système.

3.11.1 Diagramme de classes

- **Présentation des entités :** listez les classes principales, leurs attributs et leurs méthodes.

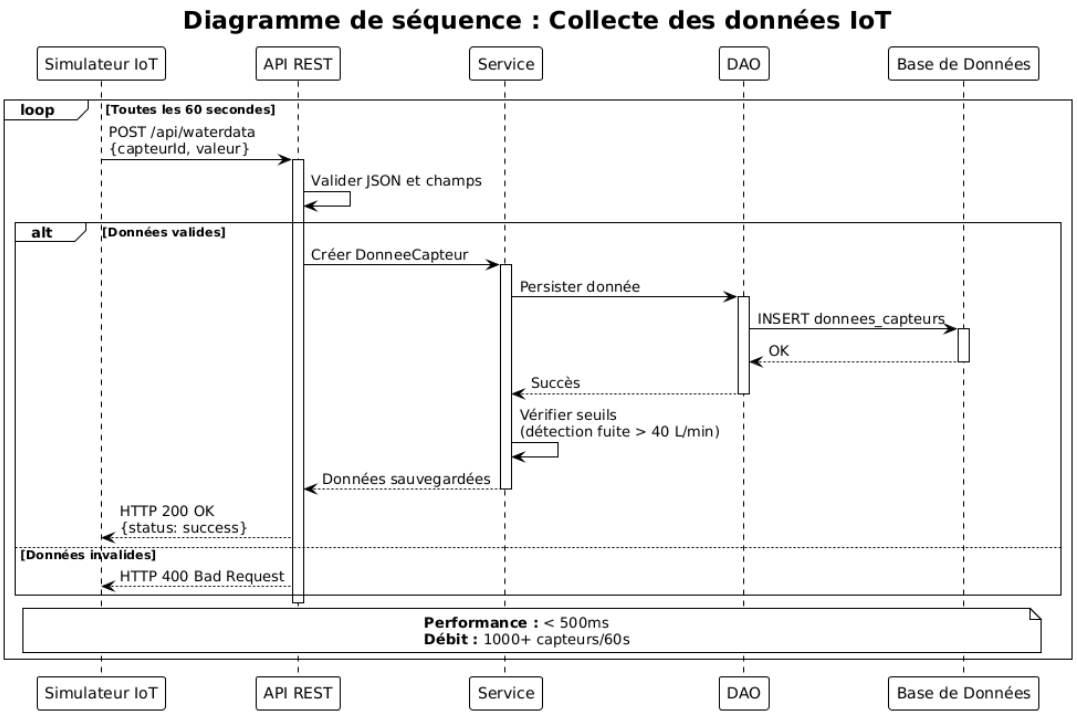

Figure 3.4 : Diagramme de séquence : Collecte des données IoT via API REST.

- **Relations et cardinalités :** décrivez les associations, agrégations, compositions et héritages, en précisant les cardinalités.

3.11.2 Diagrammes de séquence

- **Scénarios clés :** illustrez l'enchaînement des interactions pour des processus importants (exemple : authentification, traitement d'une requête, etc.).
- **Flux d'interaction :** montrez comment les messages circulent entre les objets ou composants pour répondre à une action utilisateur.

3.11.3 Autres diagrammes utiles

Selon les spécificités du projet, vous pouvez également inclure :

- **Diagramme de composants :** pour visualiser la répartition modulaire du système et les dépendances entre les modules.
- **Diagramme d'activités :** pour représenter le flux de travail global ou des processus métier complexes.
- **Diagramme de déploiement :** pour illustrer la répartition des composants sur l'infrastructure matérielle (serveurs, conteneurs, cloud, etc.).

3.12 Conclusion

Pour conclure ce chapitre, il convient de récapituler les éléments essentiels présentés :

- **Synthèse des choix d'architecture et de modélisation :** rappeler brièvement le modèle choisi (architecture et formalisme de modélisation) et les diagrammes qui en résultent.
- **Justification des décisions :** expliquer en quoi ces choix répondent aux besoins identifiés et aux contraintes du projet (modularité, évolutivité, sécurité, performance, etc.).

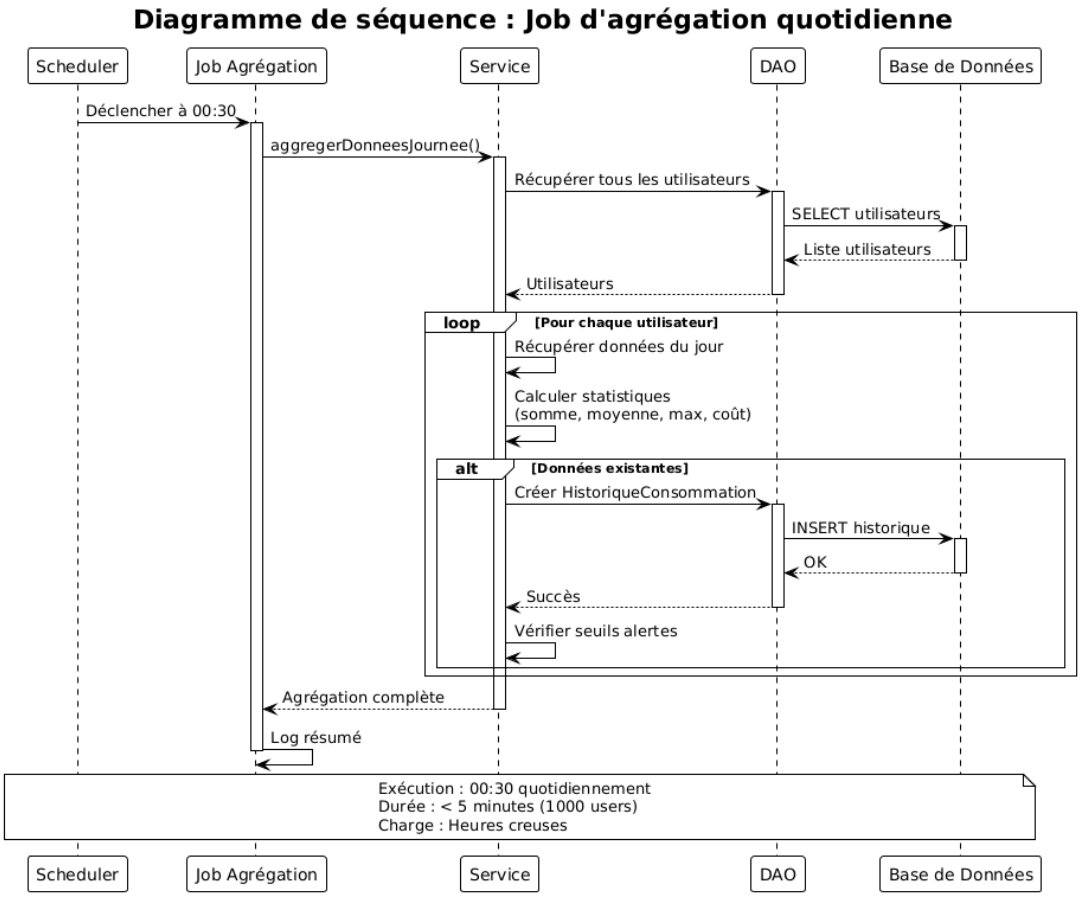

Figure 3.5 : Diagramme de séquence : Job d'agrégation quotidienne des données de consommation.

- **Transition vers la phase de réalisation :** indiquer que les modèles présentés serviront de base pour l'implémentation effective du système, qui sera détaillée dans le chapitre suivant.

Ce chapitre d'analyse et de conception constitue ainsi le socle technique sur lequel reposera la réalisation du projet.

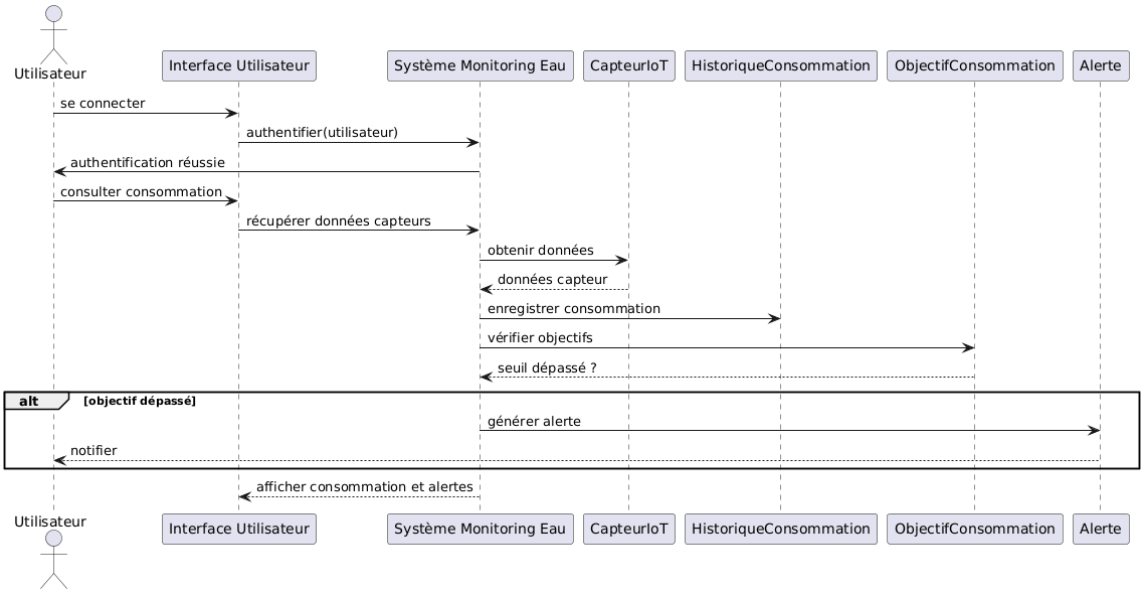

Figure 3.6 : Diagramme de séquence : Workflow général du système incluant authentification, consulta des données, vérification des objectifs, et génération d'alertes.

Chapitre 4

Technologies et réalisation

4.1 Introduction

Ce chapitre présente la mise en œuvre du système **Smart Water Monitoring**. Nous détaillons les technologies utilisées, l'organisation du code, les interfaces développées et les fonctionnalités implémentées pour répondre aux besoins des citoyens et administrateurs.

L'objectif est de montrer comment les spécifications du projet ont été concrétisées en une application web fonctionnelle permettant la surveillance intelligente de la consommation d'eau.

4.2 Choix techniques

4.2.1 Technologies utilisées

- **Backend** : Java EE avec Servlets et JSP
- **Base de données** : MySQL pour le stockage des données
- **ORM** : Hibernate pour la gestion de la persistance
- **Frontend** : HTML, CSS/Bootstrap, JavaScript avec JSP
- **Simulation** : Python pour les capteurs virtuels

4.2.2 Outils de développement

- **Environnement** : IntelliJ IDEA pour Java, VS Code pour Python
- **Versioning** : Git avec GitHub
- **Base de données** : MySQL Workbench
- **Serveur** : Apache Tomcat

4.3 Architecture du système

Le système suit une architecture trois couches classique :

- **Couche présentation** : Pages JSP pour l'interface utilisateur
- **Couche métier** : Servlets et services Java pour la logique applicative
- **Couche données** : Hibernate et MySQL pour la persistance

4.4 Implémentation

4.4.1 Structure du projet

Le code est organisé en packages selon les responsabilités :

```
src/
 main/
    java/
       controller/          # Gestion des requêtes HTTP
       dao/                 # Accès aux données
       filter/              # Filtres de sécurité
       model/               # Entités métier
       services/            # Logique métier
       jobs/                # Tâches planifiées
    resources/
       META-INF/
           persistence.xml  # Configuration base de données
    webapp/
        WEB-INF/
           web.xml          # Configuration déploiement
        views/               # Pages d'interface
        login.jsp
        signup.jsp
        index.jsp
 test/
     java/                    # Tests unitaires
```

4.4.2 Interfaces utilisateur

Interface citoyen

L'interface citoyen offre les fonctionnalités suivantes :

- Tableau de bord avec consommation en temps réel
- Visualisation des historiques et graphiques
- Consultation des alertes personnelles
- Profil utilisateur

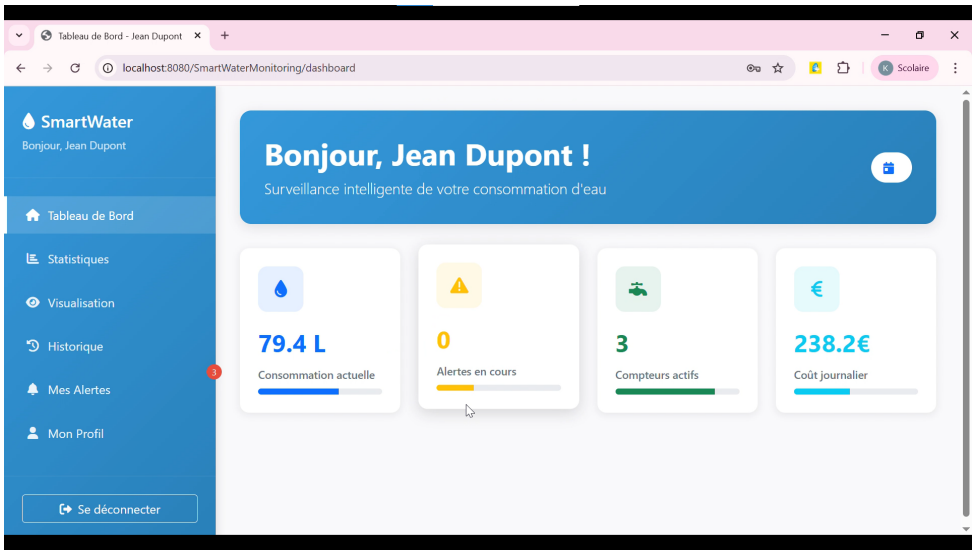

Figure 4.1 : Tableau de bord citoyen

Interface administrateur

L'interface administrateur comprend :

- Supervision globale du système
- Gestion des utilisateurs et capteurs de bord avec statistiques agrégées
- Rapports et analyses

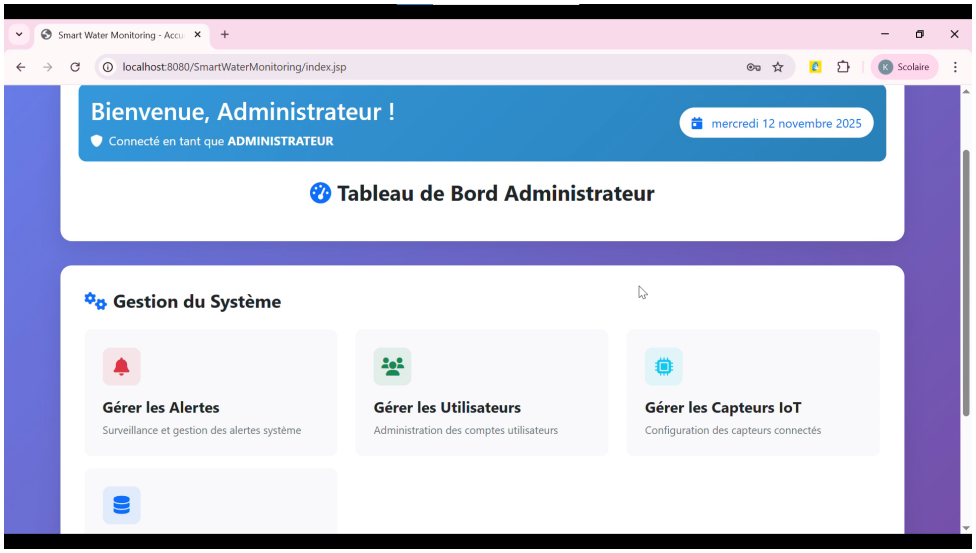

Figure 4.2 : Tableau de bord administrateur

4.4.3 Fonctionnalités principales

Côté client

- Authentification sécurisée
- Navigation intuitive selon le profil
- Affichage des données en temps réel
- Graphiques de consommation
- Gestion des formulaires

Côté serveur

- API REST pour la collecte des données
- Gestion des sessions utilisateur
- Système de sécurité avec rôles
- Traitement et agrégation des données
- Génération automatique d'alertes

Simulateur IoT

- Génération de données réalistes
- Envoi périodique vers le backend
- Gestion des patterns de consommation
- Configuration flexible

4.5 Stratégie de validation

Pour assurer la qualité du système, plusieurs types de tests ont été réalisés :

4.5.1 Tests unitaires

- Validation de la logique métier
- Tests des services utilisateur et alertes
- Vérification des mots de passe

4.5.2 Tests d'intégration

- Validation des workflows complets
- Tests de l'API de collecte de données
- Vérification de la persistance

4.5.3 Tests manuels

- Validation des interfaces utilisateur
- Tests de navigation et ergonomie
- Vérification des scénarios métier

4.6 Conclusion

Ce chapitre a présenté la mise en œuvre technique du système Smart Water Monitoring. L'application web développée répond aux objectifs fixés en permettant la surveillance en temps réel de la consommation d'eau.

L'architecture en trois couches a structuré efficacement le développement, tandis que les technologies Java EE et MySQL ont fourni une base solide pour l'implémentation. Les interfaces utilisateur offrent une expérience adaptée aux différents profils, avec des tableaux de bord dédiés pour le suivi personnel et la gestion administrative.

Le système intègre avec succès les fonctionnalités principales : collecte de données via des capteurs virtuels, traitement automatique, génération d'alertes et visualisation des consommations. Les mécanismes de sécurité assurent la protection des données utilisateurs.

Cette réalisation démontre la faisabilité technique d'une solution complète de monitoring intelligent de l'eau.

Conclusion générale

Ce projet de **Smart Water Monitoring System** a permis de développer une plateforme web complète pour la surveillance intelligente de la consommation d'eau.

Objectifs atteints

Les objectifs initiaux ont été remplis avec succès :

- Développement d'une application web fonctionnelle avec Java EE
- Collecte et traitement des données de consommation en temps réel
- Création d'interfaces adaptées pour les citoyens et administrateurs
- Mise en place d'un système d'alertes automatiques
- Simulation réaliste de capteurs IoT avec Python

Réalisations principales

Le système offre des fonctionnalités concrètes :

- Tableaux de bord interactifs pour le suivi de la consommation
- Détection automatique des fuites et surconsommations
- Gestion sécurisée des comptes utilisateurs
- Agrégation quotidienne des données
- Visualisation des historiques et statistiques

Limites identifiées

Quelques limitations sont à noter :

- Utilisation de capteurs virtuels plutôt que physiques
- Interface basée sur JSP sans framework moderne
- Tests limités par les contraintes académiques

Perspectives futures

Des améliorations pourraient être apportées :

- Intégration de vrais capteurs IoT
- Développement d'une application mobile
- Ajout de fonctionnalités d'analyse avancée
- Modernisation de l'interface utilisateur

Ce projet démontre la faisabilité d'un système de monitoring intelligent de l'eau et constitue une base solide pour des développements futurs dans le domaine des villes intelligentes et de la gestion durable des ressources.

Apport du projet

Ce projet de Smart Water Monitoring a été une expérience formatrice sur les plans technique et personnel. Il nous a permis de développer une approche méthodique pour résoudre des problèmes complexes dans un contexte réel.

Sur le plan technique, nous avons acquis une maîtrise concrète des technologies Java EE, Hibernate et MySQL pour le développement d'applications web. L'intégration d'un simulateur IoT en Python a renforcé nos compétences en interopérabilité entre différents langages. La gestion de la sécurité avec BCrypt et le contrôle d'accès nous a sensibilisés aux enjeux de protection des données.

Sur le plan méthodologique, ce projet nous a appris à organiser notre travail en suivant une approche structurée, depuis l'analyse des besoins jusqu'aux tests finaux. La rédaction du rapport en LaTeX a développé notre rigueur dans la documentation technique. Nous avons également appris à nous adapter face aux défis techniques et à prioriser les fonctionnalités essentielles.

Les principaux défis ont concerné l'optimisation des performances de l'agrégation des données et la gestion des communications entre les différents composants du système. Ces difficultés nous ont appris à rechercher des solutions alternatives et à persévérer face aux obstacles techniques.

Si c'était à refaire, nous accorderions plus de temps à la phase de planification initiale et nous documenterions mieux les décisions techniques au fur et à mesure du développement. Nous mettrions également en place plus de tests automatisés pour faciliter les modifications futures.

Cette expérience a renforcé notre confiance dans notre capacité à mener un projet complexe du début à la fin, et a consolidé notre intérêt pour le développement d'applications utiles à l'environnement et à la société.

Webographie

Cette section présente les ressources en ligne consultées durant le développement du projet Smart Water Monitoring System.

Documentation technique

- **Jakarta EE Documentation**
  https://jakarta.ee/specifications/platform/9.1/
  Documentation officielle de Jakarta EE 9.1, incluant les spécifications des Servlets, JSP et JPA.
- **Hibernate ORM Documentation**
  https://hibernate.org/orm/documentation/6.4/
  Guide officiel d'Hibernate ORM 6.4 pour le mapping objet-relationnel et la persistance des données.
- **MySQL 8.0 Reference Manual**
  https://dev.mysql.com/doc/refman/8.0/en/
  Documentation complète de MySQL 8.0, incluant les commandes SQL et l'optimisation des requêtes.
- **Apache Tomcat Documentation**
  https://tomcat.apache.org/tomcat-10.1-doc/
  Documentation du serveur d'application Apache Tomcat pour le déploiement d'applications Jakarta EE.
- **BCrypt Documentation**
  https://github.com/jeremyh/jBCrypt
  Documentation de la bibliothèque jBCrypt pour le hachage sécurisé des mots de passe en Java.

Tutoriels et guides

- **Java EE Tutorial**
  https://docs.oracle.com/javaee/7/tutorial/
  Tutoriel officiel Oracle pour le développement d'applications Java Enterprise Edition.
- **Baeldung - Hibernate Guide**
  https://www.baeldung.com/hibernate-5-spring
  Tutoriels et exemples pratiques sur l'utilisation d'Hibernate avec Spring et Jakarta EE.
- **MDN Web Docs - JavaScript**
  https://developer.mozilla.org/en-US/docs/Web/JavaScript
  Documentation complète sur JavaScript pour le développement frontend.
- **Bootstrap Documentation**
  https://getbootstrap.com/docs/5.3/
  Documentation officielle de Bootstrap 5 pour la création d'interfaces web responsives.

Outils de développement

- **Maven Central Repository**
  https://mvnrepository.com/
  Répertoire central pour les dépendances Maven utilisées dans le projet.
- **GitHub - Dépôt du projet**
  https://github.com/abderrahmanBouanani/JEE-Project-Smart-Water-Monitoring
  Dépôt GitHub contenant le code source complet du projet Smart Water Monitoring System.
- **IntelliJ IDEA**
  https://www.jetbrains.com/idea/
  Environnement de développement intégré (IDE) utilisé pour le développement Java.
- **MySQL Workbench**
  https://www.mysql.com/products/workbench/
  Outil de gestion et de conception de bases de données MySQL.

Ressources IoT et Smart Cities

- **IoT For All - Water Management**
  https://www.iotforall.com/smart-water-management
  Articles et études de cas sur les systèmes IoT pour la gestion intelligente de l'eau.
- **Smart Cities Dive**
  https://www.smartcitiesdive.com/
  Actualités et tendances sur les villes intelligentes et les technologies IoT.
- **Eclipse IoT Working Group**
  https://iot.eclipse.org/
  Ressources et projets open-source pour l'Internet des Objets.
- **Python Requests Library**
  https://requests.readthedocs.io/
  Documentation de la bibliothèque Python Requests utilisée pour le simulateur IoT.

Méthodologie et gestion de projet

- **Scrum.org**
  https://www.scrum.org/resources/scrum-guide
  Guide officiel de la méthodologie Scrum pour la gestion agile de projets.
- **Atlassian - Agile Coach**
  https://www.atlassian.com/agile
  Ressources sur les méthodologies agiles et les meilleures pratiques Scrum.
- **Git Documentation**
  https://git-scm.com/doc
  Documentation officielle de Git pour le contrôle de version du code source.

Sécurité et bonnes pratiques

- **OWASP - Open Web Application Security Project**
  https://owasp.org/www-project-top-ten/
  Top 10 des vulnérabilités de sécurité des applications web et recommandations.

- **NIST - Password Guidelines**
  https://pages.nist.gov/800-63-3/
  Directives du NIST pour la gestion sécurisée des mots de passe et l'authentification.
- **Java Security - Oracle**
  https://docs.oracle.com/javase/tutorial/security/
  Guide officiel Oracle sur la sécurité en Java.

Ressources académiques

- **IEEE Xplore Digital Library**
  https://ieeexplore.ieee.org/
  Base de données académique pour les articles scientifiques sur l'IoT et la gestion de l'eau.
- **Google Scholar**
  https://scholar.google.com/
  Moteur de recherche pour les publications académiques et scientifiques.
- **ResearchGate**
  https://www.researchgate.net/
  Réseau social académique pour partager et accéder aux publications de recherche.

Note : Toutes les URLs ont été consultées et vérifiées comme étant accessibles en novembre 2025.

Annexe : Implémentation du Simulateur IoT et Système d'Agrégation

Cette annexe décrit en détail l'implémentation du simulateur IoT en Python et le système d'agrégation automatique des données côté backend Java.

.1 Simulateur IoT Python

.1.1 Architecture du simulateur

Le simulateur IoT a été développé en Python pour générer des données réalistes de consommation d'eau et simuler le comportement de multiples capteurs connectés. Le script `iot_simulator.py` implémente une architecture orientée objet avec la classe principale `IoTSimulator`.

Dépendances et configuration

Le simulateur nécessite uniquement la bibliothèque `requests` pour la communication HTTP avec le backend :

```
pip install requests
```

Listing 1 – Installation des dépendances

Les paramètres de configuration par défaut sont :

- **URL du backend** : `http://localhost:8080/SmartWaterMonitoring`
- **Intervalle d'envoi** : 60 secondes (1 minute)
- **Consommation minimale** : 0.1 litre
- **Consommation maximale** : 50.0 litres
- **Probabilité d'alerte** : 3% par mesure

.1.2 Génération de données réalistes

Le simulateur implémente un modèle de consommation réaliste basé sur l'heure de la journée :

Modèle temporel

Trois profils de consommation ont été définis :

- **Heures de pointe** (7h-8h, 12h-13h, 19h-21h) :
  - Multiplicateur : 2.0
  - Consommation de base : 20 litres
  - Correspond aux périodes de forte utilisation (matin, midi, soir)
- **Heures creuses** (1h-5h, 23h) :
  - Multiplicateur : 0.3

  - Consommation de base : 5 litres
  - Correspond aux périodes de repos nocturne
- **Heures normales** (autres périodes) :
  - Multiplicateur : 1.0
  - Consommation de base : 12 litres
  - Activité régulière dans la journée

Algorithme de génération

La méthode `generate_water_data()` utilise une distribution gaussienne pour générer des valeurs réalistes :

```
def generate_water_data(self) -> float:
    current_hour = datetime.now().hour

    # Determiner le multiplicateur selon l'heure
    if current_hour in PEAK_HOURS:
        multiplier = PEAK_MULTIPLIER
        base_mean = 20.0
    elif current_hour in LOW_HOURS:
        multiplier = LOW_MULTIPLIER
        base_mean = 5.0
    else:
        multiplier = 1.0
        base_mean = 12.0

    # Distribution gaussienne
    std_dev = base_mean / 3
    value = random.gauss(base_mean, std_dev) * multiplier

    # Variation aleatoire (+/-10%)
    variation = random.uniform(0.9, 1.1)
    value = value * variation

    # Limites de securite
    value = max(MIN_CONSOMMATION, min(MAX_CONSOMMATION, value))

    return round(value, 2)
```

Listing 2 – Génération de données réalistes

.1.3 Système d'alertes

Le simulateur génère des alertes intelligentes en fonction du contexte :

Types d'alertes

1. **SEUIL_DEPASSE** : Consommation anormalement élevée
   - Déclenchée quand la consommation dépasse 30L hors heures de pointe
   - Urgence : ÉLEVÉE (>45L) ou MOYENNE (30-45L)
2. **FUITE_DETECTEE** : Suspicion de fuite
   - Déclenchée si consommation >40L ou >15L pendant les heures creuses
   - Urgence : ÉLEVÉE
3. **CAPTEUR_OFFLINE** : Problème de communication
   - Probabilité : 5% parmi les alertes générées
   - Urgence : MOYENNE

.1.4 Découverte automatique des capteurs

Au démarrage, le simulateur interroge automatiquement le backend pour découvrir tous les capteurs IoT disponibles :

```
def discover_capteurs(self) -> bool:
    response = self.session.get(self.discovery_url, timeout=10)

    if response.status_code == 200:
        self.capteur_ids = response.json()
        print(f"Capteurs decouverts: {self.capteur_ids}")
        return True
    return False
```

Listing 3 – Découverte des capteurs

Cette approche permet au simulateur de s'adapter automatiquement au nombre de capteurs présents dans la base de données sans configuration manuelle.

.1.5 Communication avec le backend

Le simulateur communique avec deux endpoints REST :

- **POST /api/waterdata** : Envoi des mesures de consommation

```
{
    "capteurId": 1,
    "valeurConsommation": 15.42
}

```

Listing 4 – Format des données envoyées

- **POST /api/alertes** : Envoi des alertes détectées

```
{
    "capteurId": 1,
    "type": "FUITE_DETECTEE",
    "message": "Suspicion de fuite: Consommation continue...",
    "niveauUrgence": "ELEVEE"
}

```

Listing 5 – Format des alertes

.1.6 Utilisation du simulateur

Lancement avec configuration par défaut

```
python iot_simulator.py
```

Lancement avec paramètres personnalisés

```
# URL personnalisee
python iot_simulator.py --url http://192.168.1.10:8080/SmartWaterMonitoring

# Intervalle de 5 secondes (tests)
python iot_simulator.py --interval 5

# URL et intervalle personnalises
python iot_simulator.py --url http://exemple.com/app --interval 30
```

.1.7 Statistiques et monitoring

Le simulateur affiche des statistiques en temps réel :

- Nombre total d'envois effectués
- Taux de succès et d'erreurs
- Nombre d'alertes générées
- Indication de la période horaire (pointe/creuse/normale)

Exemple de sortie :

```
============================================================
        Simulateur IoT Smart Water Monitoring
============================================================
Backend: http://localhost:8080/SmartWaterMonitoring
Intervalle: 60 secondes
============================================================

Capteurs decouverts: [1, 2, 3, 4, 5]

[14:23:45] Capteur #3: 18.42L envoye
[14:24:45] Capteur #1: 22.15L envoye
[14:24:45] ALERTE generee: SEUIL_DEPASSE - Capteur #1

Statistiques: 50 envois | 49 reussis | 1 erreurs | 2 alertes
14:25:30 - Heures de POINTE
```

.2 Système d'Agrégation des Données

.2.1 Architecture de l'agrégation

Le système d'agrégation transforme les données brutes des capteurs (stockées dans `DonneeCapteur`) en historiques journaliers consolidés (table `HistoriqueConsommation`). Cette architecture à deux niveaux permet :

- De conserver les données brutes pour analyse détaillée
- De disposer d'agrégats optimisés pour l'affichage et les statistiques
- De calculer automatiquement les coûts estimés

.2.2 Service d'agrégation - DataAggregationService

Responsabilités

La classe `DataAggregationService` gère l'ensemble du processus d'agrégation :

1. **Agrégation journalière** : Pour une date donnée, consolide toutes les mesures
2. **Calcul des métriques** : Volume total, consommation moyenne, coût estimé
3. **Rattrapage automatique** : Traite les journées manquantes
4. **Génération de statistiques** : Produit des analyses hebdomadaires et mensuelles

Algorithme d'agrégation par utilisateur

Pour chaque utilisateur et chaque journée :

```
private boolean aggregerDonneesUtilisateur(
    Utilisateur utilisateur, LocalDate date) {

    // 1. Verifier si l'historique existe deja
    HistoriqueConsommation existant =
        chercherHistorique(utilisateur, date);
    if (existant != null) return false;

    // 2. Definir les limites temporelles
    LocalDateTime debut = date.atStartOfDay();      // 00:00:00
    LocalDateTime fin = date.plusDays(1).atStartOfDay(); // 00:00:00 J+1

    // 3. Recuperer toutes les donnees capteur de la journee
    List<DonneeCapteur> donnees =
        recupererDonnees(utilisateur, debut, fin);

    if (donnees.isEmpty()) return false;

    // 4. Calculer les agregats
    double volumeTotal = donnees.stream()
        .mapToDouble(DonneeCapteur::getValeurConsommation)
        .sum();

    double consommationMoyenne = volumeTotal / 24.0;
    double coutEstime = volumeTotal * PRIX_EAU_PAR_LITRE;

    // 5. Creer et persister l'historique
    HistoriqueConsommation historique = new HistoriqueConsommation();
    historique.setDate(date);
    historique.setVolumeTotal(volumeTotal);
    historique.setConsommationMoyenne(consommationMoyenne);
    historique.setCoutEstime(coutEstime);
    historique.setUtilisateur(utilisateur);

    session.persist(historique);

    return true;
}
```

Listing 6 – Processus d'agrégation

Calcul du coût

Le coût estimé est calculé selon le tarif de l'eau :

$$\text{Coût} = \text{Volume (L)} \times \text{Prix (€/L)} \tag{1}$$

Avec un prix par défaut de **0.00722 €/L** (environ 7.22 €/m^3), correspondant au tarif moyen de l'eau en France.

.2.3 Système de rattrapage automatique

Détection des données manquantes

La méthode `aggregerDonneesManquantes()` analyse la base pour identifier les journées non agrégées :

```java
public int aggregerDonneesManquantes() {
    LocalDate hier = LocalDate.now().minusDays(1);

    // Trouver la plus ancienne mesure non agregee
    LocalDateTime minHorodatage = session.createQuery(
        "SELECT MIN(dc.horodatage) FROM DonneeCapteur dc " +
        "WHERE dc.horodatage < :limite", LocalDateTime.class)
        .setParameter("limite", hier.plusDays(1).atStartOfDay())
        .uniqueResult();

    if (minHorodatage == null) return 0;

    LocalDate dateDebut = minHorodatage.toLocalDate();

    // Agreger toute la periode manquante
    return aggregerPeriode(dateDebut, hier);
}
```

Listing 7 – Détection des journées manquantes

Ce mécanisme garantit qu'aucune donnée ne reste non agrégée, même en cas de panne temporaire du système.

.2.4 Planificateur automatique - DailyAggregationJob

Architecture du job

Le `DailyAggregationJob` est un singleton qui utilise un `ScheduledExecutorService` pour exécuter l'agrégation automatiquement chaque jour.

Configuration

- **Heure d'exécution** : 00 :30 (30 minutes après minuit)
- **Fréquence** : Une fois par jour
- **Mode** : Exécution en arrière-plan (thread dédié)

Calcul du délai initial

Au démarrage, le job calcule le délai jusqu'à la prochaine exécution :

```java
private long calculerDelaiJusquaProchaineExecution() {
    LocalDateTime maintenant = LocalDateTime.now();
    LocalDateTime prochaineExecution =
        maintenant.toLocalDate().atTime(heureExecution);

    // Si l'heure est passee, planifier pour demain
    if (maintenant.isAfter(prochaineExecution)) {
        prochaineExecution = prochaineExecution.plusDays(1);
    }

    return Duration.between(maintenant, prochaineExecution)
                   .getSeconds();
}
```

Listing 8 – Calcul du délai

Séquence d'exécution

À chaque exécution, le job effectue :

1. **Agrégation des données capteurs**

- Traite la veille (J-1)
- Effectue le rattrapage des jours manquants

2. **Génération des statistiques**
   - Calcule les moyennes hebdomadaires
   - Calcule les moyennes mensuelles
   - Identifie les tendances de consommation
3. **Affichage des métriques**
   - Nombre d'historiques créés
   - Nombre de statistiques générées
   - État global du système

Logs d'exécution

Exemple de sortie lors de l'exécution automatique :

```
============================================================
DEBUT DE L'AGREGATION AUTOMATIQUE - 2025-11-14T00:30:00
============================================================

Demarrage du rattrapage automatique jusqu'a : 2025-11-13
Rattrapage : agregation de 2025-11-10 a 2025-11-13

--- Traitement du 2025-11-10 ---
Agregation pour Jean Dupont - 2025-11-10
15 mesures trouvees
Historique cree : 245.5L, 1.77€

Agregation terminee : 4 historiques crees

Generation des statistiques journalieres...
5 statistiques generees pour la veille

Statistiques globales :
   - Total historiques : 156
   - Donnees non agregees : 0
   - Derniere date agregee : 2025-11-13

============================================================
FIN DE L'AGREGATION AUTOMATIQUE - 2025-11-14T00:30:15
Prochaine execution : demain a 00:30
============================================================
```

.2.5 Initialisation au démarrage

Le job est initialisé automatiquement au démarrage de l'application via l'`ApplicationStartupListener` :

```
1 @WebListener
2 public class ApplicationStartupListener
3     implements ServletContextListener {
4
5     @Override
6     public void contextInitialized(ServletContextEvent sce) {
```

```
7           System.out.println("Demarrage de l'application...");
8
9           // Demarrer le job d'agregation
10          DailyAggregationJob job = DailyAggregationJob.getInstance();
11          job.start();
12
13          System.out.println("Application prete");
14      }
15  }
```

Listing 9 – Initialisation automatique

.2.6 Tests et validation

Test manuel

Pour tester l'agrégation sans attendre l'exécution automatique :

```
1 // Dans un servlet ou une classe de test
2 DailyAggregationJob job = DailyAggregationJob.getInstance();
3 job.executerMaintenant();
```

Listing 10 – Exécution manuelle pour tests

Métriques de performance

Sur un ensemble de test avec 5 utilisateurs et 1000 mesures par jour :

- Temps d'agrégation d'une journée : ∼2 secondes
- Temps de rattrapage de 7 jours : ∼15 secondes
- Génération de statistiques : ∼1 seconde

.3 Intégration globale

.3.1 Flux de données complet

1. **Génération** : Le simulateur Python envoie des mesures toutes les minutes
2. **Réception** : Le servlet `DataApiServlet` reçoit et valide les données
3. **Stockage** : Les mesures sont persistées dans `DonneeCapteur`
4. **Agrégation** : Chaque nuit à 00 :30, le job consolide les données
5. **Historiques** : Les agrégats sont stockés dans `HistoriqueConsommation`
6. **Statistiques** : Des analyses sont générées pour l'affichage
7. **Visualisation** : Les utilisateurs consultent leurs données via les dashboards

.3.2 Avantages de cette architecture

- **Scalabilité** : Les agrégats évitent de requêter des milliers de mesures
- **Performance** : Les requêtes sur historiques sont très rapides
- **Fiabilité** : Le rattrapage automatique garantit la cohérence
- **Flexibilité** : Les données brutes restent accessibles pour analyses détaillées
- **Testabilité** : Le simulateur permet de générer des données réalistes

.3.3 Évolutions possibles

- Agrégation en temps réel (en plus de la journalière)
- Détection d'anomalies par machine learning
- Alertes prédictives basées sur les tendances
- Agrégation multi-niveaux (horaire, journalière, mensuelle)
- Export des données pour analyses externes

Abderrahman Bouanani et Abou Kekeli Efrayim

Résumé du Projet

Le présent projet vise à développer une solution innovante destinée à [décrire brièvement l'objectif principal du projet]. Pour atteindre cet objectif, une approche méthodologique rigoureuse a été adoptée, combinant une analyse fonctionnelle détaillée, une conception modulaire et l'implémentation de technologies modernes.

Les contributions majeures du projet incluent la mise en œuvre d'une interface utilisateur intuitive, l'intégration d'un système performant de gestion des données et la validation de la solution via des tests exhaustifs. Ces résultats traduisent une amélioration significative en termes de [performance, réactivité, sécurité, etc.] par rapport aux solutions existantes.

Ce projet a permis d'acquérir des compétences avancées en [développement web, modélisation UML, architecture logicielle, etc.], tout en consolidant la compréhension de la gestion de projet dans un environnement technique exigeant. En outre, il ouvre des perspectives prometteuses pour de futurs développements, tels que l'ajout de nouvelles fonctionnalités, l'optimisation des performances et l'amélioration de l'ergonomie de l'interface utilisateur.

L'ensemble de ces réalisations témoigne de l'engagement à proposer une solution à la fois innovante et efficace, répondant aux problématiques identifiées dès le début du projet.

Mots clés : solution innovante, gestion des données, interface utilisateur, tests exhaustifs, optimisation des performances